

Oracle: малоизвестные факты. Часть 3

Нужно ли индексировать маленькие таблицы?

Сергей Маркеленков,
компания РДТЕХ

<http://www.rdtex.ru>,
markelenkov@rdtex.ru



Очень часто, читая курсы по оптимизации производительности, я задаю слушателям такой вопрос: “давайте представим себе маленькую таблицу (обычную, *heap-table*), такую, что все ее строки помещаются в один блок данных. Предположим для простоты, что в этой таблице есть столбец ID с ограничением NOT NULL, и приложение очень часто выполняет поиск в этой таблице по условию WHERE ID=:B1. Причем значения ID уникальны или почти уникальны. Как вы думаете, нужно или нет в данном случае строить индекс по столбцу ID?”

Как правило, после недолгих раздумий все отвечают, что не нужно. Разумеется, все понимают, что видимо где-то здесь таится подвох, раз я задаю такой вопрос. Но в наши головы уже крепко вбито правило: полный просмотр небольших таблиц – более эффективный способ поиска данных, чем поиск по индексу. Да, обычно это именно так, если выбирается большой процент данных таблицы. Но если предикату запроса удовлетворяет небольшое число строк таблицы, то поиск по индексу чаще всего будет эффективней. Даже если все строки таблицы помещаются в один единственный блок.

Дело в том, что, когда идет речь о настройке производительности, считается, что чем меньше выполняется логических чтений при выполнении SQL-команды, тем это эффективней. В подавляющем большинстве случаев такой подход оправдан. Но все дело в том, что одно логическое чтение другому – рознь. То есть, их цена с точки зрения потребляемых ресурсов может разительно отличаться. И отличие заключается в том, сколько работы (с точки зрения потребления процессорного времени) необходимо проделать серверному процессу уже *внутри* блока, когда блок найден в буферном кэше, на него установлен *pin* в нужном режиме и выполнены все другие необходимые перед этим подготовительные действия.

Кроме этого, не каждое обращение серверного процесса к блоку является логическим чтением. Если серверный процесс уже установил *pin* на какой-то блок, то он при определенных обстоятельствах может его какое-то время не снимать, если есть высокая вероятность очень скорого

повторного обращения к этому же блоку при выполнении текущей фазы SQL-команды (курсора). Таким образом, отпадает необходимость в неоднократном поиске одних и тех же блоков в буферном кэше, во взятии и освобождении защепок *cache buffers chains*, *cache buffer handles*, формировании и удалении *State Objects (SO)* для блоков данных и т.д. То есть, нет нужды выполнять огромную часть работы, которая, фактически, и является логическим чтением.

Целью данной статьи является демонстрация отличия в объеме потребляемых ресурсов при выполнении одинакового числа логических чтений. Поэтому детали, касающиеся уменьшения в некоторых случаях числа логических чтений, в этой статье я рассматривать не буду. Описание понятия *buffer pin* с примерами даны в (1), а понятие логического чтения и краткое описание параметров инициализации *_cursor_db_buffers_pinned*, *_db_handles*, *_db_handles_cached*, влияющих на него – в (2). В обоих источниках можно почитать про статистику *buffer is pinned count*, *buffer is not pinned count* и *buffer is not pinned count*.

Для проведения экспериментов я создал табличное пространство с размером блока 8k, а в нем - таблицу TEST, которую наполнил тестовыми данными:

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.8.0 - Production
With the Partitioning, OLAP and Oracle Data Mining
options
JServer Release 9.2.0.8.0 - Production
```

```
SQL> create tablespace testtbs datafile 'c:\test.dbf' size
16m reuse
2 segment space management manual blocksize 8k;
```

```
Tablespace created.
```

```
SQL> create table test(id number not null,c1 char(1),c2
char(1))
2 pctfree 0 tablespace testtbs;
```

Table created.

