



Servicio Andaluz de Salud
CONSEJERÍA DE SALUD

*Oficina Técnica para la Gestión y Supervisión de
Servicios TIC
Subdirección de Tecnologías de la Información*

Estabilidad del Plan de Ejecución en Oracle. SQL Plan Management.

*Referencia documento:
InfV5_JASAS_EstabilidadPlanEjecucion_SPM_V920.docx
Fecha: 13 de noviembre de 2018
Versión: 9.2.0*

Registro de Cambios

Fecha	Autor	Versión	Notas
09 de Julio de 2015	Isidro Granados	1.0.0	Versión inicial
16 de Diciembre de 2015	Isidro Granados	6.2.0	Revisión de Diciembre de 2015, contrato 2014-2016
16 de Junio de 2016	Isidro Granados	7.1.0	Revisión de Junio de 2016, contrato 2014-2016
14 de octubre de 2016	Isidro Granados	7.2.0	Revisión de Noviembre de 2016, contrato 2014-2016
14 de junio de 2017	Isidro Granados	8.1.0	Revisión de Junio de 2017, contrato 2016-2018
16 de noviembre de 2017	Isidro Granados	8.2.0	Revisión de Octubre de 2017, contrato 2016-2018
16 de junio de 2018	Isidro Granados	9.1.0	Revisión de junio de 2018, contrato 2016-2018
13 de noviembre de 2018	Isidro Granados	9.2.0	Revisión de noviembre de 2018, contrato 2016-2018

Revisiones

Nombre	Role
Jonathan Ortiz	Advanced Support Engineer
Gregorio Adame	Advanced Support Engineer
Jose María Gómez	TAM

Distribución

Copia	Nombre	Empresa
1	Subdirección de Tecnologías de la Información	Servicio Andaluz de Salud, Junta de Andalucía
2	Servicio de Coordinación de Informática de la Consejería de Innovación	Consejería de Innovación, Junta de Andalucía

Índice de Contenidos

CONTROL DE CAMBIOS	4
INTRODUCCIÓN	5
OBJETIVOS DE ESTE DOCUMENTO	6
STORED OUTLINES.....	7
<i>Descripción</i>	<i>7</i>
<i>Casos de uso</i>	<i>7</i>
<i>Notas y buenas prácticas relativas al uso de outlines</i>	<i>8</i>
<i>Referencias/Documentación relativa a stored outlines</i>	<i>9</i>
SQL PROFILES.....	10
<i>Descripción</i>	<i>10</i>
<i>Casos de uso</i>	<i>11</i>
<i>Referencias/Documentación relativa a SQL Profiles</i>	<i>12</i>
SQL PLAN MANAGEMENT, SQL BASELINES.....	14
<i>Descripción</i>	<i>14</i>
<i>Beneficios</i>	<i>14</i>
<i>Conceptos</i>	<i>15</i>
<i>Componentes SPM</i>	<i>17</i>
<i>SPM Evolve Advisor</i>	<i>20</i>
<i>Visualizando planes en un SQL Plan Baseline.....</i>	<i>20</i>
<i>Almacenamiento</i>	<i>21</i>
<i>SPM versus Stored Outlines</i>	<i>22</i>
<i>SPM versus SQL Profiles</i>	<i>23</i>
<i>Ejemplo 1: Captura automática de SQL Plan Baselines filtrando las sentencias a capturar</i>	<i>24</i>
<i>Ejemplo 2: Creación SQL plan baseline manualmente desde la shared Cursor Area.....</i>	<i>34</i>
<i>Ejemplo 3: Cambio del plan de ejecución con Plan Baseline añadiendo un hint en la sentencia.</i>	<i>35</i>
<i>Ejemplo 4: Copia de SQL Plan Baseline entre bases de datos</i>	<i>40</i>
<i>Ejemplo 5: Creación de SQL Plan Baseline de un plan guardado en AWR.....</i>	<i>43</i>
<i>Referencias/Documentación recomendada relativa a SPM.....</i>	<i>46</i>

Control de cambios

Cambio	Descripción	Página
1	No se realizan cambios en esta versión del documento.	N/A

Introducción

Aunque el optimizador de Oracle está diseñado para evaluar el mejor plan posible de ejecución de forma transparente sin que el usuario tenga que intervenir, el plan de ejecución de una sentencia SQL puede cambiar de forma "inesperada", por distintas razones, por ejemplo: recalcular de estadísticas del optimizador, cambio de parámetros del optimizador, cambio de definiciones de esquemas, cambio de versión, etc.

Cada versión de Oracle el optimizador es mejorado para detectar el plan de ejecución más óptimo en cada momento. A pesar de esto, no es posible garantizar 100% que un plan cambiará siempre al plan más óptimo, esto hace que algunos clientes prefieran fijar su plan de ejecución antes de sufrir regresiones en el rendimiento de las aplicaciones.

Un cambio importante en el rendimiento de una aplicación puede ocurrir en el cambio del optimizador de Reglas (RBO) a Costes (CBO). El RBO fue desoportado en versión Oracle Database 10g, esto implica que aplicaciones antiguas al ser migradas a nuevas versiones de Oracle gran parte de su código SQL cambie el plan de ejecución, en muchas ocasiones para mejorar el tiempo de respuesta, pero también en algunas ocasiones un nuevo plan puede empeorar este tiempo. En estas ocasiones el uso de técnicas para fijar planes de ejecución puede ayudar a minimizar la revisión de código SQL de una aplicación.

Este informe hace un repaso de las técnicas o alternativas que se pueden usar en versión 12cR2 sin cambiar el código SQL de la sentencia para estabilizar el plan de ejecución o al menos asegurar que el nuevo plan de ejecución usado sea más óptimo que el generado inicialmente por el optimizador.

Las técnicas revisadas en el informe son:

- Stored Outlines. Disponible desde versión 9i. En 12c y 12cR2 deprecado.
- SQL Profiles. Disponibles a partir de versión 10g.
- SQL Plan Management. Disponible a partir de versión 11g.

El principal problema de fijar planes de ejecución es que se impide que los entornos nunca tomen ventaja de las nuevas funcionalidades o mejoras del optimizador, así como que pueda optar por mejores caminos de acceso a datos que mejorarían el rendimiento de las sentencias SQL. La nueva funcionalidad introducida en 11g SPM permite ir evolucionando los planes de ejecución, pero asegurando que sólo los planes conocidos o verificados como aquellos que tienen una mejora de rendimiento respecto del plan actual puedan ser usados. Es la técnica recomendada en versión 12.2.0.1.

Objetivos de este documento

- Indicar las técnicas o alternativas que se pueden usar desde versión 9i hasta versión 12cR2 sin cambiar el código SQL de una sentencia para estabilizar el plan de ejecución de la misma.
- Mostrar en detalle los conceptos y las características principales de la funcionalidad SQL Plan Management (SPM).
- Indicar las ventajas e inconvenientes en el uso de una u otras técnicas.
- Mostrar ejemplos y casos de uso de SPM.
- Dar referencias sobre documentación y tutoriales recomendados relativos a cada una de las técnicas analizadas.

NOTA: El informe está basado en versión 12.2.0.1 aunque muchos de los conceptos y ejemplos también son aplicables a versiones anteriores.

Stored Outlines

Descripción

Una Stored Outline es una funcionalidad introducida en versión Oracle 9i que permite estabilizar el plan de ejecución de una sentencia. Una Stored Outline es implementada como un conjunto de hints que están asociados a una sentencia SQL. Si una sentencia tiene un Stored Outline habilitada, Oracle automáticamente considerará los hints guardados para generar un plan de ejecución de acuerdo con esos hints, esto permite que la base de datos mantenga el plan de ejecución de forma consistente independientemente de los cambios de configuración o de estadísticas.

Las Stored Outlines pueden agruparse en categorías. Para usar una Stored Outline, la sesión (o la instancia) tendrán que tener la correspondiente categoría activada. Una outline sólo podrá estar en una categoría, pero una sentencia podrá tener varias outlines en diferentes categorías, por ejemplo, se puede tener la categoría OLTP y categoría DW, y una sentencia tenga un plan de ejecución en una categoría y otra en la otra, las sesiones se podrían activar la categoría deseada dependiendo de su naturaleza.

Se puede usar el comando `CREATE OUTLINE` para crear una stored outline, o crearlas de forma automática para todas las sentencias que se ejecuten, activando el parámetro `CREATE_STORED_OUTLINES` a nivel instancia o sesión.

Para ver si una base de datos está usando una outline la columna `OUTLINE_CATEGORY` de la vista `V$SQL` indicará la bbdd está aplicando la outline y la categoría a la que pertenece.

Para usar una outline la sesión o la instancia tendrán que tener activado el parámetro `USE_STORED_OUTLINES`.

Casos de uso

- Migraciones:

En un proceso de migración, para forzar que el optimizador use invariablemente un plan de ejecución, se podrían usar stored outlines para capturar los planes en una bbdd e importarlos en la base de datos migrada. También podrían ser usadas para prevenir los posibles problemas de rendimiento que podría encontrarse tras la migración en ciertas sentencias, para esto, se podría capturar antes de la migración Stored Outlines activando la recolección automática de todas las SQL ejecutadas en una categoría sin llegar a usarlas. Si tras la migración hay sentencias que empeoran el rendimiento y ninguna otra técnica de la nueva versión mejora el rendimiento se podría usar como una forma de restaurar el antiguo comportamiento para ciertas sentencias, bastaría activar una categoría de stored outlines e ir moviendo los outlines deseados a esta categoría.

- Para forzar a cambiar el plan de ejecución sin cambiar el código SQL.

En algunas ocasiones SQL generadas por aplicaciones cuyo código no es posible ser cambiado obtienen un plan de ejecución que no es el más óptimo. El DBA detecta que añadiéndole a la sentencia SQL que sufre la regresión de rendimiento uno o varios hints manualmente el rendimiento mejora. Con el uso de stored outlines es posible sustituir el plan de ejecución utilizado sin cambiar el código de la aplicación.

- Copia de planes entre base de datos.

Por ejemplo, se detecta que en desarrollo el plan de ejecución es mejor que en producción y se quiere fijar este plan en producción.

Notas y buenas prácticas relativas al uso de outlines

- En la versión Oracle 11g Oracle introdujo una nueva técnica llamada SQL Plan Management (SPM). SQL plan management crea SQL plan baselines, los cuales ofrecen una mayor estabilidad y rendimiento de los SQL comparados con los stored outlines.

Los stored outlines se siguen manteniendo en versión 12cR2 pero están considerados “deprecados” desde versión 12c, esto significa que la funcionalidad no será mejorada, pero está totalmente soportada. En futuras versiones Oracle ha anunciado que no soportará el uso de Stored Outlines en favor SQL plan management. Si se utilizan outlines, se puede considerar la migración de estos a SQL plan baselines, con el mecanismo que ofrece la versión 12cR2 para la migración automática. Ver

Oracle Database SQL Tuning Guide 12cR2

["Migrating Stored Outlines to SQL Plan Baselines"](#)

<http://docs.oracle.com/database/122/TGSQL/migrating-stored-outlines.htm#GUID-E1CBB4DA-2F83-4F4D-845B-CAAA8333DF1A>

- Un cambio futuro en la aplicación que usara sentencias para los que se hubieran creado outlines, tendría que ser tenido en cuenta porque si cambia el código SQL los outlines dejarán de utilizarse. Por tanto sería recomendable documentar todas las sentencias que requieren Stored Outlines.
- Las Stored Outlines no cambiarán con el tiempo, y aunque cuando se crean pueden ser buenos planes de ejecución, no se adaptarán a los cambios que va la bbdd sufriendo, y por tanto podrían convertirse en malos planes empeorando el rendimiento. El uso de SPM utiliza otras técnicas para evitar este problema.
- Los hints guardados en una Stored Outline se pueden convertir en inválidos, por ejemplo un index hint de un índice que se ha borrado. En ese caso, la bbdd todavía usa el outline pero excluye el hint inválido, produciendo otro plan que es a menudo peor que el original o del propio que el optimizador genera para la sentencia sin el outline.
- Cuando se utiliza outline, se pierde la capacidad de que el optimizador en versión superior pueda utilizar un mejor plan de ejecución. Por ejemplo muchas sentencias que mejoran con el cambio de versión por el uso de

outlines se impediría que se aprovechara esta mejora en el tiempo de respuesta de esas sentencias.

