异步图书
www.epubit.com

罗炳森 黄超 钟侥 著

SQL优化核心思想

本书内容源自“道森起点”高级SQL优化实战班

专业团队倾力打造，集多年SQL优化实战经验和专职教学经验于一体

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

罗炳森 黄超 钟侥 著

SQL 优化核心思想

人民邮电出版社
北京

图书在版编目（CIP）数据

SQL优化核心思想 / 罗炳森，黄超，钟侥著. -- 北京 : 人民邮电出版社，2018.4
ISBN 978-7-115-47849-8

Ⅰ. ①S… Ⅱ. ①罗… ②黄… ③钟… Ⅲ. ①SQL语言 Ⅳ. ①TP311.138

中国版本图书馆CIP数据核字(2018)第019196号

内 容 提 要

结构化查询语言（Structured Query Language，SQL）是一种功能强大的数据库语言。它基于关系代数运算，功能丰富、语言简洁、使用方便灵活，已成为关系数据库的标准语言。

本书旨在引导读者掌握 SQL 优化技能，以更好地提升数据库性能。本书共分 10 章，从 SQL 基础知识、统计信息、执行计划、访问路径、表连接方式、成本计算、查询变换、调优技巧、经典案例、全自动 SQL 审核等角度介绍了有关 SQL 优化的方方面面。

本书基于 Oracle 进行编写，内容讲解由浅入深，适合各个层次的读者学习。本书面向一线工程师、运维工程师、数据库管理员以及系统设计与开发人员，无论是初学者还是有一定基础的读者，都将从中获益。

◆ 著　　罗炳森　黄　超　钟　侥
责任编辑　胡俊英
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市君旺印务有限公司印刷

◆ 开本：800×1000　1/16
印张：20
字数：445 千字　　2018 年 4 月第 1 版
印数：1－2 400 册　　2018 年 4 月河北第 1 次印刷

定价：79.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

前　言

近年来，随着系统的数据量逐年增加，并发量也成倍增长，SQL 性能越来越成为 IT 系统设计和开发时首要考虑的问题之一。SQL 性能问题已经逐步发展成为数据库性能的首要问题，80%的数据库性能问题都是因 SQL 而导致。面对日益增多的 SQL 性能问题，如何下手以及如何提前审核已经成为越来越多的 IT 从业者必须要考虑的问题。

现在将 8 年专职 SQL 优化的经验和心得与大家一起分享，以揭开 SQL 优化的神秘面纱，让一线工程师在实际开发中不再寝食难安、谈虎色变，最终能够对 SQL 优化技能驾轻就熟。

编写本书也是对多年学习积累的一个总结，鞭策自己再接再厉。如果能够给各位读者在 SQL 优化上提供一点帮助，也不枉个中辛苦。

2014 年，作者罗炳森与有教无类（网名）联合编写了《Oracle 查询优化改写技巧与案例》一书，该书主要侧重于 SQL 优化改写技巧。到目前为止，该书仍然是市面上唯一一本专门讲解 SQL 改写技巧的图书。

因为《Oracle 查询优化改写技巧与案例》只专注于 SQL 改写技巧，并没有涉及 SQL 优化的具体思想、方法和步骤，本书可以看作是对《Oracle 查询优化改写技巧与案例》一书的进一步补充。

本书共 10 章，各章的主要内容如下。

第 1 章详细介绍了 SQL 优化的基础知识以及初学者切实需要掌握的基本内容，本章可以帮助初学者快速入门。

第 2 章详细讲解统计信息定义、统计信息的重要性、统计信息相关参数设置方案以及统计信息收集策略。

第 3 章详细讲解执行计划、各种执行计划的使用场景以及执行计划的阅读方法，通过定制执行计划，读者可以快速找出 SQL 性能瓶颈。

第 4 章详细讲解常见的访问路径，这是阅读执行计划中比较重要的环节，需要掌握各种常见的访问路径。

第 5 章详细讲解表的各种连接方式、各种表连接方式的等价改写以及相互转换，这也是本书的核心章节。

第 6 章介绍单表访问以及索引扫描的成本计算方法，并由此引出 SQL 优化的核心思想。

第 7 章讲解常见的查询变换，分别是子查询非嵌套、视图合并和谓词推入。如果要对复杂的 SQL（包含各种子查询的 SQL）进行优化，读者就必须掌握查询变换技巧。

第 8 章讲解各种优化技巧，其中涵盖分页语句优化思想、分析函数减少表扫描次数、超大表与超大表关联优化方法、dblink 优化思路，以及大表的 rowid 切片优化技巧。掌握这些调优技巧往往能够事半功倍。

第 9 章分享在 SQL 优化实战中遇到的经典案例，读者可以在欣赏 SQL 优化案例的同时学习罗老师多年专职 SQL 优化的经验，同时学到很多具有实战意义的优化思想以及优化方法与技巧。

第 10 章讲解全自动 SQL 审核，将有性能问题的 SQL 扼杀在“摇篮”里，确保系统上线之后，不会因为 SQL 写法导致性能问题，同时还能抓出不符合 SQL 编码规范但是已经上线的 SQL。

本书对系统面临性能压力挑战的一线工程师、运维工程师、数据库管理员（DBA）、系统设计与开发人员，具有极大的参考价值。

为了满足不同层次的读者需求，本书在写作的内容上尽量由浅入深，前 5 章比较浅显易懂，适合 SQL 优化初学者阅读。通读完前 5 章之后，初学者能够对 SQL 优化有一定认识。后 5 章属于进阶和高级内容，适合有一定基础的人阅读。通读完后 5 章的内容之后，无论是初学者或是有一定基础的读者都能从中获益良多。

本书专注于 SQL 优化技巧，因此书中不会涉及太多数据库系统优化的内容。

虽然本书是基于 Oracle 编写的，但是关系型数据库的优化方法都殊途同归，因此无论是 DB2 从业者、SQL SERVER 从业者、MYSQL 从业者，亦或是 PostGre SQL 从业者等，都能从本书中学到所需要的 SQL 优化知识。

因水平有限，本书在编写过程中难免有错漏之处，恳请读者批评、指正。联系我们的方式如下：692162374@qq.com（QQ 好友数已达上限）或者 327165427@qq.com（新开 QQ 账号）。

如果有读者想进一步学习 SQL 优化技能或者一些公司或机构需要开展 SQL 优化方面的培训，都可以联系作者。另外，作者还开设了实体培训班，可以实现零基础学习，结业后可以顺利就业，欢迎联系罗老师。

本书约定

在阅读本书之前请读者安装好 Oracle 数据库并且配置好示例账户 Scott，因为本书均以 Scott 账户进行讲解。推荐读者安装与本书相同版本的数据库进行测试，具有专研精神的读者请安装好 Oracle12c 进行对比实验，这样一来，你将发现 Oracle12c CBO 的一些新特征。本书使用的版本是 Oracle11gR2。

```
SQL> select * from v$version where rownum=1;

BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production

SQL> show user
USER is "SYS"

SQL> grant dba to scott;

Grant succeeded.

SQL> alter user scott account unlock;

User altered.

SQL> alter user scott identified by tiger;

User altered.

SQL> conn scott/tiger
Connected.

SQL> create table test as select * from dba_objects;

Table created.
```

目　录

第1章 SQL优化必懂概念

1.1 基数（CARDINALITY）

某个列唯一键（Distinct_Keys）的数量叫作基数。比如性别列，该列只有男女之分，所以这一列基数是2。主键列的基数等于表的总行数。基数的高低影响列的数据分布。

以测试表test为例，owner列和object_id列的基数分别如下所示。

```
SQL> select count(distinct owner),count(distinct object_id),count(*) from test;

COUNT(DISTINCTOWNER) COUNT(DISTINCTOBJECT_ID)   COUNT(*)
-------------------- ------------------------ ----------
                  29                    72462      72462
```

TEST表的总行数为72 462，owner列的基数为29，说明owner列里面有大量重复值，object_id列的基数等于总行数，说明object_id列没有重复值，相当于主键。owner列的数据分布如下。

```
SQL> select owner,count(*) from test group by owner order by 2 desc;

OWNER                  COUNT(*)
---------------------- ----------
SYS                         30808
PUBLIC                      27699
SYSMAN                       3491
ORDSYS                       2532
APEX_030200                  2406
MDSYS                        1509
XDB                           844
OLAPSYS                       719
SYSTEM                        529
CTXSYS                        366
WMSYS                         316
EXFSYS                        310
SH                            306
ORDDATA                       248
OE                            127
DBSNMP                         57
IX                             55
HR                             34
PM                             27
FLOWS_FILES                    12
OWBSYS_AUDIT                   12
ORDPLUGINS                     10
OUTLN                           9
BI                              8
SI_INFORMTN_SCHEMA              8
ORACLE_OCM                      8
SCOTT                           7
```

```
APPQOSSYS                      3
OWBSYS                         2
```

owner 列的数据分布极不均衡，我们运行如下 SQL。

```
select * from test where owner='SYS';
```

SYS 有 30 808 条数据，从 72 462 条数据里面查询 30 808 条数据，也就是说要返回表中 42.5%的数据。

```
SQL> select 30808/72462*100 "Percent" from dual;

   Percent
----------
42.5160774
```

那么请思考，你认为以上查询应该使用索引吗？现在我们换一种查询语句。

```
select * from test where owner='SCOTT';
```

SCOTT 有 7 条数据，从 72 462 条数据里面查询 7 条数据，也就是说要返回表中 0.009%的数据。

```
SQL> select 7/72462*100 "Percent" from dual;

   Percent
----------
.009660236
```

请思考，返回表中 0.009%的数据应不应该走索引？

如果你还不懂索引，没关系，后面的章节我们会详细介绍。如果你回答不了上面的问题，我们先提醒一下。当查询结果是返回表中 5%以内的数据时，应该走索引；当查询结果返回的是超过表中 5%的数据时，应该走全表扫描。

当然了，返回表中 5%以内的数据走索引，返回超过 5%的数据就使用全表扫描，这个结论太绝对了，因为你还没掌握后面章节的知识，这里暂且记住 5%这个界限就行。我们之所以在这里讲 5%，是怕一些初学者不知道上面问题的答案而纠结。

现在有如下查询语句。

```
select * from test where owner=:B1;
```

语句中，“:B1”是绑定变量，可以传入任意值，该查询可能走索引也可能走全表扫描。

现在得到一个结论：如果某个列基数很低，该列数据分布就会非常不均衡，由于该列数据分布不均衡，会导致 SQL 查询可能走索引，也可能走全表扫描。在做 SQL 优化的时候，如果怀疑列数据分布不均衡，我们可以使用 select 列，count(*) from 表 group by 列 order by 2 desc 来查看列的数据分布。

如果 SQL 语句是单表访问，那么可能走索引，可能走全表扫描，也可能走物化视图扫描。在不考虑有物化视图的情况下，单表访问要么走索引，要么走全表扫描。现在，回忆一下走索引的条件：返回表中 5%以内的数据走索引，超过 5%的时候走全表扫描。相信大家读到这里，已经搞懂了单表访问的优化方法。

我们来看如下查询。

```
select * from test where object_id=:B1;
```

不管 object_id 传入任何值，都应该走索引。

我们再思考如下查询语句。

```
select * from test where object_name=:B1;
```

不管给 object_name 传入任何值，请问该查询应该走索引吗？

请你去查看 object_name 的数据分布。写到这里，其实有点想把本节名称改为“数据分布”。大家在以后的工作中一定要注意列的数据分布！

1.2 选择性（SELECTIVITY）

基数与总行数的比值再乘以 100%就是某个列的选择性。

在进行 SQL 优化的时候，单独看列的基数是没有意义的，基数必须对比总行数才有实际意义，正是因为这个原因，我们才引出了选择性这个概念。

下面我们查看 test 表各个列的基数与选择性，为了查看选择性，必须先收集统计信息。关于统计信息，我们在第 2 章会详细介绍。下面的脚本用于收集 test 表的统计信息。

```
SQL> BEGIN
  2    DBMS_STATS.GATHER_TABLE_STATS(ownname          => 'SCOTT',
  3                                  tabname          => 'TEST',
  4                                  estimate_percent => 100,
  5                                  method_opt => 'for all columns size 1',
  6                                  no_invalidate    => FALSE,
  7                                  degree           => 1,
  8                                  cascade          => TRUE);
  9  END;
 10  /

PL/SQL procedure successfully completed.
```

下面的脚本用于查看 test 表中每个列的基数与选择性。

```
SQL> select a.column_name,
  2         b.num_rows,
  3         a.num_distinct Cardinality,
  4         round(a.num_distinct / b.num_rows * 100, 2) selectivity,
  5         a.histogram,
  6         a.num_buckets
  7    from dba_tab_col_statistics a, dba_tables b
  8   where a.owner = b.owner
  9     and a.table_name = b.table_name
 10     and a.owner = 'SCOTT'
 11     and a.table_name = 'TEST';

COLUMN_NAME       NUM_ROWS CARDINALITY SELECTIVITY HISTOGRAM NUM_BUCKETS
--------------- ---------- ----------- ----------- --------- -----------
OWNER                72462          29         .04 NONE                1
OBJECT_NAME          72462       44236       61.05 NONE                1
SUBOBJECT_NAME       72462         106         .15 NONE                1
OBJECT_ID            72462       72462         100 NONE                1
```

```
DATA_OBJECT_ID          72462          7608         10.5 NONE                  1
OBJECT_TYPE             72462            44          .06 NONE                  1
CREATED                 72462          1366         1.89 NONE                  1
LAST_DDL_TIME           72462          1412         1.95 NONE                  1
TIMESTAMP               72462          1480         2.04 NONE                  1
STATUS                  72462             1            0 NONE                  1
TEMPORARY               72462             2            0 NONE                  1
GENERATED               72462             2            0 NONE                  1
SECONDARY               72462             2            0 NONE                  1
NAMESPACE               72462            21          .03 NONE                  1
EDITION_NAME            72462             0            0 NONE                  0

15 rows selected.
```

请思考：什么样的列必须建立索引呢？

有人说基数高的列，有人说在 where 条件中的列。这些答案并不完美。基数高究竟是多高？没有和总行数对比，始终不知道有多高。比如某个列的基数有几万行，但是总行数有几十亿行，那么这个列的基数还高吗？这就是要引出选择性的根本原因。

当一个列选择性大于 20%，说明该列的数据分布就比较均衡了。测试表 test 中 object_name、object_id 的选择性均大于 20%，其中 object_name 列的选择性为 61.05%。现在我们查看该列数据分布（为了方便展示，只输出前 10 行数据的分布情况）。

```
SQL> select *
  2    from (select object_name, count(*)
  3            from test
  4           group by object_name
  5           order by 2 desc)
  6   where rownum <= 10;

OBJECT_NAME          COUNT(*)
-------------------- ----------
COSTS                        30
SALES                        30
SALES_CHANNEL_BIX            29
COSTS_TIME_BIX               29
COSTS_PROD_BIX               29
SALES_TIME_BIX               29
SALES_PROMO_BIX              29
SALES_PROD_BIX               29
SALES_CUST_BIX               29
DBMS_REPCAT_AUTH              5

10 rows selected.
```

由上面的查询结果我们可知，object_name 列的数据分布非常均衡。我们查询以下 SQL。

```
select * from test where object_name=:B1;
```

不管 object_name 传入任何值，最多返回 30 行数据。

什么样的列必须要创建索引呢？当一个列出现在 where 条件中，该列没有创建索引并且选择性大于 20%，那么该列就必须创建索引，从而提升 SQL 查询性能。当然了，如果表只有几百条数据，那我们就不用创建索引了。

下面抛出 SQL 优化核心思想第一个观点：只有大表才会产生性能问题。

也许有人会说：“我有个表很小，只有几百条，但是该表经常进行 DML，会产生热点块，

也会出性能问题。”对此我们并不想过多地讨论此问题，这属于应用程序设计问题，不属于 SQL 优化的范畴。

下面我们将通过实验为大家分享本书第一个全自动优化脚本。

抓出必须创建索引的列（请读者对该脚本适当修改，以便用于生产环境）。

首先，该列必须出现在 where 条件中，怎么抓出表的哪个列出现在 where 条件中呢？有两种方法，一种是可以通过 V$SQL_PLAN 抓取，另一种是通过下面的脚本抓取。

先执行下面的存储过程，刷新数据库监控信息。

```
begin
  dbms_stats.flush_database_monitoring_info;
end;
```

运行完上面的命令之后，再运行下面的查询语句就可以查询出哪个表的哪个列出现在 where 条件中。

```
select r.name owner,
       o.name table_name,
       c.name column_name,
       equality_preds, ---等值过滤
       equijoin_preds, ---等值 JOIN 比如 where a.id=b.id
       nonequijoin_preds, ----不等 JOIN
       range_preds, ----范围过滤次数 > >= < <= between and
       like_preds, ----LIKE 过滤
       null_preds, ----NULL 过滤
       timestamp
  from sys.col_usage$ u, sys.obj$ o, sys.col$ c, sys.user$ r
 where o.obj# = u.obj#
   and c.obj# = u.obj#
   and c.col# = u.intcol#
   and r.name = 'SCOTT'
   and o.name = 'TEST';
```

下面是实验步骤。

我们首先运行一个查询语句，让 owner 与 object_id 列出现在 where 条件中。

```
SQL> select object_id, owner, object_type
  2    from test
  3   where owner = 'SYS'
  4     and object_id < 100
  5     and rownum <= 10;

 OBJECT_ID OWNER                OBJECT_TYPE
---------- -------------------- -----------
        20 SYS                  TABLE
        46 SYS                  INDEX
        28 SYS                  TABLE
        15 SYS                  TABLE
        29 SYS                  CLUSTER
         3 SYS                  INDEX
        25 SYS                  TABLE
        41 SYS                  INDEX
        54 SYS                  INDEX
        40 SYS                  INDEX

10 rows selected.
```

其次刷新数据库监控信息。

```
SQL> begin
  2    dbms_stats.flush_database_monitoring_info;
  3  end;
  4  /

PL/SQL procedure successfully completed.
```

然后我们查看 test 表有哪些列出现在 where 条件中。

```
SQL> select r.name owner, o.name table_name, c.name column_name
  2    from sys.col_usage$ u, sys.obj$ o, sys.col$ c, sys.user$ r
  3   where o.obj# = u.obj#
  4     and c.obj# = u.obj#
  5     and c.col# = u.intcol#
  6     and r.name = 'SCOTT'
  7     and o.name = 'TEST';

OWNER      TABLE_NAME COLUMN_NAME
---------- ---------- ------------------------------
SCOTT      TEST       OWNER
SCOTT      TEST       OBJECT_ID
```

接下来我们查询出选择性大于等于 20%的列。

```
SQL> select a.owner,
  2         a.table_name,
  3         a.column_name,
  4         round(a.num_distinct / b.num_rows * 100, 2) selectivity
  5    from dba_tab_col_statistics a, dba_tables b
  6   where a.owner = b.owner
  7     and a.table_name = b.table_name
  8     and a.owner = 'SCOTT'
  9     and a.table_name = 'TEST'
 10     and a.num_distinct / b.num_rows >= 0.2;

OWNER      TABLE_NAME COLUMN_NAME   SELECTIVITY
---------- ---------- ------------- -----------
SCOTT      TEST       OBJECT_NAME         61.05
SCOTT      TEST       OBJECT_ID             100
```

最后，确保这些列没有创建索引。

```
SQL> select table_owner, table_name, column_name, index_name
  2    from dba_ind_columns
  3   where table_owner = 'SCOTT'
  4     and table_name = 'TEST';
未选定行
```

把上面的脚本组合起来，我们就可以得到全自动的优化脚本了。

```
SQL> select owner,
  2         column_name,
  3         num_rows,
  4         Cardinality,
  5         selectivity,
  6         'Need index' as notice
  7    from (select b.owner,
  8                 a.column_name,
  9                 b.num_rows,
```

```
 10                 a.num_distinct Cardinality,
 11                 round(a.num_distinct / b.num_rows * 100, 2) selectivity
 12            from dba_tab_col_statistics a, dba_tables b
 13           where a.owner = b.owner
 14             and a.table_name = b.table_name
 15             and a.owner = 'SCOTT'
 16             and a.table_name = 'TEST')
 17   where selectivity >= 20
 18     and column_name not in (select column_name
 19                               from dba_ind_columns
 20                              where table_owner = 'SCOTT'
 21                                and table_name = 'TEST')
 22     and column_name in
 23         (select c.name
 24            from sys.col_usage$ u, sys.obj$ o, sys.col$ c, sys.user$ r
 25           where o.obj# = u.obj#
 26             and c.obj# = u.obj#
 27             and c.col# = u.intcol#
 28             and r.name = 'SCOTT'
 29             and o.name = 'TEST');

OWNER      COLUMN_NAME     NUM_ROWS CARDINALITY SELECTIVITY NOTICE
---------- ------------ ---------- ----------- ----------- ----------
SCOTT      OBJECT_ID         72462       72462         100 Need index
```

