

Вынудим оптимизатор использовать другой тип соединения (Хэш)

```
select /*+ USE_HASH(E D) */ ename, e.deptno, d.deptno, d.dname from
emp e, dept d where e.deptno = d.deptno and ename like 'A%';
```

Query Result x Autotrace x					
SQL HotSpot 0,059 seconds					
OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT			7		
HASH JOIN		1	7	12	905
Access Predicates E.DEPTNO=D.DEPTNO					
TABLE ACCESS (FULL)	EMP	1	3	6	156
Filter Predicates ENAME LIKE 'A%'					
TABLE ACCESS (FULL)	DEPT	4	3	6	49

В. **Что получили?** (Сравнить стоимости и затраты на чтение данных)

PLAN_TABLE_OUTPUT							
Plan hash value: 1123238657							

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	

0	SELECT STATEMENT		1	22	5 (20)	00:00:01	
* 1	HASH JOIN		1	22	5 (20)	00:00:01	
* 2	TABLE ACCESS FULL	EMP	1	9	2 (0)	00:00:01	
3	TABLE ACCESS FULL	DEPT	4	52	2 (0)	00:00:01	

Predicate Information (identified by operation id):							

1 - access("E"."DEPTNO"="D"."DEPTNO")							
2 - filter("ENAME" LIKE 'A%')							

При использовании данного запроса, Cost увеличился, по сравнению с запросом А.

Принудительно используем метод соединения слиянием и сортировкой

```
select /*+ USE_MERGE(E D) */ ename, e.deptno, d.deptno, d.dname from
emp e, dept d where e.deptno = d.deptno and ename like 'A%';
```

Query Result x Autotrace x		SQL HotSpot 0,072 seconds				
OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	
SELECT STATEMENT			8			
MERGE JOIN		1	8	15	175	
TABLE ACCESS (CLUSTER)	EMP	1	4	9	77	
Filter Predicates						
ENAME LIKE 'A%'						
INDEX (FULL SCAN)	EMP_DEPT_CLUSTER_IDX	1	1	1	28	
SORT (JOIN)		4	4	6	70	
Access Predicates						
E.DEPTNO=D.DEPTNO						
Filter Predicates						
E.DEPTNO=D.DEPTNO						
TABLE ACCESS (FULL)	DEPT	4	3	6	44	

С. Что получили? (Сравнить стоимости и затраты на чтение данных)

PLAN_TABLE_OUTPUT							
Plan hash value: 2125045483							
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
0	SELECT STATEMENT		1	22	5 (20)	00:00:01	
1	MERGE JOIN		1	22	5 (20)	00:00:01	
2	TABLE ACCESS BY INDEX ROWID	DEPT	4	52	2 (0)	00:00:01	
3	INDEX FULL SCAN	DEPT_PK	4		1 (0)	00:00:01	
* 4	SORT JOIN		1	9	3 (34)	00:00:01	
* 5	TABLE ACCESS FULL	EMP	1	9	2 (0)	00:00:01	
Predicate Information (identified by operation id):							
4 - access("E"."DEPTNO"="D"."DEPTNO")							
filter("E"."DEPTNO"="D"."DEPTNO")							
5 - filter("ENAME" LIKE 'A%')							

При использовании данного запроса, Cost увеличился, по сравнению с запросом С.

Оптимизатор выбрал метод Merge Join. Принудительно заставим оптимизатор выполнить HASH JOIN.

```
SELECT /*+ USE_HASH(h l) */ * FROM OEorders h, OEorder_items l WHERE l.order_id = h.order_id;
```

SQL HotSpot | 0,131 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT			7		
HASH JOIN		665	7	8	1354
Access Predicates L.ORDER_ID=H.ORDER_ID					
TABLE ACCESS (FULL)	OEORDERS	105	3	6	182
TABLE ACCESS (FULL)	OEORDER_ITEMS	665	3	2	78

