U1M10.LW.Basic Parallel Execution

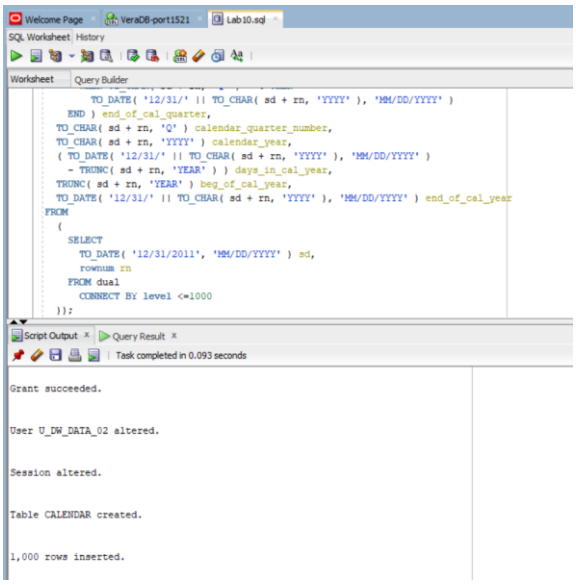
Shkrabatouskaya Vera

https://github.com/VeraShkrabatouskaya/DataMola_Data-Camping-2022

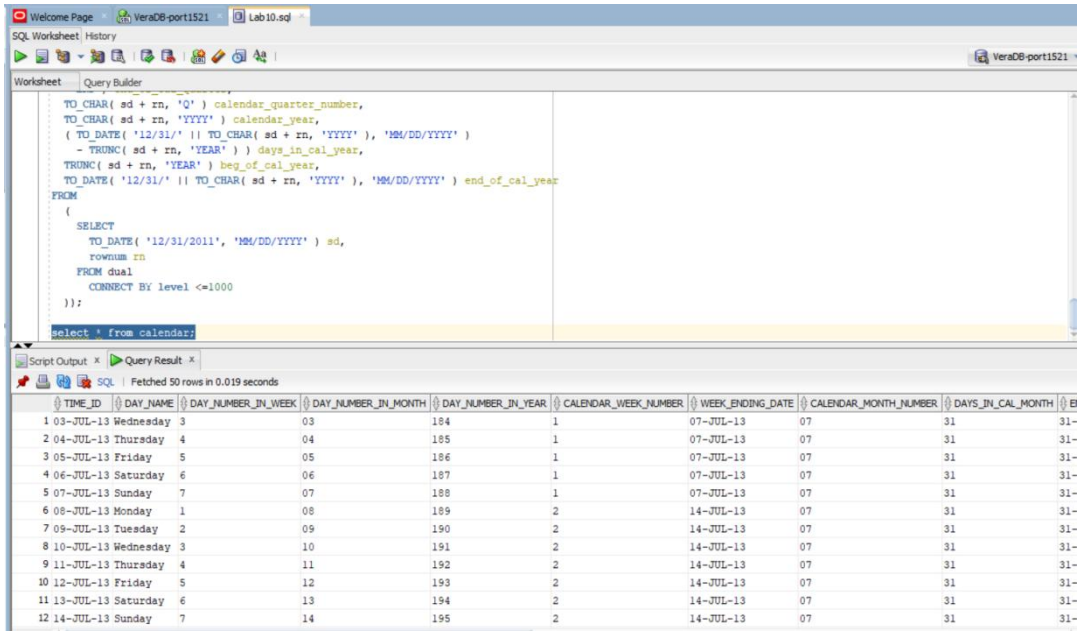
2. Oracle Architecture - Parallel execution

2.1. Task 01: CREATE Example of Select Parallel execution

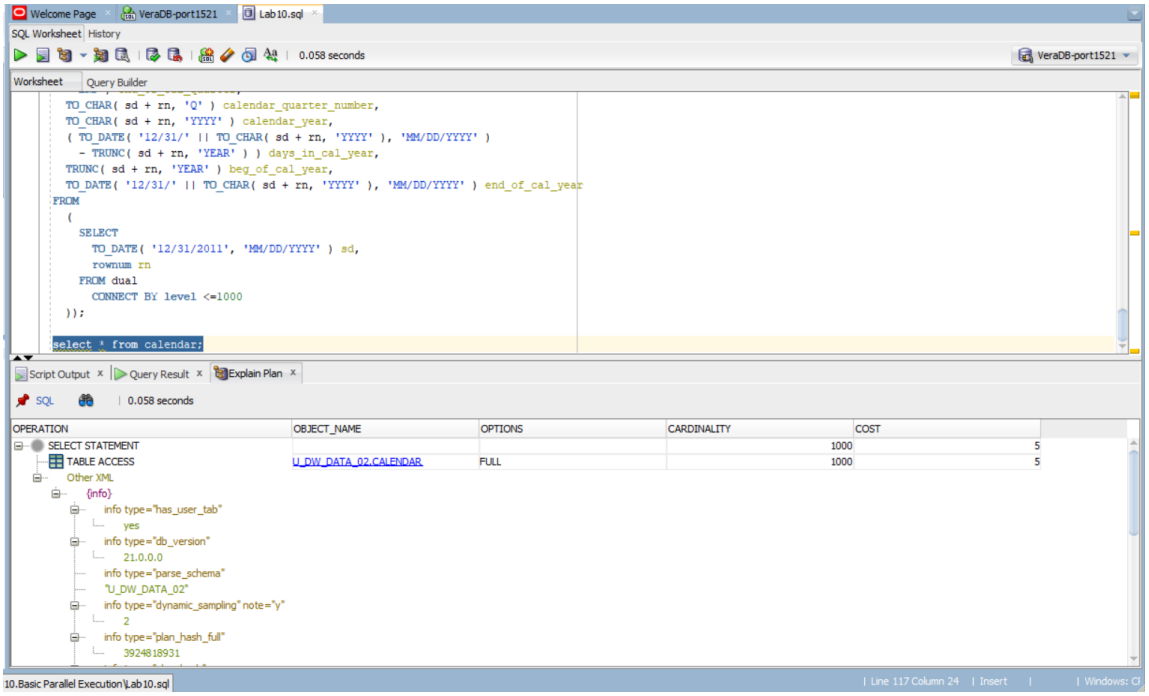
Create table Calendar on 1000 rows.



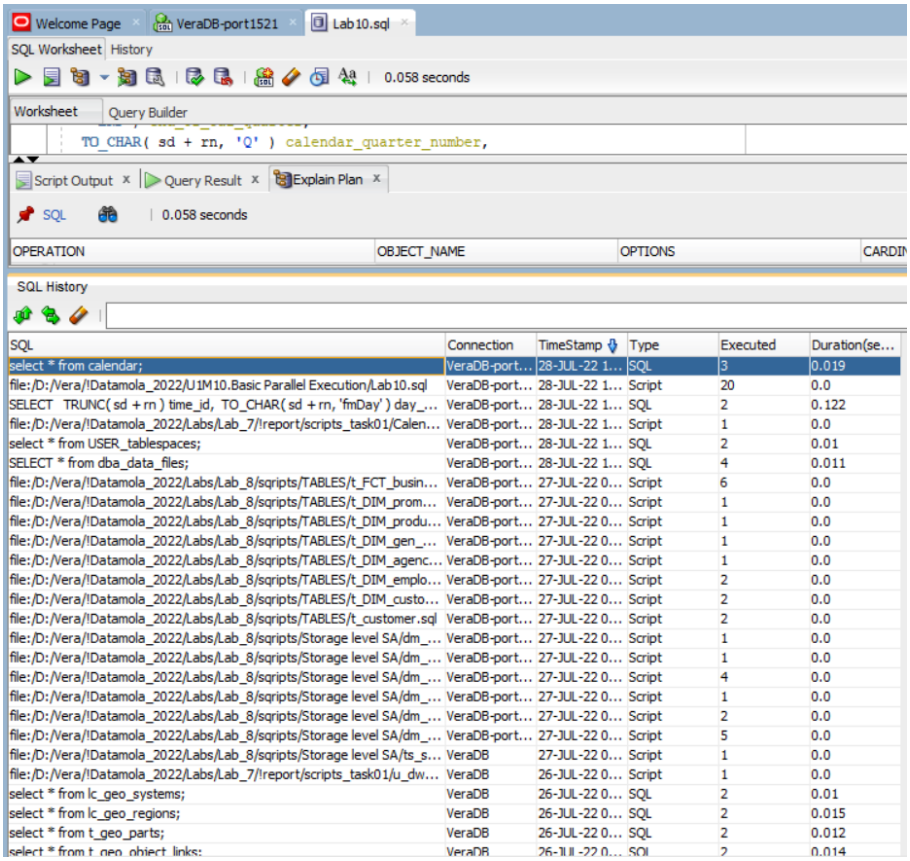
Insert values into the table.



Let's look at the explain plan.

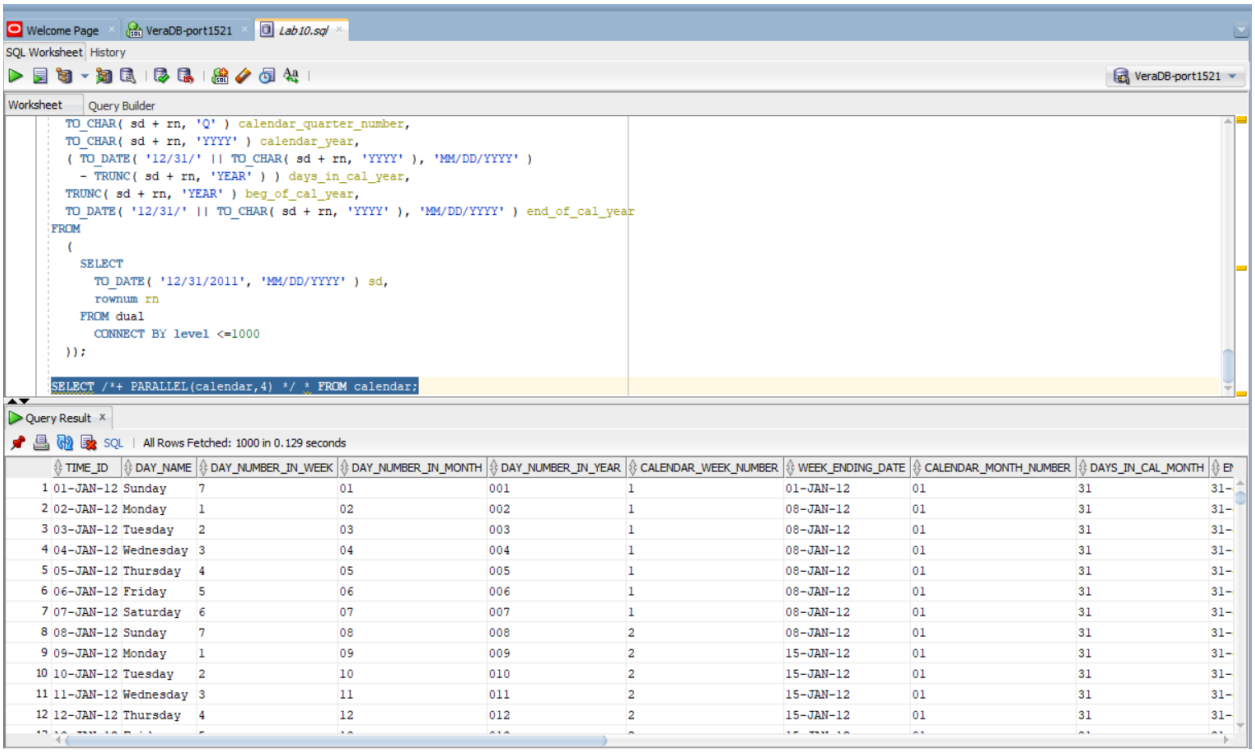


Look at the SQL history.

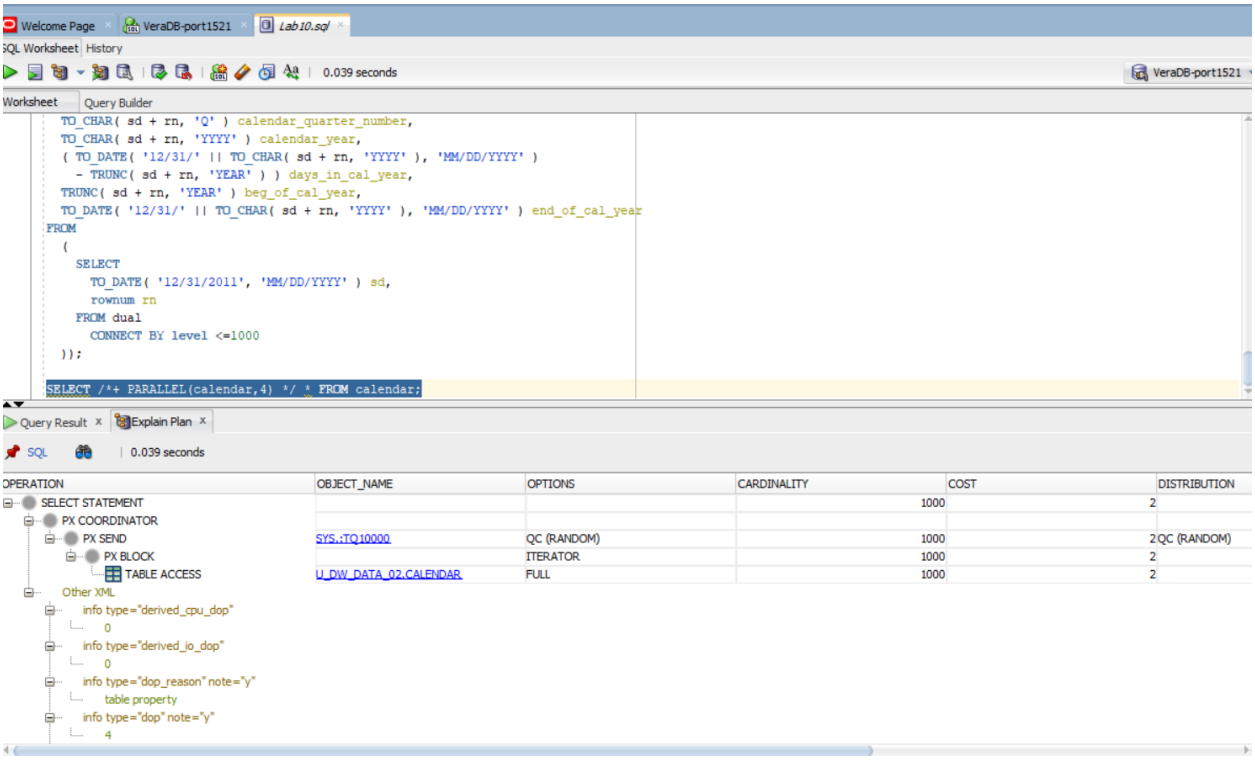


The time it takes to explain plan is 0.058 seconds. The time it takes to execute a request is 0.019 seconds. Cost is 5.

