Analyzing Query Rewrites of Materialized Views

Purpose

This module shows you how to use execution plans to make query rewrites with materialized views easier to interpret and use REWRITE_OR_ERROR hint to make query rewrite debugging easier.

Topics

This module will discuss the following topics:

Place the cursor on this icon to display all screenshots. You can also place the cursor on each icon to see only the screenshot associated with it.

Overview

Back to List

Using Explain Plans to Analyze Query Rewrites

Prior to Oracle Database 10g, external tables were read-only. In Oracle Database 10g, external tables can also be written to. Although neither data manipulation language (DML) operations nor index creation are allowed on an external table, it is possible to use the CREATE TABLE AS SELECT command to populate an external table composed of proprietary format (Direct Path API) flat files that are operating system independent.

In the context of external tables, loading data refers to the act of data being read from an external table and loaded into a table in the database. Unloading data refers to the act of reading data from a table in the database and inserting it into an external table. Both these operations can be used with external tables using the new Data Pump access driver.

REWRITE_OR_ERROR Hint

There may be situations where you want to stop the query from executing if it did not rewrite. One such situation can be when you expect the un-rewritten query to take an unacceptably long time to execute. In order to support this requirement, Oracle Database 10G provides a new hint called REWRITE_OR_ERROR. This is a query block level hint. For example, if the SELECT statement is not rewritten, the error message shown is thrown. This feature allows you to run DBMS_MVIEW.EXPLAIN_REWRITE() on the query, resolve the problems that caused rewrite to fail, and run the query again.

Prerequisites

Back to List

Before starting this module, you should have:

☐ **1.** Completed the Configuring Linux for the Installation of Oracle Database 10g lesson

☐ **2.** Completed the Installing the Oracle Database 10g on Linux lesson

☐ **3.** Completed the Postinstallation Tasks lesson.

☐ **4.** Download and unzip mvplans.zip into your working directory (i.e. /home/oracle/wkdir)

Generating Explain Plans and Interpreting the Results

Back to Topic List

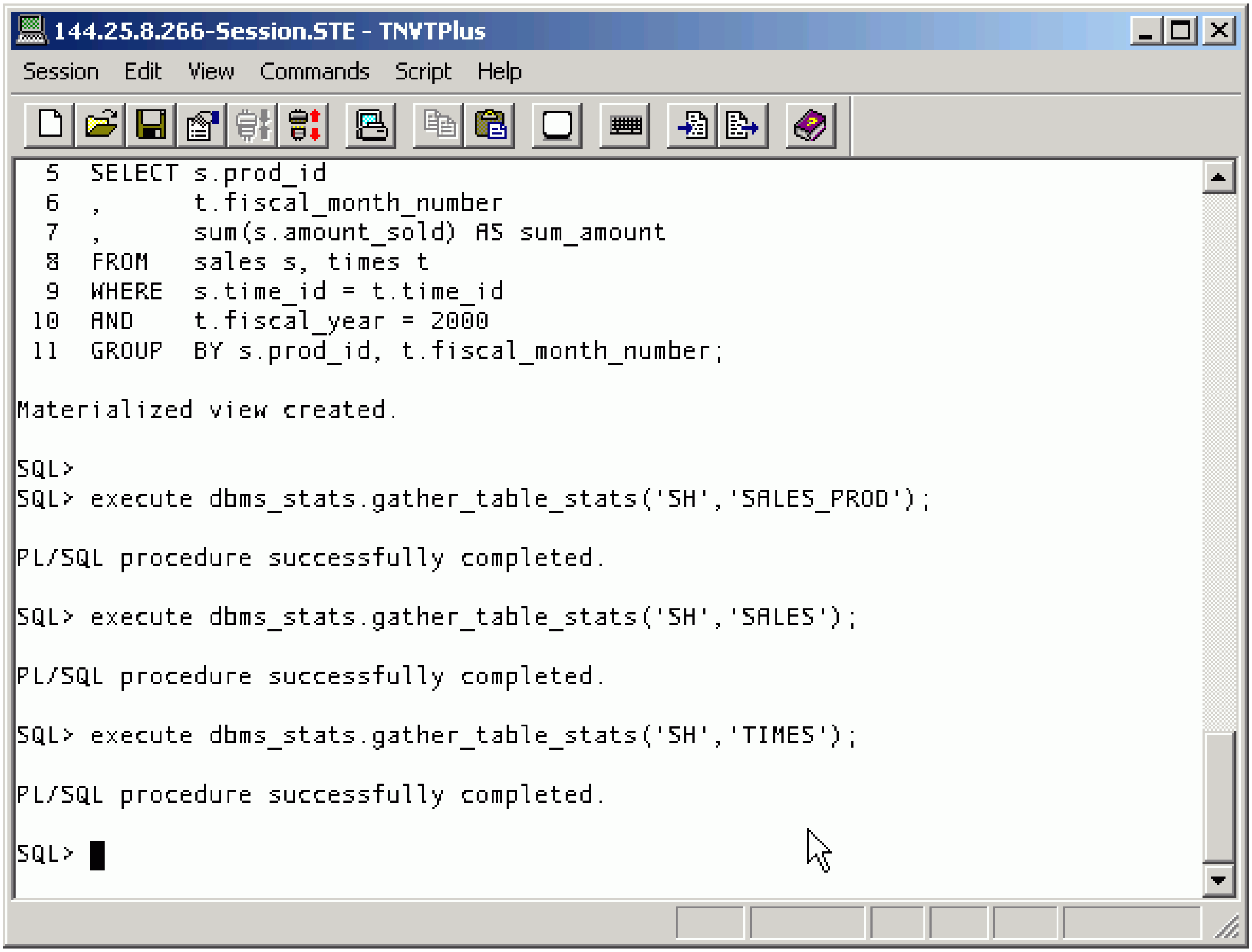
It is common practice to use naming conventions for MVs; for example, to distinguish MVs from regular tables in execution plans. Oracle Database 10g improves this situation by providing better information in the PLAN_TABLE and in the V$SQL_PLAN view; they show MATERIALIZED VIEW instead of TABLE. Moreover, they show the difference between MV usage as a result of query rewrite and direct MV access. Perform the following steps:

☐ **1.** You need to create the materialized view and gather statistics on the materialized view and its underlying tables. From your terminal window, execute the following commands:

```
cd wkdir
sqlplus sh/sh
@mvsetup
```

The query in the **`mvsetup.sql`** script is as follows:

```
drop materialized view sales_prod;

create materialized view sales_prod

       build immediate

       enable query rewrite

as

SELECT s.prod_id

,      t.fiscal_month_number
,       sum(s.amount_sold) AS sum_amount

FROM    sales s, times t

WHERE   s.time_id = t.time_id

AND     t.fiscal_year = 2000

GROUP   BY s.prod_id, t.fiscal_month_number;

execute dbms_stats.gather_table_stats('SH','SALES_PROD');

execute dbms_stats.gather_table_stats('SH','SALES');

execute dbms_stats.gather_table_stats('SH','TIMES');
```



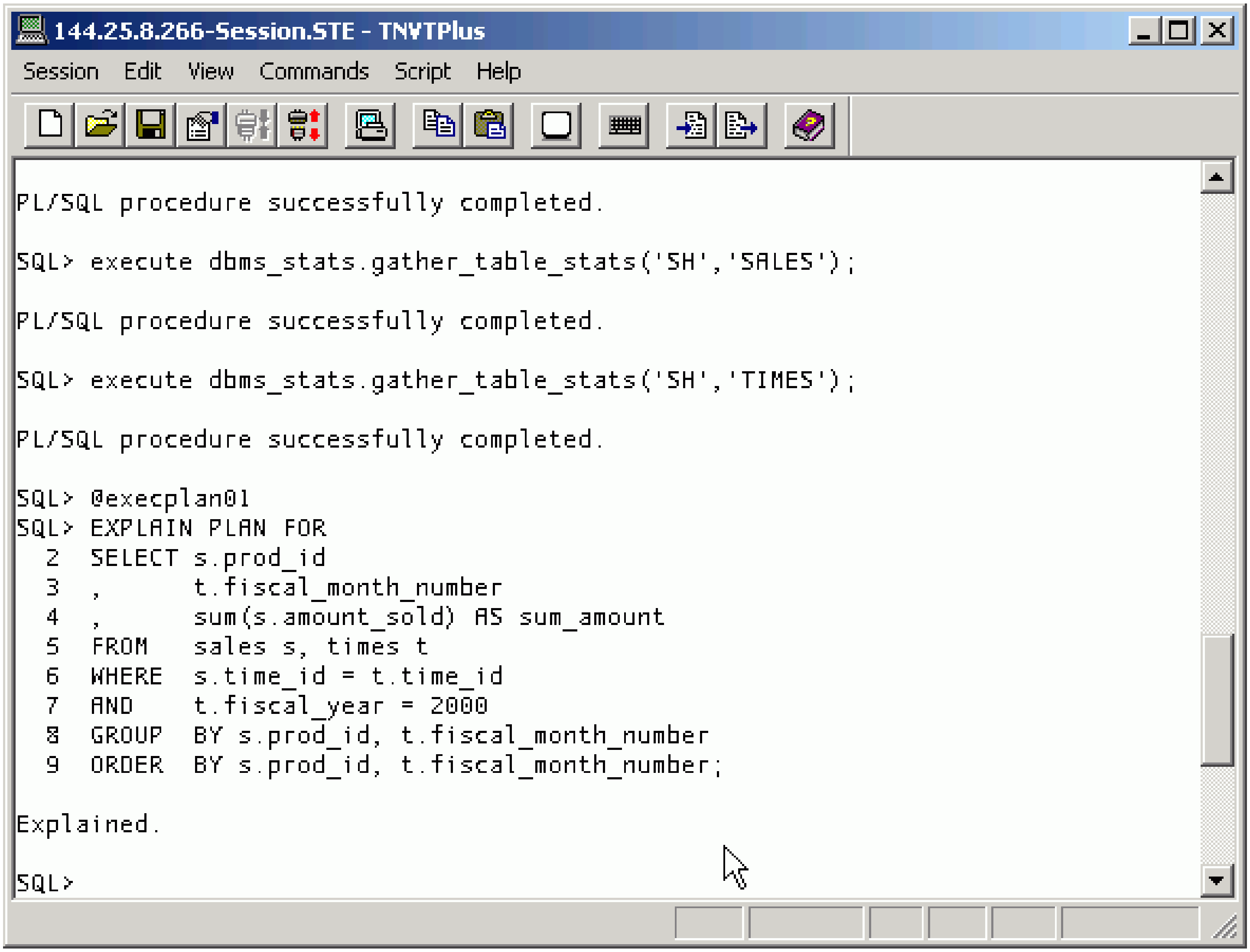
```
  5  SELECT s.prod_id
  6  ,      t.fiscal_month_number
  7  ,      sum(s.amount_sold) AS sum_amount
  8  FROM   sales s, times t
  9  WHERE  s.time_id = t.time_id
 10  AND    t.fiscal_year = 2000
 11  GROUP  BY s.prod_id, t.fiscal_month_number;

Materialized view created.

SQL>
SQL> execute dbms_stats.gather_table_stats('SH','SALES_PROD');

PL/SQL procedure successfully completed.

SQL> execute dbms_stats.gather_table_stats('SH','SALES');

PL/SQL procedure successfully completed.

SQL> execute dbms_stats.gather_table_stats('SH','TIMES');

PL/SQL procedure successfully completed.

SQL>
```

☐ **2.** Now you can generate an execution plan for a query that will be rewritten. Execute the following script from your terminal window:

```
@execplan01
```

The query in the **`execplan01.sql`** script is as follows:

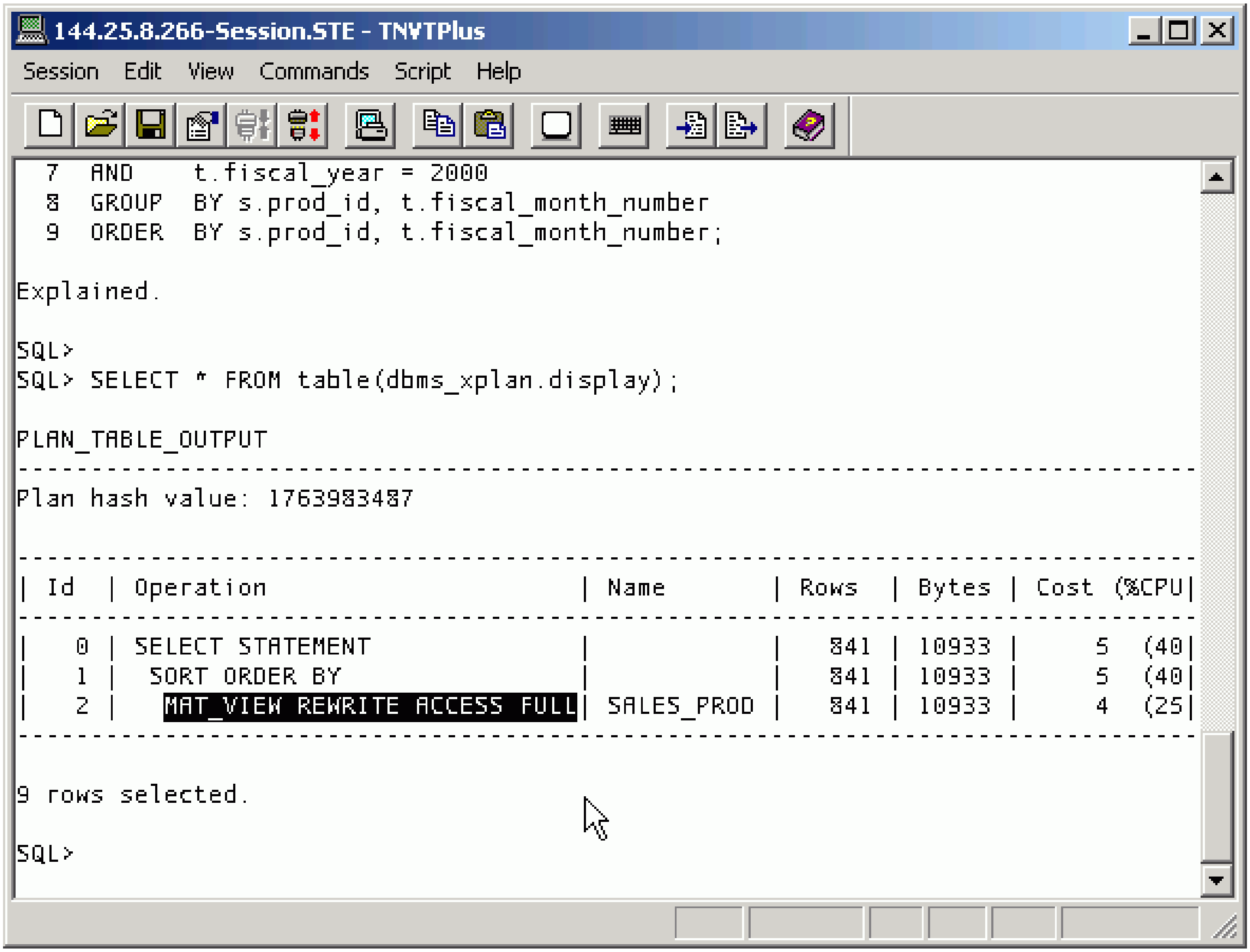
```
EXPLAIN PLAN FOR

SELECT s.prod_id

,      t.fiscal_month_number

,      sum(s.amount_sold) AS sum_amount

FROM   sales s, times t

WHERE  s.time_id = t.time_id

AND    t.fiscal_year = 2000

GROUP  BY s.prod_id, t.fiscal_month_number

ORDER  BY s.prod_id, t.fiscal_month_number;
```


144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
PL/SQL procedure successfully completed.

SQL> execute dbms_stats.gather_table_stats('SH','SALES');

PL/SQL procedure successfully completed.

SQL> execute dbms_stats.gather_table_stats('SH','TIMES');

PL/SQL procedure successfully completed.

SQL> @execplan01
SQL> EXPLAIN PLAN FOR
  2  SELECT s.prod_id
  3  ,      t.fiscal_month_number
  4  ,      sum(s.amount_sold) AS sum_amount
  5  FROM   sales s, times t
  6  WHERE  s.time_id = t.time_id
  7  AND    t.fiscal_year = 2000
  8  GROUP  BY s.prod_id, t.fiscal_month_number
  9  ORDER  BY s.prod_id, t.fiscal_month_number;

Explained.

SQL>
```

☐ **3.** Now that the execution plan is generated, you can use the DBMS_XPLAN package to display the execution plan. From your terminal window, execute the following command:

```
SELECT * FROM table(dbms_xplan.display);
```

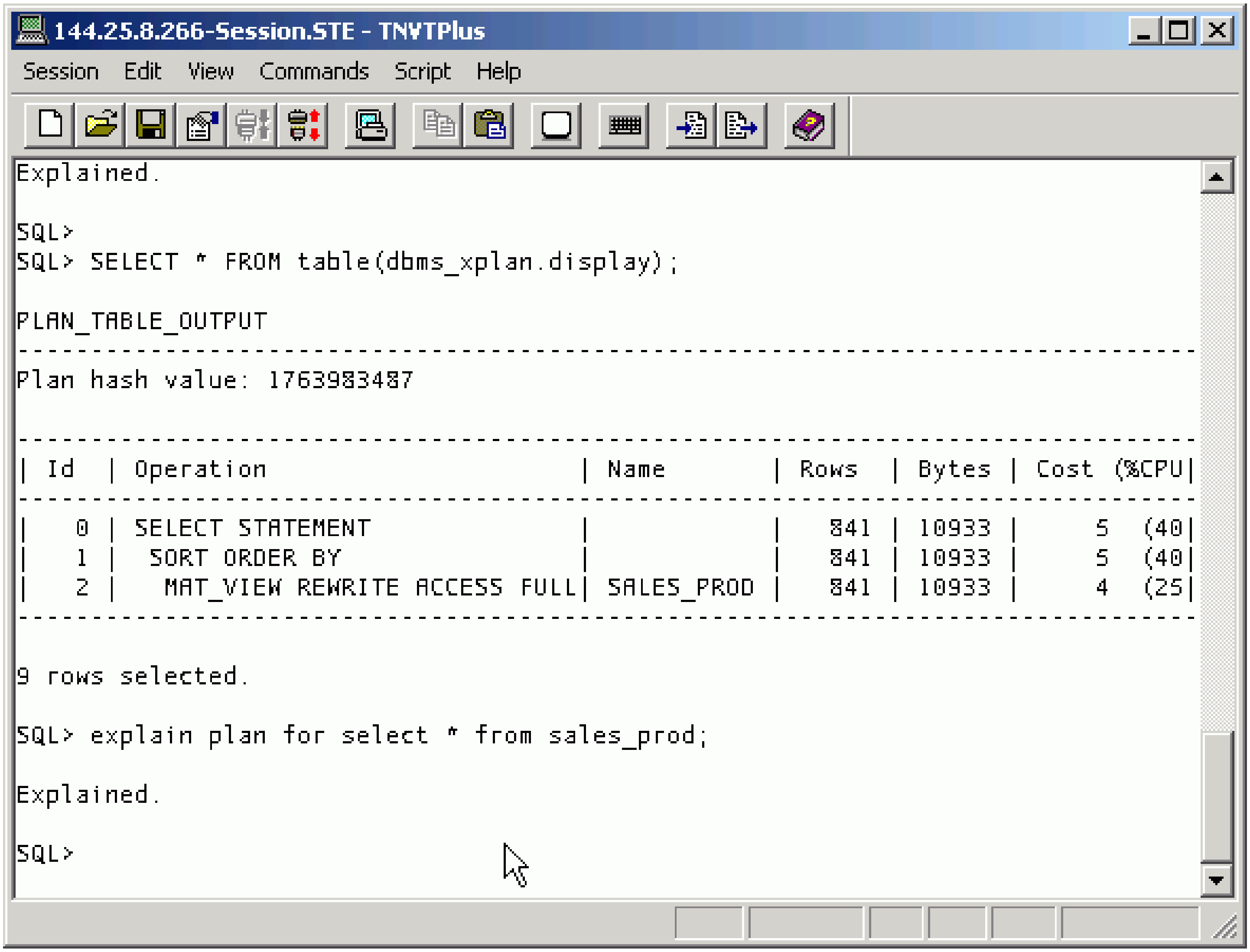

144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
  7  AND    t.fiscal_year = 2000
  8  GROUP  BY s.prod_id, t.fiscal_month_number
  9  ORDER  BY s.prod_id, t.fiscal_month_number;

Explained.

SQL>
SQL> SELECT * FROM table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1763983487

--------------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes | Cost  (%CPU|
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |   841 | 10933 |     5   (40|
|   1 |  SORT ORDER BY                |            |   841 | 10933 |     5   (40|
|   2 |   MAT_VIEW REWRITE ACCESS FULL| SALES_PROD |   841 | 10933 |     4   (25|
--------------------------------------------------------------------------------

9 rows selected.

SQL>
```

Note the execution plan explicitly shows materialized view usage and the purpose (REWRITE).

☐ **4.** To see what would happen if the materialized view was explicitly accessed in the FROM component of a query, execute the following command:

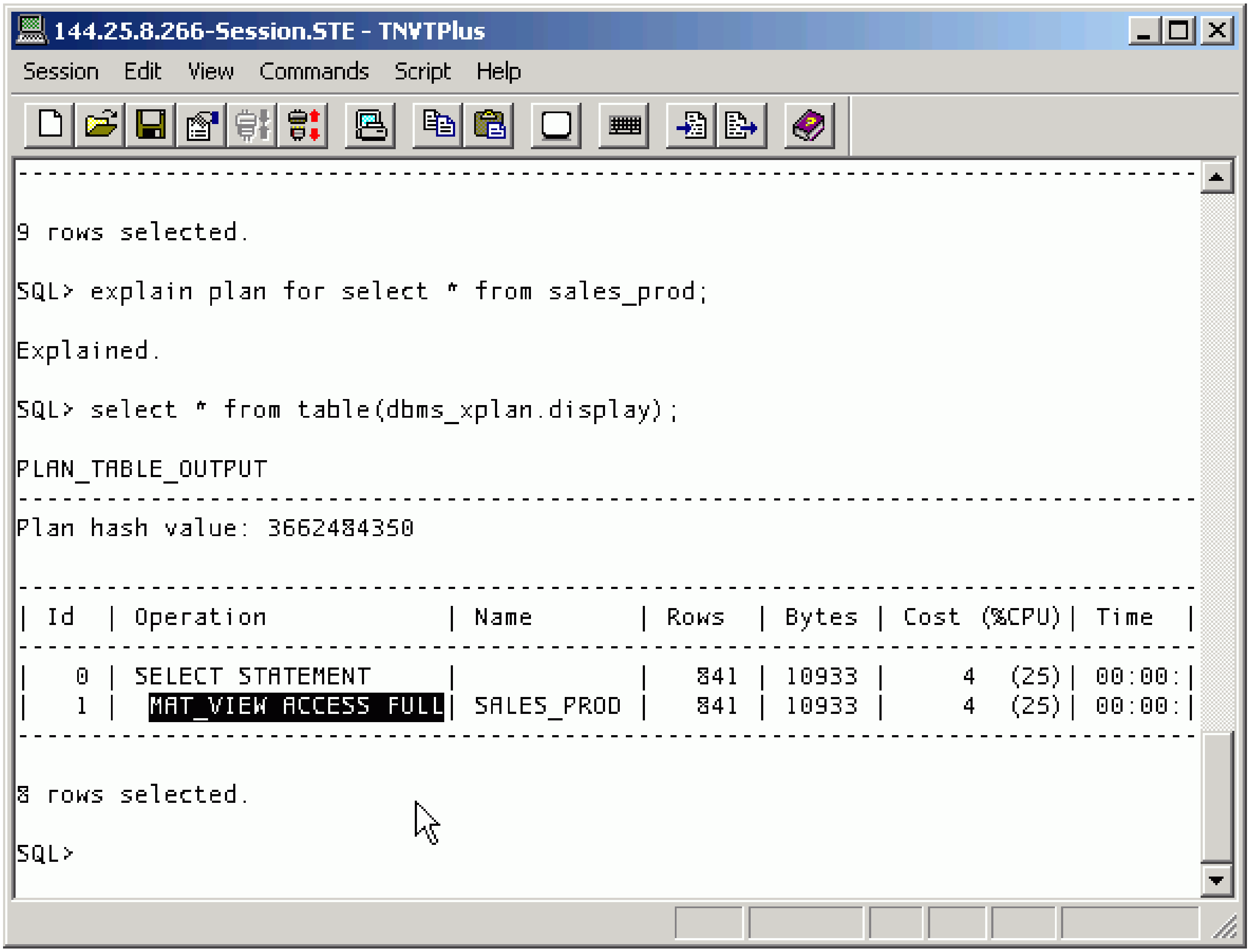
```
EXPLAIN PLAN FOR SELECT * FROM sales_prod;
```


144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
Explained.

