Using Join Paths

В этой практике вы исследуете различные пути доступа, которые может использовать оптимизатор, и сравните их. У вас есть возможность исследовать различные сценарии, каждый из которых самодостаточен.

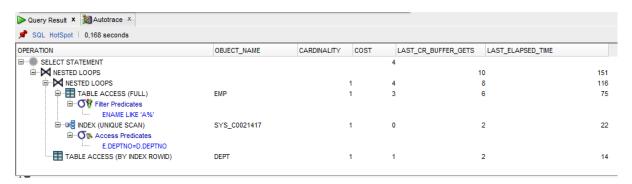
Nested Loops Join Path

Метод соединения вложенного цикла хорошо работает с очень маленькими таблицами и большими таблицами с индексами в столбце соединения или с одной таблицей, которая создает небольшой источник строк. Метод вложенного цикла будет создавать объединенные строки, как только будет найдено совпадение, поэтому иногда это предпочтительный метод для цели доступа FIRST_ROWS.

select ename, e.deptno, d.deptno, d.dname from emp e, dept d where e.deptno = d.deptno and ename like 'A%';

Оптимизатор по умолчанию выбирает метод объединения вложенного цикла для этого запроса. Стоимость невысокая. Это ожидается, потому что обе таблицы в запросе очень малы.

Для Autotrace получим:



А. Что получили? (анализ плана выполнения запроса)

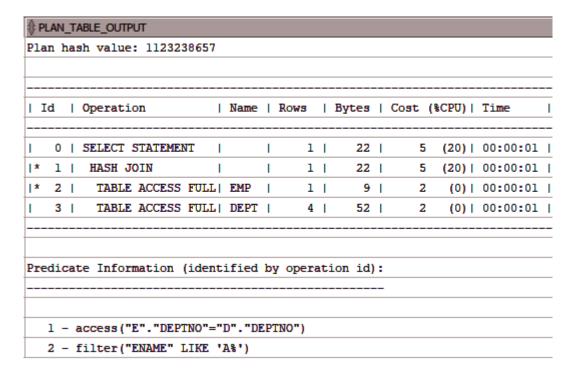
Ic	i	 I	O _I	peration		I	Name	 I	Rows	 I	Bytes	1	Cost	(%CPU)	Time
	0			ELECT STATEME	 Int	 I		 I	1	 I	22		3	(0)	00:00:0
	1	Ī	1	NESTED LOOPS		i		Ť		Ť		Ť		1	
	2	ı		NESTED LOOPS	5	1		П	1	Т	22	ī	3	(0)	00:00:0
*	3	ı		TABLE ACCES	S FULL	- 1	EMP	- 1	1	Т	9	ī	2	(0)	00:00:0
*	4	ı		INDEX UNIQU	JE SCAN	- 1	DEPT_	PK	1	Т		ī	0	(0)	00:00:0
	5			TABLE ACCESS	BY INDEX	ROWID	DEPT	1	1	1	13	1	1	(0)	00:00:0
rec	li	ca	te	Information	(identifi	ed by o	perati	on i	i):						

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%';



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



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

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%';



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

∯ PL	AΝ	U	ABLE_OUTPUT											
Pla	n	ha	sh value: 2125045483											
							_							
I	i	ı	Operation	- 1	Name		ī	Rows	ī	Bytes	Ī	Cost	(%CPU)	Time
							_							
I	0	-	SELECT STATEMENT	- 1			Ī	1	1	22	1	5	(20)	00:00:01
I	1	- 1	MERGE JOIN	- 1			Ī	1	1	22	Ī	5	(20)	00:00:01
I	2	ı	TABLE ACCESS BY INDEX	ROWID	DEPT		ī	4	ī	52	ī	2	2 (0) 1	00:00:01
I	3	ī	INDEX FULL SCAN	- 1	DEPT	PK	ī	4	ī		ī	1	(0)	00:00:01
*	4	ı	SORT JOIN	I			ī	1	ī	9	ī	3	3 (34)	00:00:01
*	5	ı	TABLE ACCESS FULL	I	EMP		Ī	1	ī	9	ī	2	2 (0) 1	00:00:01
							_							
Pred	li	ca	te Information (identifie	d by o	perat:	ion	i	i):						
	_	_					_							
	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 C.

