Query Execution Plans

1. Tools to Show Execution Plans

An execution plan is a list of steps that Oracle will follow in order to execute an SQL statement. Each step is one of a finite number of basic operations known to the database server. Even the most complex SQL statement can be broken down into a series of basic operations

There are two ways to produce an execution plan in SQL.

1. Autotrace
2. Explain Plan

1.1. Autotrace

The syntax of Autotrace is:

Set Autot[race] {Off | On | Trace[only]} [Exp[lain]] [Stat[istics]]

As an example, we will use the following five tables:

```
SQL> Select * From Tab;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
CUSTOMER1                      TABLE
INVENTORY1                     TABLE
ITEM1                          TABLE
ORDER1                         TABLE
ORDER_INV1                     TABLE
```

To start Autotrace,

SQL> Set Autotrace on;

Once the Autotrace is set, you can enter any SQL query. In addition to the query results, you will also see some statistical data and query execution plan.

```
SQL> Select ItemID, ItemDesc From Item1;

    ITEMID ITEMDESC
---------- ------------------------------
       894 Women's Hiking Shorts
       897 Women's Fleece Pullovers
       995 Children's Beachcomber Sandals
       559 Men's Expedition Parka
       786 3-Season Jacket
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 3354656151

--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     5 |   140 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| ITEM1|     5 |   140 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------------

Note
-----
   - dynamic sampling used for this statement

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          8  consistent gets
          0  physical reads
          0  redo size
        410  bytes sent via SQL*Net to client
        234  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          5  rows processed
```

If you want to see the Execution Plan only, then you need to use the "Traceonly Explain" options in Set Autotrace:

```
SQL> Set Autotrace Traceonly Explain;
SQL> Select ItemID, ItemDesc From Item1;

Execution Plan
----------------------------------------------------------
Plan hash value: 3354656151

--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     5 |   140 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| ITEM1|     5 |   140 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------------

Note
-----
   - dynamic sampling used for this statement
```

The most important elements in the Execution Plan are: *Operation*, *Name*, and *Rows*. The Operation (and Name) indicates the operation on the table name, whereas Rows indicate number of rows being processed. The other columns (e.g. Bytes, CPU Cost, and

Time) are trivial in our examples, because the number of records being processed is small and consequently the processing time is very fast (e.g. 1 second or less).

The execution order of operations is the most indented operation is executed first. Therefore, in the above example, TABLE ACCESS FULL will be executed first, and then followed by SELECT STATEMENT.

TABLE ACCESS FULL operation is to read all records from the table name stated in the Name column, which in this case is table Item1. Since table Item1 has 5 records, the Row column in the execution plan shows Rows=5.

SELECT STATEMENT operation is basically a projection operation, which is displaying the results of the SQL query. In this example, all the 5 rows read by the TABLE ACCESS FULL operation are all being displayed as the query results.

If your SQL*Plus does not show the execution plan in a tabular format like the above, you might see a result something like this:

```
SQL> Select ItemID, ItemDesc From Item1;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=3 Card=5 Bytes=140)

   1    0   TABLE ACCESS (FULL) OF 'ITEM1' (TABLE) (Cost=3 Card=5 Bytes=140)
```

It is likely that the version of the SQL*Plus is not the recent one. If you see the format execution plan like the above, it is suggested that you either upgrade the SQL*Plus to a latest version, or use the Explain Plan option (which will be discussed next).

1.2. Explain Plan

The syntax of Explain Plan is:

Explain Plan For [SQL Query]

For example:

```
SQL> Select ItemID, ItemDesc From Item1;

    ITEMID ITEMDESC
---------- ------------------------------
       894 Women's Hiking Shorts
       897 Women's Fleece Pullovers
       995 Children's Beachcomber Sandals
       559 Men's Expedition Parka
       786 3-Season Jacket
```

```
SQL> Explain Plan For Select ItemID, ItemDesc From Item1;

Explained.

SQL> Select * From Table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 1595666565

------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |     5 |   140 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| ITEM1 |     5 |   140 |     3   (0)| 00:00:01 |
------------------------------------------------------------------------------
```

When the Explain Plan command is used, the Select/From SQL will not display the query results; instead it only says "Explained". In order to see the Execution Plan of the Explain Plan, we need to use

Select * From Table(dbms_xplan.display);

The execution plan displayed by the Select * From Table(dbms_xplan.display) will identical to that of the Set Autotrace, shown previously.

1.3. Using SQL Developer

If you are using SQL Developer, you could use either the Explain Plan method:

```
EXPLAIN PLAN FOR
SELECT <attribute(s)>
FROM <tablename>
WHERE <conditions>;

/*
Once you have run those queries to view the execution plan run
the following query
*/

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

OR click the icon highlighted by the red box.

2. Execution Plans of SQL Basic Operations

In this section, we are going to learn more basic operations, covering

a. Order By,
b. Distinct,
c. Group By,
d. Group By and Order By
e. Distinct and Order By, and
f. Join.

2.1. Order By

The Order By operation is basically a Sort operation.

