收获，不止 SQL 优化

第 七 章

且慢，探寻表设计让 SQL 飞

E-Mail:45240040@qq.com

目录

1.分区表的好处

◆ 说说分区表的主要好处是什么，为什么会有这些好处。

1.1 较少访问路径

1.1.1 环境准备

```
SQL> create table norm_tab (id number,deal_date date,area_code number,nbr number,contents varchar2(4000));

Table created.

insert into norm_tab(id,deal_date,area_code,nbr,contents)
      select rownum,
             to_date( to_char(sysdate-365,'J')+TRUNC(DBMS_RANDOM.VALUE(0,365)),'J'),
             ceil(dbms_random.value(590,599)),
             ceil(dbms_random.value(18900000001,18999999999)),
             rpad('*',400,'*')
        from dual
  8        connect by rownum <= 100000;

100000 rows created.

SQL> commit;

Commit complete.
```

构造了一个非分区表并插入了10万条数据

```
create table range_part_tab (id number,deal_date date,area_code number,nbr number,contents varchar2(4000))
          partition by range (deal_date)
          (
          partition p_201301 values less than (TO_DATE('2013-02-01', 'YYYY-MM-DD')),
          partition p_201302 values less than (TO_DATE('2013-03-01', 'YYYY-MM-DD')),
          partition p_201303 values less than (TO_DATE('2013-04-01', 'YYYY-MM-DD')),
          partition p_201304 values less than (TO_DATE('2013-05-01', 'YYYY-MM-DD')),
          partition p_201305 values less than (TO_DATE('2013-06-01', 'YYYY-MM-DD')),
          partition p_201306 values less than (TO_DATE('2013-07-01', 'YYYY-MM-DD')),
          partition p_201307 values less than (TO_DATE('2013-08-01', 'YYYY-MM-DD')),
          partition p_201308 values less than (TO_DATE('2013-09-01', 'YYYY-MM-DD')),
          partition p_201309 values less than (TO_DATE('2013-10-01', 'YYYY-MM-DD')),
          partition p_201310 values less than (TO_DATE('2013-11-01', 'YYYY-MM-DD')),
          partition p_201311 values less than (TO_DATE('2013-12-01', 'YYYY-MM-DD')),
          partition p_201312 values less than (TO_DATE('2014-01-01', 'YYYY-MM-DD')),
          partition p_201401 values less than (TO_DATE('2014-02-01', 'YYYY-MM-DD')),
          partition p_201402 values less than (TO_DATE('2014-03-01', 'YYYY-MM-DD')),
          partition p_max values less than (maxvalue)
 19       );

Table created.

insert into range_part_tab (id,deal_date,area_code,nbr,contents)
      select rownum,
             to_date( to_char(sysdate-365,'J')+TRUNC(DBMS_RANDOM.VALUE(0,365)),'J'),
             ceil(dbms_random.value(591,599)),
             ceil(dbms_random.value(18900000001,18999999999)),
             rpad('*',400,'*')
        from dual
  8        connect by rownum <= 100000;

100000 rows created.

SQL> commit;

Commit complete.
```

构造了一个范围分区表并插入了与上一个非分区表同样数量的数据，用来比较两者的差异

1.1.2 分区表与非分区表存储差异

比较分区与非分区在空间分配上的差异，非分区表分配了一个大段，分区表每个分区都分配一个段

```
SET LINESIZE 666
set pagesize 5000
column segment_name format a20
column partition_name format a20
column segment_type format a20
select segment_name,
       partition_name,
       segment_type,
       bytes / 1024 / 1024 "字节数(M)",
       tablespace_name
  from user_segments
  7   where segment_name IN('RANGE_PART_TAB','NORM_TAB');

SEGMENT_NAME         PARTITION_NAME       SEGMENT_TYPE          ??????(M) TABLESPACE_NAME
-------------------- -------------------- ------------------ ---------- --------------------
NORM_TAB                                  TABLE                      49 TBS_1
RANGE_PART_TAB       P_201301             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201302             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201303             TABLE PARTITION             2 TBS_1
RANGE_PART_TAB       P_201304             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201305             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201306             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201307             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201308             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201309             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201310             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201311             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201312             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201401             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201402             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_MAX                TABLE PARTITION             3 TBS_1

16 rows selected.
```

