Memoptimized Rowstore - Oracle Database 18c

Por Francisco Riccio

Introducción

Memoptimized Rowstore es una nueva funcionalidad disponible en Oracle Database 18c y nos permite obtener máximo desempeño en aquellas consultas que filtran por el primary key de la tabla de manera frecuente. A partir de esta nueva funcionalidad, se proyecta una reducción en el tiempo de ejecución de estas consultas SQL en un 50% aproximadamente.

Para ello, Oracle ha creado un nuevo pool de memoria en el SGA llamado **memoptimize** y por defecto está configurado a 0 bytes, teniendo como requisito mínimo 100 MB en caso se desee habilitar esta funcionalidad. Este nuevo pool, almacenará un hash index basado en el primary key de cada tabla y los bloques de la tabla serán alojados en el Buffer Cache.

Las tablas a habilitar con esta nueva opción deberán cumplir con los siguientes requisitos:

- No tener habilitado la opción de compresión.
- Deben tener un constraint de tipo primary key.

Una vez habilitado esta opción en las tablas que deseemos, cualquier consulta basado en el primary key y únicamente utilizando el operador de igualdad, utilizará el feature **Fast Lookup** que permitirá al optimizador de la base de datos primero consultar el hash index para luego ir directamente al bloque de datos que se encuentra en el Buffer Cache con la finalidad de evitar el acceso a disco.

Implementación

Se creará una tabla llamada Producto con su primary key para luego insertar 1 millón de registros. Posterior a ello se realizará la implementación y pruebas de la nueva funcionalidad.

```
SQL> create table Producto (cod number, nombre varchar(20));

Table created.

SQL> alter table Producto add constraint PK_Producto primary key (cod);

Table altered.

SQL> insert into Producto select rownum, 'Producto'||to_char(rownum)
     from dual connect by level<=1000000;

1000000 rows created.

SQL> commit;

Commit complete.

SQL> execute dbms_stats.gather_table_stats('FRICCIO','PRODUCTO');

PL/SQL procedure successfully completed.
```

a. Habilitación del Feature

Se deberá modificar el parámetro **MEMOPTIMIZE_POOL_SIZE**. Para la implementación se asignará 256 MB de memoria RAM. Recordemos que este pool mantiene en memoria un arreglo hash basado en los valores del campo de la primary key de una tabla específica.

```
[oracle@PRD ~]$ sqlplus / as sysdba

SQL*Plus: Release 18.0.0.0.0 Production on Wed Apr 11 02:29:46 2018
Version 18.1.0.0.0

Copyright (c) 1982, 2017, Oracle.  All rights reserved.


Connected to:
Oracle Database 18c Enterprise Edition Release 18.0.0.0.0 - Production
Version 18.1.0.0.0

SQL> alter system set memoptimize_pool_size=250M scope=spfile;

System altered.

SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.

Total System Global Area 2768239832 bytes
Fixed Size                  8899800 bytes
Variable Size             922746880 bytes
Database Buffers         1761607680 bytes
Redo Buffers               74985472 bytes
Database mounted.
Database opened.
SQL> show parameter memoptimize_pool_size

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
memoptimize_pool_size                big integer 256M
```

b. Poblar el Hash Index

Para poblar el hash index primero se debe configurar la tabla con la opción MEMOPTIMIZE FOR READ y posteriormente ejecutar el paquete DBMS_MEMOPTIMIZE.

```
SQL> alter table friccio.producto memoptimize for read;

Table altered.

SQL> execute dbms_memoptimize.populate(schema_name=>'FRICCIO',table_name=>'PRODUCTO');

PL/SQL procedure successfully completed.
```

En caso se desee trabajar únicamente con la partición de una tabla, se deberá ingresar el nombre de este como tercer parámetro en el procedimiento DBMS_MEMOPTIMIZE.POPULATE.