V\$STATNAME Name	V\$MYSTAT Value
bytes received via SQL*Net from client	477
bytes sent via SQL*Net to client	27640
calls to get snapshot scn: kcmgss	2
calls to kcmgcs	2
consistent gets	8
consistent gets from cache	8
consistent gets from cache (fastpath)	8
CPU used by this session	5
CPU used when call started	5
cursor authentications	1

Е. Что получили? (Сравнить стоимости и затраты на чтение данных)

PLAN_TABLE_OUTPUT							
Plan hash value: 3076855139							

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	

0	SELECT STATEMENT		665	105K	6 (17)	00:00:01	
* 1	HASH JOIN		665	105K	6 (17)	00:00:01	
2	TABLE ACCESS FULL	OEORDERS	105	10185	2 (0)	00:00:01	
3	TABLE ACCESS FULL	OEORDER_ITEMS	665	43225	3 (0)	00:00:01	

Predicate Information (identified by operation id):							

1 - access("L"."ORDER_ID"="H"."ORDER_ID")							

Note							

- dynamic sampling used for this statement (level=2)							

При использовании данного запроса, Cost увеличился, по сравнению с запросом Е.

Сравним с планом выполнения запроса, в котором используется NESTED LOOPS.

```
SELECT /*+ USE_NL(h l) */ * FROM OEorders h, OEorder_items l WHERE
l.order_id = h.order_id;
```

SQL HotSpot 0,058 seconds					
OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT			148		
NESTED LOOPS		665	148	44	606
TABLE ACCESS (FULL)	OEORDERS	105	3	6	59
TABLE ACCESS (FULL)	OEORDER_ITEMS	6	1	38	388
Filter Predicates					
L.ORDER_ID=H.ORDER_ID					

VSSTATNAME Name	VSMYSTAT Value
buffer is not pinned count	8
buffer is pinned count	1
bytes received via SQL*Net from client	475
bytes sent via SQL*Net to client	26500
calls to get snapshot scn: kcmgss	4
calls to kcmgcs	8
consistent gets	48
consistent gets from cache	48
consistent gets from cache (fastpath)	48
CPU used by this session	2

Е. Что получили? (Сравнить стоимости и затраты на чтение данных)

PLAN_TABLE_OUTPUT							
Plan hash value: 4217497175							

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	

0	SELECT STATEMENT		665	105K	3 (0)	00:00:01	
1	NESTED LOOPS						
2	NESTED LOOPS		665	105K	3 (0)	00:00:01	
3	TABLE ACCESS FULL	OEORDER_ITEMS	665	43225	3 (0)	00:00:01	
* 4	INDEX UNIQUE SCAN	ORDER_PK	1		0 (0)	00:00:01	
5	TABLE ACCESS BY INDEX ROWID	OEORDERS	1	97	0 (0)	00:00:01	

Predicate Information (identified by operation id):							

4 - access("L"."ORDER_ID"="H"."ORDER_ID")							

Note							

- dynamic sampling used for this statement (level=2)							

При использовании данного запроса, Cost уменьшился, по сравнению с запросом Е.

Hash Join

Метод хэш-соединения очень хорошо работает с большими источниками строк и с источниками строк, где один источник строки меньше. Хэш-соединение создает хэш-таблицу в памяти (и переполняется до временного табличного пространства, если источник строки слишком велик) для меньшего из источников строк. Затем процедура считывает источник второй строки, исследуя хэш-значение в первой. Поскольку строки объединяются, как только обнаруживается совпадение, метод хэш-соединения также является предпочтительным для цели FIRST_ROWS.

Для выполнения примеров этого раздела используются таблицы схемы **SH** (**SHSales**, **SHCustomres**). Файл для создания таблицы Sales прилагается (таблица **SHCustomres** была создана ранее).

