

## 第10章 分 区

本章要点：

什么是分区

Oracle 8分区：一些例子

分区索引

维护操作

并行能力

附加的考虑

分区 (partitioning) 可能是 Oracle 8 RDBMS 中最重要的一个新特性。尽管它的有些形式已经存在于 Oracle 7 中，但是它在 Oracle 8 得到增强，并更完全地并入 RDBMS 引擎。分区这个术语被“重载”。换言之，对于数据库理论和其他厂商（例如 Informix），它有很多层意思。但是，当然这里所处理的是特定的 Oracle 分区表和索引能力。

### 10.1 什么是分区

在 Oracle 7 中首次出现了用分区视图进行分区的能力。分区视图帮助按照用户定义的业务规则、条件或规范，物理地分开磁盘上的数据存储。从本质上讲，表和索引是能够根据给定的值被分离开的。例如，一个州的列包括两个字符的州名缩写，一个表能够按组成国家不同地区的州（例如 East Coast）存储在分离的磁盘上。

这是通过使用检查约束来完成的，通过互斥来执行。也就是说，对给定的列的所有不同子集的值，当被分配时，不能重叠。如果它们会重叠，你将可能在多个磁盘中存储同一行记录，但是 Oracle 如何知道使用哪一个记录呢？另外，手工建立相互排斥的分区，你应将设置 partition\_view\_enabled 初始化参数为真。这将告诉优化器去研究对所有表被分区的可能性，以使 WHERE 语句可能需要访问唯一的或少量的几个分区，而不是访问组成整个表的全部分区。对经历全表扫描的大表，这种有效性更为显著，但在其他环境中也能意识到这种有效性。例如，用大的索引扫描或中等尺寸的表或索引访问能够提高分区的有效性。优化器操作如果知道分区边界，就能够跳过分区，称为分区消除。清单 10-1 显示了如何在 Oracle 7 中建立分区视图：

清单10-1 在Oracle 7中创建分区视图

```
SQL> create table EAST_REGION (STATEID,...) ...;
SQL> create table WEST_REGION (STATEID,...) ...;
SQL> create table NORTH_REGION (STATEID,...) ...;
SQL> create table SOUTH_REGION (STATEID,...) ...;
```

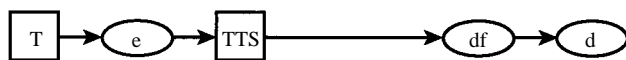
```
SQL> create view NATIONAL as
2 select * from EAST_REGION union all
3 select * from WEST_REGION union all
4 select * from NORTH_REGION union all
5 select * from SOUTH_REGION union all;
```

```
SQL> alter table EAST_REGION
2 add constraint CKSTATE check
3 (STATE_ID between 1 and 13);
...
SQL> alter table SOUTH_REGION
2 add constraint CKSTATE check
3 (STATE_ID between 39 and 50);
```

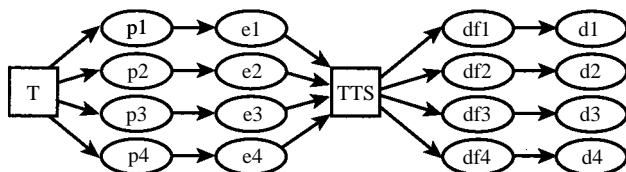
Oracle 7 分区视图的能力被加强和合并到 Oracle 8 中，现在没有像 `partition_view_enabled` 这样的初始化参数需要告诉优化器。除了在 Oracle 7 中的初始化参数以外，分区通过约束和视图来实现。实际上，整个被分区的表是一个由约束创建和存储的所有这些平行的片段的联合体。平行片段自身在实际的表中。

在 Oracle 8 中情况有所不同，因为在 Oracle 8 数据库存储结构中，分区是一个新的物理层，驻留在表和它的区间之间。表被映射为一系列分区，这些分区反过来逐一映射到一系列区间，这些区间中的每个映射到一个或多个表空间，每个表空间具有一个或多个数据文件。图 10-1 比较了分区表和非分区表。

非分区的表



分区的表



T=表，e=分区，TS=表空间，df=数据文件，d=磁盘，p=分区

图10-1 比较分区表和非分区表

更进一步，分区是 Oracle 扩展 SQL 数据定义语言 (DDL) 的一部分。分区作为语言的一部分是十分重要的，这意味着优化器是“知道分区”的，并可以在不需要像初始化参数这种东西的情况下，利用分区消除的优点。表和索引能够被分区。

