

如何将一个普通表转换为分区表

1.1 BLOG 文档结构图

└─ 如何将一个普通表转换为分区表
1.1 BLOG 文档结构图
└─ 1.2 前言部分
1.2.1 导读和注意事项
1.2.2 相关参考文章链接
1.2.3 本文简介
----- ...
└─ 第 2 章非分区表转换为分区表的 4 种方法
└─ 2.1 Export/import method
└─ 2.2 利用原表重建分区表(插入)
└─ 2.3 使用交换分区的方法(Partition exchange method...)
2.3.1 单个分区示例
└─ 2.3.2 多个分区示例
2.3.2.1 MOS 上的例子
└─ 2.4 利用在线重定义功能(DBMS_REDEFINITION)
└─ 2.4.1 在线重定义的相关知识
2.4.2 我的示例
└─ ----- ...
2.5 注意
About Me

1.2 前言部分

1.2.1 导读和注意事项

各位技术爱好者，看完本文后，你可以掌握如下的技能，也可以学到一些其它你所不知道的知识，~O(∩_∩)O~：

① 将一个普通表转换为分区表的常用方法（重点）

② 在线重定义的使用

③ ctas 和 insert 的优化

④ DML 语句如何开启并行操作，如何查看 DML 是否开启了并行

Tips：

① 若文章代码格式有错乱，推荐使用 QQ、搜狗或 360 浏览器，也可以下载 pdf 格式的文档来查看，pdf 文档下载地址 <http://yunpan.cn/cdEQedhCs2kFz>（提取码：ed9b）

② 本篇 BLOG 中命令的输出部分需要特别关注的地方我都用灰色背景和粉红色字体来表示，比如下边的例子中，thread 1 的最大归档日志号为 33，thread 2 的最大归档日志号为 43 是需要特别关注的地方；而命令一般使用黄色背景和红色字体标注；对代码或代码输出部分的注释一般采用蓝色字体表示。

```
List of Archived Logs in backup set 11
Thrd Seq    Low SCN    Low Time    Next SCN    Next Time
-----

```

1	32	1621589	2015-05-29 11:09:52	1625242	2015-05-29 11:15:48
1	33	1625242	2015-05-29 11:15:48	1625293	2015-05-29 11:15:58
2	42	1613951	2015-05-29 10:41:18	1625245	2015-05-29 11:15:49
2	43	1625245	2015-05-29 11:15:49	1625253	2015-05-29 11:15:53

```
[ZHLHRDB1:root]:/>ls -l
```

```
T_XDESK_APP1_vg
```

```
rootvg
```

```
[ZHLHRDB1:root]:/>
```

```
00:27:22 SQL> alter tablespace idxtbs read write;
```

```
====> 2097152*512/1024/1024/1024=1G
```

本文如有错误或不完善的地方请大家多多指正，ITPUB 留言或 QQ 皆可，您的批评指正正是我写作的最大动力。

1.2.2 相关参考文章链接

参考文档都是 MOS 上 How to Partition a Non-partitioned / Regular / Normal Table (文档 ID 1070693.6)，已上传到云盘，大家可自行下载。

1.2.3 本文简介

本文介绍了 4 种非分区表转换为分区表的几种方法，参考文档来自于 MOS。

将普通表转换成分区表有 4 种方法，这个在 MOS 文档上有说明 (How to Partition a Non-partitioned / Regular / Normal Table (文档 ID 1070693.6))：

1. Export/import method
2. Insert with a subquery method
3. Partition exchange method
4. DBMS_REDEFINITION



How to Partition a Non-partitioned , Regular , Normal Table (文档 ID 1070693.6).mhtml

第 2 章 非分区表转换为分区表的 4 种方法

2.1 Export/import method

采用逻辑导出导入很简单，首先在源库建立分区表，然后将数据导出，然后导入到新建的分区表即可，

- 1) 导出表：exp usr/pswd tables=numbers file=exp.dmp
- 2) 删除表：drop table numbers;

创建普通表并插入测试数据

Table created.

```
87069 rows created.
```

Commit complete.

