Wie kommt der Hint in das SQL?

… ohne die Anwendung zu ändern …

Mathias Zarick
Principal Consultant

BASEL BERN BRUGG GENF LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

Vorstellung – Mathias Zarick

- Principal Consultant bei Trivadis Delphi GmbH in Wien
- Trainer
  - Data Guard, Architektur und Interna für fortgeschrittene DBAs, Maximum Availability Architecture Workshop
- E-Mail: Mathias.Zarick@trivadis.com
- Hauptthemen:
  - Oracle Datenbank
  - Oracle Hochverfügbarkeitsprojekte (Real Application Clusters, Data Guard, Maximum Availability Architecture, Replikation mit Streams und GoldenGate)
  - Backup/Recovery
  - Entwicklungsleiter der Trivadis Toolbox
  - Entwickler von TVD-Standby
  - Forschungsprojekte im Trivadis Technology Center (TTC)

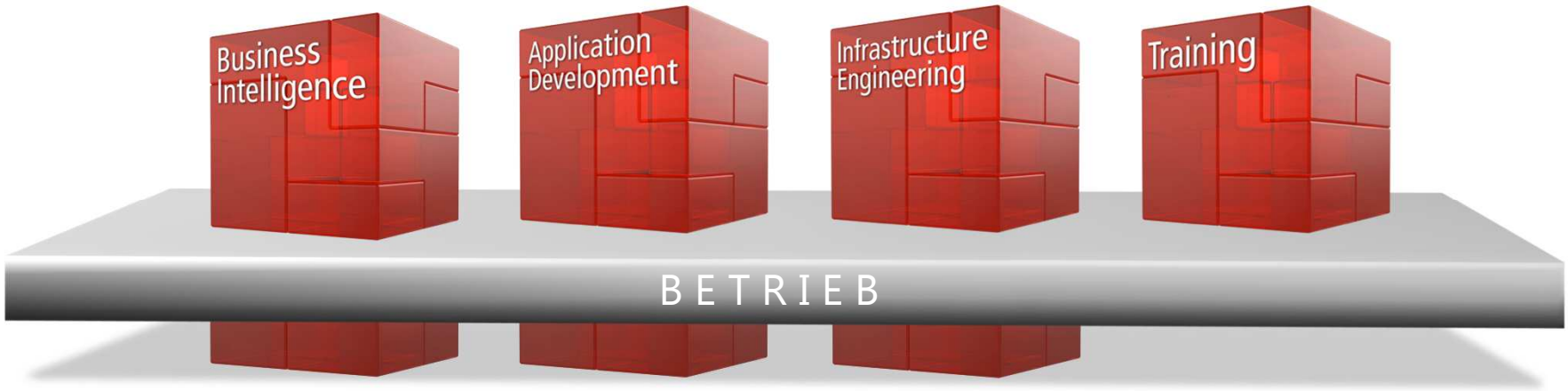
ORACLE Certified Master
Oracle Database 11*g* Administrator

ORACLE Certified Professional

Unser Unternehmen

Trivadis ist führend bei der IT-Beratung, der Systemintegration, dem Solution-Engineering und der Erbringung von IT-Services mit Fokussierung auf ORACLE und Microsoft Technologien im D-A-CH-Raum.

Unsere Leistungen erbringen wir aus den strategischen Geschäftsfeldern:



Trivadis Services übernimmt den korrespondierenden Betrieb Ihrer IT Systeme.

Mit über 600 IT- und Fachexperten bei Ihnen vor Ort

12 Trivadis Niederlassungen mit über 600 Mitarbeitenden

200 Service Level Agreements

Mehr als 4'000 Trainingsteilnehmer

Forschungs- und Entwicklungs-budget: CHF 5.0 Mio. / EUR 4.0 Mio.

Finanziell unabhängig und nachhaltig profitabel

Erfahrung aus mehr als 1'900 Projekten pro Jahr bei über 800 Kunden

Stand 12/2013

Trivadis an der DOAG

Ebene 3 - gleich neben der Rolltreppe

Wir freuen uns auf Ihren Besuch.

Denn mit Trivadis gewinnen Sie immer.

Wie kommt der Hint in das SQL?

1. Einführung
2. Stored Outlines
3. SQL Profiles
4. SQL Patches
5. SQL Plan Baselines
6. Allgemeines und Fazit

Einführung

Was sind Hints?

- Hints ermöglichen, den Oracle Optimizer und/oder seine Entscheidungen zu beeinflussen
- wenn möglich, wird ihnen gefolgt
- Hints werden in Kommentaren nach dem ersten Schlüsselwort eines Query Blocks benutzt
- Das erste Zeichen im Kommentar muss ein + sein
- 2 Kategorien:
  - Nicht-Optimizer Hints, z.B.
    `APPEND, CACHE, MONITOR, GATHER_PLAN_STATISTICS, RESULT_CACHE, …`
  - Optimizer Hints, z.B.
    `FULL, INDEX, LEADING, ORDERED, …`
- Welche Hints gibt es?
  Siehe View **`v$sql_hint`**

Warum (keine) Hints?