### 分区策略

这里存在着几种分区策略。数据库对象（如表或索引）能够使用列值范围被分区，在 Oracle 7 或 Oracle 8 中均可以这么做，这就是通常所说的键值分区，因为列时常是表的索引或键值。Oracle 7 能够用 `BETWEEN...AND` 或 `<=` 或 `>=` 不等式限定范围，Oracle 8 要求你使用新语言 `LESS THAN`。

有一些 Oracle 目前没有实现的其他类型的分区，这些分区包括散列分区和表达式分区。正如你所见的，键值分区是表达式分区的特殊情况。为什么有人需要 Oracle 7 或 Oracle 8 不提供的表达式呢？当前使用的 Oracle 只能采用一系列的顺序范围的连续子集对表进行分区。假如纯粹是为性能的原因，你想使用某些 `MOD` 或 `RANDOM` 函数以某种方式平均分布数据，而以任何特定的列值无关。通常而言，这有可能损坏正常能获得的分区消除的好处；但在有些情况下，

它可能是有益的。例如，分区可有可能作为 Oracle数据条和RAID的替代品。图 10-2显示这个效果。看一看分割的方法是怎样强烈影响行的分布的。

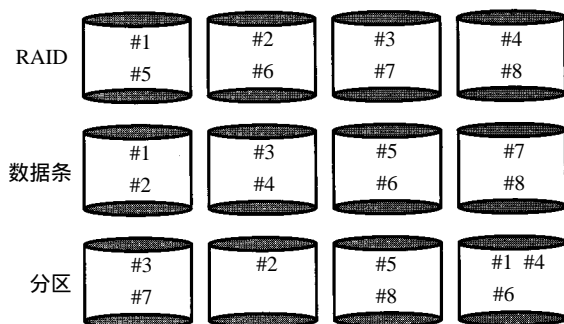


图10-2 RAID、数据条和分区的比较

花费某些代价，你能在 Oracle中使用键值分区和每个表中的一个附加列模仿表达式分区。例如，假如你想将5亿行的Oracle表随意分区，你可以在你的表中增加一列，该列含有对每一行唯一的机器产生的随机数，然后使用 LESS THAN关键字应用Oracle 8的键值分区。你能做到这一点，是因为你控制范围或随机数的产生；因为你仅需要一定范围的数字做键值分区。当然，缺点是在很大的表中有一个必须删除的附加列。

在Oracle 7或Oracle 8中，通常将并行查询与分区视图或分区联合使用。通常设置并行程度等于在版本7中组成分区视图的个数的倍数，或等于在版本8中组成表的分区数量。你不仅能够获得优化器分区消除的优势，而且你也能获得在其余那些不能被消除的分区中进行并行查询的优势。因此，通过使用并行查询，通常在存储大量数据的应用中使用分区。这些应用倾向于巨型数据库、数据中心或数据仓库。但是，联机事务进程系统也能受益，因为它们自身也可能是很大的数据库。

## 10.2 Oracle 8分区的一些例子

在Oracle 8中，一个表或索引能够达到64 000个分区，分区键是被分区的表或索引上的列。与主键一样，分区键能够组合，最多可以达到16列。在分区表中，既不支持簇，也不支持大对象（如LONG或RAW）。非分区表（通常）既能有分区索引，又能有非分区索引；同样，一个分区表既能有分区索引，又能有非分区索引。但是，除简单性外，如可管理性和性能等问题也影响你的物理设计。清单10-2是一个Oracle 8分区表的例子。

清单10-2 在Oracle 8中创建一个分区表

```
SQL> create table EMPS
2 (EMPID number(5), EMPNAME varchar2(30), EMPADDR varchar2(75))
3 partition by range (EMPID)
4 (partition P1 values less than (10001),
5 partition P2 values less than (20001),
6 partition P3 values less than (30001),
7 partition P4 values less than (40001),
8 partition P5 values less than (50001));
```