```
SQL> begin
2 for i in 1..256 loop
3 insert into test values(i,'1','2');
4 insert into test values(513-i,'1','2');
5 end loop;
6 commit;
7 end;
8 /
```

PL/SQL procedure successfully completed.

SQL>

Убедимся, что все строки поместились в один блок:

```
SQL> select dbms_rowid.rowid_relative_fno(rowid) file#,
2 dbms_rowid.rowid_block_number(rowid) block#,
count(*)
3 from test
4 group by dbms_rowid.rowid_relative_fno(rowid),
5 dbms_rowid.rowid_block_number(rowid);
```

FILE#	BLOCK#	COUNT(*)
9	10	512

SQL>

Убедимся, что строка с ID=128 находится приблизительно в середине блока (ROWID нумеруются с нуля, поэтому строка – 255-я в блоке):

```
SQL> select dbms_rowid.rowid_row_number(rowid) from
test where id=128;
```

DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)
254

SQL>

Таким образом, все строки таблицы TEST поместились в один блок данных. Значения столбца ID для упрощения тестирования уникальны. Логическая структура блока будет приблизительно следующей:

Блок таблицы TEST

Заголовок блока			
Номер строки в блоке	ID	C1	C2
1	1	'1'	'2'
2	512	'1'	'2'
3	2	'1'	'2'
4	511	'1'	'2'
...
255	128	'1'	'2'
...
510	258	'1'	'2'
511	256	'1'	'2'
512	257	'1'	'2'

А теперь, предположим, что у нас имеется часто выполняемая команда

```
SELECT C2 FROM TEST WHERE ID=:B1
```

Легко понять, что придется проделать серверному процессу для поиска нужных строк в блоке. Если нет индекса по столбцу ID, ему придется сканировать весь блок данных, то есть, выполнить 512 операций сравнения значений ID и :B1. Строки в блоках обычных, организованных в виде кучи таблиц обычно не упорядочены. А если они даже и упорядочены, то Oracle это никак не учитывает, так как это исключение, а не правило, и полагаться на это однозначно нельзя. Поэтому он вынужден просмотреть всю таблицу (в нашем случае – один блок) целиком.

Конечно, если предикату WHERE ID=:B1 удовлетворяет большой процент строк, то такой способ поиска (полный просмотр всех блоков) скорей всего будет весьма эффективным, причем тем эффективней, чем выше этот процент, чем больше блоков занимает таблица и чем хуже фактор кластеризации (*clustering factor*) индекса. Но если значения ID уникальны или почти уникальны, то здесь поиск по индексу предпочтительней. Почему? Давайте сначала рассмотрим структуру блока индекса, построенного командой

```
CREATE INDEX TEST_I ON TEST(ID)
```

Этот индекс тоже будет занимать один блок:

Блок таблицы TEST_I

Заголовок блока		
Номер строки в блоке	ID	ROWID
1	1	rowid1
2	2	rowid2
3	3	rowid3
4	4	rowid4
...	...	
128	128	rowid128
...	...	
510	510	rowid510
511	511	rowid511
512	512	rowid512

Так как строки в блоках индекса упорядочены, легко понять примерный алгоритм поиска строки с нужным ID в блоке индекса:

- зная общее число строк в блоке индекса, читаем “среднюю” строку в блоке (в нашем случае это 256 строка)
- если нашли нужный ID, то
 - в случае уникального индекса идем по ROWID в блок таблицы, причем никакого поиска строк внутри блока таблицы уже выполнять не надо – ROWID указывает на конкретную строку таблицы
 - в случае неуникального индекса сканируем предыдущие и последующие строки индекса, до тех пор, пока в обоих направлениях не наткнемся на другое значение ID или на начало/конец блока. Для каждой найденной в таком диапазоне строки индекса нужно будет прочитать по ROWID строку таблицы
- если не нашли нужный ID, то
 - если искомый ID больше, чем в “средней” строке, значит искомая строка может быть только во второй (большей) половине диапазона
 - если искомый ID меньше, чем в “средней” строке, значит искомая строка может быть только в первой (меньшей) половине диапазона
 - если мы при поиске наткнулись на начало/конец блока, значит искомой строки в блоке нет
- повторяем цикл заново, каждый раз логически разделяя диапазон поиска пополам

Таким образом, в нашем случае, чтобы найти строку, удовлетворяющую условию WHERE ID=:B1, нужно в худшем случае сделать 9 операций (29=512) сравнения: B1 и ID в строках индекса, и по ROWID уже напрямую обратиться к нужной строке в блоке таблицы. Почувствуйте разницу: 512 против 9 операций сравнения в случае уникальности ID и 9+число_строк_с_искомым_ID+2 операций сравнения в случае не уникальности ID.

Давайте проведем набор тестов. Каждый раз, немного меняя начальные условия, мы будем искать строку с ID=128 сначала при помощи полного просмотра таблицы TEST, а потом – по индексу. Для анализа я буду включать трассировку, обрабатывать полученные трассировочные файлы утилитой tkprof и сравнивать результаты. Планы выполнения будут в каждом случае идентичны – полный просмотр таблицы или поиск по индексу, поэтому я их приводить не буду. Команда SELECT будет в цикле выполняться по 100 тысяч раз.

1. Поиск всех строк, удовлетворяющих условию

В данном случае при полном просмотре мы просматриваем все 512 строк таблицы.

Без индекса:

```
SQL> exec dbms_stats.gather_table_stats(user,tabname=>
'TEST',cascade=>true);

PL/SQL procedure successfully completed.

SQL> alter session set sql_trace=true;

Session altered.

SQL> declare
  2 i pls_integer;
  3 l_c2 char(1);
  4 begin
  5 for i in 1..100000 loop
  6   select c2 into l_c2 from test where id=128;
  7 end loop;
  8 end;
  9 /

PL/SQL procedure successfully completed.

SQL>
```

Результаты обработки трассировочного файла утилитой tkprof:



Современные технологии

Oracle: малоизвестные факты. Часть 3
Нужно ли индексировать маленькие таблицы?

```
SELECT C2
FROM
TEST WHERE ID=128
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	100000	1.59	1.52	0	0	0	0
Fetch	100000	17.29	17.10	0	300000	0	100000
total	200001	18.89	18.63	0	300000	0	100000

Создаем индекс и повторяем тест:

```
SQL> create index test_i on test(id) compute statistics;
```

Index created.

```
SQL> declare
2 i pls_integer;
3 l_c2 char(1);
4 begin
5 for i in 1..100000 loop
6 select c2 into l_c2 from test where id=128;
7 end loop;
8 end;
9 /
```

PL/SQL procedure successfully completed.

```
SQL>
```

Обратите внимание на то, что число логических чтений (сумма столбцов query и current) в обоих случаях совпадает. Но насколько сильно при этом отличается процессорное время, ушедшее на фазе Fetch. Причина именно в том, что в первом случае пришлось выполнить намного больше операций сравнения ID со значением 128. Причем в этом тесте не принципиально, какое число указано в условии WHERE: при полном просмотре выполняется сравнение всех строк таблицы от первой до последней; при поиске по индексу может просто “повезти” (когда при первом-втором сравнении натываемся на искомую строку) или нет, но разница не существенна.

Итого – разница во времени Fetch приблизительно в 11 раз.

```
SELECT C2
FROM
TEST WHERE ID=128
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	100000	1.96	1.78	0	0	0	0
Fetch	100000	2.06	1.55	0	300000	0	100000
total	200001	4.03	3.34	0	300000	0	100000

2. Поиск самой первой строки, удовлетворяющей условию

Ситуацию с полным просмотром таблицы можно в нашем случае улучшить в среднем в 2 раза. Если приложение требует только самую первую строку (или знает, что все ID в таблице уникальны), то можно остановиться после нахождения и выборки первой строки. Для этого обязательно нужно или установить *fetch size* равный одной строке, или использовать опцию `OCI_EXACT_FETCH` у функции `OCIStmtExecute()`, или любую другую возможность прервать фазу *Fetch* сразу после выборки первой строки. Я буду использовать условие `rownum<=1`. В данном случае при полном просмотре мы просматриваем примерно половину строк таблицы, что соответствует среднестатистическому показателю. Разумеется, при поиске по `ID=1` мы бы сразу нашли искомую строку (она в блоке первая), при поиске по `ID=257` пришлось бы выполнить 512 операций сравнения. Но в среднем придется всегда сканировать половину блока.

Без индекса:

```
SQL> drop index test_i;

Index dropped.