1.1.3 分区表的执行效率

相同语句下的分区表与分区表执行效率

```
set linesize 1000
set autotrace traceonly
set timing on
select *
      from range_part_tab
     where deal_date >= TO_DATE('2013-08-04', 'YYYY-MM-DD')
SQL> SQL>   2    3    4         and deal_date <= TO_DATE('2013-08-07', 'YYYY-MM-DD');

1090 rows selected.

Elapsed: 00:00:00.08

Execution Plan
----------------------------------------------------------
Plan hash value: 16125146

-------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name          | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |               |   706 |  1413K|   147   (0)| 00:00:02 |       |       |
|   1 |  PARTITION RANGE SINGLE|              |   706 |  1413K|   147   (0)| 00:00:02 |     8 |     8 |
|*  2 |   TABLE ACCESS FULL   | RANGE_PART_TAB |   706 |  1413K|   147   (0)| 00:00:02 |     8 |     8 |
-------------------------------------------------------------------------------------------------------------

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

   2 - filter("DEAL_DATE">=TO_DATE('2013-08-04 00:00:00', 'yyyy-mm-dd hh24:mi:ss') AND
              "DEAL_DATE"<=TO_DATE('2013-08-07 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
```

```
Note
-----
   - dynamic sampling used for this statement


Statistics
----------------------------------------------------------
        148  recursive calls
          0  db block gets
        994  consistent gets
          0  physical reads
      22940  redo size
      40051  bytes sent via SQL*Net to client
       1176  bytes received via SQL*Net from client
         74  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
       1090  rows processed
```

1.1.4 非分区表的执行效率

```
SQL>
select *
      from norm_tab
     where deal_date >= TO_DATE('2013-08-04', 'YYYY-MM-DD')
  4          and deal_date <= TO_DATE('2013-08-07', 'YYYY-MM-DD');

1129 rows selected.                    非分区下查询效率

Elapsed: 00:00:06.24

Execution Plan
----------------------------------------------------------
Plan hash value: 278673677

--------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |   893 |  1787K|  1701   (1)| 00:00:21 |
|*  1 |  TABLE ACCESS FULL| NORM_TAB |   893 |  1787K|  1701   (1)| 00:00:21 |
--------------------------------------------------------------------------------

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

   1 - filter("DEAL_DATE">=TO_DATE('2013-08-04 00:00:00', 'yyyy-mm-dd
              hh24:mi:ss') AND "DEAL_DATE"<=TO_DATE('2013-08-07 00:00:00',
              'yyyy-mm-dd hh24:mi:ss'))
```

```
Note
-----
   - dynamic sampling used for this statement


Statistics
----------------------------------------------------------
         28  recursive calls
          0  db block gets
      10134  consistent gets
          0  physical reads
     268692  redo size
      41523  bytes sent via SQL*Net to client
       1209  bytes received via SQL*Net from client
         77  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
       1129  rows processed
```

1.1.5 分析原因

对比上面的执行计划，从执行消耗的时间、执行产生的一致性读、系统计算的 COST 代价等可以看出，就本条执行语句分区表明显是优于分分区表的，这是因为本次查询结果正好在第 8 个分区中，系统只扫描第 8 个分区就可以查询出所有的结果，全不必对扫描所有分区，而非分区表下的查询是需要扫描整个段。

1.2 truncate 操作方便

```
select segment_name,
       partition_name,
       segment_type,                                    试验1.1创建的表
       bytes / 1024 / 1024 "M",
       tablespace_name
  from user_segments
  7   where segment_name IN('RANGE_PART_TAB');

SEGMENT_NAME         PARTITION_NAME       SEGMENT_TYPE                 M TABLESPACE_NAME
-------------------- -------------------- ------------------ ---------- --------------------
RANGE_PART_TAB       P_201301             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201302             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201303             TABLE PARTITION             2 TBS_1
RANGE_PART_TAB       P_201304             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201305             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201306             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201307             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201308             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201309             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201310             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201311             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201312             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201401             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201402             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_MAX                TABLE PARTITION             3 TBS_1

15 rows selected.
```

```
SQL> select count(*) from RANGE_PART_TAB partition(p_201303);

  COUNT(*)
----------
      3259

SQL> alter table range_part_tab truncate partition p_201303;

Table truncated.

SQL> select count(*) from RANGE_PART_TAB partition(p_201303);

  COUNT(*)
----------
         0
```