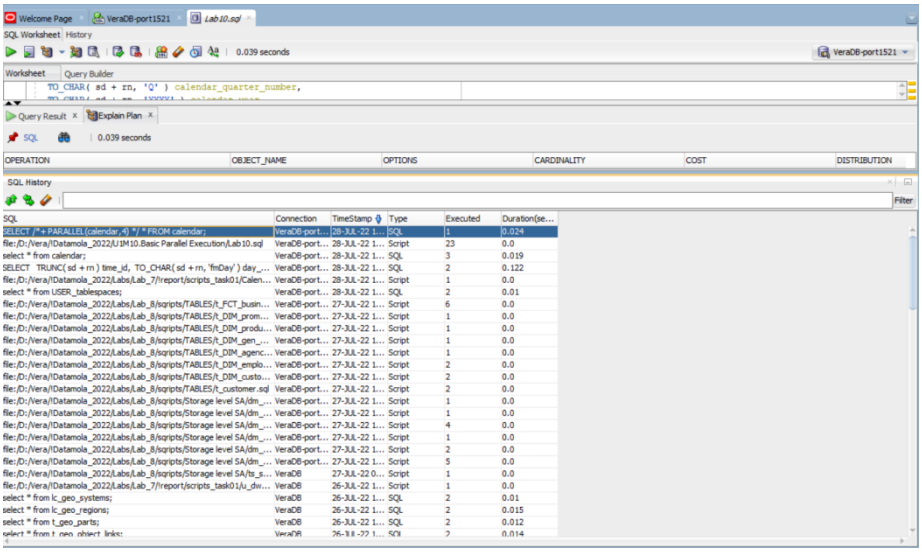
Use Parallel execution:



Let's look at the explain plan with Parallel Hint.

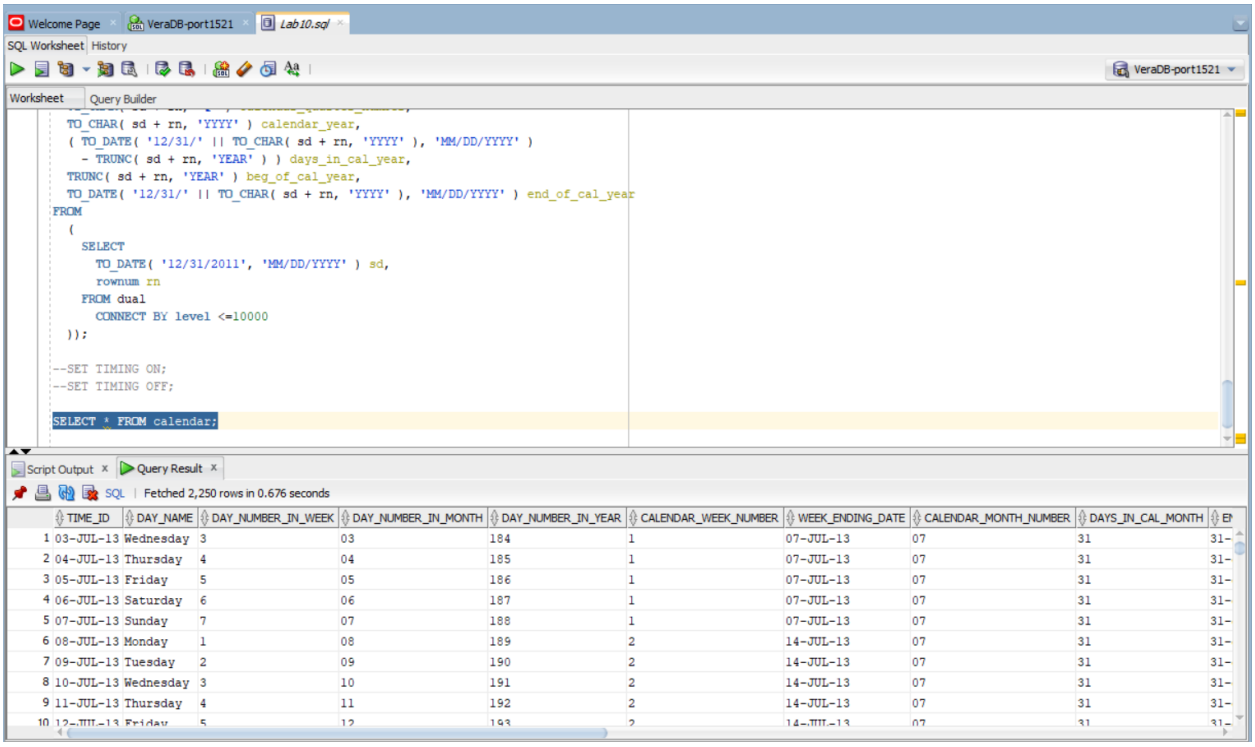


Look at the SQL history of query with Parallel Hint.



The time it takes to explain plan with Parallel Hint is 0.039 seconds. The time it takes to execute a request with Parallel Hint is 0.024 seconds. Cost is 2.

Create table Calendar on 10 000 rows and Insert values into the table.

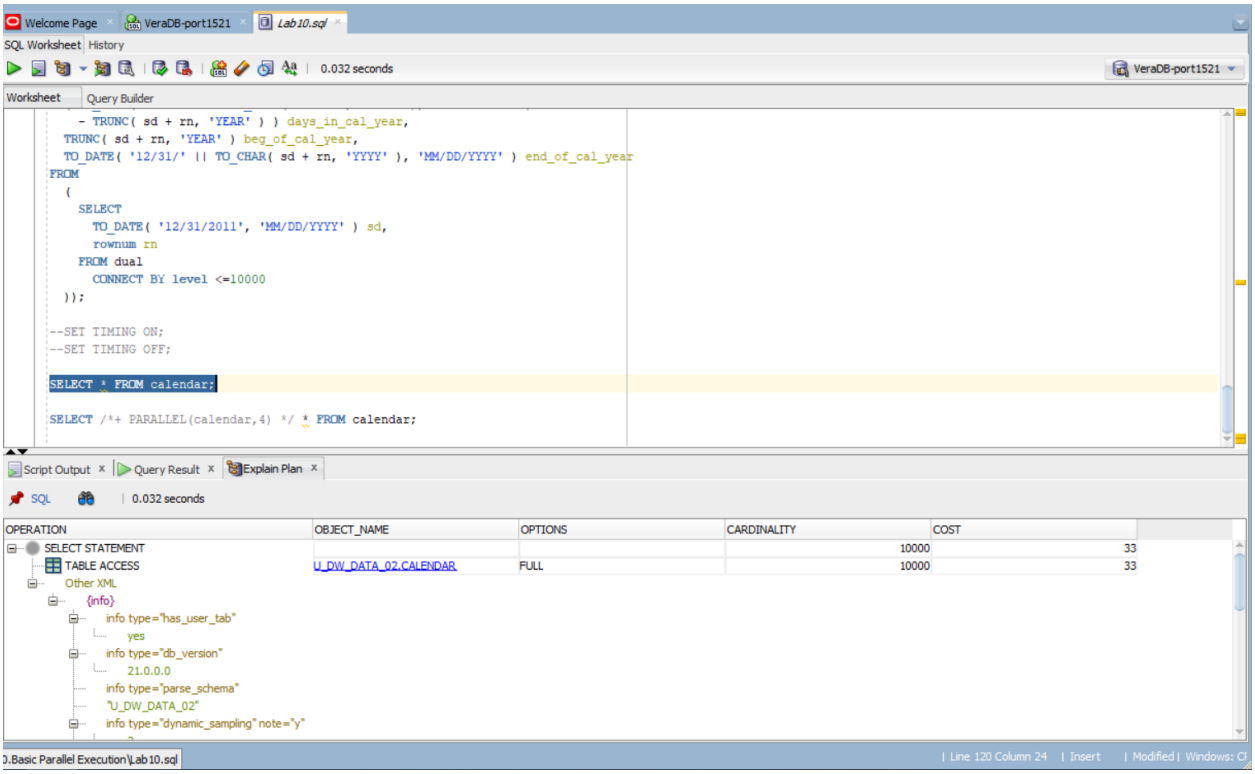


```
  TO_CHAR( sd + rn, 'YYYY' ) calendar_year,
  ( TO_DATE( '12/31/' || TO_CHAR( sd + rn, 'YYYY' ), 'MM/DD/YYYY' )
    - TRUNC( sd + rn, 'YEAR' ) ) days_in_cal_year,
  TRUNC( sd + rn, 'YEAR' ) beg_of_cal_year,
  TO_DATE( '12/31/' || TO_CHAR( sd + rn, 'YYYY' ), 'MM/DD/YYYY' ) end_of_cal_year
FROM
  (
    SELECT
      TO_DATE( '12/31/2011', 'MM/DD/YYYY' ) sd,
      rownum rn
    FROM dual
      CONNECT BY level <=10000
  ));

--SET TIMING ON;
--SET TIMING OFF;

SELECT * FROM calendar;
```

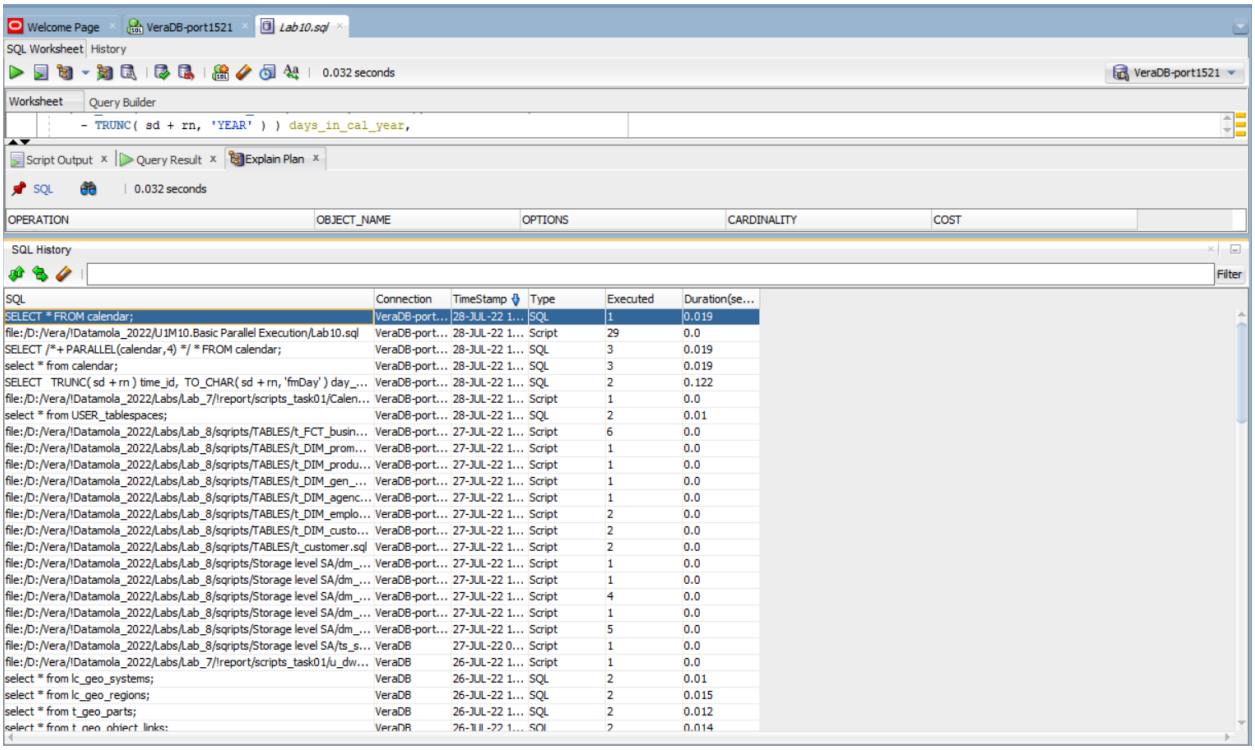
Fetched 2,250 rows in 0.676 seconds

	TIME_ID	DAY_NAME	DAY_NUMBER_IN_WEEK	DAY_NUMBER_IN_MONTH	DAY_NUMBER_IN_YEAR	CALENDAR_WEEK_NUMBER	WEEK_ENDING_DATE	CALENDAR_MONTH_NUMBER	DAYS_IN_CAL_MONTH	EN
1	03-JUL-13	Wednesday	3	03	184	1	07-JUL-13	07	31	31-
2	04-JUL-13	Thursday	4	04	185	1	07-JUL-13	07	31	31-
3	05-JUL-13	Friday	5	05	186	1	07-JUL-13	07	31	31-
4	06-JUL-13	Saturday	6	06	187	1	07-JUL-13	07	31	31-
5	07-JUL-13	Sunday	7	07	188	1	07-JUL-13	07	31	31-
6	08-JUL-13	Monday	1	08	189	2	14-JUL-13	07	31	31-
7	09-JUL-13	Tuesday	2	09	190	2	14-JUL-13	07	31	31-
8	10-JUL-13	Wednesday	3	10	191	2	14-JUL-13	07	31	31-
9	11-JUL-13	Thursday	4	11	192	2	14-JUL-13	07	31	31-
10	12-JUL-13	Friday	5	12	193	2	14-JUL-13	07	31	31-

Let's look at the explain plan.

