SQL или PL/SQL? — взгляд с точки зрения производительности

Егор Рогов, AT Consulting (erogov@at-consulting.ru)

При написании больших запросов из нескольких десятков таблиц, связанных нетривиальной логикой, часто возникает желание разбить сложный запрос на несколько более простых. Например, вынести часть запроса в функцию или организовать обработку данных в виде вложенных циклов PL/SQL. При этом мы делаем то, что обычно любим и ценим в процедурном программировании: выполняем декомпозицию кода, упрощаем задачу, повышаем читаемость и т. д. Хорошо это или плохо в контексте базы данных? Попробуем разобраться.

Допущения. Мы будем говорить только о «тяжелых» запросах, которые встречаются в отчетах. Для таких отчетов характерно относительно большое время выполнения (от нескольких минут до нескольких часов — хотя последнее уже говорит о необходимости оптимизации) и невысокая частота запусков (возможно, несколько раз в сутки). Понятно, что к OLTP-запросам подход должен быть совсем другой.

Тестовый пример. Разбираться будем на простом «библиотечном» примере. Мы храним библиографическую информацию о *книгах* (books) и сведения о *писателях* (writers). Книги и писатели связаны отношением «многие ко многим»: писатели является *авторами* (authors) книг.

На уровне таблиц это выглядит следующим образом:[1]

```
create table books(
    book_id        number primary key
  , name           varchar2(100)     -- название книги
  , year_published char(4)           -- год издания
);
create table writers(
    writer_id      number primary key
  , name           varchar2(100)     -- имя писателя
);
```

И промежуточная таблица для связи книг и писателей:

```
create table authors(
    seq_num        number            -- порядковый номер автора
  , book_id        number references books(book_id)
  , writer_id      number references writers(writer_id)
);
```

Данные. Предполагаем, что в число книг достаточно велико (100 000). Писателей тоже много (70 000), хотя и меньше, чем книг. Информация о книгах охватывает 5 лет (с 2000 по 2005), и книги распределены по годам равномерно. Пусть половина книг имеет одного автора, а половина — двух авторов.

Эти конкретные значения взяты исключительно ради определенности. Очевидно, что при других цифрах (и при других настройках базы данных) количественные результаты будут отличаться, но нам важен качественный результат.

Задача 1. Требуется вывести все книги, изданные в 2000 году, с указанием только первого автора, причем, если авторов несколько, то после имени первого автора надо добавить «и др.».

Решение 1.1 (запрос с функцией). Если разработчик привычен к процедурному стилю мышления, у него может возникнуть (вполне резонное) желание вынести вычисление авторов в отдельную функцию:

---

[1] Полностью скрипт для создания объектов и заполнения таблиц тестовыми данными приведен в конце статьи.

```
create or replace function get_authors(p_book_id number)
return varchar2
is
  name varchar2(110);
  name2 varchar2(100);
  cursor c(p_book_id number) is
    select w.name
      from writers w
         , authors a
     where a.book_id = p_book_id
       and w.writer_id = a.writer_id
     order by a.seq_num;
begin
  open c(p_book_id);
  fetch c into name;
  fetch c into name2;
  if c%found then
    name := name||' и др.';
  end if;
  close c;
  return name;
end;
```

Чтобы функция не делала при каждом обращении полное сканирование таблицы authors, очевидно, потребуется индекс:

```
create index a_book_id on authors(book_id);
```

После этого сам запрос можно написать так:

```
select b.name, get_authors(b.book_id)
  from books b
 where b.year_published = '2000';
```

И функция, и запрос получились достаточно простыми и понятными. Теперь нам надо как-то понять, получились ли они хорошими.

Что такое хорошо. Из общих соображений, хороший запрос — это запрос, который работает быстро. В конце концов, именно это нужно пользователям.

Однако для разработчика такое представление довольно опасно.