我们知道删除数据的时候 truncate 一定会比 delete 删除更快，truncate 缺点是不能

使用 where 谓词,而分区表就巧妙第避开了这个缺点，达到了 truncate 部分数据的目的。（一般分区表每个分区存放的也是分类数据）

1.3 drop 操作方便

```
select segment_name,
       partition_name,
       segment_type,                                  试验1.1创建的表
       bytes / 1024 / 1024 "M",
       tablespace_name
  from user_segments
  7   where segment_name IN('RANGE_PART_TAB');

SEGMENT_NAME         PARTITION_NAME       SEGMENT_TYPE                M TABLESPACE_NAME
-------------------- -------------------- ------------------ ---------- --------------------
RANGE_PART_TAB       P_201301             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201302             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201303             TABLE PARTITION             2 TBS_1
RANGE_PART_TAB       P_201304             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201305             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201306             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201307             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201308             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201309             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201310             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201311             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201312             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201401             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201402             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_MAX                TABLE PARTITION             3 TBS_1

15 rows selected.
```

```
SQL> alter table range_part_tab drop partition p_201303;

Table altered.
                                          已查询不到p_201303分区
select segment_name,
       partition_name,
       segment_type,
       bytes / 1024 / 1024 "M",
       tablespace_name
  from user_segments
  7   where segment_name IN('RANGE_PART_TAB');

SEGMENT_NAME         PARTITION_NAME       SEGMENT_TYPE                M TABLESPACE_NAME
-------------------- -------------------- ------------------ ---------- --------------------
RANGE_PART_TAB       P_201301             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201302             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201304             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201305             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201306             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201307             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201308             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201309             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201310             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201311             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201312             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201401             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201402             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_MAX                TABLE PARTITION             3 TBS_1

14 rows selected.
```

通过上面的操作可以发现分区表就算有很多段，那么删除一个段只需要一个

DDL 语句就可以完成，而且不影响其他段的数据。

1.4 add 操作方便

```
select segment_name,
       partition_name,
       segment_type,                          之前试验的表
       bytes / 1024 / 1024 "M",
       tablespace_name
  from user_segments
  7   where segment_name IN('RANGE_PART_TAB');

SEGMENT_NAME         PARTITION_NAME       SEGMENT_TYPE                 M TABLESPACE_NAME
-------------------- -------------------- ------------------ ---------- ------------------
RANGE_PART_TAB       P_201301             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201302             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201304             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201305             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201306             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201307             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201308             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201309             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201310             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201311             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201312             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201401             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201402             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_MAX                TABLE PARTITION             3 TBS_1

14 rows selected.
```

```
SQL> alter table range_part_tab  drop partition p_max;

Table altered.

SQL> alter table range_part_tab  add PARTITION p_201404 values less than (TO_DATE('2014-04-01', 'YYYY-MM-DD'));

Table altered.

SQL> alter table range_part_tab  add PARTITION p_max  values less than (maxvalue);

Table altered.
```

```
select segment_name,
       partition_name,
       segment_type,
       bytes / 1024 / 1024 "M",
       tablespace_name
  from user_segments
  7   where segment_name IN('RANGE_PART_TAB');
                                              增加的分区
SEGMENT_NAME         PARTITION_NAME       SEGMENT_TYPE                 M TABLESPACE_NAME
-------------------- -------------------- ------------------ ---------- ------------------
RANGE_PART_TAB       P_201301             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201302             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_201304             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201305             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201306             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201307             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201308             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201309             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201310             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201311             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201312             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201401             TABLE PARTITION             5 TBS_1
RANGE_PART_TAB       P_201402             TABLE PARTITION             4 TBS_1
RANGE_PART_TAB       P_201404             TABLE PARTITION          .0625 TBS_1
RANGE_PART_TAB       P_MAX                TABLE PARTITION          .0625 TBS_1
```