1.3 直方图（HISTOGRAM）

前面提到，当某个列基数很低，该列数据分布就会不均衡。数据分布不均衡会导致在查询该列的时候，要么走全表扫描，要么走索引扫描，这个时候很容易走错执行计划。

如果没有对基数低的列收集直方图统计信息，基于成本的优化器（CBO）会认为该列数据分布是均衡的。

下面我们还是以测试表 test 为例，用实验讲解直方图。

首先我们对测试表 test 收集统计信息，在收集统计信息的时候，不收集列的直方图，语句 `for all columns size 1` 表示对所有列都不收集直方图。

```
SQL> BEGIN
  2    DBMS_STATS.GATHER_TABLE_STATS(ownname          => 'SCOTT',
  3                                  tabname          => 'TEST',
  4                                  estimate_percent => 100,
  5                                  method_opt       => 'for all columns size 1',
  6                                  no_invalidate    => FALSE,
  7                                  degree           => 1,
  8                                  cascade          => TRUE);
  9  END;
 10  /

PL/SQL procedure successfully completed.
```

Histogram 为 none 表示没有收集直方图。

```
SQL> select a.column_name,
  2         b.num_rows,
  3         a.num_distinct Cardinality,
  4         round(a.num_distinct / b.num_rows * 100, 2) selectivity,
  5         a.histogram,
```

```
  6         a.num_buckets
  7    from dba_tab_col_statistics a, dba_tables b
  8   where a.owner = b.owner
  9     and a.table_name = b.table_name
 10     and a.owner = 'SCOTT'
 11     and a.table_name = 'TEST';

COLUMN_NAME       NUM_ROWS CARDINALITY SELECTIVITY HISTOGRAM NUM_BUCKETS
--------------- ---------- ----------- ----------- --------- -----------
OWNER                72462          29         .04 NONE                1
OBJECT_NAME          72462       44236       61.05 NONE                1
SUBOBJECT_NAME       72462         106         .15 NONE                1
OBJECT_ID            72462       72462         100 NONE                1
DATA_OBJECT_ID       72462        7608        10.5 NONE                1
OBJECT_TYPE          72462          44         .06 NONE                1
CREATED              72462        1366        1.89 NONE                1
LAST_DDL_TIME        72462        1412        1.95 NONE                1
TIMESTAMP            72462        1480        2.04 NONE                1
STATUS               72462           1           0 NONE                1
TEMPORARY            72462           2           0 NONE                1
GENERATED            72462           2           0 NONE                1
SECONDARY            72462           2           0 NONE                1
NAMESPACE            72462          21         .03 NONE                1
EDITION_NAME         72462           0           0 NONE                0

15 rows selected.
```

owner 列基数很低，现在我们对 owner 列进行查询。

```
SQL> set autot trace
SQL> select * from test where owner='SCOTT';

7 rows selected.

Execution Plan
----------------------------------------------------------
Plan hash value: 1357081020

--------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|  0  | SELECT STATEMENT  |      |  2499 |   236K|   289   (1)| 00:00:04 |
|* 1  |  TABLE ACCESS FULL| TEST |  2499 |   236K|   289   (1)| 00:00:04 |
--------------------------------------------------------------------------------

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

   1 - filter("OWNER"='SCOTT')
```

请注意看粗体字部分，查询 owner='SCOTT'返回了 7 条数据，但是 CBO 在计算 **Rows** 的时候认为 owner='SCOTT'返回 **2 499** 条数据，Rows 估算得不是特别准确。从 72 462 条数据里面查询出 7 条数据，应该走索引，所以现在我们对 owner 列创建索引。

```
SQL> create index idx_owner on test(owner);

Index created.
```

我们再来查询一下。