在清单10-2中，假设你有50 000个员工，员工号是1~50 000，在EMPS表创建5个分区，每个分区存储10 000个员工。

一个表的单个分区能够有不同的表空间和存储参数，如同清单 10-3所示。

清单10-3 分区表的表空间和参数

```
SQL> create table DEPTS
1 (DEPTID number(2), DEPTNAME varchar2(20),
2 DEPTBUDGET number(9)
3 partition by range (DEPTID)
4 (partition P1 values less than (51)
5 tablespace T1 storage (initial 10M next 10M),
6 partition P2 values less than (101)
7 tablespace T2 storage (initial 20M next 20M));
```

VALUES LESS THAN子句指定了那个分区的上限（不含此值）。例如，在清单 10-3中，分区P1能包含DEPTID值最高为50，但不能是51或更高。因此在VALUES LESS THAN子句中的LESS THAN的意思是严格地小于。每个分区自动在分区上创建一个约束。例如，企图插入一行DEPTID=101会引起ORA-14400错误：“插入的分区键值超过最高合法分区键值”，因为最高分区（P2）的最大允许值是100。为了避免发生这种情况，可以使用关键字 MAXVALUE，如清单10-4所示，这是一个对清单 10-3进行的简单修改。

清单10-4 分区表使用MAXVALUE

```
SQL> create table DEPTS
1 (DEPTID number(2), DEPTNAME varchar2(20),
2 DEPTBUDGET number(9)
3 partition by range (DEPTID)
4 (partition P1 values less than (51)
5 tablespace T1 storage (initial 10M next 10M),
6 partition P2 values less than (maxvalue)
7 tablespace T2 storage (initial 20M next 20M));
```

唯一不同的是在最高分区 P2中值101被关键字MAXVALUE替代。另外，NULL不能用于VALUE LESS THAN语句，当被插入的分区键是NULL值时，它们被分类为大于所有直接量值，但不大于MAXVALUE。

#### 使用分区的例子

正如前面提到的，分区键可以由多列组成（也就是复合）。进行比较的顺序是从左到右；如果MAXVALUE被使用，MAXVALUE使用之后的所有值被忽略。当在一个表上有一个组合键时，或者你想在频繁连结的表上通过主键和外键组合进行分区时，组合分区能够起作用。清单 10-5给出有一个主键的分区表的例子。清单 10-6给出了在一个表上有主键和外键组合的分区例子。

清单10-5 用主键分区

```
SQL> create table EMPS
2 (SSN number(9), SEQNO number(3),
3 EMPNAME varchar2(30), EMPADDR varchar2(75))
4 primary key (SSN, SEQNO)
5 partition by range (SSN, SEQNO)
6 (partition P1 values less than (333333334,334),
7 partition P2 values less than (666666668,668),
8 partition P3 values less than (1000000000,1000));
```

清单10-6 用主键和外键组合分区

```
SQL> create table EMPS
2 (EMPID number(5), EMPNAME varchar2(30),
3 EMPADDR varchar2(75), DEPTID number(2))
4 primary key (EMPID, DEPTID)
5 foreign key (DEPTID) references DEPTS
6 partition by range (EMPID,DEPTID)
7 (partition P1 values less than (10001,21),
8 partition P2 values less than (20001,41),
9 partition P3 values less than (30001,61),
10 partition P4 values less than (40001,81),
11 partition P5 values less than (50001,101));
```

## 10.3 分区索引

有四种主要类型的索引可以用于分区或非分区表，它们是：

- 1) 非分区（常规的）索引。
- 2) 全局前缀索引。
- 3) 本地前缀索引。
- 4) 本地非前缀索引。

为了理解这些索引类型，首先需要学习一些背景知识。回想被分区的表能够有分区索引或非分区索引；在索引分区上唯一真正的限制是一个聚簇的索引不能再被分区。如果索引的最左列正好与分区键具有相同的顺序和数量，该索引被认为是前缀的。另外，索引列可以是分区键的超集，而不是子集。