- No es posible añadir el parámetro `USE_STORED_OUTLINES` en `spfile` o `pfile`, de forma que cuando se reinicie la base de datos el optimizador siga usando la categoría de outlines que deseemos usar. Por tanto habrá que lanzar este `ALTER SYSTEM` cuando se reinicie la bbdd, o si se quiere automatizar, usar un trigger - after database startup trigger - que configure el parámetro como deseemos.
- Para consultar los datos relativos a las outlines se usarán las vistas `USER_OUTLINES`, `USER_OUTLINE_HINTS`, `DBA_OUTLINES`, `DBA_OUTLINE_HINTS`, cuyos datos están basados en las tablas `OL$`, `OL$HINTS`, y `OL$NODES`, que residen en el esquema `outln`.

El esquema `outln` guarda los datos en el tablespace `SYSTEM`. Si las outlines van a usar mucho espacio en el tablespace `SYSTEM` la recomendación es moverlas a un tablespace separado. Por ejemplo, si se utiliza el parámetro `CREATE_STORED_OUTLINES` y hay muchas sentencias con SQL literal, se podría llenar el tablespace `SYSTEM`. Para mover las tablas a un nuevo tablespace, usar el método indicado en: Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) http://docs.oracle.com/cd/E11882_01/server.112/e41573/outlines.htm#PFGRF94964 20.1.8 Moving Outline Tables

Referencias/Documentación relativa a stored outlines

Referencias/Documentación Relativa:

Oracle® Database SQL Tuning Guide 12c Release 2 (12.2)

[30 Migrating Stored Outlines to SQL Plan Baselines](#)

Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) manual
Chapter 20 Using Plan Stability

Notas MOS:

Note 67536.1 Stored Outline Quick Reference

Note 67536.1 132547.1 Using Stored Outlines

Note:728647.1 How to Transfer Stored Outlines from One Database to Another (9i and above)

Note 726802.1 Editing Stored Outlines in Oracle10g and Oracle11g

Note 144194.1 Editing Stored Outlines in Oracle9i - an example

SQL Profiles

Descripción

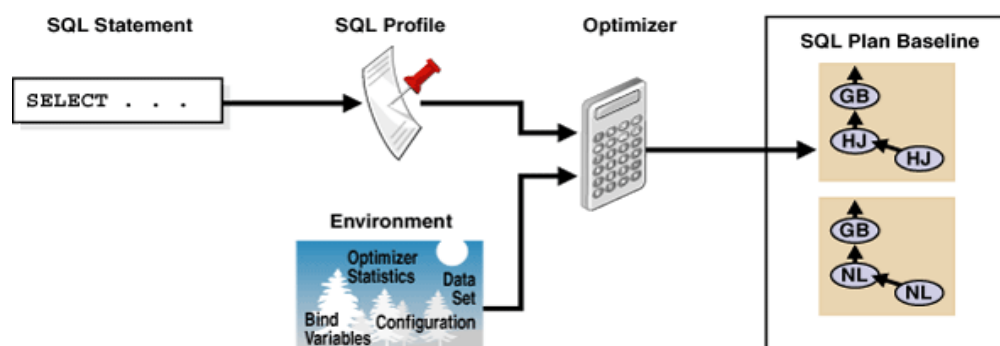
A partir de la versión 10g Oracle introduce la funcionalidad SQL Profiles. El objetivo de los SQL Profiles es generar un plan de ejecución más óptimo que el generado directamente por el optimizador.

Los SQL Profiles son un componente del Automatic Tuning Optimizer, para generarlos se usará el SQL Tuning Advisor, a través del package DBMS_SQLTUNE package o a través de Oracle Enterprise Manager. Estas funcionalidades están incluidas dentro del Tuning Pack, el cual tiene un coste extra de licencia dentro de la Enterprise Edition.

El SQL Tuning Advisor realizará una pseudo ejecución tomando métricas que le permiten conocer mayor detalle y analizar las correlaciones entre las columnas referenciadas en los predicados de la sentencia. El SQL Tuning Advisor generara un informe con recomendaciones de posible necesidad de creación de índices, reescritura de sentencias, cálculo de estadísticas, y finalmente podrá crear SQL Profiles que tendrán que ser aceptados explícitamente por el administrador.

Un SQL Profile es una colección de información guardada en el diccionario específica para una sentencia SQL que permite al optimizador crear un plan de ejecución más óptimo. Los SQL profiles también son implementados usando hints, pero estos hints no especificarán un plan concreto. En lugar de esto, los hints corrigen los posibles estimaciones erróneas que hace el optimizador y que le llevan a optar por un plan menos óptimo. Por ejemplo, un hint puede corregir la estimación de cardinalidad de una tabla. Se puede ver un SQL Profile a una sentencia SQL como las estadísticas a una tabla o un índice.

Un stored outline tiende a bloquear un plan, un SQL Profile hace más inteligente al optimizador.



Un SQL Profile no asegura que el plan de ejecución de una sentencia SQL no vaya a cambiar como haría con un Stored outline, esto es, ya que las tablas e índices van cambiando, el plan de ejecución de una sentencia puede cambiar con el mismo SQL Profile. Puede ser que sea necesario volver a ejecutar el SQL Tuning Advisor para regenerar el SQL Profile.

El uso de SQL Profile por parte de la base de datos es transparente al usuario.

Para crear SQL Profiles, a través del SQL Tuning Advisor se invoca al Automatic Tuning Optimizer para generar una recomendación SQL profile, estas serían las etapas:

- Etapas:
- Etapas 1: Crear un Tuning Task
- Etapas 2: Ejecutar el Tuning Task
- Etapas 3: Revisar Recomendaciones
- Etapas 4: Aceptar el Profile
- Etapas 5: Confirmar que el Profile está habilitado

El procedure `SET_TUNING_TASK_PARAMETERS` de los packages `DBMS_AUTO_SQLTUNE` o `DBMS_SQLTUNE` permite configurar distintos parámetros de la tuning task (`TIME_LIMIT`, `ACCEPT_SQL_PROFILES`, etc) dependiendo si la tarea es manualmente ejecutada o a través de las tareas automáticas `SYS_AUTO_SQL_TUNING_TASK`. Por ejemplo, `ACCEPT_SQL_PROFILES` determina cuando la base de datos automáticamente acepta un SQL Profile. Por defecto en 12cR2 el valor es `FALSE` esto es, que el administrador tendrá que explícitamente aceptarlo, o cambiar el parámetro.

Notas importantes relativas al uso de SQL Profiles

- Para la generación de SQL Profiles hay que lanzar tareas de SQL Tuning Advisor que son tareas que tardan bastante en ejecutarse y que son grandes consumidoras de CPU. Por defecto en 12cR2, hay una tarea automática en el periodo nocturno (`SYS_AUTO_SQL_TUNING_TASK`) que ejecuta SQL Tuning Advisor para las sentencias que AWR detecte que más recursos consume.
- SQL Tuning Advisor no sólo puede recomendar usar un SQL Profile, también puede hacer recomendaciones como creación de índices concretos. En un proceso de migración habría que revisar no sólo la recomendación de creación de SQL Profile, también la de los índices.

Casos de uso

- Se utilizan SQL Profiles para solucionar de forma sencilla problemas de rendimiento de sentencias SQL en cualquier base de datos. En caso de encontrar SQLs con un alto consumo de recursos, se puede invocar al SQL Tuning Advisor para que automáticamente genere recomendaciones como SQL Profiles. Igualmente la invocación del ADDM puede detectar estas SQLs y recomendar la ejecución del SQL Tuning Advisor sobre ellas.
- En un entorno de migración el uso de SQL Profiles puede ahorrar tiempo de revisión de sentencias SQL junto con el uso de SQL Performance Analyzer, ya que esta herramienta puede detectar qué sentencias tienen una regresión en el rendimiento, y con el uso de SQL Tuning Advisor se pueden automáticamente asignar a estas los SQL Profiles que se generen para minimizar esta regresión. Con esta técnica no se asegurarían los mismos planes de ejecución pero sí minimizar el impacto de una posible pérdida de rendimiento.

- Una gran ventaja con el uso de SQL Profiles respecto a outlines o SQL Plan Baselines, es que un SQL Profile permite adaptarse a sentencias SQL que usan literales en lugar de bind variables. Por defecto, el uso de SQL Profile creado para sentencias con literales será usado sólo cuando se ejecute la sentencia con los mismos valores de literales. En cambio existe la posibilidad de indicar `force_match=true` al aceptar un SQL Profile para cambiar este comportamiento.

Ejemplo: `dbms_sqltune.accept_sql_profile(task_name => 'my_sql_tuning_task', replace => TRUE, force_match => TRUE);`

Con esta opción, el SQL Profile también actuará para aquellas sentencias que tengan el mismo texto, después de normalizar los valores de los literales a bind variables. Esto es útil para aplicaciones que no utilicen bind variables y quieran compartir un SQL Profile. Si ambos valores literales y bind variables están en el SQL Text, o si el `force_match` se configura FALSE (default) entonces los literales no serán normalizados.

- También existen SQL Profiles que pueden "fijar" el plan y reproducir exactamente el mismo plan siempre, actuarían de forma equivalente a los "stored outlines"

Ejemplos:

Using Sqltplain to create a 'SQL Profile' to consistently reproduce a good plan (Doc ID 1487302.1)

How to use hints to customize SQL Profile or SQL PLAN Baseline (Doc ID 1400903.1)

Referencias/Documentación relativa a SQL Profiles

Referencias/Documentación Relativa:

Oracle® Database SQL Tuning Guide 12c Release 2 (12.2)

27 Managing SQL Profiles

<http://docs.oracle.com/database/122/TGSQL/managing-sql-profiles.htm#TGSQL596>

Oracle® Database 2 Day DBA 12c Release 2 (12.2.0.1)

10 Monitoring and Tuning the Database

<https://docs.oracle.com/database/122/ADMQS/monitoring-and-tuning-the-database.htm#GUID-95716F9A-C500-4CCD-AC3F-6541E3191E40>

Oracle Database Online Documentation 12c Release 2 (12.2.0.1)

Database PL/SQL Packages and Types Reference

Section 157 DBMS_SQLTUNE

https://docs.oracle.com/database/122/ARPLS/DBMS_SQLTUNE.htm#GUID-821462BF-1695-41CF-AFF7-FD23E9999C6A

Notas MOS:

Automatic SQL Tuning and SQL Profiles (Doc ID 271196.1)

How To Use SQL Profiles for Queries Using Different Literals Using the Force_Match Parameter of DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (Doc ID 1253696.1)

SQL Plan Management. SQL Baselines

Descripción

SQL Plan Management (SPM) es el mecanismo que permite al optimizador manejar planes de ejecución, asegurando que la base de datos sólo usará planes conocidos o verificados.

El primer objetivo del SQL Plan Management es prevenir regresiones de rendimiento causadas por cambios de planes. El segundo objetivo es ser capaz de adaptarse a los cambios como por ejemplo nuevas estadísticas o nuevos índices verificando y aceptando sólo los nuevos planes que mejoren el rendimiento.

SPM depende de un mecanismo llamado SQL plan baselines. Un SQL plan baseline contiene uno o más planes de ejecución aceptados. El plan history es el conjunto de planes aceptados y no aceptados que el optimizador va generando a lo largo del tiempo, sólo los aceptados estarán en el SQL plan baseline.