- Workaround für suboptimale Ausführungspläne
- Sollten keine permanente Lösung sein
  - Die Daten können sich ändern, der Plan kann unangebracht werden
  - Der Optimizer lernt in einer neueren Version dazu
- Nützlich in Testszenarien / Was-wäre-wenn-Analysen
- Einige Hints müssen verwendet werden, um ein spezielles Verhalten der Oracle Datenbank zu erreichen, z.B.
  - **`APPEND, BIND_AWARE, RESULT_CACHE, …`**

Jonathan Lewis' Rules for Hinting

http://jonathanlewis.wordpress.com/2008/05/02/rules-for-hinting/

1. Don't
2. If you must use hints, then assume you've used them incorrectly.
3. On every patch or upgrade to Oracle, assume every piece of hinted SQL is going to do the wrong thing … because of (2) above. You've been lucky so far, but the patch/upgrade lets you discover your mistake.
4. Every time you apply some DDL to an object that appears in a piece of hinted SQL assume that the hinted SQL is going to do the wrong thing … because of (2) above. You've been lucky so far, but the structural change lets you discover your mistake.

You will appreciate from these guidelines that I don't really approve of using hints. The only reason that I leave them in place on a production system is when I'm ***sure*** that there is no alternative ***safe*** mechanism for making the optimizer do what I want. (And that does mean that I will use hints sometimes on a production system.)

Beispiel für diese Präsentation (1)

- Ursprüngliches Statement

```
SELECT count(*) FROM t WHERE id=0815
```

- Ursprünglicher Plan

```
SELECT * FROM
table(dbms_xplan.display_cursor(format=>'BASIC'))
-------------------------------------
| Id  | Operation           | Name  |
-------------------------------------
|   0 | SELECT STATEMENT    |       |
|   1 |  SORT AGGREGATE     |       |
|   2 |   INDEX UNIQUE SCAN| T_PK  |
-------------------------------------
```

Beispiel für diese Präsentation (2)

- Alternatives Statement mit Hint

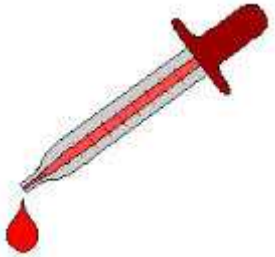
```
SELECT /*+ FULL(t) */ count(*) FROM t WHERE id=0815
```

- Alternativer Plan

```
SELECT * FROM
table(dbms_xplan.display_cursor(format=>'BASIC'))
-----------------------------------
| Id  | Operation           | Name |
-----------------------------------
|   0 | SELECT STATEMENT    |      |
|   1 |  SORT AGGREGATE     |      |
|   2 |   TABLE ACCESS FULL| T    |
-----------------------------------
```

Die Herausforderung

- Kein Zugriff auf die Applikation um das SQL zu ändern
- Der Hint muss auf irgendeine andere Weise injiziert werden



SQL_ID btuu7yga9bcp1

Wie kommt der Hint in das SQL?

1. Einführung
2. Stored Outlines
3. SQL Profiles
4. SQL Patches
5. SQL Plan Baselines
6. Allgemeines und Fazit

Stored Outlines

Was sind Stored Outlines?

- Stored Outlines stellen Planstabilität durch das Speichern der Outline-Daten eines assoziierten Planes eines SQL Statements zur Verfügung
- Es werden alle Hints gespeichert, um den Ausführungsplan zu fixieren
- Die Assoziierung zwischen SQL und Stored Outline geschieht durch eine SQL Signatur, welche nach einer Normalisierung produziert wird
  - Großbuchstaben – außer Literale
  - Whitespace wird entfernt – Kommentare nicht
  - Literal-insensitiver Match nur durch **`cursor_sharing=force`**
- Hints werden gespeichert in
  - **`OUTLN.OL$`**
  - **`OUTLN.OL$HINTS`**
  - **`OUTLN.OL$NODES`**
- Views
  - **`CDB_/DBA_/ALL_/USER_OUTLINES`**
  - **`CDB_/DBA_/ALL_/USER_OUTLINE_HINTS`**

Unser Beispiel mit einer Stored Outline (1)

- Einmal pro Instanz oder für die gewünschten Sessions
  Achtung: **`use_stored_outlines`** kann nicht persistent im init.ora eingetragen werden, man braucht einen Startup Trigger

```
ALTER SYSTEM SET use_stored_outlines=TRUE;
```

- Wir erzeugen 2 Outlines für den ursprünglichen und den gewünschten Plan

```
CREATE OUTLINE my_outline ON
SELECT count(*) FROM t WHERE id=0815;

CREATE OUTLINE my_outline_interim ON
SELECT /*+ FULL(t) */ count(*) FROM t WHERE id=0815;
```

