Cursors In Oracle Database

Cursors

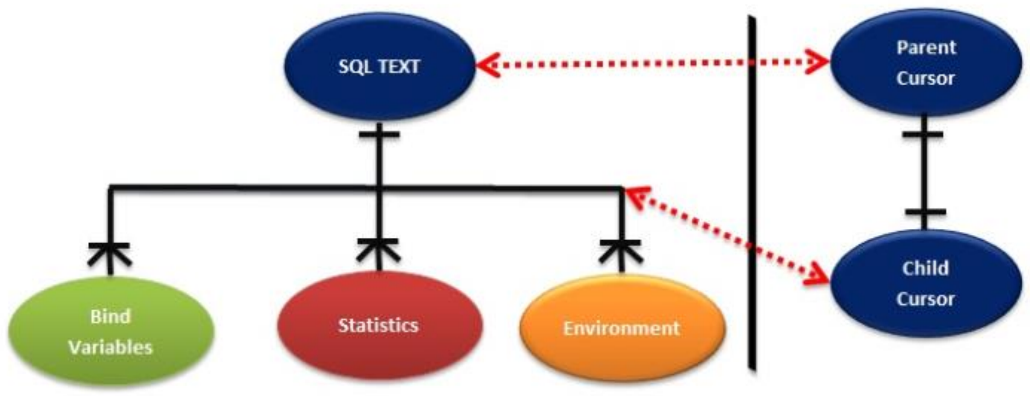
In Oracle, *Cursors* are the temporary private working area where queries are processed. It is used to access the result set present in memory or in DBA terms, A “cursor” is a memory area in the library cache that is allocated to the SQL statement which users execute. This memory area stores key information about the SQL statement like SQL text, SQL execution plan, statistics etc.

Each SQL statement has
- One Parent cursor
- One or more child cursors

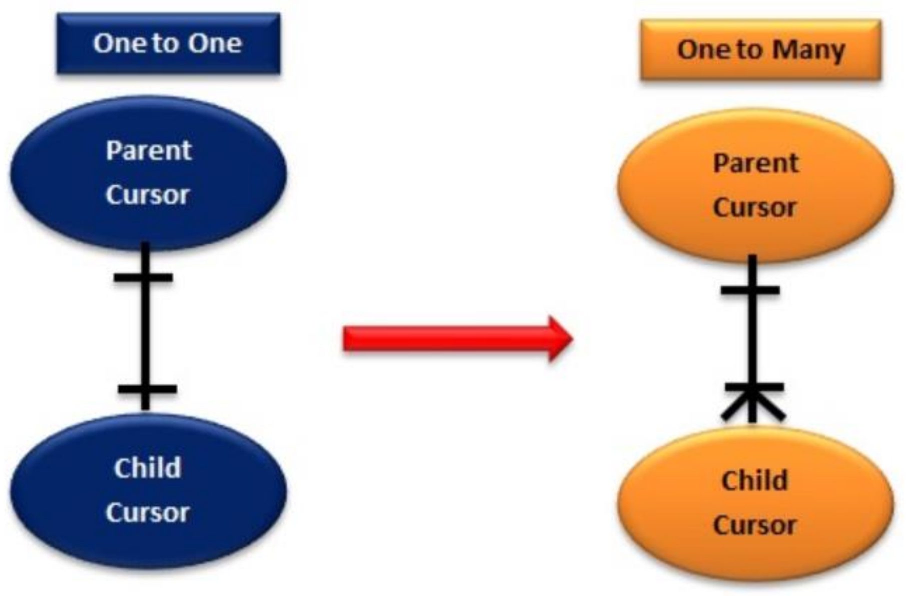
WHY TWO KINDS OF CURSOR?

This is by Oracle database design that you have two kinds of cursors: Parent and Child. For each SQL statement that you execute, Oracle engine will generate two cursors: parent and child cursor. Two cursors are generated because for the same SQL statement, there could be other differences like there can be different bind values or two different schema or different literals values, etc. The parent Cursor will hold the SQL statement and the child cursor will hold the information related to the differences. This essentially makes child cursor as deciding factor as to SQL statement will go for hard or soft parse.

Based on the changing and non-changing components of a cursor they are splitted into parent and child cursors, which looks like below



Thus for every cursor, Oracle internally treats them in a parent child format. It is just the value given as “0” for the initial child cursor. The parent cursor is a representation of the Hash Value and the child cursor represents the metadata for the SQL. When the metadata associated with the SQL starts changing it leads to a creation of a new child cursor. Every cursor has one parent and one or more child cursors. Therefore under these circumstances the 1:1 relation b/w parent and child cursor becomes 1:n i,.e as



PARENT CURSOR

- It stores the SQL text of the cursor. When two statements are identical word-by-word, they will share the same parent Cursor.
- Every parent cursor would execute with at least one child cursor created for it.
- Parent cursors are represented in the view V$SQLAREA. VERSION_COUNT column in the v$sqlarea can tell us how many child cursors does this parent cursor have.