对比 drop 一个分区，add 一个分区也是相当方便简单，方便数据的扩展。

1.5 split 操作方便

```
select segment_name,
       partition_name,
       segment_type,                          目前表的分区情况
       bytes / 1024 / 1024 "M",
       tablespace_name
  from user_segments
  7   where segment_name IN('RANGE_PART_TAB');

SEGMENT_NAME         PARTITION_NAME       SEGMENT_TYPE                  M TABLESPACE_NAME
-------------------- -------------------- ------------------ ---------- ---------------
RANGE_PART_TAB       P_201301             TABLE PARTITION           .0625 TBS_1
RANGE_PART_TAB       P_201302             TABLE PARTITION           .0625 TBS_1
RANGE_PART_TAB       P_201304             TABLE PARTITION              4 TBS_1
RANGE_PART_TAB       P_201305             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201306             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201307             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201308             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201309             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201310             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201311             TABLE PARTITION              4 TBS_1
RANGE_PART_TAB       P_201312             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201401             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201402             TABLE PARTITION              4 TBS_1
RANGE_PART_TAB       P_201404             TABLE PARTITION           .0625 TBS_1
RANGE_PART_TAB       P_MAX                TABLE PARTITION           .0625 TBS_1

15 rows selected.
```

```
SQL> alter table range_part_tab SPLIT PARTITION P_MAX  at (TO_DATE('2014-05-01', 'YYYY-MM-DD')) into (PARTITION p_201405  ,PARTITION P_MAX);

Table altered.

SQL> alter table range_part_tab SPLIT PARTITION P_MAX  at (TO_DATE('2014-06-01', 'YYYY-MM-DD')) into (PARTITION p_201406  ,PARTITION P_MAX);

Table altered.

select segment_name,
       partition_name,
       segment_type,
       bytes / 1024 / 1024 "M",
       tablespace_name
  from user_segments
  7   where segment_name IN('RANGE_PART_TAB');

SEGMENT_NAME         PARTITION_NAME       SEGMENT_TYPE                  M TABLESPACE_NAME
-------------------- -------------------- ------------------ ---------- ---------------
RANGE_PART_TAB       P_201301             TABLE PARTITION           .0625 TBS_1
RANGE_PART_TAB       P_201302             TABLE PARTITION           .0625 TBS_1
RANGE_PART_TAB       P_201405             TABLE PARTITION           .0625 TBS_1
RANGE_PART_TAB       P_201304             TABLE PARTITION              4 TBS_1
RANGE_PART_TAB       P_201305             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201306             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201307             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201308             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201309             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201310             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201311             TABLE PARTITION              4 TBS_1
RANGE_PART_TAB       P_201312             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201401             TABLE PARTITION              5 TBS_1
RANGE_PART_TAB       P_201402             TABLE PARTITION              4 TBS_1
RANGE_PART_TAB       P_201406             TABLE PARTITION           .0625 TBS_1
RANGE_PART_TAB       P_201404             TABLE PARTITION           .0625 TBS_1
RANGE_PART_TAB       P_MAX                TABLE PARTITION           .0625 TBS_1

17 rows selected.
```

1.6 exchage 操作方便

```
SQL> select count(*) from NORM_TAB;

  COUNT(*)
----------
    100000

SQL> create table norm_tab as select * from norm_tab where rownum <1000;

SQL> create table norm_tab2 as select * from norm_tab where rownum <1000;

Table created.

SQL> select count(*) from norm_tab2;

  COUNT(*)
----------
       999

SQL> select count(*) from RANGE_PART_TAB partition(P_201404);

SQL> select count(*) from RANGE_PART_TAB partition(P_201404);

  COUNT(*)
----------
         0
```

```
SQL> alter table RANGE_PART_TAB exchange partition P_201404 with table norm_tab2;
alter table RANGE_PART_TAB exchange partition P_201404 with table norm_tab2
                                                                 *
ERROR at line 1:
ORA-14099: all rows in table do not qualify for specified partition


SQL> alter table RANGE_PART_TAB exchange partition P_201404 with table norm_tab2 without validation;

Table altered.

SQL> select count(*) from norm_tab2;

  COUNT(*)
----------
         0

SQL> select count(*) from RANGE_PART_TAB partition(P_201404);

  COUNT(*)
----------
       999
```