SQL>
SQL> SELECT * FROM table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1763983487

--------------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes | Cost (%CPU|
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |   841 | 10933 |     5  (40|
|   1 |  SORT ORDER BY                |            |   841 | 10933 |     5  (40|
|   2 |   MAT_VIEW REWRITE ACCESS FULL| SALES_PROD |   841 | 10933 |     4  (25|
--------------------------------------------------------------------------------

9 rows selected.

SQL> explain plan for select * from sales_prod;

Explained.

SQL>
```

☐ **5.** Now you can display the execution plan again to see the difference. From your terminal window, execute the following command:

```
SELECT * FROM table(dbms_xplan.display);
```



```
--------------------------------------------------------------------------------

9 rows selected.

SQL> explain plan for select * from sales_prod;

Explained.

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 3662484350

--------------------------------------------------------------------------------
| Id  | Operation            | Name       | Rows  | Bytes | Cost (%CPU)| Time  |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |            |   841 | 10933 |     4  (25)| 00:00:|
|   1 |  MAT_VIEW ACCESS FULL| SALES_PROD |   841 | 10933 |     4  (25)| 00:00:|
--------------------------------------------------------------------------------

8 rows selected.

SQL>
```

Note the execution plan still shows MAT_VIEW access (as opposed to TABLE access) but the REWRITE is gone.

Using the REWRITE_OR_ERROR Hint

Back to Topic List

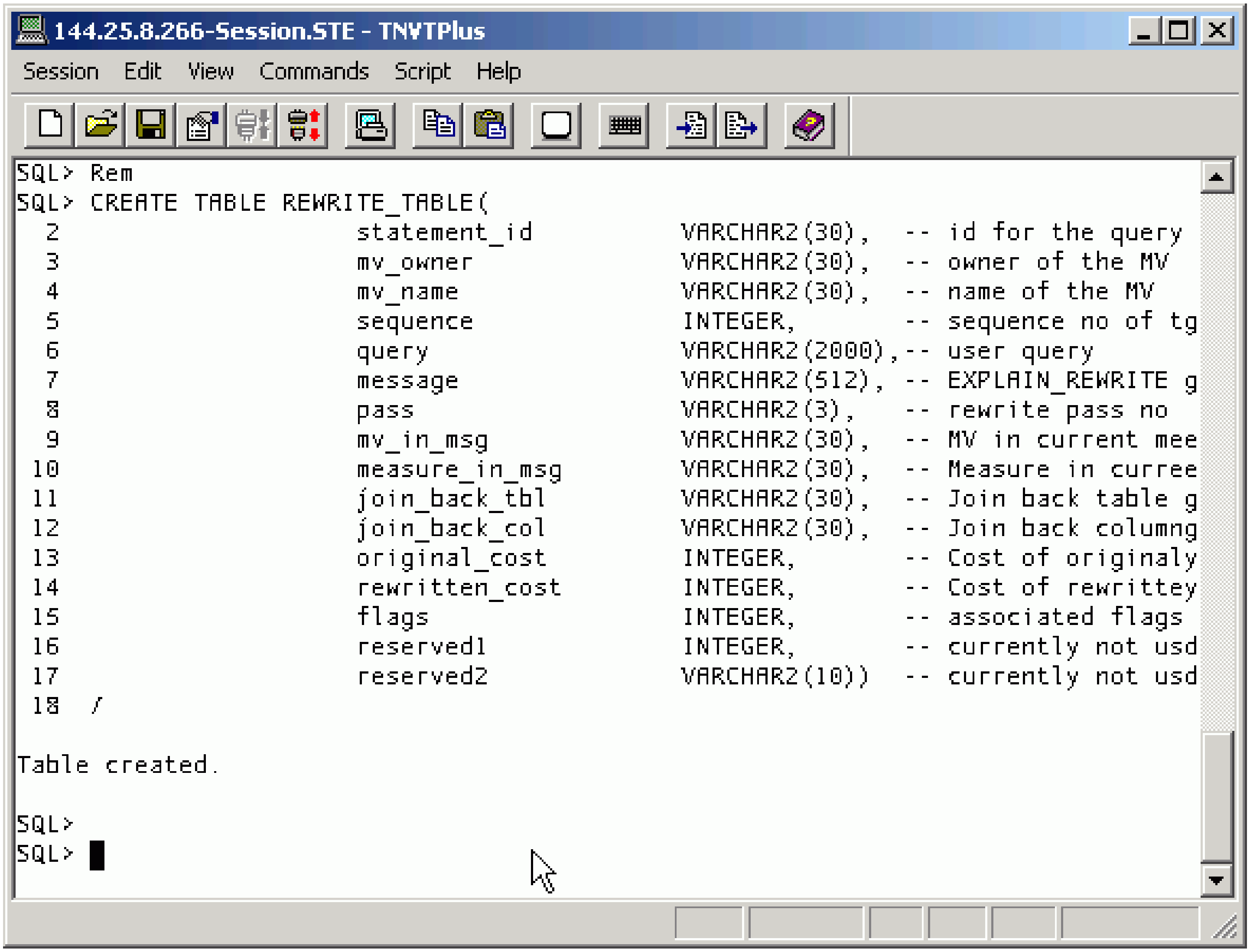
There may be situations where you want to stop a query from executing if it did not rewrite. One such situation is when you expect the un-rewritten query to take an unacceptably long time to execute. To support this requirement, Oracle Database 10g provides a new hint called REWRITE_OR_ERROR. If a query is not rewritten, error ORA-30393 is thrown. This feature allows you to run dbms_mview.EXPLAIN_REWRITE() on the query, resolve the problems that caused rewrite to fail, and run the query again. To obtain EXPLAIN_REWRITE output into a table, you must run the utlxrw.sql script before calling EXPLAIN_REWRITE. This script creates a table named REWRITE_TABLE in the current schema. Perform the following steps:

☐ **1.** Before you begin you need to recreate your rewrite table. The utlxrw.sql script creates the REWRITE table. You need this table to capture the output of the EXPLAIN_REWRITE procedure. From your terminal window, execute the following command:

```
crewrt01
```

The command in the `crewrt01 .sql` script is as follows:

```
drop table rewrite_table;

@$ORACLE_HOME/rdbms/admin/utlxrw
```



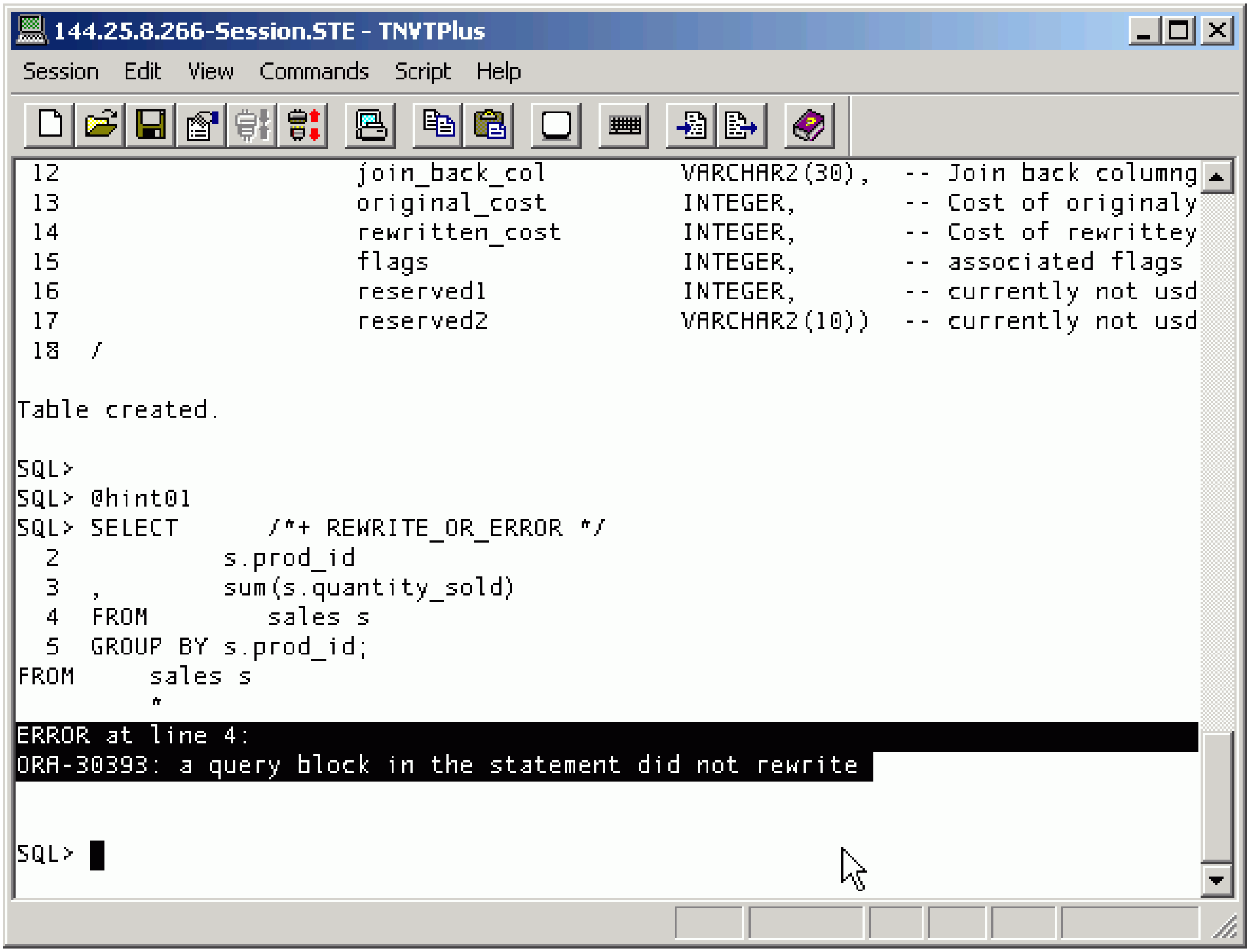
```
SQL> Rem
SQL> CREATE TABLE REWRITE_TABLE(
  2                        statement_id          VARCHAR2(30),   -- id for the query
  3                        mv_owner              VARCHAR2(30),   -- owner of the MV
  4                        mv_name               VARCHAR2(30),   -- name of the MV
  5                        sequence              INTEGER,        -- sequence no of tg
  6                        query                 VARCHAR2(2000),-- user query
  7                        message               VARCHAR2(512),  -- EXPLAIN_REWRITE g
  8                        pass                  VARCHAR2(3),    -- rewrite pass no
  9                        mv_in_msg             VARCHAR2(30),   -- MV in current mee
 10                        measure_in_msg        VARCHAR2(30),   -- Measure in curree
 11                        join_back_tbl         VARCHAR2(30),   -- Join back table g
 12                        join_back_col         VARCHAR2(30),   -- Join back columng
 13                        original_cost         INTEGER,        -- Cost of originaly
 14                        rewritten_cost        INTEGER,        -- Cost of rewrittey
 15                        flags                 INTEGER,        -- associated flags
 16                        reserved1             INTEGER,        -- currently not usd
 17                        reserved2             VARCHAR2(10))   -- currently not usd
 18  /

Table created.