全局索引只能被前缀，但是它的分区分布通常与它的相关表不同。如果想得到全局非前缀索引，应使用一个常规的索引替代。Oracle选择不实现这种能力，因为利用它的开销将比使用常规的索引的性能差。全局索引不被Oracle维护。也就是说，在被创建之后，Oracle就不再对在索引分区和表分区之间的特定关系（值范围一致性）进行维护。全局索引最主要的缺点是Oracle优化器不能利用全局索引进行分区消除。另一方面，全局索引对分区索引值的范围提供选择，而不是对相关表的值范围提供选择。这在多索引的情况下是有用的，例如查找使用数据仓库和相关联机分析处理应用的多维数据库。同时，当存储非常珍贵时，全局索引是有吸引力的选择，因为它们可能比它们相关的表需要更少的分区。清单 10-7给出了分区表和相关的全球前缀索引的例子。

清单10-7 分区表的全局前缀索引

```
SQL> create table EMPS
2 (EMPID number(5), SSN number(9), EMPNAME varchar2(30), EMPADDR varchar2(75))
3 partition by range (EMPID)
4 (partition P1 values less than (33334),
5 partition P2 values less than (66668),
6 partition P3 values less than (100000));

SQL> create index EMPS_IDX
2 on EMPS (EMPID, SSN)
3 global
4 partition by range (EMPID)
5 partition P2 values less than (50001),
6 partition P3 values less than (100000));
```

本地索引既可以被前缀，也可以不被前缀。但是，在索引分区和它们相关表分区之间有一一对应的关系。当建立时，这种关系被认为是等分区的，因此由 Oracle 维护。Oracle 优化器能够利用分区消除的优势，这样相对于数据条与 RAID 更能够潜在地提供本质的性能增强。因此，本地索引最主要的优点是自动等分区维护和相关的潜在性能收益。用本地索引唯一不利的是，它严格要求必须与它的相关表具有相同的分区模式，不管是否前缀。对多索引需要，这可能需要太复杂的管理。同时相关的交叉存储可能会成为一个难题。

清单 10-8 给出了分区表和相关的本地前缀索引的例子，清单 10-9 给出了分区表和相关的本地非前缀索引的例子。注意，在这两种情况下都没有 PARTITION BY RANGE 或 VALUES LESS THAN 语句，因为 Oracle 知道这是本地索引，将按照完全相同的表声明对它进行同等分区（LOCAL 关键字告诉 Oracle）。将清单 10-8、清单 10-9 与清单 10-7 相比较可以看出本地和全局的不同，本地前缀与本地非前缀之间的不同。

注意 尽管本地索引分区的区别仅在于选择了不同的索引列，但实际上决定它们是否被前缀，全局索引分区的区别取决于值范围的选择的不同。

清单 10-8 分区表和相关的本地前缀索引

---

```
SQL> create table EMPS
2 (EMPID number(5), SSN number(9),
3 EMPNAME varchar2(30), EMPADDR varchar2(75))
4 partition by range (EMPID)
5 (partition P1 values less than (33334),
6 partition P2 values less than (66668),
7 partition P3 values less than (100000));

SQL> create index EMPS_IDX
2 on EMPS (EMPID, SSN)
3 local
4 (partition IP1,
5 partition IP2,
6 partition IP3);
```

---

清单 10-9 分区表和相关的本地非前缀索引

---

```
SQL> create table EMPS
2 (EMPID number(5), SSN number(9),
3 EMPNAME varchar2(30), EMPADDR varchar2(75))
4 partition by range (EMPID)
5 (partition P1 values less than (33334),
6 partition P2 values less than (66668),
7 partition P3 values less than (100000));

SQL> create index EMPS_IDX
2 on EMPS (SSN)
3 local
4 (partition IP1,
5 partition IP2,
6 partition IP3);
```

---

## 10.4 维护操作

有两个基本的语句执行所有的分区操作，它们是 ALTER TABLE 和 ALTER INDEX。对

ALTER TABLE语句的分区扩展语句如下：

```
DROP PARTITION
ADD PARTITION
RENAME PARTITION
MODIFY PARTITION
TRUNCATE PARTITION
SPLIT PARTITION
MOVE PARTITION
EXCHANGE PARTITION
```

对ALTER INDEX语句的分区扩展如下：

```
DROP PARTITION
RENAME PARTITION
REBUILD PARTITION
MODIFY PARTITION
SPLIT PARTITION
PARALLEL
UNUSABLE
```

对有经验的Oracle DBA来说，大多数这些分区操作应该是熟悉的和自我说明的，因为它们通常看上去同平常的表和索引操作（如 DROP TABLE和DROP INDEX）一样，但是你应该仔细观察那些分区独有的操作（SPLIT、MOVE、EXCHANGE和UNUSABLE），清单10-10给出一个SPLIT的例子。