Unser Beispiel mit einer Stored Outline (2)

- Outline Hints austauschen – das erste Set kommt zur zweiten Outline, das zweite Set zur ersten Outline

```
UPDATE outln.ol$hints
SET ol_name=decode(ol_name,'MY_OUTLINE','MY_OUTLINE_INTERIM',
                           'MY_OUTLINE_INTERIM','MY_OUTLINE')
WHERE ol_name IN ('MY_OUTLINE','MY_OUTLINE_INTERIM');

UPDATE outln.ol$
SET hintcount = (
  SELECT  count(*)
  FROM    outln.ol$hints
  WHERE   outln.ol$.ol_name = outln.ol$hints.ol_name
)
WHERE outln.ol$.ol_name in
('MY_OUTLINE','MY_OUTLINE_INTERIM');

COMMIT;
```

Ups! DML auf OUTLN Tables?

- Was ist das denn für ein Hack? Ist das erlaubt?
- Anerkannt durch „My Oracle Support - How to Specify Hidden Hints (Outlines) on SQL Statements in Oracle 8i (Doc ID 92202.1)”
- Komplettiert von Jonathan Lewis: Plan Stability in Oracle 8i/9i - http://www.jlcomp.demon.co.uk/04_outlines.rtf

Verifikation

- Überprüfen, ob der gewünschte Plan verwendet wird

```
SELECT count(*) FROM t WHERE id=0815;

SELECT * FROM
table(dbms_xplan.display_cursor(format=>'basic +note'));
...
-------------------------------------
| Id  | Operation           | Name |
-------------------------------------
|   0 | SELECT STATEMENT    |      |
|   1 |  SORT AGGREGATE     |      |
|   2 |   TABLE ACCESS FULL| T    |
-------------------------------------

Note
-----
   - outline "MY_OUTLINE" used for this statement
```

Zwischenbilanz: Stored Outlines

	Stored Outlines
Verfügbar seit	8i
Deprecated ?	Seit 11g
XE/SE/SE1 ?	Ja
Tuning Pack benötigt	Nein
Match wird nach Normalisierung durchgeführt	Ja
Literal-insensitiver Match (force_match=true)	Nein
Datenänderungen werden berücksichtigt	Nein
Nutzbar um einen Hint im SQL unwirksam zu machen	Ja
Kategorien für verschiedene Workloads und Perioden	Ja
`gather_plan_statistics` Hint injizieren	Nein
Parallelverarbeitung ändern	Ja

Wie kommt der Hint in das SQL?

1. Einführung
2. Stored Outlines
3. SQL Profiles
4. SQL Patches
5. SQL Plan Baselines
6. Allgemeines und Fazit

SQL Profiles

Was sind SQL Profiles?

- SQL Profiles helfen dem Optimizer, einen besseren Ausführungsplan zu finden
- Werden durch SQL Tuning Advisor (oder manuell angelegt)
- Brauchen Tuning Pack (daher auch Enterprise Edition und Diagnostics Pack)
  - Brauchen **`control_management_pack_access`** auf **`DIAGNOSTIC+TUNING`** gesetzt
- SQL Profiles speichern nicht alle (wie stored outlines) sondern nur einige Hints um den Ausführungsplan zu beeinflussen
- Assoziierung auf das SQL geschieht durch eine Signatur, welche nach Normalisierung erzeugt wird (es ist außerdem ein literal-insensitiver Match konfigurierbar)
- Hints sind gespeichert in
  - **`sys.sqlobj$`**
  - **`sys.sqlobj$auxdata`**
  - **`sys.sql$text`**
  - **`sys.sql$`** **`(12c)`**
- Views
  - **`CDB_/DBA_/~~ALL_/USER~~_SQL_PROFILES`**

Unser Beispiel mit einem SQL Profile

- Manuelle Erstellung des SQL Profile mit undokumentierter Prozedur

```
BEGIN
  dbms_sqltune.import_sql_profile(
    name        => 'MY_SQL_PROFILE',
    description => 'full table scan',
    sql_text    => 'SELECT count(*) FROM t WHERE id=0815',
    force_match => true,
    profile     => sqlprof_attr('FULL(@"SEL$1" "T"@"SEL$1")')
  );
END;
```

- `force_match`: Literal-insensitiver Match

Undokumentierte Prozeduren benutzen?