Во-первых, скорость работы зависит от текущей загруженности и от аппаратной конфигурации сервера. Сегодня программа может работать десять минут, а завтра — пятнадцать. Как в таком случае сравнить два запроса? Или как сравнить два запроса, один из которых работает пять минут на продуктивной среде, а другой — десять минут на тестовой (менее мощной) среде?

Во-вторых, скорость работы запроса может зависеть от состояния кэша. Первый раз выполнение происходит долго (данные с диска считываются и помещаются в кэш), а второй раз — уже быстрее (все необходимое читается непосредственно из оперативной памяти). Но эта только кажущаяся быстрота: пользователи не станут запускать один и тот же отчет подряд несколько раз, так что для них каждый запуск, скорее всего, будет «первым» и долгим.

В-третьих, даже быстрый с виду, но читающий избыточно много данных запрос плох для системы, поскольку чтение из кэша сериализуется с помощью *защелок* и, следовательно, создает проблемы не только себе, но и всем параллельно работающим процессам.[2]

Поэтому нужна более стабильная и надежная характеристика запроса, нежели время, и такой характеристикой является число логических чтений. С одной стороны, число логических чтений не зависит от состояния и конфигурации сервера, с другой — достаточно адекватно отражает нагрузку на базу данных, создаваемую запросом.

Методика измерения. SQL-запросы будем выполнять в SQL*Plus с включенной опцией `autotrace traceonly`; это покажет нам статистику выполнения запроса, в том числе и логические чтения. Для анализа PL/SQL-кода будем включать трассировку (`alter session set events='10046 TRACE NAME CONTEXT FOREVER, LEVEL 1'`) и обрабатывать ее с помощью tkprof. Это позволит нам экспериментально выяснить, как часто вызывается наша функция внутри запроса и сколько логических чтений при этом происходит.

---

[2] Отличная статья о том, чем вредны избыточные логические чтения: Cary Millsap, «Why You Should Focus on LIOs Instead of PIOs» (доступна по адресу http://www.hotsos.com/e-library/, но требуется регистрация).

В теории. Прежде, чем начинать эксперимент, попробуем дать теоретическую оценку числа логических чтений предложенного решения. Эта количественная оценка пригодится в дальнейшем для качественного понимания, на что конкретно тратятся чтения.

Для этого посмотрим план основного запроса:

```
select b.name, get_authors(b.book_id)
  from books b
 where b.year_published = '2000';

-----------------------------------------------------------------------
| Id  | Operation            |  Name       | Rows  | Bytes | Cost  |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT     |             | 20000 |   429K|  2170 |
|*  1 |  TABLE ACCESS FULL   | BOOKS       | 20000 |   429K|  2170 |
-----------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("B"."YEAR_PUBLISHED"='2000')
```

Выполняется полное сканирование таблицы `books` (что логично, так как мы рассчитываем выбрать довольно много — 20% всех записей). На это потребуется как минимум столько логических чтений, сколько блоков в таблице `books`:

```
SQL> select blocks from dba_tables where table_name='BOOKS';

    BLOCKS
----------
     14286
```

Далее для каждой из примерно 20 000 выбранных строк будет вызвана функция `get_authors`. Посмотрим теперь на план выполнения ее запроса:

```
select w.name
  from writers w
     , authors a
 where a.book_id = :p_book_id
   and w.writer_id = a.writer_id
 order by a.seq_num;

--------------------------------------------------------------------------------
| Id  | Operation                     |  Name        | Rows  | Bytes | Cost  |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     2 |    58 |    30 |
|   1 |  SORT ORDER BY                |              |     2 |    58 |    30 |
|   2 |   NESTED LOOPS                |              |     2 |    58 |     5 |
|   3 |    TABLE ACCESS BY INDEX ROWID| AUTHORS      |     2 |    24 |     3 |
|*  4 |     INDEX RANGE SCAN          | A_BOOK_ID    |     2 |       |     1 |
|   5 |    TABLE ACCESS BY INDEX ROWID| WRITERS      |     1 |    17 |     1 |
|*  6 |     INDEX UNIQUE SCAN         | W_WRITER_ID  |     1 |       |       |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("A"."BOOK_ID"=TO_NUMBER(:Z))
   6 - access("W"."WRITER_ID"="A"."WRITER_ID")
```

