Data Science and Database Technology

Practice #5 – Oracle Optimizer

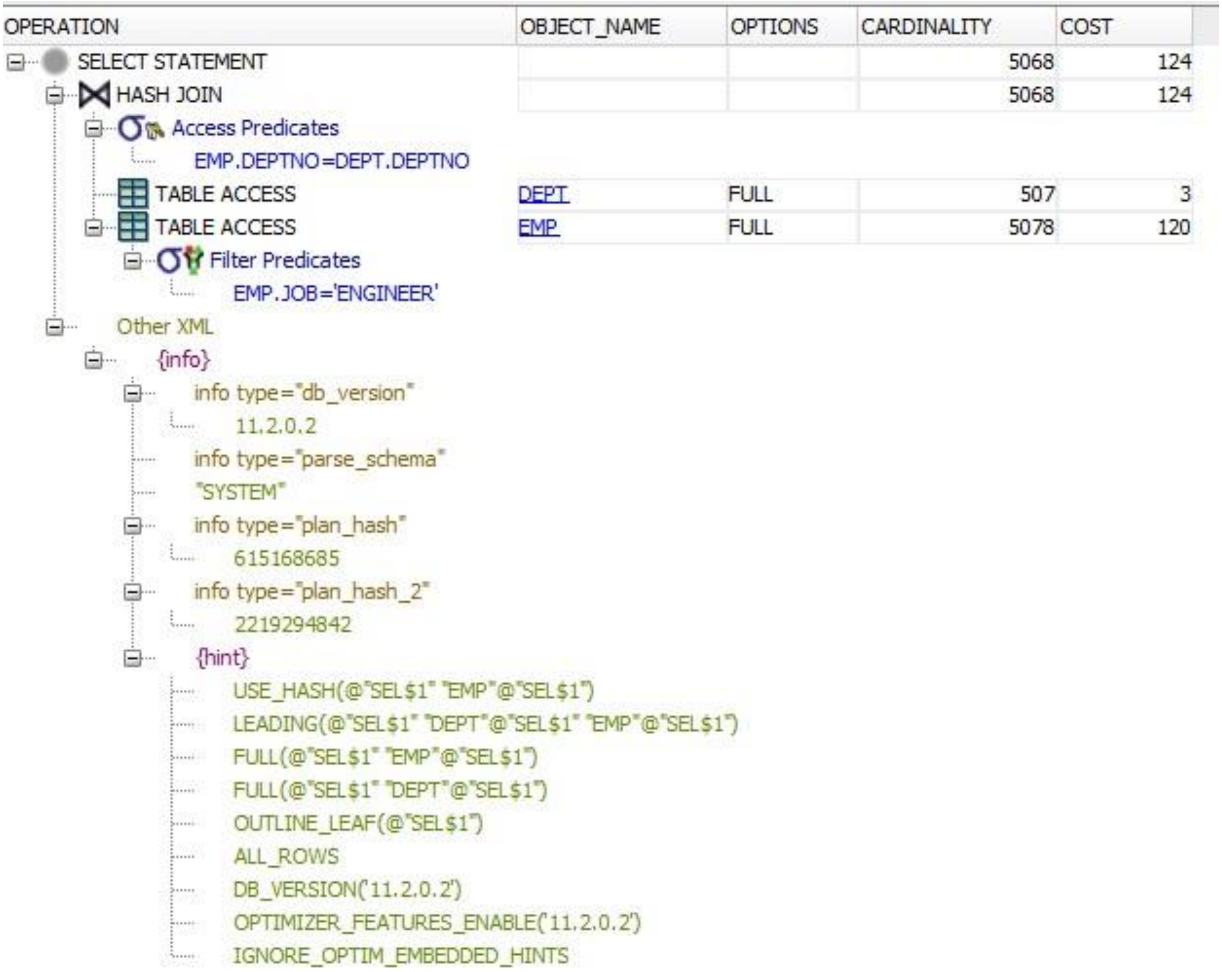
Queries

Query #1

```
SELECT *
FROM emp, dept
WHERE emp.deptno = dept.deptno AND emp.job = 'ENGINEER';
```

Change the optimizer goal from ALL ROWS (best throughput) to FIRST_ROWS (best response time) by means of the following hint. Set different values for n.

```
SELECT /*+ FIRST_ROWS(n) */ *
FROM emp, dept
WHERE emp.deptno = dept.deptno AND emp.job = 'ENGINEER';
```



```
OPERATION                              OBJECT_NAME  OPTIONS  CARDINALITY  COST
SELECT STATEMENT                                                    5068   124
  HASH JOIN                                                         5068   124
    Access Predicates
      EMP.DEPTNO=DEPT.DEPTNO
    TABLE ACCESS                       DEPT         FULL             507     3
    TABLE ACCESS                       EMP          FULL            5078   120
      Filter Predicates
        EMP.JOB='ENGINEER'
  Other XML
    {info}
      info type="db_version"
        11.2.0.2
      info type="parse_schema"
      "SYSTEM"
      info type="plan_hash"
        615168685
      info type="plan_hash_2"
        2219294842
      {hint}
        USE_HASH(@"SEL$1" "EMP"@"SEL$1")
        LEADING(@"SEL$1" "DEPT"@"SEL$1" "EMP"@"SEL$1")
        FULL(@"SEL$1" "EMP"@"SEL$1")
        FULL(@"SEL$1" "DEPT"@"SEL$1")
        OUTLINE_LEAF(@"SEL$1")
        ALL_ROWS
        DB_VERSION('11.2.0.2')
        OPTIMIZER_FEATURES_ENABLE('11.2.0.2')
        IGNORE_OPTIM_EMBEDDED_HINTS
```

- A Hash Join is performed on DEPTNO, following the filtering of the rows on the Job attribute. Access to the table: full (= read all rows).

