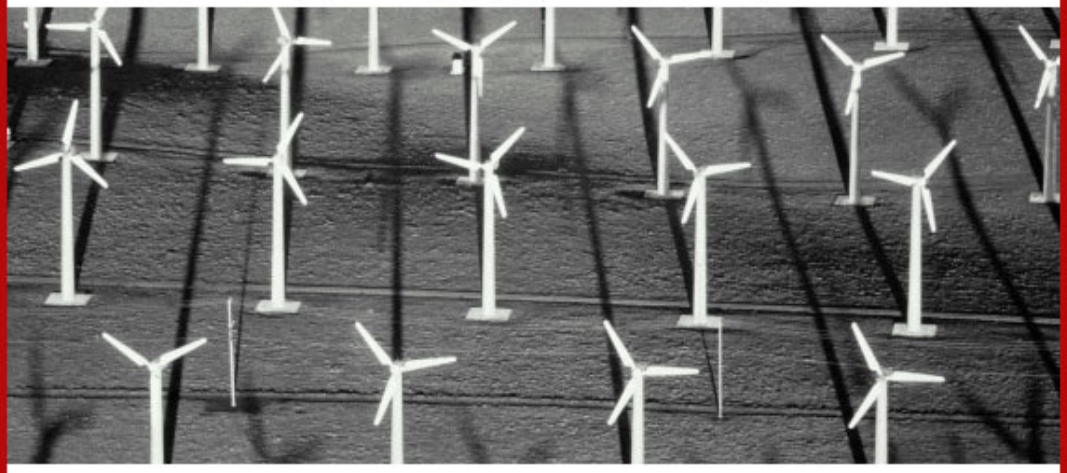
Oracle

2004/12 第2期 总第2期

eMag Magazines from Csdn



Oracle job系列

简单的 oracle job，也有如此多的奥妙

一条sql导致数据库整体性能下降的诊断和解决的全过程

oracle数据库的性能问题，有时就是一个细节导致的
哪怕是一个动作或者一条sql

关于字符集更改的内部操作

经常有胆大的人这样干，但是你是胸有成竹呢还是无知者无畏？

如何处理Oracle数据库中的坏块问题

数据库块的损坏，是一个让DBA们头疼的问题
如果不幸您真的遇到了，该怎么来处理呢

emag.csdn.net

Csdn.net

CSDN Oracle 电子杂志

2004 年 12 月 第 2 期 总第 2 期

本期内容导读

关于动态sql的使用

开发中有时我们的 sql 不能确定，期望通过一些方式动态生成 sql 执行。

关于竖表转横表的问题

有时我们在做报表的时候，需要进行行列转置，这是一个很多人关心的话题。

利用游标返回数据集两则

oracle 到底如何返回一组数据给前台，很多人为此疑惑。

Oracle job 基础

Oracle 是如何定期的执行任务，其中又包含了很多技巧。

Oracle 计划作业(JOB)的探讨

Oracle 的 job 在多次执行过程中，如何确定下次执行时间是一个有趣的话题。

Oracle 诊断案例-Job 任务停止执行

大千世界，总是千奇百怪，oracle job 也一样，匪夷所思。

一条 sql 导致数据库整体性能下降的诊断和解决的全过程

oracle 数据库的性能问题，有时就是一个细节导致的，哪怕是一个动作或者一条 sql。

如何使用触发器实现数据库级守护

DBA 们注意了，你该怎么来预防可能对数据库造成的破坏。

DBMS_ROWID 包的使用

这是一个经常被专业人士用来处理细节问题而使用的包。

关于字符集更改的内部操作

经常有胆大的人这样干，但是你是胸有成竹呢还是无知者无畏？

如何处理 Oracle 数据库中的坏块问题

数据库块的损坏，是一个让 DBA 们头疼的问题，如果不幸您真地遇到了，该怎么来处理呢？

篇首寄语

这是本篇杂志的第二期，创刊的热情过去后，还需要我们一如既往的投入，当然也需要读者朋友们的继续支持。Oracle 的杂志在国内并不多，比起国外的 dbazine 等等受人瞩目的杂志而言，我们才刚刚迈开脚步。希望接下来的日子，CSDN Oracle 电子杂志能走好。

由于我们的杂志还很不成熟，各方面都在变动中，要想成长为一份专业的电子杂志，还需要各位的支持，任何建议及投稿你都可以发送到:

emag_oracle@csdn.net

你的需求，是我们前进的动力。

最后让我们感谢为此杂志付出努力的朋友们,也就是 Oracle 杂志所有的作者和编辑们，他们是:

chanet,coolyl,dinya,eygle ,kamus, biti_rainy， simore

关于编辑的具体情况你可以在杂志首页找到详细的介绍。

http://emag.csdn.net/Default.aspx?tabid=49

Biti_rainy 2004-12-08

目录

关于动态 SQL 的使用

本文作者：dinya

内容摘要：

在 PL/SQL 开发过程中，使用 SQL，PL/SQL 可以实现大部份的需求，但是在某些特殊的情况下，在 PL/SQL 中使用标准的 SQL 语句或 DML 语句不能实现自己的需求，比如需要动态建表或某个不确定的操作需要动态执行。这就需要使用动态 SQL 来实现。本文通过几个实例来详细的讲解动态 SQL 的使用。

本文适宜读者范围：

Oracle 初级，中级

系统环境：

OS：windows 2000 Professional (英文版)

Oracle：8.1.7.1.0

正文：

一般的 PL/SQL 程序设计中，在 DML 和事务控制的语句中可以直接使用 SQL，但是 DDL 语句及系统控制语句却不能在 PL/SQL 中直接使用，要想实现在 PL/SQL 中使用 DDL 语句及系统控制语句，可以通过使用动态 SQL 来实现。

首先我们应该了解什么是动态 SQL，在 Oracle 数据库开发 PL/SQL 块中我们使用的 SQL 分为：静态 SQL 语句和动态 SQL 语句。所谓静态 SQL 指在 PL/SQL 块中使用的 SQL 语句在编译时是明确的，执行的是确定对象。而动态 SQL 是指在 PL/SQL 块编译时 SQL 语句是不确定的，如根据用户输入的参数的不同而执行不同的操作。编译程序对动态语句部分不进行处理，只是在程序运行时动态地创建语句、对语句进行语法分析并执行该语句。

Oracle 中动态 SQL 可以通过本地动态 SQL 来执行，也可以通过 DBMS_SQL 包来执行。下面就这两种情况分别进行说明：

一．本地动态 SQL

本地动态 SQL 是使用 EXECUTE IMMEDIATE 语句来实现的。

1．本地动态 SQL 执行 DDL 语句：

需求：根据用户输入的表名及字段名等参数动态建表。

```
create or replace procedure proc_test
(
   table_name in varchar2,      --表名
   field1 in varchar2,           -字段名
   datatype1 in varchar2,       -字段类型
   field2 in varchar2,           --字段名
   datatype2 in varchar2        --字段类型
) as
   str_sql varchar2(500);
begin
   str_sql:='create           table           '||table_name||'('||field1||'
'||datatype1||','||field2||' '||datatype2||')';
   execute immediate str_sql;   --动态执行 DDL 语句
   exception
      when others then
         null;
end ;
```

以上是编译通过的存储过程代码。下面执行存储过程动态建表。

```
SQL>        execute        proc_test('dinya_test','id','number(8)        not
null','name','varchar2(100)');

PL/SQL procedure successfully completed

SQL> desc dinya_test;
Name Type         Nullable Default Comments
---- ------------- -------- ------- --------
ID   NUMBER(8)

NAME VARCHAR2(100) Y

SQL>
```

到这里，就实现了我们的需求，使用本地动态 SQL 根据用户输入的表名及字段名、字段类型等参数来实现动态执行 DDL 语句。

2．本地动态 SQL 执行 DML 语句。

需求：将用户输入的值插入到上例中建好的 dinya_test 表中。

```
create or replace procedure proc_insert
(
   id in number,                              --输入序号
```

```
   name in varchar2                          --输入姓名
) as
   str_sql varchar2(500);
begin
   str_sql:='insert into dinya_test values(:1,:2)';
   execute immediate str_sql using id,name; --动态执行插入操作
   exception
      when others then
         null;
end ;
```

执行存储过程，插入数据到测试表中。

```
SQL> execute proc_insert(1,'dinya');
PL/SQL procedure successfully completed
SQL> select * from dinya_test;
      ID     NAME
      1     dinya
```

在上例中，本地动态 SQL 执行 DML 语句时使用了 using 子句，按顺序将输入的值绑定到变量，如果需要输出参数，可以在执行动态 SQL 的时候，使用 RETURNING INTO 子句，如：

```
declare
   p_id number:=1;
   v_count number;
begin
   v_string:='select count(*) from table_name a where a.id=:id';
   execute immediate v_string into v_count using p_id;
end ;
```

更多的关于动态 SQL 中关于返回值及为输出输入绑定变量执行参数模式的问题，请读者自行做测试。

二．使用 DBMS_SQL 包

使用 DBMS_SQL 包实现动态 SQL 的步骤如下：A、先将要执行的 SQL 语句或一个语句块放到一个字符串变量中。B、使用 DBMS_SQL 包的 parse 过程来分析该字符串。C、使用 DBMS_SQL 包的 bind_variable 过程来绑定变量。D、使用 DBMS_SQL 包的 execute 函数来执行语句。

1．使用 DBMS_SQL 包执行 DDL 语句。

需求：使用 DBMS_SQL 包根据用户输入的表名、字段名及字段类型建表。

```
create or replace procedure proc_dbms_sql
(
   table_name in varchar2,      --表名
   field_name1 in varchar2,     --字段名
   datatype1 in varchar2,       --字段类型
   field_name2 in varchar2,     --字段名
   datatype2 in varchar2        --字段类型
)as
   v_cursor number;            --定义光标
   v_string varchar2(200);     --定义字符串变量
   v_row number;               --行数
begin
   v_cursor:=dbms_sql.open_cursor;     --为处理打开光标
   v_string:='create         table         '||table_name||'('||field_name1||'
'||datatype1||','||field_name2||' '||datatype2||')';
   dbms_sql.parse(v_cursor,v_string,dbms_sql.native);    --分析语句
   v_row:=dbms_sql.execute(v_cursor);   --执行语句
   dbms_sql.close_cursor(v_cursor);     --关闭光标
   exception
      when others then
         dbms_sql.close_cursor(v_cursor);  --关闭光标
         raise;
end;
```

以上过程编译通过后，执行过程创建表结构：

```
SQL>       execute       proc_dbms_sql('dinya_test2','id','number(8)       not
null','name','varchar2(100)');

PL/SQL procedure successfully completed

SQL> desc dinya_test2;
Name Type         Nullable Default Comments
---- ------------ -------- ------- --------
ID   NUMBER(8)
NAME VARCHAR2(100) Y

SQL>
```

2、使用 DBMS_SQL 包执行 DML 语句。

需求：使用 DBMS_SQL 包根据用户输入的值更新表中相对应的记录。

查看表中已有记录：

```
SQL> select * from dinya_test2;
      ID NAME
       1 Oracle
       2 CSDN
       3 ERP
SQL>
```

建存储过程，并编译通过：

```
create or replace procedure proc_dbms_sql_update
(
   id number,
   name varchar2
)as
   v_cursor number;          --定义光标
   v_string varchar2(200);   --字符串变量
   v_row number;             --行数
begin
   v_cursor:=dbms_sql.open_cursor;    --为处理打开光标
   v_string:='update dinya_test2 a set a.name=:p_name where a.id=:p_id';
   dbms_sql.parse(v_cursor,v_string,dbms_sql.native);   --分析语句
   dbms_sql.bind_variable(v_cursor,':p_name',name);    --绑定变量
   dbms_sql.bind_variable(v_cursor,':p_id',id);          --绑定变量
   v_row:=dbms_sql.execute(v_cursor);                --执行动态ＳＱＬ
   dbms_sql.close_cursor(v_cursor);                    --关闭光标
   exception
      when others then
         dbms_sql.close_cursor(v_cursor);             --关闭光标
         raise;
end;
```

执行过程，根据用户输入的参数更新表中的数据：

```
SQL> execute proc_dbms_sql_update(2,'csdn_dinya');
```

```
PL/SQL procedure successfully completed

SQL> select * from dinya_test2;
      ID NAME
       1 Oracle
       2 csdn_dinya
       3 ERP
SQL>
```

执行过程后将第二条的 name 字段的数据更新为新值 csdn_dinya。这样就完成了使用 dbms_sql 包来执行 DML 语句的功能。

使用 DBMS_SQL 中，如果要执行的动态语句不是查询语句，使用 DBMS_SQL.Execute 或 DBMS_SQL.Variable_Value 来执行，如果要执行动态语句是查询语句，则要使用 DBMS_SQL.define_column 定义输出变量，然后使用 DBMS_SQL.Execute, DBMS_SQL.Fetch_Rows, DBMS_SQL.Column_Value 及 DBMS_SQL.Variable_Value 来执行查询并得到结果。

总结说明：

在 Oracle 开发过程中，我们可以使用动态 SQL 来执行 DDL 语句、DML 语句、事务控制语句及系统控制语句。但是需要注意的是，PL/SQL 块中使用动态 SQL 执行 DDL 语句的时候与别的不同，在 DDL 中使用绑定变量是非法的（bind_variable(v_cursor,':p_name',name)），分析后不需要执行 DBMS_SQL.Bind_Variable，直接将输入的变量加到字符串中即可。另外，DDL 是在调用 DBMS_SQL.PARSE 时执行的，所以 DBMS_SQL.EXECUTE 也可以不用，即在上例中的 v_row:=dbms_sql.execute(v_cursor)部分可以不要。

