Data Science and Database Technology

Practice #5 – Oracle Optimizer

Practice objective

Generate the execution plan for some SQL statements analyzing the following issues:

1. access paths
2. join orders and join methods
3. operation orders
4. exploitation of indexes defined by the user.

The evaluation will be performed using Oracle Database 18c Express Edition (Oracle XE).

Database schema

The database consists of 3 tables: (EMP, DEPT e SALGRADE). The table schema and some records are shown in the following.

Table ***EMP***

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	COZZA MARIA	PROFESSOR	0	09-JUN-81	1181		126
2	ECO LUIGI	PHDSTUDENT	0	09-JUN-81	1360		189
3	CORONA CLARA	PHDSTUDENT	2	09-JUN-81	624		15

Table ***DEPT***

DEPTNO	DNAME
1	*INFORMATION*
2	*CHAIRMANSHIP*
3	*ENVIRONMENT*
4	*PHYSICS*

Table ***SALGRADE***

GRADE	LOSAL	HISAL
1	*478*	*1503*
2	*661*	*1346*
3	*489*	*1358*
4	*942*	*1320*

Preliminary steps to perform the practice:

Connection to the database

Open the Oracle SQL Developer program and create a new connection.

Available materials

Some scripts with SQL statements are available to perform the following operations:

1. create an index on a table column
2. compute statistics for the database

The scripts are available at the course website in the `Scripts.zip` archive and the database at Lab5Database_OPT

The scripts can be loaded clicking on "Open" in the File Menu and selecting the .sql file. To execute the script click on the "Esegui Script" button as shown in the following figure.

To view the index statistics, execute the script show_indexes.sql (or copy the script content and paste it as SQL command).

Setting up the optimizer environment

At the beginning of working session you need to perform the following steps:

1. compute statistics on tables by means of the Web Interface or by the following script `comp_statistics_tables.sql`
2. check if there exist secondary indexes by means of the following SQL query `select INDEX_NAME from USER_INDEXES;`
   if secondary indexes (without considering system indexes, e.g., SYS_#) have been created, please, drop them by means of the following SQL statement `DROP INDEX IndexName;`

Execution plan computation for a query

To obtain the execution plan for a query through Web interface, it is necessary to execute the query and then to click the "Piano di esecuzione" button (as shown in Fig.1) to display the execution plan of the query.

Fig.1

The command to view the execution plan queries the PLAN_TABLE table.

If this already exists an error will be shown ("Invalid column name"). In this case, delete the table (drop table PLAN_TABLE;) and reissue the command.

The execution plan will be shown with the following format:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			50012	124
HASH JOIN			50012	124
Access Predicates				
EMP.DEPTNO=DEPT.DEPTNO				
TABLE ACCESS	DEPT	FULL	507	3
TABLE ACCESS	EMP	FULL	50111	120

OPERATION: operation to be performed on tables / indexes.
CARDINALITY: table / index size.
COST: cost of the operation, estimated value proportional to the resources used (CPU, I / O, memory). The cost refers to a specific node of the algebraic tree and is cumulative. This means that the cost of a node includes, in addition to the cost of the operation considered, the costs of all its child nodes.

In the example image, a hash join is performed between the two tables DEPT and EMP. The cost of accessing the EMP table is 120, that of the DEPT table is 3. The hash join node between the two tables has a cumulative cost of 124 (120 + 3 + 1). We therefore deduce that the single hash join operation has a cost of 1. The select has a cumulative cost of 124 because it does not introduce additional operations after the hash join.

Types of operations that can be found in the execution plan:
- **JOIN, GROUP BY, TABLE ACCESS, INDEX SCAN**
- **Access Predicates**: indicate one or more conditions that the records must satisfy in order to be selected. They are used on indexed (sorted) data. They allow you to specify the range **(start, stop)** of sorted records that satisfy the condition.
It is possible to find this node in a join (specify the join conditions, as in the example image) or in filter operations on the attributes of a table (WHERE conditions).

- **Filter Predicates**: indicate one or more conditions that the records must satisfy in order to be selected. Unlike access predicates, the filter operation is performed as you scroll through the sorted records. If both access and filter predicates are present, the former specify a range of indexed records, the latter allow filtering in that range.
It is possible to find this node in the filtering operations on the attributes of a table (WHERE conditions).

Useful SQL statements

- To view the table schema with all attributes: `DESCRIBE TableName;`
- To create an index:

```
CREATE INDEX IndexName ON TableName(ColumnName);
```

- To compute statistics related to indexes:

```
ANALYZE INDEX IndexName COMPUTE STATISTICS;
```

- To remove an index:

```
DROP INDEX IndexName;
```

- To view the indexes related to a table:

```
SELECT INDEX_NAME FROM USER_INDEXES
WHERE table_name='Table Name needs to be written in capital letters';
```

- Display statistics related to indexes:

```
SELECT USER_INDEXES.INDEX_NAME as INDEX_NAME, INDEX_TYPE,
USER_INDEXES.TABLE_NAME, COLUMN_NAME||'('||COLUMN_POSITION||')' as
COLUMN_NAME, BLEVEL, LEAF_BLOCKS, DISTINCT_KEYS, AVG_LEAF_BLOCKS_PER_KEY,
AVG_DATA_BLOCKS_PER_KEY, CLUSTERING_FACTOR
FROM user_indexes, user_ind_columns
WHERE user_indexes.index_name=user_ind_columns.index_name and
user_indexes.table_name=user_ind_columns.table_name;
```

- Display statistics related to tables:

```
SELECT TABLE_NAME, NUM_ROWS, BLOCKS, EMPTY_BLOCKS, AVG_SPACE, CHAIN_CNT,
AVG_ROW_LEN
FROM USER_TABLES;
```

- Display statistics related to table columns:

```
SELECT COLUMN_NAME, NUM_DISTINCT, NUM_NULLS, NUM_BUCKETS, DENSITY
FROM USER_TAB_COL_STATISTICS
WHERE TABLE_NAME = 'TableName' ORDER BY COLUMN_NAME;
```

- Display histograms:

```
SELECT *
FROM USER_HISTOGRAMS;
```

Queries

The following queries should be analyzed during the practice performing the following steps:

1. algebraic expression represented like a tree structure of the query
2. execution plan selected by Oracle optimizer when no physical secondary structures are defined
3. **Only** for queries from #4 to #7, Select one or more secondary physical structures to increase query performance.

Resume of table structures

EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
DEPT (DEPTNO, DNAME, LOC)
SALGRADE (GRADE, LOSAL, HISAL)

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';
```

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;
```

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);
```

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.

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');
```

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';
```

Optional exercise. With reference to the laboratory 3, analyze the execution plan generated for each of the 6 queries proposed and evaluate how it varies with / without the use of materialized views.