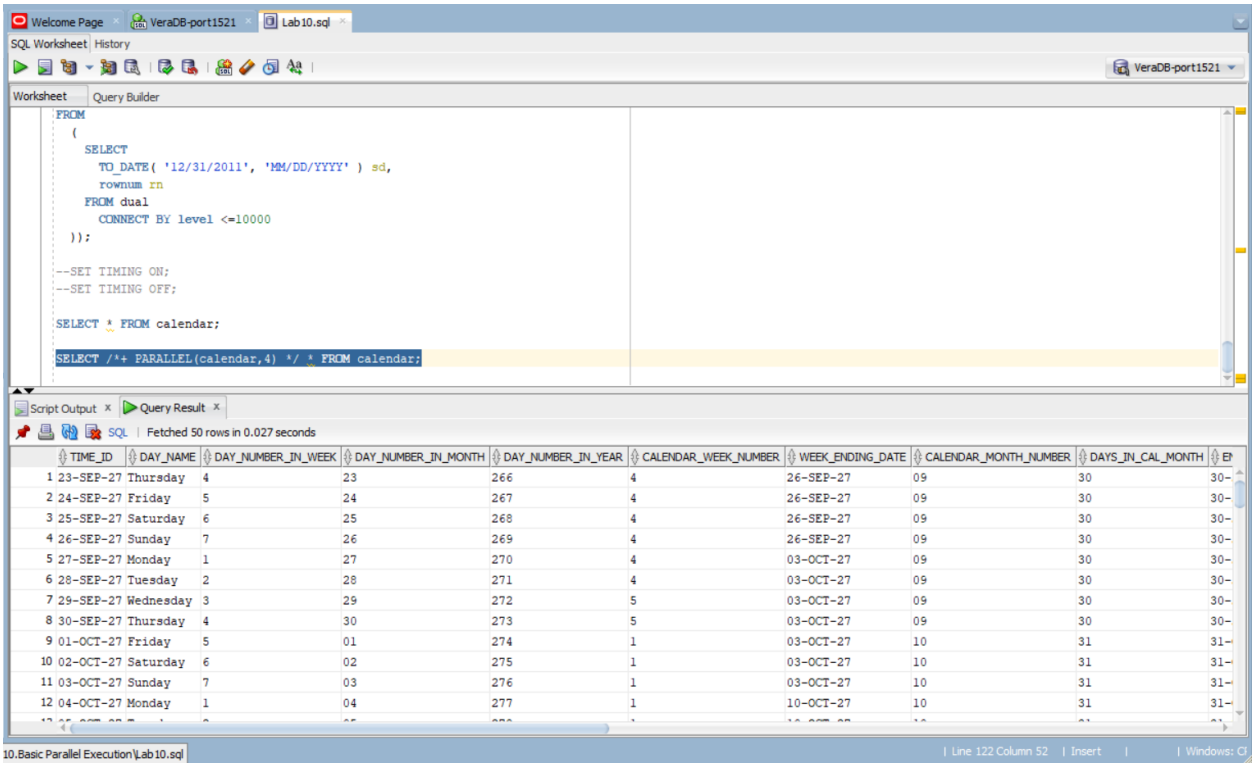


Look at the SQL history.

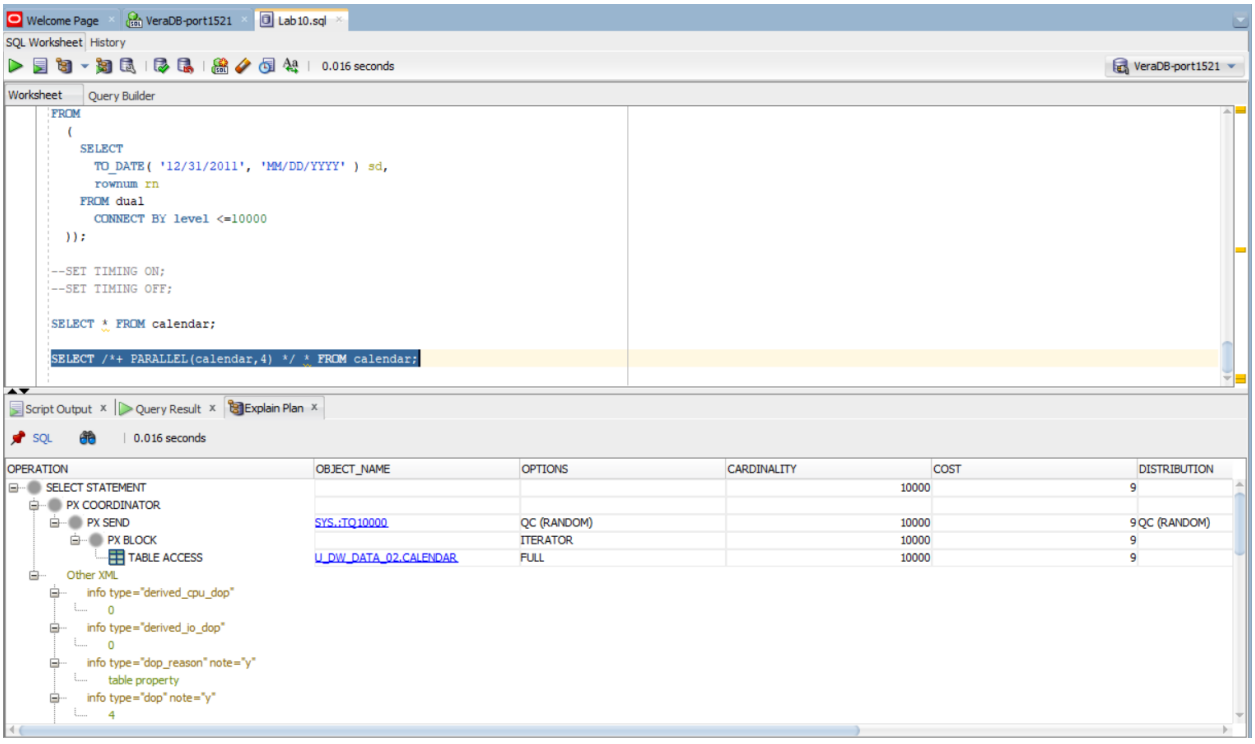


The time it takes to explain plan is 0.032 seconds. The time it takes to execute a request is 0.019 seconds. Cost is 33.

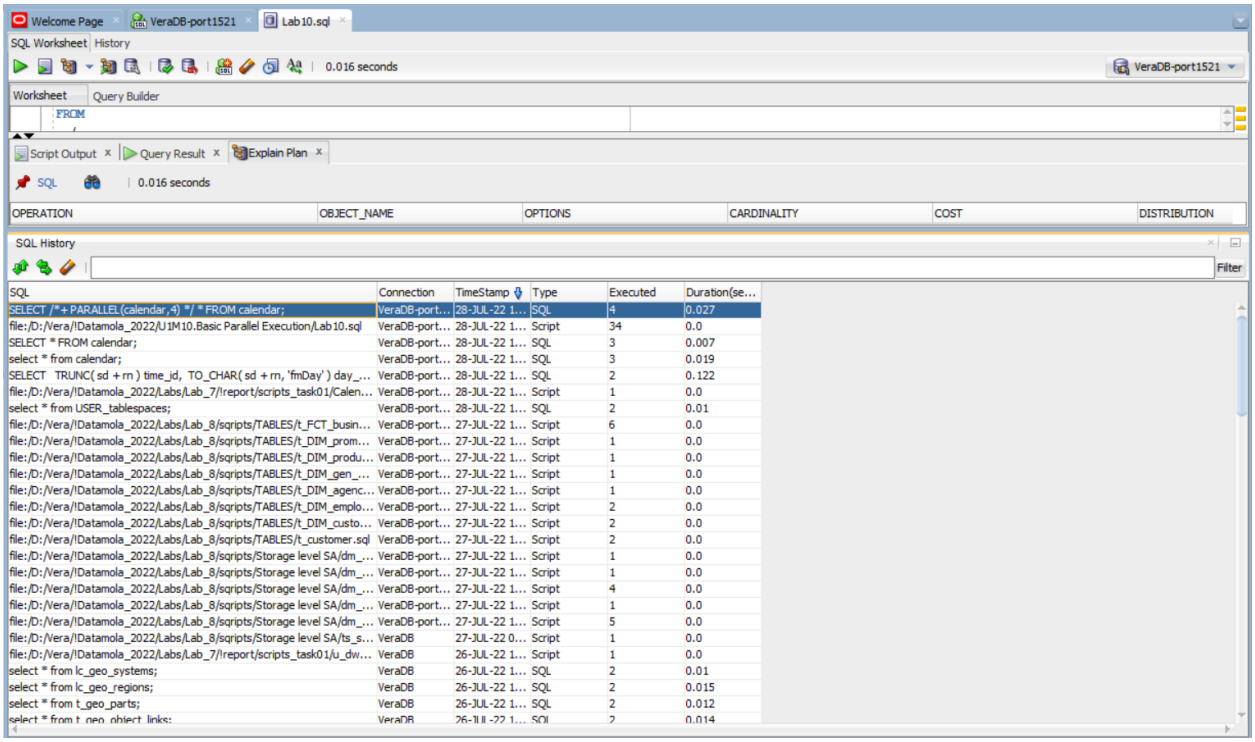
Use Parallel execution:



Let's look at the explain plan with Parallel Hint.



Look at the SQL history of query with Parallel Hint.



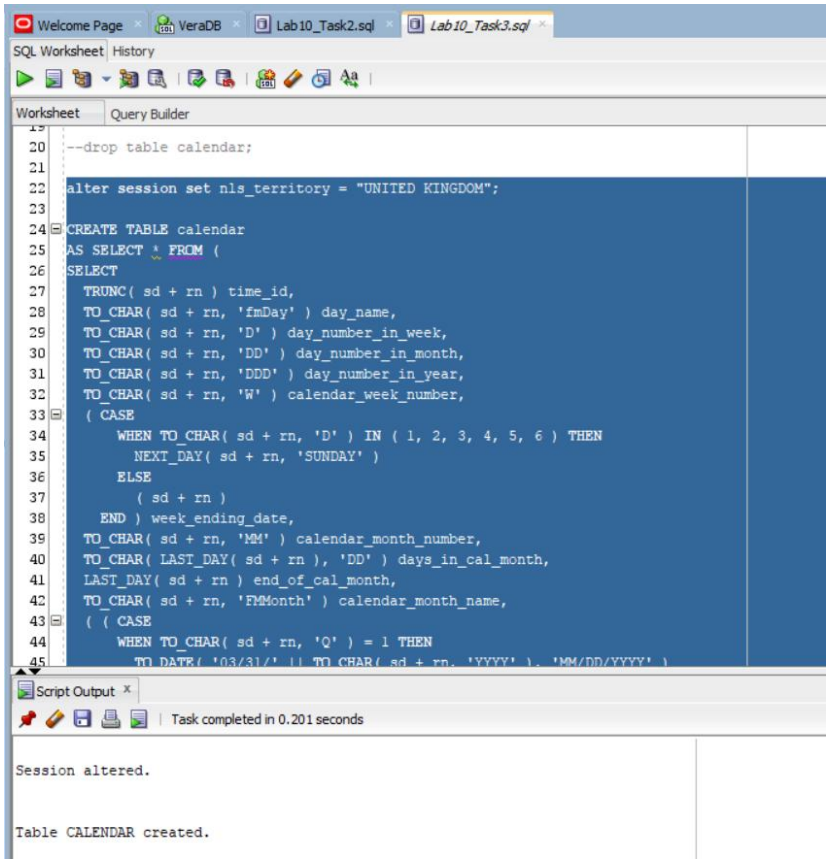
Summary:

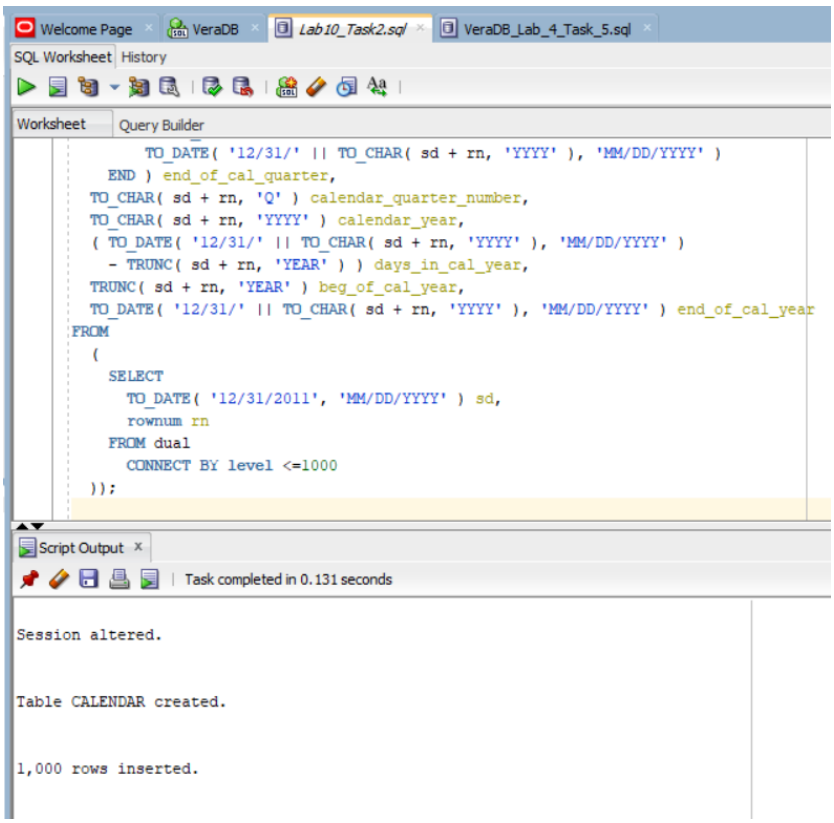
	Row Count	Time to Execute Plan	Time to request	Cost
Simple SELECT	1000	0.058 seconds	0.019 seconds	5
SELECT with Parallel Hint	1000	0.039 seconds	0.024 seconds	2
Simple SELECT	10 000	0.032 seconds	0.019 seconds	33
SELECT with Parallel Hint	10 000	0.016 seconds	0.027 seconds	9

As we can see, when using Select Parallel execution, the cost becomes less, but the execution time on the query increases compared to a simple query.

2.2. Task 02: CREATE Example of Parallel DML

Create one table on 1000 rows from another by adding a SELECT statement at the end of the CREATE TABLE statement without parallelization.