本文你可以在作者的 Blog 上找到，更多内容请登陆作者的 Blog。

作者简介:

丁亚军，网名：dinya

曾就职于机械相关行业，后做 Windows 程序开发。

现就职于一家制造企业，做 Oracle ERP 二次开发工作，对 Oracle ERP 二次开发和 Oracle 数据库开发有一定研究,现任 CSDN 论坛 Oracle 开发版版主。

作者Mail：dinya20@tom.com

作者Blog：http://blog.csdn.net/dinya2003/

关于竖表转横表的问题

本文作者 dinya

内容摘要：

在开发过程，经常遇到一些将表的显示方式进行转换的需求，我们习惯性称之为竖表到横表的转换，本文通过一个例子来简要说明常见的两种竖表转横表的问题。

本文适宜读者范围：

Oracle 初级，中级

系统环境：

OS：windows 2000 Professional (英文版)

Oracle：8.1.7.1.0

正文：

在实际的应用中，我们经常遇到需要转换数据显示方式，比如将横表转为竖表，或将竖表转换为横表的情况，如：课程表的显示方式，部门平均工资的排名等情况。下面将将根据两个实例子的需求描述给出两种常见的竖表转横表的解决办法（本例中的数据意思是：一、二、三年级的各科目最高分统计）。

表结构：

```
create table test_table
(
   grade_id number(8),           --年级：1、一年级，2、二年级，3、三年级
   subject_name varchar2(30),    --科目：包含语文、数学、外语、政治等科目
   max_score number(8)           --最高分
)
```

表中数据：

```
SQL> select * from test_table;

 GRADE_ID    SUBJECT_NAME                MAX_SCORE

       1     语文                             95
```

```
         1      数学                              98

         2      语文                              86

         2      数学                              90

         2      政治                              87

         3      语文                              93

         3      数学                              88

         3      英语                              88

         3      政治                              97

9 rows selected.
```

第一种转换方式：

需求描述：查看每个年级在系统中存在的科目信息，并各年级的科目信息按下面的格式显示：

```
GRADE_ID     SUBJECT_NAME

       1      语文  数学

       2      语文  数学  政治

       3      语文  数学  英语  政治
```

分析：在要求得到的结果中，每个年级的科目将变成一条记录，而且每个年级的科目是不固定的。所以考虑写个函数来解决，输入年级信息，使用游标得到该年级的所有科目信息并返回值。

1、建函数：

```
SQL> create or replace function test_fun(p_grade number) return varchar2 as
  2     v_temp varchar2(100):='';
  3     v_out varchar2(500):='';
  4          cursor  c  is  select  a.subject_name  from  test_table  a  where
a.grade_id=p_grade;
  5  begin
  6     open c ;      --打开游标
  7     loop
  8        fetch c into v_temp;
  9        exit when c%notfound;
```

```
 10        v_out:=v_out||'  '||v_temp;
 11     end loop;
 12     close c;      --关闭游标
 13     return v_out;
 14     exception
 15        when others then
 16           return 'An error occured';
 17  end ;
 18  /
Function created.


SQL> create or replace function test_fun(p_grade number) return varchar2 as
  2     v_out varchar2(500):='';
  3          cursor  c  is  select  a.subject_name  from  test_table  a  where
a.grade_id=p_grade;
  4  begin
  5     for v_temp in c loop
  6        v_out:=v_out||'  '||v_temp.subject_name;
  7     end loop;      --系统自动关闭游标
  8     return v_out;
  9     exception
 10        when others then
 11           return 'An error occured';
 12  end ;
 13  /
Function created.
```

2、调用函数得到输入结果：

```
SQL> select distinct a.grade_id,test_fun(a.grade_id) subject from test_table a;

 GRADE_ID          SUBJECT
```

```
语文  数学

语文  数学  政治

语文  数学  英语  政治
```

第二种转换方式：

需求描述：要求将表中的年级、科目及最高的信息按照下表的格式显示,如果该年级没开的课程,则其最高分用 0 表示:

```
年级      语文        数学        英语        政治

一年级      95          98          0           0

二年级      86          90          0          87

三年级      93          88         88          97
```

分析：该需求将年级的分数及科目信息由纵向转为横向，这样就要针对每个年级的，对其科目进行判断，存在科目则显示科目的最高分，如果不存在显示 0。这时候就考虑到使用 decode 函数来解决。实现如下：

```
 select
    decode(t.grade_id,1,'一年级',2,'二年级',3,'三年级') 年级,
    sum(decode(t.subject_name,'语文',t.max_score,0)) 语文,
    sum(decode(t.subject_name,'数学',t.max_score,0)) 数学,
    sum(decode(t.subject_name,'英语',t.max_score,0)) 英语,
    sum(decode(t.subject_name,'政治',t.max_score,0)) 政治
from
    test_table t
group by
    t.grade_id
```

需要说明的是，在第一种转换方式中写了两个函数,两个函数实现的是同一个需求，所不同的是，两个函数中游标使用方式不同，地一个函数中手动打开游标，循环结束后要求手动关闭。而后一个函数使用 for 循环，循环结束后系统自动关闭光标。在第二种转换方式中，使用了 decode 函数，关于 decode 的详细用法，请参考 oracle 函数相关文档。

总结：

上面的两种转换方式是在开发中经常遇到的情况,在开发中的其他类似的转换都可以参考上面的转换方式,使用 decode,nvl 等函数进行一些特别的处理即可得到想要的显示方式.

本文你可以在作者的 Blog 上找到，更多内容请登陆作者的 Blog。

作者简介:

丁亚军，网名：**dinya**

曾就职于机械相关行业，后做 **Windows** 程序开发。

现就职于一家制造企业，做 **Oracle ERP** 二次开发工作，对 **Oracle ERP** 二次开发和 **Oracle** 数据库开发有一定研究,现任 **CSDN** 论坛 **Oracle** 开发版版主。

作者**Mail**：**dinya20@tom.com**

作者**Blog**：**http://blog.csdn.net/dinya2003/**

利用游标返回数据集两则 - Delphi & VB

本文作者：chanet

[测试环境]

数据库版本：Oracle 9.2

操作系统：Win2000

一. Delphi 例子：

```
一、先在 Oracle 建好
    CREATE OR REPLACE PACKAGE pkg_test
    AS
       TYPE myrctype IS REF CURSOR;
       PROCEDURE get(i_test INTEGER,p_rc OUT myrctype);
    END pkg_test;

   CREATE OR REPLACE PACKAGE BODY pkg_test
   AS
      PROCEDURE get(i_test INTEGER,p_rc OUT myrctype) IS
      BEGIN
        IF i_test = 0 THEN
           OPEN p_rc FOR SELECT SYSDATE FROM dual;
        ELSE
           OPEN p_rc FOR SELECT * FROM tab;
        END IF;
      END get;
   END pkg_test;

二、用 Delphi 调用
   建一个窗体,拖动控件 AdoConnection1 , ADOStoredProc1 和 Button1.
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    try
      with ADOConnection1 do
      begin
        ConnectionString:=
            'Provider=OraOLEDB.Oracle.1;'
```

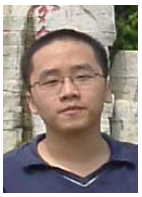
```
          + 'Password=密码;'
          + 'Persist Security Info=True;'
          + 'User ID=用户名;'
          + 'Data Source=数据库名;'
          + 'Extended Properties="PLSQLRSet=1;"';
        Open;
      end;
    except
      showMessage('连接不成功');
      exit;
    end;

    try
      with ADOStoredProc1 do
      begin
        Connection := ADOConnection1;
        Parameters.Clear;
        ProcedureName:= 'pkg_test.get';
        Parameters.CreateParameter('p1',ftInteger,pdInput,10,1);
        Open;
      end;
    except
      showMessage('无法执行过程.');
    end;
  end;
```

二．VB 例子：

```
PL/SQL 代码:
CREATE OR REPLACE PACKAGE "PKG_TEST" AS
       TYPE myrcType IS REF CURSOR;
       FUNCTION get(strbarcode VARCHAR) RETURN myrcType;
END pkg_test;

CREATE OR REPLACE PACKAGE BODY "PKG_TEST" AS
    FUNCTION get(strbarcode IN VARCHAR) RETURN myrcType IS
      rc myrcType;
    BEGIN
      OPEN rc FOR strbarcode;
      RETURN rc;
    END get;
END pkg_test;
```

```
VB 代码:
Private Sub Command1_Click()
On Error GoTo cursorErr:
    Dim cnn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    Dim cmd As New ADODB.Command

    cnn.ConnectionString = "Provider=OraOLEDB.Oracle.1;Password=tiger;Persist
Security      Info=True;User      ID=scott;Data      Source=oraAny;Extended
Properties=PLSQLRSet=1"
    cnn.Open

    With cmd
        .ActiveConnection = cnn
        .CommandType = adCmdText
        .CommandText = "{CALL scott.pkg_test.get(?)}"
            .Parameters.Append   .CreateParameter("strBarCode",   adVarChar,
adParamInput, 100, "SELECT * FROM TAB")
    End With

    rst.CursorType = adOpenStatic
    rst.LockType = adLockReadOnly
    Set rst.Source = cmd
    rst.Open

    MsgBox rst.RecordCount

    Set rst = Nothing
    Set cmd = Nothing
    Exit Sub

cursorErr:
    Set cmd = Nothing
    Set rst1 = Nothing
    MsgBox Err.Description
End Sub
```

作者简介:



chanet(牧师)

曾就职于机械相关行业，后做 Windows 程序开发。

现就职于广州某软件公司；主要负责软件的后台设计、数据接口技术支持、数据库开发与维护工作。有丰富的 pl/sql 编程,c/s 架构,数据库设计 经验;熟悉 Oracle 开发,基本的管理;

Oracle job 基础

本文作者:biti_rainy (biti_rainy@itpub.net)

摘要:

本文简单介绍了 oracle job 的使用和一些基础常识，为后面 job 文章做个铺垫。

一、什么是 oracle job?

Oracle job 是数据库自身提供的定期执行特定代码的功能（通常是执行 pl/sql 代码），这是我们在数据库内部完成定时任务的基础。与此对应的，在 unix 类(包含 linux)操作系统中通过 crontab 提供能定期任务的功能，而 windows 中也提供了定期任务的功能，从道理上来讲，凡是 oracle 能实现的定期任务我们都可能通过 os 来实现，只不过任务放置在数据库内部，也有其灵活和方便的一面。总体来讲，只有深入地了解了 crontab 与 oracle job 之后，我们才能针对数据库处理任务在适当的场合选择适当的处理方式。

二、oracle job 基础

job 进程与数据库初始化参数

在 oracle 9i 以前版本,要运行 job 需要设置参数 job_queue_interval 和 job_queue_processes。job_queue_interva 是进程去检查是否有任务需要执行的检查周期（单位是秒），也就是说一个任务执行时间点和定义时间点可能会出现这个大的偏差（假定 job 空闲进程足够），job_queue_processes 表示 oracle 允许的执行执行 job 任务的最大并发进程数量,这意味着如果同时执行的任务数量大于允许进程数量则要出现等待。而在 oracle9i 中则取消了 job_queue_interval 参数，使用 ora_cjq*_sid 进程来定期检查和启动 job 进程执行任务。

Oracle8.0.4 版本

```
$   svrmgrl

Oracle Server Manager Release 3.0.4.0.0 - Production

(c) Copyright 1997, Oracle Corporation.  All Rights Reserved.
```

```
Oracle8 Release 8.0.4.0.0 - Production

PL/SQL Release 8.0.4.0.0 - Production


SVRMGR> connect  internal

Connected.

SVRMGR> show parameter  job

NAME                           TYPE    VALUE

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

job_queue_interval               integer 60

job_queue_keep_connections        boolean FALSE

job_queue_processes              integer 0

SVRMGR> exit

Server Manager complete.

$
```

Oracle9.2.0.4 版本

```
[oracle@ocn1 oracle]$ sqlplus  "/ as sysdba"


SQL*Plus: Release 9.2.0.4.0 - Production on Mon Dec 6 15:35:10 2004


Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected to:

Oracle9i Enterprise Edition Release 9.2.0.4.0 - Production

With the Partitioning, Real Application Clusters, OLAP and Oracle Data Mining
options

JServer Release 9.2.0.4.0 - Production


SQL> show parameter  job
```

```
NAME                           TYPE       VALUE

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

job_queue_processes               integer    2

SQL>
```

Linux OS

```
[oracle@ocn1 oracle]$ ps -ef|grep  cjq

oracle   26299    1  0 Oct29 ?       00:00:01 ora_cjq0_ocn1

oracle   23120 18364  0 15:36 pts/2    00:00:00 grep cjq

[oracle@ocn1 oracle]$
```

创建与修改 job

我有一个存储过程名称为 my_proc,需要每天中午 12:30 定时执行

```
declare

i  number;

begin

dbms_job.submit(i,'my_proc;',trunc(sysdate)+1+12.5/24,'trunc(sysdate)+1+12.5
/24');

commit;

end;
```

这里想提醒一点如果我们的存储过程只执行一个简单的任务，比如 delete from t ，那么没必要创建一个存储过程可以直接把 my_proc 替换成 begin delete from t ; end 。