SQL Plan Management está habilitado por defecto con la Enterprise Edition y no requiere un coste extra de licencia.

Beneficios

- El uso de SPM permite prevenir las posibles regresiones que puedan tener las sentencias SQL por posibles cambios de los planes de ejecución:
 - o Cambio de estadísticas de objetos o sistema
 - o Cambios en el esquema como creación de nuevos índices
 - o Cambio de parámetros a nivel de instancia o sesión
- Permite que se sigan evolucionando los planes ya que el optimizador seguirá capturando posibles nuevos planes de ejecución, que podrán ser verificados y aceptados en caso de que sean mejores para que el usuario final los use.
- Facilita migraciones de base de datos:

Se pueden descargar directamente los planes de ejecución en un SQL Plan Baseline y transportarlos entre base de datos. Esto permite que en un escenario de migración de base de datos, se puedan transportar los planes de ejecución bien probados a la base de datos migrada, reduciendo así el riesgo de regresión.

Especialmente en migraciones de Oracle Database 10g y 11g a versiones 12c o 12cR2 puede ser muy útil, ya que se permite que las sentencias capturadas en un SQL Tuning Set desde 10g/11g se carguen directamente en una SQL Plan Baseline de una base de datos migrada a 12cR2.

- Despliegue controlado de nuevos módulos de aplicación:

Se pueden validar los planes en entornos de prueba o preproducción e importarlos en el entorno productivo para asegurar que no van a ocurrir regresiones por cambios del plan de ejecución.

Conceptos

Los principales componentes de SPM son:

- Plan Capture: Este componente guarda información relevante sobre los planes de una o varias sentencias SQL.
- Plan Selection: Este componente es la detección por el optimizador de los cambios de planes que se van guardando en el plan history, y el uso de SQL plan baselines para la selección del plan apropiado para evitar potenciales regresiones de rendimiento.
- Plan Evolution: Este componente es el proceso de añadir nuevos planes a SQL plan baselines existentes, bien de forma manual o automáticamente.

Para el administrar SPM, se usará Enterprise manager o vistas DBA_SQL_PLAN_BASELINES y una PLSQL API. En EM, en Query Optimizer -> SQL Plan Control -> SQL Plan Baseline se podrá ver toda la información de SQL plan baselines y operaciones (aceptar planes en un SQL plan baseline, cargar planes desde cache cursor o un sql_id o un STS, etc).

También se puede ver en V\$SQL cuando una sentencia está usando un SQL plan baseline (nueva columna sql_plan_baseline).

Hay 2 parámetros de inicialización (a nivel de instancia y/o sesión) para el manejo de SPM:

- OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES: A true captura SQL plan baselines para sentencias SQL que se hayan ejecutado más de una vez. Por defecto false. Cuando se ejecute el plan de ejecución actual se creará como SQL plan baseline que será marcado como aceptado. Los siguientes planes que se creen para estas sentencias no serán aceptados hasta que sean evolucionados (dbms_spm.evolve_sql_plan_baseline).
- OPTIMIZER_USE_SQL_PLAN_BASELINES: Controla el uso de SQL Plan Baselines. Por defecto true.

Los valores de configuración de estos parámetros son independientes. Por ejemplo, si OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES está a true, entonces la base de datos creará SQL plan baselines iniciales independientemente de que el parámetro OPTIMIZER_USE_SQL_PLAN_BASELINES esté a true o a false.

Estados del plan respecto del SQL plan baseline:

En la vista dba_sql_baselines hay 4 columnas importantes que determinan el estado del plan: **enabled, accepted, fixed, autopurge**

Ejemplo:

```
select sql_handle, substr(sql_text,1,30) "SQL", plan_name,  
origin,parsing_schema_name,  
enabled, accepted, fixed, autopurge  
from dba_sql_plan_baselines;
```

SQL_HANDLE	SQL	PLAN_NAME		
ORIGIN	ENA	ACC	FIX	AUT

```
-----  
-----  
SQL_356584134d867c7f  select  *    from  tabla1  where  col  
SQL_PLAN_3atc42d6scz3zafa6595a  MANUAL-LOAD      YES  YES  YES  YES
```

Enabled: El plan es elegible para ser usado por el optimizador. Automáticamente la base de datos marca todos los planes en el plan history como "enabled" aunque todavía estén no-aceptados. Se puede manualmente cambiar un plan de enabled a disabled, y hacer que el optimizador deje de usarlo aunque el plan haya sido aceptado. Ejemplo:

```
DECLARE  
    report  varchar2(1000);  
BEGIN  
report := dbms_spm.ALTER_SQL_PLAN_BASELINE  
('SQL_356584134d867c7f',  
 'SQL_PLAN_3atc42d6scz3zafa6595a',  
 ATTRIBUTE_NAME => 'ENABLED',  
 ATTRIBUTE_VALUE => 'NO');  
END;  
/
```

Accepted: Un plan es aceptado si y sólo si está en el plan baseline. El plan history de una sentencia contiene todos los planes, aceptados y no-aceptados. Después de que el optimizador genera el primer plan aceptado en un plan baseline, todos los siguientes planes no-aceptados serán añadidos al plan history, esperando verificación pero no estarán en el SQL plan baseline. Para pasar de no-aceptado a aceptado habrá que hacer un evolve del plan.

Fixed: Un fixed plan es un plan aceptado que es marcado como preferente, de forma que el optimizador considere sólo los planes fixed. Si hay más de un plan fixed, el optimizador seleccionará el que menos coste tenga entre ellos. Además en caso de que exista un fixed plan, el optimizador deja de hacer nuevas capturas de planes al plan history.

Para cambiar a fixed:

```
DECLARE  
    report  varchar2(1000);  
BEGIN  
report := dbms_spm.ALTER_SQL_PLAN_BASELINE  
('SQL_356584134d867c7f',  
 'SQL_PLAN_3atc42d6scz3zafa6595a',  
 ATTRIBUTE_NAME => 'FIXED',  
 ATTRIBUTE_VALUE => 'YES');  
END;  
/
```

Autopurge: Indica cuando un plan si no es usado, automáticamente será borrado del plan history después de un tiempo concreto, a menos de que el autopurge se deshabilite. Por defecto 53semanas, aunque puede cambiarse como se indicará en el apartado de almacenamiento.

Componentes SPM

Plan Capture

El SQL plan capture se refiere a las técnicas para capturar y guardar información relevante sobre planes en el SQL Management Base para un conjunto de sentencias SQL. Capturar un plan significa hacer al SQL Plan Management ser consciente del plan.

Se puede capturar automáticamente con el parámetro `optimizer_capture_sql_plan_baselines` a `true`, a nivel de sesión o manualmente usando el package `DBMS_SPM`.

Captura automática:

Básicamente, si la captura automática está habilitada, si la base de datos ejecuta una sentencia SQL más de una vez, el algoritmo de captura actuará de esta forma:

- Si el SQL plan baseline no existe, creará un SQL plan baseline para la sentencia añadiéndole al baseline ese plan, y marcará ese plan inicial como aceptado y lo ejecutará.
- Si el SQL plan baseline existe, comprueba si el plan ya está en el SQL plan baseline, en caso de que esté, no añade nada y lo usará, en caso de que no esté añadirá el nuevo plan al baseline como no-aceptado y ejecutará otro plan del SQL plan baseline.

También es posible restringir las sentencias repetidas que van a ser capturadas creando un filtro con el procedure `DBMS_SPM.CONFIGURE`. De esa forma podemos evitar que cuando se active la captura automática no se capturen sentencias para las que no queremos SQL Plan Baselines y así evitar el crecimiento de espacio en el tablespace `SYSAUX`.

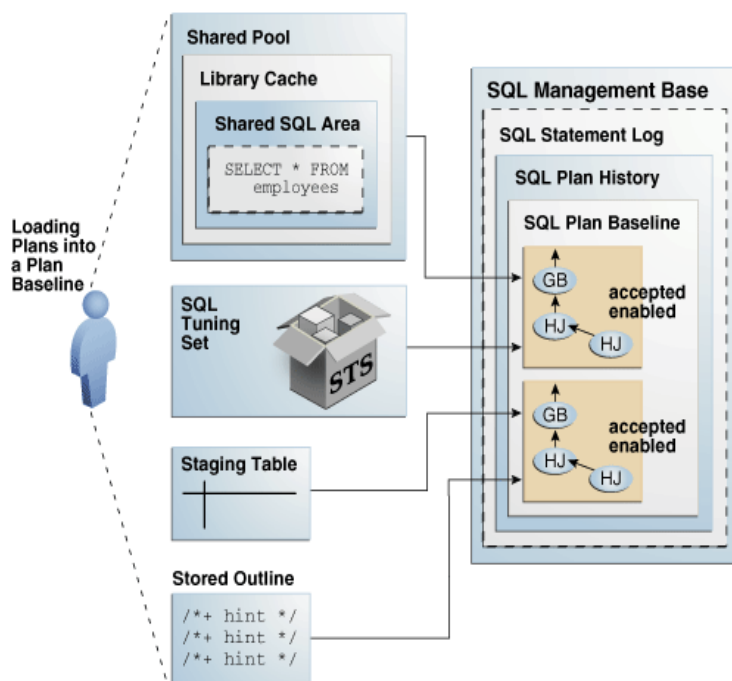
Un filtro puede incluir (`allow=>TRUE`) o excluir (`allow=>FALSE`) los planes de las sentencias que cumplan los valores que se especifican. Para poder ser capturada la sentencia además de ser repetitiva no deberá ser excluida por ningún filtro. El procedure soporta filtros para el texto del SQL, el nombre del esquema que parsea, module y action. En la vista `DBA_SQL_MANAGEMENT_CONFIG` se pueden ver los filtros configurados.

El ejemplo 1 más adelante mostrará un ejemplo de uso de captura automática con filtros.

Captura manual:

Con la captura manual, a través de EM o PL/SQL, se podrán cargar los plan de ejecución de sentencias SQL desde distintas fuentes: sentencias dentro de un SQL tuning set (STS), desde la shared SQL area, desde una tabla "staging", o desde una stored outline. En los ejemplos indicados en el informe en las siguientes secciones se mostrará cómo utilizar estas distintas formas de capturar SQL Plan Baselines.

La siguiente gráfica muestra las distintas formas de cargar planes manualmente a un SQL plan baseline:

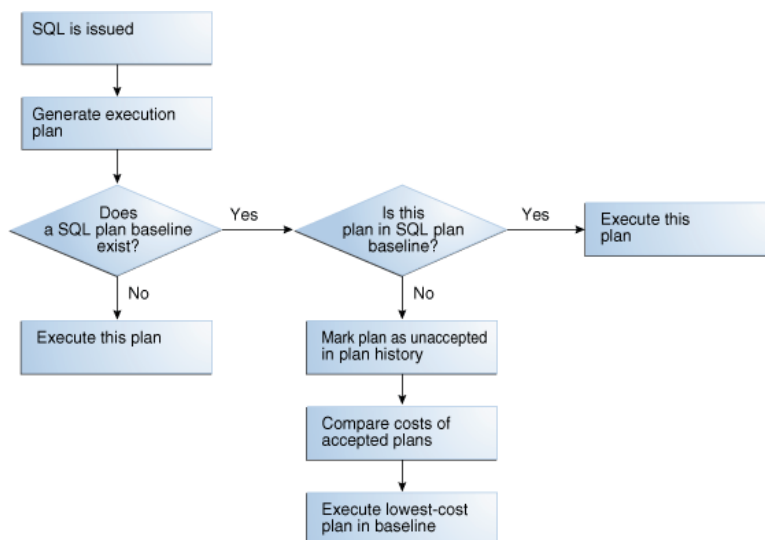


Plan Selection

Si el parámetro `OPTIMIZER_USE_SQL_PLAN_BASELINES` está a `true` a nivel de sesión, cuando se ejecuta una nueva sentencia SQL el optimizador hace un hard parse de la sentencia y calcula el plan de ejecución basándose en el coste del plan, tras esto comprueba si existe un SQL plan baseline. Si no existe SQL plan baseline usa el plan generado por el optimizador. Si existe el SQL Plan Baseline, comprueba si el plan generado coincide con alguno de los guardados en el SQL Plan Baseline en tal caso lo usará excepto si existe un plan fixed que tendrá preferencia, si no está en el SQL Plan baseline, lo pondrá para verificación en el plan history como un non-accepted plan, y cogerá el plan de los SQL Plan baseline que están verificados y aceptados que tenga menos coste, o el fixed si existiera, o el fixed con menos coste si hay varios fixed.