Let's delete the data without parallelization:

```
SET TIMING ON;
DELETE FROM calendar WHERE CALENDAR_MONTH_NAME IN ('June','July','August');
SET TIMING OFF;
```

Script Output | Task completed in 0.028 seconds

```
276 rows deleted.

Elapsed: 00:00:00.016
```

Let's look at the explain plan.

```
explain plan for
DELETE FROM calendar WHERE CALENDAR_MONTH_NAME IN ('June','July','August');

select * from table (dbms_xplan.display);
```

Script Output | Task completed in 0.025 seconds

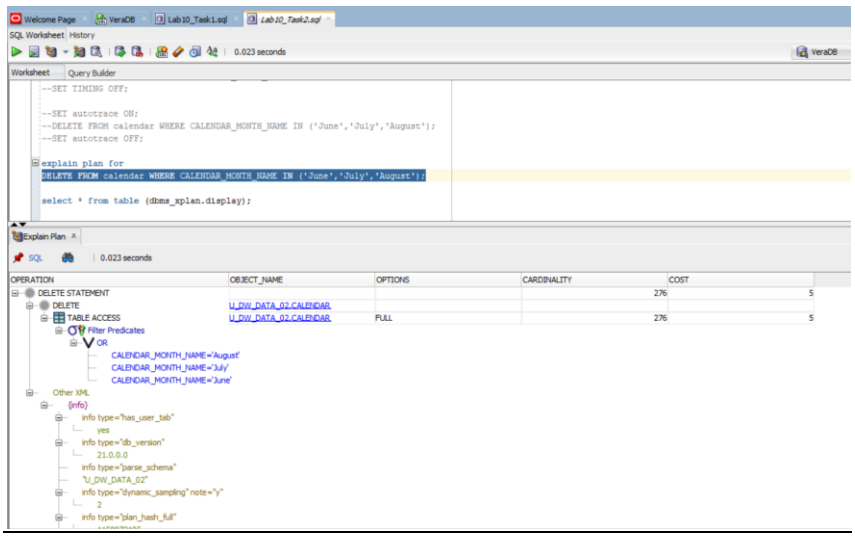
```
Table CALENDAR dropped.


Session altered.


Table CALENDAR created.


1,000 rows inserted.


Explained.
```

```
explain plan for
DELETE FROM calendar WHERE CALENDAR_MONTH_NAME IN ('June','July','August');

select * from table (dbms_xplan.display);
```

Script Output | Query Result | SQL | All Rows Fetched: 19 in 0.104 seconds

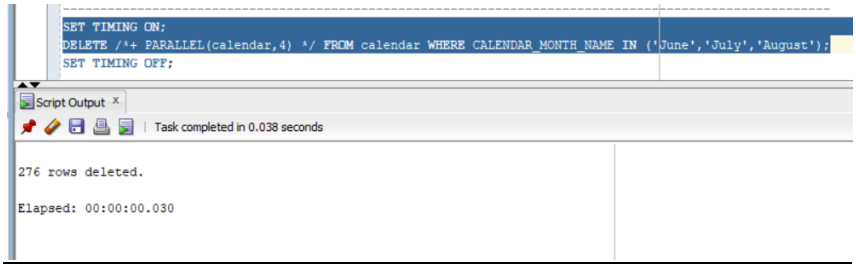
PLAN_TABLE_OUTPUT

```
Plan hash value: 2886354060

-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | DELETE STATEMENT   |          |   276 |  4968 |     5   (0)| 00:00:01 |
|   1 |  DELETE            | CALENDAR |       |       |            |          |
|*  2 |   TABLE ACCESS FULL| CALENDAR |   276 |  4968 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - filter("CALENDAR_MONTH_NAME"='August' OR
              "CALENDAR_MONTH_NAME"='July' OR "CALENDAR_MONTH_NAME"='June')

Note
```

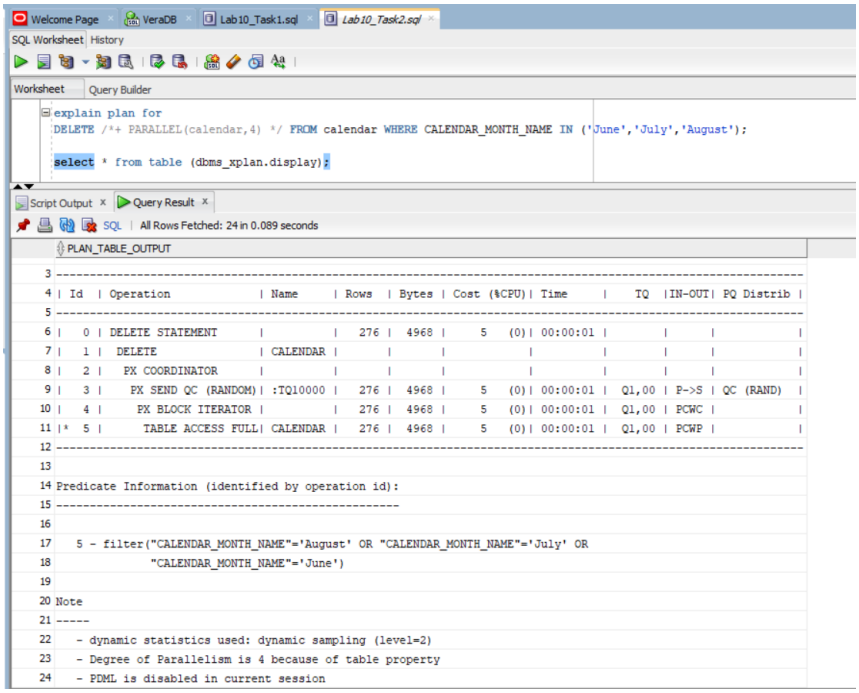


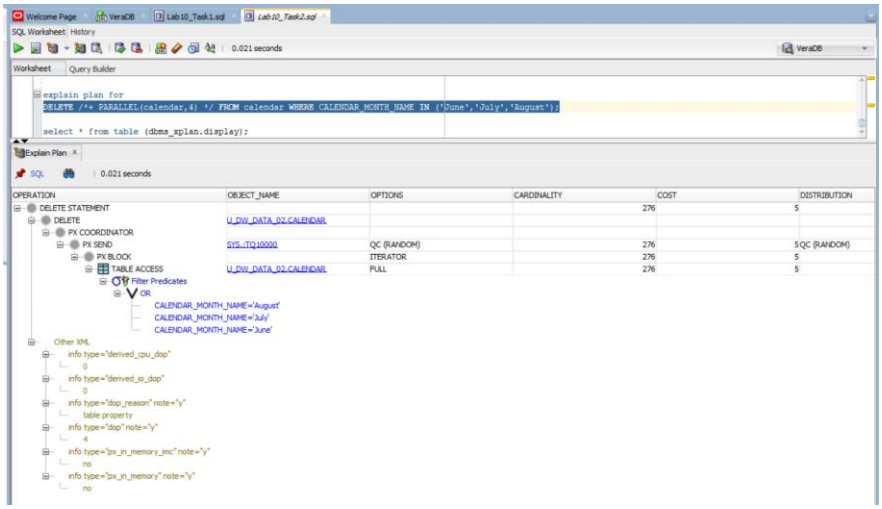
Consider table Calendar on 1000 rows.

Let's delete the data with parallelization:

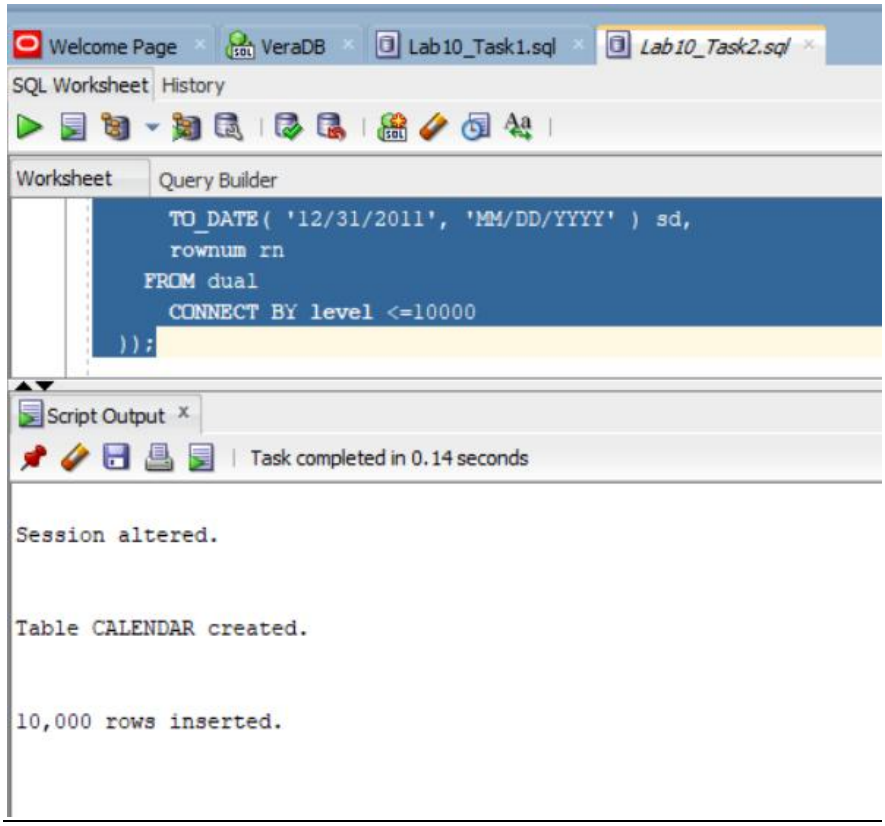


Let's look at the explain plan.

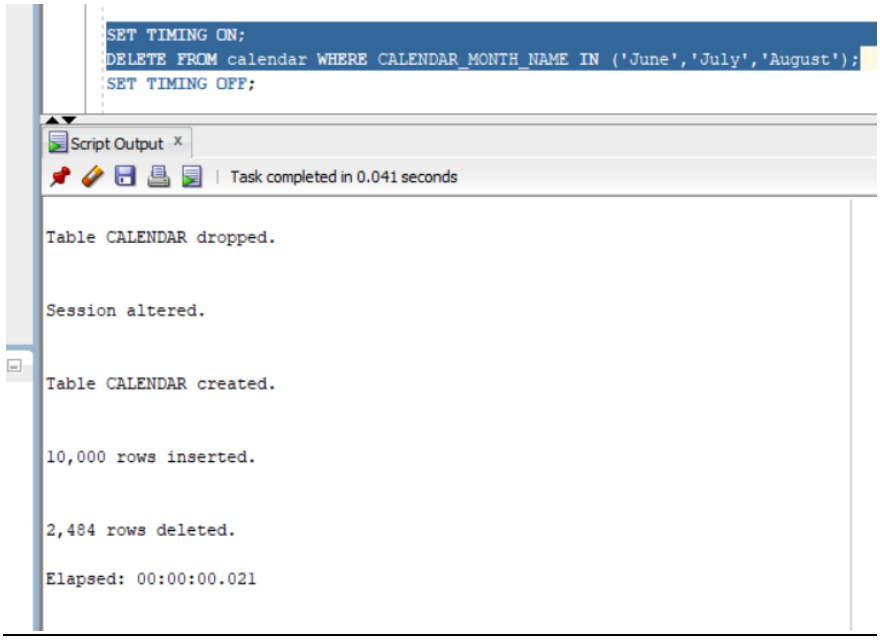




Create table Calendar on 10 000 rows and Insert values into the table.



Let's delete the data without parallelization:

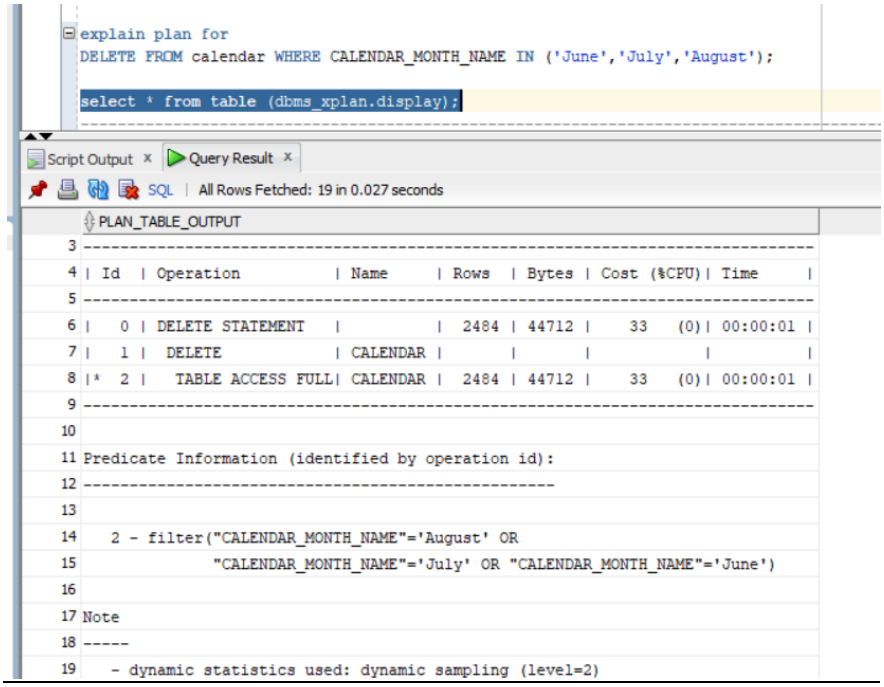


```
SET TIMING ON;
DELETE FROM calendar WHERE CALENDAR_MONTH_NAME IN ('June','July','August');
SET TIMING OFF;
```

Script Output | Task completed in 0.041 seconds

```
Table CALENDAR dropped.


Session altered.


Table CALENDAR created.


10,000 rows inserted.


2,484 rows deleted.

Elapsed: 00:00:00.021
```

Let's look at the explain plan.