Рекомендуется для приведения в соответствие Id Customer выполнить изменения данных в таблице **SHSales** (`update shsales set cust_id=cust_id+30000;`)

```
SELECT c.cust_first_name, c.cust_last_name, c.cust_id,
COUNT(s.prod_id) FROM shcustomers c, shsales s where c.cust_id =
s.cust_id and c.cust_id < 1000000 GROUP BY c.cust_first_name,
c.cust_last_name, c.cust_id;
```

SQL HotSpot | 1,108 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT			280		
HASH (GROUP BY)		2804	280	993	1009045
HASH JOIN		2804	279	993	1008111
Access Predicates					
C.CUST_ID=S.CUST_ID					
TABLE ACCESS (FULL)	SHSALES	2804	5	16	3421
Filter Predicates					
S.CUST_ID<1000000					
TABLE ACCESS (FULL)	SHCUSTOMERS	49802	273	977	782637
Filter Predicates					
C.CUST_ID<1000000					

V\$STATNAME Name	V\$MYSTAT Value
bytes received via SQL*Net from client	593
bytes sent via SQL*Net to client	26158
calls to get snapshot scn: kcmgss	5
calls to kcmgcs	22
cell physical IO interconnect bytes	7954432
consistent gets	1044
consistent gets direct	970
consistent gets from cache	74
consistent gets from cache (fastpath)	73
CPU used by this session	35
CPU used when call started	35
DB time	106

G. Что получили? (анализ плана выполнения запроса)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT						
HASH		GROUP BY		6	10	28669
HASH JOIN				6	10	28669
Access Predicates					9	23
C.CUSTOMER_ID=S.CUSTOMER_ID						23
TABLE ACCESS	DECUSTOMERS	FULL		2	3	7
Filter Predicates						66
C.CUSTOMER_ID<1000000						
TABLE ACCESS	SHSALES	FULL		2804	5	16
Filter Predicates						7367
AND						
S.CUSTOM_ID<1000000						
S.CUSTOM_ID>0						

Выполним тот же запрос еще раз, но с использованием подсказки для принудительного использования метода вложенных циклов.

```
SELECT /*+ USE_NL(c s) */c.cust_first_name,
c.cust_last_name,c.cust_id, COUNT(s.prod_id)FROM shcustomers c,
shsales s
where c.cust_id = s.cust_id and c.cust_id < 100090
GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id;
```

Для корректной работы примера в условия выбора строк внесены изменения.

Рекомендация: найти пороговое значение для условия `c.cust_id < 100090`, чтобы этот и предыдущий пример работали корректно без переписывания условия.

SQL HotSpot | 0,109 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT			7		
HASH (GROUP BY)		1	7	16	151
NESTED LOOPS				16	117
NESTED LOOPS		1	6	16	110
TABLE ACCESS (FULL)	SHSALES	1	5	16	105
Filter Predicates					
S.CUST_ID<100					
INDEX (UNIQUE SCAN)	CUSTOMERS_PK	1	0	0	0
Access Predicates					
C.CUST_ID=S.CUST_ID					
Filter Predicates					
C.CUST_ID<100					
TABLE ACCESS (BY INDEX ROWID)	SHCUSTOMERS	1	1	0	0

VSSTATNAME Name	VSMYSTAT Value
bytes received via SQL*Net from client	607
bytes sent via SQL*Net to client	24860
calls to get snapshot scn: kcmgss	6
calls to kcmgcs	16
consistent gets	69
consistent gets from cache	69
consistent gets from cache (fastpath)	69
CPU used by this session	4
CPU used when call started	4

Н. Что получили? (Сравнить стоимости и затраты на чтение данных)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT					9	16
HASH		GROUP BY		1	9	249
NESTED LOOPS					8	16
TABLE ACCESS	SHSALES	FULL		1	5	164
Filter Predicates						152
AND						
S.CUST_ID<100090						
S.CUST_ID>0						
TABLE ACCESS	SHCUSTOMERS	FULL		1	3	0
Filter Predicates						
AND						
C.CUSTOMER_ID<100090						
C.CUSTOMER_ID=S.CUST_ID						

При использовании данного запроса, Cost уменьшился, по сравнению с запросом Г.