- Was ist das denn für ein Hack? Ist das erlaubt?
- Man lese “My Oracle Support - SQLT Diagnostic Tool (Doc ID 215187.1)”
- Zip file sqlt.zip, script sqlt\utl\coe_xfr_sql_profile.sql

```
REM DESCRIPTION
REM   This script generates another that contains the commands to
REM   create a manual custom SQL Profile out of a known plan from
REM   memory or AWR. The manual custom profile can be implemented
REM   into the same SOURCE system where the plan was retrieved,
REM   or into another similar TARGET system that has same schema
REM   objects referenced by the SQL that generated the known plan.
```

Verifikation

- Überprüfen, ob der gewünschte Plan verwendet wird

```
SELECT count(*) FROM t WHERE id=0815;

SELECT * FROM
table(dbms_xplan.display_cursor(format=>'basic +note'));
...
-------------------------------------
| Id  | Operation           | Name |
-------------------------------------
|   0 | SELECT STATEMENT    |      |
|   1 |  SORT AGGREGATE     |      |
|   2 |   TABLE ACCESS FULL| T    |
-------------------------------------

Note
-----
   - SQL profile MY_SQL_PROFILE used for this statement
```

Zwischenbilanz: SQL Profiles

	SQL Profiles
Verfügbar seit	10g
Deprecated ?	Nein
XE/SE/SE1 ?	Nein
Tuning Pack benötigt	Ja
Match wird nach Normalisierung durchgeführt	Ja
Literal-insensitiver Match (force_match=true)	Ja
Datenänderungen werden berücksichtigt	Einigermaßen
Nutzbar um einen Hint im SQL unwirksam zu machen	Ja
Kategorien für verschiedene Workloads und Perioden	Ja
`gather_plan_statistics` Hint injizieren	Ja
Parallelverarbeitung ändern	Ja

Wie kommt der Hint in das SQL?

1. Einführung
2. Stored Outlines
3. SQL Profiles
4. SQL Patches
5. SQL Plan Baselines
6. Allgemeines und Fazit

SQL Patches

Was sind SQL Patches?

- SQL Patches helfen dem Optimizer, einen alternativen Ausführungsplan zu finden
- Wurden designt, um Ausführungsfehler zu vermeiden, welche bei einem bestimmten Ausführungsschritt passieren und werden durch den SQL Repair Advisor (oder manuell) erstellt
- Auch in Standard Edition verfügbar, siehe
  http://www.oracle.com/webfolder/technetwork/de/community/dbadmin/tipps/advisor/index.html
- SQL Patches speichern nicht alle (wie stored outlines) sondern nur einige Hints, um den Ausführungsplan zu beeinflussen
- Assoziierung geschieht wieder durch dieselbe Signatur nach Normalisierung (Literal-insensitiver Match ist auch konfigurierbar)
- Hints sind gespeichert in
  - `sys.sqlobj$`
  - `sys.sqlobj$auxdata`
  - `sys.sql$text`
  - `sys.sql$` (12c)
- Views
  - `CDB_/DBA_/~~ALL_/USER~~_SQL_PATCHES`

Unser Beispiel mit einem SQL Patch (1)

- Manuelle Erstellung eines SQL Patch mit undokumentiertem Package und Prozedur

```
BEGIN
  sys.dbms_sqldiag_internal.i_create_patch(
      sql_text  => 'SELECT count(*) FROM t WHERE id=0815',
      hint_text => 'FULL(@"SEL$1" "T"@"SEL$1")',
      name      => 'MY_SQL_PATCH');
END;
```

Unser Beispiel mit einem SQL Patch (2)

- Text wird etwas länger, wenn wir literal-insensitiven Match (forced text match) brauchen

```
DECLARE
  v_name VARCHAR2(128);
BEGIN
  v_name :=
    sys.dbms_sqltune_internal.i_create_sql_profile(
      sql_text    => 'SELECT count(*) FROM t WHERE id=0815',
      profile_xml => '<outline_data><hint>
                       <![CDATA[FULL(@"SEL$1" "T"@"SEL$1")]]>
                      </hint></outline_data>',
      is_patch    => true,
      name        => 'MY_SQL_PATCH',
      force_match => true);
END;
```

Undokumentierte Prozeduren benutzen?

- Was ist das denn für ein Hack? Ist das erlaubt?
- Lösung wird durch das Oracle Optimizer Team auf ihrem Blog vorgeschlagen: https://blogs.oracle.com/optimizer/entry/how_can_i_hint_a

Verifikation

- Überprüfen, ob der gewünschte Plan verwendet wird

```
SELECT count(*) FROM t WHERE id=0815;

SELECT * FROM
table(dbms_xplan.display_cursor(format=>'basic +note'));
...
-------------------------------------
| Id  | Operation           | Name |
-------------------------------------
|   0 | SELECT STATEMENT    |      |
|   1 |  SORT AGGREGATE     |      |
|   2 |   TABLE ACCESS FULL| T    |
-------------------------------------

Note
-----
   - SQL patch "MY_SQL_PATCH" used for this statement
```

Zwischenbilanz: SQL Patches

	SQL Patches
Verfügbar seit	11g
Deprecated ?	Nein
XE/SE/SE1 ?	Ja
Tuning Pack benötigt	Nein
Match wird nach Normalisierung durchgeführt	Ja
Literal-insensitiver Match (force_match=true)	Ja
Datenänderungen werden berücksichtigt	Einigermaßen
Nutzbar um einen Hint im SQL unwirksam zu machen	Ja
Kategorien für verschiedene Workloads und Perioden	Ja
`gather_plan_statistics` Hint injizieren	Ja
Parallelverarbeitung ändern	Ja

Wie kommt der Hint in das SQL?