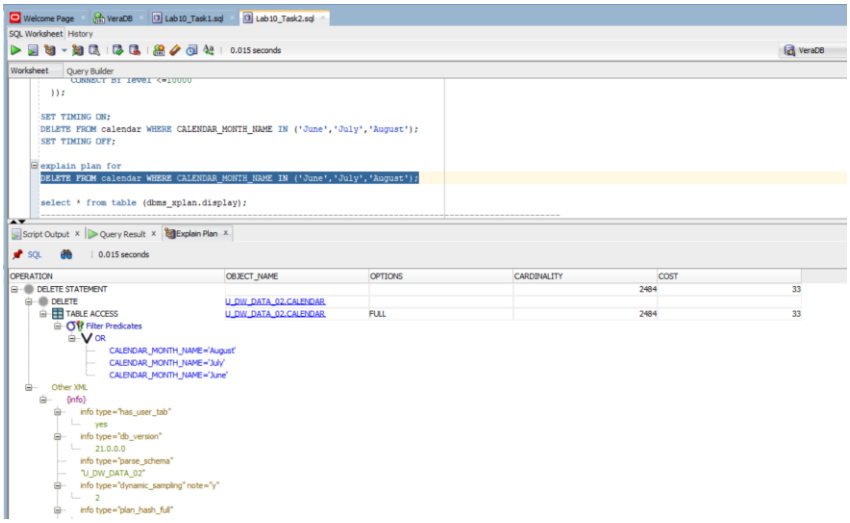


```
explain plan for
DELETE FROM calendar WHERE CALENDAR_MONTH_NAME IN ('June','July','August');

select * from table (dbms_xplan.display);
```

Script Output | Query Result | SQL | All Rows Fetched: 19 in 0.027 seconds

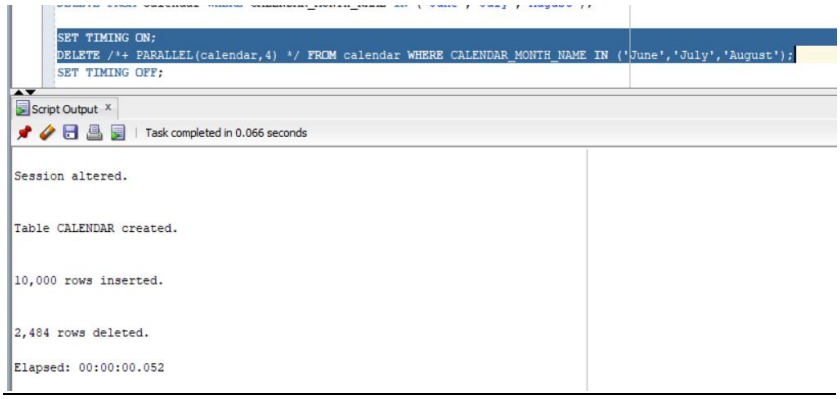
PLAN_TABLE_OUTPUT

```
--------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | DELETE STATEMENT   |          |  2484 | 44712 |    33   (0)| 00:00:01 |
|   1 |  DELETE            | CALENDAR |       |       |            |          |
|*  2 |   TABLE ACCESS FULL| CALENDAR |  2484 | 44712 |    33   (0)| 00:00:01 |
--------------------------------------------------------------------------------

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

   2 - filter("CALENDAR_MONTH_NAME"='August' OR
              "CALENDAR_MONTH_NAME"='July' OR "CALENDAR_MONTH_NAME"='June')

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

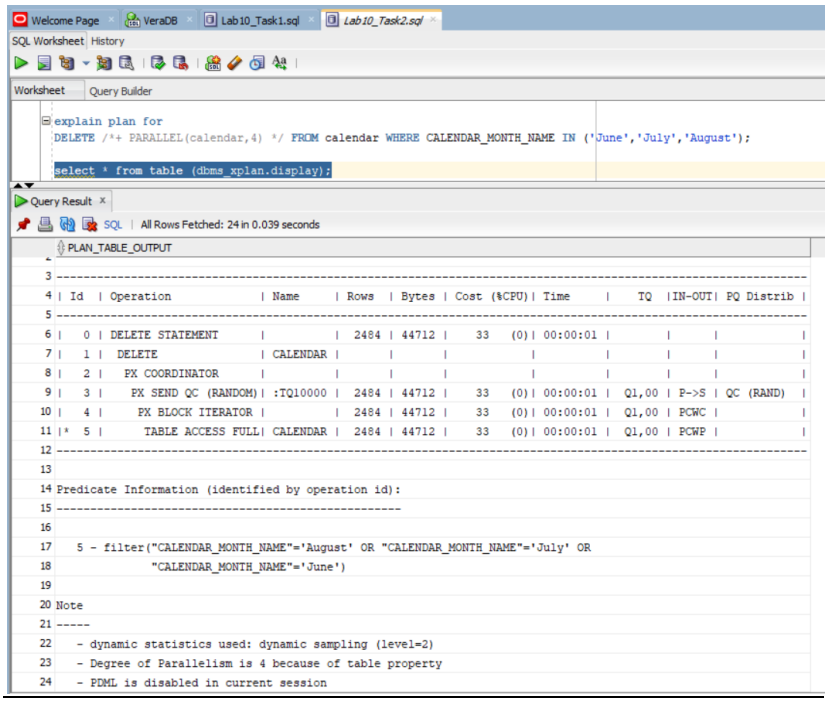


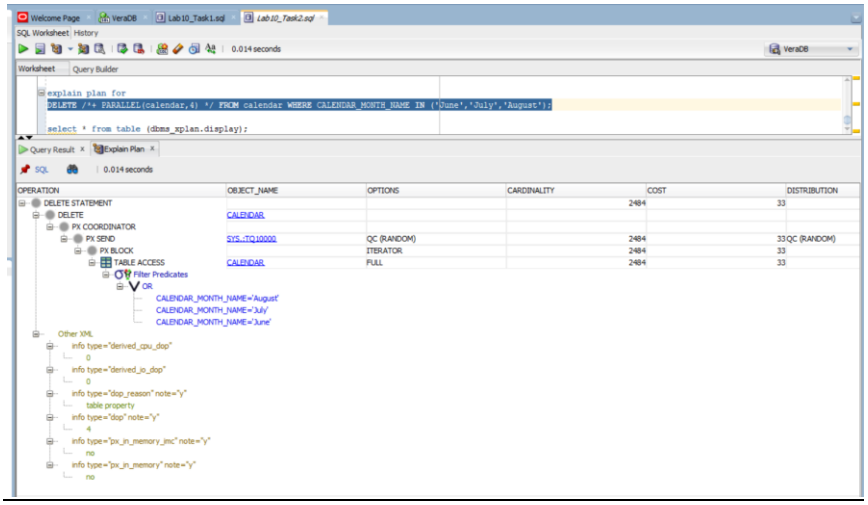
Consider table Calendar on 10 000 rows with parallelization.

Let's delete the data with parallelization:



Let's look at the explain plan.





Summary:

	Number of rows deleted	Time to Execute Elapsed Time	Elapsed Time	Time to DELETE of Explain plane	Cost
Simple DELETE	276	0.028 seconds	00:00:00.016	00:00:01	5
DELETE with Parallel Hint	276	0.038 seconds	00:00:00.030	00:00:01	5
Simple DELETE	2484	0.041 seconds	00:00:00.021	00:00:01	33
DELETE with Parallel Hint	2484	0.066 seconds	00:00:00.052	00:00:01	33

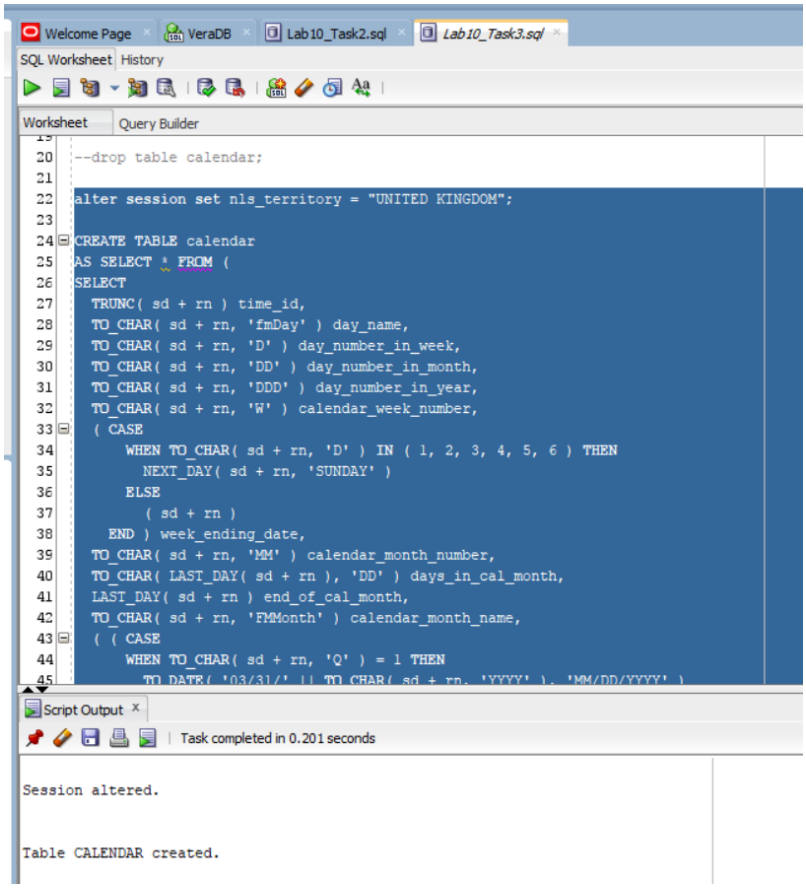
Data Manipulation Language (DML) operations such as INSERT, UPDATE, and DELETE can be parallelized by Oracle. Parallel execution can speed up large DML operations and is particularly advantageous in data warehousing environments where it's necessary to maintain large summary or historical tables. In OLTP systems, parallel DML sometimes can be used to improve the performance of long-running batch jobs.

As you can see from the different options of job 02, when performing operations (DML) with Parallel Hint, the cost does not change, but it takes more time compared to normal operations without Parallel Hint.

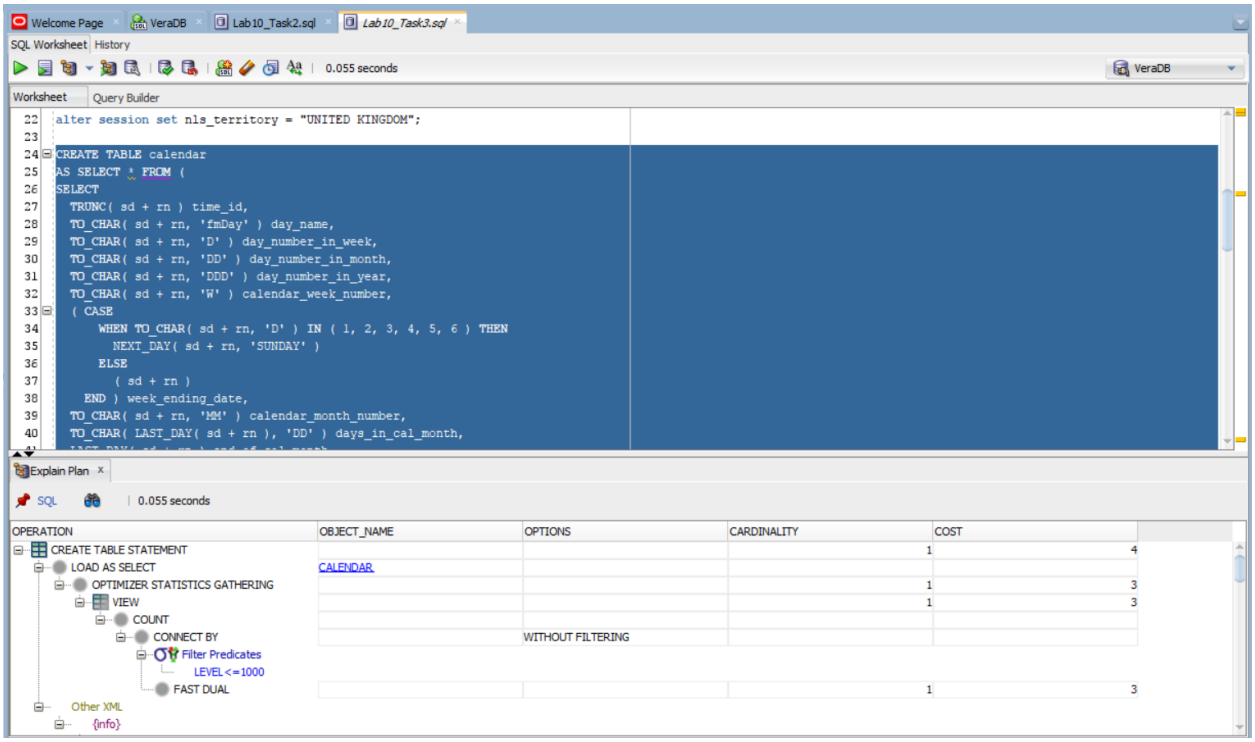
It can be supposed that paralleling does not always improve results for small amounts of data.

2.3. Task 03: CREATE Example of Parallel DDL

Create one table on 1000 rows from another by adding a SELECT statement at the end of the CREATE TABLE statement without Parallel DDL.



Consider Explain plan.

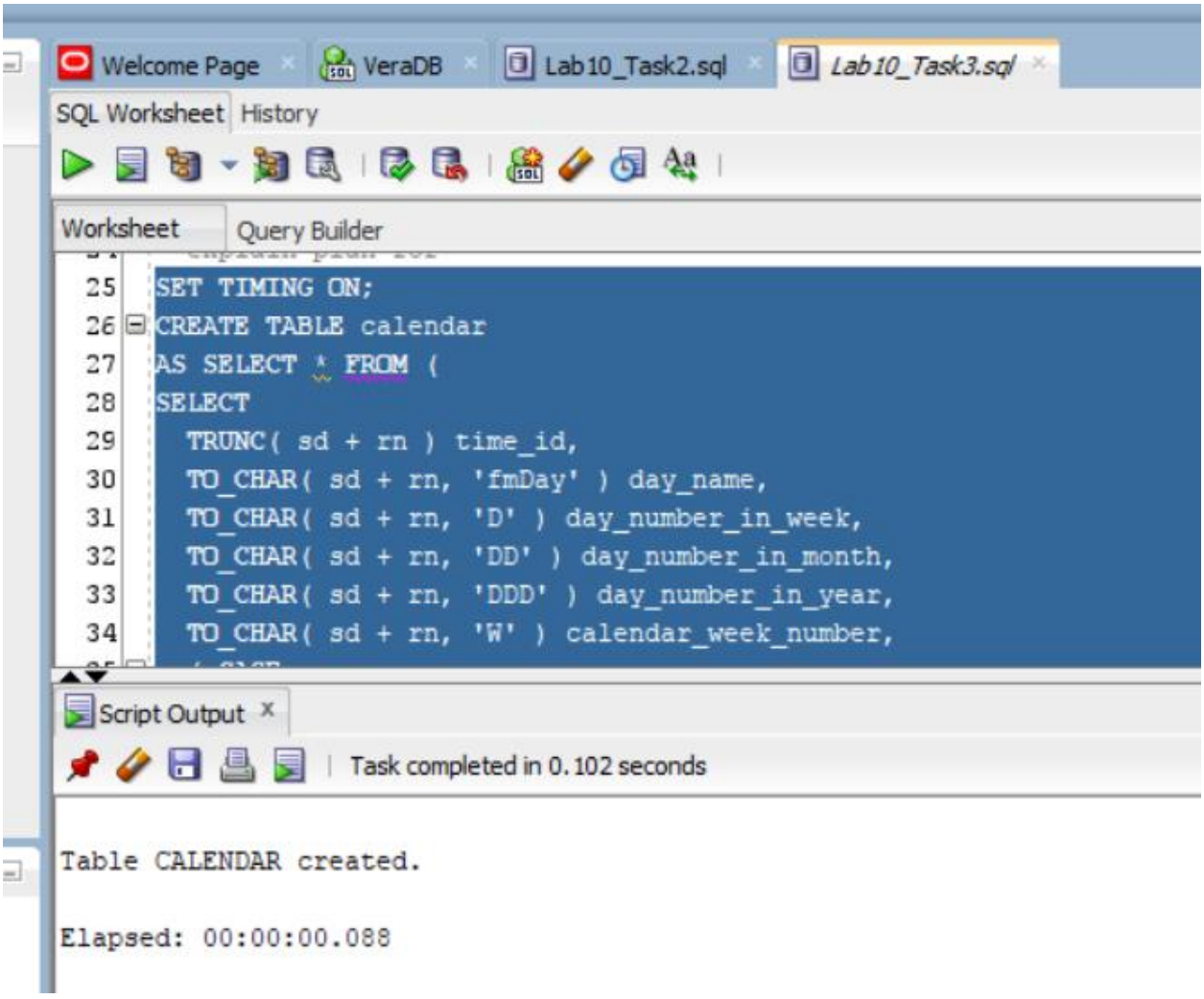


