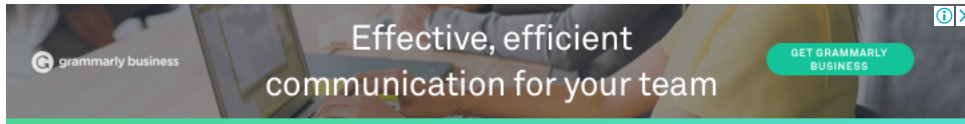


# ORACLE<sup>®</sup> DATABASE PL/SQL

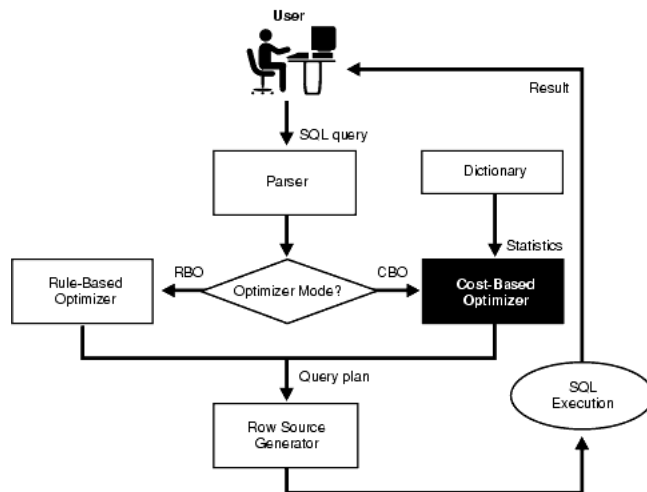
Select Language

Powered by  Google Translate

## ORACLE OPTIMIZER- RBO/CBO

### How OPTIMIZER works in ORACLE? What is RULE BASED and COST BASED OPTIMIZATION in ORACLE.

The optimizer determines the most efficient way to execute a SQL statement after considering many factors related to the objects referenced and the conditions specified in the query. This determination is an important step in the processing of any SQL statement and can greatly affect execution time.



IMG Source- <https://docs.oracle.com>

**RBO- Rule Based Optimizer** – Whenever we execute any SQL, the Optimizer will use execution plan based on some predefined rules. So, Query should be matched with the rules specified by RBO if not matched then it will result in Full Table scan. It works just like a Machine works on some rules and instructions.

The functionality is still present but no new functionality has been included in it and it is no longer supported by Oracle. It is only present to provide backwards compatibility during the migration to the query optimizer (Cost Based Optimizer).

**CBO- Cost Based Optimizer** – CBO uses Artificial Intelligence to decide the Execution Plan for SQL Query based on the Query Statistics. The cost-based method means the database must decide which query execution plan to choose using best guess approach that takes into account what data is stored in db. The Oracle cost-based optimizer is designed to determine the most efficient way to carry out a SQL statement, but it can't reach do this without good, up-to-date statistical information on the data being accessed.

With the cost-based approach, the optimizer factors in statistical information about the contents of the particular schema objects (tables, clusters, or indexes) being accessed.

Earlier, the only optimizer in the Oracle database was the Rule-Based Optimizer (RBO). Basically, the RBO used a set of rules to determine how to execute a query. If an index was available on a table, the RBO rules said to always use the index. There are some cases where the use of an index slowed down a query.

#### For Example:

There is an index on the GENDER column which holds one of two values MALE and FEMALE. Then someone issues the following query:  
 SELECT \* FROM Employee WHERE gender 'FEMALE';

If the above query returned approximately 50 of the rows, then using an index would actually slow things down. It would be faster to read the entire table and throw away all rows that have MALE values. Rule of thumb says if the number of rows returned is more than 5-10 of the total table volume using an index would slow things down. The RBO would always use an index if present because its rules said to.

#### Follow For More UPDATES

Ravikant Baluni


[google.com/+RavikantBaluni](https://google.com/+RavikantBaluni)

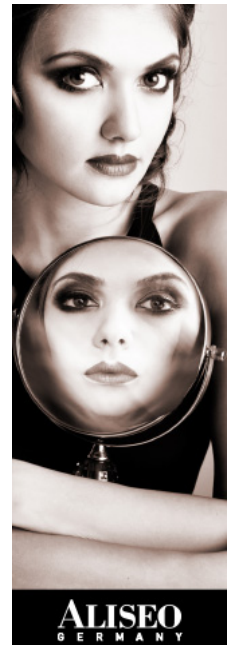
 Follow

104 followers

Ad closed by Google

Report this ad

Why this ad? 



#### ORACLE Topics

- COLLECTIONS and TYPES of COLLECTI
- Which Collection type should be used
- Examples of COLLECTIONS and COLLECTION Methods
- DATABASE Normalization Techniques
- PACKAGE Overloading
- Creating PACKAGES and Call it's Metho
- ORACLE 11g Features
- CONTINUE and CONTINUE WHEN Statement
- Passing parameters in Functions/Procedures

The biggest problem with the RBO was that it did not take the data distribution into account. So, the Cost-Based Optimizer (CBO) was came into picture. The CBO uses statistics about the table its indexes and the data distribution to make better informed decisions.

Now, assume that the company has employees that are 95 females and 5 males. If you query for females, then you do not want to use the index. If you query for males, then you would like to use the index. The CBO has information at hand to help make these kinds of determinations that were not available in the Rule Based Optimization.

SQL processing uses the following main components to execute a SQL query:

- The Parser checks both syntax and semantic analysis.
- The Optimizer uses costing methods, cost-based optimizer (CBO), or internal rules, rule-based optimizer (RBO), to determine the most efficient way of producing the result of the query.
- The Row Source Generator receives the optimal plan from the optimizer and outputs the execution plan for the SQL statement.
- The SQL Execution Engine operates on the execution plan associated with a SQL statement and then produces the results of the query.

A SQL statement can be executed in many different ways, including the following:

- Full table scans
- Index scans
- Nested loops
- Hash joins

#### Full Table Scans

This type of scan reads all rows from a table and filters out those that do not meet the selection criteria. During a full table scan, all blocks in the table that are under the high-water mark are scanned. Each row is examined to determine whether it satisfies the statement's WHERE clause.

When Oracle performs a full table scan, the blocks are read sequentially. Because the blocks are adjacent, I/O calls larger than a single block can be used to speed up the process.

#### Why a Full Table Scan Is Faster for Accessing Large Amounts of Data?

Full table scans are cheaper than index range scans when accessing a large fraction of the blocks in a table. This is because full table scans can use larger I/O calls, and making fewer large I/O calls is cheaper than making many smaller calls.

#### When the Optimizer Uses, Full Table Scans?

The optimizer uses a full table scan in any of the following cases:

- **Lack of Index or Indexed column used with Functions:** If the query is unable to use any existing indexes, then it uses a full table scan. For example, if there is a function used on the indexed column in the query, the optimizer is unable to use the index and instead uses a full table scan

```
SELECT FIRST_NAME, LAST_NAME FROM Employees
```

**WHERE UPPER(LAST\_NAME) LIKE :B1; /\*Even if there is an Index on LAST\_NAME it will not be used by ORACLE\*/**

- **Access Large Amount of Data:** If the optimizer thinks that the query will access most of the blocks in the table, then it uses a full table scan, even though indexes might be available.
- **High Degree of Parallelism:** A high degree of parallelism for a table skews the optimizer toward full table scans over range scans. Examine the DEGREE column in ALL\_TABLES for the table to determine the degree of parallelism.

#### Full Table Scan Hints

FULL hint in ORACLE is used if you want to force the use of a full table scan.

Without Using the FULL Hint

```
SQL> SELECT EMP_NO, ENAME
2 From EMPLOYEES
3 Where DEPTNO IN(10,20);
```

```
EMP_NO ENAME
-----
1001 Ravi
1002 Surya
1006 Rajan
1007 Manu
1008 Karan
```

Execution Plan

Plan hash value: 522316573

| Id  | Operation                   | Name      | Rows | Bytes | Cost (%CPU) | Time     |
|-----|-----------------------------|-----------|------|-------|-------------|----------|
| 0   | SELECT STATEMENT            |           | 3    | 42    | 2 (0)       | 00:00:01 |
| 1   | INLIST ITERATOR             |           |      |       |             |          |
| 2   | TABLE ACCESS BY INDEX ROWID | EMPLOYEES | 3    | 42    | 2 (0)       | 00:00:01 |
| * 3 | INDEX RANGE SCAN            | IND_DEP   | 3    |       | 1 (0)       | 00:00:01 |

Predicate Information (identified by operation id):

```
3 - access("DEPTNO"=10 OR "DEPTNO"=20)
```

Statistics

```
0 recursive calls
0 db block gets
```

- Stored Procedure Vs. Functions
- SQL Query Order Execution
- DWH(OLAP) Vs. Operational DB(OLTP)
- Data Migration Steps and SCD Change
- ROLLBACK behaviour when FOR ALL is used
- Handling BULK Exception using SAVE EXCEPTION
- BULK Collect with NATIVE Dynamic SC
- BULK Collect and Collection of Record
- Using BULK Collect and BULK Binds
- ORACLE Table Locking
- How to kill ORACLE Session?
- Handling PL/SQL Errors(Exception Handling)
- RAISE\_APPLICATION\_ERROR Built-IN Procedure
- Exception Trapping Functions
- WHERE and HAVING clause Alternatives
- TRIGGER and Types of TRIGGERS
- Identify Columns having all NULLS
- TABLE Vs. MATERIALIZED View
- VIEWS in ORACLE
- SYNONYMS in ORACLE
- How INDEXES stored in DB
- Local and Global Indexes
- CLUSTERED and NON-CLUSTERED Indexes
- INDEXES in ORACLE
- Opening Parameterized Cursor in Different ways
- Sub-Queries-And-Types-of-Sub-Query
- COMMIT inside Trigger
- Difference between Primary and Unique Key
- Difference between %TYPE Vs. %ROWTYPE
- WITH Clause in ORACLE
- DECODE Vs. CASE
- ROWNUM Vs. ROW\_NUMBER()
- ROWNUM Vs. ROWID
- INSERT and DELETE Execution Plan
- Different types of JOINS in ORACLE
- NOT IN Vs. NOT EXISTS Operator
- IN Vs. EXISTS Operator
- How Count Function behaves with different operators
- DELETE Vs. TRUNCATE Vs. DROP
- Find Highest/Minimum Salary and Employee Information
- Identify and Remove DUPLICATE Records
- MUTATING Table Error and How to Avoid It.
- GLOBAL TEMPORARY Tables in ORACLE
- CHAR-NCHAR-VARCHAR-VARCHAR2-NVARCHAR
- UNION Vs. UNION ALL(SET OPERATORS)
- How CURSOR works Internally?
- ORACLE Cursors and its Types
- Separate NUMERIC/NON-NUMERIC/DATE values From a Column
- ANALYTICAL Vs. AGGREGATE Function
- DBMS\_PROFILER Installation Steps
- How DBMS\_PROFILER helps in identifying long running SQL's
- ORACLE SQL Tuning Tips
- Dynamic Where Clause
- ORACLE SQL EXECUTION PLAN
- COLLECTION having NULL Values
- ORACLE SQL\* Loader
- ORACLE External Tables
- RULE BASED and COST BASED OPTIMIZER
- OPTIMIZER Modes in ORACLE

```

5 consistent gets
0 physical reads
0 redo size
702 bytes sent via SQL*Net to client
519 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
5 rows processed

```

#### After Using the FULL Hint

```

SQL> SELECT /*+ FULL(EMP) */ EMP_NO, ENAME
2 From EMPLOYEES EMP
3 Where DEPTNO IN(10,20);

```

```

EMP_NO ENAME
-----
1001 Ravi
1002 Surya
1006 Rajan
1007 Manu
1008 Karan

```

#### Execution Plan

Plan hash value: 1445457117

| Id  | Operation         | Name      | Rows | Bytes | Cost (%CPU) | Time     |
|-----|-------------------|-----------|------|-------|-------------|----------|
| 0   | SELECT STATEMENT  |           | 3    | 42    | 3 (0)       | 00:00:01 |
| * 1 | TABLE ACCESS FULL | EMPLOYEES | 3    | 42    | 3 (0)       | 00:00:01 |

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

1 - filter("DEPTNO"=10 OR "DEPTNO"=20)

#### Statistics

```

1 recursive calls
0 db block gets
4 consistent gets
0 physical reads
0 redo size
702 bytes sent via SQL*Net to client
519 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
5 rows processed

```

#### Index Scans

In Index Scans rows are accessed and retrieved using an Index on column values specified by the statement. An index scan retrieves data from an index based on the value of one or more columns in the index. To perform an index scan, Oracle searches the index for the indexed column values accessed by the statement.

#### Index Scan Hints

A hint might be required if the optimizer chooses some other index or uses a full table scan. The hint INDEX(table\_alias index\_name) specifies the index to use.

```

SQL> SELECT /*+ INDEX(EMP IND_DEP) */ EMP_NO, ENAME, SALARY, DEPTNO, DNAME
2 From EMPLOYEES EMP
3 Where DEPTNO IN(10,20,30);

```

```

EMP_NO ENAME      SALARY  DEPTNO DNAME
-----
1001 Ravi          48000    10 ACCOUNTS
1002 Surya          89000    20 SALES
1006 Rajan          77770    20 SALES
1007 Manu           98888    20 BANKING
1008 Karan          48000    20 BANKING
1003 Ankit          89000    30 ACCOUNTS

```

6 rows selected.

#### Execution Plan

Plan hash value: 522316573

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|-----------|------|------|-------|-------------|------|
|----|-----------|------|------|-------|-------------|------|

- ORACLE Driving Tables
- EXECUTION PLAN and It's Components
- CURSOR\_SHARING in ORACLE
- INDEX Usage with LIKE Operator and DOMAIN Index
- DYNAMIC\_SAMPLING and its Impact on OPTIMIZER
- NOT NULL and Indexed Column
- ORACLE AUTOTRACE Utility
- Pass COMMA Separated Value to IN Operator
- ANALYTICAL & AGGREGATE Functions Examples
- ORACLE UTL\_FILE Package
- ORACLE UTL\_FILE Exceptions
- UTL\_FILE Operations and Functions
- Comma Separated Values
- RETURNING Table From a Function
- REGULAR Expressions in ORACLE
- RESTRICT DROP/TRUNCATE on TABLE
- Export Table Data to CSV
- IMPORT Data from Flat Files to ORACLE Tables
- ORACLE PIVOT/UNPIVOT
- PARTITIONING IN ORACLE
- Equality Test of Two COLLECTION Type
- Compare and Merge COLLECTION Objects
- NESTED Table Functions
- DETERMINISTIC FUNCTIONS
- HANDLING CURSOR Exceptions
- When CROSS JOIN Will Be Useful?
- DML Error Logging
- Handle CONCURRENT Updates
- Pessimistic and Optimistic Oracle Locks
- Returning REF CURSOR From a Procedure
- Prevent VALUE\_ERROR Exception
- SYS\_REFCURSOR Vs. REF CURSOR

Ad closed by Google

Report this ad

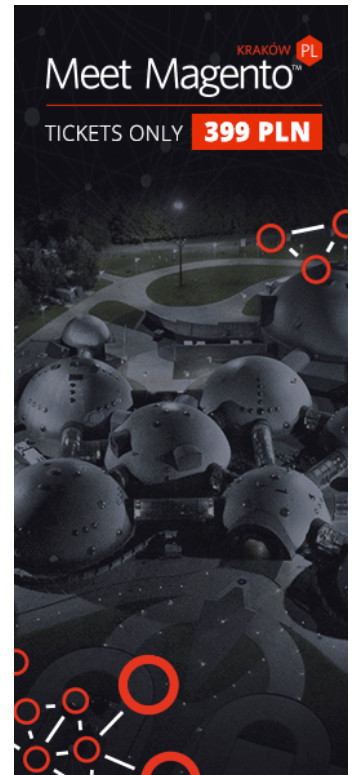
Why this ad? ⓘ



#### Popular Posts

- Analytic Functions Vs. Aggregate Functions

- ORACLE COLLECTIONS
- SYS\_REFCURSOR Vs. REF CURSOR
- ORACLE SQL\* Loader
- EXPORT TABLE Data to Flat Files
- ORACLE TABLE PARTITIONING
- UTL\_FILE Import Data into ORACLE TA
- ORACLE PL/SQL Tuning
- ORACLE UTL\_FILE
- Using BULK Collect



|     |                             |           |   |     |   |     |          |
|-----|-----------------------------|-----------|---|-----|---|-----|----------|
| 0   | SELECT STATEMENT            |           | 5 | 135 | 2 | (0) | 00:00:01 |
| 1   | INLIST ITERATOR             |           |   |     |   |     |          |
| 2   | TABLE ACCESS BY INDEX ROWID | EMPLOYEES | 5 | 135 | 2 | (0) | 00:00:01 |
| * 3 | INDEX RANGE SCAN            | IND_DEP   | 5 |     | 1 | (0) | 00:00:01 |

Predicate Information (identified by operation id):

3 - access("DEPTNO"=10 OR "DEPTNO"=20 OR "DEPTNO"=30)

Statistics

```

1 recursive calls
0 db block gets
5 consistent gets
0 physical reads
0 redo size
1072 bytes sent via SQL*Net to client
519 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
6 rows processed

```

[Check another example of INDEX Range Scan Descending:](#)

An index range scan descending is identical to an index range scan, except that the data is returned in descending order. Indexes, by default, are stored in ascending order. Usually, this scan is used when ordering data in a descending order to return the most recent data first, or when seeking a value less than a specified value.

```

SQL> SELECT /*+ INDEX_DESC(EMP IND_EMPNO) */ EMP_NO, ENAME, SALARY, DEPTNO, DNAME
2 From EMPLOYEES EMP
3 Where EMP_NO<1006 Order By EMP_NO DESC;

```

| EMP_NO | ENAME   | SALARY | DEPTNO | DNAME    |
|--------|---------|--------|--------|----------|
| 1005   | Pritesh | 48000  | 50     | ACCOUNTS |
| 1004   | Nikhil  | 77770  | 40     | SALES    |
| 1003   | Ankit   | 89000  | 30     | ACCOUNTS |
| 1002   | Surya   | 89000  | 20     | SALES    |
| 1001   | Ravi    | 48000  | 10     | ACCOUNTS |

Execution Plan

Plan hash value: 507161224

| Id  | Operation                   | Name      | Rows | Bytes | Cost (%CPU) | Time     |
|-----|-----------------------------|-----------|------|-------|-------------|----------|
| 0   | SELECT STATEMENT            |           | 6    | 162   | 2 (0)       | 00:00:01 |
| 1   | TABLE ACCESS BY INDEX ROWID | EMPLOYEES | 6    | 162   | 2 (0)       | 00:00:01 |
| * 2 | INDEX RANGE SCAN DESCENDING | IND_EMPNO | 6    |       | 1 (0)       | 00:00:01 |

Predicate Information (identified by operation id):

2 - access("EMP\_NO"<1006)

Statistics

```

1 recursive calls
0 db block gets
4 consistent gets
0 physical reads
0 redo size
1037 bytes sent via SQL*Net to client
519 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)

```

#### [When the Optimizer Uses, Nested Loop Joins](#)

The optimizer uses nested loop joins when joining small number of rows, with a good driving condition between the two tables. You drive from the outer loop to the inner loop, so the order of tables in the execution plan is important.

The outer loop is the driving row source. It produces a set of rows for driving the join condition. The row source can be a table accessed using an index scan or a full table scan. Also, the rows can be produced from any other operation. For example, the output from a nested loop join can be used as a row source for another nested loop join.

So, when joining table A and B (A is driving table, B is the probed table), then a nested loop join can take 1st row from A and perform a lookup to B using that value (of the column(s) you join by). Then nested loop takes the next row from A and performs another lookup to table B using

the new value. And so on and so on and so on.

### Nested Loop Join Hints

If the optimizer is choosing to use some other join method, you can use the USE\_NL(table1 table2) hint, where table1 and table2 are the aliases of the tables being joined.

The USE\_NL hint causes Oracle to join each specified table to another row source with a nested loop join, using the specified table as the inner table.

```
SELECT /*+ USE_NL(E,D) */ E.EMP_NO, E.ENAME, D.DEPTNAME
  From EMP_TEST E, DEPT_TEST D
```

```
Where E.DEPTNO=D.DEPTNO;
```

### Hash Joins

Hash joins are used for joining large data sets. The optimizer uses the smaller of two tables or data sources to build a hash table on the join key in memory. It then scans the larger table, probing the hash table to find the joined rows.

This method is best used when the smaller table fits in available memory. The cost is then limited to a single read pass over the data for the two tables.

Essentially, a hash join is a technique whereby Oracle loads the rows from the driving table (the smallest table, first after the where clause). Oracle then uses a hashing technique to locate the rows in the larger second table. A hash join is often combined with parallel query in cases where both tables are very large.

### When the Optimizer Uses, Hash Joins

The optimizer uses a hash join to join two tables if they are joined using an equijoin and if either of the following conditions are true:

- A large amount of data needs to be joined.
- A large fraction of the table needs to be joined.

### Hash Join Hints

Apply the USE\_HASH hint to advise the optimizer to use a hash join when joining two tables together. If you are having trouble getting the optimizer to use hash joins, investigate the values for the HASH\_AREA\_SIZE and HASH\_JOIN\_ENABLED parameters.

```
SQL> SELECT /*+ Use_Hash(E,D) */ E.EMP_NO, E.ENAME, D.DEPTNAME
  2  From EMP_TEST E, DEPT_TEST D
  3  Where E.DEPTNO=D.DEPTNO;
```

| EMP_NO | ENAME   | DEPTNAME  |
|--------|---------|-----------|
| 1001   | Ravi    | Accounts  |
| 1002   | Surya   | Retail    |
| 1003   | Ankit   | Insurance |
| 1004   | Nikhil  | Banking   |
| 1005   | Pritesh | Cloud     |
| 1006   | Rajan   | Retail    |
| 1007   | Manu    | Retail    |
| 1008   | Karan   | Retail    |

8 rows selected.

#### Execution Plan

Plan hash value: 2815500728

| Id  | Operation         | Name      | Rows | Bytes | Cost (%CPU) | Time     |
|-----|-------------------|-----------|------|-------|-------------|----------|
| 0   | SELECT STATEMENT  |           | 8    | 200   | 7 (15)      | 00:00:01 |
| * 1 | <b>HASH JOIN</b>  |           | 8    | 200   | 7 (15)      | 00:00:01 |
| 2   | TABLE ACCESS FULL | DEPT_TEST | 5    | 55    | 3 (0)       | 00:00:01 |
| 3   | TABLE ACCESS FULL | EMP_TEST  | 8    | 112   | 3 (0)       | 00:00:01 |

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

1 - access("E"."DEPTNO"="D"."DEPTNO")

#### Statistics

```

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

Hash joins are often faster than nested loop joins, especially in cases where the driving table is filtered into a small number of rows in the query's where clause.

In general, nested loop joins work best when there are useful indexes and the percentage of data returned is small. Hash joins work best when there are no useful indexes or when the join will return a large percentage of rows. It is usually better to avoid hints and let Oracle decide how to build an execution plan.

Get involved and leave your Comments in the Box Below. The more people get involved, the more we all benefit. So, leave your thoughts before you leave the page.



#### 4 comments:



**Vicky ETL** February 15, 2017 at 8:07 PM

Could you pls explain me when optimizer go for RBO. Becoz as per my knowlege after oracle 9i there is only CBO optimizer , please correct me if I m wrong.

Reply

Replies



**Ravikant Baluni** February 16, 2017 at 5:24 AM

Yes, I have already written it under RBO section.

---

Reply



**Vicky ETL** February 15, 2017 at 8:13 PM

On what basis we decide Driving and driven table  
for example  
select e.\*,d.\* from emp e , dept d where e.deptno=d.deptno  
which table is driving and driven table.  
Could you please explain me and also why ?

Reply

Replies



**Ravikant Baluni** February 16, 2017 at 8:37 AM

here you go...  
<https://tipsfororacle.blogspot.in/2017/02/oracle-driving-tables.html>

---

Reply

Enter your comment...

Comment as:

Google Accour ▼

Publish

Preview

#### Links to this post

Create a Link

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