不验证数据是否满足分区条件

如果有索引的话可以加上 including indexes update global indexes

2.全局临时表的好处

- 说说全局临时表的主要好处是什么。

2.1 自动清理数据

```
create global temporary table ljb_tmp_session on commit preserve rows
  2  as select  * from dba_objects where 1=2;

Table created.

SQL> select table_name,temporary,duration from user_tables  where table_name='LJB_TMP_SESSION';

TABLE_NAME                     T DURATION
------------------------------ - ---------------
LJB_TMP_SESSION                Y SYS$SESSION

create global temporary table ljb_tmp_transaction on commit delete rows
  2  as select * from dba_objects where 1=2;

Table created.

SQL> select table_name, temporary, DURATION from user_tables  where table_name='LJB_TMP_TRANSACTION';

TABLE_NAME                     T DURATION
------------------------------ - ---------------
LJB_TMP_TRANSACTION            Y SYS$TRANSACTION
```

分别创建基于session及基于事务的全局临时表

```
insert all
   into  ljb_tmp_transaction
   into  ljb_tmp_session
  4  select * from dba_objects;

18882 rows created.

select session_cnt,transaction_cnt from (select count(*) session_cnt from ljb_tmp_session),
  2   (select count(*) transaction_cnt from ljb_tmp_transaction);

SESSION_CNT TRANSACTION_CNT
----------- ---------------
       9441            9441

SQL> commit;

Commit complete.

select session_cnt,transaction_cnt from (select count(*) session_cnt from ljb_tmp_session),
  2  (select count(*) transaction_cnt from ljb_tmp_transaction);

SESSION_CNT TRANSACTION_CNT
----------- ---------------
       9441               0

SQL> disc
Disconnected from Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
SQL> conn scott/tiger
Connected.
select session_cnt,transaction_cnt from (select count(*) session_cnt from ljb_tmp_session),
  2  (select count(*) transaction_cnt from ljb_tmp_transaction);

SESSION_CNT TRANSACTION_CNT
----------- ---------------
          0               0
```

提交基于事务的全局临时表数据被清空，基于session的数据仍在

退出基于session的全局临时表数据被清空

2.2 不同 session 数据独立

```
SQL> drop table ljb_tmp_session;

Table dropped.                    创建两种类型的全局临时表

create global temporary table ljb_tmp_session on commit preserve rows
  2  as select  * from dba_objects where 1=2;

Table created.

select table_name,temporary,duration from user_tables
  2  where table_name='LJB_TMP_SESSION';

TABLE_NAME                     T DURATION
------------------------------ - ---------------
LJB_TMP_SESSION                Y SYS$SESSION

SQL> drop table  ljb_tmp_transaction;

Table dropped.

create global temporary table ljb_tmp_transaction on commit delete rows
  2  as select * from dba_objects where 1=2;

Table created.

select table_name, temporary, DURATION from user_tables
  2  where table_name='LJB_TMP_TRANSACTION';

TABLE_NAME                     T DURATION
------------------------------ - ---------------
LJB_TMP_TRANSACTION            Y SYS$TRANSACTION
```

```
SQL> select sid from v$mystat where rownum=1;

       SID
----------
        50

SQL> insert into ljb_tmp_session select * from dba_objects where rownum <=10;

10 rows created.

SQL> commit;

Commit complete.

SQL> select count(*) from ljb_tmp_session;

  COUNT(*)
----------
        10
```

```
SQL> select sid from v$mystat where rownum=1;

       SID
----------
        31

SQL> insert into ljb_tmp_session select * from dba_objects where rownum <=20;

20 rows created.

SQL> select count(*) from ljb_tmp_session;

  COUNT(*)
----------
        20
```

另开一session，原session不关闭，就算在sid=50中提交一次，在sid=31中也看不到之前的10条数据

2.3 产生的日志较少

```
SQL> create table t as select * from dba_objects ;

Table created.

select a.name,b.value
from v$statname a,v$mystat b
  3  where a.statistic#=b.statistic# and a.name='redo size';

NAME                                                                  VALUE
---------------------------------------------------------------- ----------
redo size                                                        1045452

SQL> delete t;

9441 rows deleted.

SQL> commit;

Commit complete.

select a.name,b.value
from v$statname a,v$mystat b
  3  where a.statistic#=b.statistic# and a.name='redo size';

NAME                                                                  VALUE
---------------------------------------------------------------- ----------
redo size                                                        4455452

SQL> select 4455452-1045452 as redo from dual;

      REDO
----------
   3410000
```

只演示delete

普通堆栈表一次delete产生的日志数

```
select a.name,b.value
from v$statname a,v$mystat b          全局临时表产生的日志数
  3  where a.statistic#=b.statistic# and a.name='redo size';

NAME                                                                  VALUE
---------------------------------------------------------------- ----------
redo size                                                          4455452

SQL> insert into LJB_TMP_TRANSACTION select * from dba_objects;

9441 rows created.

SQL> commit;

Commit complete.

select a.name,b.value
from v$statname a,v$mystat b
  3  where a.statistic#=b.statistic# and a.name='redo size';

NAME                                                                  VALUE
---------------------------------------------------------------- ----------
redo size                                                          4505092

SQL> select 4505092-4455452 as redosize from dual;

  REDOSIZE
----------
     49640
```

3.分区表性能比普通表差的情况

◆ 举例说明一下什么时候使用分区表性能比普通表还差。

3.1 构造分区表

```
create table part_tab (id int,col2 int,col3 int)
       partition by range (id)
       (
       partition p1 values less than (10000),
       partition p2 values less than (20000),
       partition p3 values less than (30000),
       partition p4 values less than (40000),
       partition p5 values less than (50000),
       partition p6 values less than (60000),
       partition p7 values less than (70000),
       partition p8 values less than (80000),
       partition p9 values less than (90000),
       partition p10 values less than (100000),
       partition p11 values less than (maxvalue)
 15    );

Table created.
```

```
insert into part_tab
  2  select rownum,rownum+1,rownum+2 from dual connect by rownum <=110000;

110000 rows created.

SQL> commit;

Commit complete.

SQL> create  index idx_par_tab_col2 on part_tab(col2) local;

Index created.

SQL> create  index idx_par_tab_col3 on part_tab(col3) ;

Index created.
```

构造数据并分别创建全局索引和本地索引

3.2 构造普通堆栈表

```
SQL> drop table norm_tab purge;

Table dropped.

SQL> create table norm_tab  (id int,col2 int,col3 int);

Table created.

insert into norm_tab
  2  select rownum,rownum+1,rownum+2 from dual connect by rownum <=110000;

110000 rows created.

SQL> commit;

Commit complete.

SQL> create  index idx_nor_tab_col2 on norm_tab(col2);

Index created.

SQL> create  index idx_nor_tab_col3 on norm_tab(col3);

Index created.
```

普通表与分区表结构及数据相同

3.3 第 1 次实验比较

```
SQL> select * from part_tab where col2=8 ;

Execution Plan
----------------------------------------------------------
Plan hash value: 2955748241

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name            | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                 |     1 |    39 |    13   (0)| 00:00:01 |       |       |
|   1 |  PARTITION RANGE ALL               |                 |     1 |    39 |    13   (0)| 00:00:01 |     1 |    11 |
|   2 |   TABLE ACCESS BY LOCAL INDEX ROWID| PART_TAB        |     1 |    39 |    13   (0)| 00:00:01 |     1 |    11 |
|*  3 |    INDEX RANGE SCAN                | IDX_PAR_TAB_COL2|     1 |       |    12   (0)| 00:00:01 |     1 |    11 |
-------------------------------------------------------------------------------------------------------------

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

   3 - access("COL2"=8)
```

```
Note
-----
   - dynamic sampling used for this statement


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         24  consistent gets
          0  physical reads
          0  redo size
        519  bytes sent via SQL*Net to client
        384  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

```
SQL> select * from norm_tab where col2=8 ;

                                                从代价上看还是普通表开销小
Execution Plan
----------------------------------------------------------
Plan hash value: 3649198428

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                 |     1 |    39 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID       | NORM_TAB        |     1 |    39 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                 | IDX_NOR_TAB_COL2|     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