Si los planes en el SQL plan baseline no son reproducibles, lo que podría ocurrir por ejemplo si todos los planes de un baseline hacen referencia a un índice que ha sido borrado, entonces el optimizador usará el nuevo plan generado basado en el coste.

La siguiente gráfica muestra el árbol de decisión para el SQL plan selection:



Plan Evolution

El SQL plan evolution es el proceso por el cual el optimizador verifica nuevos planes y los añade como aceptados al SQL plan baseline.

Consiste en 2 etapas:

- Verificar si un plan no-aceptado tiene un rendimiento al menos como los planes aceptados del SQL plan baseline (plan verification)
- Añadir planes no-aceptados al plan baseline como aceptados bien después de verificar que tienen mejor rendimiento o sin verificar.

Se puede configurar SQL plan management para realizar las etapas de evolution de forma independiente. Esto es:

- Se puede verificar planes y automáticamente ser aceptados si tienen mejor rendimiento.
- Se puede añadir planes sin ser verificados.
- Se pueden verificar pero no ser aceptados.

Se puede manualmente realizar estas etapas utilizando DBMS_SPM.CREATE_EVOLVE_TASK para crear la tarea, DBMS_SPM.EXECUTE_EVOLVE_TASK para ejecutarla y DBMS_SPM.IMPLEMENT_EVOLVE_TASK para implementar las recomendaciones y añadir los planes.

También se podrá delegar en la tarea automática de mantenimiento SQL Evolve Advisor (SYS_AUTO_SPM_EVOLVE_TASK) que realizará automáticamente estas acciones:

- Selecciona y asigna un rango a los planes no aceptados para verificarlos.
- Acepta aquellos planes que satisfacen un determinado umbral de rendimiento.

SPM Evolve Advisor

SPM Evolve Advisor es un SQL advisor que realiza el plan evolve de aquellos planes que han sido recientemente añadidos a un SQL plan baseline.

El SPM Evolve Advisor por defecto añade una tarea de mantenimiento automático llamada SYS_AUTO_SPM_EVOLVE_TASK que se ejecuta diariamente en la ventana de mantenimiento del Scheduler. La tarea realiza las siguientes acciones de forma automática:

- Selecciona y marca un ranking de los planes no-aceptados de los SQL Baselines existentes y realiza tantas verificaciones de planes cómo es posible durante la ventana de mantenimiento.
- Acepta cada plan si se considera que mejora el rendimiento.

El advisor simplifica la evolución de los planes ya que elimina tener que hacer el evolve manualmente. Es la opción recomendada por Oracle para hacer el evolve de los planes.

El propietario de la tarea es SYS, y se deshabilita y habilita conjuntamente con el Automatic SQL Tuning Advisor. Se permite con el procedure SET_EVOLVE_TASK_PARAMETER cambiar ciertos parámetros de la tarea:

- ACCEPT_PLANS. Por defecto a true. Indicará si los planes verificados recomendados son aceptados automáticamente. Cuando se configura a false, la tarea verifica los planes pero no realiza el evolve de los planes.
- TIME_LIMIT. Timeout de la tarea para un SQL. Por defecto DBMS_SPM.AUTO_LIMIT en minutos. El default DBMS_SPM.AUTO_LIMIT deja al sistema escoger el límite más apropiado dependiendo del número de verificaciones que tiene que realizar.

NOTA: Aunque el uso de SPM no requiere licenciamiento extra, en caso de uso de SYS_AUTO_SPM_EVOLVE_TASK se requiere licenciar la opción de Tuning Pack.

Visualizando planes en un SQL Plan Baseline

Desde versión 12c, el SMB también guarda el plan para nuevos planes añadidos al "plan history". Cuando se utiliza DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE la función muestra el plan guardado en el SMB.

Con la siguiente consulta se podrán consultar los SQL Plan Baselines y plan de ejecución para un SQLID concreto:

```
SELECT PLAN_TABLE_OUTPUT
FROM   V$SQL s, DBA_SQL_PLAN_BASELINES b,
       TABLE (
DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE(b.sql_handle,b.plan_name,'basic
')
) t
WHERE  s.EXACT_MATCHING_SIGNATURE=b.SIGNATURE
AND    b.PLAN_NAME=s.SQL_PLAN_BASELINE
AND    s.SQL_ID='31d96zzzpcys9';
```

Almacenamiento

La información del SQL Plan Management es guardado en el SQL management base (SMB) dentro del tablespace SYSAUX.

El SMB es un repositorio lógico en el data dictionary que contiene:

- SQL statement log. Esto es usado cuando hay captura automática de SQL Plan Baselines para registrar el SQL ID de las sentencias para saber cuando son repetidas. Si un filtro excluye una sentencia de la captura, también excluirá del log el correspondiente SQL ID.
- SQL plan history, que incluye las SQL Plan Baselines
- SQL profiles
- SQL patches

SQL Plan history:

El SQL plan history es el conjunto de planes que son generados a lo largo del tiempo para sentencias SQL. Contiene SQL plan baselines y planes no-aceptados.

El uso de espacio del SMB dentro del SYSAUX puede controlarse modificando 2 parámetros:

- SPACE_BUDGET_PERCENT: Máximo porcentaje del espacio en SYSAUX que el SQL management puede usar. El valor por defecto es 10. Se puede cambiar entre 1% y 50%.
- PLAN_RETENTION_WEEKS: Número de semanas que permanecerán los planes no-usados antes de ser borrados. El valor por defecto es 53. Se puede cambiar a un valor entre 5 y 523 semanas.

La vista DBA_SQL_MANAGEMENT_CONFIG muestra la configuración actual del SMB:

```
SELECT parameter_name, parameter_value
FROM   dba_sql_management_config;

PARAMETER_NAME                                PARAMETER_VALUE
-----
SPACE_BUDGET_PERCENT                           10
PLAN_RETENTION_WEEKS                          53

2 rows selected.
```

Un proceso background semanalmente medirá el espacio total ocupado por el SMB, cuando el espacio excede el límite, escribirá un warning en el alert.log. Esta alerta continuará repitiéndose semanalmente hasta que se amplíe el límite de espacio para SMB (SPACE_BUDGET_PERCENT), se aumente el tamaño del SYSAUX o se baje el espacio usado por el SMB borrando objetos del SQL management (SQL plan baselines, SQL profiles o SQL patches).

Igualmente, una tarea automática semanalmente irá borrando planes que no han sido usados durante el periodo de retención (PLAN_RETENTION_WEEKS), y que son identificados por el valor LAST_EXECUTED en el SMB para ese plan.

A continuación se indica cómo cambiar ambos valores, por ejemplo:

```
BEGIN
  DBMS_SPM.configure('space_budget_percent', 20);
  DBMS_SPM.configure('plan_retention_weeks', 24);
END;
/

SELECT parameter_name, parameter_value
FROM   dba_sql_management_config;

PARAMETER_NAME                                PARAMETER_VALUE
-----
SPACE_BUDGET_PERCENT                           20
PLAN_RETENTION_WEEKS                          24

2 rows selected.
```

Para revisar el espacio actualmente ocupado:

```
SELECT PARAMETER_NAME, PARAMETER_VALUE AS "%_LIMIT",
       ( SELECT sum(bytes/1024/1024) FROM DBA_DATA_FILES
         WHERE TABLESPACE_NAME = 'SYSAUX' ) AS SYSAUX_SIZE_IN_MB,
       PARAMETER_VALUE/100 *
       ( SELECT sum(bytes/1024/1024) FROM DBA_DATA_FILES
         WHERE TABLESPACE_NAME = 'SYSAUX' ) AS "CURRENT_LIMIT_IN_MB"
FROM DBA_SQL_MANAGEMENT_CONFIG
WHERE PARAMETER_NAME = 'SPACE_BUDGET_PERCENT';

PARAMETER_NAME                                %_LIMIT SYSAUX_SIZE_IN_MB CURRENT_LIMIT_IN_MB
-----
SPACE_BUDGET_PERCENT                           10          211.4375          21.14375
```

SPM versus Stored Outlines

Aunque los stored outlines se siguen manteniendo en versión 12cR2 como un requisito legal para permitir plan stability, es una funcionalidad deprecada desde versión 12c y Oracle recomienda el cambio al uso del SPM para asegurar planes de ejecución. Los SQL plan baselines ofrecen una mayor estabilidad y rendimiento de los SQL comparados con los stored outlines. En futuras versiones Oracle ha anunciado que desoportará el uso de Stored outlines en favor SQL plan management. En caso de que una sentencia tenga un outline y además un SQL Plan baseline aceptado, utilizará el outline. Algunas ventajas del uso de SPM respecto a Stored Outlines:

- Los stored outlines no pueden automáticamente evolucionar en el tiempo. Esto es, aunque un stored outline puede ser bueno cuando se crea, se puede convertir en un mal plan después de que cambie la bbdd, produciendo una pérdida de rendimiento. Con SPM podemos habilitar al optimizador a usar un plan, pero

seguir evolucionado, capturando otros que pueden ser mejores cuando cambie la base de datos y que podrán ser validados y aceptados.

- Los hints guardados en una stored outline se puede convertir en inválidos, por ejemplo un index hint de un índice que se ha borrado. En ese caso, la bbdd todavía usa el outline pero excluye el hint inválido, produciendo un plan que es a menudo peor que el original, o del propio que el optimizador genera para la sentencia. Con SPM el plan no será utilizable y el optimizador usará otro.
- El optimizador no puede elegir otros outlines en diferentes categorías. Con SPM podremos tener varios planes aceptados.

SPM versus SQL Profiles

Pueden verse como diferentes alternativas para influenciar al optimizador a elegir un plan de ejecución más óptimo:

- SPM baselines (Más proactivos):
 - o Fuerza un plan específico y garantiza una estabilidad del plan.
 - o Contiene múltiples planes
 - o Puede evolucionar guardando nuevos planes que puedan potencialmente mejorar el rendimiento
 - o Sólo se utilizarán si la sentencia SQL coincide completamente.
 - o Sólo planes que han sido explícitamente aceptados serán usados
 - o No requiere coste adicional
- SQL Profiles (Más reactivos)
 - o No fuerza un plan específico ni contiene un plan.
 - o No garantiza la estabilidad del plan, el plan puede cambiar.
 - o Ayuda al optimizador a calcular un plan mejor para una sentencia SQL.
 - o Es automáticamente generado por el SQL Tuning Advisor, y de forma rápida se pueden solucionar problemas de rendimiento de un plan de ejecución
 - o Requiere licencia de Tuning Pack
 - o Se pueden generar y aceptar automáticamente desde una sesión del SQL Performance Analyzer, para sentencias SQLs que se hayan degradado
 - o Una gran ventaja es que un SQL Profile permite adaptarse a sentencias SQL que usan literales en lugar de bind variables.

Ambas funcionalidades se integran:

- Si existe un SQL plan baseline para la sentencia SQL y se implementa una recomendación de un SQL Profile por el SQL Tuning Advisor, el plan de ejecución se añadirá como un plan aceptado al SQL plan baseline.
- Si para una sentencia SQL existe un SQL profile y un SQL plan baseline, si el plan generado por el CBO con el SQL Profile no existe en el SQL plan baseline,

este se añade como un plan no-aceptado y usará uno aceptado del SQL plan baseline. El generado por el sql profile solo se usará cuando sea aceptado.