1. Einführung
2. Stored Outlines
3. SQL Profiles
4. SQL Patches
5. SQL Plan Baselines
6. Allgemeines und Fazit

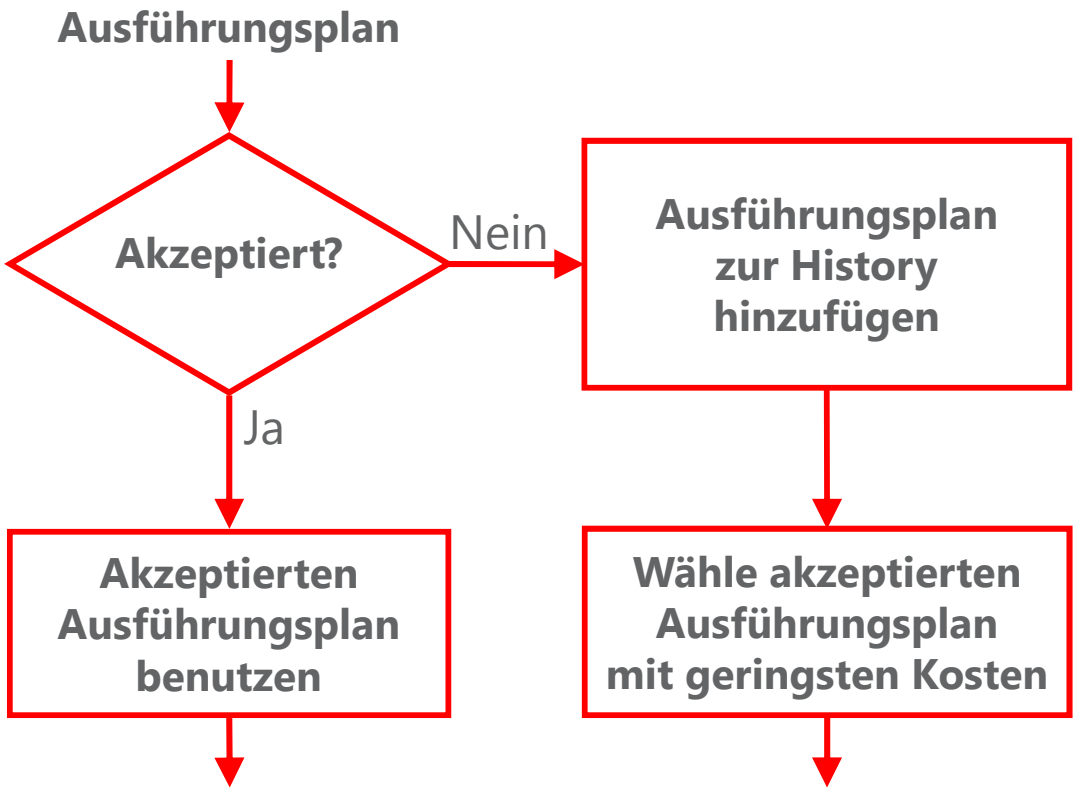
SQL Plan Baselines

Was sind SQL Plan Baselines?

- SQL Plan Baselines sind Teil des SQL Plan Management Feature
- Es sind Objekte, welche vom Optimizer berücksichtigt werden
- Sie enthalten Pläne für bestimmte Statements, welche als Hints gespeichert sind
- Pläne können akzeptiert (verifiziert), fixed, enabled/disabled sein
- Keine weitere Lizenz https://blogs.oracle.com/optimizer/entry/does_the_use_of_sql aber sie funktionieren nicht in Standard Edition
- Assoziierung geschieht wieder durch dieselbe Signatur nach Normalisierung, ein literal-insensitiver Match ist leider nicht konfigurierbar
- Hints sind gespeichert in
  - `sys.sqlobj$`
  - `sys.sqlobj$auxdata`
  - `sys.sql$text`
  - `sys.sql$` (12c)
- Views
  - `CDB_/DBA_/~~ALL_/USER~~_SQL_PLAN_BASELINES`

SQL Plan Baselines – Auswahl

- Ablauf (vereinfacht)



Unser Beispiel mit einer SQL Plan Baseline (1)

- Wir müssen die SQL ID kennen, dann können wir den ungewünschten Plan aus dem Cursor Cache in eine Baseline laden

```
DECLARE
  v_cnt NUMBER;
BEGIN
  v_cnt := dbms_spm.load_plans_from_cursor_cache(
             sql_id=>'btuu7yga9bcp1');
END;
```