```
97  select * from table (dbms_xplan.display);
```

Script Output | Query Result | SQL | All Rows Fetched: 18 in 0.043 seconds

```
PLAN_TABLE_OUTPUT
1  Plan hash value: 1220224350
2
3  ----------------------------------------------------------------------------------------------
4  | Id  | Operation                         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
5  ----------------------------------------------------------------------------------------------
6  |   0 | CREATE TABLE STATEMENT            |          |     1 |    19 |     4   (0)| 00:00:01 |
7  |   1 |  LOAD AS SELECT                   | CALENDAR |       |       |            |          |
8  |   2 |   OPTIMIZER STATISTICS GATHERING  |          |     1 |    19 |     3   (0)| 00:00:01 |
9  |   3 |    VIEW                           |          |     1 |    19 |     3   (0)| 00:00:01 |
10 |   4 |     COUNT                         |          |       |       |            |          |
11 |*  5 |      CONNECT BY WITHOUT FILTERING |          |       |       |            |          |
12 |   6 |       FAST DUAL                   |          |     1 |       |     3   (0)| 00:00:01 |
13 ----------------------------------------------------------------------------------------------
14
15 Predicate Information (identified by operation id):
16 ---------------------------------------------------
17
18   5 - filter(LEVEL<=1000)
```

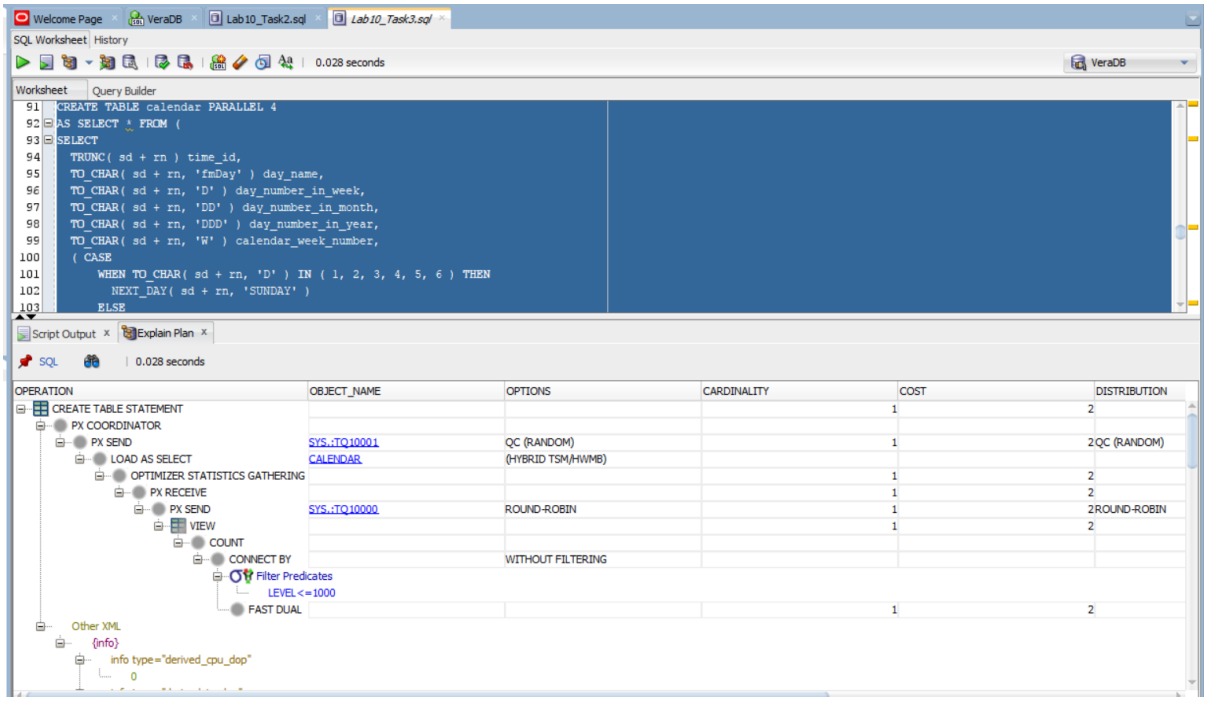
Look on the Elapsed time.



With DDL statements, we do not specify the degree of parallelism using hints. Instead, we use the PARALLEL clause.

Create one table on 1000 rows from another by adding a SELECT statement at the end of the CREATE TABLE statement with Parallel DDL.

Consider Explain plan.



```
CREATE TABLE calendar PARALLEL 4
AS SELECT * FROM (
SELECT
  TRUNC( sd + rn ) time_id,
  TO_CHAR( sd + rn, 'fmDay' ) day_name,
  TO_CHAR( sd + rn, 'D' ) day_number_in_week,
  TO_CHAR( sd + rn, 'DD' ) day_number_in_month,
  TO_CHAR( sd + rn, 'DDD' ) day_number_in_year,
  TO_CHAR( sd + rn, 'W' ) calendar_week_number,
  ( CASE
      WHEN TO_CHAR( sd + rn, 'D' ) IN ( 1, 2, 3, 4, 5, 6 ) THEN
        NEXT_DAY( sd + rn, 'SUNDAY' )
      ELSE
```

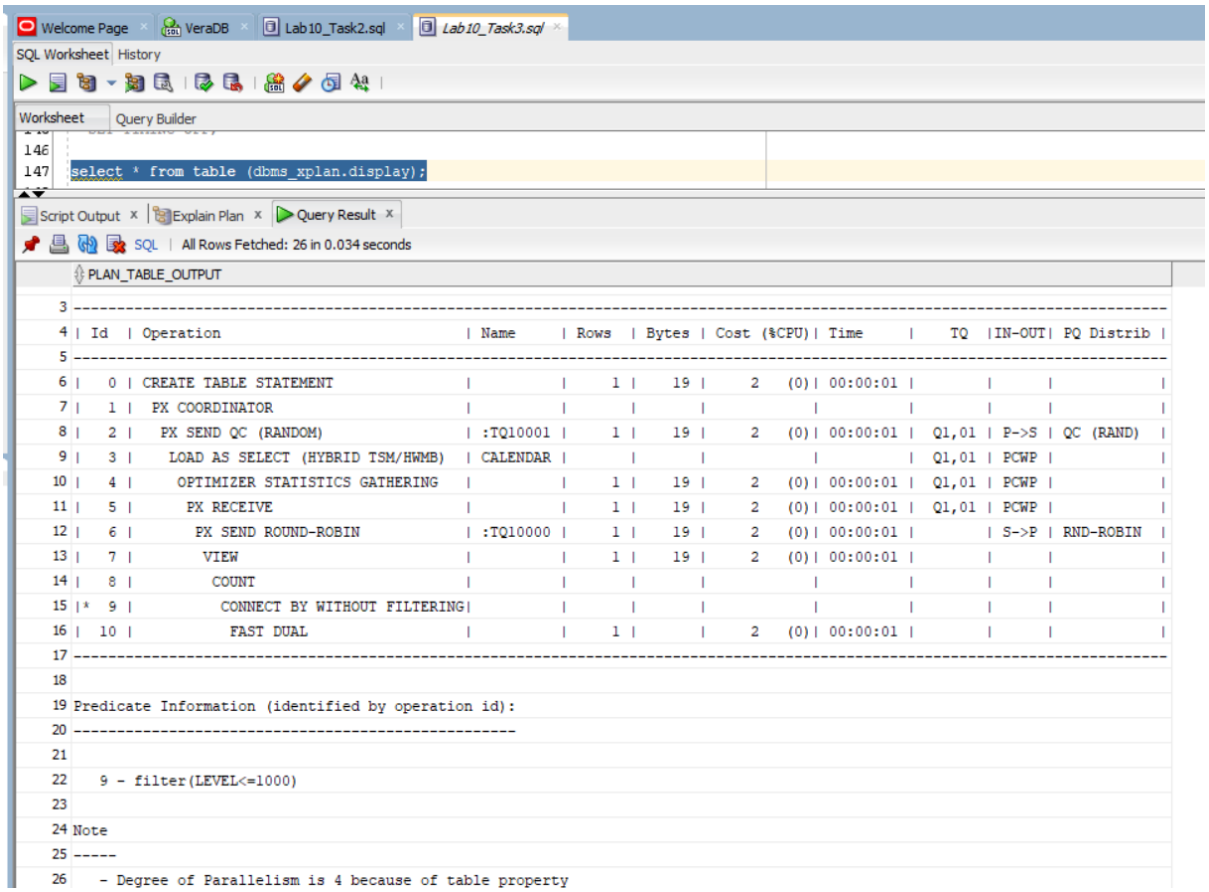
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	DISTRIBUTION
CREATE TABLE STATEMENT			1	2	
PX COORDINATOR					
PX SEND	SYS.:TQ10001	QC (RANDOM)	1	2	QC (RANDOM)
LOAD AS SELECT	CALENDAR	(HYBRID TSM/HWMB)			
OPTIMIZER STATISTICS GATHERING			1	2	
PX RECEIVE			1	2	
PX SEND	SYS.:TQ10000	ROUND-ROBIN	1	2	ROUND-ROBIN
VIEW			1	2	
COUNT					
CONNECT BY		WITHOUT FILTERING			
Filter Predicates					
LEVEL<=1000					
FAST DUAL			1	2	

Other XML

{info}

info type="derived_cpu_dop"

0



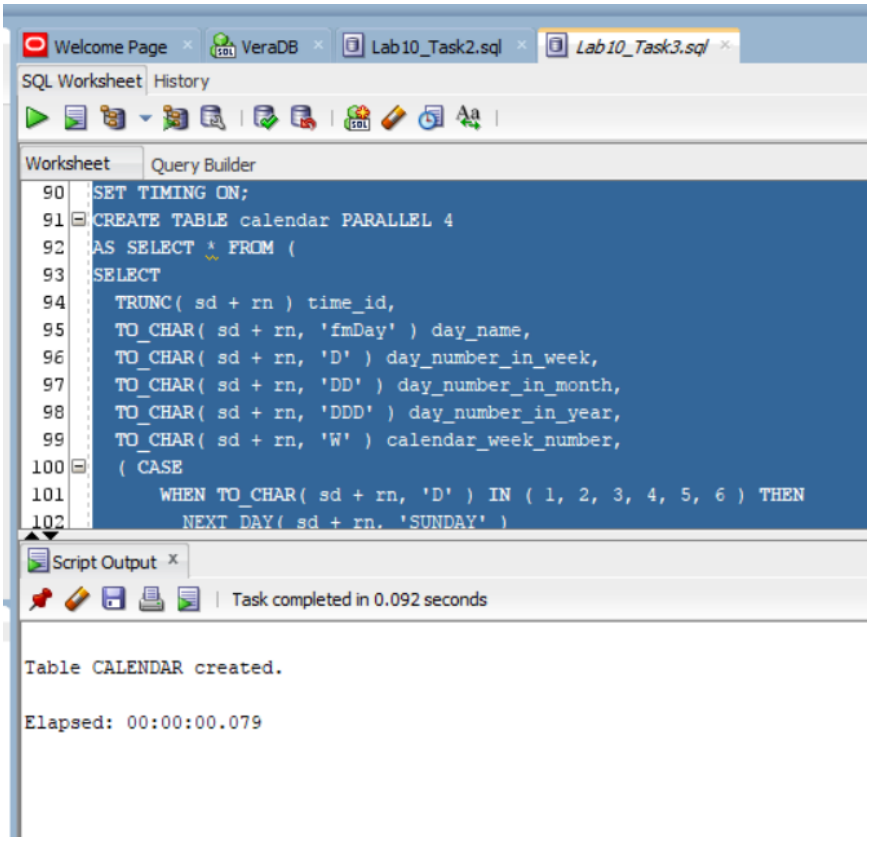
```
select * from table (dbms_xplan.display);
```

All Rows Fetched: 26 in 0.034 seconds

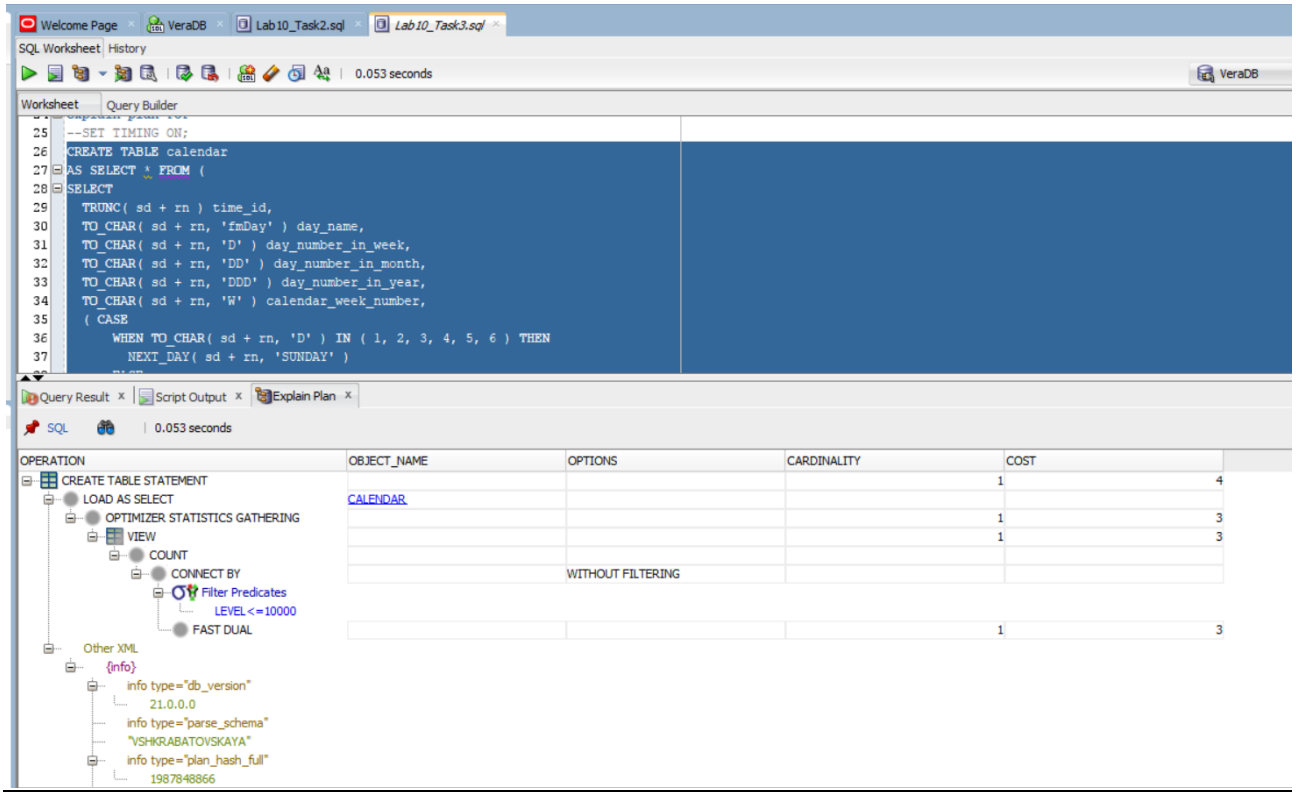
PLAN_TABLE_OUTPUT

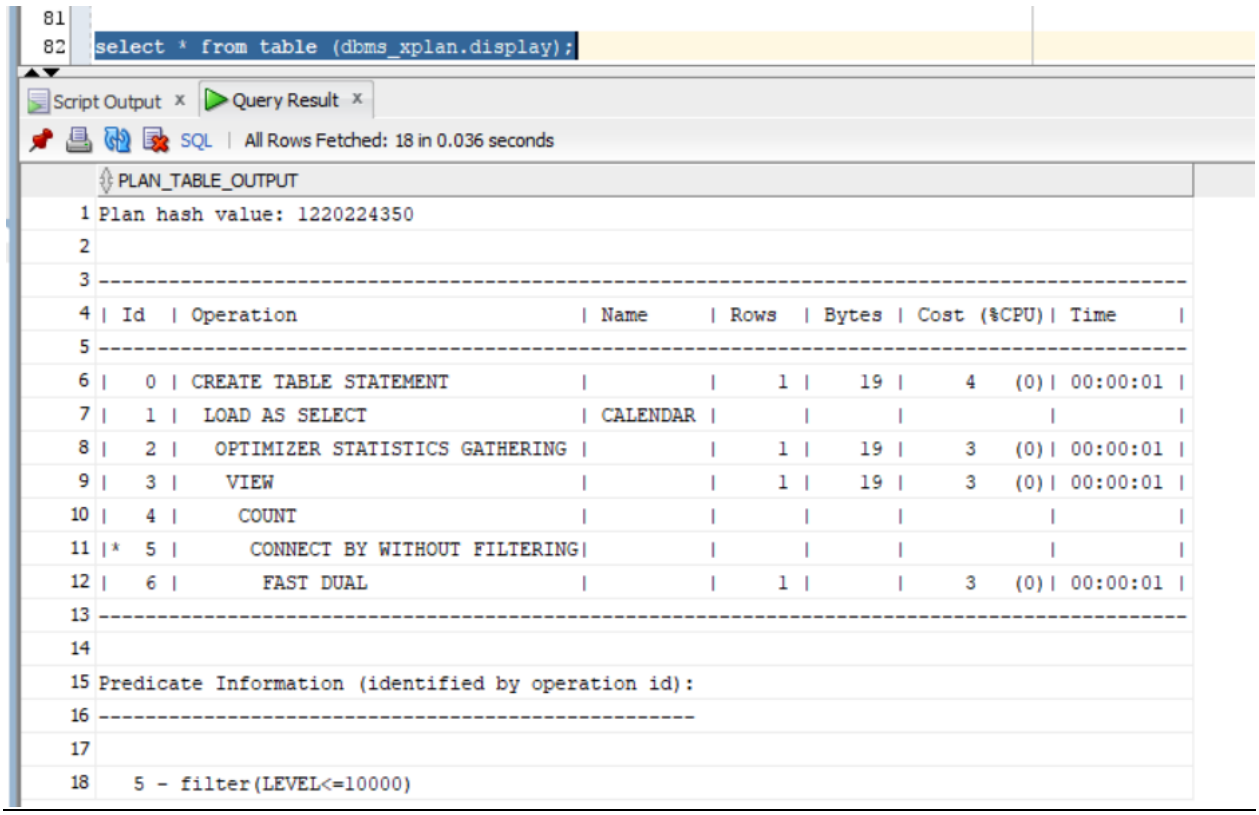
```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------------
|   0 | CREATE TABLE STATEMENT             |          |     1 |    19 |     2   (0)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR                    |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)              | :TQ10001 |     1 |    19 |     2   (0)| 00:00:01 |  Q1,01 | P->S | QC (RAND)  |
|   3 |    LOAD AS SELECT (HYBRID TSM/HWMB)| CALENDAR |       |       |            |          |  Q1,01 | PCWP |            |
|   4 |     OPTIMIZER STATISTICS GATHERING |          |     1 |    19 |     2   (0)| 00:00:01 |  Q1,01 | PCWP |            |
|   5 |      PX RECEIVE                    |          |     1 |    19 |     2   (0)| 00:00:01 |  Q1,01 | PCWP |            |
|   6 |       PX SEND ROUND-ROBIN          | :TQ10000 |     1 |    19 |     2   (0)| 00:00:01 |        | S->P | RND-ROBIN  |
|   7 |        VIEW                        |          |     1 |    19 |     2   (0)| 00:00:01 |        |      |            |
|   8 |         COUNT                      |          |       |       |            |          |        |      |            |
|*  9 |          CONNECT BY WITHOUT FILTERING|        |       |       |            |          |        |      |            |
|  10 |           FAST DUAL                |          |     1 |       |     2   (0)| 00:00:01 |        |      |            |
-------------------------------------------------------------------------------------------------------------------------