Выполним тот же запрос еще раз, но с использованием подсказки для принудительного использования метода соединения слиянием.

```
SELECT /*+ USE_MERGE(c s) */
c.cust_first_name, c.cust_last_name, c.cust_id, COUNT(s.prod_id)
FROM shcustomers c, shsales s
where c.cust_id = s.cust_id
and c.cust_id < 100090
GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id
```

SQL HotSpot | 0.466 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT			218		
HASH (GROUP BY)		210	218	244	358539
MERGE JOIN		210	217	244	358010
TABLE ACCESS (BY INDEX ROWID)	SHCUSTOMERS	9810	211	228	44682
INDEX (RANGE SCAN)	CUSTOMERS_PK	9810	22	31	14297
Access Predicates					
C.CUST_ID<100090					
SORT (JOIN)		210	6	16	250024
Access Predicates					
C.CUST_ID=S.CUST_ID					
Filter Predicates					
C.CUST_ID=S.CUST_ID					
TABLE ACCESS (FULL)	SHSALES	210	5	16	349
Filter Predicates					
S.CUST_ID<100090					

VSSTATNAME Name	VSMYSTAT Value
buffer is not pinned count	199
buffer is pinned count	19422
bytes received via SQL*Net from client	614
bytes sent via SQL*Net to client	25110
calls to get snapshot scn: kcmgss	6
calls to kcmgcs	16
consistent gets	297

I. Что получили? (Сравнить стоимости и затраты на чтение данных)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT						
HASH		GROUP BY		6	11	29162
HASH JOIN				6	11	29162
Access Predicates					10	28901
ITEM_1=S.CUST_ID						
VIEW	VW_GSC_S			2	4	148
HASH		GROUP BY		2	4	130
TABLE ACCESS	OCUSTOMERS	FULL		2	3	18
Filter Predicates						
C.CUSTOMER_ID<100090						
TABLE ACCESS	SHSALES	FULL	2804		5	7323

При использовании данного запроса, Cost увеличился, по сравнению с запросом H.

Ж. Указать причины, по которым оптимизатор выбирает тот или иной метод соединения.

Oracle применяется оптимизатор по стоимости (Cost-Based Optimizer — CBO), который основываясь на статистических данных по таблицам и индексам, порядке таблиц и столбцов в SQL-операторах, доступных индексах и любых предоставляемых пользователем подсказках касательно методов доступа, выбирает наиболее эффективный способ для получения к ним доступа. Самым эффективным, по его мнению, является метод доступа, имеющий наименьшую стоимость в плане ресурсов подсистемы ввода-вывода и ЦП, которые он требует затратить на извлечение строк. Получение доступа к необходимым строкам означает выполнение Oracle считывания блоков базы данных из файловой системы в буферный пул. Связанное с этим потребление ресурсов подсистемы ввода-вывода является наиболее дорогостоящим этапом выполнения SQL-оператора, поскольку предусматривает чтение с диска. В прежних версиях Oracle можно было выбирать между режимом оптимизации по синтаксису (rule-based) и режимом оптимизации по стоимости (cost-based). В режиме оптимизации по синтаксису применяется эвристический метод для осуществления выбора среди нескольких альтернативных путей доступа с помощью определенных правил. Всем путям доступа присваивался ранг, после чего из них выбирался тот, ранг у которого оказывался самым низким. Операции с более низким рангом обычно выполнялись быстрее операций с высоким рангом. Например, стоимость запроса, в котором для поиска строки используется идентификатор ROWID, будет равна 1. Этого следует ожидать, потому что выявление строки с помощью идентификатора ROWID, т.е. подобного указателя механизма Oracle, является самым быстрым способом для обнаружения строки. Стоимость запроса, в котором используется операция полного сканирования таблицы, с другой стороны, будет составлять 19, что является самым высоким из возможных значений стоимости при оптимизации по синтаксису. Режим оптимизации по стоимости практически всегда работает лучше прежнего режима оптимизации по синтаксису, поскольку, помимо всего прочего, предусматривает принятие во внимания и самых недавних статистических данных по объектам базы данных.