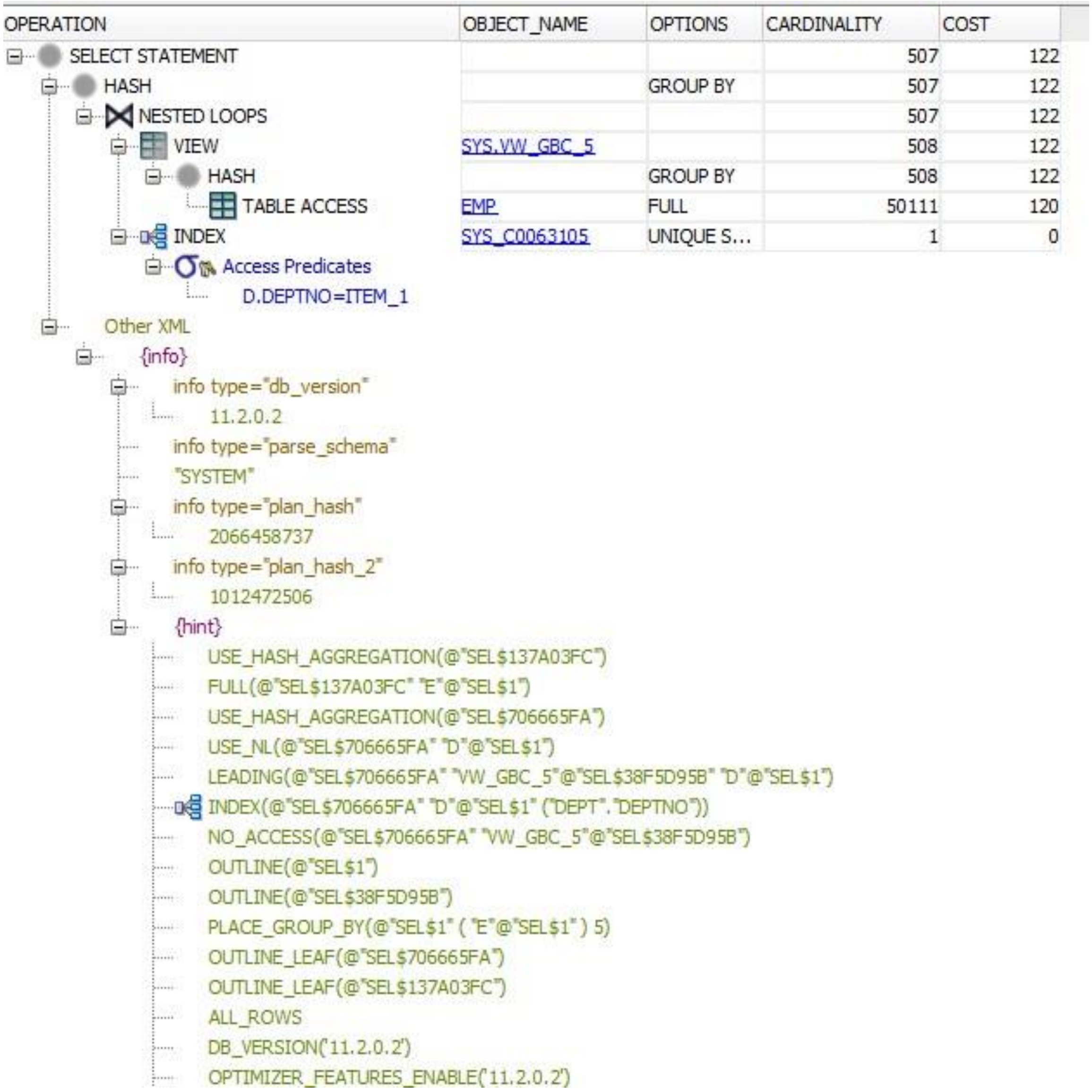
- The cost represented in the right column is cumulative. This increases in fact following the path from the leaves to the root of the tree of the execution plan. For each operation the cost of the latter is increased. In this example the leaves of the tree are 2: the two table accesses on dept and emp, with respective costs 3 and 120. Going up the tree, the costs of the two accesses are added together with the cost of the Hash Join (3 + 120 + 1 = 124). The hash join has a unit cost.

Query #2

Disable the hash join method by means of the following hint: (/*+ NO_USE_HASH(e d) */)

```
SELECT /*+ NO_USE_HASH(e d) */ d.deptno, AVG(e.sal)
FROM emp e, dept d
WHERE d.deptno = e.deptno
GROUP BY d.deptno;
```

With hint NO_USE_HASH:



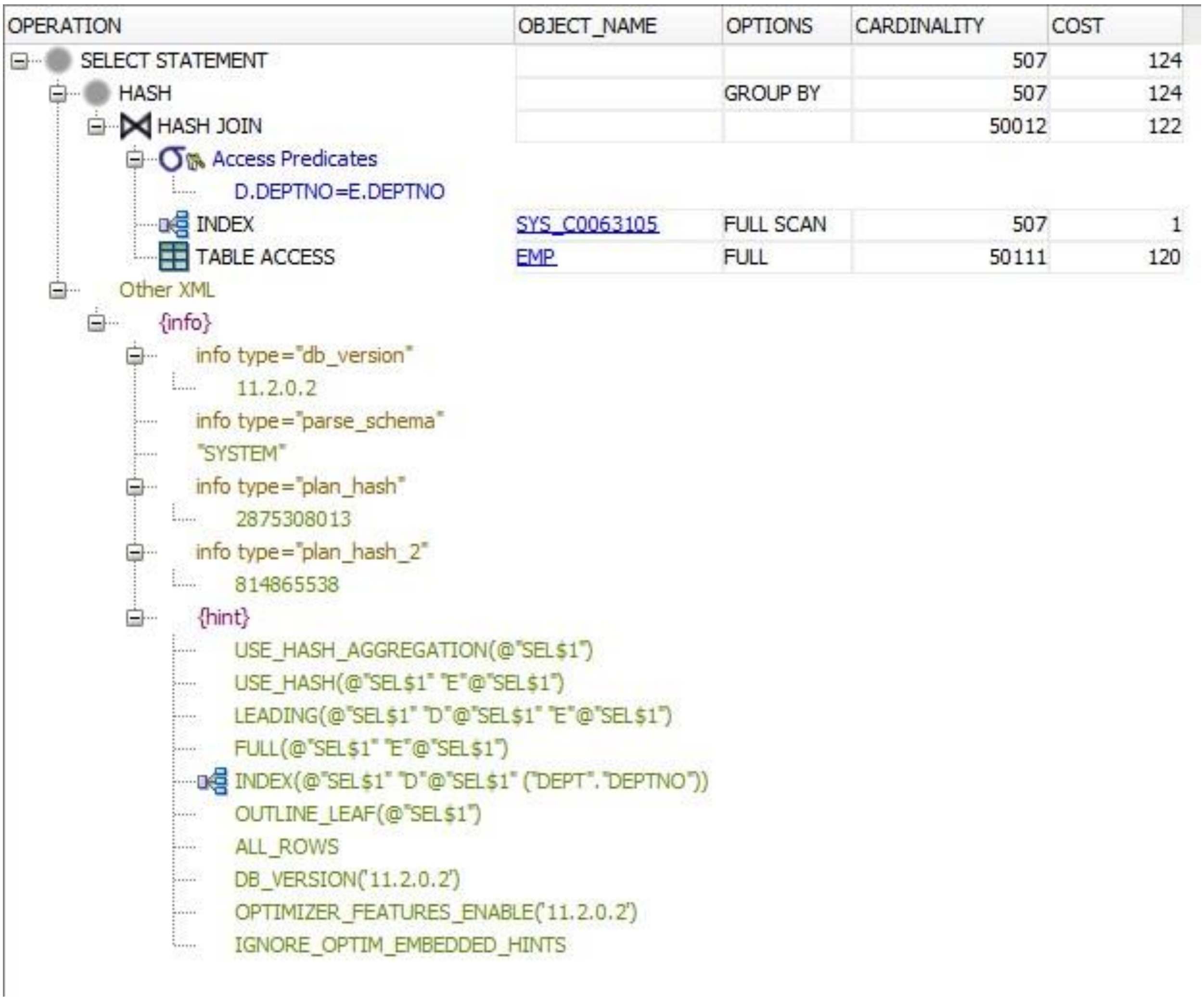
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			507	122
HASH		GROUP BY	507	122
NESTED LOOPS			507	122
VIEW	SYS.VW_GBC_5		508	122
HASH		GROUP BY	508	122
TABLE ACCESS	EMP	FULL	50111	120
INDEX	SYS_C0063105	UNIQUE S...	1	0
Access Predicates				
D.DEPTNO=ITEM_1				

```
Other XML
  {info}
    info type="db_version"
      11.2.0.2
    info type="parse_schema"
    "SYSTEM"
    info type="plan_hash"
      2066458737
    info type="plan_hash_2"
      1012472506
    {hint}
      USE_HASH_AGGREGATION(@"SEL$137A03FC")
      FULL(@"SEL$137A03FC" "E"@"SEL$1")
      USE_HASH_AGGREGATION(@"SEL$706665FA")
      USE_NL(@"SEL$706665FA" "D"@"SEL$1")
      LEADING(@"SEL$706665FA" "VW_GBC_5"@"SEL$38F5D95B" "D"@"SEL$1")
      INDEX(@"SEL$706665FA" "D"@"SEL$1" ("DEPT"."DEPTNO"))
      NO_ACCESS(@"SEL$706665FA" "VW_GBC_5"@"SEL$38F5D95B")
      OUTLINE(@"SEL$1")
      OUTLINE(@"SEL$38F5D95B")
      PLACE_GROUP_BY(@"SEL$1" ( "E"@"SEL$1" ) 5)
      OUTLINE_LEAF(@"SEL$706665FA")
      OUTLINE_LEAF(@"SEL$137A03FC")
      ALL_ROWS
      DB_VERSION('11.2.0.2')
      OPTIMIZER_FEATURES_ENABLE('11.2.0.2')
```

• DEPTNO = ITEM_1: when the Group By is executed, a view is created in which the key attribute (DEPTNO) is renamed with ITEM_1. ITEM_1 also becomes the view index for quick access to groups.

• The GROUP BY operation appears twice. The innermost one represents the advance of the group by (anticipated before the join). The outermost one represents Group By in the initial position, before the advance. In fact, if we observe the cumulative cost, this only increases with the internal group by (which does the work) and not with the external one (which must no longer group as the groups have already been created by the innermost one).