SQL> declare
  2 i pls_integer;
  3 l_c2 char(1);
  4 begin
  5 for i in 1..100000 loop
  6   select c2 into l_c2 from test where id=128 and row-
  7   num<=1;
  8 end loop;
  9 /

PL/SQL procedure successfully completed.

SQL>
```

```
SELECT C2
FROM
TEST WHERE ID=128 AND ROWNUM<=1
```

<i>call</i>	<i>count</i>	<i>cpu</i>	<i>elapsed</i>	<i>disk</i>	<i>query</i>	<i>current</i>	<i>rows</i>
<i>Parse</i>	1	0.00	0.00	0	0	0	0
<i>Execute</i>	100000	1.79	1.56	0	0	0	0
<i>Fetch</i>	100000	9.46	9.70	0	300000	0	100000
<i>total</i>	200001	11.26	11.27	0	300000	0	100000

С индексом:

```
SQL> create index test_i on test(id) compute statistics;

Index created.

SQL> declare
  2 i pls_integer;
  3 l_c2 char(1);
  4 begin
  5 for i in 1..100000 loop
  6   select c2 into l_c2 from test where id=128 and row-
  7   num<=1;
  8 end loop;
  9 /

PL/SQL procedure successfully completed.

SQL>
```



Современные технологии

Oracle: малоизвестные факты. Часть 3
Нужно ли индексировать маленькие таблицы?

```
SELECT C2
FROM
TEST WHERE ID=128 AND ROWNUM<=1
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	100000	2.07	1.90	0	0	0	0
Fetch	100000	1.62	1.57	0	300000	0	100000
total	200001	3.70	3.48	0	300000	0	100000

Как видим, при поиске по индексу практически никакой разницы нет. Зато при полном просмотре таблицы фаза Fetch, как и предполагалось, выполнялась примерно в 2 раза быстрее. Но все равно поиск по индексу более чем в 6 раз быстрее.

Что интересно, даже в самом лучшем случае при ID=1 полный просмотр выполняется дольше поиска по индексу:

```
SQL> drop index test_i;
```

Index dropped.

```
SQL> declare
2 i pls_integer;
3 l_c2 char(1);
4 begin
5 for i in 1..100000 loop
6 select c2 into l_c2 from test where id=1 and row-
num<=1;
7 end loop;
8 end;
9 /
```

PL/SQL procedure successfully completed.

```
SQL>
```

```
SELECT C2
FROM
TEST WHERE ID=1 AND ROWNUM<=1
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	100000	1.51	1.59	0	0	0	0
Fetch	100000	2.32	2.25	0	300000	0	100000
total	200001	3.84	3.85	0	300000	0	100000

То есть, при поиске по индексу на фазу Fetch ушло около 1.57 секунды, а при полном просмотре даже в идеальном варианте – 2.25 секунды.

А если, учитывая уникальность ID, построить по этому столбцу уникальный индекс, можно получить дополнительный выигрыш в производительности (и еще небольшой – в размере индекса):

```
SQL> create unique index test_i on test(id) compute statistics;
```

```
Index created.
```

```
SQL> declare
```

```
2 i pls_integer;
```

```
3 l_c2 char(1);
```

```
4 begin
```

```
5 for i in 1..100000 loop
```

```
6 select c2 into l_c2 from test where id=1 and rownum<=1;
```

```
7 end loop;
```

```
8 end;
```

```
9 /
```

```
PL/SQL procedure successfully completed.
```

```
SQL>
```

При поиске по уникальному индексу фаза Fetch теперь выполняется за 1.15 секунды вместо 1.57. А если еще убрать за ненадобностью в этом случае условие `rownum<=1`, то можно еще сэкономить около 0.05 секунды.

Таким образом, даже в идеальном случае полный просмотр таблицы при поиске одной строки по уникальному ID оказывается медленнее индексного поиска.