Выполнение запроса происходит следующим образом:

1. Читается индекс a_book_id и в нем находится rowid записи в таблице `authors`, такой, что `book_id = :p_book_id` (строка 4);
2. По rowid выбирается блок из таблицы `authors` (строка 3);
3. Читается индекс w_writer_id и в нем находится rowid записи в таблице `writers`, такой, что `writer_id` равно полученному на предыдущем шаге значению `a.writer_id` (строка 6);
4. По rowid выбирается блок из таблицы `writers` (строка 5);
5. Шаги 1–4 повторяются, пока индекс `a_book_id` продолжает давать новые rowid (строка 2);
6. Полученные результаты сортируются (строка 1) и возвращается результат (строка 0).

Для чтения индекса нужно спуститься от корня до листового блока, на это потребуется blevel+1 логическое чтение:

```
select index_name,blevel+1 from dba_indexes where index_name in ('A_BOOK_ID','W_WRITER_ID');

INDEX_NAME                      BLEVEL+1
------------------------------ ----------
A_BOOK_ID                               2
W_WRITER_ID                             2
```

Выборка блока данных по найденному rowid занимает ровно одно логической чтение.

Итого на выборку первого значения потребуется 2+1+2+1 = 6 логических чтений.

Число чтений для выборки второго значения (а мы помним, что половина книг имеет двух авторов) оценить несколько сложнее: повторные обращения к индексу могут потребовать переход от текущего листового блока к следующему подходящему. Но допустим, что все нужные rowid сосредоточены в одном блоке, тогда на выборку второго значения потребуется 1+1+1+1 = 4 логических чтений.

Всего на 20 000 записей потратится примерно 14 286+20 000×6+10 000×4 ≈ 174 000 чтений.

НА ПРАКТИКЕ. Проверяем:

```
SQL> set autotrace traceonly
SQL> select b.name, get_authors(b.book_id)
  from books b
 where b.year_published = '2000';

20060 строк выбрано.

Статистика
----------------------------------------------------------
      70192  recursive calls
          0  db block gets
     176051  consistent gets
      24195  physical reads
          0  redo size
     614018  bytes sent via SQL*Net to client
       9636  bytes received via SQL*Net from client
       1339  SQL*Net roundtrips to/from client
      20060  sorts (memory)
          0  sorts (disk)
      20060  rows processed
```

Получили 176 051 логическое чтение, что довольно хорошо согласуется с теоретической оценкой.

Возникает вопрос: много это или нормально? Трудно предложить какой-либо универсальный критерий, но стоит насторожиться, если общее число логических чтений существенно превышает суммарный размер (в блоках) всех таблиц, участвующих в запросе.

```
SQL> select sum(blocks) from dba_tables where table_name in ('BOOKS','WRITERS','AUTHORS');

    BLOCKS
----------
     24656
```

Попробуем теперь переписать запрос так, чтобы он не использовал функцию, и посмотрим, что из этого получится.

РЕШЕНИЕ 1.2 (ЧИСТЫЙ SQL). Переписать запрос можно, например, используя аналитическую функцию:

```
SQL> set autotrace traceonly
SQL> select b.name
     , min(w.name) keep (dense_rank first order by a.seq_num) ||
       case when count(*) > 1 then ' и др.' else '' end
  from books b
     , writers w
     , authors a
 where b.year_published = '2000'
   and a.book_id = b.book_id
   and w.writer_id = a.writer_id
 group by b.name;

20060 строк выбрано.
```

```
Статистика
----------------------------------------------------------
          0  recursive calls
          0  db block gets
      24823  consistent gets
      24662  physical reads
          0  redo size
     614115  bytes sent via SQL*Net to client
       9635  bytes received via SQL*Net from client
       1339  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
      20060  rows processed
```

Всего 24 823 логических чтения, это в семь раз лучше, чем у запроса с функцией!