Ejemplo 1: Captura automática de SQL Plan Baselines filtrando las sentencias a capturar

- Para el ejemplo utilizaremos 2 sesiones en base de datos, una como usuario user1 y otra como SYSDBA:

1. Cargamos datos para la prueba

```
SQL> conn user1/user1
SQL> create table tabla1 (col1 number, col2 varchar2(128));
Table created.
SQL> insert into tabla1 SELECT OBJECT_ID, OBJECT_NAME FROM DBA_OBJECTS;
72882 rows created.
SQL> commit;
Commit complete.
SQL> exec dbms_stats.gather_table_stats('USER1','TABLA1');

SQL> conn user2/user2
SQL> create table tabla2 (col1 number, col2 varchar2(128));
Table created.
SQL> insert into tabla2 SELECT OBJECT_ID, OBJECT_NAME FROM DBA_OBJECTS;
72883 rows created.
SQL> commit;
Commit complete.
SQL> exec dbms_stats.gather_table_stats('USER2','TABLA2');
```

2. Antes de activar la captura automática vamos a añadir filtros. En este caso por ejemplo se indicará que sólo las sentencias que utilicen el esquema user1 en el parse y que tengan el texto SISPM en el código SQL sean capturadas para esto:

- Revisamos que no hay ningún filtro previo:

```
SQL> COL PARAMETER_NAME FORMAT a32
SQL> COL PARAMETER_VALUE FORMAT a32

SQL> SELECT PARAMETER_NAME, PARAMETER_VALUE
FROM   DBA_SQL_MANAGEMENT_CONFIG
WHERE  PARAMETER_NAME LIKE '%AUTO%';
```

PARAMETER_NAME	PARAMETER_VALUE
AUTO_CAPTURE_PARSING_SCHEMA_NAME	
AUTO_CAPTURE_MODULE	
AUTO_CAPTURE_ACTION	
AUTO_CAPTURE_SQL_TEXT	

- Añadimos los filtros y los comprobamos:

```
SQL> EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_PARSING_SCHEMA_NAME','user1',true);
SQL> EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_SQL_TEXT','%SISPM%',true);

SQL> SELECT PARAMETER_NAME, PARAMETER_VALUE
FROM   DBA_SQL_MANAGEMENT_CONFIG
WHERE  PARAMETER_NAME LIKE '%AUTO%';
```

PARAMETER_NAME	PARAMETER_VALUE
AUTO_CAPTURE_PARSING_SCHEMA_NAME	parsing_schema IN (USER1)
AUTO_CAPTURE_MODULE	
AUTO_CAPTURE_ACTION	
AUTO_CAPTURE_SQL_TEXT	(sql_text LIKE %SISPM%)

3. Activamos captura de SQL plan baselines

```
SQL> conn / as sysdba
SQL> alter system set optimizer_capture_sql_plan_baselines = TRUE;
```

4. Ejecutamos 2 veces estas sentencias:

```
SQL> conn user1/user1
SQL> SELECT /* SISPM */ * from tabla1 where col1=1;
SQL> SELECT /* SISPM */ * from tabla1 where col1=1;
SQL> SELECT * from tabla1 where col1=2;
SQL> SELECT * from tabla1 where col1=2;

SQL> conn user2/user2
SQL> SELECT /* SISPM */ * from tabla2 where col1=1;
SQL> SELECT /* SISPM */ * from tabla2 where col1=1;
SQL> SELECT * from tabla2 where col1=2;
SQL> SELECT * from tabla2 where col1=2;
```

5. Desactivamos captura de SQL plan baselines

```
SQL> conn / as sysdba
SQL> alter system set optimizer_capture_sql_plan_baselines = FALSE;
```

6. Desde otra sesión como administrador vemos que se ha creado sólo un SQL plan baseline para la sentencia que cumplía el filtro y por defecto está el plan aceptado:

```
column SQL_HANDLE format a20
column SQL format a31
column PARSING_SCHEMA_NAME format a10
column plan_name format a30
column origin format a12
set linesize 170
select sql_handle, substr(sql_text,1,30) "SQL", plan_name, origin,parsing_schema_name,
enabled, accepted, fixed, autopurge from dba_sql_plan_baselines;
```

SQL_HANDLE	SQL	PLAN_NAME
ORIGIN	PARSING_SC	ENA ACC FIX AUT
SQL_e96db75765a939eb	SELECT /* SISPM */ * from tabl	SQL_PLAN_fkvdaxkukfgbafa6595a
AUTO-CAPTURE USER1	YES YES NO YES	

7. Revisamos el plan guardado en el sql plan baseline:

```
select * from table(DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE('SQL_e96db75765a939eb'));
```

PLAN TABLE OUTPUT

```
-----
SQL handle: SQL_e96db75765a939eb
SQL text: SELECT /* SISPM */ * from tabla1 where coll=1
-----
```

```
-----
Plan name: SQL_PLAN_fkvdraxkukfgbafa6595a      Plan id: 2946914650
Enabled: YES      Fixed: NO      Accepted: YES      Origin: AUTO-CAPTURE
Plan rows: From dictionary
-----
```

PLAN_TABLE_OUTPUT

Plan hash value: 3375727661

```
-----
| Id | Operation          | Name   | Rows | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT   |        |     5 |    395 |    137   (1)| 00:00:01 |
|*  1 | TABLE ACCESS FULL| TABLA1 |     5 |    395 |    137   (1)| 00:00:01 |
-----
```

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

```
-----
1 - filter("COL1"=1)
-----
```

Note

```
-----
- dynamic statistics used: dynamic sampling (level=2)
-----
```

8. Revisamos que es el plan que utiliza en la ejecución

```
SQL> conn user1/user1
SQL> SELECT /* SISPM */ * from tabla1 where coll=1;
SQL> select * from table(dbms_xplan.display_cursor);
```

PLAN_TABLE_OUTPUT

SQL ID 24puppz0qh3mw, child number 0

SELECT /* SISPM */ * from tabla1 where coll=1

Plan hash value: 3375727661

```
-----
| Id | Operation          | Name   | Rows | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT   |        |     5 |    395 |    137 (100)| 00:00:01 |
|*  1 | TABLE ACCESS FULL| TABLA1 |     5 |    395 |    137   (1)| 00:00:01 |
-----
```

PLAN_TABLE_OUTPUT

Predicate Information (identified by operation id):

```
-----
1 - filter("COL1"=1)
-----
```

Note

```
-----
- dynamic statistics used: dynamic sampling (level=2)
- SQL plan baseline SQL_PLAN_fkvdraxkukfgbafa6595a used for this statement
-----
```

9. Desde la sesión de usuario creamos un índice nuevo y ejecutamos la sentencia y vemos el plan que sigue cogiendo el SQL plan baseline:

```
SQL> create index i_tab1 on tabla1(coll1);
Index created.
SQL> SELECT /* SISPM */ * from tabla1 where coll1=1;
no rows selected

SQL> select * from table(dbms_xplan.display_cursor);

PLAN_TABLE_OUTPUT
-----
SQL_ID  24puppz0qh3mw, child number 0
-----
SELECT /* SISPM */ * from tabla1 where coll1=1

Plan hash value: 3375727661

-----
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0  | SELECT STATEMENT   |        |       |       |  137 (100)|          |
|*  1  | TABLE ACCESS FULL| TABLA1 |     5 |   395 |   137   (1)| 00:00:01 |
-----

PLAN_TABLE_OUTPUT
-----

Predicate Information (identified by operation id):
-----

   1 - filter("COL1"=1)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - SQL plan baseline SQL_PLAN_fkvdraxxkukfgbafa6595a used for this statement
```

10. Desde la otra sesión como administrador vemos que se ha creado un nuevo plan no-aceptado, vemos sus planes de ejecución:

```
column SQL_HANDLE format a20
column SQL format a31
column PARSING_SCHEMA_NAME format a10
column plan_name format a30
column origin format a12
set linesize 170
select sql_handle, substr(sql_text,1,30) "SQL", plan_name, origin,parsing_schema_name,
enabled, accepted, fixed, autopurge from dba_sql_plan_baselines;
```

SQL_HANDLE	SQL	PLAN_NAME			
ORIGIN	PARSING_SC	ENA	ACC	FIX	AUT
SQL_e96db75765a939eb	SELECT /* SISPM */ * from tabl	SQL_PLAN_fkvdraxxkukfgbafa6595a			
AUTO-CAPTURE USER1	YES YES NO YES				
SQL_e96db75765a939eb	SELECT /* SISPM */ * from tabl	SQL_PLAN_fkvdraxxkukfgbb2f25575			
AUTO-CAPTURE USER1	YES NO NO YES				

```
select * from table(dbms_xplan.display_sql_plan_baseline( sql_handle =>
'SQL_e96db75765a939eb', format => 'basic'));
```

PLAN_TABLE_OUTPUT

```
-----
SQL handle: SQL_e96db75765a939eb
SQL text: SELECT /* SISPM */ * from tabla1 where coll=1
-----
```

```
-----
Plan name: SQL_PLAN_fkvdraxkukfgbafa6595a      Plan id: 2946914650
Enabled: YES      Fixed: NO      Accepted: YES      Origin: AUTO-CAPTURE
Plan rows: From dictionary
-----
```

PLAN_TABLE_OUTPUT

Plan hash value: 3375727661

```
-----
| Id | Operation          | Name |
-----
| 0 | SELECT STATEMENT   |      |
| 1 | TABLE ACCESS FULL| TABLA1 |
-----
```

PLAN_TABLE_OUTPUT

```
-----
Plan name: SQL_PLAN_fkvdraxkukfgbb2f25575      Plan id: 3002226037
Enabled: YES      Fixed: NO      Accepted: NO      Origin: AUTO-CAPTURE
Plan rows: From dictionary
-----
```

Plan hash value: 4213111612

```
-----
| Id | Operation          | Name |
-----
| 0 | SELECT STATEMENT   |      |
-----
```

PLAN_TABLE_OUTPUT

```
-----
| 1 | TABLE ACCESS BY INDEX ROWID BATCHED| TABLA1 |
| 2 | INDEX RANGE SCAN   | I_TAB1 |
-----
```

11. Desde la sesión del administrador hacemos evolve de este plan:

- Revisamos que hay un plan no aceptado

```
SQL> select sql_handle, plan_name, accepted, enabled, fixed from dba_sql_plan_baselines
where sql_text like '%SISPM%';
```

```
SQL_HANDLE      PLAN_NAME      ACC ENA FIX
-----
SQL_e96db75765a939eb SQL_PLAN_fkvdraxkukfgbafa6595a YES YES NO
SQL_e96db75765a939eb SQL_PLAN_fkvdraxkukfgbb2f25575 NO YES NO
```

- Creamos una tarea evolve SPM task:

```
SQL> var task_name varchar2(50)
SQL> exec :task_name := DBMS_SPM.CREATE_EVOLVE_TASK(sql_handle=>'SQL_e96db75765a939eb',
plan_name=> NULL);
SQL> print task_name
```

TASK_NAME

TASK_351

- Ejecutamos la tarea del SPM:

```
SQL> var exec_name varchar2(50)
SQL> exec :exec_name := DBMS_SPM.EXECUTE_EVOLVE_TASK(task_name => :task_name);
SQL> print exec_name
```

```
EXEC_NAME
-----
EXEC_381
```

- Generamos el reporte de la tarea

```
SQL> set long 100000000
SQL> var evol_out clob
SQL> col evol_out format a120
SQL> exec :evol_out := DBMS_SPM.REPORT_EVOLVE_TASK(task_name=>:task_name, type=>'TEXT',
execution_name=>:exec_name);
SQL> print evol_out
```

```
EVOL_OUT
-----
```