```
SQL> Select *
  2  From Customer1
  3  Order By LName;

  CUSTID LNAME           FNAME           ADDRESS            PHONE    CITY
-------- --------------- --------------- ------------------ -------- ---------
     181 Jane            Adam            229 Clayton Road   9543877  Clayton
     183 Judy            Backhouse       122 Rose Street    9235345  Caulfield
     179 Narayan         Ramesh          975 Fire Road      9456738  Carlton
     107 Smith           John            731 Plenty Road    9231455  Clayton
     154 Wallace         Jennifer        291 Berry Street   9234536  Preston
     232 Wong            Franklin        638 Voss Street    9756945  Preston
     133 Zelaya          Alicia          3321 Castle Ave    9867055  Balwyn

7 rows selected.
```

The execution plan for the Order By query is as follows (Note that in the examples below, Explain Plan is used. However, you could use Set Autotrace, instead):

```
SQL> Explain Plan For Select * from Customer1 Order By LName;

Explained.


SQL> Select * From Table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 1687277296

------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |     7 |   343 |     4  (25)| 00:00:01 |
|   1 |  SORT ORDER BY     |           |     7 |   343 |     4  (25)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| CUSTOMER1 |     7 |   343 |     3   (0)| 00:00:01 |
------------------------------------------------------------------------------

9 rows selected.
```

It first reads all the records (7 records) from table Customer1, and then sorts the 7 records, and finally displays the 7 records.

The SORT ORDER BY operation is a sorting operation, which in this example, the query sorts the records based on the attribute specified by the Order By clause in the SQL (i.e. LName).

2.2. Distinct

The Distinct operation is a duplicate removal operation. The following example shows that out of the 10 records in table Order1, there are only two payment methods (e.g. Card and Cheque).

```
SQL> Select * From Order1;

   ORDERID  ORDERDATE  PAYMETHOD       ORDERSOURCE     CUSTID
----------  ---------  --------------  --------------  ------
      1057  20/FEB/06  CARD            WEB SITE           107
      1058  03/MAR/06  CARD            PHONE              232
      1059  12/MAR/06  CHEQUE          WEB SITE           133
      1060  20/MAR/06  CHEQUE          WEB SITE           133
      1061  10/APR/06  CARD            FAX                179
      1062  01/APR/06  CARD            FAX                179
      1063  07/SEP/06  CARD            WEB SITE           154
      1064  14/JUL/06  CARD            WEB SITE           154
      1065  30/NOV/06  CARD            PHONE              179
      1066  20/JAN/06  CHEQUE          WEB SITE           179

10 rows selected.

SQL> Select Distinct Paymethod From Order1;

PAYMETHOD
--------------------
CARD
CHEQUE
```

The execution plan of the query with a Distinct clause is as follows:

```
SQL> Explain Plan For Select Distinct Paymethod From Order1;

Explained.