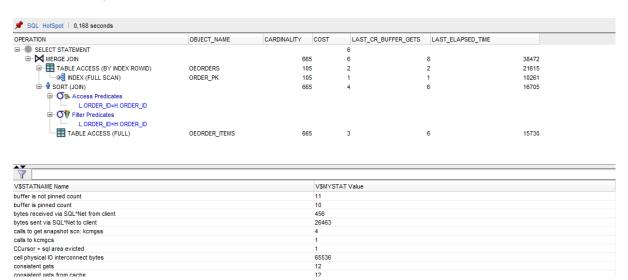
Merge Join

Объединение слиянием - это метод объединения общего назначения, который может работать, когда другие методы не могут. Соединение слиянием должно отсортировать каждый из двух источников строк перед выполнением соединения. Поскольку оба источника строк должны быть отсортированы до того, как будет возвращена первая соединенная строка, метод объединения слиянием не очень подходит для использования с целью FIRST_ROWS.

Для выполнения примеров этого раздела используются таблицы схемы **OE** (Order Entry). Файлы для создания таблиц и их наполнения данными прилагаются.

Выполним Autotrace для запроса:

SELECT * FROM OEorders h, OEorder_items l WHERE l.order_id =
h.order_id;



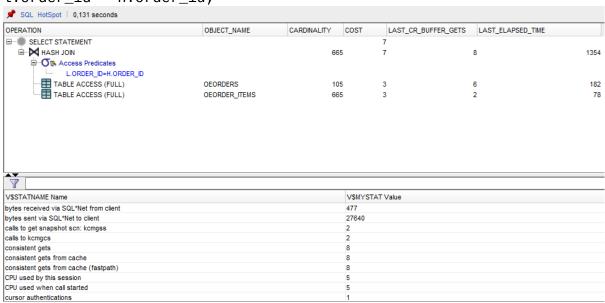
D. **Что получили?** (анализ плана выполнения запроса)

Id		0pe1	ration	1		- 1	Name	1	Rows	1	Bytes	Cost	(%CPU)	Time
()	SELE	ECT ST	ATEMENT		1		1	665	1	105K	3	(0)	00:00:01
	L	NES	STED I	.00PS		- 1		Т		ī	1		- 1	
:	2	NE	ESTED	LOOPS		- 1		-1	665	ī	105K	3	(0)	00:00:01
	3	1	TABLE	ACCESS FU	LL	1	OEORDER_ITEMS	П	665	ī	43225	3	(0)	00:00:01
k i	1]	INDEX	UNIQUE SC	AN	1	ORDER_PK	Т	1	ī	1	0	(0)	00:00:01
	5	T7	ABLE A	CCESS BY	INDEX RO	WID	OEORDERS	Т	1	ī	97	0	(0)	00:00:01
red	ĹC	ite Ir	nforma	tion (ide	ntified l	by op	eration id):							
_		20000	99 / "T."	"."ORDER I	D"="H" "/	חשחשם	וייתד							

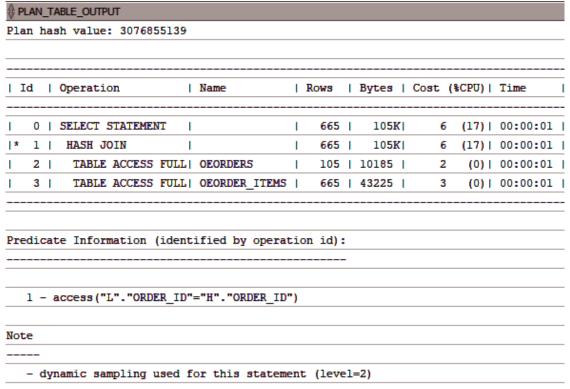
dynamic sampling used for this statement (level=2)

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;



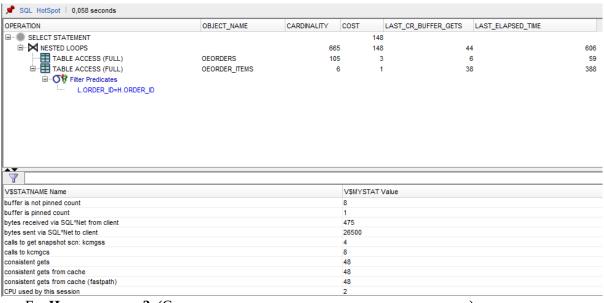
Е. Что получили? (Сравнить стоимости и затраты на чтение данных)



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

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;