TO_CHA	COUNT(1)
201310	85984
201605	1107

采用 expdp 导出表

Export: Release 11.2.0.4.0 - Production on Fri May 27 11:07:46 2016

Copyright (c) 1982, 2011, Oracle and/or its affiliates. All rights reserved.

```
Connected to: Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options
Starting "SYS"."SYS_EXPORT_SCHEMA_01":  "/***** AS SYSDBA" directory=DATA_PUMP_DIR dumpfile=lhr_t.dmp INCLUDE=TABLE:"IN ('T')" SCHEMAS=LHR LOGFILE=expdp_T.log
Estimate in progress using BLOCKS method...
Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 2 MB
Processing object type SCHEMA_EXPORT/TABLE/TABLE
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
. . exported "LHR"."T"                      1.406 MB   87091 rows
Master table "SYS"."SYS_EXPORT_SCHEMA_01" successfully loaded/unloaded
*****
Dump file set for SYS.SYS_EXPORT_SCHEMA_01 is:
/oracle/app/oracle/admin/dlhr/dpdump/lhr_t.dmp
Job "SYS"."SYS_EXPORT_SCHEMA_01" successfully completed at Fri May 27 11:07:57 2016 elapsed 0 00:00:11
```

删除原表，创建一个分区表结构：

```
LHR@dlhr> drop table t;

Table dropped.

LHR@dlhr> CREATE TABLE T (ID NUMBER PRIMARY KEY, TIME DATE )
2 PARTITION BY RANGE (TIME)
3 (PARTITION T1 VALUES LESS THAN (TO_DATE('201311', 'YYYYMM')),
4 PARTITION T2 VALUES LESS THAN (TO_DATE('201606', 'YYYYMM')),
5 PARTITION T3 VALUES LESS THAN (MAXVALUE))
6 ;

Table created.

LHR@dlhr>
```

导入到分区表

```
[ZFXDESKDB2:oracle]:/tmp> impdp \'/ as sysdba\ ' directory=DATA_PUMP_DIR dumpfile=lhr_t.dmp SCHEMAS=LHR table_exists_action=APPEND LOGFILE=impdp_T.log

Import: Release 11.2.0.4.0 - Production on Fri May 27 11:12:40 2016

Copyright (c) 1982, 2011, Oracle and/or its affiliates. All rights reserved.
```

```

Connected to: Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options
Master table "SYS"."SYS_IMPORT_SCHEMA_01" successfully loaded/unloaded
Starting "SYS"."SYS_IMPORT_SCHEMA_01":  "/***** AS SYSDBA" directory=DATA_PUMP_DIR dumpfile=ldr_t.dmp SCHEMAS=LHR table_exists_action=APPEND LOGFILE=impdp_T.log
Processing object type SCHEMA_EXPORT/TABLE/TABLE
Table "LHR"."T" exists. Data will be appended to existing table but all dependent metadata will be skipped due to table_exists_action of append
Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
. . imported "LHR"."T"                1.406 MB   87091 rows
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Job "SYS"."SYS_IMPORT_SCHEMA_01" successfully completed at Fri May 27 11:12:46 2016 elapsed 0 00:00:05

[ZFXDESKDB2:oracle]:/tmp>

```

查询导入后的情况：

```

SYS@dlhr> select to_char(t.time, 'YYYYMM'), COUNT(1)
2      from t
3      group by to_char(t.time, 'YYYYMM');

```

TO_CHA	COUNT(1)
201310	85984
201605	1083

```

SYS@dlhr> SELECT D.TABLE_OWNER,D.TABLE_NAME,D.PARTITION_NAME FROM DBA_TAB_PARTITIONS d WHERE d.table_name='T';

```

TABLE_OWNER	TABLE_NAME	PARTITION_NAME
LHR	T	T1
LHR	T	T2
LHR	T	T3

```

SYS@dlhr>

```

2.2 利用原表重建分区表(插入)

这种方法的特点是：

优点：方法简单易用，由于采用 DDL 语句，不会产生 UNDO，且只产生少量 REDO，效率相对较高，而且建表完成后数据已经在分布到各个分区中了。

不足：对于数据的一致性方面还需要额外的考虑。由于几乎没有办法通过手工锁定 T 表的方式保证一致性，在执行 CREATE TABLE 语句和 RENAME T_NEW TO T 语句直接的修改可能会丢失，如果要保证一致性，需要在执行完语句后对数据进行检查，而这个代价是比较大的。另外在执行两个 RENAME 语句之间执行的对 T 的访问会失败。

适用于修改不频繁的表，在闲时进行操作，表的数据量不宜太大。

主要有 2 种方式，ctas 和 insert 方式，下边分别介绍：

2.2.1 例一：CTAS+RENAME

利用 CTAS 语法在创建分区表的时候可以一起插入数据，也可以创建好表结构再 insert 进去。CTAS 这种方法采用 DDL 语句，不产生 UNDO，只产生少量 REDO，建表完成后数据已经在分布到各个分区中。

创建普通表并插入测试数据

```
LHR@dlhr> CREATE TABLE T (ID NUMBER PRIMARY KEY, TIME DATE);  
Table created.  
LHR@dlhr> INSERT INTO T SELECT ROWNUM, CREATED FROM DBA_OBJECTS;  
87069 rows created.  
LHR@dlhr> commit;  
Commit complete.  
LHR@dlhr> select to_char(t.time, 'YYYYMM'), COUNT(1)  
2      from t  
3      group by to_char(t.time, 'YYYYMM');
```