SQL> Select * From Table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 2865907199
```

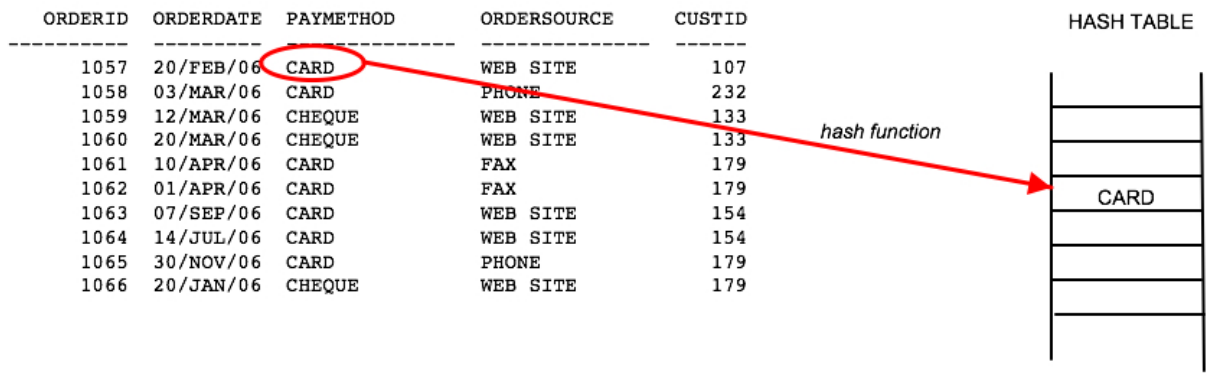
```
-------------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |     2 |    12 |     4  (25)| 00:00:01|
|   1 |  HASH UNIQUE       |        |     2 |    12 |     4  (25)| 00:00:01|
|   2 |   TABLE ACCESS FULL| ORDER1 |    10 |    60 |     3   (0)| 00:00:01|
-------------------------------------------------------------------------------
```

The execution plan shows that first it reads 10 records from table Order1 (TABLE ACCESS FULL), followed by a HASH UNIQUE operation of 2 records, and then displays the two records (SELECT STATEMENT). In order words, the Distinct clause is implemented by the HASH UNIQUE operation.

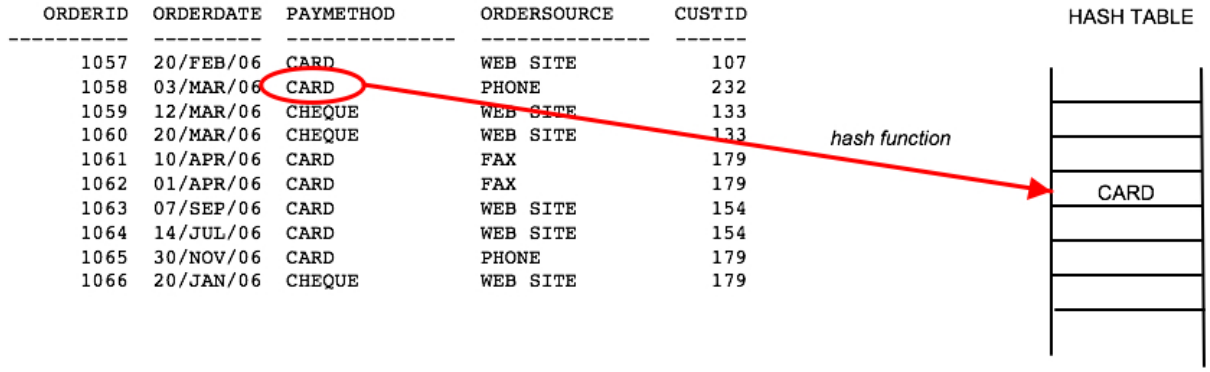
The HASH UNIQUE operation basically reads the record one by one, and then hashes it (using a particular hash function) to the Hash Table. If a record *B* is hashed to an entry in the Hash Table that is already occupied (by record *A*, for example), record *B* will be ignored – this means that record *B* is identical to any of the previous record (e.g. record *A*), because record *B* is hashed to the same hash entry of any of the previous record. Consequently, record *B* is duplicated and will not be added to the Hash Table. At the end, the Hash Table will contain only unique records.

The following picture illustrates how the HASH UNIQUE operation works.

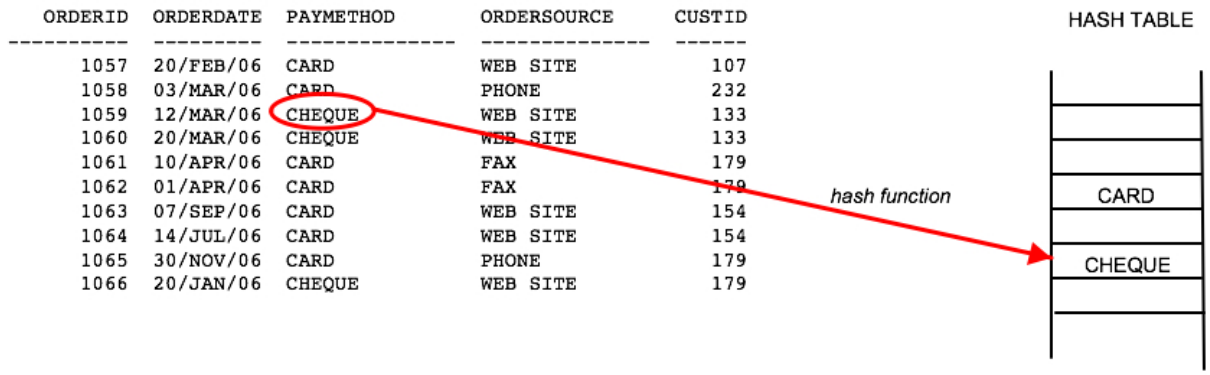
Hash the first record



Hash the second record



Hash the third record



2.3. Group By

The Group By operation is similar to the Distinct operation. The Distinct operation removes duplicates, and hence gets unique values only, whereas the Group By operation groups records based on certain attribute(s) as well as obtaining aggregate values for each group.

The grouping operation in the Group By can be thought of removing duplicates in the Distinct operation. The difference between Group By and Distinct is that in the Group By, it also calculates an aggregate value for each group.

The following example counts number of orders for each payment method.

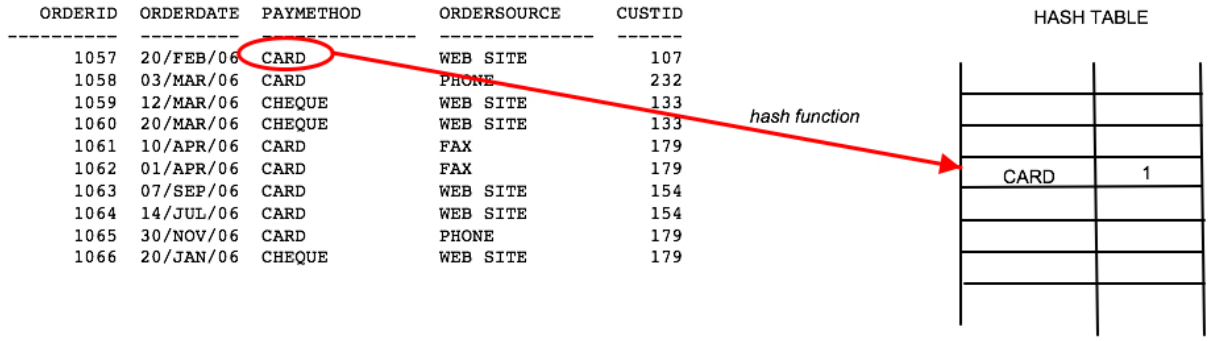
```
SQL> Select Paymethod, Count(*)
  2  From Order1
  3  Group By Paymethod;