我们可以通过 view user_jobs 查询相关 job 信息

```
select  job,what, INTERVAL from  user_jobs where  what  like  'my_proc%';

      JOB WHAT      NTERVAL

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

      147 my_proc;    trunc(sysdate)+1+12.5/24
```

```
SQL>  select   job,NEXT_DATE,INSTANCE   from   user_jobs  where   what   like
'my_proc%';


     JOB NEXT_DATE         INSTANCE

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

     147 20041207 12:30:00        1
```

job 的创建和修改都可以通过 dbms_job 包来完成，这个包的用法很简单，要修改 job 的属性，一般都是输入 job 编号和要修改的属性的值，主要包含以下过程或者函数

```
alibaba@OCN> desc  dbms_job

FUNCTION BACKGROUND_PROCESS RETURNS BOOLEAN

PROCEDURE BROKEN

 Argument Name                  Type                    In/Out Default?

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

 JOB                            BINARY_INTEGER          IN

 BROKEN                         BOOLEAN                 IN

 NEXT_DATE                      DATE                    IN     DEFAULT

PROCEDURE CHANGE

 Argument Name                  Type                    In/Out Default?

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

 JOB                            BINARY_INTEGER          IN

 WHAT                           VARCHAR2                IN

 NEXT_DATE                      DATE                    IN

 INTERVAL                       VARCHAR2                IN

 INSTANCE                       BINARY_INTEGER          IN     DEFAULT

 FORCE                          BOOLEAN                 IN     DEFAULT

PROCEDURE INSTANCE

 Argument Name                  Type                    In/Out Default?

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

 JOB                            BINARY_INTEGER          IN
```

```
 INSTANCE                       BINARY_INTEGER          IN

 FORCE                          BOOLEAN                 IN     DEFAULT

PROCEDURE INTERVAL

 Argument Name                   Type                    In/Out Default?

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

 JOB                            BINARY_INTEGER          IN

 INTERVAL                       VARCHAR2                IN

PROCEDURE ISUBMIT

 Argument Name                   Type                    In/Out Default?

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

 JOB                            BINARY_INTEGER          IN

 WHAT                           VARCHAR2                IN

 NEXT_DATE                      DATE                    IN

 INTERVAL                       VARCHAR2                IN     DEFAULT

 NO_PARSE                       BOOLEAN                 IN     DEFAULT

FUNCTION IS_JOBQ RETURNS BOOLEAN

PROCEDURE NEXT_DATE

 Argument Name                   Type                    In/Out Default?

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

 JOB                            BINARY_INTEGER          IN

 NEXT_DATE                      DATE                    IN

PROCEDURE REMOVE

 Argument Name                   Type                    In/Out Default?

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

 JOB                            BINARY_INTEGER          IN

PROCEDURE RUN

 Argument Name                   Type                    In/Out Default?

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

 JOB                            BINARY_INTEGER          IN

 FORCE                          BOOLEAN                 IN     DEFAULT
```

```
PROCEDURE SUBMIT
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 JOB                            BINARY_INTEGER          OUT
 WHAT                           VARCHAR2                IN
 NEXT_DATE                      DATE                    IN     DEFAULT
 INTERVAL                       VARCHAR2                IN     DEFAULT
 NO_PARSE                       BOOLEAN                 IN     DEFAULT
 INSTANCE                       BINARY_INTEGER          IN     DEFAULT
 FORCE                          BOOLEAN                 IN     DEFAULT
PROCEDURE USER_EXPORT
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 JOB                            BINARY_INTEGER          IN
 MYCALL                         VARCHAR2                IN/OUT
PROCEDURE USER_EXPORT
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 JOB                            BINARY_INTEGER          IN
 MYCALL                         VARCHAR2                IN/OUT
 MYINST                         VARCHAR2                IN/OUT
PROCEDURE WHAT
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 JOB                            BINARY_INTEGER          IN
 WHAT                           VARCHAR2                IN
```

比如我们要修改 job 的的定期时间为凌晨 2 点 50 分

```
begin
dbms_job.interval(147,'trunc(sysdate)+1+170/24/60')
```

```
end;
```

在 oracle 里面时间是按天为单位的，比如 1 表示 1 天，170/24/60 表示 170 分钟则是 2 小时 50 分，trunc(sysdate)+1+170/24/60 表示当前 job 执行完毕后，从当天零点开始计算增加 1 天零 170 分钟，把这个时间点作为下次 job 执行的起点。

Job interval 的设置技巧

关于定时任务的设置技巧主要集中在 intelval 上，我个人习惯习惯是，如果按天的任务，使用 trunc(sysdate)+增量 的方式，如果要在一天的某些特定时间点执行定期任务，比如我们的系统中有一个需求就是在 8:30,9:30,10:30,15:00,16:00,17:00,23:00 这些点搜集 statspack 信息供长期分析，那么我使用的 interval 是

```
trunc(sysdate)+
decode(to_char(sysdate,'hh24'),8,9.5,9,10.5,10,15,15,16,16,17,17,23,32.5)/24
```

这里 decode 表示如果当前小时数是 8 点则下一个时间点就是 9.5/24 表示 9 点半，如果是 9 则下一个时间点是 10.5/24 表示 10 点半……else 是 32.5/24 表示第二天的 8 点半，这样构成了一个循环，当然我这里的前提是我确保这个任务能在半小时内完成。

如果我们期望一个任务只执行一次，那么我们可以将 interval 设置为 null，job 执行一次后就消失了。当然，如果有更特别的要求，通常使用 decode 或者结合其他函数能满足大部分的要求，实在复杂甚至可以自己定义一个函数来实现也成。

关于 interval 影响到 next_date（下次执行时间点）的问题，kamus 的文章中有详细的探讨。

三．Job 的问题诊断

通常，一个数据库的任务没有正常执行，首先是检查相关的 job 的初始化参数。根据我对网上问题的总结大部分都是因为进程参数设置为 0 导致的，然后接下来就是检查 job 的状态和信息。通过 user_jobs 我们可以发现 job 的很多信息，包括 interval 设置是否合理、failures 表示当前 job 连续失败次数、LAST_DATE 表示上次 job 执行时间、TOTAL_TIME 表示 job 任务运行时间总和、BROKEN 表示 job 是否被停止、INSTANCE 表示 job 运行的节点编号（仅仅针对 ops 或者 rac 有效）。通过这些信息的分析，我们能发现绝大部分问题并给予解决。

有一种特例，如果一个 job 没有运行，并且手工 dbms_job.run(job#) 返回成功但就是没有执行，这时我们需要检查 v$lock 看是否有正锁定但没有正常执行的 job。

```
select * from  v$lock where  type =  'JQ';

alibaba@OCN>desc  v$lock

 Name                                      Null?    Type

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

 ADDR                                               RAW(4)
```

```
KADDR                                          RAW(4)
SID                                            NUMBER
TYPE                                           VARCHAR2(2)
ID1                                            NUMBER
ID2                                            NUMBER
LMODE                                          NUMBER
REQUEST                                        NUMBER
CTIME                                          NUMBER
BLOCK                                          NUMBER
```

这里 type 表示 lock 类型，ctime 表示锁存在了多长时间,如果 type='JQ'并且 ctime 非常的大（单位是秒），而我们的 job 通常存在问题（可能因为网络、os 等等异常导致），当然还可以进一步的明确问题，在这里不做过多的介绍了，通过这里我们可以得到 SID，这表示正在执行这个任务的 session 的 SID，然后执行

```
select sid,serial# from  v$session where  sid =  ? ;

select  spid from  v$process where addr = (select  padrr from  v$session where sid
= ?);
```

通过查询获得的 sid 和 serial ，通过拥有 alter system 权限的数据库用户执行

```
alter  system  kill session  'sid,serial#';
```

为了保险起见，我们要在 os 上 kill 通过 v$process 获得的 spid 这个进程号，因为在实际环境中我们发现不做这步很容易出现问题。

```
kill -9   spid
```

windows 上可以在命令行下使用 oracle 提供的 orakill 工具杀掉线程

```
C:\Documents and Settings\hz>orakill


Usage:  orakill sid thread


  where sid    = the Oracle instance to target

        thread = the thread id of the thread to kill
```

The thread id should be retrieved from the spid column of a query such as:

```
select spid, osuser, s.program from
v$process p, v$session s where p.addr=s.paddr
```

作者简介:

网名 biti_rainy

CSDN eMag Oracle 电子杂志主编。

开发出身，对数据库应用设计中如何正确地应用 oracle 特性以扬长避短具有深刻理解。

有丰富的 oracle 实践经验，对数据库的体系结构、备份恢复、sql 优化、数据库整体性能优化、oracle internal 都有深入研究。

曾于某电信集成公司负责计费系统的开发，然后成为某系统集成公司的 DBA，再辗转在香港一家跨国公司珠海研发中心担任技术负责人(公司主要产品就是 sql 与数据库优化工具,产品主要销往欧洲和北美),此后成为自由职业者，为客户提供独立的 oracle 数据库的技术服务和高级性能调整等培训，同时提供 ITPUB 华南和华东的培训。

目前服务于国内某大型电子商务网站，维护系统数据库并提供开发支持.

个人Blog等
http://blog.csdn.net/biti_rainy
http://blog.itpub.net/biti_rainy

Oracle 计划作业(JOB)的探讨

本文作者：kamus(kamus@itpub.net)

摘要:

本文通过实验和事件跟踪来分析 Oracle Job 执行过程中修改下次执行时间的机制，JOB 执行错误以后的尝试执行间隔以及 Oracle 作业的其它相关方面。

问题的提出

有些人问，Oracle 的 JOB 在设定完 next_date 和 interval 之后，到底是什么时候决定下一次运行时间的。可以归纳成以下几个问题。

假设我们的 JOB 设定第一次运行的时间是 12:00，运行的间隔是 1 小时，JOB 运行需要耗时 30 分钟，那么第二次运行是在 13:00 还是 13:30？

如果是在 13:00 那是不是说明只要 JOB 一开始运行，next_date 就被重新计算了？

JOB 的下一次运行会受到上一次运行时间的影响吗？如果受到影响，如何可以避免这个影响而让 JOB 在每天的指定时刻运行？

假设我们的 JOB 设定第一次运行的时间是 12:00，运行的间隔是 30 分钟，JOB 运行需要耗时 1 小时，那么第二次运行是在 12:30 还是 13:00 还是根本就会报错？

对于一个执行错误的 JOB，oracle 是否会尝试继续执行，如果会尝试继续执行那么隔多长时间尝试一次，总共尝试多少次以后会放弃？

Oracle 的作业是不是只能调用存储过程，如果我们想在 Oracle 内部管理操作系统级别命令（比如一个 shell 脚本）的计划任务可以实现吗？

本文通过一些实验和跟踪来解释上面的所有问题。

问题解答 I

首先我们选择一个测试用户，假设该用户名为 kamus。

由于我们在实验用的存储过程中会用到 dbms_lock 包，所以需要由 sys 用户先授予 kamus 用户使用 dbms_lock 包的权限。

```
d:\Temp>sqlplus "/ as sysdba"


SQL*Plus: Release 9.2.0.5.0 - Production on 星期三 12 月 1 23:56:32 2004


Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.



连接到:

Oracle9i Enterprise Edition Release 9.2.0.5.0 - Production

With the Partitioning, OLAP and Oracle Data Mining options

JServer Release 9.2.0.5.0 - Production


SQL> grant execute on dbms_lock to kamus;


授权成功。
```

然后用 kamus 用户登录数据库，创建我们测试使用的存储过程 sp_test_next_date。

```
create or replace procedure sp_test_next_date as
  p_jobno    number;
  P_nextdate date;
begin
  --将调用此存储过程的job 的next_date 设置为30 分钟以后
  select job into p_jobno from user_jobs where what = 'sp_test_next_date;';
  execute  immediate  'begin  dbms_job.next_date('  ||  to_char(p_jobno)  ||
',sysdate+1/48);commit;end;';
  --修改完毕以后检查user_jobs 视图，输出job 目前的next_date
  select next_date
    into P_nextdate
    from user_jobs
   where what = 'sp_test_next_date;';
```

```
  dbms_output.put_line('JOB执行中的next_date: ' ||
                       to_char(p_nextdate,'YYYY-MM-DD HH24:MI:SS'));
  --等待10秒再退出执行
  dbms_lock.sleep(seconds => 10);
end sp_test_next_date;
```

创建调用该存储过程的 JOB，定义 interval 为每天一次，也就是这次执行以后，下次执行时间应该在 1 天以后。

```
SQL> variable jobno number;

SQL> BEGIN

  2  DBMS_JOB.SUBMIT(job => :jobno,

  3  what => 'sp_test_next_date;',

  4  next_date => SYSDATE,

  5  interval => 'SYSDATE+1');

  6  COMMIT;

  7  END;

  8  /


PL/SQL 过程已成功完成。


jobno

---------

1
```

然后我们手工执行存储过程，执行完毕以后再手工从 user_jobs 视图中获得 JOB 的下次执行时间，可以看到在存储过程中修改的 JOB 的下次执行时间已经生效，变成了当前时间的 30 分钟以后，而不是默认的 1 天以后。

```

SQL> conn kamus

请输入口令:

已连接。
```

```
SQL> set serverout on

SQL> exec sp_test_next_date();

JOB 执行中的 next_date: 2004-12-02 00:44:11


PL/SQL 过程已成功完成。


SQL> col next_date for a20

SQL> select to_char(next_date,'YYYY-MM-DD HH24:MI:SS') next_date from user_jobs

where what = 'sp_test_next_date;';


NEXT_DATE

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

2004-12-02 00:44:11
```