With hint USE_HASH:



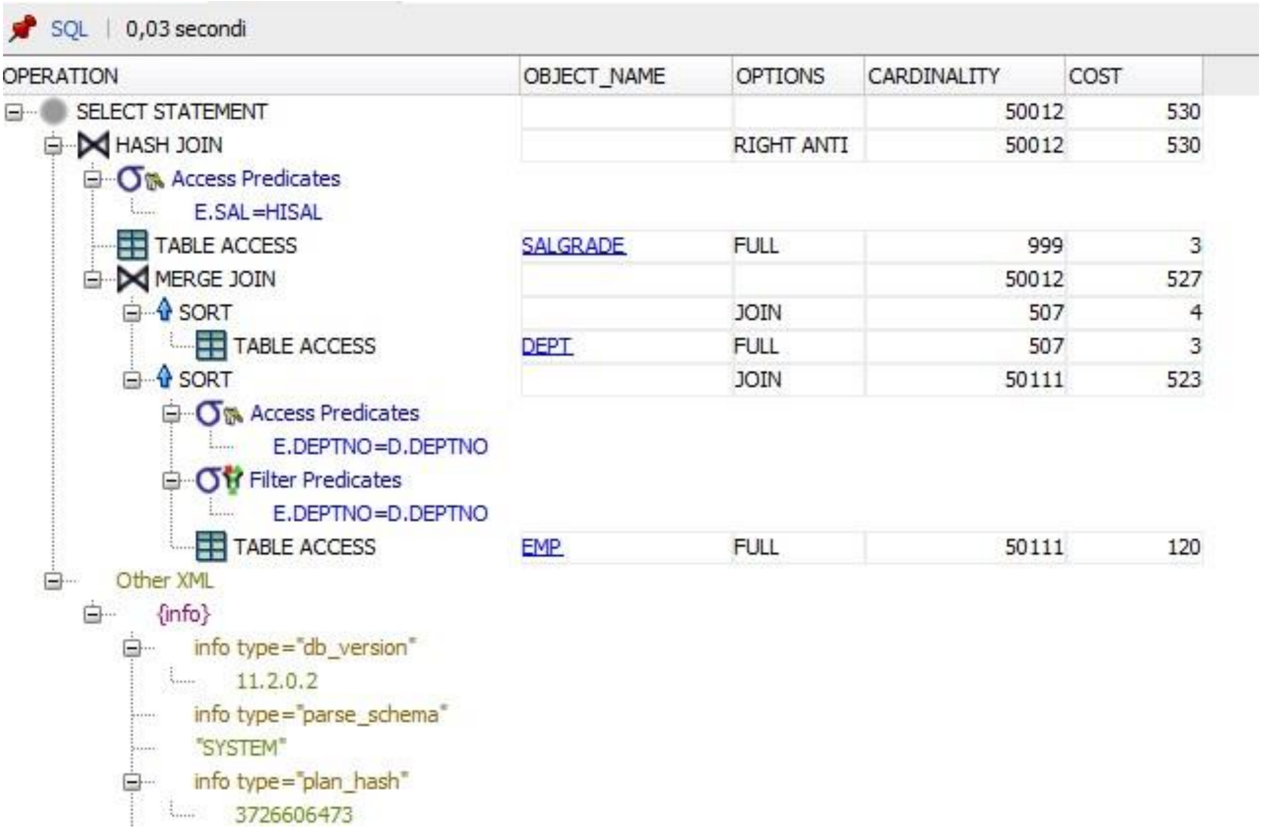
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			507	124
HASH		GROUP BY	507	124
HASH JOIN			50012	122
Access Predicates				
D.DEPTNO=E.DEPTNO				
INDEX	SYS_C0063105	FULL SCAN	507	1
TABLE ACCESS	EMP	FULL	50111	120
Other XML				
{info}				
info type="db_version"				
11.2.0.2				
info type="parse_schema"				
"SYSTEM"				
info type="plan_hash"				
2875308013				
info type="plan_hash_2"				
814865538				
{hint}				
USE_HASH_AGGREGATION(@"SEL$1")				
USE_HASH(@"SEL$1" "E"@"SEL$1")				
LEADING(@"SEL$1" "D"@"SEL$1" "E"@"SEL$1")				
FULL(@"SEL$1" "E"@"SEL$1")				
INDEX(@"SEL$1" "D"@"SEL$1" ("DEPT"."DEPTNO"))				
OUTLINE_LEAF(@"SEL$1")				
ALL_ROWS				
DB_VERSION('11.2.0.2')				
OPTIMIZER_FEATURES_ENABLE('11.2.0.2')				
IGNORE_OPTIM_EMBEDDED_HINTS				

• A hash join is used here. The group by, on the other hand, is not anticipated. Despite the hash join, the sequence of these operations remains less efficient than the previous point (probably due to the failure to advance the GROUP BY).

Query #3

Disable the hash join method by means of the following hint: (/*+ NO_USE_HASH(e d) */)

```
SELECT /*+ NO_USE_HASH(e d) */ ename, job, sal, dname
FROM emp e, dept d
WHERE e.deptno = d.deptno
AND NOT EXISTS
   (SELECT * FROM salgrade WHERE e.sal = hisal);
```



SQL | 0,03 secondi

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			50012	530
HASH JOIN		RIGHT ANTI	50012	530
Access Predicates				
E.SAL=HISAL				
TABLE ACCESS	SALGRADE	FULL	999	3
MERGE JOIN			50012	527
SORT		JOIN	507	4
TABLE ACCESS	DEPT	FULL	507	3
SORT		JOIN	50111	523
Access Predicates				
E.DEPTNO=D.DEPTNO				
Filter Predicates				
E.DEPTNO=D.DEPTNO				
TABLE ACCESS	EMP	FULL	50111	120
Other XML				
{info}				
info type="db_version"				
11.2.0.2				
info type="parse_schema"				
"SYSTEM"				
info type="plan_hash"				
3726606473				

Note. To write the schema of the query in relational algebra, the operator “NOT EXISTS” could be rewritten with a **NOT IN**. Indeed:

```
SELECT ename, job, sal, dname

FROM emp e, dept d

WHERE e.deptno = d.deptno

AND e.sal NOT IN

   (SELECT hisal FROM salgrade);
```