PAYMETHOD              COUNT(*)
-------------------- ----------
CARD                          7
CHEQUE                        3

SQL> Explain Plan For
  2  Select Paymethod, Count(*)
  3  From Order1
  4  Group By Paymethod;

Explained.

SQL> Select * From Table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 3024968047

-------------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |     2 |    12 |     4  (25)| 00:00:01|
|   1 |  HASH GROUP BY     |        |     2 |    12 |     4  (25)| 00:00:01|
|   2 |   TABLE ACCESS FULL| ORDER1 |    10 |    60 |     3   (0)| 00:00:01|
-------------------------------------------------------------------------------
```

The above execution plan shows that first it reads the 10 records in table Order1 (TABLE ACCESS FULL). Then, it performs HASH GROUP BY using a hash table, which produces 2 records (e.g. 2 group by records), and then it displays the results (SELECT STATEMENT).

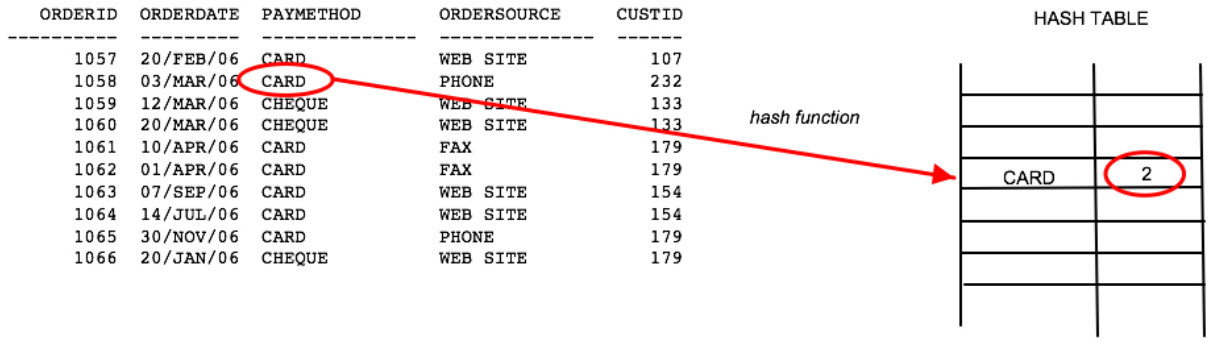
The HASH GROUP BY operation reads each record from table Order1 and hashes the record using a hash function to the Hash Table. In the Hash Table, apart from the key attribute, which is the group by attribute (e.g. Paymethod attribute in the above example), it also keeps track the aggregate value (e.g. count(*) in the above example).

Like that in HASH UNIQUE, using the HASH GROUP BY operation, when a record is hashed to the same hash entry in the Hash Table, the record will be ignored, but the aggregate value will be updated. For example, when the second Card record is hashed, it clashes with the previous Card record which already exists in the Hash Table. But the aggregate value (e.g. count(*)) will be updated (e.g. incremented by 1). The following figure illustrates how the HASH GROUP BY operation works.

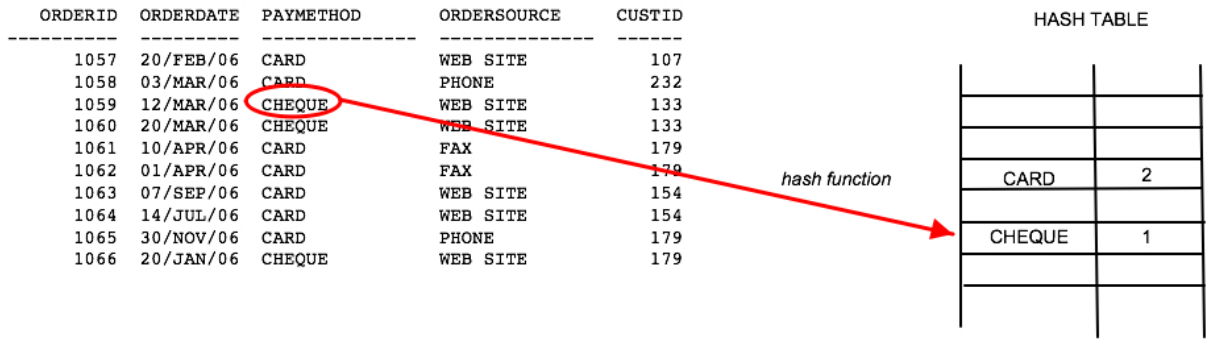
Hash the first record



Hash the second record



Hash the third record



Another example of Group By query is as follows. It uses the Sum aggregate function, and the grouping is based on InvID attribute. In this example, there are 7 different InvID, and for each InvID, it sums the Quantity.

