

```

SQL>
SQL> set echo on
SQL> set feedback on
SQL> set linesize 300
SQL> set pagesize 500
SQL>
SQL> /* (1) First, the script processes the original SELECT statement create above. Use TIMING
option of SQLcl client
SQL>          to measure the total processing time. Setting and using TIMING option of SQLcl client
is described in "
SQL>          How to â€¦" Cookbook, Recipe 2.1, Step 2.
*/

```

```

SQL>
SQL> set timing on
SQL>
SQL> SELECT SUM(L_QUANTITY)
2  FROM LINEITEM JOIN ORDERS
3              ON L_ORDERKEY = O_ORDERKEY
4              JOIN CUSTOMER
5              ON O_CUSTKEY = C_CUSTKEY
6              JOIN NATION
7              ON C_NATIONKEY = N_NATIONKEY
8              WHERE N_NAME = 'INDIA';

```

```
SUM(L_QUANTITY)
```

```
-----
1794291
```

```
1 row selected.
```

```
Elapsed: 00:00:01.245
```

```

SQL>
SQL> set timing off
SQL>
SQL> /* (2) List query processing plan of the original SELECT statement
*/

```

```

SQL>
SQL> EXPLAIN PLAN FOR
2  SELECT SUM(L_QUANTITY)
3  FROM LINEITEM JOIN ORDERS
4              ON L_ORDERKEY = O_ORDERKEY
5              JOIN CUSTOMER
6              ON O_CUSTKEY = C_CUSTKEY
7              JOIN NATION
8              ON C_NATIONKEY = N_NATIONKEY
9              WHERE N_NAME = 'INDIA';

```

```
Explained.
```

```

SQL>
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

```

```
PLAN_TABLE_OUTPUT
```

```
-----
-----
-----
-----
Plan hash value: 2878459079
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	49	16663 (1)	00:00:01
1	SORT AGGREGATE		1	49		
* 2	HASH JOIN		72004	3445K	16663 (1)	00:00:01
* 3	HASH JOIN		18000	738K	3099 (1)	00:00:01
* 4	HASH JOIN		1800	59400	402 (1)	00:00:01

* 5	TABLE ACCESS FULL	NATION	1	27	12	(0)	00:00:01
* 6	TABLE ACCESS FULL	CUSTOMER	45000	263K	390	(1)	00:00:01
* 7	TABLE ACCESS FULL	ORDERS	450K	3955K	2696	(1)	00:00:01
8	TABLE ACCESS FULL	LINEITEM	1800K	12M	13559	(1)	00:00:01

Predicate Information (identified by operation id):

```

2 - access("L_ORDERKEY"="O_ORDERKEY")
3 - access("ORDERS"."O_CUSTKEY"="C_CUSTKEY")
4 - access("CUSTOMER"."C_NATIONKEY"="N_NATIONKEY")
5 - filter("NATION"."N_NAME"='INDIA')
6 - filter("CUSTOMER"."C_NATIONKEY">=0)
7 - filter("ORDERS"."O_CUSTKEY">=0)

```

Note

-----

- this is an adaptive plan

29 rows selected.

SQL>

SQL> /\* (3) Next, the script performs denormalization of a relational table that speeds up the processing of

SQL> a given SELECT statement in the best possible way. In this case, there is NO need for indexing and

SQL> there is no need for creation of materialized views or any additional relational tables.

SQL>

SQL> It is recommended to denormalize a conceptual schema given in a file tpchr.pdf before performing

SQL> any changes to the relational tables of TPC-HR database. There is no need to provide the outcomes

SQL> of denormalization of a conceptual schema.

\*/

SQL>

SQL> ALTER TABLE LINEITEM ADD L\_NATION CHAR(25) NULL;

Table altered.

SQL>

SQL> UPDATE LINEITEM

```

2 SET L_NATION = ( SELECT DISTINCT N_NAME
3                  FROM NATION JOIN CUSTOMER
4                      ON C_NATIONKEY = N_NATIONKEY
5                      JOIN ORDERS
6                      ON O_CUSTKEY = C_CUSTKEY
7                  WHERE O_ORDERKEY = L_ORDERKEY );

```

1800093 rows updated.

SQL>

SQL> /\* (4) Next, the script processes a new SELECT statement that accesses a denormalized relational table

SQL> and retrieves the same results as the original SELECT statement. Use TIMING option to measure

SQL> processing time. Note, that processing time should be shorter than processing time of the statement

SQL> is processed before denormalization.

\*/

SQL>

SQL> set timing on

SQL>

SQL> SELECT SUM(L\_QUANTITY)

```

2 FROM LINEITEM

```

```

3 WHERE L_NATION = 'INDIA';

```

SUM(L\_QUANTITY)

```
-----
1794291
```

```
1 row selected.
```

```
Elapsed: 00:00:01.073
```

```
SQL>
SQL> set timing off
SQL>
SQL> /* (5) List query processing plan of SELECT statement that access denormalized relational
table. */
```

```
SQL>
SQL> EXPLAIN PLAN FOR
  2 SELECT SUM(L_QUANTITY)
  3 FROM LINEITEM
  4 WHERE L_NATION = 'INDIA';
```

```
Explained.
```

```
SQL>
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
PLAN_TABLE_OUTPUT
```

```
-----
Plan hash value: 2287326370
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	29	13583 (1)	00:00:01
1	SORT AGGREGATE		1	29		
* 2	TABLE ACCESS FULL	LINEITEM	18001	509K	13583 (1)	00:00:01

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

```
2 - filter("L_NATION"='INDIA')
```

```
14 rows selected.
```

```
SQL>
SQL> /* (6) Next, the script creates an index to speed up processing of the new SELECT statement
in the best possible
SQL> way and again uses TIMING option to measure processing time. Note, that processing
time should be much
SQL> shorter than processing time in the previous step.
*/
SQL>
SQL> CREATE INDEX LINEITEM_IDX_NATION ON LINEITEM(L_NATION, L_QUANTITY);
```

```
Index created.
```

```
SQL>
SQL> set timing on
SQL>
SQL> SELECT SUM(L_QUANTITY)
  2 FROM LINEITEM
  3 WHERE L_NATION = 'INDIA';
```

```
SUM(L_QUANTITY)
```

```
-----
1794291
```

```
1 row selected.
```

```
Elapsed: 00:00:00.038
SQL>
SQL> set timing off
SQL>
SQL> /* (7) Again, list query processing plan of SELECT statement that access denormalized
relational table.          */
SQL>
SQL> EXPLAIN PLAN FOR
  2  SELECT SUM(L_QUANTITY)
  3  FROM LINEITEM
  4  WHERE L_NATION = 'INDIA';
```

Explained.

```
SQL>
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

PLAN\_TABLE\_OUTPUT

-----  
-----  
-----  
-----

Plan hash value: 2324175065

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	29	43 (0)	00:00:01
1	SORT AGGREGATE		1	29		
* 2	INDEX RANGE SCAN	LINEITEM_IDX_NATION	18001	509K	43 (0)	00:00:01

Predicate Information (identified by operation id):

-----

2 - access("L\_NATION"='INDIA')

14 rows selected.

```
SQL>
SQL> spool off
```