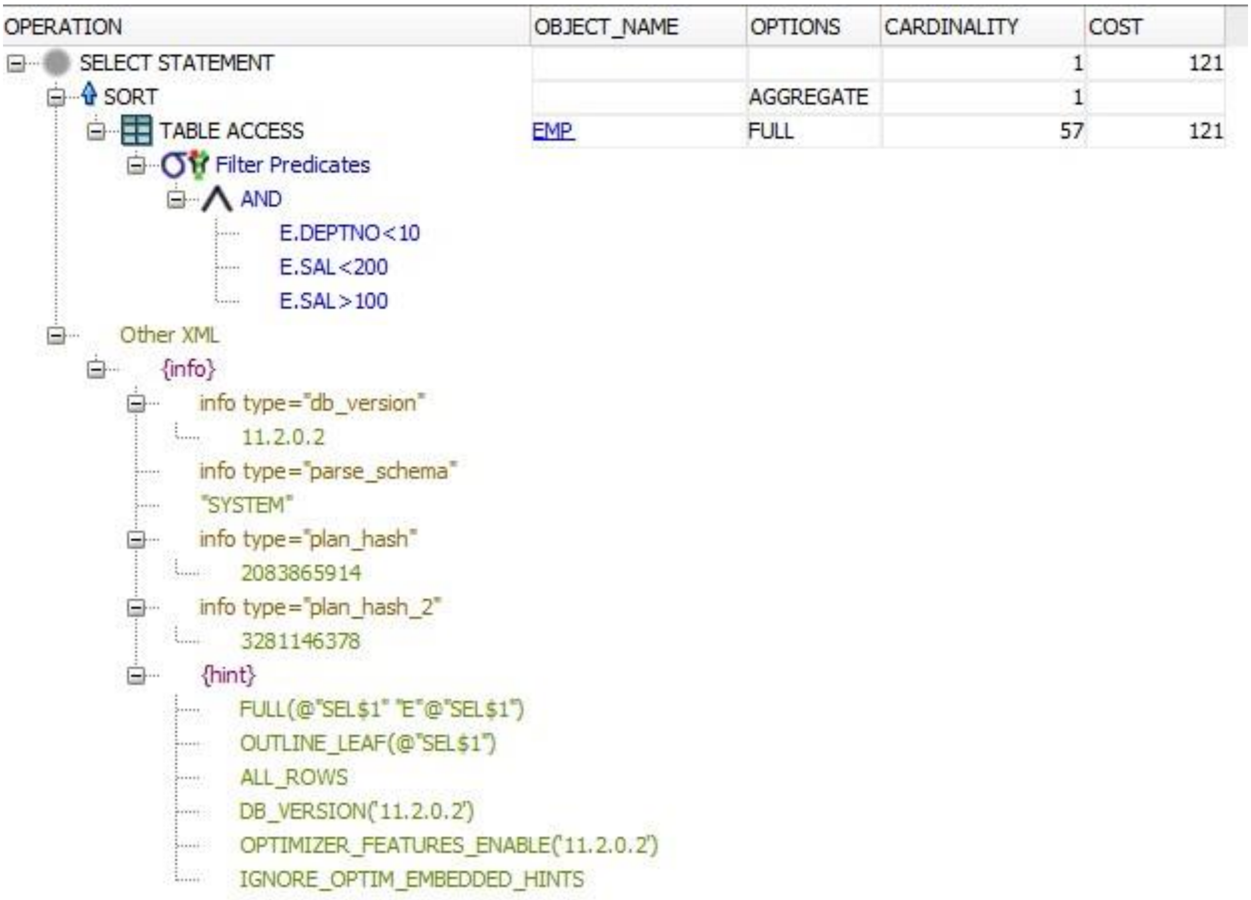
Queries #4

Select one or more secondary structures to optimize the following query:

```
select avg(e.sal) from
emp e where e.deptno <
10 and
e.sal > 100 and e.sal < 200;
```

Compare query performance using distinct secondary structures on different attributes with the one achieved by a unique secondary structure on multiple attributes.