```
SQL> Select * From Order_inv1;

   ORDERID      INVID ORDERPRICE   QUANTITY
---------- ---------- ---------- ----------
      1057      11668        260         10
      1058      11668        240         20
      1059      11780         22          5
      1060      11776         21         50
      1061      11779         30         25
      1061      11780         30         50
      1062      11669        230         40
      1063      11778         26         50
      1064      11779         30         12
      1065      11780         27         32
      1066      11775         30         30

11 rows selected.

SQL> Select invid, Sum(quantity) From Order_inv1 Group By invid;

     INVID SUM(QUANTITY)
---------- -------------
     11668            30
     11776            50
     11779            37
     11669            40
     11778            50
     11780            87
     11775            30

7 rows selected.
```

The execution plan of the above query is as follows:

```
SQL> explain plan for
  2  select invid, sum(quantity) from order_inv1 group by invid;

Explained.


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

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 157645868

------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |     7 |    56 |     4  (25)| 00:00:01|
|   1 |  HASH GROUP BY     |            |     7 |    56 |     4  (25)| 00:00:01|
|   2 |   TABLE ACCESS FULL| ORDER_INV1 |    11 |    88 |     3   (0)| 00:00:01|
------------------------------------------------------------------------------
```

The HASH GROUP BY hashes record by record from the Order_Inv1 table (11 records). In the Hash Table, the sum is recorded (not the count like in the previous example).

In summary, the difference and similarity between HASH UNIQUE (used in Distinct queries) and HASH GROUP BY (used in Group By queries) are as follows:

- *The difference*: The Hash Table used in the HASH GROUP BY operation has an additional attribute to store the aggregate value. This attribute does not exist in the Hash Table used in the HASH UNIQUE operation.
- *The similarity*: Both HASH UNIQUE and HASH GROUP BY operations hash record by record from the table to the Hash Table. When a record is hashed to a non-empty entry in the Hash Table, the record is ignored – resulting in duplicate records being removed (however, in HASH GROUP BY, an aggregate value will still be updated).

2.4. Group By and Order By

If a query contains Group By and Order By, which method is going to be used: HASH GROUP BY as in the Group By query, or SORT ORDER BY as in the Order By query? An example, the query is as follows:

```
SQL> Select Paymethod, Count(*)
  2  From Order1
  3  Group By Paymethod
  4  Order By Paymethod DESC;

PAYMETHOD              COUNT(*)
-------------------- ----------
CHEQUE                        3
CARD                          7