```
SELECT C2
```

```
FROM
```

```
TEST WHERE ID=1 AND ROWNUM<=1
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	100000	1.76	1.61	0	0	0	0
Fetch	100000	1.34	1.15	0	300000	0	100000
total	200001	3.10	2.77	0	300000	0	100000

3. Поиск всех строк, удовлетворяющих условию, когда ID – не первый столбец в таблице

Пока что я рассматривал случай, когда столбец, по которому выполняется поиск, расположен первым в каждой строке таблицы. Если же он не первый, то появляются дополнительные расходы процессорного времени. Как известно, у каждой строки (или куска строки – `row piece`, что в нашем случае одно и то же) обычной таблицы есть 3 байта заголовка, после которых идут пары: 1 или 3 байта длины, значение столбца, 1 или 3 байта длины, значение столбца и т.д. Значения NULL хранятся одним байтом, “хвостовые” NULL-столбцы и вовсе не хранятся.

Таким образом, если создать таблицу TEST с другим порядком столбцов:

```
CREATE TABLE TEST(C1 CHAR(1), C2 CHAR(1), ID NUMBER NOT NULL)
```

то каждая строка в блоке будет выглядеть так:

Заголовок 3 байта	1 байт длины столбца C1	Значение столбца C1	1 байт длины столбца C2	Значение столбца C2	1 байт длины столбца ID	Значение столбца ID

Теперь при полном просмотре таблицы для каждой сравниваемой строки потребуется выполнить дополнительные 2 операции суммирования длин столбцов C1 и C2. Только после этого мы доберемся до значения столбца ID и будем его сравнивать с искомым значением. Конечно, операция суммирования целых чисел – не самая ресурсоемкая. Но, как известно, курочка по зернышку клюет :)

В индексе по столбцу ID никаких дополнительных суммирований производить не нужно – вне зависимости от числа столбцов и их порядка в таблице в индексе он будет всегда первым столбцом .

Давайте повторим первый тест, увеличив число итераций до 1 млн., а затем – то же самое, но переставив столбцы в таблице.



* Современные технологии

Oracle: малоизвестные факты. Часть 3
Нужно ли индексировать маленькие таблицы?

Столбец ID - первый

```
SQL> drop table test;

Table dropped.

SQL> create table test(id number not null,c1 char(1),c2
char(1))
  2 pctfree 0 tablespace testtbs;

Table created.

SQL> begin
  2 for i in 1..256 loop
  3 insert into test values(i,'1','2');
  4 insert into test values(513-i,'1','2');
  5 end loop;
  6 commit;
  7 end;
  8 /

PL/SQL procedure successfully completed.

SQL> declare
  2 i pls_integer;
  3 l_c2 char(1);
  4 begin
  5 for i in 1..1000000 loop
  6 select c2 into l_c2 from test where id=128;
  7 end loop;
  8 end;
  9 /

PL/SQL procedure successfully completed.

SQL>
```

```
SELECT C2
FROM
TEST WHERE ID=128
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1000000	16.71	14.99	0	0	0	0
Fetch	1000000	173.48	170.69	0	3000000	0	1000000
total	2000001	190.20	185.70	0	3000000	0	1000000

Столбец ID - третий

```
SQL> drop table test;

Table dropped.

SQL> create table test(c1 char(1),c2 char(1),id number not
null)
  2 pctfree 0 tablespace testtbs;

Table created.

SQL> begin
  2 for i in 1..256 loop
  3 insert into test values('1','2',i);
  4 insert into test values('1','2',513-i);
  5 end loop;
  6 commit;
  7 end;
  8 /

PL/SQL procedure successfully completed.

SQL> declare
  2 i pls_integer;
  3 l_c2 char(1);
  4 begin
  5 for i in 1..1000000 loop
  6 select c2 into l_c2 from test where id=128;
  7 end loop;
  8 end;
  9 /

PL/SQL procedure successfully completed.