SQL>
SQL>
```

☐ **2.** Normally when a query rewrite fails, the Oracle Database executes the statement against the underlying base tables. The REWRITE_OR_ERROR hint changes the behavior, and generates an error message when query rewrite fails. To use a hint in a query, execute the following command from your terminal window:

```
@hint01
```

The command in the **`hint01.sql`** script is as follows:

```
SELECT   /*+ REWRITE_OR_ERROR */
         s.prod_id
,        sum(s.quantity_sold)
FROM     sales s
GROUP BY s.prod_id;
```



```
 12                    join_back_col          VARCHAR2(30),    -- Join back columng
 13                    original_cost          INTEGER,         -- Cost of originaly
 14                    rewritten_cost         INTEGER,         -- Cost of rewrittey
 15                    flags                  INTEGER,         -- associated flags
 16                    reserved1              INTEGER,         -- currently not usd
 17                    reserved2              VARCHAR2(10))    -- currently not usd
 18   /

Table created.

SQL>
SQL> @hint01
SQL> SELECT      /*+ REWRITE_OR_ERROR */
  2           s.prod_id
  3   ,       sum(s.quantity_sold)
  4   FROM        sales s
  5   GROUP BY s.prod_id;
FROM      sales s
          *
ERROR at line 4:
ORA-30393: a query block in the statement did not rewrite


SQL> 
```

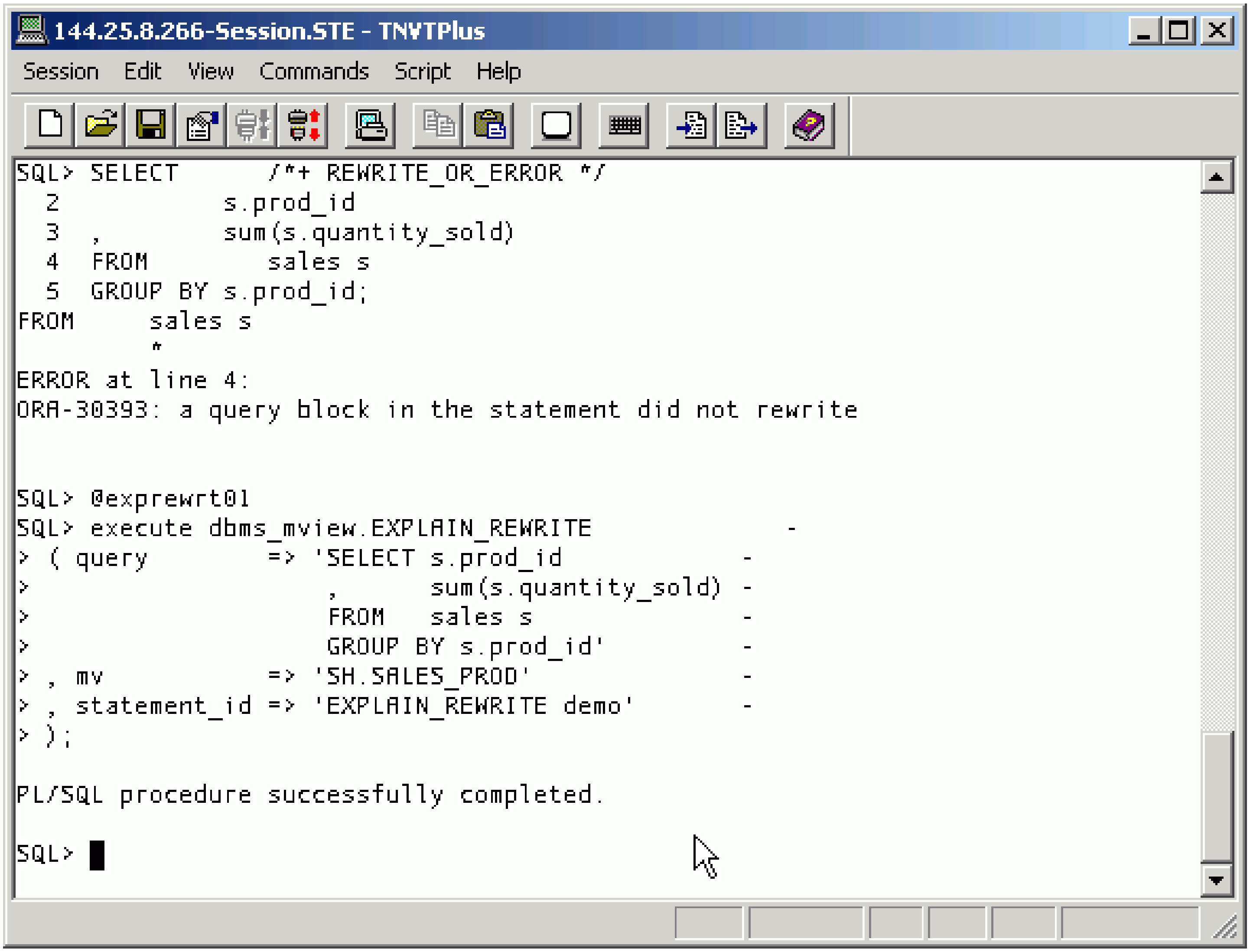
Note that an error message was generated.

☐ **3.** You can use the EXPLAIN_REWRITE procedure to find the reason why query rewrite failed. The results will be captured in the REWRITE_TABLE you created previously. Execute the following script:

`@exprewrt01`

The command in the **`exprewrt 01.sql`** script is as follows:

```
execute dbms_mview.EXPLAIN_REWRITE             -
( query        => 'SELECT s.prod_id            -
                   ,      sum(s.quantity_sold) -
                   FROM   sales s              -
                   GROUP BY s.prod_id'         -
, mv           => 'SH.SALES_PROD'              -
, statement_id => 'EXPLAIN_REWRITE demo'       -
);
```



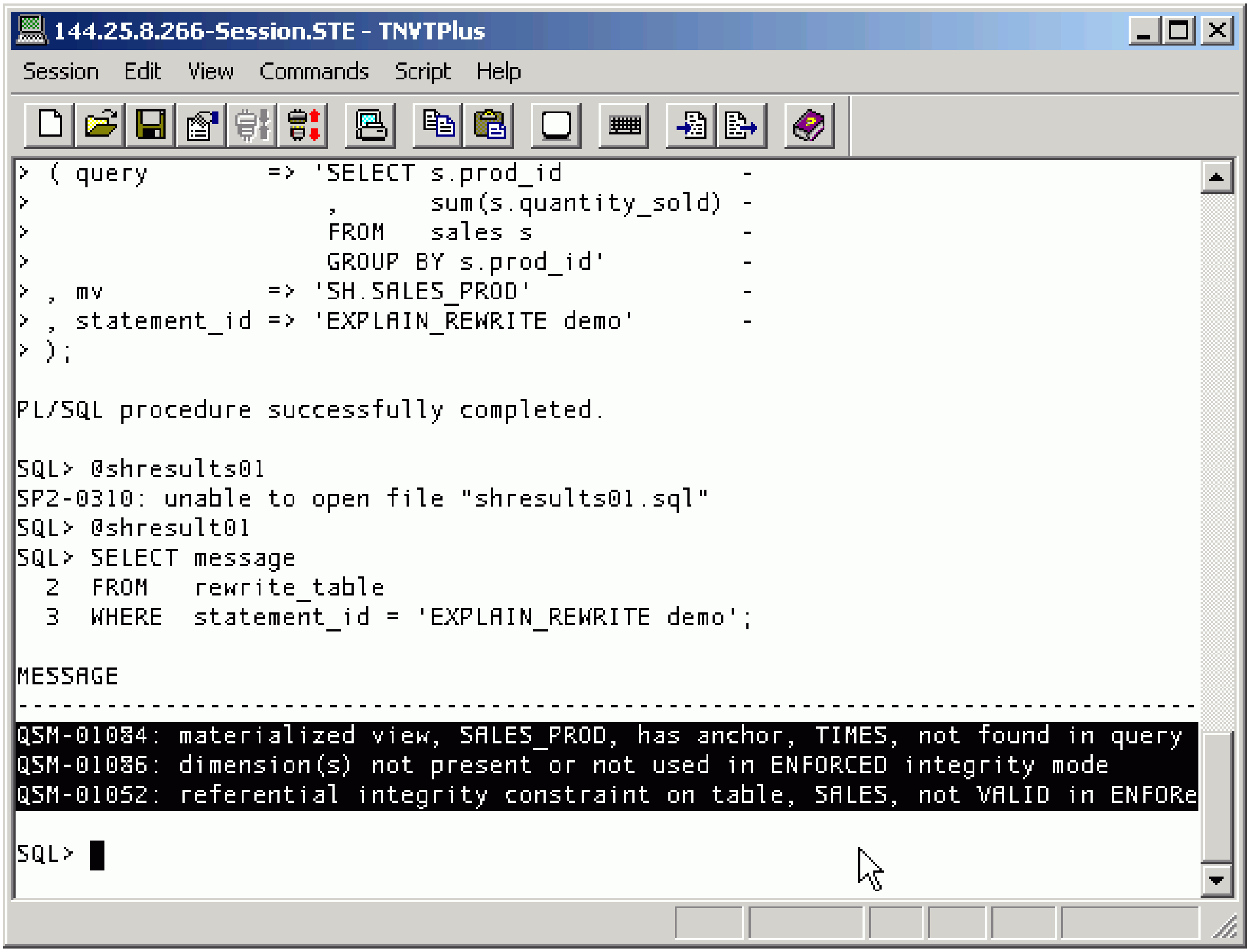
```
SQL> SELECT       /*+ REWRITE_OR_ERROR */
  2           s.prod_id
  3  ,        sum(s.quantity_sold)
  4  FROM        sales s
  5  GROUP BY s.prod_id;
FROM     sales s
         *
ERROR at line 4:
ORA-30393: a query block in the statement did not rewrite


SQL> @exprewrt01
SQL> execute dbms_mview.EXPLAIN_REWRITE             -
> ( query        => 'SELECT s.prod_id            -
>                    ,      sum(s.quantity_sold) -
>                    FROM   sales s              -
>                    GROUP BY s.prod_id'         -
> , mv           => 'SH.SALES_PROD'              -
> , statement_id => 'EXPLAIN_REWRITE demo'       -
> );

PL/SQL procedure successfully completed.

SQL>
```

☐ **4.** Now you can query the REWRITE_TABLE to query the results. Execute the following script:

```
@shresult01
```

The command in the **`shresult 01.sql`** script is as follows:

```
SELECT message
FROM   rewrite_table
WHERE  statement_id = 'EXPLAIN_REWRITE demo';
```



```
> ( query        => 'SELECT s.prod_id            -
>                    ,      sum(s.quantity_sold) -
>                    FROM   sales s              -
>                    GROUP BY s.prod_id'         -
> , mv           => 'SH.SALES_PROD'              -
> , statement_id => 'EXPLAIN_REWRITE demo'       -
> );

PL/SQL procedure successfully completed.

SQL> @shresults01
SP2-0310: unable to open file "shresults01.sql"
SQL> @shresult01
SQL> SELECT message
  2  FROM   rewrite_table
  3  WHERE  statement_id = 'EXPLAIN_REWRITE demo';

MESSAGE
--------------------------------------------------------------------------------
QSM-01084: materialized view, SALES_PROD, has anchor, TIMES, not found in query
QSM-01086: dimension(s) not present or not used in ENFORCED integrity mode
QSM-01052: referential integrity constraint on table, SALES, not VALID in ENFORe

SQL>
```

Note that there are three issues with the query.

☐ **5.** To see where the issues are, you need to look at the materialized view definition you created at the beginning of this lesson as follows:

```
CREATE

 MATERIALIZED VIEW sales_prod
  build immediate
  enable query rewrite
  as
     SELECT s.prod_id
          , t.fiscal_month_number
          , sum(s.amount_sold) AS sum_amount
     FROM sales s, times t
     WHERE s.time_id = t.time_id
     AND t.fiscal_year = 2000
     GROUP BY s.prod_id, t.fiscal_month_number;
```

Then you need to look at the query you executed as follows:

```
SELECT s.prod_id

     , sum(s.quantity_sold)
FROM sales s
GROUP BY s.prod_id;
```

As you can see, the materialized view utilizes the TIMES dimension which is not used in the query, and the materialized view aggregates AMOUNT_SOLD while the query is aggregating QUANTITY_SOLD. Therefore query rewrite is impossible.

Place the cursor on this icon to hide all screenshots.