SQL>
```

The HASH GROUP BY method as in the Group By query performs a hashing function on the group by attribute, as well as performing an aggregate function (e.g. count, sum, etc). The records in the hash tables are not in any particular order. This is because when a hash function is applied to each record of the query table, the hash function performs a one-to-one operation from the record of the query table to the hash table. Therefore, the HASH GROUP BY method alone cannot be used.

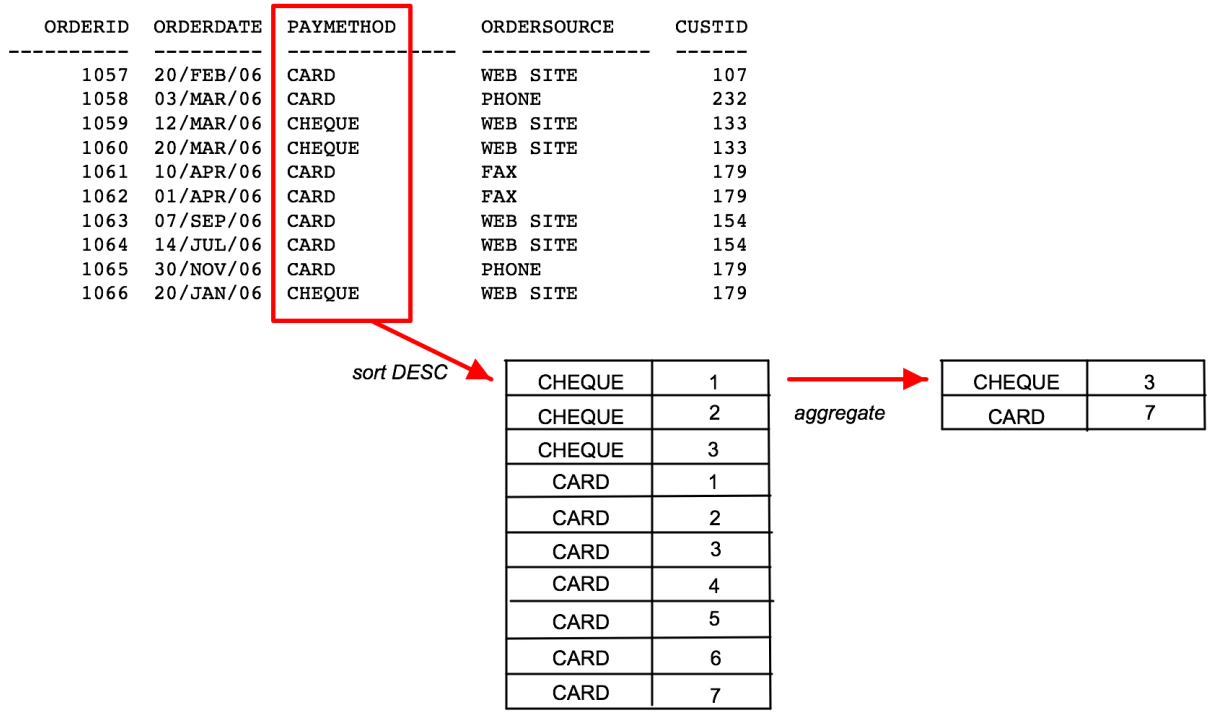
The SORT ORDER BY method as in the Order By query performs a sorting of the query result based on the attribute specified in the Order By clause. The SORT ORDER BY does not do any aggregate function. Therefore, the SORT ORDER BY alone cannot be used.

Performing two operations, firstly HASH GROUP BY to form the groups and the aggregate values for each group, followed by SORT ORDER BY is not optimal. The method to use to process a query with Group By and Order By is **SORT GROUP BY**, which is shown as follows.

```
SQL> Explain Plan For
  2  Select Paymethod, Count(*)
  3  From Order1
  4  Group By Paymethod
  5  Order By Paymethod DESC;

Explained.

SQL> Select * From Table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1529597951

-------------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |     2 |    12 |     4  (25)| 00:00:01 |
|   1 |  SORT GROUP BY     |        |     2 |    12 |     4  (25)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| ORDER1 |    10 |    60 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

9 rows selected.

SQL>
```

The SORT GROUP BY operation basically performs an aggregation function while sorting the records. As a result, the grouping as required by the Group By clause and the sorting as required by the Order By clause are achieved.

Only for illustration purposes, we show the operations in two steps: sorting, followed by aggregating.



2.5. Distinct and Order By

A query with Distinct is performed by the HASH UNIQUE operation, whereas a query with Order By is done by the SORT ORDER BY operation. The HASH UNIQUE operation does not sort the records, because hashing each record from the query table to the hash table is done on a one-by-one basis, and there is no relationship among records. Therefore, it is not guaranteed that the hash table will have their records sorted. SORT ORDER BY on the other hand only sorts the records, but does not remove duplicates as required by the Distinct.

Therefore, a query that contains both Distinct and Order By needs a new processing method, called **SORT UNIQUE**, which is basically a sorting operation, but duplicates are removed during the sorting process.

```
SQL> Explain Plan For
  2  Select Distinct Paymethod
  3  From Order1
  4  Order By Paymethod DESC;

Explained.

SQL> Select * From Table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1479042086

-------------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |     2 |    12 |     5  (40)| 00:00:01 |
|   1 |  SORT UNIQUE       |        |     2 |    12 |     4  (25)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| ORDER1 |    10 |    60 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

9 rows selected.
```

2.6. Join

A Join operation is to join two tables at a time. The following example joins table Customer1 and Order1 (the contents of Customer1 and Order1 are shows for reference).