清单10-10 SPLIT PARTITION 维护操作

---

```
SQL> alter table EMPS split partition P3
1 at (75000)
2 into (partition P3, partition P4);
```

---

在清单10-10中所发生的是P3分区保持所有EMPID的值小于75000（但大于在P2分区中的值）并产生一个新的P4分区存储EMPID的值等于或大于75000的行。正如你所见的SPLIT总产生一个附加的新分区。

清单10-11给出一个MOVE操作的例子。正如你所见，这只是将一个分区从一个表空间移动到另一个表空间。

清单10-11 移动分区

---

```
SQL> alter table EMPS move partition P3
1 tablespace T3;
```

---

清单10-12显示EXCHANGE语句的一个类型。EXCHANGE将一个分区表转换为一个非分区表或将一个非分区表转换为一个分区表。当你想将 Oracle 7分区视图移植到 Oracle 8分区表时很可能使用EXCHANG。

清单10-12 将分区表转换为非分区表

---

```
SQL> alter table EMPS8 exchange partition P1
```

---

```
1 with table EMPS7
2 including indexes
3 without validation;
```

清单10-12也说明，你可以有选择地包括索引列（用一定的限制）和不需要确认被移动的行范围。缺省时不包括索引并需要确认。

索引分区可以被Oracle标记为索引无效（IU），意思是你如果不重建整个索引，也必须重建至少是那些组成这个索引的IU分区。

标志一个索引为IU的6种维护操作如下：

- 1) SQL\*Loader或导入具有忽略索引的选项的分区。
- 2) SQL\*Loader或导入具有忽略索引的选项的分区。
- 3) ALTER TABLE TRUNCATE PARTITION
- 4) ALTER TABLE SPLIT PARTITION
- 5) ALTER TABLE MOVE PARTITION
- 6) ALTER INDEX SPLIT PARTITION

## 10.5 并行能力

有三种主要并行能力能够用于分区以增强性能：

- 1) 并行查询（PQ），也称为并行查询选项（PQO）。
- 2) 并行数据定义语言（PDDL）。
- 3) 并行数据操作语言（PDML）。

并行查询操作在分区表和非分区表中仅有微小的不同。在非分区表，仅当表中的行数大于或等于在编译时被检查的内部门限时，表的全表扫描才被并行化。相反，在分区表中，仅当相关分区总的大小大于或等于那些相同的内部门限时（该门限不仅在编译时被检查，而且如有需要，在运行时也进行检查，这是因为查询自身的分区限制），才将该表的全表扫描并行化。如果在检索与查询需要相应的分区值范围（边界）时，优化器在分区消除时，分区不被跳过，就称为是分区相关的。使用并行查询，并行度由ROWID范围确定，与表的分区无关。

并行数据定义语言被限制为DDL命令中可以并行化的一个很小的子集：

```
CREATE INDEX
CREATE TABLE AS SELECT
ALTER TABLE MOVE PARTITION
ALTER TABLE SPLIT PARTITION
ALTER INDEX REBUILD PARTITION
```

除了附加的特定分区的操作，这种类型的并行化在Oracle 8中与在Oracle 7中相同。

并行数据操作语言（DML）可能有点模糊，但基本上，在Oracle中通常指的是INSERT、UPDATE和DELETE操作，不是SELECT操作。尽管在Oracle之外的数据库的书本和著作中，首字母缩写DML代表数据操作语言，但是Oracle更愿意让你认为DML是数据修改语言。如果你认为DML是数据修改语言，而SELECT是查询，在Oracle术语中是指并行查询处理SELECT的并行化，而PDML是处理INSERT、UPDATE和DELETE的并行化。

正如你可能想象的，PDML在性能方面的影响通常很少与PQ一样大，特别是在OLTP和DSS系统中。但是，有时有一些孤立的处理或在这些应用类型中高度易变的表，这些表的增