TO_CHA	COUNT(1)
201310	85984
201605	1085

创建一个分区表，注意这里的分区表的列后边没有数据类型：

```
LHR@dlhr> CREATE TABLE T_NEW (ID, TIME) PARTITION BY RANGE (TIME)
2 (PARTITION T1 VALUES LESS THAN (TO_DATE('201311', 'YYYYMM')),
3 PARTITION T2 VALUES LESS THAN (TO_DATE('201606', 'YYYYMM')),
4 PARTITION T3 VALUES LESS THAN (MAXVALUE))
5 AS SELECT ID, TIME FROM T;
```

Table created.

LHR@dlhr>

改变表名

Table renamed.

```
LHR@dlhr> rename t_new to t;
```

Table renamed.

验证新表数据

```
LHR@dlhr> select to_char(t.time, 'YYYYMM'), COUNT(1)
2 from t
3 group by to_char(t.time, 'YYYYMM');
```

TO_CHA	COUNT(1)
201310	85984
201605	1085

LHR@dlhr>

2.2.1.1 ctas 性能提升

对于 CTAS 建表语句性能的提升可以通过如下的方式，① 加 nologging ② 并行 DDL ③ 查询并行，需要说明的是建表完成后根据需要将表修改为 logging 模式。

```
CREATE TABLE T_NEW (ID, TIME)
PARTITION BY RANGE (TIME)
(PARTITION T1 VALUES LESS THAN (TO_DATE('201311', 'YYYYMM')),
 PARTITION T2 VALUES LESS THAN (TO_DATE('201606', 'YYYYMM')),
 PARTITION T3 VALUES LESS THAN (MAXVALUE))
nologging parallel 4
AS SELECT /*+PARALLEL*/ ID, TIME FROM T;
```

执行计划：

```
SYS@dlhr> explain plan for CREATE TABLE T_NEW (ID, TIME)
2  PARTITION BY RANGE (TIME)
3      (PARTITION T1 VALUES LESS THAN (TO_DATE(' 201311', 'YYYYMM')),
4      PARTITION T2 VALUES LESS THAN (TO_DATE(' 201606', 'YYYYMM')),
5      PARTITION T3 VALUES LESS THAN (MAXVALUE))
6      nologging parallel 4
7  AS SELECT /*+PARALLEL*/ ID, TIME FROM T;
```

Explained.

```
SYS@dlhr> select * from table(dbms_xplan.display());
```

PLAN_TABLE_OUTPUT

Plan hash value: 4064487821

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	TQ	IN-OUT	PQ Distrib
0	CREATE TABLE STATEMENT		82787	1778K	14 (0)	00:00:01			
1	PX COORDINATOR								
2	PX SEND QC (RANDOM)	:TQ10000	82787	1778K	4 (0)	00:00:01	Q1,00	P->S	QC (RAND)
3	LOAD AS SELECT	T_NEW					Q1,00	PCWP	
4	PX BLOCK ITERATOR		82787	1778K	4 (0)	00:00:01	Q1,00	PCWC	
5	TABLE ACCESS FULL	T	82787	1778K	4 (0)	00:00:01	Q1,00	PCWP	

Note

```
- dynamic sampling used for this statement (level=2)
- automatic DOP: skipped because of IO calibrate statistics are missing
```

17 rows selected.

SYS@dlhr>

可以看到对 T 表的查询是并行的，create table 也是并行的，这在源表的数据量非常大的情况下性能显著。

2.2.2 例二：Insert with a subquery method

这种方法就是先建立表结构然后使用 insert 来实现。

看示例：

创建普通表 T_LHR_20160527

```
LHR@dlhr> CREATE TABLE T_LHR_20160527 (ID NUMBER PRIMARY KEY, TIME DATE);
```

Table created.

```
LHR@dlhr> INSERT INTO T_LHR_20160527 SELECT ROWNUM, CREATED FROM DBA_OBJECTS;
```

87098 rows created.

```
LHR@dlhr> commit;
```

Commit complete.

```
LHR@dlhr> select to_char(t.time, 'YYYYMM'), COUNT(1)
2      from T_LHR_20160527 t
3      group by to_char(t.time, 'YYYYMM');
```

```
TO_CHA    COUNT(1)
-----
```

201310	85984
201605	1114

创建一个分区表 T_LHR_20160527_NEW :

```
LHR@dlhr> CREATE TABLE T_LHR_20160527_NEW (ID NUMBER, TIME DATE)
2 PARTITION BY RANGE (TIME)
3   (PARTITION T1 VALUES LESS THAN (TO_DATE('201311', 'YYYYMM')),
4     PARTITION T2 VALUES LESS THAN (TO_DATE('201606', 'YYYYMM')),
5     PARTITION T3 VALUES LESS THAN (MAXVALUE));
```

Table created.