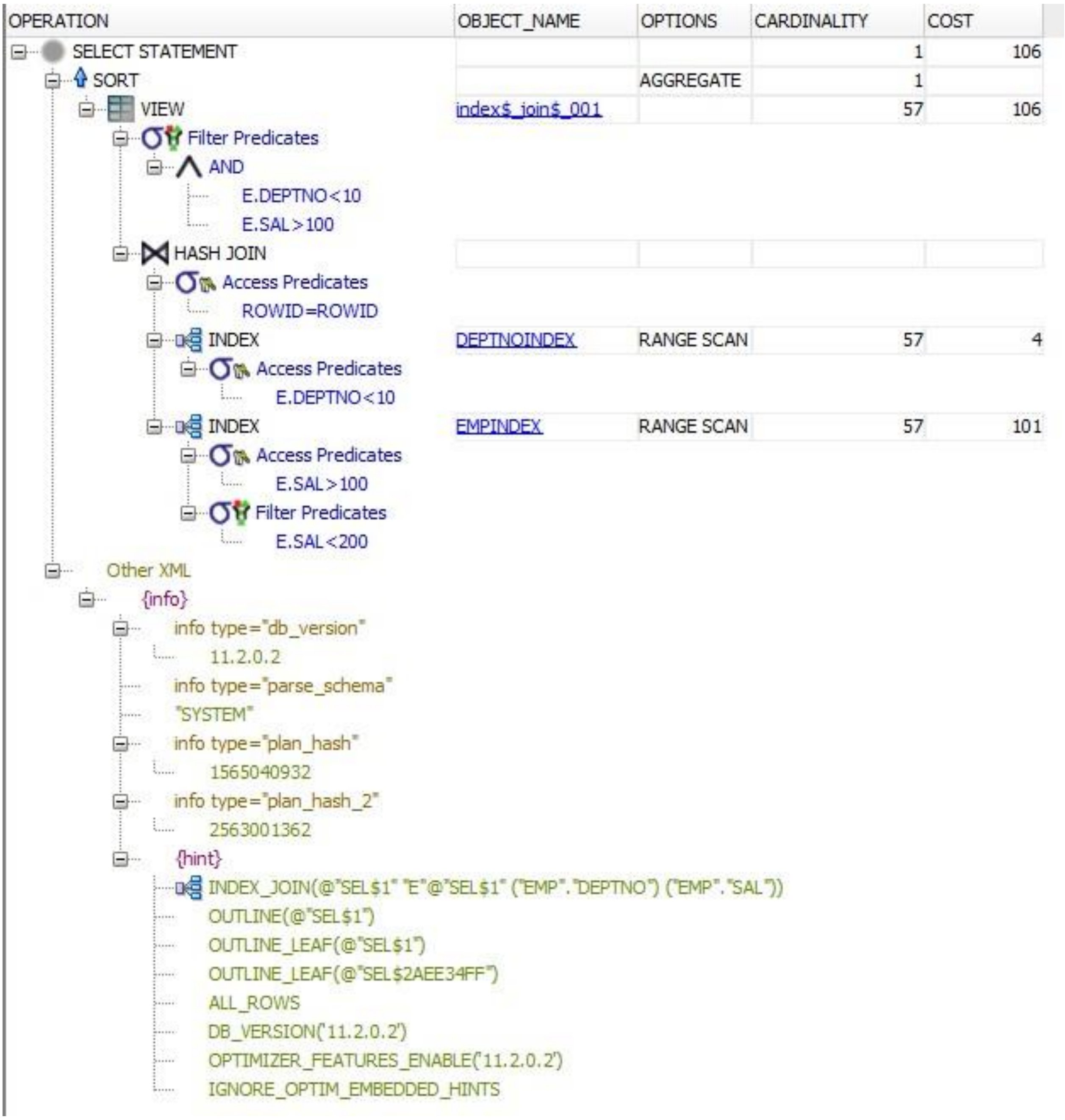
Without indexes:



```
OPERATION                                   OBJECT_NAME   OPTIONS     CARDINALITY   COST
SELECT STATEMENT                                                                1    121
  SORT                                                    AGGREGATE             1
    TABLE ACCESS                            EMP           FULL                 57    121
      Filter Predicates
        AND
            E.DEPTNO<10
            E.SAL<200
            E.SAL>100
  Other XML
    {info}
      info type="db_version"
          11.2.0.2
      info type="parse_schema"
      "SYSTEM"
      info type="plan_hash"
          2083865914
      info type="plan_hash_2"
          3281146378
      {hint}
          FULL(@"SEL$1" "E"@"SEL$1")
          OUTLINE_LEAF(@"SEL$1")
          ALL_ROWS
          DB_VERSION('11.2.0.2')
          OPTIMIZER_FEATURES_ENABLE('11.2.0.2')
          IGNORE_OPTIM_EMBEDDED_HINTS
```

With indexes (secondary) on sal and deptno:

```
create index SalIndex on EMP(sal)
create index DepIndex on EMP(deptno)
```



```
OPERATION                         OBJECT_NAME        OPTIONS      CARDINALITY   COST
SELECT STATEMENT                                                            1    106
  SORT                                               AGGREGATE              1
    VIEW                          index$_join$_001                         57    106
      Filter Predicates
        AND
          E.DEPTNO<10
          E.SAL>100
      HASH JOIN
        Access Predicates
          ROWID=ROWID
        INDEX                     DEPTNOINDEX        RANGE SCAN            57      4
          Access Predicates
            E.DEPTNO<10
        INDEX                     EMPINDEX           RANGE SCAN            57    101
          Access Predicates
            E.SAL>100
          Filter Predicates
            E.SAL<200
  Other XML
    {info}
      info type="db_version"
        11.2.0.2
      info type="parse_schema"
      "SYSTEM"
      info type="plan_hash"
        1565040932
      info type="plan_hash_2"
        2563001362
      {hint}
        INDEX_JOIN(@"SEL$1" "E"@"SEL$1" ("EMP"."DEPTNO") ("EMP"."SAL"))
        OUTLINE(@"SEL$1")
        OUTLINE_LEAF(@"SEL$1")
        OUTLINE_LEAF(@"SEL$2AEE34FF")
        ALL_ROWS
        DB_VERSION('11.2.0.2')
        OPTIMIZER_FEATURES_ENABLE('11.2.0.2')
        IGNORE_OPTIM_EMBEDDED_HINTS
```

- Indexes are used to select (access predicates) rows based on Sal and Deptno.
- An access predicate indicates, in the case of a B + Tree index, that the data is selected based on a certain attribute by descending the tree hierarchy

(logarithmic cost). The leaves of the tree group the data in such a way "Coarse" (= a leaf can contain many values for the indexed attribute).

- For this reason, an access predicate is often followed by a filter predicate, in which the data in the selected tree leaf is analyzed. During the filter predicate, a finer filter is then performed on the attribute values. This operation has a linear cost (all data in the leaf is analyzed).

- In this example the B + Tree is used to select Sal> 100, then linear search is used to select data with Sal <200.
- Subsequently, the index on DeptNo is used to select the departments.