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

   9 - filter(LEVEL<=1000)

Note
-----
   - Degree of Parallelism is 4 because of table property
```

Look on the Elapsed time with Parallel DDL.



Let's check the same operations for a table of 10,000 rows.



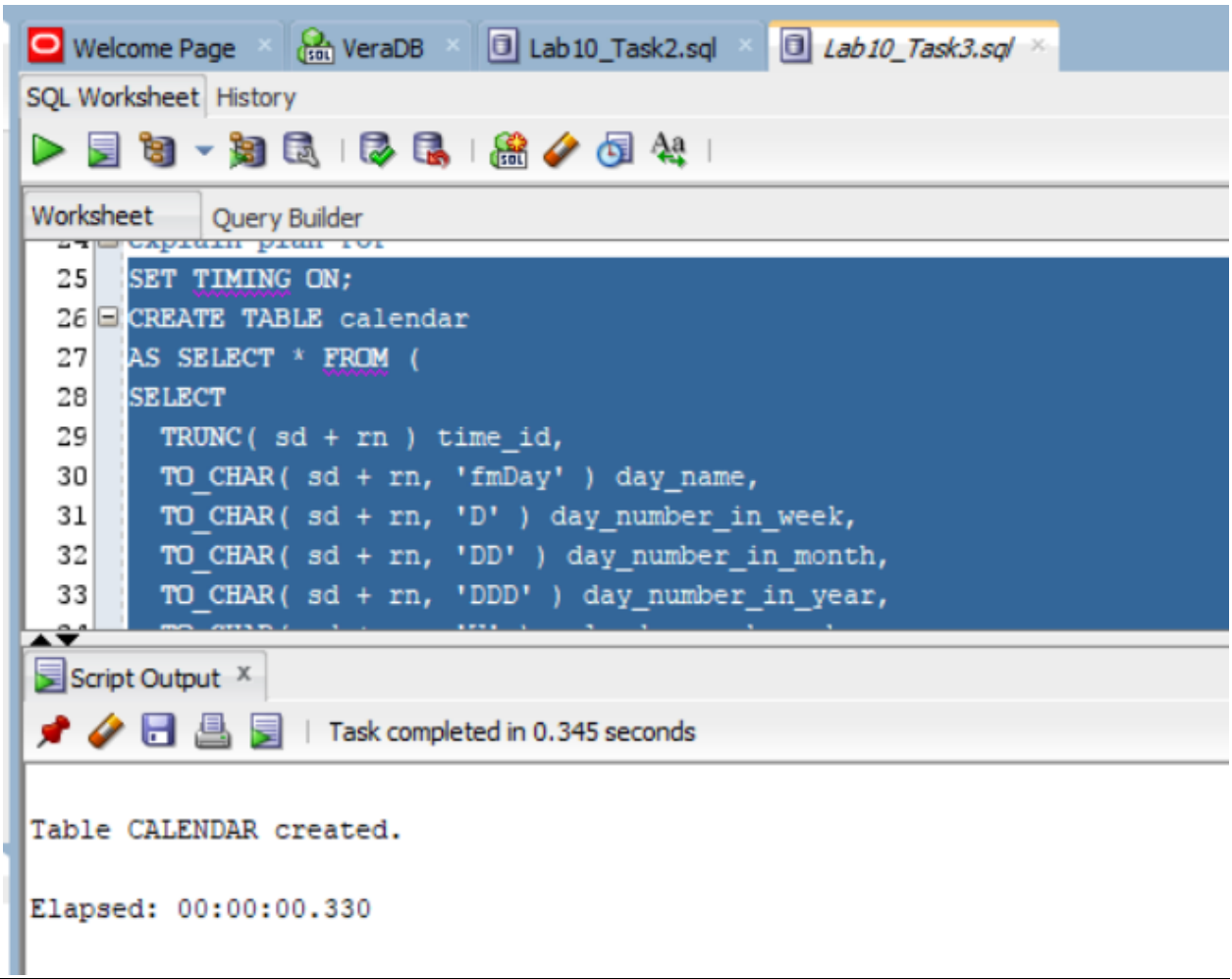


```
select * from table (dbms_xplan.display);
```

Script Output | Query Result | SQL | All Rows Fetched: 18 in 0.036 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 1220224350

-----------------------------------------------------------------------------------------------
| Id  | Operation                          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | CREATE TABLE STATEMENT             |          |     1 |    19 |     4   (0)| 00:00:01 |
|   1 |  LOAD AS SELECT                    | CALENDAR |       |       |            |          |
|   2 |   OPTIMIZER STATISTICS GATHERING   |          |     1 |    19 |     3   (0)| 00:00:01 |
|   3 |    VIEW                            |          |     1 |    19 |     3   (0)| 00:00:01 |
|   4 |     COUNT                          |          |       |       |            |          |
|*  5 |      CONNECT BY WITHOUT FILTERING  |          |       |       |            |          |
|   6 |       FAST DUAL                    |          |     1 |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   5 - filter(LEVEL<=10000)
```

Look on the Elapsed time.