从源表查询插入到新表中 :

```
LHR@dlhr> alter table T_LHR_20160527_NEW nologging;
```

Table altered.

```
LHR@dlhr> alter session enable parallel dml;
```

Session altered.

```
LHR@dlhr> insert /*+APPEND PARALLEL*/ into T_LHR_20160527_NEW (ID, TIME) select * from T_LHR_20160527;
```

87098 rows created.

```
LHR@dlhr> commit;
```

Commit complete.

删除源表，重命名新表

```
LHR@dlhr> drop table T_LHR_20160527;
```

Table dropped.

```
LHR@dlhr> rename T_LHR_20160527_NEW to T_LHR_20160527;
```

Table renamed.

验证新表数据：

```
LHR@dlhr> select to_char(t.time, 'YYYYMM'), COUNT(1)
2      from T_LHR_20160527 t
3      group by to_char(t.time, 'YYYYMM');
```

TO_CHA	COUNT(1)
201310	85984
201605	1114

LHR@dlhr>

2.2.2.1 insert 性能提升

INSERT 性能提升的方式，① 表修改为 nologging ② 禁用表上的索引，可以将数据插入完成后再建索引 ③ 启用并行 DML alter session enable parallel

dml; ④ 采用 append 方式插入

```
commit;
alter session enable parallel dml;
alter table T_LHR_20160527_NEW nologging;
insert /*+APPEND PARALLEL*/ into T_LHR_20160527_NEW (ID, TIME) select /*+PARALLEL(t3,4)*/ * from T_LHR_20160527;
```

采用并行 DML 必须执行 alter session enable parallel dml; 才可以启用并行 DML, 执行计划:

```
LHR@dlhr> explain plan for insert /*+APPEND PARALLEL*/ into T_LHR_20160527 (ID, TIME) select /*+PARALLEL(t3,4)*/ * from t3;
```

Explained.

```
LHR@dlhr> select * from table(dbms_xplan.display());
```

PLAN_TABLE_OUTPUT

Plan hash value: 584641640

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	TQ	IN-OUT	PQ Distrib
0	INSERT STATEMENT		6897K	144M	272 (4)	00:00:04			
1	LOAD AS SELECT	T_LHR_20160527							
2	PX COORDINATOR								
3	PX SEND QC (RANDOM)	:TQ10000	6897K	144M	272 (4)	00:00:04	Q1,00	P->S	QC (RAND)
4	PX BLOCK ITERATOR		6897K	144M	272 (4)	00:00:04	Q1,00	PCWC	
5	TABLE ACCESS FULL	T3	6897K	144M	272 (4)	00:00:04	Q1,00	PCWP	

Note

- dynamic sampling used for this statement (level=2)
- automatic DOP: skipped because of IO calibrate statistics are missing

17 rows selected.

LHR@dlhr> commit;

Commit complete.

LHR@dlhr> alter session enable parallel dml;

Session altered.

LHR@dlhr> explain plan for insert /*+APPEND PARALLEL*/ into T_LHR_20160527 (ID, TIME) select /*+PARALLEL(t3,4)*/* from t3;

Explained.

LHR@dlhr> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT

Plan hash value: 576433284

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	TQ	IN-OUT	PQ Distrib
0	INSERT STATEMENT		6897K	144M	272 (4)	00:00:04			
1	PX COORDINATOR								
2	PX SEND QC (RANDOM)	:TQ10000	6897K	144M	272 (4)	00:00:04	Q1,00	P->S	QC (RAND)
3	LOAD AS SELECT	T_LHR_20160527					Q1,00	PCWP	
4	PX BLOCK ITERATOR		6897K	144M	272 (4)	00:00:04	Q1,00	PCWC	
5	TABLE ACCESS FULL	T3	6897K	144M	272 (4)	00:00:04	Q1,00	PCWP	

Note

- dynamic sampling used for this statement (level=2)
- automatic DOP: skipped because of IO calibrate statistics are missing

17 rows selected.