- Wir brauchen sql_handle und plan_name aus dba_sql_plan_baselines

```
SELECT sql_handle, sql_text, plan_name, description, enabled
FROM dba_sql_plan_baselines
WHERE sql_text LIKE 'SELECT count(*) FROM t WHERE id=0815'
```

Unser Beispiel mit einer SQL Plan Baseline (2)

- Plan deaktivieren und optional umbenennen und kommentieren

```
DECLARE
  v_cnt NUMBER;
BEGIN
  v_cnt := dbms_spm.alter_sql_plan_baseline(
    sql_handle      => 'SQL_d7cd0a6e6c80df97',
    plan_name       => 'SQL_PLAN_dgm8adtq81rwr916941b3',
    attribute_name  => 'enabled',
    attribute_value => 'NO');
  v_cnt := dbms_spm.alter_sql_plan_baseline(
    sql_handle      => 'SQL_d7cd0a6e6c80df97',
    plan_name       => 'SQL_PLAN_dgm8adtq81rwr916941b3',
    attribute_name  => 'plan_name',
    attribute_value => 'NOT_DESIRED_PLAN');
  v_cnt := dbms_spm.alter_sql_plan_baseline(
    sql_handle      => 'SQL_d7cd0a6e6c80df97',
    plan_name       => 'NOT_DESIRED_PLAN',
    attribute_name  => 'description',
    attribute_value => 'original, but not desired plan');
END;
```

sql_handle und plan_name stammen aus dba_sql_plan_baselines

Unser Beispiel mit einer SQL Plan Baseline (3)

- Das Statement mit Hint ausführen, um es in den Cursor Cache zu laden

```
SELECT /*+ FULL(t) */ count(*) FROM t WHERE id=0815
```

- sql_id und plan_hash_value dieses Statement mit Hint besorgen

```
SELECT sql_id, plan_hash_value
FROM v$sql WHERE sql_text LIKE
        'SELECT /*+ FULL(t) */ count(*) FROM t WHERE id=0815'
```

- Den gewünschten Plan nun mit der bereits bestehenden Baseline assoziieren

```
DECLARE v_cnt NUMBER;  BEGIN
  v_cnt := dbms_spm.load_plans_from_cursor_cache(
    sql_id          => 'buzbqk5t2m81y',
    plan_hash_value => '2966233522',
    sql_handle      => 'SQL_d7cd0a6e6c80df97');
END;
```

sql_id und plan_hash_value stammen aus v$sql
sql_handle ist dieselbe wie zuvor

Unser Beispiel mit einer SQL Plan Baseline (4)

- Optional den zweiten Plan umbenennen und kommentieren

```
DECLARE
  v_cnt NUMBER;
BEGIN
  v_cnt := dbms_spm.alter_sql_plan_baseline(
    sql_handle => 'SQL_d7cd0a6e6c80df97',
    plan_name => 'SQL_PLAN_dgm8adtq81rwr3fdbb376',
    attribute_name => 'plan_name',
    attribute_value => 'DESIRED_PLAN');
  v_cnt := dbms_spm.alter_sql_plan_baseline(
    sql_handle => 'SQL_d7cd0a6e6c80df97',
    plan_name => 'DESIRED_PLAN',
    attribute_name => 'description',
    attribute_value => 'desired plan introduced by the hint');
END;
```

sql_handle ist dieselbe wie zuvor

plan_name stammt aus dba_sql_plan_baselines

Einen Plan zu einem anderen Statement assoziieren?

- Was ist das denn für ein Hack? Ist das erlaubt?
- Die gezeigten Schritte werden durch Maria Colgan (ehemals Product Manager Oracle Optimizer) in verschiedenen Posts auf https://blogs.oracle.com/optimizer empfohlen
- Ebenso in ihrer Präsentation „Harnessing the Power of Optimizer Hints”
- Aus dieser Präsentation stammt das Mantra „If you can hint it, baseline it“
- Außerdem – alle benutzten Statements sind regulär dokumentiert

Verifikation

- Überprüfen, ob der gewünschte Plan verwendet wird

```
SELECT count(*) FROM t WHERE id=0815;

SELECT * FROM
table(dbms_xplan.display_cursor(format=>'basic +note'));
...
-------------------------------------
| Id  | Operation           | Name |
-------------------------------------
|   0 | SELECT STATEMENT    |      |
|   1 |  SORT AGGREGATE     |      |
|   2 |   TABLE ACCESS FULL| T    |
-------------------------------------

Note
-----
   - SQL plan baseline DESIRED_PLAN used for this statement
```

Zwischenbilanz: SQL Plan Baselines

	SQL Plan Baselines
Verfügbar seit	11g
Deprecated ?	Nein
XE/SE/SE1 ?	Nein
Tuning Pack benötigt	Nein
Match wird nach Normalisierung durchgeführt	Ja
Literal-insensitiver Match (force_match=true)	Nein
Datenänderungen werden berücksichtigt	Ja
Nutzbar um einen Hint im SQL unwirksam zu machen	Ja
Kategorien für verschiedene Workloads und Perioden	Nein
`gather_plan_statistics` Hint injizieren	Nein
Parallelverarbeitung ändern	Ja

Wie kommt der Hint in das SQL?