长和收缩十分迅速。更进一步，表的这些类型在某些应用中有可能变得十分巨大。当后面这些条件似乎会在你的组织中出现时，PDML应该与PQ一样对你有益，甚至更好。

和使用并行查询一样，PDML的优点依赖于被访问表的行数、这些表所跨的磁盘驱动器数、服务于这些磁盘的控制器数目和类型、CPU的数量和其他软硬件因素。例如 Oracle 使用磁盘到数据文件再到分区的映射来帮助将 PDML 语句的工作分离和分配到它们的伺服器，同时，与并行查询一样，PDML 提供一个失败安全水平。如同所有真正的面向事务的 RDBMS 一样，如果一个 PDML 伺服器的工作失败，所有伺服器都回滚，这样全部 PDML 语句失败，保证了事务的原子性。这种操作很像一个本地两阶段提交操作。用 PDML 时，UPDATE 和 DELETE 操作要求表被分区，但 INSERT...SELECT 不要求这么做，就和用 SELECT 和 PQ 时一样。

注意 没有初始化参数能启用 PDML。如果要启用 PDML，必须使用 ALTER SESSION ENABLE PARALLEL DML。但是优化器可能由于其他限制，选择不并行 DML。在启用的 PDML 会话中的每一条语句必须跟随一条 COMMIT 或 ROLLBACK，以阻止 ORA-12830 错误：“在执行并行 INSERT/UPDATE/DELETE 返回后，必须提交或回滚”。另外，如果在一个事务（PL/SQL 代码块）中的第一条 DML 语句没有以并行方式执行，所有随后的语句被顺序执行，而不管提示或缺省表的并行度。

## 10.6 附加的考虑

PDML 是在 Oracle 8 中与分区有关的一项主要功能。除此之外，许多其他特性和功能与这些新的分区能力有关。分区扩展表名就是一个伴随分区的增强，这种语言的扩展是非标准的（也就是非 ANSI），因此你不得不使用视图来封装代码，来促进标准化。清单 10-13 给出了使用分区扩展表名的例子。

清单 10-13 一个分区扩展表名

---

```
SQL> select * from EMPS partition (p1);
```

---

分区锁增加了另一层对 Oracle 7 锁规划的粒度，以支持 Oracle 8 的分区独立的概念。分区独立的简单含意是你能够完全像对待 Oracle 7 中的表一样对待 Oracle 8 中的分区。使用这种方法，一个 DBA 能够维护一个分区而不影响用户访问其他分区。本质上，分区级锁作为在表级和行级锁之间的锁级别存在。总之，分区独立的优点是独立管理恢复和备份这些事，这也能被看作高可用性特性，因为一个分区失败时其他分区仍可被访问。

警告 我建议在使用分区时不要更新分区键的列，那将引起行从一个分区移动到另一个分区。在 Oracle 8 分区中，没有自动的行移植。如果你试图做这样的更新，你将接收到错误“ORA-14402：更新分区键列，将引起分区改变”。用 Oracle 8 时，DBA 不得不手动移植行。

现在，假设你有一个员工表（EMPS），它是使用员工标识符（EMPLID）进行分区的，并有两个分区 P1 和 P2，P1 中的值小于 51，P2 中有 51 以及以上的值。清单 10-14 给出两个更新语句例子，第一条语句成功，但是第二条语句失败，并伴随一个 ORA-14402 错误，如前面的警告中所提到的。解决这个问题的一种方法是做一个插入和删除，但是对成批操作这是无效

的。在这种情况下，你最好用前面讨论的分区操作，或卸载和重新装载你的分区。

清单10-14 分区更新语句

---

```
SQL> update EMPS set EMPID = 67
2 where EMPID = 66;
/* This works ok, since this particular employee stays in partition P2 */

SQL> update EMPS set EMPID = 67
2 where EMPID = 33;
/* This will not work, since it would cause this particular employee to move
from partition P1 to P2. Hence, it will generate an ORA-14402 error. */

SQL> insert into EMPS partition (P2)
2 values (33, ...);
SQL> delete from EMPS
2 where EMPID = 67;
```

---

最后一个考虑是新的 Oracle 8 ROWID 格式是与 Oracle 7 相对的。在 Oracle 8 中，ROWID 被扩展，主要用于调节分区，但是也被应用于调节对象选项，这将在第 11 章“大对象的历史”中讨论。这个新的 ROWID 格式在第 13 章“Net 8 网络：新特性和概念”中深入进行讨论。