Welcome Page | VeraDB | Lab10_Task2.sql | *Lab10_Task3.sql*

SQL Worksheet | History

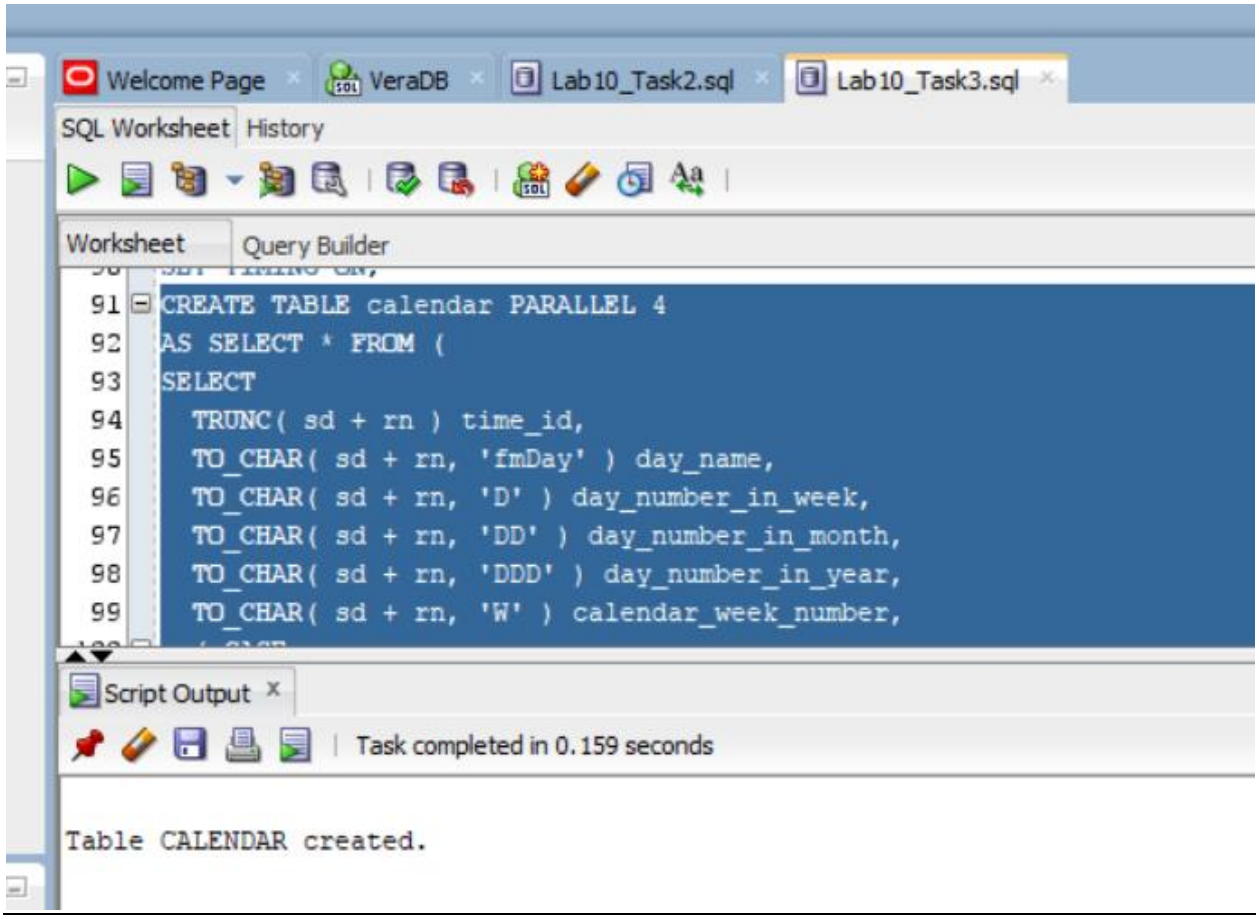
Worksheet | Query Builder

```
SET TIMING ON;
CREATE TABLE calendar
AS SELECT * FROM (
SELECT
  TRUNC( sd + rn ) time_id,
  TO_CHAR( sd + rn, 'fmDay' ) day_name,
  TO_CHAR( sd + rn, 'D' ) day_number_in_week,
  TO_CHAR( sd + rn, 'DD' ) day_number_in_month,
  TO_CHAR( sd + rn, 'DDD' ) day_number_in_year,
```

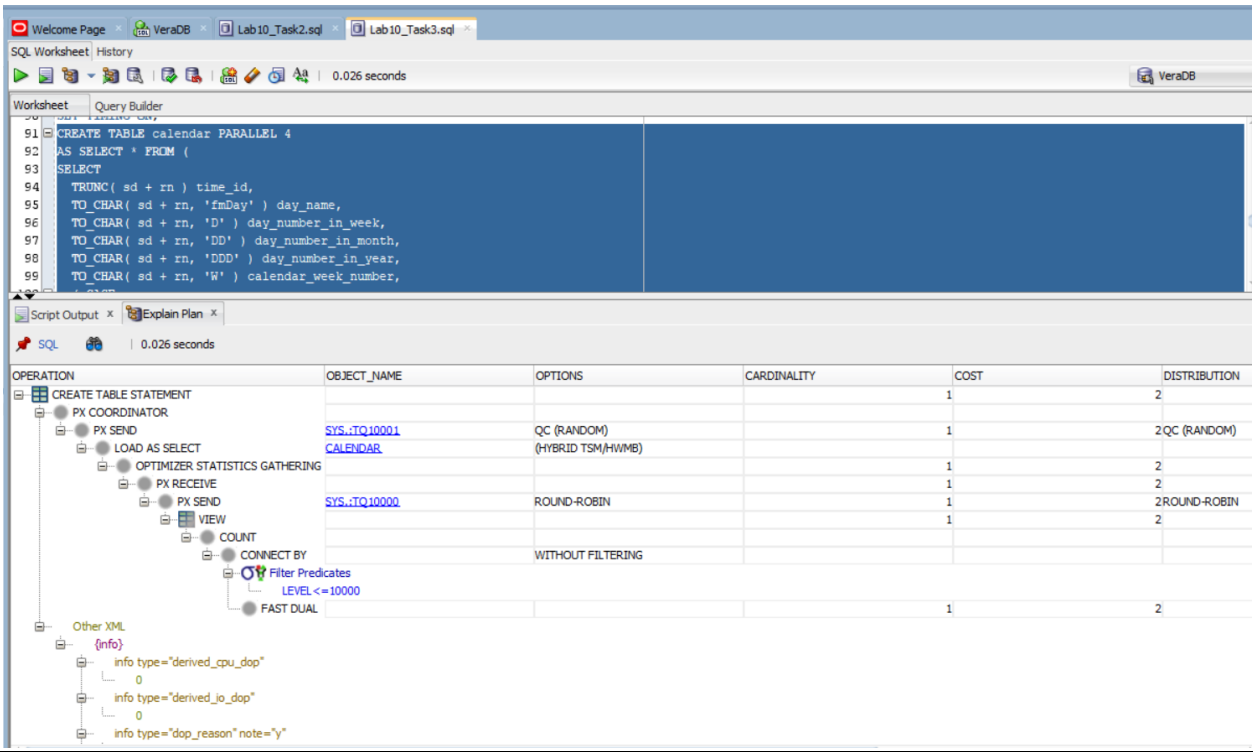
Script Output | Task completed in 0.345 seconds

```
Table CALENDAR created.

Elapsed: 00:00:00.330
```

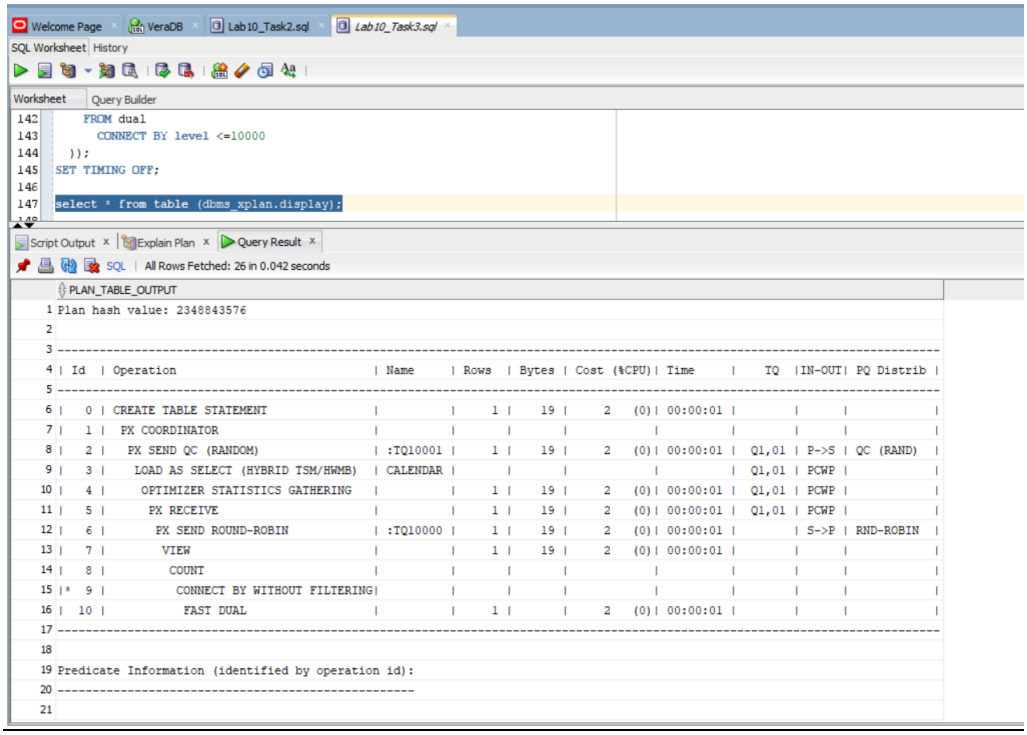
Create one table on 10 000 rows from another by adding a SELECT statement at the end of the CREATE TABLE statement with Parallel DDL.

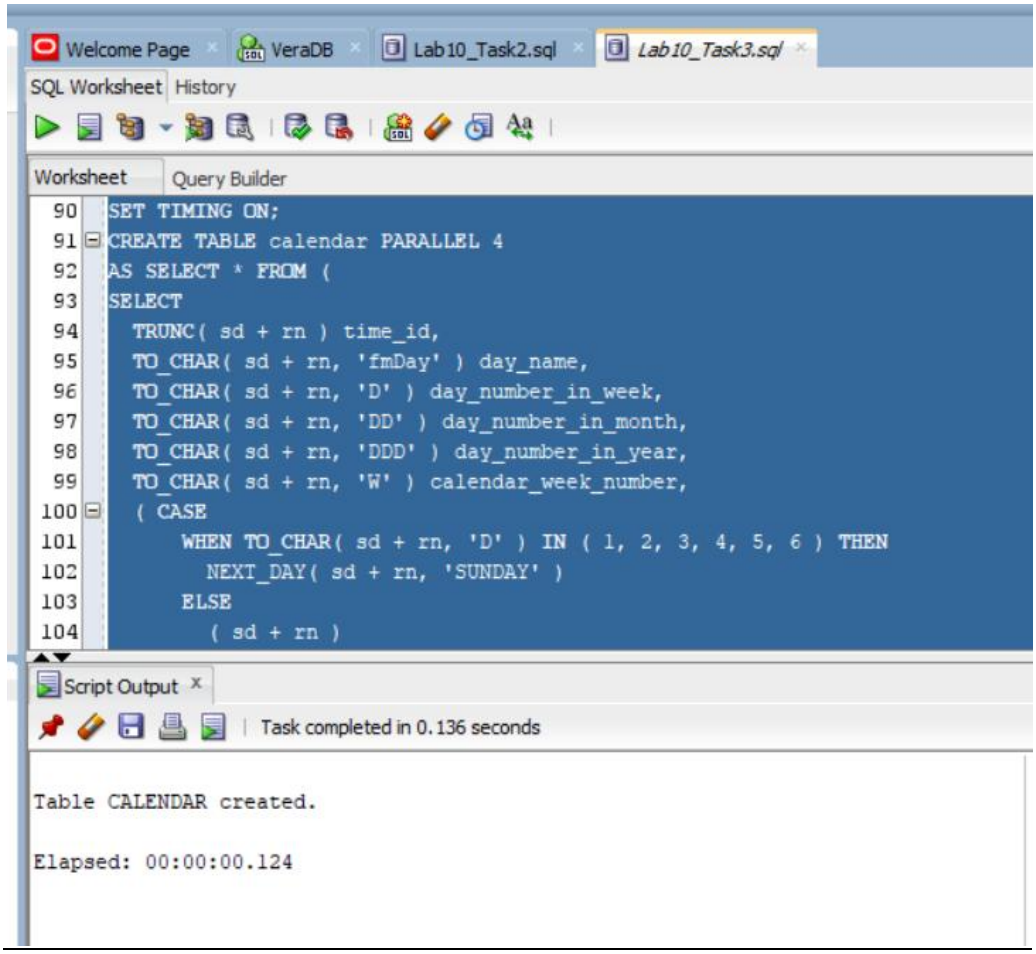


Consider Explain plan.



Look on the Elapsed time.





Summary:

	Number of rows deleted	Time to Execute Elapsed Time	Elapsed Time	Time to Simple CREATE TABLE …AS SELECT of Explain plane	Cost
Simple CREATE TABLE …AS SELECT	1000	0.102 seconds	00:00:00.088	00:00:01	4
Simple CREATE TABLE …AS SELECT with Parallel DDL	1000	0.092 seconds	00:00:00.079	00:00:01	2
Simple CREATE TABLE …AS SELECT	10 000	0.345 seconds	00:00:00.330	00:00:01	4
Simple CREATE TABLE …AS SELECT with Parallel DDL	10 000	0.136 seconds	00:00:00.124	00:00:01	2

Parallel DDL works for both tables and indexes, whether partitioned or nonpartitioned. For nonpartitioned tables and indexes, only the following types of DDL statements can be parallelized:

CREATE TABLE...AS SELECT
CREATE INDEX
ALTER INDEX...REBUILD
When working with partitioned tables and indexes, the scope of Oracle's parallel DDL support broadens. The following statements can be parallelized for partitioned tables and indexes:

CREATE TABLE...AS SELECT
ALTER TABLE...MOVE PARTITION
ALTER TABLE...SPLIT PARTITION
CREATE INDEX
ALTER INDEX...REBUILD PARTITION
ALTER INDEX...SPLIT PARTITION

Not all tables allow these operations to be executed in parallel. Tables with object columns or LOB columns don't allow parallel DDL.
Parallel execution lets you execute the query in parallel and create operations of creating a table as a subquery from another table or set of tables. This can be extremely useful in the creation of summary or rollup tables.
With DDL statements, you do not specify the degree of parallelism using hints. Instead, you use the PARALLEL clause.

In our case, when using parallel DDL, the cost and execution time of the operation decreased, indicating the optimization of the process.

3. Business Task - Parallel execution

3.1. Task 03: CREATE Strategy of Parallel execution

Strategy of Parallel refreshing Fact Tables and Dimension Tables.

Databases today contain a large amount of information. However, finding, updating and presenting the right information in a timely fashion can be a challenge because of the vast quantity of data involved.

Parallel execution is the capability that addresses this challenge. Using parallel execution (also called parallelism), terabytes of data can be processed in minutes, not hours or days, simply by using multiple processes to accomplish a single task. This dramatically reduces response time for data-intensive operations on large databases typically associated with decision support systems (DSS) and data warehouses.

Parallelism is the idea of breaking down a task so that, instead of one process doing all of the work in a query, many processes do part of the work at the same time.

When it becomes necessary to make changes to Fact Tables and Dimension Tables in the data warehouse, we can use parallel queries and parallel subqueries in SELECT statements. We can also parallelize the query portions of DDL statements and DML statements (INSERT, UPDATE, and DELETE). Parallel execution can accelerate large DML operations and is especially useful in data warehouse environments where large summary or history tables must be maintained.