```
SQL> Select * From Customer1 Order By CustID;

  CUSTID LNAME           FNAME           ADDRESS           PHONE    CITY
-------- --------------- --------------- ----------------- -------- ---------
     107 Smith           John            731 Plenty Road   9231455  Clayton
     133 Zelaya          Alicia          3321 Castle Ave   9867055  Balwyn
     154 Wallace         Jennifer        291 Berry Street  9234536  Preston
     179 Narayan         Ramesh          975 Fire Road     9456738  Carlton
     181 Jane            Adam            229 Clayton Road  9543877  Clayton
     183 Judy            Backhouse       122 Rose Street   9235345  Caulfield
     232 Wong            Franklin        638 Voss Street   9756945  Preston

7 rows selected.
```

```
SQL> Select * From Order1 Order By CustID;

   ORDERID ORDERDATE PAYMETHOD            ORDERSOURCE              CUSTID
---------- --------- -------------------- -------------------- ----------
      1057 20/FEB/06 CARD                 WEB SITE                    107
      1059 12/MAR/06 CHEQUE               WEB SITE                    133
      1060 20/MAR/06 CHEQUE               WEB SITE                    133
      1063 07/SEP/06 CARD                 WEB SITE                    154
      1064 14/JUL/06 CARD                 WEB SITE                    154
      1062 01/APR/06 CARD                 FAX                         179
      1066 20/JAN/06 CHEQUE               WEB SITE                    179
      1061 10/APR/06 CARD                 FAX                         179
      1065 30/NOV/06 CARD                 PHONE                       179
      1058 03/MAR/06 CARD                 PHONE                       232

10 rows selected.
```

```
SQL> Select *
  2  From Customer1 C, Order1 O
  3  Where C.CustID = O.CustID;

CUSTID LNAME   FNAME    ADDRESS          PHONE   CITY     ORDERID ORDERDATE PAYMETHOD ORDERSOURCE CUSTID
------ ------- -------- ---------------- ------- -------- ------- --------- --------- ----------- ------
   107 Smith   John     731 Plenty Road  9231455 Clayton  1057    20/FEB/06 CARD      WEB SITE    107
   133 Zelaya  Alicia   3321 Castle Ave  9867055 Balwyn   1059    12/MAR/06 CHEQUE    WEB SITE    133
   133 Zelaya  Alicia   3321 Castle Ave  9867055 Balwyn   1060    20/MAR/06 CHEQUE    WEB SITE    133
   154 Wallace Jennifer 291 Berry Street 9234536 Preston  1063    07/SEP/06 CARD      WEB SITE    154
   154 Wallace Jennifer 291 Berry Street 9234536 Preston  1064    14/JUL/06 CARD      WEB SITE    154
   179 Narayan Ramesh   975 Fire Road    9456738 Carlton  1062    01/APR/06 CARD      FAX         179
   179 Narayan Ramesh   975 Fire Road    9456738 Carlton  1066    20/JAN/06 CHEQUE    WEB SITE    179
   179 Narayan Ramesh   975 Fire Road    9456738 Carlton  1061    10/APR/06 CARD      FAX         179
   179 Narayan Ramesh   975 Fire Road    9456738 Carlton  1065    30/NOV/06 CARD      PHONE       179
   232 Wong    Franklin 638 Voss Street  9756945 Preston  1058    03/MAR/06 CARD      PHONE       232

10 rows selected.
```

The execution plan for the above join query between Customer1 and Order1 is as follows. The Plan Table shows that it uses a HASH JOIN operation. First it reads the two tables (CUSTOMER1 7 records, and ORDER1 10 records). Then it performs a HASH JOIN operation that produces 10 records, and finally, it displays the 10 records.

```
SQL> Explain Plan For
  2  Select *
  3  From Customer1 C, Order1 O
  4  Where C.CustID = O.CustID;

Explained.

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 3340424740

--------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |    10 |   780 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN         |           |    10 |   780 |     7  (15)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| CUSTOMER1 |     7 |   343 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| ORDER1    |    10 |   290 |     3   (0)| 00:00:01 |
```

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

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

   1 - access("C"."CUSTID"="O"."CUSTID")

15 rows selected.
```

If table Customer1 has a PRIMARY KEY (e.g. PK is CustID), attribute CustID is indexed. When an attribute is indexed, the values of the attributes are sorted in the index file. Therefore, if Customer1 has a CustID as PK, when we join Customer1 and Order1 tables based on CustID, the index in the Customer1 table can be used. When the index is used, Oracle uses a SORT-MERGE JOIN, instead of a HASH JOIN. But because only Customer1 is sorted, table Order1 needs to be sorted too, so that a Merge operation can be carried out later.