我们再手工运行 JOB，看看这次的结果，可以发现 JOB 没有运行完毕以前被修改了的下次运行时间跟 JOB 运行完毕以后再次手工检索 user_jobs 视图获得的下次运行时间已经不相同了。由此我们可以得出一个结论，next_date 是在 JOB 运行完毕以后被 Oracle 自动修改的，而不是在 JOB 刚开始运行的时候，因为我们在存储过程中修改的 next_date 在 JOB 运行结束之后又被修改为默认的 1 天以后了。

```

SQL> exec dbms_job.run(1);

JOB 执行中的 next_date: 2004-12-02 00:54:52


PL/SQL 过程已成功完成。


SQL> select to_char(next_date,'YYYY-MM-DD HH24:MI:SS') next_date from user_jobs

where what = 'sp_test_next_date;';


NEXT_DATE

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

2004-12-03 00:24:52
```

现在我们再次修改存储过程，输出存储过程开始执行的时间，便于跟执行完毕以后的 JOB 下次执行时间进行比较。

```
create or replace procedure sp_test_next_date as
  p_jobno    number;
  P_nextdate date;
begin
  --输出 JOB 刚开始执行的时间
  dbms_output.put_line(' JOB 开始执行的时间: ' ||
                       to_char(sysdate, 'YYYY-MM-DD HH24:MI:SS'));
  --将调用此存储过程的 job 的 next_date 设置为 30 分钟以后
  select job into p_jobno from user_jobs where what = 'sp_test_next_date;';
  execute  immediate  'begin  dbms_job.next_date('  ||  to_char(p_jobno)  ||
',sysdate+1/48);commit;end;';
  --修改完毕以后检查 user_jobs 视图，输出 job 目前的 next_date
  select next_date
    into P_nextdate
    from user_jobs
   where what = 'sp_test_next_date;';
  dbms_output.put_line(' JOB 执行中的 next_date: ' ||
                       to_char(p_nextdate,'YYYY-MM-DD HH24:MI:SS'));
  --等待 10 秒再退出执行
  dbms_lock.sleep(seconds => 10);
end sp_test_next_date;
```

重新进行测试，我们可以发现 JOB 的 next_date 是 JOB 开始执行时间的 1 天以后，而不是 JOB 结束时间的 1 天以后（因为 JOB 结束需要经过 10 秒钟）

```
SQL> exec dbms_job.run(1);

JOB 开始执行的时间: 2004-12-02 00:38:24

JOB 执行中的 next_date: 2004-12-02 01:08:24


PL/SQL 过程已成功完成。


SQL> select to_char(next_date,'YYYY-MM-DD HH24:MI:SS') next_date from user_jobs

where what = 'sp_test_next_date;';
```

```
NEXT_DATE

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

2004-12-03 00:38:24
```

至此，我们已经说明了两个问题。就是：JOB 在运行结束之后才会更新 next_date，但是计算的方法是 JOB 刚开始的时间加上 interval 设定的间隔。

下面我们通过 trace 来再次求证这个结论。

```
SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 12';

会话已更改。

SQL> exec dbms_job.run(1);

PL/SQL 过程已成功完成。

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

会话已更改。
```

执行完毕以后在 udump 目录中查看生成的 trace 文件。如果我们用 tkprof 来格式化这个 trace 文件然后再查看格式化后的结果，我们会感到很诧异。因为在格式化完毕的 SQL 执行顺序中，更新 job$表的语句出现在 dbms_job.next_date 语句之前，也就是看上去是 Oracle 先按照 interval 自动更新了 JOB 的 next_date，然后才继续往下执行存储过程中定义的 next_date 更新语句，而这样显然无法解释我们在上面的实验中看到的结果。

但是当我们跳过 tkprof 而直接去查看生成的 trace 文件，就会恍然大悟，同时也印证了 steve adams 在 ixora 上提到的观点：tkprof 格式化完的结果会省略一些信息，甚至在有时候会给我们错误的信息。

直接查看 trace 文件，我们可以看到如下的执行顺序：

```
parse cursor #10（oracle 根据 interval 和先前保存的 this_date 字段值更新 job$表的语句，包括更新 failures, last_date, next_date, total 等）
```

```
parse cursor #15（存储过程中的 begin dbms_job.next_date 语句）

binds cursor #15（将加上了 30 分钟的时间绑定到 cursor #15 上）

exec cursor #15（执行 cursor #15）

wait cursor #11（经历一个 PL/SQL lock timer 事件，也就是存储过程中执行的 dbms_lock.sleep 方法）

binds cursor #10（将 JOB 刚开始执行时候的时间绑定到 cursor #10 上）

exec cursor #10（执行 cursor #10）
```

也就是说虽然更新 job$的语句被很早地解析过了，但是直到 JOB 运行结束时这个被解析过的游标才开始作变量绑定进而开始执行。

正是因为解析 update sys.job$语句的时间早于解析 begin dbms_job.next_date 语句的时间，所以 tkprof 的结果将前者放在了前面。

问题解答 II

接下来我们进入另外一个问题的探讨，本文最开始提到的第四个问题：

假设我们的 JOB 设定第一次运行的时间是 12:00，运行的间隔是 30 分钟，JOB 运行需要耗时 1 小时，那么第二次运行是在 12:30 还是 13:00 还是根本就会报错？

通过分析 trace 文件我们可以找到更新 next_date 的 SQL 语句是：

```
update sys.job$ set failures=0, this_date=null, flag=:1, last_date=:2,
next_date=greatest(:3,sysdate),total=total+(sysdate - nvl(this_date, sysdate))
where job=:4
```

注意到更新 next_date 字段的公式是 greatest(:3, sysdate)，此处的:3 绑定的是 job 的 this_date+interval。所以我们猜测实际上应该是有一个跟当前时间的比较机制，如果在执行完 JOB 之后的时间比按照 this_date+interval 计算出的时间更晚一些，那么 next_date 就更新为当前时间，也就是几乎会立刻再重新执行 JOB。

同样这样的猜测我们也需要通过实验来验证一下。

创建一个新的存储过程 sp_test_next_date1，简单地等待 2 分钟，但是我们将调用这个存储过程的 JOB 的 interval 设置为 1 分钟，看看会有什么情况。

为了更方便得比较，我们创建一个表用来记录每次 JOB 执行的开始时间。

```
SQL> create table t (cdate date);


Table created
```

创建存储过程的脚本

```
create or replace procedure sp_test_next_date1 as
begin
  --输出 JOB 开始执行的时间
  insert into t(cdate) values(sysdate);
  commit;
  --等待 120 秒退出
  dbms_lock.sleep(seconds => 120);
end sp_test_next_date1;
```

创建调用此存储过程的 JOB

```
SQL> variable jobno number;
SQL> BEGIN
  2  DBMS_JOB.SUBMIT(job => :jobno,
  3  what => 'sp_test_next_date1;',
  4  next_date => SYSDATE,
  5  interval => 'SYSDATE+1/1440');
  6  COMMIT;
  7  END;
  8  /

PL/SQL 过程已成功完成。


jobno
---------
```

```
7
```

执行此 JOB，然后过一段时间开始检查表 t 中的输出。

```
SQL> select * from t order by cdate;

CDATE
--------------------
2004-12-3 14:10:43
2004-12-3 14:12:47
2004-12-3 14:14:55
2004-12-3 14:16:59
2004-12-3 14:19:07
2004-12-3 14:21:11

6 rows selected
```

首先我们确认 JOB 每次都是成功执行了，并没有任何报错，然后检查 cdate 字段，发现时间间隔都是 2 分钟左右，也就是说因为 JOB 本身的 interval 设定比 JOB 本身的执行时间要长，所以 Oracle 将 next_date 设置为每次 JOB 结束的时间。

同时我们也注意到，每次开始的时间都有 4 秒到 8 秒的延迟，没有继续深究，不确认这是因为 oracle 本身计算的误差，还是内部比如启动 Job Process 需要的时长。

不论如何，到此我们也已经回答了第四个问题，即使 interval 的时长短于 JOB 执行的时间，整个作业仍然会继续进行，只是执行间隔变为了 JOB 真实运行的时长。

由于trace文件过长，所以不在本文中贴出了，如果有兴趣可以发邮件给我。我的邮件地址是：kamus@itpub.net

BTW：在 trace 文件中发现虽然通过 select rowid from table 返回的结果已经是扩展 ROWID 格式（Data Object number + File + Block + ROW）了，但是 oracle 内部检索数据仍然在使用限制 ROWID 格式（Block number.Row number.File number）。

问题解答 III

本小节解答本文开头提出的第三个问题，也就是：

JOB 的下一次运行会受到上一次运行时间的影响吗？如果受到影响，如何可以避免这个影响而让 JOB 在每天的指定时刻运行？

JOB 的下一次运行时间是会受上一次影响的，如果我们的 interval 仅仅是 sysdate+1/24 这样的形式的话，无疑，上次执行的时间再加上 1 小时就是下次执行的时间。那么如果 JOB 因为某些原因延迟执行了一次，这样就会导致下一次的执行时间也同样顺延了，这通常不是我们希望出现的现象。

解决方法很简单，只需要设定正确的 interval 就可以了。

比如，我们要 JOB 在每天的凌晨 3:30 执行而不管上次执行到底是几点，只需要设置 interval 为 trunc(SYSDATE)+3.5/24+1 即可。完整的 SQL 如下：

```
SQL> variable jobno number;
SQL> BEGIN
  2  DBMS_JOB.SUBMIT(job => :jobno,
  3  what => 'sp_test_next_date;',
  4  next_date => SYSDATE,
  5  interval => 'trunc(SYSDATE)+3.5/24+1');
  6  COMMIT;
  7  END;
  8  /
```

问题解答 IV

JOB 中调用存储过程，由于种种原因，不可避免地会出现错误。当被调用的存储过程出错，JOB 就会标志为一次 failure（failure 的次数可以从 user_jobs.failures 字段中得到），按照 Oracle 文档中提到的当一个 JOB 连续出现了 16 次 failure，该 JOB 就会标志为 Broken，除非手动修改这个 JOB 的 Broken 标志为 False，或者成功地手工执行一次 JOB，否则该 JOB 将永远不会被再次自动执行。

但是这个 JOB 的 16 次尝试执行是在多长的时间内发生的？每次尝试执行之间的时间间

隔又是多少？

下面仍然通过一个实验来解答。

思路如下，创建一个存储过程，在存储过程里面通过一个 dblink 从远程数据库上检索数据，然后将远程数据库上的监听关闭，此时这个存储过程就会执行出错，然后观察调用这个存储过程的 JOB 执行频率来得出我们的实验结果。

创建存储过程的脚本，其中 fail 为 dblink 的名称。创建 dblink 的脚本在本文中就省略了，如果要得到相关知识可以阅读我的其它文章。

```
create or replace procedure sp_test_fail as
begin
  -输出 JOB 开始的执行时间
  insert into t(cdate) values(sysdate);
  commit;
  --执行一次将会失败的语句
  insert into t1 select sysdate from dual@fail;
end sp_test_fail;
```

创建调用此存储过程的 JOB，设置 JOB 的 interval 为 9 分钟

```
SQL> variable jobno number;
SQL> BEGIN
  2  DBMS_JOB.SUBMIT(job => :jobno,
  3  what => 'sp_test_fail;',
  4  next_date => SYSDATE,
  5  interval => 'SYSDATE+9/1440');
  6  COMMIT;
  7  END;
  8  /

PL/SQL 过程已成功完成。

jobno
```

```
---------

24
```

在测试中，我们让 JOB 先成功地执行两次，然后停止远程数据库上的监听，等待足够长的时间，我们再检查表 T 中的记录。从第三条记录开始（2004-12-6 15:26:19）出现 JOB 失败。

```
SQL> select rownum,cdate from (select * from t order by cdate);

   ROWNUM CDATE
---------- --------------------
         1 2004-12-6 15:08:10
         2 2004-12-6 15:17:15
         3 2004-12-6 15:26:19
         4 2004-12-6 15:28:23
         5 2004-12-6 15:32:30
         6 2004-12-6 15:40:38
         7 2004-12-6 15:49:43
         8 2004-12-6 15:58:47
         9 2004-12-6 16:07:52
        10 2004-12-6 16:16:56
        11 2004-12-6 16:26:00
        12 2004-12-6 16:35:05
        13 2004-12-6 16:44:09
        14 2004-12-6 16:53:14
        15 2004-12-6 17:02:18
        16 2004-12-6 17:11:22
        17 2004-12-6 17:20:26
        18 2004-12-6 17:29:30
```

```
SQL> select job,failures,broken from user_jobs;


      JOB   FAILURES BROKEN

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

       24         16 Y
```

上面的结果，应该已经很清楚地反映了我们的实验结果。

JOB 第一次发生失败以后，隔 2 分钟尝试下一次，如果仍然失败，则隔 2*2 分钟（4 分钟）再尝试执行一次，仍然失败，则隔 2*2*2 分钟（8 分钟）尝试执行第三次，如此类推，一直到下面三种情况的发生：

尝试执行了 16 次，标志 JOB 为 broken，不再自动执行

尝试执行的时间间隔超过了 JOB 本身 interval 的设置，比如本次测试中，interval 设置为 9 分钟，当进行第 3 次尝试执行以后仍然失败，此时到第 4 次的尝试执行应该是间隔 2*2*2*2*2（16 分钟），但是 16 分钟已经大于了 interval 的 9 分钟，所以第 4 次尝试执行实际上是在第 3 次尝试的 9 分钟之后发生的，同样的这之后的所有尝试都是间隔 9 分钟发生一次，一直到尝试 16 次之后，标志 JOB 为 borken。假设 16 次尝试中的间隔都不大于 interval 值，那么可以计算得到最后一次的尝试间隔将是 22 天以上。

```
SQL> select power(2,15) max_sec, power(2,15)/60 max_hour, power(2,15)/60/24
max_day from dual;


   MAX_SEC   MAX_HOUR    MAX_DAY

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

     32768 546.133333 22.7555555
```

