Warning

The executions plans below are generated by Oracle in the current environment, Oracle version and data. The logic behind the generated execution plans is also understandable.

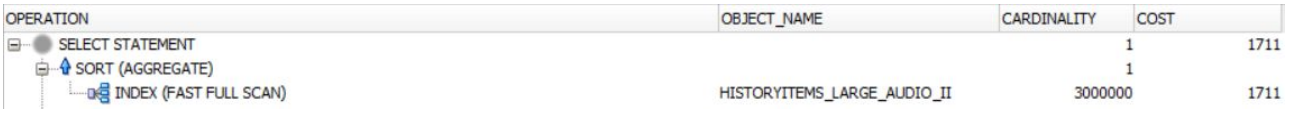
It can happen, however, that for the same query another execution plan is generated. Please document those and send them to me. I would particularly appreciate, if you also have an explenation.

Gergely Lukacs

Check the execution plan of the following for queries! Can you find any differences? What are the reasons for the differences?

1

```
SELECT COUNT(*)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```

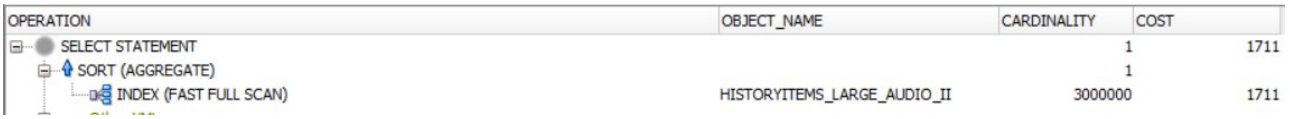


OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	1711
SORT (AGGREGATE)		1	
INDEX (FAST FULL SCAN)	HISTORYITEMS_LARGE_AUDIO_II	3000000	1711

1. How can it happen, that the join is not executed? (Check the schema: Is table audio_large required for the result? Why not? ;-))
2. Why is it possible to answer the query by accessing only the index historyitems_large_audio_ii (instead of the table historyitems_large)? What attributes are contained in the index? Check it with sqldeveloper!

2

```
SELECT COUNT(historyitems_large.rating)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```

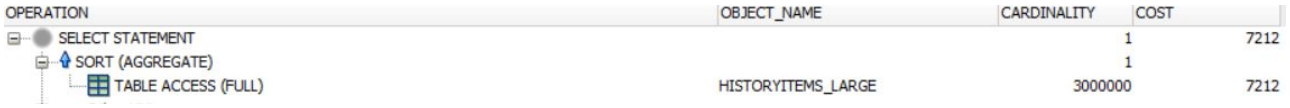


OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	1711
SORT (AGGREGATE)		1	
INDEX (FAST FULL SCAN)	HISTORYITEMS_LARGE_AUDIO_II	3000000	1711

1. What is the difference in the query with respect to the previous one?
2. In the execution plan?
3. Why is it possible to answer the query by accessing only the index historyitems_large_audio_ii (instead of the table historyitems_large)? The reasons from 1.2 are not sufficient, there is an additional reason in this case.

3

```
SELECT AVG(historyitems_large.rating)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```

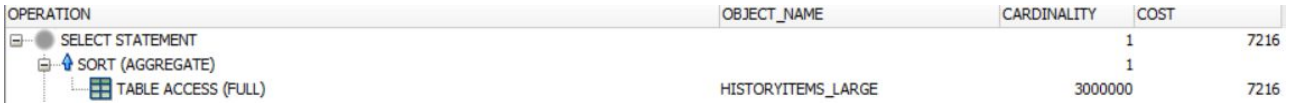


OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	7212
SORT (AGGREGATE)		1	
TABLE ACCESS (FULL)	HISTORYITEMS_LARGE	3000000	7212

1. What is the difference in the query with respect to 2?
2. In the execution plan?
3. Why did the execution plan change?

4

```
SELECT COUNT(historyitems_large.updated_at)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```



OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	7216
SORT (AGGREGATE)		1	
TABLE ACCESS (FULL)	HISTORYITEMS_LARGE	3000000	7216

1. What is the difference in the query with respect to 2?
2. In the execution plan?
3. Why did the execution plan change?

Check the execution plan of the following queries! (For the 2nd and 3rd queries, please only check the execution plans, without starting the queries.) What are the differences? Why?