1. Einführung
2. Stored Outlines
3. SQL Profiles
4. SQL Patches
5. SQL Plan Baselines
6. Allgemeines und Fazit

Allgemeines und Fazit

Ungewünschte Hints unwirksam machen

- Alle gezeigten Techniken können verwendet werden, um ungewünschte Hints in einem SQL statement unwirksam zu machen, z.B.

```
SELECT /*+ FULL(t) */ count(*) FROM t WHERE id=0815
```

- Einfachster Weg

```
ALTER SESSION SET "_optimizer_ignore_hints" = true
```

kann auch auf System-Ebene gesetzt werden

- Ansonsten einen konträren Hint oder den **`IGNORE_OPTIM_EMBEDDED_HINTS`** Hint verwenden

unhint_patch.sql
or unhint_profile.sql
maybe: unhint_plan_baseline.sql

Unhint mit Stored Outline

- Für Stored Outlines gibt es einen Spezialfall, wir können die Hints auf einen einzigen reduzieren:

```
CREATE OUTLINE my_outline ON
SELECT /*+ FULL(t) */ count(*) FROM t WHERE id=0815;

DELETE FROM outln.ol$hints
WHERE ol_name = 'MY_OUTLINE'
AND hint_text != 'IGNORE_OPTIM_EMBEDDED_HINTS'
;
UPDATE outln.ol$
SET hintcount = 1
WHERE outln.ol$.ol_name = 'MY_OUTLINE'
;
COMMIT;
```

Outlines können zu SQL Plan Baselines migriert werden

- Die Migration erfolgt mit dem **`dbms_spm`** Package

```
variable v clob
exec :v :=
dbms_spm.migrate_stored_outline('outline_name','MY_OUTLINE');
DROP OUTLINE my_outline;
```

Planstabilitäts-Objekte transferieren

- Stored Outlines: Inhalte der **`outln`** Tabellen transferieren, siehe Troubleshooting Oracle Performance 2 / Christian Antognini
- Für SQL profiles, SQL patches, SQL plan baselines gibt es unpack/pack Prozeduren
  - Staging Tabelle erzeugen
    - **`dbms_sqltune.create_stgtab_sqlprof`**
    - **`dbms_sqldiag.create_stgtab_sqlpatch`**
    - **`dbms_spm.create_stgtab_baseline`**
  - Die Staging Tabellen haben dieselbe Struktur → man kann sogar ein und dieselbe Tabelle als ein Container für alle drei Typen verwenden
  - Objekt packen, Tabelle transferieren, Objekt auspacken (unpack)
    - **`dbms_sqltune.(un)pack_stgtab_sqlprof`**
    - **`dbms_sqldiag.(un)pack_stgtab_sqlpatch`**
    - **`dbms_spm.(un)pack_stgtab_baseline`**

result_cache_patch.sql
append_patch.sql or append_profile.sql
maybe: gather_plan_statistics_patch.sql

Besondere non-Optimizer Hints

- Tuning Sessions können von Plan Statistiken profitieren
  - Ein **`GATHER_PLAN_STATISTICS`** Hint kann in eine bestimmte sql_id eingefügt werden
  - Funktioniert nur mit SQL profiles und SQL patches
- Manchmal möchte man Result Cache für bestimmte sql_id einschalten
  - Ein Weg wäre Result Cache auf Objektebene auf force zu setzen
  - Wenn man es aber nicht für alle Queries auf dem Objekt einschalten möchte: → benutze man einen unsichtbaren **`RESULT_CACHE`** Hint
  - Bei meinen Tests funktionierte das nur mit SQL patches
- Andere gute Beispiele: **`APPEND, BIND_AWARE, CACHE, DYNAMIC_SAMPLING, MONITOR`**
- Im Allgemeinen können alle diese Hints

  „ **`SELECT name FROM v$sql_hint WHERE version_outline IS NULL`** “

  nicht in Stored Outlines und SQL plan baselines, aber in manchen Fällen in SQL Profiles und/oder SQL Patches verwendet werden