LHR@dlhr>

2.3 使用交换分区的方法(Partition exchange method)

这种方法的特点

优点：只是对数据字典中分区和表的定义进行了修改，没有数据的修改或复制，效率最高。如果对数据在分区中的分布没有进一步要求的话，实现比较简单。在执行完 RENAME 操作后，可以检查 T_OLD 中是否存在数据，如果存在的话，直接将这些数据插入到 T 中，可以保证对 T 插入的操作不会丢失。

不足：仍然存在一致性问题，交换分区之后 RENAME T_NEW TO T 之前，查询、更新和删除会出现错误或访问不到数据。如果要求数据分布到多个分区中，则需要进行分区的 SPLIT 操作，会增加操作的复杂度，效率也会降低。

适用于包含大数据量的表转到分区表中的一个分区的操作。应尽量在闲时进行操作。

2.3.1 单个分区示例

举例来说明

创建普通表并插入测试数据

```
LHR@dlhr> CREATE TABLE T (ID NUMBER PRIMARY KEY, TIME DATE);
```

Table created.

```
LHR@dlhr> INSERT INTO T SELECT ROWNUM, CREATED FROM DBA_OBJECTS where CREATED<=to_date('201311','YYYYMM');
```

85984 rows created.

```
LHR@dlhr> COMMIT;
```

Commit complete.

```
LHR@dlhr> select to_char(t.time, 'YYYYMM'), COUNT(1)
2      from t
3      group by to_char(t.time, 'YYYYMM');
```

TO_CHA	COUNT(1)
201310	85984

创建分区表

```
LHR@dlhr> CREATE TABLE T_NEW (ID NUMBER PRIMARY KEY, TIME DATE) PARTITION BY RANGE (TIME)
2      (PARTITION T1 VALUES LESS THAN (TO_DATE('2013-11-1', 'YYYY-MM-DD')),
3      PARTITION T2 VALUES LESS THAN (MAXVALUE));
```

Table created.

交换数据

```
LHR@dlhr> ALTER TABLE T_NEW EXCHANGE PARTITION T1 WITH TABLE T;
```

Table altered.

改变表名

```
LHR@dlhr> rename t to t_old;
```

Table renamed.


```
LHR@dlhr> rename t_new to t;
```

Table renamed.

查询数据

```
LHR@dlhr> select to_char(t.time, 'YYYYMM'), COUNT(1)
2      from t
3      group by to_char(t.time, 'YYYYMM');
```

TO_CHA	COUNT(1)
201310	85984

2.3.2 多个分区示例

交换分区操作步骤如下：

1. 创建分区表，假设有 2 个分区，P1，P2。
2. 创建表 A 存放 P1 规则的数据。
3. 创建表 B 存放 P2 规则的数据。
4. 用表 A 和 P1 分区交换。把表 A 的数据放到 P1 分区
5. 用表 B 和 P2 分区交换。把表 B 的数据存放到 P2 分区。

2.3.2.1 MOS 上的例子

This example creates the exchange table with the same structure as the partitions of the partitioned table p_emp.

```
SQL> CREATE TABLE p_emp
2 (sal NUMBER(7,2))
3 PARTITION BY RANGE(sal)
4 (partition emp_p1 VALUES LESS THAN (2000),
5 partition emp_p2 VALUES LESS THAN (4000));
```

Table created.

```
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

14 rows selected.

```
SQL> CREATE TABLE exhtab1 as SELECT sal FROM emp WHERE sal<2000;
```

Table created.

```
SQL> CREATE TABLE exhtab2 as SELECT sal FROM emp WHERE sal BETWEEN 2000 AND 3999;
```

Table created.

```
SQL> alter table p_emp exchange partition emp_p1 with table exhtab1;
```

Table altered.