1

```
SELECT COUNT(audio_large.user_id)
FROM audio_large
```

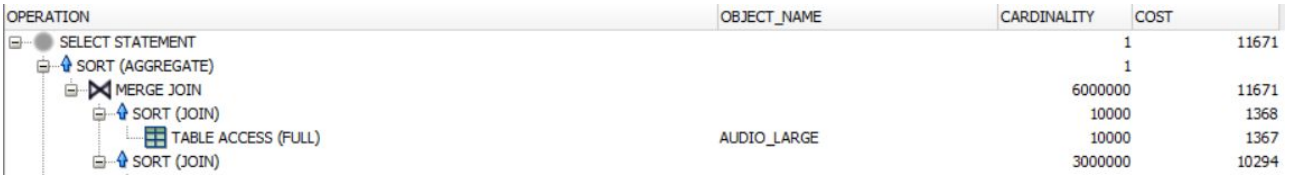
```
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	3086
SORT (AGGREGATE)		1	
HASH JOIN		3000000	3086
Access Predicates			
AUDIO_LARGE.ID=HISTORYITEMS_LARGE.AUDIO_ID			
NESTED LOOPS		3000000	3086
STATISTICS COLLECTOR			
TABLE ACCESS (FULL)	AUDIO_LARGE	10000	1367
INDEX (RANGE SCAN)	HISTORYITEMS_LARGE_AUDIO_II	300	1711
Access Predicates			
AUDIO_LARGE.ID=HISTORYITEMS_LARGE.AUDIO_ID			
INDEX (FAST FULL SCAN)	HISTORYITEMS_LARGE_AUDIO_II	3000000	1711

1. What operation is used in the join condition?
2. Which join execution method could be theoretically used?
3. Which join execution method is actually used? Why?
4. How high is the estimated cost of the query?

2

```
SELECT COUNT(audio_large.user_id)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id BETWEEN historyitems_large.audio_id - 0.5 AND
historyitems_large.audio_id + 0.5 ;
```



OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	11671
SORT (AGGREGATE)		1	
MERGE JOIN		6000000	11671
SORT (JOIN)		10000	1368
TABLE ACCESS (FULL)	AUDIO_LARGE	10000	1367
SORT (JOIN)		3000000	10294

1. What operation is used in the join condition?
2. Which join execution method could be theoretically used?
3. Which join execution method is actually used? Why?
4. How high is the estimated cost of the query? Change of cost with respect to 1?

3

```
SELECT COUNT(audio_large.user_id)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id - historyitems_large.audio_id = 0;
```

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	17178072
SORT (AGGREGATE)		1	
NESTED LOOPS		300000000	17178072
TABLE ACCESS (FULL)	AUDIO_LARGE	10000	1367
INDEX (FAST FULL SCAN)	HISTORYITEMS_LARGE_AUDIO_II	30000	1718
Filter Predicates			
AUDIO_LARGE.ID-HISTORYITEMS_LARGE.AUDIO_ID=0			

1. What operation is used in the join condition?
2. Which join execution method could be theoretically used?
3. Which join execution method is actually used? Why?
4. How high is the estimated cost of the query? Change of cost with respect to 1 and 2?

Check changes in the execution plan as the selectivity of the condition in the WHERE clause changes! (Change the condition on the user_id first!):

```
SELECT *
FROM   audio_large
       inner join historyitems_large
               ON audio_large.id = historyitems_large.audio_id
WHERE  To_char(historyitems_large.started_at, 'yyyymmdd') =
'20160302'
       AND audio_large.user_id < 5;
```

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		13	2571
HASH JOIN		13	2571
Access Predicates			
AUDIO_LARGE.ID=HISTORYITEMS_LARGE.AUDIO_ID			
NESTED LOOPS		13	2571
NESTED LOOPS		1200	2571
STATISTICS COLLECTOR			
TABLE ACCESS (FULL)	AUDIO_LARGE	4	1367
Filter Predicates			
AUDIO_LARGE.USER_ID<5			
INDEX (RANGE SCAN)	HISTORYITEMS_LARGE_AUDIO_II	300	2
Access Predicates			
AUDIO_LARGE.ID=HISTORYITEMS_LARGE.AUDIO_ID			
TABLE ACCESS (BY INDEX ROWID)	HISTORYITEMS_LARGE	3	301
Filter Predicates			
TO_CHAR(INTERNAL_FUNCTION(HISTORYITEMS_LARGE.STARTED_AT),'yyy			
TABLE ACCESS (FULL)	HISTORYITEMS_LARGE	3	301
Filter Predicates			
TO_CHAR(INTERNAL_FUNCTION(HISTORYITEMS_LARGE.STARTED_AT),'yyyymm			

```
SELECT *
FROM   audio_large
       inner join historyitems_large
               ON audio_large.id = historyitems_large.audio_id
```

```
WHERE  To_char(historyitems_large.started_at, 'yyyymmdd') =
'20160302'
       AND audio_large.user_id < 500;
```

1. This is a somewhat complicated case.
2. For any case, the execution plan changes fundamentally as you change the constant value in the condition.
3. For stricter, more selective query condition the index is used, even though in a tricky way.
4. The “STATISTICS COLLECTOR” operation is quite tricky: Please Google: https://www.google.hu/search?q=oracle+statistics+collector