Privilegien

- Stored Outlines
  - **`create any outline, alter any outline, drop any outline`**
  - **`alter system`** or **`alter session`**
  - DML Privilegien auf 3 outln Tabellen
- SQL Profiles und SQL Patches
  - **`create any sql profile, alter any sql profile, drop any sql profile`** (deprecated)
  - **`administer sql management object`**
  - Für einige gezeigte Fälle: **`execute on sys.dbms_sqldiag_internal, sys.dbms_sqltune_internal`**
- SQL Plan Baselines
  - **`administer sql management object`**
  - Für einige gezeigte Fälle: **`select on v$sql, dba_sql_plan_baselines`**
- Im Allgemeinen ist es hilfreich, folgende Rechte zu haben:
  - **`select on cdb_/dba_ outlines, sql_profiles, sql_patches, sql_plan_baselines`**
  - Oder **`select any dictionary`**

Zusammenfassender Vergleich

	Stored Outlines	SQL Profiles	SQL Patches	SQL Plan Baselines
Verfügbar seit	8i	10g	11g	11g
Deprecated ?	Seit 11g	Nein	Nein	Nein
XE/SE/SE1 ?	Ja	Nein	Ja	Nein
Tuning Pack benötigt	Nein	Ja	Nein	Nein
Match nach Normalisierung	Ja	Ja	Ja	Ja
Literal-insensitiver Match (force_match=true)	Nein	Ja	Ja	Nein
Datenänderungen werden berücksichtigt	Nein	Etwas	Etwas	Ja
Nutzbar, um Hints in SQL unwirksam zu machen	Ja	Ja	Ja	Ja
Kategorien für verschiedene Workloads und Perioden	Ja	Ja	Ja	Nein
gather_plan_statistics Hint	Nein	Ja	Ja	Nein
result_cache Hint	Nein	Nein	Ja	Nein
Parallelverarbeitung ändern	Ja	Ja	Ja	Ja

Fazit

- Es ist möglich, unsichtbar Hints zu injizieren, auch dafür, um ungewünschte Hints zu „entfernen“
- Oracle treibt SMB und SQL Plan Baselines als zukünftiges Planstabilitätsfeature voran
- 4 verschiedene Technologien, jede hat seine eigenen Stärken und Limitierungen

- Die Welt ist nicht nur schwarz und weiß
- Manchmal auch ein bisschen rot, gelb und grün ☺
- Interessanterweise zeigen SQL Patches eine Menge grün
- Testen Sie sorgfältig und bedenken Sie, dass Hints in den meisten Fällen nur Workarounds sind!

Weitere Informationen...

www.trivadis.com

Für Quellen siehe nächste Seite

Quellenangabe

- Maria Colgan - How do I migrate stored outlines to SQL Plan Management? - https://blogs.oracle.com/optimizer/entry/how_do_i_migrate_stored
- Maria Colgan - How do I deal with a third party application that has embedded hints that result in a sub-optimal execution plan in my environment? - https://blogs.oracle.com/optimizer/entry/how_do_i_deal_with_a_third_party_application_that_has_embedded_hints_that_result_in_a_sub-optimal_ex
- Maria Colgan - Oracle Database Optimizer: Harnessing the Power of Optimizer Hints - http://www.nocoug.org/download/2012-11/NoCOUG_201211_Maria_Colgan_Optimizer_Hints.pdf
- Allison / Oracle Optimizer Blog - Additional Information on SQL Patches - https://blogs.oracle.com/optimizer/entry/additional_information_on_sql_patches
- Allison / Oracle Optimizer Blog - What should I do with old hints in my workload? - https://blogs.oracle.com/optimizer/entry/what_should_i_do_with_old_hints_in_my_workload
- Allison / Oracle Optimizer Blog - Using SQL Patch to add hints to a packaged application - https://blogs.oracle.com/optimizer/entry/how_can_i_hint_a
- Christian Antognini - SQL Profiles - http://antognini.ch/papers/SQLProfiles_20060622.pdf
- Christian Antognini - Troubleshooting Oracle Performance
- My Oracle Support - How to Specify Hidden Hints (Outlines) on SQL Statements in Oracle 8i (Doc ID 92202.1)
- Enkitec Blog - http://blog.enkitec.com/enkitec_scripts/exchange_outline_hints.sql
- Jonathan Lewis - Plan Stability in Oracle 8i/9i - http://www.jlcomp.demon.co.uk/04_outlines.rtf
- Jonathan Lewis - Hints on Hints - http://jonathanlewis.files.wordpress.com/2009/05/hints_on_hints.pdf
- Jonathan Lewis - Rules for Hinting - http://jonathanlewis.wordpress.com/2008/05/02/rules-for-hinting/
- Kerry Osborne - Licensing Requirements for SQL Profiles - http://kerryosborne.oracle-guy.com/2011/01/licensing-requirements-for-sql-profiles/

Fragen und Antworten...

Mathias Zarick

Principal Consultant

+43 664 85 44 295

Mathias.Zarick@trivadis.com

BASEL BERN BRUGG GENF LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

20 JAHRE TRIVADIS We love IT.

trivadis makes IT easier.