GENERAL INFORMATION SECTION

Task Information:

```
-----
Task Name           : TASK_351
Task Owner          : USER1
Execution Name       : EXEC_381
Execution Type       : SPM_EVOLVE
Scope               : COMPREHENSIVE
Status              : COMPLETED
```

```
EVOL_OUT
-----
```

```
-----
Started             : 10/20/2017 07:02:44
Finished            : 10/20/2017 07:02:46
Last Updated        : 10/20/2017 07:02:46
Global Time Limit   : 2147483646
Per-Plan Time Limit : UNUSED
Number of Errors     : 0
```

SUMMARY SECTION

```
-----
Number of plans processed : 1
```

```
EVOL_OUT
-----
```

```
-----
Number of findings       : 1
Number of recommendations : 1
Number of errors         : 0
```

DETAILS SECTION

```
-----
Object ID              : 2
Test Plan Name         : SQL_PLAN_fkvdraxxkukfgbb2f25575
Base Plan Name         : SQL_PLAN_fkvdraxxkukfgbafa6595a
SQL Handle             : SQL_e96db75765a939eb
```

```
EVOL_OUT
```

```
-----
Parsing Schema      : USER1
Test Plan Creator   : USER1
SQL Text            : SELECT /* SISPM */ * from tabla1 where col1=1
-----
```

Execution Statistics:

```
-----
                        Base Plan                        Test Plan
-----
Elapsed Time (s):      .000346                        .000001
CPU Time (s):          .000144                        0
Buffer Gets:           50                             0
-----
```

EVOL_OUT

```
-----
Optimizer Cost:        137                            1
Disk Reads:            0                              0
Direct Writes:         0                              0
Rows Processed:        0                              0
Executions:            10                             10
-----
```

FINDINGS SECTION

Findings (1):

EVOL_OUT

1. The plan was verified in 0.18900 seconds. It passed the benefit criterion because its verified performance was 250.07220 times better than that of the baseline plan.

Recommendation:

```
-----
Consider accepting the plan. Execute
dbms_spm.accept_sql_plan_baseline(task_name => 'TASK_351', object_id => 2, task_owner
=> 'USER1');
```

EVOL_OUT

EXPLAIN PLANS SECTION

Baseline Plan

```
-----
Plan Id              : 1
Plan Hash Value       : 2946914650
-----
```

```
-----
| Id | Operation                      | Name    | Rows | Bytes | Cost | Time    |
-----
```

EVOL_OUT

```
-----
| 0 | SELECT STATEMENT                |         | 1    | 79    | 137  | 00:00:01 |
| * 1 | TABLE ACCESS FULL             | TABLA1  | 1    | 79    | 137  | 00:00:01 |
-----
```

Predicate Information (identified by operation id):

```
-----
* 1 - filter("COL1"=1)
-----
```

Notes

```

EVOL_OUT
-----
-----
- Dynamic sampling used for this statement ( level = 2 )

Test Plan
-----
Plan Id       : 2
Plan Hash Value : 3002226037

-----
-
| Id | Operation                                | Name      | Rows | Bytes | Cost | Time |
|-----|-----|-----|-----|-----|-----|-----|
EVOL_OUT
-----
-----
-
| 0 | SELECT STATEMENT                                |           | 1 | 79 | 1 | 00:00:01 |
| 1 | TABLE ACCESS BY INDEX ROWID BATCHED | TABLA1 | 1 | 79 | 1 | 00:00:01 |
| * 2 | INDEX RANGE SCAN                            | I_TAB1 | 1 |      | 1 | 00:00:01 |
|-----|-----|-----|-----|-----|-----|
-

Predicate Information (identified by operation id):
-----
* 2 - access("COL1"=1)

```

- Vemos que la ejecución del SPM EVOLVE ADVISOR no acepta ningún sql plan baseline:

```
SQL> select sql_handle, plan_name, accepted, enabled, fixed from dba_sql_plan_baselines
where sql_text like '%SISPM%';
```

SQL_HANDLE	PLAN_NAME	ACC	ENA	FIX
SQL_e96db75765a939eb	SQL_PLAN_fkvdraxxkukfgbafa6595a	YES	YES	NO
SQL_e96db75765a939eb	SQL_PLAN_fkvdraxxkukfgbb2f25575	NO	YES	NO

- Aceptamos el plan baseline recomendado por el SPM EVOLVE ADVISOR y revisamos que lo acepta:

```
SQL> exec dbms_spm.accept_sql_plan_baseline(task_name => 'TASK_351', object_id => 2,
task_owner => 'USER1');
```

```
SQL> select sql_handle, plan_name, accepted, enabled, fixed from dba_sql_plan_baselines
where sql_text like '%SISPM%';
```

SQL_HANDLE	PLAN_NAME	ACC	ENA	FIX
SQL_e96db75765a939eb	SQL_PLAN_fkvdraxxkukfgbafa6595a	YES	YES	NO
SQL_e96db75765a939eb	SQL_PLAN_fkvdraxxkukfgbb2f25575	YES	YES	NO

NOTA: Aunque no lo recomendara el SQL Evolve Advisor podríamos aceptar el plan utilizando la opción force:

```
SQL> exec dbms_spm.accept_sql_plan_baseline(task_name => 'TASK_351', object_id =>
2, task_owner => 'USER1', force => TRUE);
```

12. Desde la sesión del usuario vemos que coge el nuevo plan a través del sql plan baseline

```
SQL> conn user1/user1
SQL> set autotrace traceonly explain
SQL> SELECT /* SISPM */ * from tabla1 where coll=1;

Execution Plan
-----
Plan hash value: 4213111612

-----
| Id | Operation                                | Name    | Rows  | Bytes | Cost (%CPU) | Time |
|----|-----|-----|-----|-----|-----|-----|
| 0  | SELECT STATEMENT                        |         |      1 |    79 |      1 (0)  |      |
| 1  | TABLE ACCESS BY INDEX ROWID BATCHED | TABLA1  |      1 |    79 |      1 (0)  |      |
|* 2  | INDEX RANGE SCAN                       | I_TAB1  |      1 |      |      1 (0)  |      |
-----

Predicate Information (identified by operation id):
-----

   2 - access("COL1"=1)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - SQL plan baseline "SQL_PLAN_fkvdraxxkukfgbb2f25575" used for this statement
```

Se ejecuta 9 veces más la sentencia en la misma sesión

```
SQL> r

   1* select * from tabla1 where coll=1

...

SQL> r

   1* select * from tabla1 where coll=1
```

13. Desde la sesión del administrador vemos el estado del plan baseline y si se está usando:

```
SQL> select sql_handle,plan_name,enabled,accepted from dba_sql_plan_baselines;
SQL_HANDLE          PLAN_NAME          ENA ACC
-----
SQL_356584134d867c7f SQL_PLAN_3atc42d6scz3zafa6595a YES YES
SQL_356584134d867c7f SQL_PLAN_3atc42d6scz3zccea6809 YES YES
SQL> select SQL_PLAN_BASELINE, sql_id, plan_hash_value,CHILD_NUMBER,EXECUTIONS from
V$SQL where SQL_ID='24puppz0qh3mw'
```

SQL_PLAN_BASELINE	SQL_ID	PLAN_HASH_VALUE	CHILD_NUMBER	EXECUTIONS
SQL_PLAN_fkvdraxxkukfgbb2f25575	24puppz0qh3mw	4213111612	0	10

Siguiendo con el mismo ejemplo, a continuación se muestra cómo el SQL plan baseline no depende del esquema si no de la sentencia SQL, esto es, si un mismo SQL es ejecutado sobre objetos de distintos esquemas, el SQL plan baseline se aplicará en ambos casos.

1. Desde otro esquema creamos un objeto con el mismo nombre, vemos que utiliza el sql baseline creado previamente:

```
SQL> conn user2/user2
Connected.
SQL> create table tabla1 (col1 number, col2 varchar2(81), col3 number);
SQL> insert into tabla1 SELECT rownum, segment_NAME, rownum*2 FROM DBA_segments;
6214 rows created.
SQL> commit;
Commit complete.

SQL> SELECT /* SISPM */ * from tabla1 where col1=1;

      COL1 COL2                                COL3
-----
      1 ACCESS$                                2

SQL> select * from table(dbms_xplan.display_cursor);

PLAN_TABLE_OUTPUT
-----
SQL_ID  24puppz0qh3mw, child number 1
-----
SELECT /* SISPM */ * from tabla1 where col1=1

Plan hash value: 3375727661

-----
| Id  | Operation                      | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0  | SELECT STATEMENT                |         |       |       |    9 (100)|          |
|*  1  | TABLE ACCESS FULL              | TABLA1  |     1 |    68 |     9   (0)| 00:00:01 |
-----

PLAN_TABLE_OUTPUT
-----

Predicate Information (identified by operation id):
-----

   1 - filter("COL1">=1)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - SQL plan baseline SQL_PLAN_fkvdra used for this statement
```

2. Desde la sesión de SYSDBA deshabilitamos el sql plan baseline:

```
SQL> DECLARE
        report  varchar2(1000);
    BEGIN
        report := dbms_spm.ALTER_SQL_PLAN_BASELINE
        ('SQL_e96db75765a939eb',
        'SQL_PLAN_fkvdra',
        ATTRIBUTE_NAME => 'ENABLED',
        ATTRIBUTE_VALUE => 'NO');
    END;
    /

PL/SQL procedure successfully completed.
```

- Desde la sesión de usuario vemos que ya no coge ningún plan del sql plan baseline, ya que el plan aceptado que queda en el SQL plan baseline no puede reproducirse ya que es inválido para este esquema (no tiene el índice):

```
SQL> conn user2/user2
Connected.
SQL> SELECT /* SISPM */ * from tabla1 where coll=1;
SQL> SELECT /* SISPM */ * from tabla1 where coll=1;
SQL> SELECT /* SISPM */ * from tabla1 where coll=1;
SQL> SELECT /* SISPM */ * from tabla1 where coll=1;
```

Desde la sesión SYSDBA, vemos que el SQL se está ejecutando en un child que no utiliza ningún SQL_PLAN_BASELINE.

```
SQL> column sql_text format a50
SQL> column SQL_PLAN_BASELINE format a30
SQL> select SQL_PLAN_BASELINE, sql_id, plan_hash_value,CHILD_NUMBER,EXECUTIONS,
hash_value, PARSING_SCHEMA_NAME from v$sql where sql_text like '%SISPM%' AND SQL_TEXT
NOT LIKE '%SQL%' and sql_text not like '%PLAN%';
```

SQL_PLAN_BASELINE HASH_VALUE PARSING_SC	SQL_ID	PLAN_HASH_VALUE	CHILD_NUMBER	EXECUTIONS
-----	-----	-----	-----	-----
3244822140 USER2	24puppz0qh3mw	3375727661	0	10
SQL_PLAN_fkvdaxkukfgbb2f25575 3244822140 USER1	24puppz0qh3mw	4213111612	1	6

- Desde la sesión de SYSDBA borramos el sql plan baseline generado en el ejemplo:

```
declare
xx PLS_INTEGER;
BEGIN
xx
:=dbms_spm.drop_sql_plan_baseline(sql_handle=>'SQL_e96db75765a939eb',plan_name=>null);
END;
/

SQL> select sql_handle,plan_name,enabled,accepted from dba_sql_plan_baselines;

no rows selected
```

Ejemplo 2: Creación SQL plan baseline manualmente desde la shared Cursor Area

- Revisar en v\$sql el sql_id y el plan_hash_value que queremos cargar a un SQL plan baseline, por ejemplo:

```
SELECT SQL_ID, CHILD_NUMBER AS "Child Num", PLAN_HASH_VALUE AS "Plan Hash",
OPTIMIZER_ENV_HASH_VALUE AS "Opt Env Hash", sql_fulltext
FROM V$SQL
WHERE sql_text LIKE '%from prueba%' and sql_text not like '%SELECT%SQL_ID%';
```

SQL_ID	Child Num	Plan Hash	Opt Env Hash	SQL_FULLTEXT
g7ug9u5vj7xqy	0	3279922394	1095763852	select * from prueba

NOTA: Si queremos ver el plan del cursor:

```
select *
from table(dbms_xplan.display_cursor('g7ug9u5vj7xqy'));

-- Un child concreto:
select *
from table(dbms_xplan.display_cursor('g7ug9u5vj7xqy', 0));
```

2. Se carga el plan a un SQL plan baseline:

```
VARIABLE v_plan_cnt NUMBER
EXECUTE :v_plan_cnt := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(sql_id => 'g7ug9u5vj7xqy');
```

3. Verificamos que el nuevo baseline es añadido como aceptado:

```
SELECT SQL_HANDLE, SQL_TEXT, PLAN_NAME,
ORIGIN, ENABLED, ACCEPTED
FROM DBA_SQL_PLAN_BASELINES;
```

SQL_HANDLE	SQL_TEXT	ORIGIN	ENA	ACC
SQL_9c2ac5223379eb37	select * from prueba	MANUAL-LOAD- FROM-CURSOR- CACHE	YES	YES

Ejemplo 3: Cambio del plan de ejecución con Plan Baseline añadiendo un hint en la sentencia.