在 16 次期限之前，执行成功了一次，JOB 重新按照 interval 设定的值开始定期执行，JOB 的 failures 值被归零。

那么如果一个 JOB 发生了错误，我们可以从哪里得知错误的原因呢。这就需要用到 DBA 诊断问题的得力助手 – alert.log 文件。检查相应 SID 的 alert.log 文件，比如本文中使用的 orcl 数据库，需要检查 bdump 目录中的 alert_orcl.log。

在告警日志文件中我们可以看到 JOB 运行错误的时间，原因。比如本例中第一次 JOB 失败时候记录的内容就明确地显示了出错的原因：ORA-12541: TNS:no listener

```
Mon Dec 06 15:26:21 2004
```

```
Errors in file c:\oracle\admin\orcl\udump\orcl_j000_2324.trc:

ORA-12012: error on auto execute of job 24

ORA-12541: TNS:no listener

ORA-06512: at "KAMUS.SP_TEST_FAIL", line 7

ORA-06512: at line 1
```

问题解答 V

我们使用 DBMS_JOB 包，只能在 JOB 中调用存储过程，或者直接写匿名的程序块，如果想调度操作系统级别的命令（比如一个 shell 脚本或者一个 bat 文件）那么就只能借助操作系统的调度功能了，在 UNIX 下面我们通常使用 cron，在 Windows 下面则可以使用 at。那么是不是有可能将这所有一切都交给 Oracle 来管理呢？

在 Oracle10g 发布之前这是很难做到的，而 Oracle10g 提供的 dbms_scheduler 包则提供了完善的解决方案。

关于 dbms_scheduler 包的使用，可以参考 Anup Nanda 发表在 OTN 上的文章。

Scheduler（英文版）：

http://www.oracle.com/technology/pub/articles/10gdba/week19_10gdba.html

调度程序（中文版）：

http://www.oracle.com/technology/global/cn/pub/articles/10gdba/week19_10gdba.html

顺便说一下，Anup Nanda 的 Oracle Database 10g: The Top 20 Features for DBAs 系列文章是每个想使用 Oracle10g 的人应该首先通读一下的。

本文涉及到的额外知识可以参看我的其它技术文章：

通过事件跟踪 SQL 执行的后台步骤

Oracle 等待事件，比如本文提到的 PL/SQL lock timer

ROWID 格式

DBLINK

作者简介:

张乐奕，网名 **kamus**

曾任 **ITPUB Oracle** 认证版版主,现任 **itpub Oracle** 管理版版主.

现任职于北京某大型软件公司，首席 **DBA**，主要负责证券行业的全国十数处核心交易系统数据库管理及维护工作。

热切关注**Oracle**技术和其它相关技术，出没于各大数据库技术论坛，目前是中国最大的**Oracle**技术论坛www.itpub.net的数据库管理版版主，

阅读更多技术文章和随笔可以登录我的个人 **blog**。

http://blog.dbform.com。

Oracle 诊断案例-Job 任务停止执行

本文作者: eygle (eygle@itpub.net)

摘要:

本文通过一次 Oracle Job 任务异常案例诊断，分析其原因及解决过程，从内部揭示 Oracle Job 任务调度及内部计时机制。

问题及环境

接到研发人员报告，数据库定时任务未正常执行，导致某些操作失败。

开始介入处理该事故.

系统环境:

```
SunOS DB 5.8 Generic_108528-21 sun4u sparc SUNW,Ultra-4

Oracle9i Enterprise Edition Release 9.2.0.3.0 - Production
```

解决过程

首先介入检查数据库任务

```
$ sqlplus "/ as sysdba"


SQL*Plus: Release 9.2.0.3.0 - Production on Wed Nov 17 20:23:53 2004


Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.




Connected to:

Oracle9i Enterprise Edition Release 9.2.0.3.0 - Production

With the Partitioning, OLAP and Oracle Data Mining options

JServer Release 9.2.0.3.0 - Production
```

```
SQL>  select  job,last_date,last_sec,next_date,next_sec,broken,failures  from
dba_jobs;


      JOB LAST_DATE LAST_SEC          NEXT_DATE NEXT_SEC          B   FAILURES
    INTERVAL
---------- --------- ---------------- --------- ---------------- - ----------
----------------------------

       31 16-NOV-04 01:00:02         17-NOV-04 01:00:00         N           0
    trunc(sysdate+1)+1/24

       27 16-NOV-04 00:00:04         17-NOV-04 00:00:00         N           0
    TRUNC(SYSDATE) + 1

       35 16-NOV-04 01:00:02         17-NOV-04 01:00:00         N           0
    trunc(sysdate+1)+1/24

       29 16-NOV-04 00:00:04         17-NOV-04 00:00:00         N           0
    TRUNC(SYSDATE) + 1

       30 01-NOV-04 06:00:01         01-DEC-04 06:00:00         N           0
    trunc(add_months(sysdate,1),'MM')+6/24

       65 16-NOV-04 04:00:03         17-NOV-04 04:00:00         N           0
    trunc(sysdate+1)+4/24

       46 16-NOV-04 02:14:27         17-NOV-04 02:14:27         N           0
    sysdate+1

       66 16-NOV-04 03:00:02         17-NOV-04 18:14:49         N           0
    trunc(sysdate+1)+3/24


8 rows selected.
```

发现 JOB 任务是都没有正常执行，最早一个应该在 17-NOV-04 01:00:00 执行。但是没有执行。

建立测试 JOB

```
create or replace PROCEDURE pining
```

```
  IS
BEGIN
       NULL;
 END;
/


variable jobno number;
variable instno number;
begin
  select instance_number into :instno from v$instance;
  dbms_job.submit(:jobno,        'pining;',        trunc(sysdate+1/288,'MI'),
'trunc(SYSDATE+1/288,''MI'')', TRUE, :instno);
end;
/
```

发现同样的，不执行。
但是通过 dbms_job.run(<job>)执行没有任何问题。

进行恢复尝试

怀疑是 CJQ0 进程失效，首先设置 JOB_QUEUE_PROCESSES 为 0，Oracle 会杀掉 CJQ0 及相应 job 进程

```
SQL> ALTER SYSTEM SET JOB_QUEUE_PROCESSES = 0;
```

等 2~3 分钟，重新设置

```
SQL> ALTER SYSTEM SET JOB_QUEUE_PROCESSES = 5;
```

此时 PMON 会重起 CJQ0 进程

```
Thu Nov 18 11:59:50 2004
```

```
ALTER SYSTEM SET job_queue_processes=0 SCOPE=MEMORY;

Thu Nov 18 12:01:30 2004

ALTER SYSTEM SET job_queue_processes=10 SCOPE=MEMORY;

Thu Nov 18 12:01:30 2004

Restarting dead background process CJQ0

CJQ0 started with pid=8
```

但是 Job 仍然不执行，而且在再次修改的时候，CJQ0 直接死掉了。

```
Thu Nov 18 13:52:05 2004

ALTER SYSTEM SET job_queue_processes=0 SCOPE=MEMORY;

Thu Nov 18 14:09:30 2004

ALTER SYSTEM SET job_queue_processes=10 SCOPE=MEMORY;

Thu Nov 18 14:10:27 2004

ALTER SYSTEM SET job_queue_processes=0 SCOPE=MEMORY;

Thu Nov 18 14:10:42 2004

ALTER SYSTEM SET job_queue_processes=10 SCOPE=MEMORY;

Thu Nov 18 14:31:07 2004

ALTER SYSTEM SET job_queue_processes=0 SCOPE=MEMORY;

Thu Nov 18 14:40:14 2004

ALTER SYSTEM SET job_queue_processes=10 SCOPE=MEMORY;

Thu Nov 18 14:40:28 2004

ALTER SYSTEM SET job_queue_processes=0 SCOPE=MEMORY;

Thu Nov 18 14:40:33 2004

ALTER SYSTEM SET job_queue_processes=1 SCOPE=MEMORY;

Thu Nov 18 14:40:40 2004

ALTER SYSTEM SET job_queue_processes=10 SCOPE=MEMORY;

Thu Nov 18 15:00:42 2004

ALTER SYSTEM SET job_queue_processes=0 SCOPE=MEMORY;
```

```
Thu Nov 18 15:01:36 2004

ALTER SYSTEM SET job_queue_processes=15 SCOPE=MEMORY;
```

尝试重起数据库

这个必须在晚上进行

```
PMON started with pid=2

DBW0 started with pid=3

LGWR started with pid=4

CKPT started with pid=5

SMON started with pid=6

RECO started with pid=7

CJQ0 started with pid=8

QMN0 started with pid=9

....
```

CJQ0 正常启动，但是 Job 仍然不执行。

没办法了...

继续研究...居然发现 Oralce 有这样一个 bug

```

1. Clear description of the problem encountered:

slgcsf() / slgcs() on Solaris will stop incrementing after

497 days 2 hrs 28 mins (approx) machine uptime.

```

```
2. Pertinent configuration information
No special configuration other than long machine uptime. .

3. Indication of the frequency and predictability of the problem
100% but only after 497 days.

4. Sequence of events leading to the problem
If the gethrtime() OS call returns a value > 42949672950000000
nanoseconds then slgcs() stays at 0xffffffff. This can
cause some problems in parts of the code which rely on
slgcs() to keep moving.
eg: In kkjssrh() does "now = slgcs(&se)" and compares that
to a previous timestamp. After 497 days uptime slgcs()
keeps returning 0xffffffff so "now - kkjlsrt" will
always return 0. .

5. Technical impact on the customer. Include persistent after effects.
In this case DBMS JOBS stopped running after 497 days uptime.
Other symptoms could occur in various places in the code.
```

好么，原来是计时器溢出了，一检查我的主机:

```
bash-2.03$ uptime
 10:00pm  up 500 day(s), 14:57,  1 user,  load average: 1.31, 1.09, 1.08
bash-2.03$ date
Fri Nov 19 22:00:14 CST 2004
```

刚好到事发时是 497 天多一点.ft.

安排重起主机系统..

这个问题够郁闷的，NND，谁曾想 Oracle 这都成...

Oracle 最后声称:

fix made it into 9.2.0.6 patchset

在 Solaris 上的 9206 尚未发布...晕.

好了，就当是个经历吧，如果有问题非常不可思议的话，那么大胆怀疑 Oracle 吧，是 Bug，可能就是 Bug。

重起以后问题解决，状态如下:

```
$ sqlplus "/ as sysdba"

SQL*Plus: Release 9.2.0.3.0 - Production on Fri Nov 26 09:21:21 2004

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.



Connected to:
Oracle9i Enterprise Edition Release 9.2.0.3.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.3.0 - Production

SQL> select job,last_date,last_sec,next_date,next_sec from user_jobs;
```

```
      JOB LAST_DATE LAST_SEC        NEXT_DATE NEXT_SEC
---------- --------- ---------------- --------- ----------------
       70 26-NOV-04 09:21:04        26-NOV-04 09:26:00



SQL> /

      JOB LAST_DATE LAST_SEC        NEXT_DATE NEXT_SEC
---------- --------- ---------------- --------- ----------------
       70 26-NOV-04 09:26:01        26-NOV-04 09:31:00

SQL>
SQL> select * from v$timer;

    HSECS
----------
   3388153

SQL> select * from v$timer;

    HSECS
----------
   3388319

SQL>
```

FAQ

一些朋友在 Pub 上问的问题

Q:对于不同平台，是否存在同样的问题?

A:对于不同平台，存在同样的问题

因为 Oracle 使用了标准 C 函数 gethrtime

参考:

http://www.eygle.com/unix/Man.Page.Of.gethrtime.htm

使用了该函数的代码都会存在问题.

在 Metalink Note:3427424.8 文档中，Oracle 定义的平台影响为:Generic (all / most platforms affected)

Q.计数器溢出，看了看 job 中基本都是 1 天左右执行一次，如果设置 3 天执行一次的 job，是否出问题的 uptime 应该是 497*3 之后呢 ？

A:不会

Oracle 内部通过计时器来增进相对时间.

由于 Oracle 内部 hrtime_t 使用了 32 位计数

那么最大值也就是 0xffffffff

0xffffffff = 4294967295

slgcs()是 10 亿分之一秒，溢出在 42949672950000000 这个点上.

注意，这里 0xffffffff，达到这个值时，本来是无符号整型，现在变成了-1，那么这个值递增时，+1 = 0 了。

时间就此停住了。

我写了一小段代码来验证这个内容，参考：

```
[oracle@jumper oracle]$ cat unsign.c

#include

int main(void){

unsigned int num = 0xffffffff;


printf("num is %d bits long\n", sizeof(num) * 8);

printf("num = 0x%x\n", num);
```

```
printf("num + 1 = 0x%x\n", num + 1);


return 0;

}


[oracle@jumper oracle]$ gcc -o unsign.sh unsign.c

[oracle@jumper oracle]$ ./unsign.sh

num is 32 bits long

num = 0xffffffff

num + 1 = 0x0

[oracle@jumper oracle]$
```

Q:内部时钟之一应该就是这个吧： v$timer 精确到 1/100 秒的数据

A:没错!