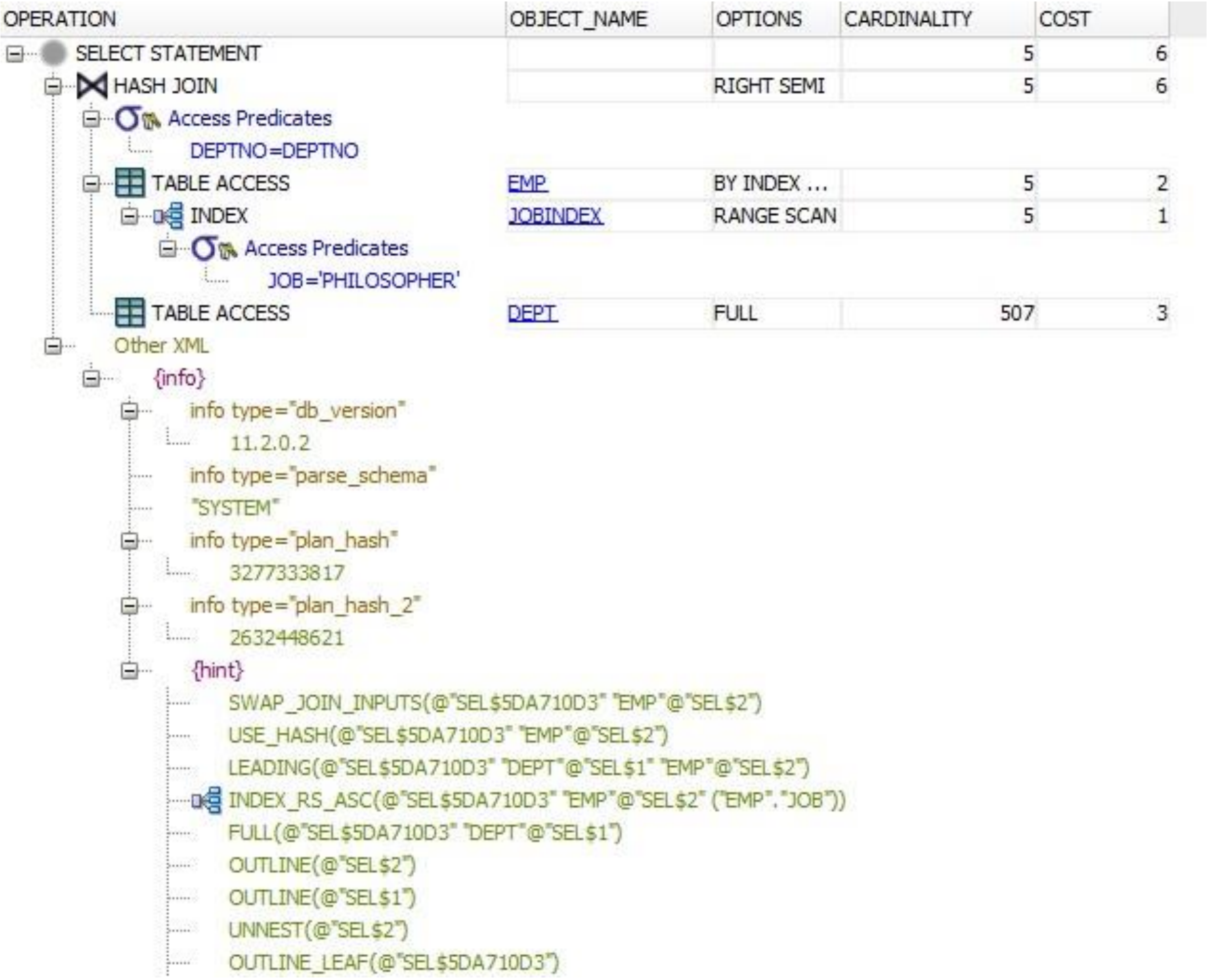
- The Hash Join is used to combine (through row id) the results obtained by filtering with the two indexes (intersection of the rows that satisfy the conditions).

Query #5

Select one or more secondary structures to optimize the following query:

```
select dname      from
dept
where deptno in (select deptno
from emp
                 where job = 'PHILOSOPHER');
```

Index to emp (job), since the internal query filters this attribute. An additional index on emp (deptno) would not help the join, since before the join a filter is performed on Job, which requires the index on the Job and an access by row id.



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5	6
HASH JOIN		RIGHT SEMI	5	6
Access Predicates				
DEPTNO=DEPTNO				
TABLE ACCESS	EMP	BY INDEX ...	5	2
INDEX	JOBINDEX	RANGE SCAN	5	1
Access Predicates				
JOB='PHILOSOPHER'				
TABLE ACCESS	DEPT	FULL	507	3
Other XML				
{info}				
info type="db_version"				
11.2.0.2				
info type="parse_schema"				
"SYSTEM"				
info type="plan_hash"				
3277333817				
info type="plan_hash_2"				
2632448621				
{hint}				
SWAP_JOIN_INPUTS(@"SEL$5DA710D3" "EMP"@"SEL$2")				
USE_HASH(@"SEL$5DA710D3" "EMP"@"SEL$2")				
LEADING(@"SEL$5DA710D3" "DEPT"@"SEL$1" "EMP"@"SEL$2")				
INDEX_RS_ASC(@"SEL$5DA710D3" "EMP"@"SEL$2" ("EMP"."JOB"))				
FULL(@"SEL$5DA710D3" "DEPT"@"SEL$1")				
OUTLINE(@"SEL$2")				
OUTLINE(@"SEL$1")				
UNNEST(@"SEL$2")				
OUTLINE_LEAF(@"SEL$5DA710D3")				