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

   2 - access("COL2"=8)
```

```
Statistics
----------------------------------------------------------
         32  recursive calls
          0  db block gets
         79  consistent gets
          6  physical reads
          0  redo size
        523  bytes sent via SQL*Net to client
        384  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

```
select index_name,
         blevel,
         leaf_blocks,                                  查看索引高度都是1
         num_rows,
         distinct_keys,
         clustering_factor
  7  from user_ind_statistics where table_name in( 'NORM_TAB');

INDEX_NAME                     BLEVEL LEAF_BLOCKS   NUM_ROWS DISTINCT_KEYS CLUSTERING_FACTOR
------------------------------ ---------- ----------- ---------- ------------- -----------------
IDX_NOR_TAB_COL3                        1         244     110000        110000               299
IDX_NOR_TAB_COL2                        1         244     110000        110000               299

select index_name,
         blevel,
         leaf_blocks,
         num_rows,
         distinct_keys,
         clustering_factor
  7  FROM USER_IND_PARTITIONS where index_name like 'IDX_PAR_TAB%';

INDEX_NAME                     BLEVEL LEAF_BLOCKS   NUM_ROWS DISTINCT_KEYS CLUSTERING_FACTOR
------------------------------ ---------- ----------- ---------- ------------- -----------------
IDX_PAR_TAB_COL2                        1          21       9999          9999                24
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
IDX_PAR_TAB_COL2                        1          23      10001         10001                28
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
IDX_PAR_TAB_COL2                        1          23      10000         10000                28
```

3.4 第 2 次实验比较

```
SQL> set autotrace traceonly
SQL> select * from part_tab where col2=8 and id=2;

no rows selected


Execution Plan
----------------------------------------------------------
Plan hash value: 702898905

-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name             | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                  |     1 |    39 |     2   (0)| 00:00:01 |       |       |
|   1 |  PARTITION RANGE SINGLE            |                  |     1 |    39 |     2   (0)| 00:00:01 |     1 |     1 |
|*  2 |   TABLE ACCESS BY LOCAL INDEX ROWID| PART_TAB         |     1 |    39 |     2   (0)| 00:00:01 |     1 |     1 |
|*  3 |    INDEX RANGE SCAN                | IDX_PAR_TAB_COL2 |     1 |       |     1   (0)| 00:00:01 |     1 |     1 |
-------------------------------------------------------------------------------------------------------------------

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

   2 - filter("ID"=2)
   3 - access("COL2"=8)
```

```
Note
-----
   - dynamic sampling used for this statement


Statistics
----------------------------------------------------------
          5  recursive calls
          0  db block gets
         46  consistent gets
          0  physical reads
          0  redo size
        379  bytes sent via SQL*Net to client
        373  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

```
SQL> select * from norm_tab where col2=8 and id=2;

no rows selected


Execution Plan
----------------------------------------------------------
Plan hash value: 3649198428

-------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                  |    11 |   429 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| NORM_TAB         |    11 |   429 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | IDX_NOR_TAB_COL2 |     3 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   1 - filter("ID"=2)
   2 - access("COL2"=8)
```

从cost值来看目前两张表消耗都差不多，但作为分区表和普通表消耗一样，这本身就是输给了普通表

```
Statistics
----------------------------------------------------------
          5  recursive calls
          0  db block gets
         74  consistent gets
          0  physical reads
          0  redo size
        379  bytes sent via SQL*Net to client
        373  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

普通表一致性读洛大

3.5 第 3 次实验比较

```
SQL> select * from part_tab where col3=8 ;

Execution Plan
----------------------------------------------------------
Plan hash value: 3488710646

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name            | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                 |     1 |    39 |     2   (0)| 00:00:01 |       |       |
|   1 |  TABLE ACCESS BY GLOBAL INDEX ROWID| PART_TAB        |     1 |    39 |     2   (0)| 00:00:01 | ROWID | ROWID |
|*  2 |   INDEX RANGE SCAN                  | IDX_PAR_TAB_COL3 |     1 |       |     1   (0)| 00:00:01 |       |       |
-------------------------------------------------------------------------------------------------------------

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

   2 - access("COL3"=8)
```

```
Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          4  consistent gets
          0  physical reads
          0  redo size
        523  bytes sent via SQL*Net to client
        384  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

```
SQL> select * from norm_tab where col3=8 ;

Execution Plan
----------------------------------------------------------
Plan hash value: 363714236

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                 |     1 |    39 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| NORM_TAB        |     1 |    39 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_NOR_TAB_COL3 |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

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

   2 - access("COL3"=8)
```

```
Statistics
----------------------------------------------------------
          9  recursive calls
          0  db block gets
         77  consistent gets
          6  physical reads
          0  redo size
        523  bytes sent via SQL*Net to client
        384  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

这次比较分区表和普通表cost消耗上相同，普通表要比分区一致性度高一点。其实还应该存在另外一种情况，如果查询结果跨多个分区，分区表的效率将会下降很多