SQL>
```

```
SELECT C2
FROM
TEST WHERE ID=128
```

<i>call</i>	<i>count</i>	<i>cpu</i>	<i>elapsed</i>	<i>disk</i>	<i>query</i>	<i>current</i>	<i>rows</i>
<i>Parse</i>	1	0.00	0.00	0	0	0	0
<i>Execute</i>	1000000	16.76	15.43	0	0	0	0
<i>Fetch</i>	1000000	174.98	174.64	0	3000000	0	1000000
<i>total</i>	2000001	191.75	190.08	0	3000000	0	1000000

Разница не существенна, но она есть. И она будет тем больше, чем больше столбцов в таблице предшествуют столбцу ID. При поиске по индексу никакой существенной разницы нет:



Современные технологии

Oracle: малоизвестные факты. Часть 3

Нужно ли индексировать маленькие таблицы?

```
SQL> create index test_i on test(id) compute statistics;
```

Index created.

```
SQL> declare
  2 i pls_integer;
  3 l_c2 char(1);
  4 begin
  5 for i in 1..100000 loop
  6   select c2 into l_c2 from test where id=128;
  7 end loop;
  8 end;
  9 /
```

PL/SQL procedure successfully completed.

```
SQL>
```

```
SELECT C2
FROM
TEST WHERE ID=128
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	100000	1.96	1.79	0	0	0	0
Fetch	100000	1.62	1.56	0	300000	0	100000
total	200001	3.59	3.36	0	300000	0	100000

- очень небольшое число строк в таблице (скажем, в пределах 10-20 строк) – в этом случае создание индекса производительность вряд ли повысит, но внесет лишние проблемы в сопровождение схемы данных, работу оптимизатора, повысит нагрузку на словарь данных и т.д. Разумеется, если у таблицы есть ограничение первичного или уникального ключа, индекс придется создать и сопровождать
- наличие индекса приведет к увеличению времени выполнения DML-команд
- мигрировавшие строки ухудшают производительность индексного доступа
- полные просмотры сжатых с опцией COMPRESS таблиц еще более ресурсоемки, в этом случае поиск по индексу еще больше повысит производительность
- не всегда возможен поиск по индексу – помним о NULL-столбцах
- когда таблица занимает несколько блоков, а индекс по столбцу – один блок, эффективность индекса может стать еще больше, а может стать и меньше. Это зависит в первую очередь от фактора кластеризации индекса и от того, какой процент строк таблицы выбирается
- в некоторых случаях все данные можно целиком получить из индекса без обращения к блокам таблицы

Обычно при настройке производительности стараются уменьшить число операций логического чтения (LIO). Соответственно, принято считать, что если у двух разных планов выполнения одной и той же SQL-команды разное число логических чтений, то лучшим является тот план, у которого их меньше. Я хотел показать, что в некоторых случаях стоимость логического чтения блока (в первую очередь с точки зрения потребления процессорного времени) в зависимости от ситуации может очень сильно варьироваться – от нескольких процентов, до нескольких раз и даже десятков раз. И оптимизатор по стоимости (CBO) при выборе плана выполнения старается учитывать данный факт.

Разумеется, перед тем как принять решение индексировать или нет небольшую таблицу, нужно учесть множество факторов. Вот только часть из них:

Поэтому в каждом конкретном случае может понадобиться провести тестирование эффективности индексирования маленьких таблиц.

Цель данной статьи – дать пищу для размышлений, показать, что объем работы, выполняемый серверным процессом внутри блока, может в зависимости от ситуации различаться в разы и даже десятки раз. Поэтому сравнивать эффективность того или иного способа доступа нужно по совокупности факторов. И в некоторых случаях ориентироваться при настройке SQL-команд только на уменьшение числа логических чтений явно не достаточно.

Если же дать краткий ответ на вопрос, поставленный в заглавии статьи, то он будет звучать так: индексуйте даже маленькие справочные таблицы

Список литературы

1. Julian Dyke, “Logical IO”, Presentation

<http://www.juliandyke.com/Presentations/LogicalIO.ppt>

2. Howard Rogers, “What are Buffer Handles”

http://dizwell.com/main/index.php?option=com_jd-wiki&Itemid=170&id=bufferhandles