注意前面说的:

```
4. Sequence of events leading to the problem

If the gethrtime() OS call returns a value > 42949672950000000

nanoseconds then slgcs() stays at 0xffffffff. This can

cause some problems in parts of the code which rely on

slgcs() to keep moving.
```

也就是说如果 gethrtime() 操作系统调用返回值大于 42949672950000000（单位 10 亿分之一秒）

也就是说 Oracle 将得到一个 cs 值为 4294967295 的时间值

而 4294967295 值就是 0xffffffff

所以当时 v$timer 的计时也就是:

```
SQL> select * from v$timer;

    HSECS
----------
4294967295

SQL> /

    HSECS
----------
4294967295

SQL> /

    HSECS
----------
4294967295

SQL>
```

作者简介:

盖国强，网名 eygle

曾任 ITPUB MS 版版主,现任 itpub Oracle 管理版版主.

曾任职于某国家大型企业,服务于烟草行业,开发过基于 Oracle 数据库的大型 ERP 系统,属国家信息产业部重点工程.同时负责 Oracle 数据库管理及优化,并为多家烟草企业提供 Oracle 数据库管理、优化及技术支持.

目前任职于北京某电信增值业务系统提供商企业,首席 DBA,负责数据库业务.管理全国 30 多个数据库系统。项目经验丰富,曾设计规划及支持中国联通增值业务等大型数据库系统.

实践经验丰富，长于数据库诊断、性能调整与 SQL 优化等。对于 Oracle 内部技术具有深入研究。

高级培训讲师,培训经验丰富,曾主讲 itpub dba 培训及 itpub 高级性能调整等主要课程.

《Oracle 数据库 DBA 专题技术精粹》一书的主编及主要作者.

你可以在http://www.eygle.com上找到关于作者的更多信息.

一条 sql 导致数据库整体性能下降的诊断和解决的全过程

本文作者:biti_rainy (biti_rainy@itpub.net)

摘要:

本文介绍了数据库出现性能问题后分析诊断以及解决的整个流程，在这个过程中使用了 statspack report，着重从等待事件、sql、10053 event 入手，并利用了 standby database 来做对比分析从而找到问题的根本。

一、现象

今天早上一来，数据库 load 就比往常高了许多。想想数据库唯一的变化是昨天早上我曾经重新分析过数据库对象。发现数据库 load 很高，首先看 top 发现没有特别异常的进程，在数据库中适时抓取正在运行的 sql 也没发现异常(通常运行时间非常短的 sql 是不能被抓取到的)。询问相关应用程序人员，最近没有变动。检查应用程序日志发现今天早上跟往常也没有过多登陆和操作。基本上可以圈定是在数据库服务器本身上面。

二、诊断与解决

由于这时候我还没有办法确定到底是哪个应用的哪个查询的问题，因为数百个进程的几十台 server 连着，我不能去及时的追踪。打算等到 10 点过去后，抽取 8/9/10 高峰期的整点的 statspack 出来，跟上星期的这个时间产生的报告对比看看。

通过对比报告我们发现 CPU TIME 今天一小时内增加了大约 1200 秒（2,341 - 1,175）。这是一个重大的变化，很显然是两种可能

1：今天过多地执行了某些 sql

2：某些 sql 的执行计划发生变化导致 cpu 使用过多

statspack report 对比 ：

```
report I:

Event                          Waits    Time (s) Ela Time

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

CPU time                                2,341    42.60

db file sequential read             387,534      2,255    41.04
```

```
global cache cr request       745,170        231     4.21
log file sync              98,041        229     4.17
log file parallel write        96,264        158     2.88


report  II:
Event                   Waits      Time (s) Ela Time
--------------------------------------------- ------------ ----------- --------
db file sequential read       346,851      1,606    47.60
CPU time                           1,175    34.83
global cache cr request      731,368        206     6.10
log file sync              90,556        91     2.71
db file scattered read        37,746         90     2.66
```

接下来我对比了 sql 部分内容，发现

```
Buffer Gets    Executions  Gets per Exec  %Total Time (s)  Time (s) Hash Value
--------------- ------------ -------------- ------ -------- ---------
17,899,606      47,667         375.5   55.6  1161.27   1170.22 3481369999
Module: /home/oracle/AlitalkSrv/config/../../AlitalkSrv/
SELECT login_id, to_char(gmt_create, 'YYYY-MM-DD HH24:MI:SS')
  from IM_BlackList     where black_id = :b1
```

这条 sql 出现在了今天报告的前列,而以往的报告中该 sql 根本不排在 buffer gets 前面位置，显然这条 sql 消耗了大约 1161.27 秒 cpu time .检查原来的报告

```
Buffer Gets    Executions  Gets per Exec  %Total Time (s)  Time (s) Hash Value
--------------- ------------ -------------- ------ -------- --------- ----------
107,937      47,128          2.3    0.8     7.39      6.94 3481369999
Module: /home/oracle/AlitalkSrv/config/../../AlitalkSrv/
SELECT login_id, to_char(gmt_create, 'YYYY-MM-DD HH24:MI:SS')
  from IM_BlackList     where black_id = :b1
```

我们发现只消耗了 7.39 秒的 cpu time 。到这个时候我基本可以断定，是由于这个 sql 没有走索引而走了全表扫描。但是为什么会走全表扫描呢，这是一个问题，接下来我检查了表的索引：

```
SQL> select  index_name,column_name from  user_ind_columns where  table_name =
'IM_BLACKLIST';


IM_BLACKLIST_PK             LOGIN_ID

IM_BLACKLIST_PK             BLACK_ID

IM_BLACKLIST_LID_IND         BLACK_ID
```

很显然存在着 black_id 的单独的索引，应该正常使用才对。于是我在生产库上执行这个 sql 一看，却发现走了全表扫描。为此我到一个前 2 天的的 standby 的 opren read only 的数据库上查询了一下该索引字段的 histogram（这个时候昨天早上分析对象的日志还没有被应用过去）

```
sys@OCN>   select     COLUMN_NAME   ,ENDPOINT_NUMBER,   ENDPOINT_VALUE     ,
ENDPOINT_ACTUAL_VALUE  from  dba_histograms
  2   where  table_name =  'IM_BLACKLIST' and  column_name = 'BLACK_ID';


COLUMN_NAME                                ENDPOINT_NUMBER  ENDPOINT_VALUE
ENDPOINT_ACTUAL_VALUE
------------------------------         ---------------         --------------
------------------------------
BLACK_ID                                 0    2.5031E+35

BLACK_ID                                 1    2.6065E+35

BLACK_ID                                 2    2.8661E+35

BLACK_ID                                 3    5.0579E+35

BLACK_ID                                 4    5.0585E+35

BLACK_ID                                 5    5.0585E+35

BLACK_ID                                 6    5.0589E+35

BLACK_ID                                 7    5.0601E+35

BLACK_ID                                 8    5.1082E+35
```

```
BLACK_ID                          9    5.1119E+35
BLACK_ID                         10    5.1615E+35
BLACK_ID                         11    5.1616E+35
BLACK_ID                         12    5.1628E+35
BLACK_ID                         13    5.1646E+35
BLACK_ID                         14    5.2121E+35
BLACK_ID                         15    5.2133E+35
BLACK_ID                         16    5.2155E+35
BLACK_ID                         17    5.2662E+35
BLACK_ID                         18    5.3169E+35
BLACK_ID                         19    5.3193E+35
BLACK_ID                         20    5.3686E+35
BLACK_ID                         21    5.3719E+35
BLACK_ID                         22    5.4198E+35
BLACK_ID                         23    5.4206E+35
BLACK_ID                         24    5.4214E+35
BLACK_ID                         25    5.4224E+35
BLACK_ID                         26    5.4238E+35
BLACK_ID                         27    5.4246E+35
BLACK_ID                         28    5.4743E+35
BLACK_ID                         29    5.5244E+35
BLACK_ID                         30    5.5252E+35
BLACK_ID                         31    5.5252E+35
BLACK_ID                         32    5.5272E+35
BLACK_ID                         33    5.5277E+35
BLACK_ID                         34    5.5285E+35
BLACK_ID                         35    5.5763E+35
BLACK_ID                         36    5.6274E+35
BLACK_ID                         37    5.6291E+35
BLACK_ID                         38    5.6291E+35
```

```
BLACK_ID                        39    5.6291E+35
BLACK_ID                        40    5.6291E+35
BLACK_ID                        42    5.6311E+35
BLACK_ID                        43    5.6794E+35
BLACK_ID                        44    5.6810E+35
BLACK_ID                        45    5.6842E+35
BLACK_ID                        46    5.7351E+35
BLACK_ID                        47    5.8359E+35
BLACK_ID                        48    5.8887E+35
BLACK_ID                        49    5.8921E+35
BLACK_ID                        50    5.9430E+35
BLACK_ID                        51    5.9913E+35
BLACK_ID                        52    5.9923E+35
BLACK_ID                        53    5.9923E+35
BLACK_ID                        54    5.9931E+35
BLACK_ID                        55    5.9947E+35
BLACK_ID                        56    5.9959E+35
BLACK_ID                        57    6.0428E+35
BLACK_ID                        58    6.0457E+35
BLACK_ID                        59    6.0477E+35
BLACK_ID                        60    6.0479E+35
BLACK_ID                        61    6.1986E+35
BLACK_ID                        62    6.1986E+35
BLACK_ID                        63    6.1994E+35
BLACK_ID                        64    6.2024E+35
BLACK_ID                        65    6.2037E+35
BLACK_ID                        66    6.2521E+35
BLACK_ID                        67    6.2546E+35
BLACK_ID                        68    6.3033E+35
BLACK_ID                        69    6.3053E+35
```

```
BLACK_ID                                70    6.3069E+35

BLACK_ID                                71    6.3553E+35

BLACK_ID                                72    6.3558E+35

BLACK_ID                                73    6.3562E+35

BLACK_ID                                74    6.3580E+35

BLACK_ID                                75    1.1051E+36
```

然后对比了一下当前的 histograms

```
COLUMN_NAME         ENDPOINT_NUMBER ENDPOINT_VALUE ENDPOINT_ACTUAL_VALUE

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

BLACK_ID                                 0    1.6715E+35

BLACK_ID                                 1    2.5558E+35

BLACK_ID                                 2    2.7619E+35

BLACK_ID                                 3    2.9185E+35

BLACK_ID                                 4    5.0579E+35

BLACK_ID                                 5    5.0589E+35

BLACK_ID                                 6    5.0601E+35

BLACK_ID                                 7    5.1100E+35

BLACK_ID                                 8    5.1601E+35

BLACK_ID                                 9    5.1615E+35

BLACK_ID                                10    5.1624E+35

BLACK_ID                                11    5.1628E+35

BLACK_ID                                12    5.1642E+35

BLACK_ID                                13    5.2121E+35

BLACK_ID                                14    5.2131E+35

BLACK_ID                                15    5.2155E+35

BLACK_ID                                16    5.2676E+35

BLACK_ID                                17    5.3175E+35

BLACK_ID                                18    5.3684E+35
```

```
BLACK_ID                        19    5.3727E+35
BLACK_ID                        20    5.4197E+35
BLACK_ID                        21    5.4200E+35
BLACK_ID                        22    5.4217E+35
BLACK_ID                        23    5.4238E+35
BLACK_ID                        24    5.4244E+35
BLACK_ID                        25    5.4755E+35
BLACK_ID                        26    5.5252E+35
BLACK_ID                        27    5.5252E+35
BLACK_ID                        28    5.5252E+35
BLACK_ID                        29    5.5283E+35
BLACK_ID                        30    5.5771E+35
BLACK_ID                        31    5.6282E+35
BLACK_ID                        32    5.6291E+35
BLACK_ID                        33    5.6291E+35
BLACK_ID                        34    5.6291E+35
BLACK_ID                        35    5.6299E+35
BLACK_ID                        36    5.6315E+35
BLACK_ID                        37    5.6794E+35
BLACK_ID                        39    5.6816E+35
BLACK_ID                        40    5.6842E+35
BLACK_ID                        41    5.7838E+35
BLACK_ID                        42    5.8877E+35
BLACK_ID                        43    5.8917E+35
BLACK_ID                        44    5.9406E+35
BLACK_ID                        45    5.9909E+35
BLACK_ID                        46    5.9923E+35
BLACK_ID                        47    5.9923E+35
BLACK_ID                        48    5.9946E+35
BLACK_ID                        49    5.9950E+35
```

```
BLACK_ID                        50    5.9960E+35
BLACK_ID                        51    5.9960E+35
BLACK_ID                        52    5.9960E+35
BLACK_ID                        53    5.9960E+35
BLACK_ID                        54    5.9960E+35
BLACK_ID                        55    5.9960E+35
BLACK_ID                        56    5.9960E+35
BLACK_ID                        57    6.0436E+35
BLACK_ID                        58    6.0451E+35
BLACK_ID                        59    6.0471E+35
BLACK_ID                        60    6.1986E+35
BLACK_ID                        61    6.1998E+35
BLACK_ID                        62    6.2014E+35
BLACK_ID                        63    6.2037E+35
BLACK_ID                        64    6.2521E+35
BLACK_ID                        65    6.2544E+35
BLACK_ID                        66    6.3024E+35
BLACK_ID                        67    6.3041E+35
BLACK_ID                        68    6.3053E+35
BLACK_ID                        69    6.3073E+35
BLACK_ID                        70    6.3558E+35
BLACK_ID                        71    6.3558E+35
BLACK_ID                        72    6.3558E+35
BLACK_ID                        73    6.3558E+35
BLACK_ID                        74    6.3580E+35
BLACK_ID                        75    1.1160E+36
```

我们发现原来的 histograms 值分布比较均匀，而昨天分析后的值分布就有一些地方是集中的，参考上面红色部分。于是我再做了个 10053 dump 对比，在分析之前的 standby database 上,执行

```
alter session set events '10053 trace name context forever';
```

然后执行相关的 sql 再去看 trace 文件

```
Table stats   Table: IM_BLACKLIST   Alias: IM_BLACKLIST
  TOTAL ::  CDN: 57477  NBLKS:  374  AVG_ROW_LEN:  38
-- Index stats
  INDEX NAME: IM_BLACKLIST_LID_IND  COL#: 2
    TOTAL ::  LVLS: 1   #LB: 219  #DK: 17181  LB/K: 1  DB/K: 2  CLUF: 44331
  INDEX NAME: IM_BLACKLIST_PK  COL#: 1 2
    TOTAL ::  LVLS: 1   #LB: 304  #DK: 57477  LB/K: 1  DB/K: 1  CLUF: 55141
_OPTIMIZER_PERCENT_PARALLEL = 0
**************************************
SINGLE TABLE ACCESS PATH
Column:   BLACK_ID  Col#: 2     Table: IM_BLACKLIST   Alias: IM_BLACKLIST
    NDV: 17181    NULLS: 0        DENS: 5.8204e-05
    NO HISTOGRAM: #BKT: 1 #VAL: 2
  TABLE: IM_BLACKLIST    ORIG CDN: 57477  ROUNDED CDN: 3  CMPTD CDN: 3
  Access path: tsc  Resc:  38  Resp:  38
  Access path: index (equal)
      Index: IM_BLACKLIST_LID_IND
  TABLE: IM_BLACKLIST
      RSC_CPU: 0   RSC_IO: 4
  IX_SEL:  0.0000e+00  TB_SEL:  5.8204e-05
  Skip scan: ss-sel 0  andv 27259
    ss cost 27259
    table io scan cost 38
  Access path: index (no sta/stp keys)
      Index: IM_BLACKLIST_PK
  TABLE: IM_BLACKLIST
```

```
    RSC_CPU: 0   RSC_IO: 309
  IX_SEL:  1.0000e+00  TB_SEL:  5.8204e-05
  BEST_CST: 4.00  PATH: 4  Degree:  1
***************************************
OPTIMIZER STATISTICS AND COMPUTATIONS
***************************************
GENERAL PLANS
**********************
Join order[1]: IM_BLACKLIST [IM_BLACKLIST]
Best so far: TABLE#: 0  CST:        4  CDN:        3  BYTES:       75
Final:
  CST: 4  CDN: 3  RSC: 4  RSP: 4  BYTES: 75
  IO-RSC: 4  IO-RSP: 4  CPU-RSC: 0  CPU-RSP: 0
```

在分析之后的当前数据库我再做一个 10053 trace ， 得到如下内容

```
SINGLE TABLE ACCESS PATH
Column:   BLACK_ID  Col#: 2     Table: IM_BLACKLIST   Alias: IM_BLACKLIST
    NDV: 17069    NULLS: 0        DENS: 1.4470e-03
    HEIGHT BALANCED HISTOGRAM: #BKT: 75 #VAL: 75
  TABLE: IM_BLACKLIST    ORIG CDN: 57267  ROUNDED CDN: 83  CMPTD CDN: 83
  Access path: tsc  Resc:  38  Resp:  38
  Access path: index (equal)
     Index: IM_BLACKLIST_LID_IND
  TABLE: IM_BLACKLIST
     RSC_CPU: 0   RSC_IO: 65
  IX_SEL:  0.0000e+00  TB_SEL:  1.4470e-03
  Skip scan: ss-sel 0  andv 27151
    ss cost 27151
    table io scan cost 38
```

```
  Access path: index (no sta/stp keys)

     Index: IM_BLACKLIST_PK

  TABLE: IM_BLACKLIST

     RSC_CPU: 0   RSC_IO: 384

  IX_SEL:  1.0000e+00  TB_SEL:  1.4470e-03

  BEST_CST: 38.00  PATH: 2  Degree:  1
***************************************

OPTIMIZER STATISTICS AND COMPUTATIONS

***************************************

GENERAL PLANS

**********************

Join order[1]: IM_BLACKLIST [IM_BLACKLIST]

Best so far: TABLE#: 0  CST:        38  CDN:        83  BYTES:      2407

Final:

  CST: 38  CDN: 83  RSC: 38  RSP: 38  BYTES: 2407

  IO-RSC: 38  IO-RSP: 38  CPU-RSC: 0  CPU-RSP: 0
```

注意上面红色部分，我发现分析之前之后全表扫描 cost 都是 38，但是分析之后的根据索引扫描却成为了 65 ，而分析之前是 4。很显然是由于这个查询导致昨天早上分析之后走了全表扫描。于是我再对表进行了分析，只不过这次我没有分析索引字段，而是

```
analyze table im_blacklist compute  statistics;
```

这样之后，dbms_histograms 中信息只剩下

```
COLUMN_NAME          ENDPOINT_NUMBER ENDPOINT_VALUE ENDPOINT_ACTUAL_VALUE

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

GMT_CREATE                                0    2452842.68

GMT_MODIFIED                              0    2452842.68

LOGIN_ID                                  0    2.5021E+35
```

```
BLACK_ID                              0    1.6715E+35
GMT_CREATE                            1     2453269.44
GMT_MODIFIED                          1     2453269.44
LOGIN_ID                              1    6.3594E+35
BLACK_ID                              1    1.1160E+36
```

再执行该 sql，就走了索引，从而使得数据库的 load 降了下来。

分析这整个过程中，我无法知道 oracle 的走索引 cost 65 是怎么计算出来的，当然是跟 histograms 有关，但计算方法我却是不清楚的。

这条 sql 是 bind var，但是却走了全表扫描，这是由于 920 中数据库在对 bind var 的 sql 进行第一次解析的时候去 histograms 中窥视了数据分布从而根据 cost 选择了 FTS。然后后面继续执行的 sql 呢，则不论是否该走索引，都走了 FTS。这是 920 这个版本的特性的弊病。也就是说，这有偶然性因素的存在。 但是对于这个表，我做了分析(不分析索引字段)之后不存在 histograms，则 sql 无论如何都走了索引扫描。

作者简介:

网名 biti_rainy

CSDN eMag Oracle 电子杂志主编。

开发出身，对数据库应用设计中如何正确地应用 oracle 特性以扬长避短具有深刻理解。

有丰富的 oracle 实践经验，对数据库的体系结构、备份恢复、sql 优化、数据库整体性能优化、oracle internal 都有深入研究。

曾于某电信集成公司负责计费系统的开发，然后成为某系统集成公司的 DBA，再辗转在香港一家跨国公司珠海研发中心担任技术负责人(公司主要产品就是 sql 与数据库优化工具,产品主要销往欧洲和北美),此后成为自由职业者，为客户提供独立的 oracle 数据库的技术服务和高级性能调整等培训，同时提供 ITPUB 华南和华东的培训。

目前服务于国内某大型电子商务网站，维护系统数据库并提供开发支持.

个人Blog等
http://blog.csdn.net/biti_rainy
http://blog.itpub.net/biti_rainy

如何使用触发器实现数据库级守护

本文作者: eygle (eygle.com@gmail.com)

摘要:

对于重要对象,实施 DDL 拒绝,防止 create,drop,truncate,alter 等重要操作.

不管是有意还是无意的,你可能会遇到数据库中重要的数据表等对象被 drop 掉的情况,这可能会给我们带来巨大的损失.

通过触发器，我们可以实现对于表等对象的数据库级守护，禁止用户 drop 操作.

以下是一个简单的范例，供参考:

```
REM this script can be used to monitor a object
REM deny any drop operation on it.
CREATE OR REPLACE TRIGGER trg_dropdeny
   BEFORE DROP ON DATABASE
BEGIN
   IF LOWER (ora_dict_obj_name ()) = 'test'
   THEN
      raise_application_error (num     => -20000,
                           msg     =>    '你疯了,想删除表 '
                                     || ora_dict_obj_name ()
                                     || ' ?!!!!!'
                                     || '你完了，警察已在途中.....'
                           );
   END IF;
END;
/
```

测试效果:

```
SQL> connect scott/tiger
Connected.
SQL> create table test as select * from dba_users;

Table created.

SQL> connect / as sysdba
Connected.
SQL> create or replace trigger trg_dropdeny
  2    before drop on database
  3  begin
  4       if lower(ora_dict_obj_name()) = 'test'
  5       then
  6       raise_application_error(
  7          num => -20000,
  8          msg => '你疯了,想删除表 ' || ora_dict_obj_name() || ' ?!!!!!' ||'你完
了，警察已在途中.....');
  9       end if;
 10    end;
 11  /

Trigger created.

SQL> connect scott/tiger
Connected.
SQL> drop table test;
drop table test
*
ERROR at line 1:
```

```
ORA-00604: error occurred at recursive SQL level 1

ORA-20000: 你疯了,想删除表 TEST ?!!!!!你完了，警察已在途中.....

ORA-06512: at line 4
```

Oracle 从 Oracle8i 开始,允许实施 DDL 事件 trigger，可是实现对于 DDL 的监视及控制,以下是一个进一步的例子:

```
create or replace trigger ddl_deny

before create or alter or drop or truncate on database

declare

  l_errmsg varchar2(100):= 'You have no permission to this operation';

begin

  if ora_sysevent = 'CREATE' then

    raise_application_error(-20001,    ora_dict_obj_owner    ||    '.'    ||
ora_dict_obj_name || ' ' || l_errmsg);

  elsif ora_sysevent = 'ALTER' then

    raise_application_error(-20001,    ora_dict_obj_owner    ||    '.'    ||
ora_dict_obj_name || ' ' || l_errmsg);

  elsif ora_sysevent = 'DROP' then

    raise_application_error(-20001,    ora_dict_obj_owner    ||    '.'    ||
ora_dict_obj_name || ' ' || l_errmsg);

  elsif ora_sysevent = 'TRUNCATE' then

    raise_application_error(-20001,    ora_dict_obj_owner    ||    '.'    ||
ora_dict_obj_name || ' ' || l_errmsg);

  end if;


exception

  when no_data_found then

    null;

end;

/
```

我们看一下效果:

```
[oracle@jumper tools]$ sqlplus "/ as sysdba"

SQL*Plus: Release 9.2.0.4.0 - Production on Sun Oct 31 11:38:25 2004


Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.



Connected to:

Oracle9i Enterprise Edition Release 9.2.0.4.0 - Production

With the Partitioning option

JServer Release 9.2.0.4.0 - Production


SQL> set echo on

SQL> @ddlt

SQL> create or replace trigger ddl_deny

2 before create or alter or drop or truncate on database

3 declare

4 l_errmsg varchar2(100):= 'You have no permission to this operation';

5 begin

6 if ora_sysevent = 'CREATE' then

7 raise_application_error(-20001, ora_dict_obj_owner || '.' || ora_dict_obj_name
|| ' ' || l_errmsg);

8 elsif ora_sysevent = 'ALTER' then

9 raise_application_error(-20001, ora_dict_obj_owner || '.' || ora_dict_obj_name
|| ' ' || l_errmsg);

10 elsif ora_sysevent = 'DROP' then

11    raise_application_error(-20001,    ora_dict_obj_owner    ||    '.'    ||
ora_dict_obj_name || ' ' || l_errmsg);

12 elsif ora_sysevent = 'TRUNCATE' then
```

```
13    raise_application_error(-20001,    ora_dict_obj_owner    ||    '.'    ||
ora_dict_obj_name || ' ' || l_errmsg);

14 end if;

15

16 exception

17 when no_data_found then

18 null;

19 end;

20 /


Trigger created.


SQL>

SQL>

SQL> connect scott/tiger

Connected.

SQL> create table t as select * from test;

create table t as select * from test

*

ERROR at line 1:

ORA-00604: error occurred at recursive SQL level 1

ORA-20001: SCOTT.T You have no permission to this operation

ORA-06512: at line 5



SQL> alter table test add (id number);

alter table test add (id number)

*

ERROR at line 1:

ORA-00604: error occurred at recursive SQL level 1
```

```
ORA-20001: SCOTT.TEST You have no permission to this operation

ORA-06512: at line 7



SQL> drop table test;

drop table test

*

ERROR at line 1:

ORA-00604: error occurred at recursive SQL level 1

ORA-20001: SCOTT.TEST You have no permission to this operation

ORA-06512: at line 9



SQL> truncate table test;

truncate table test

*

ERROR at line 1:

ORA-00604: error occurred at recursive SQL level 1

ORA-20001: SCOTT.TEST You have no permission to this operation

ORA-06512: at line 11
```

我们可以看到,ddl 语句都被禁止了,如果你不是禁止,可以选择把执行这些操作的用户及时间记录到另外的临时表中.以备查询.

作者简介:

盖国强，网名 eygle

曾任 ITPUB MS 版版主,现任 itpub Oracle 管理版版主.

曾任职于某国家大型企业,服务于烟草行业,开发过基于 Oracle 数据库的大型 ERP 系统,属国家信息产业部重点工程.同时负责 Oracle 数据库管理及优化,并为多家烟草企业提供 Oracle 数据库管理、优化及技术支持.

目前任职于北京某电信增值业务系统提供商企业,首席 DBA,负责数据库业务.管理全国 30 多个数据库系统。项目经验丰富,曾设计规划及支持中国联通增值业务等大型数据库系统.

实践经验丰富，长于数据库诊断、性能调整与 SQL 优化等。对于 Oracle 内部技术具有深入研究。

高级培训讲师,培训经验丰富,曾主讲 itpub dba 培训及 itpub 高级性能调整等主要课程.