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

Ιd		1	Operation		Name	_	Rows	_	Bytes	Cost	(&CPII)	Time
		_		_		_		_				
	0	ī	SELECT STATEMENT	ī		ī	665	ī	105K	3	(0)	00:00:0
	1	Ī	NESTED LOOPS	Ī		1		1	- 1		- 1	
	2	Ī	NESTED LOOPS	Ī		1	665	1	105K)	3	(0)	00:00:0
	3	Ī	TABLE ACCESS FULL	Ī	OEORDER_ITEMS	1	665	1	43225	3	(0)	00:00:0
k	4	Ī	INDEX UNIQUE SCAN	Ī	ORDER_PK	ī	1	ī	- 1	0	(0)	00:00:0
	5	ī	TABLE ACCESS BY INDEX ROWID	ŊΙ	OEORDERS	ī	1	ī	97	0	(0)	00:00:0
	_	_		-		-		-				
red	ic	at	te Information (identified by	o	eration id):							
	_	_		_								
			access("L"."ORDER ID"="H"."ORD									

dynamic sampling used for this statement (level=2)

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

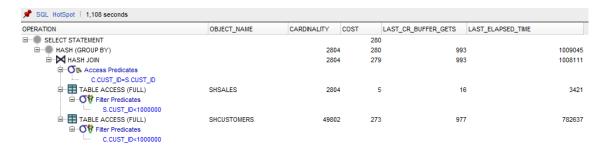
Hash Join

Метод хэш-соединения очень хорошо работает с большими источниками строк и с источниками строк, где один источник строки меньше. Хеш-соединение создает хеш-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 хеш-значение в первой. Поскольку строки объединяются, как только обнаруживается совпадение, метод хеш-соединения также является предпочтительным для цели FIRST_ROWS.

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

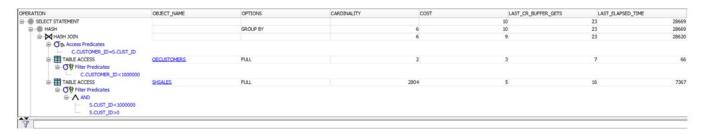
Рекомендуется для приведения в соответствие Id Customer выполнить изменения данных в таблице SHSales (update shsales set cust_id=cust_id+30000;)

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;</pre>



A		
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 kemges	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)

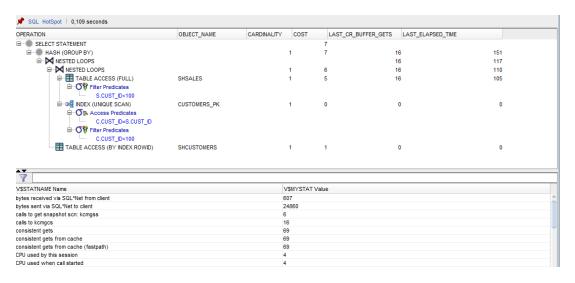


Выполним тот же запрос еще раз, но с использованием подсказки для принудительного использования метода вложенных циклов.

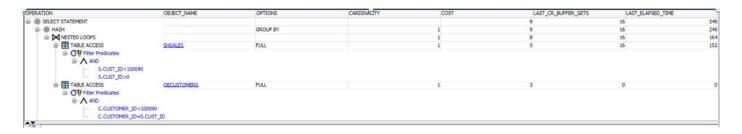
```
SELECT /*+ USE_NL(c s) */c.cust_first_name,
c.cust_last_name,c.cust_id, COUNT(s.prod_id)FROM shcustomers c,
shsales s
where c.cust_id = s.cust_id and c.cust_id < 100090
GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id;</pre>
```

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.



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

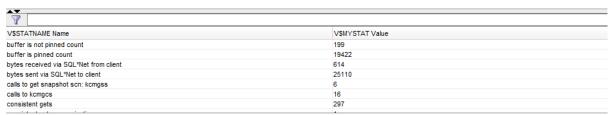


При использовании данного запроса, Cost уменьшился, по сравнению с запросом G.

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</pre>

₱ SQL HotSpot 0,466 seconds							
OPERATION	OBJECT_NAME	CARDINALITY	COST		LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	
□ SELECT STATEMENT				218			
⊞ HASH (GROUP BY)		210	0	218	24	4	358539
· ■ MERGE JOIN		21	0	217	24	4	358010
TABLE ACCESS (BY INDEX ROWID)	SHCUSTOMERS	981	0	211	22	3	44682
□···□··□·□□ INDEX (RANGE SCAN)	CUSTOMERS_PK	981	0	22	3	1	14297
Ē ··· O ™ Access Predicates							
C.CUST_ID<100090							
		210	0	6	10	6	250024
Access Predicates							
C.CUST_ID=S.CUST_ID							
- O Filter Predicates							
C.CUST_ID=S.CUST_ID							
⊞ TABLE ACCESS (FULL)	SHSALES	210	0	5	10	6	349
ĒO♥ Filter Predicates							
S.CUST_ID<100090							



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



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

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

Oracle применятся оптимизатор по стоимости (Cost-BasedOptimizer — 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 указателям 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 статистически данных по объектам базы данных.