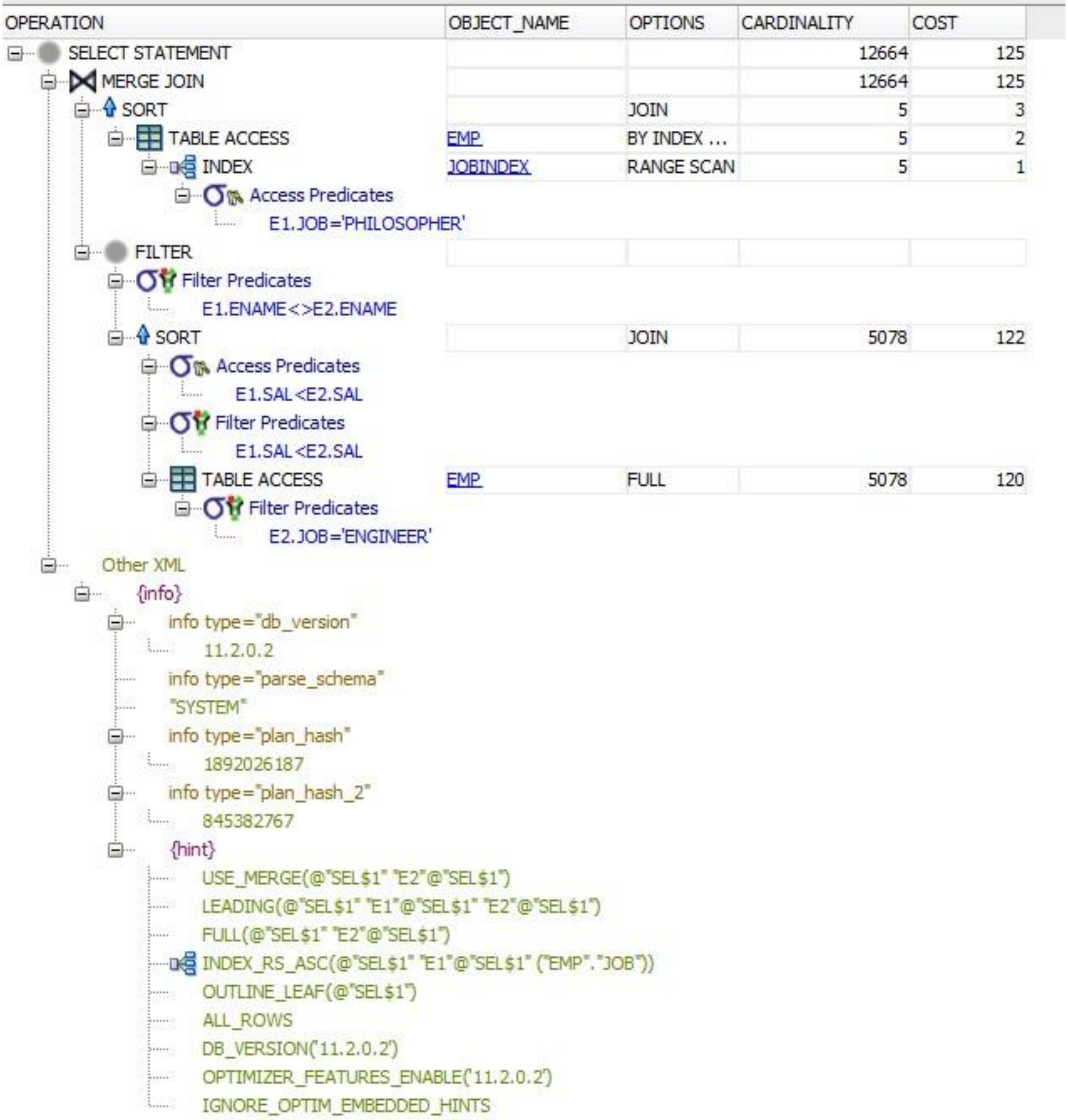
- The system performs an index reading on JOB to filter the EMP rows.
- It then performs an access by row id (OPTIONS = BY INDEX) to access the content of the other attributes of EMP.
- Finally it performs a hash join with the DEPT table

Query #6

Select one or more secondary structures to optimize the following query (remove already existing indexes to compare query performance with and without indexes):

```
select e1.ename, e1.empno, e1.sal, e2.ename, e2.empno, e2.sal          from
emp e1, emp e2
where e1.ename <> e2.ename and e1.sal < e2.sal
and e1.job = 'PHILOSOPHER' and e2.job = 'ENGINEER';
```

Indix on emp(job), emp(name) and emp(sal):



```
OPERATION                              OBJECT_NAME   OPTIONS        CARDINALITY   COST
SELECT STATEMENT                                                          12664    125
  MERGE JOIN                                                              12664    125
    SORT                                             JOIN                     5      3
      TABLE ACCESS                     EMP           BY INDEX ...             5      2
        INDEX                          JOBINDEX      RANGE SCAN               5      1
          Access Predicates
            E1.JOB='PHILOSOPHER'
    FILTER
      Filter Predicates
        E1.ENAME<>E2.ENAME
      SORT                                           JOIN                  5078    122
        Access Predicates
          E1.SAL<E2.SAL
        Filter Predicates
          E1.SAL<E2.SAL
        TABLE ACCESS                   EMP           FULL                  5078    120
          Filter Predicates
            E2.JOB='ENGINEER'
  Other XML
    {info}
      info type="db_version"
        11.2.0.2
      info type="parse_schema"
      "SYSTEM"
      info type="plan_hash"
        1892026187
      info type="plan_hash_2"
        845382767
      {hint}
        USE_MERGE(@"SEL$1" "E2"@"SEL$1")
        LEADING(@"SEL$1" "E1"@"SEL$1" "E2"@"SEL$1")
        FULL(@"SEL$1" "E2"@"SEL$1")
        INDEX_RS_ASC(@"SEL$1" "E1"@"SEL$1" ("EMP"."JOB"))
        OUTLINE_LEAF(@"SEL$1")
        ALL_ROWS
        DB_VERSION('11.2.0.2')
        OPTIMIZER_FEATURES_ENABLE('11.2.0.2')
        IGNORE_OPTIM_EMBEDDED_HINTS
```