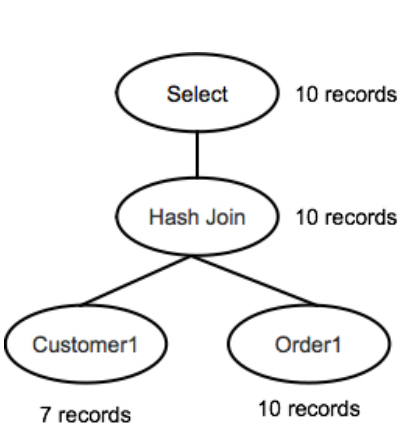
The following shows an execution plan when Customer1 table has CustID as the PK. After table Order1 is read (TABLE ACCESS FULL Order1 that has 10 records), the records are sorted using the SORT JOIN operation.

At the same time, the CustID index in Customer1 table is read (INDEX FULL SCAN of the index that contains 7 customer records). Because the index only stores CustID, whereas all attributes of Customer1 are needed in the query, the corresponding Customer1 records must be retrieved as well (TABLE ACCESS BY INDEX ROWID operation).

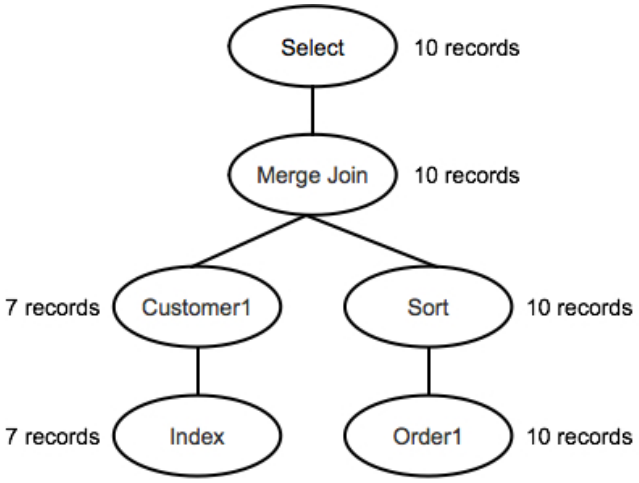
Once tables Customer1 and Order1 are read, a MERGE JOIN operation can be carried out. It produces 10 records, in which these 10 records are subsequently displayed (SELECT STATEMENT).

```
SQL> Explain Plan For
  2  Select * From Customer1 C, Order1 O Where C.CustID = O.CustID;

Explained.

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1831002277

-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |    10 |   780 |     6  (17)| 00:00:01 |
|   1 |  MERGE JOIN                   |              |    10 |   780 |     6  (17)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID | CUSTOMER1    |     7 |   343 |     2   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN            | SYS_C0012182 |     7 |       |     1   (0)| 00:00:01 |
|*  4 |   SORT JOIN                   |              |    10 |   290 |     4  (25)| 00:00:01 |
|   5 |    TABLE ACCESS FULL          | ORDER1       |    10 |   290 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   4 - access("C"."CUSTID"="O"."CUSTID")
       filter("C"."CUSTID"="O"."CUSTID")

18 rows selected.
```

The following show the query trees of both HASH JOIN and SORT-MERGE JOIN operations of the above join query.



Hash Join



Sort-Merge Join

If the query involves more than one join – there are more than two tables involved, then the join is done one after another. In the following query, there are two join, as there are three tables: Customer1, Order1, and Order_Inv1. The two joins are Customer1-Order1, and Order1-Order_Inv1. The join is based on PK-FK of the respective tables.

In this example, the join between Customer1 and Order1 is done first, and then the result of this is joined with Order_Inv1.

```
SQL> Explain Plan For
  2  Select *
  3  From Customer1 C, Order1 O, Order_Inv1 OI
  4  Where C.CustID = O.CustID
  5  And O.OrderID = OI.OrderID;

Explained.

SQL> Select * From Table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
Plan hash value: 1009371711

-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |    11 |  1023 |    10  (20)| 00:00:01 |
|*  1 |  HASH JOIN                    |              |    11 |  1023 |    10  (20)| 00:00:01 |
|   2 |   MERGE JOIN                  |              |    10 |   780 |     6  (17)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| CUSTOMER1    |     7 |   343 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN           | CUST_ID_IDX  |     7 |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                  |              |    10 |   290 |     4  (25)| 00:00:01 |
|   6 |     TABLE ACCESS FULL         | ORDER1       |    10 |   290 |     3   (0)| 00:00:01 |
|   7 |   TABLE ACCESS FULL           | ORDER_INV1   |    11 |   165 |     3   (0)| 00:00:01 |
```

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

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("O"."ORDERID"="OI"."ORDERID")
   5 - access("C"."CUSTID"="O"."CUSTID")

filter("C"."CUSTID"="O"."CUSTID")

21 rows selected.

SQL>
```

The query tree is shown as follows: