# [Oracle-HWM(High Water Mark) 高水位解读](http://blog.csdn.net/yangshangwei/article/details/53457321)

# 读前须知：Oracle的逻辑存储管理

ORACLE在逻辑存储上分4个粒度 ，由大到小为: 表空间, 段, 区 和 块.

## 块Block

块:是粒度最小的存储单位,现在标准的块大小是8K,ORACLE每一次I/O操作也是按块来操作的,也就是说当ORACLE从数据文件读数据时,是读取多少个块,而不是多少行. 每一个Block里可以包含多个row.

数据块的大小是通过kb字节个数来指定的，默认为8KB。相关参数为db\_block\_size

SQL> show parameter db\_block\_size

NAME TYPE VALUE

----------------------- --------------------- ---------

db\_block\_size integer 8192

## 区Extent

由一系列相邻的块而组成,这也是ORACLE空间分配的基本单位.

区extent是比数据块大一级的存储结构，表示的是一连串连续的数据块集合。

在进行存储数据信息的时候，Oracle将分配数据块进行存储，但是不能保证所有分配的数据块都是连续的结构。

所以，出现分区extent的概念，表示一系列连续的数据块集合。

举个例子来说,当我们创建一个表时,首先ORACLE会分配一区的空间给这个表,随着数据不断地增长,原来的这个区容不下插入的数据时,ORACLE是以区为单位进行扩展的,也就是说再分配多少个区给这个表,而不是多少个块.

视图dba\_extents（或者all\_extents、user\_extents）是我们研究分区结构和存储构成的重要手段。

## 段Segment

段: 是由一系列的区extent所组成

数据段是与数据库对象相对应，一般一个数据库对象对应一个数据段。

多个extent是对应一个数据段，每个数据段实际上就是数据库一个对象的代表。

一般来说, 当创建一个对象时(表,索引),就会分配一个段给这个对象.

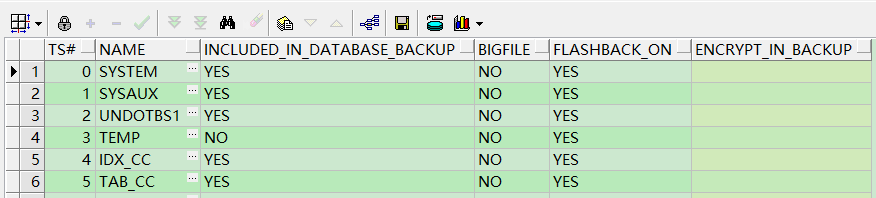
从dba\_segments、user\_segments视图中，可以比较清楚看清数据段的结构。

## 表空间Tablespace

TableSpace是存储结构中的最高层结构。建立一个表空间的时候，是需要指定存储的文件。一个表空间可以指定多个数据文件，多个文件可以在不同的物理存储上。也就是说，表空间是可以跨物理存储的。

但是有一点就是，表空间下一级对象数据段的存储，是不能指定存储在那个文件里的。所以，要想让数据对象访问IO负载均衡，需要指定不同的数据对象在不同的表空间里。这也就是为什么将数据表和索引建立在不同的表空间的原因。

表空间通过v$tablespace进行访问



其中两个参数需要注意一下。

一个是bigfile，是一个标志位，标志表空间是不是所谓的大文件表空间。

大文件表空间是在10g中推出的一个新特性，处于性能考虑，可以设置表空间为大文件表空间，存储超过百T的数据，但是要求数据文件只能有一个。

另一个是flashback\_on，表示表空间的闪回特性是否开启。

还有 dba\_tablespaces 、 user\_tablespaces。

# Oracle表段中的高水位线HWM

在数据库表刚建立的时候，由于没有任何数据，所以这个时候水位线是空的，也就是说HWM为最低值。

当插入了数据以后，高水位线就会上涨，但如果你采用delete语句删除数据的话，数据虽然被删除了，但是高水位线却没有降低，还是你刚才删除数据以前那么高的水位。

也就是说，这条高水位线在日常的增删操作中只会上涨，不会下跌。

HWM通常增长的幅度为一次5个数据块.

Select语句会对表中的数据进行一次扫描，但是究竟扫描多少数据存储块呢，这个并不是说数据库中有多少数据，Oracle就扫描这么大的数据块，而是Oracle会扫描高水位线以下的数据块。

试想一下，新建的一个空表，Select一下，由于高水位线HWM在最低的0位置上，所以没有数据块需要被扫描，扫描时间会极短。

如果这个时候你首先插入了一千万条数据，然后再用delete语句删除这一千万条数据。由于插入了一千万条数据，所以这个时候的高水位线就在一千万条数据这里。后来删除这一千万条数据的时候，由于delete语句不影响高水位线，所以高水位线依然在一千万条数据这里。

这个时候再一次用select语句进行扫描，虽然这个时候表中没有数据，但是由于扫描是按照高水位线来的，所以需要把一千万条数据的存储空间都要扫描一次，也就是说这次扫描所需要的时间和扫描一千万条数据所需要的时间是一样多的。所以有时候有人总是经常说，怎么我的表中没有几条数据，但是还是这么慢呢，这个时候其实奥秘就是这里的高水位线了。

那有没有办法让高水位线下降呢 ? 采用TRUNCATE语句删除一个表的数据的时候，类似于重新建立了表，不仅把数据都删除了，还把HWM给清空恢复为0。

所以如果需要把表清空，在有可能利用TRUNCATE语句来删除数据的时候就利用TRUNCATE语句来删除表，特别是那种数据量有可能很大的临时存储表。

在手动段空间管理（Manual Segment Space Management）中，段中只有一个HWM，

但是在Oracle 9i Release1才添加的自动段空间管理（Automatic Segment Space Management）中，又有了一个低HWM的概念出来。

为什么有了HWM还又有一个低HWM呢，这个是因为自动段空间管理的特性造成的。在手段段空间管理中，当数据插入以后，如果是插入到新的数据块中，数据块就会被自动格式化等待数据访问。

而在自动段空间管理中，数据插入到新的数据块以后，数据块并没有被格式化，而是在第一次访问这个数据块的时候才格式化这个块。

所以我们又需要一条水位线，用来标示已经被格式化的块。这条水位线就叫做低HWM。一般来说，低HWM肯定是低于等于HWM的。

# 降低ORACLE表的高水位线

在ORACLE中，执行对表的删除操作不会降低该表的高水位线。而全表扫描将始终读取一个段(extent)中所有低于高水位线标记的块。如果在执行删除操作后不降低高水位线标记，则将导致查询语句的性能低下。

rebuild, truncate, shrink,move 等操作会降低高水位。

## 执行表重建指令 alter table table\_name move

在线转移表空间ALTER TABLE … MOVE TABLESPACE ..

当你创建了一个对象如表以后,不管你有没有插入数据,它都会占用一些块,ORACLE也会给它分配必要的空间.

同样,用ALTER TABLE MOVE释放自由空间后,还是保留了一些空间给这个表.

ALTER TABLE … MOVE 后面不跟参数也行，不跟参数表还是在原来的表空间，Move后记住重建索引.

如果以后还要继续向这个表增加数据，没有必要move， 只是释放出来的空间，只能这个表用，其他的表或者segment无法使用该空间。

## 执行alter table table\_name shrink space-10g新功能

此命令为Oracle 10g新增功能，再执行该指令之前必须允许行移动 alter table table\_name enable row movement;

如果要同时压缩表的索引,可以发布:alter table test\_tab shrink space cascade

## 重建表

复制要保留的数据到临时表t，drop原表，然后rename临时表t为原表CREATE TABLE AS SELECT \* FROM

## 用逻辑导入导出: Emp/Imp

## Alter table table\_name deallocate unused

DEALLOCATE UNUSED会释放HWM上面的未使用空间,但是并不会释放HWM下面的自由空间,也不会移动HWM的位置.

## truncate(推荐使用)

truncate table xxx

# HWM的特征

## ORACLE用HWM来界定一个段中使用的块和未使用的块

当我们创建一个表时,ORACLE就会为这个对象分配一个段.在这个段中,即使我们未插入任何记录,也至少有一个区被分配,第一个区的第一个块就称为段头(SEGMENT HEADE),段头中就储存了一些信息,HWM的信息就存储在此.

我们不断插入数据时,HWM会往不断上移,这样,在HWM之下的,就表示使用过的块,HWM之上的就表示已分配但从未使用过的块.

## HWM在插入数据时,当现有空间不足而进行空间的扩展时会向上移,但删除数据时不会往下移.

ORACLE 不会释放空间以供其他对象使用，有一条简单的理由：由于空间是为新插入的行保留的，并且要适应现有行的增长。被占用的最高空间称为最高使用标记 (HWM).

## HWM的信息存储在段头当中.

HWM本身的信息是储存在段头.

在段空间是手工管理方式时,ORACLE是通过FREELIST(一个单向链表)来管理段内的空间分配.

在段空间是自动管理方式时(ASSM),ORACLE是通过BITMAP来管理段内的空间分配.

## ORACLE的全表扫描是读取高水位标记(HWM)以下的所有块.

所以问题就产生了.当用户发出一个全表扫描时，ORACLE 始终必须从段一直扫描到 HWM，即使它什么也没有发现。

该任务延长了全表扫描的时间。

## 当用直接路径插入行时，即使HWM以下有空闲的数据库块，键入在插入数据时使用了append关键字，则在插入时使用HWM以上的数据块，此时HWM会自动增大。

例如，通过直接加载插入（用 APPEND 提示插入）或通过 SQL\*LOADER 直接路径 数据块直接置于 HWM 之上。它下面的空间就浪费掉了。

# 例子

数据库版本 Connected to Oracle Database 11g Enterprise Edition Release **11.2.0.4.0**

## 创建测试表

SQL> create table tt(id number);

Table created

此时表没有分析，是原始的数据，即8个数据块。

--空的

SQL> SELECT segment\_name,segment\_type,blocks FROM dba\_segments a WHERE a.segment\_name = 'TT';

SEGMENT\_NAME SEGMENT\_TYPE BLOCKS

-------------------------------------------------------------------------------- ------------------ ----------

--空的

SQL> SELECT table\_name,num\_rows,blocks,empty\_blocks FROM user\_tables WHERE table\_name='TT';

TABLE\_NAME NUM\_ROWS BLOCKS EMPTY\_BLOCKS

------------------------------ ---------- ---------- ------------

TT

## 向表中插入一些测试数据

SQL> declare

2 i number;

3 begin

4 for i in 1 .. 10000 loop

5 insert into tt values (i);

6 end loop;

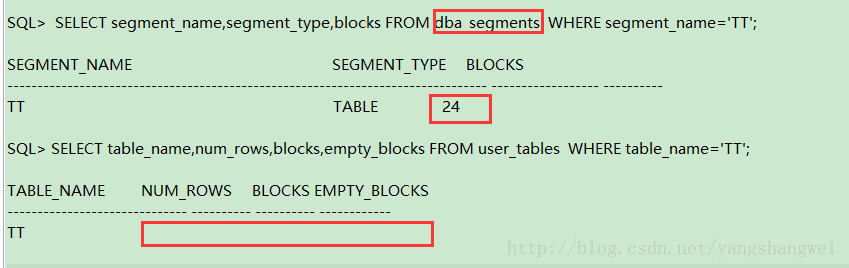
7 commit;

8 end;

9 /

PL/SQL procedure successfully completed

## 重新查询表的信息



此时表TT 占用的块已经是24个了.

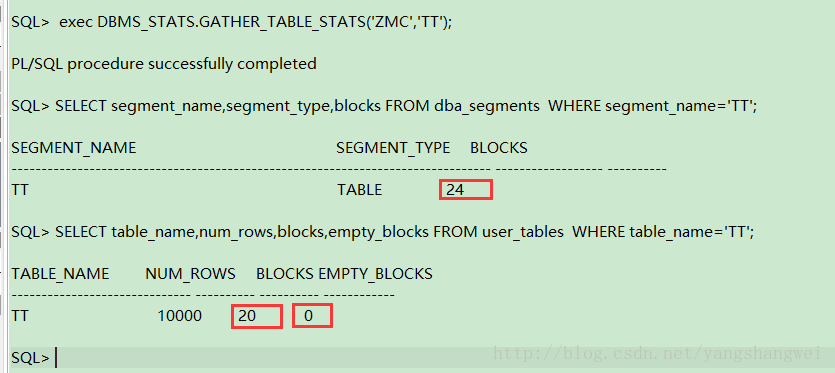
但是user\_tables 显示的信息还是为空。 因为没有做统计分析。

## 收集统计信息

SQL> exec DBMS\_STATS.GATHER\_TABLE\_STATS(user,'TT');

PL/SQL procedure successfully completed

再此查询一下

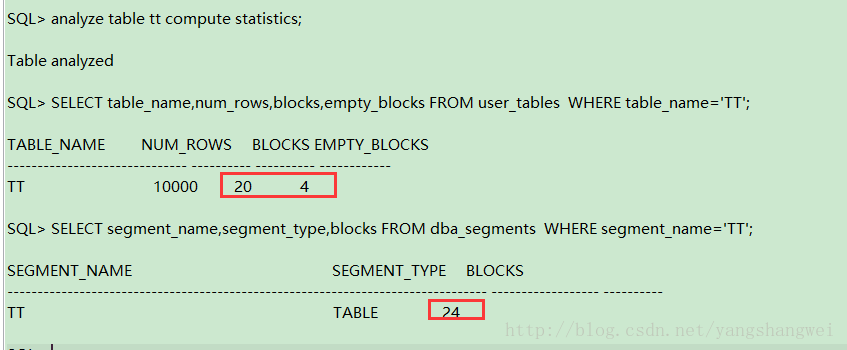


此时user\_tables 已经有了数据，显示的使用了20个数据块。 但是empty\_blocks 还是为空。 这里要注意的地方。empty\_blocks 这个字段只有使用analyze 收集统计信息之后才会有数据。

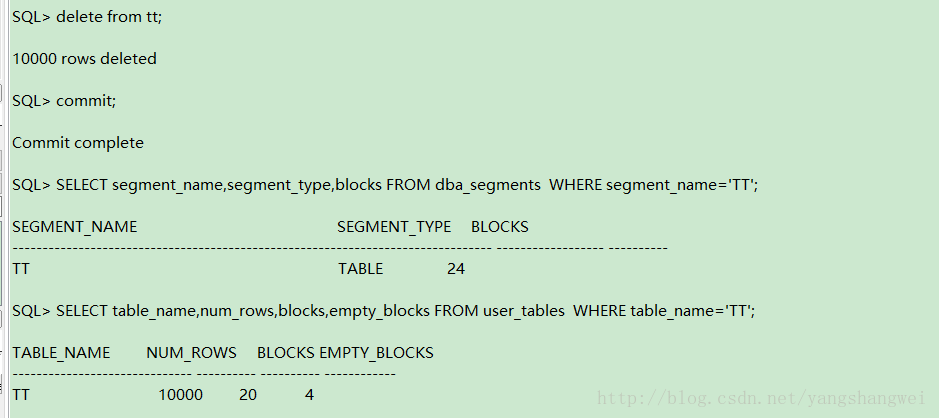
## 使用analyze搜集统计信息

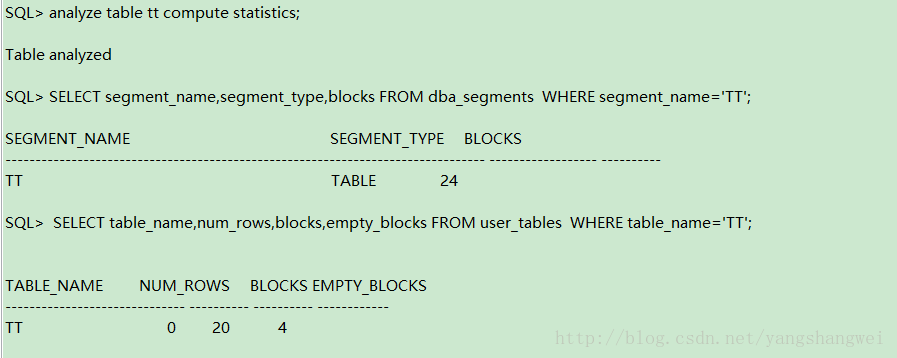
SQL> analyze table tt compute statistics;

Table analyzed



## delete 数据，不会降低高水位





可以发现 分析前后，blocks 和 empty\_blocks 都没有发生变化。

## truncate 表，可以降低高水位

--truncate 表

SQL> truncate table tt;

Table truncated

--查询段信息，blocks由24降到了8

SQL> SELECT segment\_name,segment\_type,blocks FROM dba\_segments WHERE segment\_name='TT';

SEGMENT\_NAME SEGMENT\_TYPE BLOCKS

--------- ---------- ----------

TT TABLE 8

--查询表信息，没有改变

SQL> SELECT table\_name,num\_rows,blocks,empty\_blocks FROM user\_tables WHERE table\_name='TT';

TABLE\_NAME NUM\_ROWS BLOCKS EMPTY\_BLOCKS

-------- ---------- ------ ------

TT 0 20 4

------------------------------------------------------------

------------------------------------------------------------

--收集下表信息

SQL> exec dbms\_stats.gather\_table\_stats(user,'TT');

PL/SQL procedure successfully completed

--重新统计下段信息，一样

SQL> SELECT segment\_name,segment\_type,blocks FROM dba\_segments WHERE segment\_name='TT';

SEGMENT\_NAME SEGMENT\_TYPE BLOCKS

--------- ---------- ----------

TT TABLE 8

--重新查询表信息 BLOCKS 由20降为0， 但是empyt\_blocks 还是4个

SQL> SELECT table\_name,num\_rows,blocks,empty\_blocks FROM user\_tables WHERE table\_name='TT';

TABLE\_NAME NUM\_ROWS BLOCKS EMPTY\_BLOCKS

---------- ----- ---------- ------------

TT 0 0 4

--analyze 分析下 更改EMPTY\_BLOCKS的值

SQL> analyze table tt compute statistics;

Table analyzed

--重新查询段信息，

SQL> SELECT segment\_name,segment\_type,blocks FROM dba\_segments WHERE segment\_name='TT';

SEGMENT\_NAME SEGMENT\_TYPE BLOCKS

--------- ---------- ----------

TT TABLE 8

--重新查询表信息

SQL> SELECT table\_name,num\_rows,blocks,empty\_blocks FROM user\_tables WHERE table\_name='TT';

TABLE\_NAME NUM\_ROWS BLOCKS EMPTY\_BLOCKS

---------- ----- ---------- ------------

TT 0 0 8

SQL>

总共8个数据块，8个为空

# Alter table move 和Shrink的区别

## Shrink

在10g之后，整理碎片消除行迁移的新增功能shrink space

语法

alter table <table\_name> shrink space [ <null> | compact | cascade ];

* compact: 这个参数当系统的负载比较大时可以用，不降低HWM。如果系统负载较低时，直接用alter table table\_name shrink space就一步到位了
* cascade：这个参数是在shrink table的时候自动级联索引，相当于rebulid index。

基于普通表

--shrink必须开启行迁移功能。

alter table table\_name enable row movement ;

--保持HWM，相当于把块中数据打结实了

alter table table\_name shrink space compact;

--回缩表与降低HWM

alter table table\_name shrink space;

--回缩表与相关索引，降低HWM

alter table table\_name shrink space cascade;

--回缩索引与降低HWM

alter index index\_name shrink space

虽然在10g中可以用shrink ，但也有些限制：

* 1). 对cluster，cluster table，或具有Long，lob类型列的对象 不起作用。
* 2). 不支持具有function-based indexes 或 bitmap join indexes的表
* 3). 不支持mapping 表或index-organized表。
* 4). 不支持compressed 表

## Shrink例子

SQL> create table tt(id number);

SQL> declare

i number;

begin

for i in 1 .. 10000 loop

insert into tt values (i);

end loop;

commit;

end;

SQL> exec DBMS\_STATS.GATHER\_TABLE\_STATS(user,'TT');

SQL> analyze table tt compute statistics;

SQL> delete from tt;

SQL> commit ;

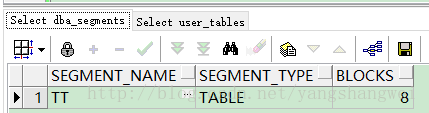
SQL> alter table tt enable row movement ;

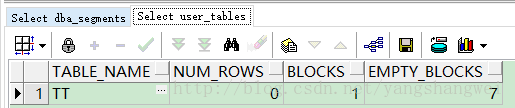
SQL> alter table tt shrink space;

SQL> analyze table tt compute statistics;

SQL> SELECT segment\_name,segment\_type,blocks FROM dba\_segments WHERE segment\_name='TT';

SQL> SELECT table\_name,num\_rows,blocks,empty\_blocks FROM user\_tables WHERE table\_name='TT';





## Move

通过desc table\_name 来检查表中是否有LOB 字段，

### 表中没有lob字段

如果表没有LOB字段， 直接 alter table move; 然后 rebuild index

### 表中包含了LOB字段

alter table owner.table\_name move tablespace tablespace\_name lob (lob\_column) store as lobsegment tablespace tablespace\_name;

也可以单独move lob，但是表上的index 同样会失效. 所以在操作结束，需要对索引进行rebuild。

alter table owner.table\_name move lob(lob\_column) store as lobsegment tablespace tablespace\_name ;

索引的rebuild：

首先用下面的SQL查看表上面有哪类索引:

SELECT a.owner,

a.index\_name,

a.index\_type,

a.partitioned,

a.status,

b.status p\_status,

b.composite

FROM dba\_indexes a

LEFT JOIN

dba\_ind\_partitions b

ON a.owner = b.index\_owner AND a.index\_name = b.index\_name

WHERE a.owner = '&owner' AND a.table\_name = '&table\_name';

对于普通索引直接rebuild online nologging parallel,   
对于分区索引，必须单独rebuild 每个分区,   
对于组合分区索引,必须单独rebuild 每个子分区。

## 总结

Move 通过移动数据来来降低HWM，因此需要更多的磁盘空间。   
Shrink 通过delete 和 insert， 会产生较多的undo 和redo。

shrink space收缩到数据存储的最小值，alter table move（不带参数）收缩到initial指定值，也可以用alter table test move storage(initial 500k)指定收缩的大小，这样可以达到shrink space效果。

总之，使用Move 效率会高点，但是会导致索引失效。Shrink 会产生undo 和redo，速度相对也慢一点。