```
SQL> alter table p_emp exchange partition emp_p2 with table exhtab2;
```

Table altered.

2.4 利用在线重定义功能(DBMS_REDEFINITION)

这种分区的特点

优点：保证数据的一致性，在大部分时间内，表 T 都可以正常进行 DML 操作。只在切换的瞬间锁表，具有很高的可用性。这种方法具有很强的灵活性，对各种不同的需要都能满足。而且，可以在切换前进行相应的授权并建立各种约束，可以做到切换完成后不再需要任何额外的管理操作。

不足：实现上比上面两种略显复杂。

适用于各种情况。

在线重定义的大致操作流程如下：

(1) 创建基础表 A，如果存在，就不需要操作。


(2) 创建临时的分区表 B 结构。

(3) 开始重定义，将基表 A 的数据导入临时分区表 B。

(4) 结束重定义，完成后在 DB 的 Name Directory 里，已经将 2 个表进行了交换。即此时基表 A 成了分区表，我们创建的临时分区表 B 成了普通表。此

时我们可以删除我们创建的临时表 B。它已经是普通表。

MOS 上的文档：

 How To Partition Existing Table Using DBMS_REDEFINITION (文档 ID 472449.1).mhtml

2.4.1 在线重定义的相关知识

2.4.1.1 在线重定义功能

这个功能只在 9.2.0.4 以后的版本才有，在线重定义表具有以下功能：

(1) 修改表的存储参数；

(2) 将表转移到其他表空间；

- (3) 增加并行查询选项；
- (4) 增加或删除分区；
- (5) 重建表以减少碎片；
- (6) 将堆表改为索引组织表或相反的操作；
- (7) 增加或删除一个列。

2. 4. 1. 2 在线重定义表的步骤

在线重定义的原理：物化视图

在线重定义表的步骤：

1. 选择一种重定义方法：

存在两种重定义方法，一种是基于主键、另一种是基于 ROWID。ROWID 的方式不能用于索引组织表，而且重定义后会存在隐藏列 M_ROW\$\$. 默认采用主键的方式。

2. 调用 DBMS_REDEFINITION.CAN_REDEF_TABLE() 过程，如果表不满足重定义的条件，将会报错并给出原因。

3. 在用一个方案中建立一个空的中间表，根据重定义后你期望得到的结构建立中间表。比如：采用分区表，增加了 COLUMN 等。

4. 调用 DBMS_REDEFINITION.START_REDEF_TABLE() 过程，并提供下列参数：被重定义的表的名称、中间表的名称、列的映射规则、重定义方法。

如果映射方法没有提供，则认为所有包括在中间表中的列用于表的重定义。如果给出了映射方法，则只考虑映射方法中给出的列。如果没有给出重定义方法，则认为使用主键方式。

5. 在中间表上建立触发器、索引和约束，并进行相应的授权。任何包含中间表的完整性约束应将状态置为 disabled。

当重定义完成时，中间表上建立的触发器、索引、约束和授权将替换重定义表上的触发器、索引、约束和授权。中间表上 disabled 的约束将在重定义表上 enable。

6. (可选) 如果在执行 DBMS_REDEFINITION.START_REDEF_TABLE() 过程和执行 DBMS_REDEFINITION.FINISH_REDEF_TABLE() 过程直接在重定义表上执行了大量的 DML 操作，那么可以选择执行一次或多次的 SYNC_INTERIM_TABLE() 过程，以减少最后一步执行 FINISH_REDEF_TABLE() 过程时的锁定时间。

7. 执行 DBMS_REDEFINITION.FINISH_REDEF_TABLE() 过程完成表的重定义。这个过程中，原始表会被独占模式锁定一小段时间，具体时间和表的数据量有关。

执行完 FINISH_REDEF_TABLE() 过程后，原始表重定义后具有了中间表的属性、索引、约束、授权和触发器。中间表上 disabled 的约束在原始表上处于 enabled 状态。

8. (可选) 可以重命名索引、触发器和约束。对于采用了 ROWID 方式重定义的表，包括了一个隐含列 M_ROW\$\$。推荐使用下列语句经隐含列置为 UNUSED 状态或删除。

```
ALTER TABLE TABLE_NAME SET UNUSED (M_ROW$$);  
ALTER TABLE TABLE_NAME DROP UNUSED COLUMNS;
```

2.4.1.3 使用在线重定义的限制条件

使用在线重定义的一些限制条件：

- (1) There must be enough space to hold two copies of the table.
- (2) Primary key columns cannot be modified.
- (3) Tables must have primary keys.
- (4) Redefinition must be done within the same schema.
- (5) New columns added cannot be made NOT NULL until after the redefinition operation.

- (6) Tables cannot contain LONGs, BFILEs or User Defined Types.
- (7) Clustered tables cannot be redefined.
- (8) Tables in the SYS or SYSTEM schema cannot be redefined.
- (9) Tables with materialized view logs or materialized views defined on them cannot be redefined.
- (10) Horizontal sub setting of data cannot be performed during the redefinition.

在 Oracle 10.2.0.4和11.1.0.7 版本下，在线重定义可能会遇到如下 bug:

Bug 7007594 - ORA-600 [12261]

<http://blog.csdn.net/tianlesoftware/archive/2011/03/02/6218681.aspx>

- 如果使用基于主键的方式，则原表后重定义后的表必须有相同的主键
- 如果使用基于 ROWID 的方式，则不能是索引组织表
- 如果原表上有物化视图或者物化视图日志，则不能在线重定义
- 物化视图容器表或者高级队列表不能在线重定义
- 索引组织表的溢出表不能在线重定义
- 拥有 BFILE，LOGN 列的表不能在线重定义
- Cluster 中的表不能在线重定义
- sys 和 system 下的表不能在线重定义
- 临时表不能在线重定义
- 不支持水平数据子集
- 在列映射时只能使用有确定结果的表达式，如子查询就不行

- 如果中间表有新增列，则不能有 NOT NULL 约束
- 原表和中间表之间不能有引用完整性
- 在线重定义无法采用 nologging

2.4.2 我的示例

创建普通表 T_LHR_20160527_UNPART 及其索引：

```
LHR@dlhr> CREATE TABLE T_LHR_20160527_UNPART (ID NUMBER PRIMARY KEY, TIME DATE);
Table created.