Se muestran 2 opciones, usando un script y otra manualmente:

- Opción 1: Con script coe_load_sql_baseline.sql

El siguiente ejemplo utiliza el script coe_load_sql_baseline.sql que se puede descargar de la nota How to use hints to customize SQL Profile or SQL PLAN Baseline (Doc ID 1400903.1). También se podría hacer sin el script, la nota siguiente indica cómo: Loading Hinted Execution Plans into SQL Plan Baseline. (Doc ID 787692.1)

1. Se ejecuta el SQL original

```
SQL> select * from tabla1 where coll=1;
```

2. Se ejecuta el SQL con el hint

```
select /*+ FULL (tabla1) */ * from tabla1 where coll=1;
```

3. Se busca el sql_id y plan_hash_value de ambos SQLs

```
conn system/oracle
column sql_text format a60
select sql_id ,plan_hash_value, sql_text from v$sql where sql_text like '%tabla1%';
```

SQL_ID	PLAN_HASH_VALUE	SQL_TEXT
a2mtn0ys9p3ct	3375727661	select /*+ FULL (tabla1) */ * from tabla1 where coll=1
43fzgxctt3zgq	4213111612	select * from tabla1 where coll=1

4. De esa sentencia registramos estos valores:

Original sql id: 43fzgxctt3zgq

Modified sql id: a2mtn0ys9p3ct

Modified Plan Hash Value: 3375727661

5. Ejecutamos el coe_load_sql_baseline.sql y le pasamos esos valores:

```
SQL> @coe_load_sql_baseline.sql
Parameter 1:
ORIGINAL_SQL_ID (required)

Enter value for 1: 43fzgxctt3zgq

Parameter 2:
MODIFIED_SQL_ID (required)

Enter value for 2: a2mtn0ys9p3ct
```

PLAN_HASH_VALUE	AVG_ET_SECS
3375727661	.006

```
Parameter 3:
PLAN_HASH_VALUE (required)

Enter value for 3: 3375727661
```

```
Values passed to coe_load_sql_baseline:
~~~~~
ORIGINAL_SQL_ID: "43fzgxctt3zgq"
MODIFIED_SQL_ID: "a2mtn0ys9p3ct"
PLAN_HASH_VALUE: "3375727661"
```

6. Ejecutamos la sentencia original:

```
conn user1/user1
```

```

set autotrace traceonly
SQL> select * from tabla1 where col1=1;
no rows selected

Execution Plan
-----
Plan hash value: 3375727661

-----
| Id | Operation          | Name | Rows | Bytes | Cost (%CPU) | Time |
-----
| 0 | SELECT STATEMENT   |      |    1 |    79 |    137 (1) | 00:00:01 |
|* 1 | TABLE ACCESS FULL| TABLA1 |    1 |    79 |    137 (1) | 00:00:01 |
-----

Predicate Information (identified by operation id):
-----

   1 - filter("COL1"=1)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - SQL plan baseline "SQL_PLAN_3atc42d6scz3zafa6595a" used for this statement

```

7. Lo borramos:

```

declare
xx PLS_INTEGER;
BEGIN
xx
:=dbms_spm.drop_sql_plan_baseline(sql_handle=>'SQL_356584134d867c7f',plan_name=>null);
END;
/

SQL> select sql_handle,plan_name,enabled,accepted from dba_sql_plan_baselines;

no rows selected

```

- Opción2:

El siguiente ejemplo se hace sin el script, similar a la nota siguiente: Loading Hinted Execution Plans into SQL Plan Baseline. (Doc ID 787692.1)

1. Cargamos los datos del ejemplo

```

conn user1/user1
drop table tabla1;
create table tabla1 (col1 number, col2 number);
insert into tabla1 SELECT OBJECT_ID, OBJECT_ID FROM DBA_OBJECTS;
commit;
create index i_tab1 on tabla1(col1);
create index i_tab2 on tabla1(col2);
exec dbms_stats.gather_table_stats('USER1','TABLA1');

```

2. Se ejecuta el SQL original

```

select * from tabla1 where col1=1 and col2=2;

SET AUTOTRACE ON
select * from tabla1 where col1=1 and col2=2;

```

Execution Plan

Plan hash value: 2400378834

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	2 (0)	00:00:01
* 1	TABLE ACCESS BY INDEX ROWID	TABLA1	1	10	2 (0)	00:00:01
2	INDEX RANGE SCAN	I_TAB1	1		1 (0)	00:00:01

3. Se ejecuta el SQL con el hint

```
select /*+ FULL (tabla1) */ * from tabla1 where col1=1 and col2=2;
```

SET AUTOTRACE ON

```
select /*+ FULL (tabla1) */ * from tabla1 where col1=1 and col2=2;
```

Execution Plan

Plan hash value: 3375727661

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	69 (2)	00:00:01
* 1	TABLE ACCESS FULL	TABLA1	1	10	69 (2)	00:00:01

4. Se busca el sql_id y plan_hash_value de ambos SQLs

conn system/oracle

```
SELECT sql_id, sql_fulltext, plan_hash_value
FROM V$SQL
```

```
WHERE sql_text LIKE '%from tabla1 where col1=1 and col2=2%' and sql_text not like
'SELECT sql%' and sql_text not like 'EXPLAIN%';
```

SQL_ID	SQL_FULLTEXT	PLAN_HASH_VALUE
--------	--------------	-----------------

```
ap2hbbaqnuab select * from tabla1 where col1=1 and col2=2
2400378834
```

```
bd2uujk6yjqy6 select /*+ FULL (tabla1) */ * from tabla1 where col1=1 and col2=2
3375727661
```

5. Creamos el SQL plan baseline para la sentencia

variable cnt number;

```
EXECUTE :cnt :=DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(sql_id=>'ap2hbbaqnuab');
```

Para ver el plan:

```
column SQL_HANDLE format a20
column SQL format a31
set linesize 170
```

```
select sql_handle, substr(sql_text,1,30) "SQL", plan_name, origin,
enabled, accepted, fixed, autopurge
from dba_sql_plan_baselines where sql_text LIKE '%from tabla1 where col1=1 and col2=2%';
```

SQL_HANDLE	SQL	ORIGIN	ENA	ACC	FIX	AUT	PLAN_NAME
------------	-----	--------	-----	-----	-----	-----	-----------

```
SQL_ed451a477df2af66 select * from tabla1 where col SQL_PLAN_fuj8u8xyz5bv6ccea6809
MANUAL-LOAD YES YES NO YES
```

1 row selected.

6. Asociamos el nuevo plan de ejecución al sql original usando el sql_handle:

```
var res number
exec :res := dbms_spm.load_plans_from_cursor_cache( -
sql_id => '&hinted_SQL_ID', -
plan_hash_value => &hinted_plan_hash_value, -
sql_handle => '&sql_handle_for_original');
```

En nuestro caso:

```
var res number
exec :res := dbms_spm.load_plans_from_cursor_cache( -
sql_id => 'bd2uujk6yjqy6', -
plan_hash_value => 3375727661, -
sql_handle => 'SQL_ed451a477df2af66');
```

7. Verificamos que el nuevo baseline es añadido como aceptado:

```
column SQL_HANDLE format a20
column SQL format a31
set linesize 170
select sql_handle, substr(sql_text,1,30) "SQL", plan_name, origin,
enabled, accepted, fixed, autopurge, OPTIMIZER_COST
from dba_sql_plan_baselines where sql_text LIKE '%from tabla1 where col1=1 and col2=2%';
```

SQL_HANDLE	SQL	PLAN_NAME	ORIGIN
ENA ACC FIX AUT OPTIMIZER_COST			
SQL_ed451a477df2af66	select * from tabla1 where col	SQL_PLAN_fuj8u8xyz5bv635b05050	
MANUAL-LOAD YES YES NO YES	2		
SQL_ed451a477df2af66	select * from tabla1 where col	SQL_PLAN_fuj8u8xyz5bv6afa6595a	
MANUAL-LOAD YES YES NO YES	69		

8. Para ver el plan de cada uno:

```
select *
from table(dbms_xplan.display_sql_plan_baseline(
sql_handle => 'SQL_ed451a477df2af66', format => 'basic'));
```

PLAN_TABLE_OUTPUT

```
SQL handle: SQL_ed451a477df2af66
SQL text: select * from tabla1 where col1=1 and col2=2
```

```
Plan name: SQL_PLAN_fuj8u8xyz5bv6afa6595a Plan id: 2946914650
Enabled: YES Fixed: NO Accepted: YES Origin: MANUAL-LOAD
```

Plan hash value: 3375727661

Id	Operation	Name
0	SELECT STATEMENT	
1	TABLE ACCESS FULL	TABLA1

```
Plan name: SQL_PLAN_fuj8u8xyz5bv6ccea6809 Plan id: 3437914121
Enabled: YES Fixed: NO Accepted: YES Origin: MANUAL-LOAD
```

Plan hash value: 2400378834

```
-----
| Id | Operation | Name |
-----
| 0 | SELECT STATEMENT | |
| 1 | TABLE ACCESS BY INDEX ROWID | TABLA1 |
| 2 | INDEX RANGE SCAN | I_TAB1 |
-----
```

9. El plan capturado inicialmente se borra:

```
var res number;

exec                                :res                                :=DBMS_SPM.DROP_SQL_PLAN_BASELINE
('SQL_ed451a477df2af66','SQL_PLAN_fuj8u8xyz5bv6ccea6809');
```

10. Se ejecuta la sentencia y se comprueba que está cogiendo el plan modificado a través del SQL plan baseline:

```
select SQL_PLAN_BASELINE from V$SQL where SQL_ID='ap2hbbafqnuab'
```

Ejemplo 4: Copia de SQL Plan Baseline entre bases de datos

- Simulo una sentencia SQL cuyo plan de ejecución se quiere mover entre bbdd mediante SQL Plan Baseline, por ejemplo entre las bbdd desa y prod:

Sentencia SQL:

```
select * from tabla1 where coll=1;
```

1. Creo el SQL plan baseline en desa

- Busco el sql_id y plan_hash_value de la sentencia

```
conn / as sysdba
SELECT sql_id, sql_fulltext, plan_hash_value,CHILD_NUMBER
FROM V$SQL
WHERE sql_text LIKE '%tabla1 where coll=%' and sql_text not like 'SELECT sql_id%' and
sql_text not like '%EXPLAIN%';
```

SQL_ID

SQL_FULLTEXT

PLAN_HASH_VALUE CHILD_NUMBER

43fzgxctt3zgq