Es posible habilitar la opción MEMOPTIMIZE FOR READ desde la creación de la tabla como a continuación se presenta:

```
SQL> create table Producto (cod number primary key, nombre varchar(20))
     segment creation immediate memoptimize for read;

Table created.
```

Si deseamos deshabilitar el feature en la tabla, ejecutamos el siguiente comando:

```
SQL> alter table producto no memoptimize for read;

Table altered.
```

También se puede eliminar el Hash Index a través del siguiente procedimiento DBMS_MEMOPTIMIZE.DROP_OBJECT.

El paquete DBMS_MEMOPTIMIZE se encuentra en el siguiente script: $ORACLE_HOME/rdbms/admin/dbmsmemoptimize.sql y por efecto viene instalado en la base de datos.

c. Revisión de Planes de Ejecución

Una vez ejecutado los pasos de implementación, se procederá a realizar una consulta sobre la tabla Producto, filtrando por el campo cod (primary key) y utilizando el operador "=".

```
SQL> set linesize 600
SQL> set autotrace traceonly
SQL> select * from producto where cod=10;


Execution Plan
----------------------------------------------------------
Plan hash value: 3117562780

-------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |              |     1 |    20 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID READ OPTIM| PRODUCTO     |     1 |    20 |     3   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN READ OPTIM        | PK_PRODUCTO  |     1 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

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

   2 - access("COD"=10)


Statistics
----------------------------------------------------------
          8  recursive calls
          0  db block gets
         11  consistent gets
          0  physical reads
          0  redo size
        632  bytes sent via SQL*Net to client
        623  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Se aprecia que el plan de ejecución cuenta con 2 nuevas operaciones: **INDEX UNIQUE SCAN READ OPTIM y TABLE ACCESS BY INDEX ROWID READ OPTIM** y se evidencia que primero es consultado el Hash Index y luego la Tabla para acceder al registro solicitado por la consulta SQL.

A continuación se realizará algunas consultas SQL donde se apreciará que el Buffer MemOptimized no será tomado en cuenta al no cumplir las condiciones previamente especificadas:

Caso #1:

```
SQL> select * from producto where cod<10;

9 rows selected.


Execution Plan
----------------------------------------------------------
Plan hash value: 4256070684

-------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |              |     9 |   180 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PRODUCTO     |     9 |   180 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                    | PK_PRODUCTO |     9 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Al utilizar un operador distinto al "=" genera que no se utilice el nuevo pool.

Caso #2:

```
SQL> select * from producto where cod=10 or cod=9;


Execution Plan
----------------------------------------------------------
Plan hash value: 2889139755

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     2 |    40 |     5   (0)| 00:00:01 |
|   1 |  INLIST ITERATOR              |              |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| PRODUCTO     |     2 |    40 |     5   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN          | PK_PRODUCTO |     2 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Este caso a pesar de utilizar el operador "=" no utiliza el nuevo pool al filtrar por más de un valor.

Caso #3:

```
SQL> select * from producto where cod=10 and nombre='Producto10';


Execution Plan
----------------------------------------------------------
Plan hash value: 3117562780

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     1 |    20 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| PRODUCTO     |     1 |    20 |     3   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN           | PK_PRODUCTO |     1 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

El nuevo pool no será utilizado debido a que el filtro se lleva por un campo adicional al primary key.

Conclusión

Esta nueva funcionalidad permite obtener alto desempeño en el tiempo de respuesta de las aplicaciones que buscan consultar datos filtrados por el primary key de una tabla mediante un operador de igualdad; donde este tipo de comportamiento es usual en soluciones de IoT (Internet de las Cosas).

La propuesta entregada de MemOptimize es muy beneficiosa pero a la vez se debe tener presente las reglas que deben cumplirse para que el optimizador de la base de datos pueda tomar ventaja de ello y por ende nuestras aplicaciones.

Publicado por:

Francisco Riccio, actualmente se desempeña como Arquitecto de Soluciones en Oracle Perú y es instructor de cursos oficiales de certificación Oracle. Es un Oracle Certified Professional en productos de Oracle Application, Base de Datos, Cloud & Virtualización.