LHR@dlhr> INSERT INTO T_LHR_20160527_UNPART SELECT ROWNUM, CREATED FROM DBA_OBJECTS;
87112 rows created.

LHR@dlhr> commit;
Commit complete.

LHR@dlhr> CREATE INDEX create_date_indx ON T_LHR_20160527_UNPART(TIME);
Index created.

LHR@dlhr> exec dbms_stats.gather_table_stats(user, 'T_LHR_20160527_UNPART', cascade => true);
PL/SQL procedure successfully completed.

LHR@dlhr>

LHR@dlhr> select to_char(t.time, 'YYYYMM'), COUNT(1)
2  from T_LHR_20160527_UNPART t
3  group by to_char(t.time, 'YYYYMM');
```


TO_CHA	COUNT(1)
201310	85984
201605	1128

创建临时分区表 T_LHR_20160527_PART, 注意这里的 time 列我换成了 CREATED_DATE

```
LHR@dlhr> CREATE TABLE T_LHR_20160527_PART (ID NUMBER PRIMARY KEY, CREATED_DATE DATE)
2 PARTITION BY RANGE (created_date)
3 (PARTITION T1 VALUES LESS THAN (TO_DATE('201311', 'YYYYMM')),
4 PARTITION T2 VALUES LESS THAN (TO_DATE('201606', 'YYYYMM')),
5 PARTITION T3 VALUES LESS THAN (MAXVALUE));
```

Table created.

然后执行 DBMS_REDEFINITION.CAN_REDEF_TABLE(USER, 'T_LHR_20160527_UNPART', DBMS_REDEFINITION.CONS_USE_PK); 检查是否可以执行在线重

定义, 若返回错误的话说明不能执行, LHR@dlhr> EXEC DBMS_REDEFINITION.CAN_REDEF_TABLE(USER, 'T', DBMS_REDEFINITION.CONS_USE_PK);

```
BEGIN DBMS_REDEFINITION.CAN_REDEF_TABLE(USER, 'T', DBMS_REDEFINITION.CONS_USE_PK); END;
```

*

ERROR at line 1:

ORA-12089: cannot online redefine table "LHR"."T" with no primary key

ORA-06512: at "SYS.DBMS_REDEFINITION", line 143

ORA-06512: at "SYS.DBMS_REDEFINITION", line 1635

ORA-06512: at line 1

```
LHR@dlhr> EXEC DBMS_REDEFINITION.CAN_REDEF_TABLE(USER, 'T_LHR_20160527_UNPART', DBMS_REDEFINITION.CONS_USE_PK);
```

PL/SQL procedure successfully completed.

没有错误，说明我们需要转换的表可以执行在线重定义，下边开始执行在线重定义，这个过程可能要等一会，根据表的大小不同而不同：

```
LHR@dlhr> EXEC DBMS_REDEFINITION.START_REDEF_TABLE(USER, 'T_LHR_20160527_UNPART', 'T_LHR_20160527_PART', DBMS_REDEFINITION.CONSTRAINT_NAME);  
BEGIN DBMS_REDEFINITION.START_REDEF_TABLE(USER, 'T_LHR_20160527_UNPART', 'T_LHR_20160527_PART', DBMS_REDEFINITION.CONSTRAINT_NAME); END;
```

```
*  
ERROR at line 1:  
ORA-42016: shape of interim table does not match specified column mapping  
ORA-06512: at "SYS.DBMS_REDEFINITION", line 56  
ORA-06512: at "SYS.DBMS_REDEFINITION", line 1498  
ORA-06512: at line 1
```

```
LHR@dlhr> EXEC DBMS_REDEFINITION.START_REDEF_TABLE(USER, 'T_LHR_20160527_UNPART', 'T_LHR_20160527_PART', 'ID ID, TIME created_date ',  
DBMS_REDEFINITION.CONSTRAINT_NAME);
```

PL/SQL procedure successfully completed.

```
LHR@dlhr>
```

```
LHR@dlhr> select count(1) from T_LHR_20160527_UNPART;
```

```
  COUNT(1)  
-----  
      87112
```

```
LHR@dlhr> select count(1) from T_LHR_20160527_PART;
```

```
  COUNT(1)  
-----  
      87112
```

```
LHR@dlhr> EXEC DBMS_REDEFINITION.SYNC_INTERIM_TABLE(USER, 'T_LHR_20160527_UNPART', 'T_LHR_20160527_PART');
```

这一步操作结束后，数据就已经同步到这个临时的分区表里来了。需要注意的是如果分区表和原表列名相同，则可以不用加列的转换，如果不同的话需要加上转换，即重新指定映射关系。另外 EXEC DBMS_REDEFINITION.SYNC_INTERIM_TABLE(USER, 'T_LHR_20160527_UNPART', 'T_LHR_20160527_PART'); 是同步新表作

用是可选的。如果在执行 DBMS_REDEFINITION.START_REDEF_TABLE() 过程和执行 DBMS_REDEFINITION.FINISH_REDEF_TABLE() 过程直接在重定义表上执行了大量的 DML 操作，那么可以选择执行一次或多次的 SYNC_INTERIM_TABLE() 过程，以减少最后一步执行 FINISH_REDEF_TABLE() 过程时的锁定时间。

下边我们在新表上创建索引，在线重定义只重定义数据，索引还需要单独建立。

```
LHR@dlhr> CREATE INDEX create_date_indx2 ON T_LHR_20160527_PART(created_date);  
  
Index created.  
  
LHR@dlhr> exec dbms_stats.gather_table_stats(user, 'T_LHR_20160527_PART', cascade => true);  
  
PL/SQL procedure successfully completed.  
  
LHR@dlhr>
```

接下来就是结束重定义了：

```
LHR@dlhr> EXEC DBMS_REDEFINITION.FINISH_REDEF_TABLE(user, 'T_LHR_20160527_UNPART', 'T_LHR_20160527_PART');  
  
PL/SQL procedure successfully completed.  
  
LHR@dlhr>  
LHR@dlhr> select D.TABLE_NAME, partitioned from user_tables D where table_name like '%T_LHR_20160527%' ;  
  
TABLE_NAME                                PAR  
-----  
T_LHR_20160527_PART                      NO  
T_LHR_20160527_UNPART                    YES  
  
LHR@dlhr> SELECT D.TABLE_NAME, partition_name  
2 FROM user_tab_partitions D  
3 WHERE table_name = 'T_LHR_20160527_UNPART';  
  
TABLE_NAME                                PARTITION_NAME  
-----  
T_LHR_20160527_UNPART                    T1  
T_LHR_20160527_UNPART                    T2  
T_LHR_20160527_UNPART                    T3  
  
LHR@dlhr>
```

结束重定义 DBMS_REDEFINITION.FINISH_REDEF_TABLE 的意义：

基表 T_LHR_20160527_UNPART 和临时分区表 T_LHR_20160527_PART 进行了交换。此时临时分区表 T_LHR_20160527_PART 成了普通表，我们的基表 T_LHR_20160527_UNPART 成了分区表。

我们在重定义的时候，基表 T_LHR_20160527_UNPART 是可以进行 DML 操作的。只有在 2 个表进行切换的时候会有短暂的锁表。

在线重定义能保证数据的一致性，在大部分时间内，表都可以正常进行 DML 操作。只在切换的瞬间锁表，具有很高的可用性。这种方法具有很强的灵活性，对各种不同的需要都能满足。而且，可以在切换前进行相应的授权并建立各种约束，可以做到切换完成后不再需要任何额外的管理操作。

还有最后一个步骤，删除临时表并索引重命名，验证数据即可：

```
LHR@dlhr> drop table T_LHR_20160527_PART;

Table dropped.

LHR@dlhr> alter index create_date_idx2 rename to create_date_idx;

Index altered.

LHR@dlhr> select to_char(t.created_date, 'YYYYMM'), COUNT(1)
2   from T_LHR_20160527_UNPART t
3   group by to_char(t.created_date, 'YYYYMM');
```

TO_CHA	COUNT(1)
201310	85984
201605	1128

2.5 注意

文章中用的相关 MOS 文档已经上传到云盘大家可自行下载。

About Me

.....

本文作者：小麦苗，只专注于数据库的技术，更注重技术的运用

ITPUB BLOG：<http://blog.itpub.net/26736162>

本文地址：<http://blog.itpub.net/26736162/viewspace-2109454/>

本文 pdf 版：<http://yunpan.cn/cdEQedhCs2kFz>（提取码：ed9b）

QQ：642808185 若加 QQ 请注明您所正在读的文章标题

于 2016-05-23 10:00~ 2016-05-27 19:00 在中行完成

【版权所有，文章允许转载，但须以链接方式注明源地址，否则追究法律责任】

.....