```
select * from tabla1 where coll=1
```

3375727661

0

- Compruebo el plan

```
select *
```

```
from table(dbms_xplan.display_cursor('43fzgxctt3zgq', 0));
```

```
Plan hash value: 3375727661
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				23 (100)	
* 1	TABLE ACCESS FULL	TABLA1	1	24	23 (0)	00:00:00

- Creo un baseline con ese plan:

```
var res number ;
```

```
exec :res := dbms_spm.load_plans_from_cursor_cache(sql_id => '43fzgxctt3zgq',  
plan_hash_value => '3375727661' );
```

- Verificamos que el nuevo baseline es añadido:

```
select substr(sql_text,1,30) "SQL", sql_handle, plan_name, enabled, accepted  
from dba_sql_plan_baselines;
```

```
SQL
```

SQL_HANDLE	PLAN_NAME	ENA	ACC
select * from tabla1 where col	SQL_PLAN_3atc42d6scz3zafa6595a	YES	YES

2. En desa creamos una tabla staging

Como sys:

```
drop table system.MIS_PERF_SPM_PLANS;  
exec DBMS_SPM.CREATE_STGTAB_BASELINE ('MIS_PERF_SPM_PLANS','SYSTEM');
```

3. Se incluye el baseline deseado en la tabla:

```
DECLARE  
    i PLS_INTEGER;  
BEGIN  
    i  
    dbms_spm.pack_stgtab_baseline('MIS_PERF_SPM_PLANS','SYSTEM','SQL_356584134d867  
c7f','SQL_PLAN_3atc42d6scz3zafa6595a');  
    dbms_output.put_line(i);  
END;  
/
```

4. Se hace un export/import desde desa a prod de la tabla MIS_PERF_SPM_PLANS

Por ejemplo

En desa:

```
SQL> create directory ttsdir as '/tmp';
```

```
$ expdp system/oracle@admin1 file=pru.dmp log=pru.log
tables=system.MIS_PERF_SPM_PLANS directory=ttsdir
```

En prod:

```
SQL> drop table system.MIS_PERF_SPM_PLANS;
```

```
$ export ORACLE_SID=asdb
```

```
$ time impdp \'/ as sysdba\ DIRECTORY=ttsdir LOGFILE=dp_userimp.log
file=pru.dmp FULL=y
```

5. En prod se extrae el sql plan baseline desde la tabla

```
DECLARE
    i PLS_INTEGER;
BEGIN
    i
    dbms_spm.unpack_stgtab_baseline('MIS_PERF_SPM_PLANS','SYSTEM','SQL_356584134d8
67c7f','SQL_PLAN_3atc42d6scz3zafa6595a');
    dbms_output.put_line(i);
END;
/
```

6. Se ejecuta la sentencia SQL y se comprueba que está cogiendo el plan a través del baseline:

```
set autotrace traceonly
select * from tabla1 where col1=1;
Plan hash value: 3375727661
```

...
Note

```
- SQL plan baseline "SQL_PLAN_3atc42d6scz3zafa6595a" used for this
statement
```

Desde SYS:

```
column SQL_HANDLE format a20
```

```
column SQL format a31
```

```
set linesize 170
```

```
select substr(sql_text,1,30) "SQL", sql_handle, plan_name, enabled, accepted,
PARSING_SCHEMA_NAME from dba_sql_plan_baselines;
```

SQL	SQL_HANDLE	PLAN_NAME
ENA ACC PARSING_SCHEMA_NAME		
-----	-----	-----
select * from tabla1 where col	SQL_356584134d867c7f	
SQL_PLAN_3atc42d6scz3zafa6595a YES YES USER1		
select * from tabla1 where col	SQL_356584134d867c7f	
SQL_PLAN_3atc42d6scz3zccea6809 YES NO USER1		

```
select SQL_PLAN_BASELINE, sql_id, plan_hash_value,CHILD_NUMBER,EXECUTIONS from
V$SQL where SQL_ID='43fzgxctt3zgq';
```

Ejemplo 5: Creación de SQL Plan Baseline de un plan guardado en AWR.

- Suponemos que queremos cambiar el plan de un SQL por el plan que tenía en un intervalo de guardado en AWR.

1. Suponemos la ejecución de una sentencia

```
select /*LOAD AWR*/ count(*) from tabla1,tabla1;
```

2. Revisamos su sql_id:

```
select SQL_PLAN_BASELINE, sql_id, plan_hash_value,CHILD_NUMBER,EXECUTIONS,
hash_value, sql_text, PARSING_SCHEMA_NAME from v$sql where sql_text like
'%tabla1,tabla1%' AND SQL TEXT NOT LIKE '%SQL%' and sqltext not like '%PLAN%'
```

```

SQL_PLAN_BASELINE          SQL_ID          PLAN_HASH_VALUE CHILD_NUMBER
-----
EXECUTIONS HASH_VALUE SQL_TEXT
-----
PARSING_SCHEMA_NAME
-----
--
                dnxm3p12xjg6w          2339188913          0
2 1171832028 select /*LOAD_AWR*/ count(*) from tabla1,tabla1
USER1

```

3. Forzamos la ejecución de un AWR snapshot y revisamos el snap_id de los últimos snapshot:

```
SQL> EXEC DBMS WORKLOAD REPOSITORY.CREATE SNAPSHOT;
```

```
SQL> SELECT *
FROM   (SELECT SNAP_ID, SNAP_LEVEL,
              TO_CHAR(BEGIN_INTERVAL_TIME, 'DD/MM/YY HH24:MI:SS') BEGIN
        FROM   DBA_HIST_SNAPSHOT
        ORDER BY SNAP_ID DESC)
WHERE   ROWNUM <= 3;
```

SNAP_ID	SNAP_LEVEL	BEGIN
57	1	06/11/17 08:00:49
56	1	06/11/17 07:00:42
55	1	06/11/17 06:00:36

4. Reviso el código y el plan de ejecución en AWR:

```
SQL> select * from TABLE(dbms_xplan.display awr('dnxm3p12xjg6w'));
```

SQL_ID dnxm3p12xjg6w

```
select /*LOAD_AWR*/ count(*) from tabla1,tabla1
```

Plan hash value: 2339188913

Id	Operation	Name	Rows	Cost (%CPU)	Time
0	SELECT STATEMENT			4840K (100)	
1	SORT AGGREGATE		1		

PLAN_TABLE_OUTPUT

2	MERGE JOIN CARTESIAN		5311M	4840K (1)	00:03:10
3	TABLE ACCESS FULL	TABLA1	72882	68 (0)	00:00:01
4	BUFFER SORT		72882	4840K (1)	00:03:10
5	TABLE ACCESS FULL	TABLA1	72882	66 (0)	00:00:01

NOTA:

Podemos ver las estadísticas de la query y la evolución de cada plan en AWR:

```
set linesize 200
select ss.snap_id, ss.instance_number node, begin_interval_time, sql_id,
plan_hash_value,
nvl(executions_delta,0) execs,
(elapsed_time_delta/decode(nvl(executions_delta,0),0,1,executions_delta))/1000000
avg_etime,
(buffer_gets_delta/decode(nvl(buffer_gets_delta,0),0,1,executions_delta)) avg_lio
from DBA_HIST_SQLSTAT S, DBA_HIST_SNAPSHOT SS
where sql_id = nvl('dnxm3p12xjg6w','0')
and ss.snap_id = S.snap_id
and ss.instance_number = S.instance_number
and executions_delta > 0
order by 1, 2, 3;
```

SNAP_ID	NODE	BEGIN_INTERVAL_TIME	SQL_ID	PLAN_HASH_VALUE
EXECES	AVG_ETIME	AVG_LIO		
57	1	06-NOV-17 08.00.49.635 AM	dnxm3p12xjg6w	2339188913
2 10.0250155	451.5			

5. Cargamos el SQL Plan baseline desde AWR:

```
VARIABLE v_plan_cnt NUMBER
```

```
EXEC :v_plan_cnt := DBMS_SPM.LOAD_PLANS_FROM_AWR(begin_snap => 56,end_snap =>
57,basic_filter => 'lower(sql_text) like ''%load_awr%tabla1%''',enabled =>
'NO',fixed=>'no');
```

NOTA: En este caso le indicamos no enabled, por defecto está habilitado.

También se podría haber cargado directamente con el sql_id:

```
VARIABLE v_plan_cnt NUMBER
```

```
EXEC :v_plan_cnt := DBMS_SPM.LOAD_PLANS_FROM_AWR(begin_snap =>
56,end_snap => 57,basic_filter => 'sql_id = 'dnxm3p12xjg6w'');
```

6. Revisamos el plan baseline

```
COL SQL_HANDLE FORMAT a20
COL SQL_TEXT FORMAT a20
COL PLAN_NAME FORMAT a30
COL ORIGIN FORMAT a20
```

```
SELECT SQL_HANDLE, SQL_TEXT, PLAN_NAME,
       ORIGIN, ENABLED, ACCEPTED
FROM   DBA_SQL_PLAN_BASELINES
WHERE  SQL_TEXT LIKE '%LOAD_AWR%';
```

```
SQL> SQL> SQL> SQL> SQL> SQL>      2      3      4
SQL_HANDLE                                SQL_TEXT                                PLAN_NAME
ORIGIN                                ENA ACC                                -----
-----
SQL_dcbb8167f356c23c  select      /*LOAD_AWR*/      SQL_PLAN_dtfw1cztpdhjweadcba38
MANUAL-LOAD-FROM-AWR NO  YES
                        count(*) from tabla1
                        ,tabla1
```

7. Habilitamos el sql_plan ya que se puso no enabled al cargar:

```
DECLARE
    report varchar2(1000);
BEGIN
    report := dbms_spm.ALTER_SQL_PLAN_BASELINE
        ('SQL_dcbb8167f356c23c',
        'SQL_PLAN_dtfw1cztpdhjweadcba38',
        ATTRIBUTE_NAME => 'ENABLED',
        ATTRIBUTE_VALUE => 'YES');
END;
/
```

8. Ejecutamos varias veces la sentencia y revisamos desde otra sesión que está cogiendo el SQL plan baseline:

```
select /*LOAD_AWR*/ count(*) from tabla1,tabla1;
```

```
select SQL_PLAN_BASELINE, sql_id, plan_hash_value,CHILD_NUMBER,EXECUTIONS,
sql_text from V$SQL where sql_text like '%tabla1,tabla1%' AND SQL_TEXT NOT
LIKE '%SQL%' and sql_text not like '%PLAN%'
```

```
SQL_PLAN_BASELINE                                SQL_ID                                PLAN_HASH_VALUE CHILD_NUMBER
EXECUTIONS SQL_TEXT                                -----
-----
                        dnxm3p12xjg6w                                2339188913                                0
2 select /*LOAD_AWR*/ count(*) from tabla1,tabla1
SQL_PLAN_dtfw1cztpdhjweadcba38 dnxm3p12xjg6w                                2339188913                                1
6 select /*LOAD_AWR*/ count(*) from tabla1,tabla1
```

Referencias/Documentación recomendada relativa a SPM

- Oracle® Database SQL Tuning Guide 12c Release 2 (12.2) 29 Managing SQL Plan Baselines
<https://docs.oracle.com/database/122/TGSQL/managing-sql-plan-baselines.htm#TGSQL94621>
- FAQ: SQL Plan Management (SPM) Frequently Asked Questions (Doc ID 1524658.1)
- Master Note: Plan Stability Features (Including SQL Plan Management (SPM)) (Doc ID 1359841.1)
- White Papers and Blog Entries for Oracle Optimizer (Doc ID 1337116.1)
- <https://blogs.oracle.com/optimizer/sql-plan-management-part-1-of-4-creating-sql-plan-baselines>
 - SQL Plan Management (Part 1 of 4) Creating SQL plan baselines
 - SQL Plan Management (Part 2 of 4) SPM Aware Optimizer
 - SQL Plan Management (Part 3 of 4): Evolving SQL Plan Baselines
 - SQL Plan Management (Part 4 of 4): User Interfaces and Other Features