Почему так получилось? Посмотрим на план выполнения:

```
-------------------------------------------------------------------------------
| Id  | Operation            |  Name        | Rows  | Bytes |TempSpc| Cost  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              | 20000 |   996K|       |  4033 |
|   1 |  SORT GROUP BY       |              | 20000 |   996K|  2960K|  4033 |
|*  2 |   HASH JOIN          |              | 30000 |  1494K|       |  3803 |
|*  3 |    HASH JOIN         |              | 30000 |   996K|       |  2252 |
|*  4 |     TABLE ACCESS FULL| BOOKS        | 20000 |   429K|       |  2170 |
|   5 |     TABLE ACCESS FULL| AUTHORS      |   150K|  1757K|       |    58 |
|   6 |    TABLE ACCESS FULL | WRITERS      | 70000 |  1162K|       |  1519 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("W"."WRITER_ID"="A"."WRITER_ID")
   3 - access("A"."BOOK_ID"="B"."BOOK_ID")
   4 - filter("B"."YEAR_PUBLISHED"='2000')
```

Как мы видим, здесь вместо доступа по индексу оптимизатор счел, что лучше будет прочитать все три таблицы полностью, и соединить их с помощью хэширования. Число логических чтений в этом случае будет порядка суммы блоков всех таблиц.

Выводы? Разделив запрос на две части, мы фактически предопределили план выполнения «большого» запроса: сначала должна прочитаться таблица `books`, затем для каждой полученной строки должна выполниться выборка из `writers` и `authors`, которую мы поместили в функцию. Для запроса в функции оптимизатору не остается ничего другого, как применить доступ по индексу, но, как мы увидели, это далеко не самый оптимальный способ. В первом варианте решения только на доступ к блокам индекса ушло примерно 20 000×4+10 000×2 = 100 000 логических чтений, то есть *больше половины* всех чтений, а ведь это не что иное, как накладные расходы — сами по себе блоки индекса (в нашем случае) не несут полезной информации.[3] Но и оставшаяся часть далеко не оптимальна, поскольку один и тот же блок данных, содержащий несколько записей, может перечитываться при доступе по индексу несколько раз (тем чаще, чем больше фактор кластеризации индекса[4]).

И это не исключение и не специально подобранный пример. В реальности для «тяжелых» отчетов, читающих большие объемы данных, полное сканирование таблиц очень часто оказывается существенно более экономичным способом выполнения.

Как уже говорилось, конкретные цифры, полученные в этом примере, на самом деле не важны — слишком от многих факторов они зависят. Но надо понимать, что чем сложнее внутренний запрос, тем больше будет происходить избыточных чтений.

Посмотрим еще два варианта решения той же задачи, которые, хотя внешне сильно отличаются от решения с функцией, по сути являются тем же самым.

---

[3] Это не всегда так: если индекс построен на всех необходимых для запроса столбцах, то вся нужная информация содержится непосредственно в нем, и обращение к таблице в этом случае происходить не будет.

[4] Про фактор кластеризации хорошо написано в книге Джонатана Льюиса «Oracle. Основы стоимостной оптимизации».

Решение 1.3 (вложенные циклы). Вместо использования функции часто встречается разделение запроса на два подзапроса во вложенных циклах PL/SQL:

```
declare
  name varchar2(110);
begin
  for i in (
    select b.book_id from books b where b.year_published = '2000'
  )
  loop

    name := null;
    for j in (
      select w.name
        from writers w
           , authors a
       where a.book_id = i.book_id
         and w.writer_id = a.writer_id
       order by a.seq_num
    )
    loop
      if name is null then
        name := j.name;
      else
        name := name || 'и др.';
        exit;
      end if;
    end loop;

    -- обработка

  end loop;
end;
```

Собрав трассу выполнения, увидим картину, вполне аналогичную первому решению:

```
SELECT B.BOOK_ID
FROM
 BOOKS B WHERE B.YEAR_PUBLISHED = 2000

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    20061      3.02       2.70      14278      31606          0       20060
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    20063      3.02       2.71      14278      31606          0       20060

********************************************************************************

SELECT W.NAME
FROM
 WRITERS W , AUTHORS A WHERE A.BOOK_ID = :B1 AND W.WRITER_ID =
  A.WRITER_ID ORDER BY A.SEQ_NUM

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute  20060      0.02       0.37          0          0          0           0
Fetch    40120      0.05       0.91          0     160516          0       30065
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    60181      0.07       1.28          0     160516          0       30065
```

Цифры получились несколько больше; в данном случае это связано с накладными расходами из-за построчной выборки значений циклом `for`.[5] Если считать записи с помощью `bulk collect`, получим результат, больше близкий к первому решению.

[5] См., например, http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:880343948514

Решение 1.4 (зафиксированный план запроса). Можно погубить производительность и неправильной подсказкой оптимизатору. Возьмем второе решение и потребуем, чтобы таблицы `writers` и `authors` соединялись методом вложенных циклов (nested loops):

```
SQL> set autotrace traceonly
SQL> select --+ leading(b) use_nl(w a)
       b.name
     , min(w.name) keep (dense_rank first order by a.seq_num) ||
       case when count(*) > 1 then ' и др.' else '' end
  from books b
     , writers w
     , authors a
 where b.year_published = '2000'
   and a.book_id = b.book_id
   and w.writer_id = a.writer_id
 group by b.name;

20060 строк выбрано.

Статистика
----------------------------------------------------------
          0  recursive calls
          0  db block gets
     124714  consistent gets
      17751  physical reads
          0  redo size
     614115  bytes sent via SQL*Net to client
       9635  bytes received via SQL*Net from client
       1339  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
      20060  rows processed
```

Здесь цифры получились несколько меньше, так как план выполнения не в точности соответствует плану первого решения. Оптимизатор справедливо счел, что выгоднее соединить таблицу book сначала с `authors`, а затем только с `writers`:

```
-------------------------------------------------------------------------------------------
| Id  | Operation                      |  Name        | Rows  | Bytes |TempSpc| Cost  |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |              | 20000 |   996K|       | 72400 |
|   1 |  SORT GROUP BY                 |              | 20000 |   996K|  2960K| 72400 |
|   2 |   NESTED LOOPS                 |              | 30000 |  1494K|       | 72170 |
|   3 |    NESTED LOOPS                |              | 30000 |   996K|       | 42170 |
|*  4 |     TABLE ACCESS FULL          | BOOKS        | 20000 |   429K|       |  2170 |
|   5 |     TABLE ACCESS BY INDEX ROWID| AUTHORS      |     2 |    24 |       |     2 |
|*  6 |      INDEX RANGE SCAN          | A_BOOK_ID    |     2 |       |       |     1 |
|   7 |    TABLE ACCESS BY INDEX ROWID | WRITERS      |     1 |    17 |       |     1 |
|*  8 |     INDEX UNIQUE SCAN          | W_WRITER_ID  |     1 |       |       |       |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - filter("B"."YEAR_PUBLISHED"='2000')
   6 - access("A"."BOOK_ID"="B"."BOOK_ID")
   8 - access("W"."WRITER_ID"="A"."WRITER_ID")
```

То есть из всех возможных планов в первом решении фактически используется самый неудачный!

Философское отступление. Как мы увидели, план, выглядящий логичным и понятным, далеко не всегда является оптимальным. Но ведь идея языка SQL состоит в том, что разработчику не надо даже задумываться о плане выполнения! Вместо процедурного, императивного стиля программирования, при котором надо явно определять алгоритм выборки данных, SQL предполагает декларативный стиль — разработчик должен описать, что он хочет получить, а оптимизатор запросов позаботится об оптимальном плане выполнения.

К сожалению, далеко не все можно реализовать на чистом SQL, поэтому в Oracle и существует его процедурное расширение PL/SQL. Но для выборки данных именно SQL подходит лучше всего.[6]

Разделяя большой запрос на два подзапроса, то есть привнося в выборку данных свое «процедурное видение», мы не даем оптимизатору ни малейшего шанса хорошо справиться со своей работой. А ведь все, что требуется для хорошей производительности в большинстве случаев — это просто ему не мешать.

Конечно, нет гарантии, что один большой запрос сразу заработает хорошо. Возможно, потребуется не только не мешать, но и оказать оптимизатору помощь.[7] Но это задача техническая и решаемая, а вот оптимизация программы, в которой запрос разбит на несколько, может быть невозможной в принципе.

В заключение рассмотрим совсем плохой случай.

Задача 2. Вывести названия всех книг, у которых несколько авторов.

Решение 2.1 (функция в where). Имея уже готовую функцию get_authors, есть соблазн решить задачу «по-простому»:

```
select b.name
  from books b
 where get_authors(b.book_id) like '%и др.';
```

Во что это нам обойдется? Если раньше функция участвовала в запросе в выражении select, то теперь она переместилась в where. Значит, она будет вызываться для каждой из 100 000 строк таблицы books, то есть мы можем ожидать еще примерно пятикратного ухудшения производительности по сравнению с первым решением! Так и есть:

```
SQL> set autotrace traceonly
SQL> select b.name
  from books b
 where get_authors(b.book_id) like '%и др.';

50000 строк выбрано.

Статистика
----------------------------------------------------------
     350066  recursive calls
          0  db block gets
     817593  consistent gets
      25139  physical reads
          0  redo size
     788029  bytes sent via SQL*Net to client
      23607  bytes received via SQL*Net from client
       3335  SQL*Net roundtrips to/from client
     100001  sorts (memory)
          0  sorts (disk)
      50000  rows processed
```

[6] Не зря Том Кайт не устает повторять и в своих книгах, и в блоге (http://tkyte.blogspot.com/), что если задача может быть решена на SQL, ее надо решать на SQL, а PL/SQL использовать только в том случае, если возможностей SQL не хватает, и только в минимальном объеме.

[7] Закон дырявых абстракций, ничего не поделаешь (http://local.joelonsoftware.com/wiki/Закон_Дырявых_Абстракций).

Решение 2.2 (чистый SQL). Запрос без функции требует всего лишь 14 756 логических чтений:

```
SQL> set autotrace traceonly
SQL> select b.name
  from books b
     , authors a
 where a.book_id = b.book_id
 group by b.name
having count(*) > 1;

50000 строк выбрано.

Статистика
----------------------------------------------------------
          0  recursive calls
          0  db block gets
      14756  consistent gets
      14659  physical reads
          0  redo size
     788029  bytes sent via SQL*Net to client
      23607  bytes received via SQL*Net from client
       3335  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
      50000  rows processed
```

Итак, вариант с функцией в 55 раз хуже по логическим чтениям, чем простой запрос на чистом SQL. Кошмарнее может быть только случай, когда функция употребляется в выражении where таким, например, образом:

```
get_authors(b.book_id) = nvl(:p, get_authors(b.book_id))
```

В этом случае функция будет вызвана два раза для каждой выбираемой строки, что ухудшит ситуацию еще в два раза.

Кэширование. Справедливости ради надо заметить, что в некоторых случаях, когда число комбинаций входных параметров функции мало, а вызывается функция часто, ситуацию можно исправить с помощью кэширования результатов в памяти. Первый раз функция вычислит значение и сохранит его, а при повторном обращении с теми же параметрами — возвратит уже вычисленное. Но во-первых, это возможно далеко не всегда (в нашем примере кэширование никак не поможет), а во-вторых, экономия логических чтений достигается за счет расхода оперативной памяти и времени процессора, и надо еще смотреть, что перевесит.

Заключение. Желание упростить задачу и разбить тем или иным способом сложный запрос на несколько более простых может привести к серьезному ухудшению производительности. На практике зачастую только за счет избавления от функций и вложенных циклов удается сократить время работы отчетов в разы, а в особо запущенных случаях и на порядок (с нескольких часов до нескольких минут).

Функция, выбирающая данные, иногда имеет право на существование в выражении select (если заведомо известно, что запрос возвращает немного строк), но использование ее в выражении where неизбежно приведет к катастрофическому падению производительности. Как правило, лучше всего изначально проектировать программу таким образом, чтобы выборка данных была сосредоточена в одном запросе (или в нескольких *последовательных* запросах).

Спасибо Николаю Лукину за помощь в поисках правды и за ценные замечания к статье, а также Владимиру Гончарову за пробуждение интереса к теме.

Приложение. Скрипт для создания объектов и заполнения таблиц тестовыми данными.

```
define N_B=100000
define N_P=70000

drop table authors;
drop table books;
drop table writers;

create table books(
    book_id number
  , name varchar2(100)
  , year_published char(4)
  , filler char(1000) default ' '
);
create unique index b_book_id on books(book_id);
alter table books add primary key(book_id);

create table writers(
    writer_id number
  , name varchar2(100)
  , filler char(1000) default ' '
);
create unique index w_writer_id on writers(writer_id);
alter table writers add primary key(writer_id);

create table authors(
    seq_num number
  , book_id number references books(book_id)
  , writer_id number references writers(writer_id)
);

exec dbms_random.initialize(0);

insert into books(book_id, name, year_published)
select rownum
     , 'Книга '||rownum
     , 2000 + round(mod(abs(dbms_random.random),5))
    from all_objects
   where rownum <= &&N_B;

insert into writers(writer_id, name)
select rownum
     , 'Автор '||rownum
    from all_objects
   where rownum <= &&N_P;

insert into authors(seq_num, book_id, writer_id)
select 1
     , book_id
     , 1 + mod(abs(dbms_random.random),&&N_P)
  from books b;

insert into authors(seq_num, book_id, writer_id)
select 2
     , book_id
     , 1 + mod(abs(dbms_random.random),&&N_P)
  from books b
 where mod(book_id,2) = 0;

commit;

create index a_book_id on authors(book_id);
create index a_writer_id on authors(writer_id);

exec dbms_stats.gather_table_stats(sys_context('userenv','current_schema'),'BOOKS');
exec dbms_stats.gather_table_stats(sys_context('userenv','current_schema'),'WRITERS');
exec dbms_stats.gather_table_stats(sys_context('userenv','current_schema'),'AUTHORS');

exec dbms_stats.gather_index_stats(sys_context('userenv','current_schema'),'B_BOOK_ID');
exec dbms_stats.gather_index_stats(sys_context('userenv','current_schema'),'W_WRITER_ID');
exec dbms_stats.gather_index_stats(sys_context('userenv','current_schema'),'A_BOOK_ID');
exec dbms_stats.gather_index_stats(sys_context('userenv','current_schema'),'A_WRITER_ID');

create or replace function get_authors(p_book_id number)
return varchar2
is
  name varchar2(110);
  name2 varchar2(100);
  cursor c(p_book_id number) is
    select w.name
      from writers w
         , authors a
     where a.book_id = p_book_id
       and w.writer_id = a.writer_id
     order by a.seq_num;
begin
  open c(p_book_id);
  fetch c into name;
  fetch c into name2;
  if c%found then
    name := name||' и др.';
  end if;
  close c;
  return name;
end;
/
```