CHILD CURSOR

- Each parent has at least one child cursor and can have more than 1 child cursors also
- While parent cursor stores the SQL Text, the child cursor stores other important information related to SQL statement like: Environment details, Statistics details, Bind Variables details, Execution Plan details Bind Variables details.
- Child Cursor takes less memory space as SQL Text is not stored in child cursor
- Every child cursor must belong to a parent
- Child cursor decides whether a query will undergo a hard parse or a soft parse. You may find situation that SQL query is same for two statements so Parent cursors are same but the child cursor is not shareable to SQL goes for hard parse (re-compile).
- Parent cursors are represented in the view V$SQL
- V$SQL_SHARED_CURSOR is very useful view as it provides the reasons why the optimizer decided mark the cursor as un-shared. So anytime you see that SQL statement was same and still hard parse happened, look at this view.

Since we want to economize on the memory consumption, we would like that equivalent SQL statements should use the same cursor e.g. select * from employees and SELECT * FROM EMPLOYEES achieve the same objective and have the same execution plan and hence only one cursor should be created and should be used when either of the statements is issued. But it won't be so and two parent and hence two child cursors will be created since the two statements are textually different .

If we have two textually identical statements, only one parent cursor will be created but multiple child cursors and hence multiple execution plans can be created if for example bind variables have different values/sizes for different executions of the same statement.

When you have the same statement that has several versions (children), the view v$sql_shared_cursor shows the reason why the statement cannot be shared. You may be able to find that for each child cursor except the first one, why it was not possible to share a previously created child cursor. For several types of incompatibility there is a column that is set to either N (not a mismatch) or Y (mismatch).

Types of cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

CURSOR_SHARING

Oracle 9i introduced the CURSOR_SHARING parameter, which determines how the database handles statements containing literal values.

The parameter CURSOR_SHARING can take 3 values :

- – EXACT
- – SIMILAR
- – FORCE

CURSOR_SHARING = EXACT

In this case when the same statement is issued with different literals, multiple parent cursors will be created.

```
Parent  Parent  Parent
   |       |       |
 Child   Child  Child
```

select count(*) from test where id1=1;
select count(*) from test where id1=2;
select count(*) from test where id1=3;

Above three statements will create 6 cursors , 3 parent and 3 child cursors .

CURSOR_SHARING=SIMILAR

If we replace literal with a bind variable, all the 3 statements
will be identical and hence only parent cursor needs to be created. Multiple
child cursors can be created for different values of the bind variables.That's what
CURSOR_SHARING=SIMILAR does. It replaces literals in the otherwise
identical SQL statements with bind variables and only one parent cursor is
created.

```
 Parent


      +
 +—- —+——-+
 |      |      |
 Child  Child  Child
```

CURSOR_SHARING=FORCE

- Only one child cursor is created if optimizer does not know about skew in   data

- - If optimizer is aware of the skew in data, Multiple child cursors are created   for each distinct value of the bind   variable even if they have the same executiion plan.

Ideally we would like one child cursor to be created if execution plan is same for different values of the bind variable. Setting **CURSOR_SHARING=FORCE** does precisely this but only if the optimizer is aware about the skew in the data.

Cursor Invalidation

There are two components for an executing cursor. A session cached cursor (private area where row source are getting processed) in PGA and sharable execution plan (shared excution plan) in SGA. When a cursor invalidated, the shared cursor in SGA will get obsoleted while the session will continue the execution.

Cursor invalidation occurs when the oracle feels that underlying cursor Is no longer safe to eexute this can happen for many reasons one of then is like if we index that is used by the cursor, we must invalidate the cursor so it is recompiled to get a new plan that does not use the index

The re-execution of the cursor is controlled by the parameter CURSOR_INVALIDATION

CURSOR_INVALIDATION = { DEFERRED | IMMEDIATE }

Deferred invalidation reduces the number of cursor invalidations and spreads the recompilation workload over time. Note that when the recompilation workload is spread over time, a cursor may run with a sub-optimal plan until it is recompiled, and may incur small execution-time overhead.

Immediate invalidation re-execute the cursor as soon as it gets invalid , which might have high workload on a busy system

In the Next Doc we will Dig dip into Cursor Sharing and Invalidations with examples