《Oracle 数据库 DBA 专题技术精粹》一书的主编及主要作者.

你可以在http://www.eygle.com上找到关于作者的更多信息.

DBMS_ROWID 包的使用

本文作者: coolyl (allan@itpub.net)

摘要:

DBMS_ROWID 是一个比较有用的系统自带的 package，主要可以用来处理坏块的问题，于是仔细的研究了一下，这个包可以用来了解 file、block、object id 和 rowid 之间的关系，在 Oracle8 中被引用进来，Oracle7 不支持这个包。这个包的定义可以在 dbmsutil.sql 中找到，在 catproc.sql 中被调用，并被给予 public 执行权限.

首先来了解一下这个包中使用的常量：

ROWID 类型：

```
rowid_type_restricted   RESTRICTED - Restricted ROWID

rowid_type_extended     EXTENDED   - Extended ROWID
```

ROWID 验证结果：

```
rowid_is_valid          VALID   - Valid ROWID

rowid_is_invalid        INVALID - Invalid ROWID
```

目标类型：

```
rowid_object_undefined UNDEFINED - Object Number not defined
(for restricted ROWIDs)
```

ROWID 转换类型：

```
rowid_convert_internal  INTERNAL - convert to/from column of ROWID type

rowid_convert_external  EXTERNAL - convert to/from string format
```

意外错误：

```
ROWID_INVALID          invalid rowid format

ROWID_BAD_BLOCK         block is beyond end of file
```

在 DBMS_ROWID 这个包里面可以使用下面的功能：

```
function ROWID_CREATE(rowid_type    IN number,
                      object_number IN number,
                      relative_fno  IN number,
                      block_number  IN number,
                      row_number    IN number)
```

```
                  return ROWID;
-- rowid_type     - 类型(restricted=0/extended=1)
-- object_number   - 对象号
-- relative_fno    - relative file number
-- block_number    - 文件包含的 block 号
-- row_number     - block 中的行的行号
```

下面具体的讨论一下 DBMS_ROWID 包的用法

1.1 DBMS_ROWID.ROWID_BLOCK_NUMBER

返回一个 rowid 的 block 号

定义如下：

```
function dbms_rowid.rowid_block_number
(row_id in rowid)
return number
```

例子：

```
SQL> select dbms_rowid.rowid_block_number(rowid) "block" from test;
    block
----------
    23722
```

1.2 DBMS_ROWID.ROWID_CREATE

创建并返回一个基于单独行的 rowid，创建的 rowid 类型是 RESTRICTED 或者是 EXTENDED，这种功能一般都是用于测试目的，因为只有 oracle 才能创建一个合法的 rowid 指向数据。

定义如下：

```
function dbms_rowid.rowid_create
(rowid_type in number
,object_number in number
,relative_fno in number
```

```
,block_number in number
,row_number in number)
return rowid
```

例子：

创建一个 restricted rowid：

```
SQL> select dbms_rowid.rowid_create(0, 6877,1,23722,0) from dual;
DBMS_ROWID.ROWID_C
------------------
00005CAA.0000.0001
```

创建一个 extended rowid：

```
SQL> select dbms_rowid.rowid_create(1, 6877,1,23722,0) from dual;
DBMS_ROWID.ROWID_C
------------------
AAABrdAABAAAFyqAAAa
```

1.3 DBMS_ROWID.ROWID_INFO

返回一个单独组件的一个指定的 rowid，它只能用于 PL/SQL,而不能用于 sql 语句中。

定义如下：

```
procedure dbms_rowid.rowid_info
(rowid_in in rowid
,rowid_type out number
,object_number out number
,relative_fno out number
,block_number out number
,row_number out number)
```

例子：

```
SQL> set serverout on
SQL> set echo on
SQL> declare
```

```
 2  my_rowid rowid;
 3  rowid_type number;
 4  object_number number;
 5  relative_fno number;
 6  block_number number;
 7  row_number number;
 8  begin
 9  my_rowid :=dbms_rowid.rowid_create(1, 6877,1,23722,0);
10  dbms_rowid.rowid_info(my_rowid, rowid_type, object_number,
11  relative_fno, block_number, row_number);
12  dbms_output.put_line('ROWID:   ' || my_rowid);
13  dbms_output.put_line('Object#:     ' || object_number);
14  dbms_output.put_line('RelFile#:    ' || relative_fno);
15  dbms_output.put_line('Block#:      ' || block_number);
16  dbms_output.put_line('Row#:        ' || row_number);
17  end;
18  /
ROWID:   AAABrdAABAAAFyqAAA
Object#:     6877
RelFile#:    1
Block#:      23722
Row#:        0
PL/SQL 过程已成功完成。
```

1.4 DBMS_ROWID.ROWID_OBJECT

返回一个 rowid 的对象号。如果是 restricted 的 rowid，则返回 0。

定义如下：

```
function dbms_rowid.rowid_object
```

```
(row_id in rowid)
return number
```

例子：

```
SQL> select dbms_rowid.rowid_object(rowid) "OBJECT" from test;
   OBJECT
----------
     6877
SQL>select
dbms_rowid.rowid_object(dbms_rowid.rowid_to_restricted(rowid,0))
" OBJECT " from test;
   OBJECT
----------
       0
```

1.5 DBMS_ROWID.ROWID_RELATIVE_FNO

返回一个 rowid 的相对文件号。

定义如下：

```
function dbms_rowid.rowid_relative_fno
(row_id in rowid)
return number
```

例子：

```
SQL> select dbms_rowid.rowid_relative_fno(rowid) "relative fno"
from test;
relative fno
------------
         1
```

1.6 DBMS_ROWID.ROWID_ROW_NUMBER

返回一个 rowid 的行号。（从零开始）

定义如下：

```
function dbms_rowid.rowid_row_number

(row_id in rowid)

return number

例子：

SQL> select dbms_rowid.rowid_row_number(rowid) "row" from test;

      row

----------

         0
```

1.7 DBMS_ROWID.ROWID_TO_ABSOLUTE_FNO

返回一个 rowid 的完全文件号。

定义如下：

```
function dbms_rowid.rowid_to_absolute_fno

(rowid in rowid

,schema_name in varchar2

,object_name in varchar2)

return number
```

例子：

```
SQL> select dbms_rowid.rowid_to_absolute_fno (rowid, 'SYS', 'TEST')

"absolute

fno" from test;

absolute fno

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

           1
```

1.8 DBMS_ROWID.ROWID_TO_EXTENDED

转换一个 restricted rowid 为一个 extended rowid.如果原始的 rowid 存储在列中，转

换的 就是 internal 类型；如果原始的 rowid 是以字符串形式存储的，那转换的就是 external 类型。

定义如下：

```
function dbms_rowid.rowid_to_extended

(old_rowid in rowid

,schema_name in varchar2

,object_name in varchar2

,conversion_type in integer)

return rowid
```

例子：

转换 restricted internal rowid 为 extended 格式

```
SQL>select dbms_rowid.rowid_to_extended

(dbms_rowid.rowid_to_restricted(rowid,0),'SYS','TEST',0)

"extended rowid" from

test;

extended rowid

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

AAABrdAABAAAFyqAAA
```

转换 restricted external rowid 为 extended 格式

```
SQL> select dbms_rowid.rowid_to_extended

('00005CAA.0000.0001','SYS','TEST',1) from dual;

DBMS_ROWID.ROWID_T

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

AAABrdAABAAAFyqAAA
```

如果参数中的 SCHEMA 和 OBJECT 为 null，则默认是当前的对象

```
SQL>select dbms_rowid.rowid_to_extended

(dbms_rowid.rowid_to_restricted(rowid,0),null,null,0) "extended

rowid" from test;

extended rowid

------------------
```

```
AAABrdAABAAAFyqAAA
```

1.9 DBMS_ROWID.ROWID_TO_RESTRICTED

转换一个 exteneded 的 rowid 为一个 restricted 的 rowid，restricted 的 rowid 格式为 BBBBBBB.RRRR.FFFFF, BBBBBBB 代表 block，RRRR 代表在 block 中的行号，从 0 开始，FFFFF 代表文件号。这个包可以使用 rowid 或者 rowid 转换类型（ROWID_CONVERT_INTERNAL (0)和 ROWID_CONVERT_EXTERNAL (1)）。

定义如下：

```
function dbms_rowid.rowid_to_restricted
(old_rowid in rowid
,conversion_type in integer)
return rowed
```

例子：

```
SQL> select dbms_rowid.rowid_to_restricted(rowid, 1) "restricted
rowid" from test;
restricted rowid
------------------
00005CAA.0000.0001
Block 计算：5*16*16*16+C*16*16+A*16+A=20480+3072+160+10=23722
```

1.10 DBMS_ROWID.ROWID_TYPE

返回 rowid 的类型，ROWID_TYPE_RESTRICTED(0)和 ROWID_TYPE_EXTENDED(1)。

定义如下：

```
function dbms_rowid.rowid_type
(row_id in rowid)
return number;
```

例子：

Oracle8 以后的版本的 rowid 都是 extended 类型

```
SQL> select dbms_rowid.rowid_type(rowid) "type" from test;

     type

----------

      1
```

Oracle7 的 rowid 是 restricted 类型

```
SQL> select

dbms_rowid.rowid_type(chartorowid('00005CAA.0000.0001')) "type"

from dual;

     type

----------

       0
```

1.11 DBMS_ROWID.ROWID_VERIFY

验证一个 restricted 的 rowid 是否能够转换成 extended 的 rowid，它可以用来发现存在问题的 rowid。

定义如下：

```
function rowid_verify(rowid_in IN rowid,

schema_name IN varchar2,

object_name IN varchar2,

conversion_type IN integer)

return number;
```

1.11 利用 DBMS_ROWID 包恢复的一个例子

介绍了这个包的用法后，其实对于我们最主要的还是利用 DBMS_ROWID.ROWID_CREATE 来解决坏块的一些问题，下面举一个具体如何使用这个包来解决坏块的例子。

```
SQL> create tablespace test datafile

'd:\oracle\oradata\orcl\test.dbf' size 5M;

表空间已创建。
```

```
SQL> create user coolyl identified by coolyl
  2  default tablespace test
  3  temporary tablespace temp;
用户已创建。
SQL> grant connect,resource,dba to coolyl;
授权成功。
SQL> connect coolyl/coolyl
已连接。
SQL> create table test as select * from dba_objects;
表已创建。
SQL> insert into test select * from test;
已创建 6238 行。
SQL> insert into test select * from test;
已创建 12476 行
SQL> r
  1* insert into test select * from test
insert into test select * from test*
ERROR 位于第 1 行:
ORA-01653: 表 COOLYL.TEST 无法通过 128（在表空间 TEST 中）扩展
SQL> select count(*) from test;
  COUNT(*)
----------
    24952
SQL> commit;
提交完成。
SQL> create index indx_test on test(object_id);
索引已创建。
SQL> select status from dba_indexes where index_name='INDX_TEST';
STATUS
--------
```

```
VALID

SQL> connect sys/oracle as sysdba

已连接

SQL> shutdown immediate

数据库已经关闭。

已经卸载数据库。

ORACLE 例程已经关闭。

用 winhex 软件手工修改 d:\oracle\oradata\orcl\test.dbf 这个文件中的几个 block 的数值，
模拟产生坏块。

SQL> startup

ORACLE 例程已经启动。

Total System Global Area   97591024 bytes

Fixed Size                   454384 bytes

Variable Size              71303168 bytes

Database Buffers           25165824 bytes

Redo Buffers                 667648 bytes

数据库装载完毕。

数据库已经打开。

SQL> select count(*) from coolyl.test;

select count(*) from coolyl.test

*ERROR 位于第 1 行:

ORA-01578: ORACLE 数据块损坏（文件号 3，块号 23）

ORA-01110: 数据文件 3: 'D:\ORACLE\ORADATA\ORCL\TEST.DBF'

D:\oracle\oradata\orcl>dbv file=test.dbf blocksize=8192

DBVERIFY: Release 9.2.0.5.0 - Production on 星期一 11 月 29 17:26:44 2004

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

DBVERIFY - 验证正在开始 : FILE = test.dbf

标记为损坏的页 23

***

Corrupt block relative dba: 0x00c00017 (file 3, block 23)
```

```
Bad check value found during dbv:
Data in bad block -
type: 6 format: 2 rdba: 0x00c00017
last change scn: 0x0000.0005cc2a seq: 0x2 flg: 0x04
consistency value in tail: 0xcc2a0602
check value in block header: 0x9206, computed block checksum: 0x314b
spare1: 0x0, spare2: 0x0, spare3: 0x0
***
DBVERIFY - 验证完成
检查的页总数        : 640
处理的页总数（数据）: 510
失败的页总数（数据）: 0
处理的页总数（索引）: 54
失败的页总数（索引）: 0
处理的页总数（其它）: 10
处理的总页数 (段)  : 0
失败的总页数 (段)  : 0
空的页总数          : 65
标记为损坏的总页数: 1
汇入的页总数         : 0
由 DBV 工具检测出来的结果也可以看到 TEST.DBF 文件存在一个坏块。
SQL> SELECT tablespace_name, relative_fno,segment_type, owner,
segment_name, partition_name FROM dba
_extents  WHERE file_id = 3 AND 23 between block_id and block_id +
blocks -1;
TABLESPACE_NAME             RELATIVE_FNO SEGMENT_TYPE
------------------------------ ------------ ------------------
OWNER
------------------------------
SEGMENT_NAME
```

```
---------------

PARTITION_NAME

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

TEST                                3 TABLE

COOLYL

TEST

SQL> SELECT data_object_id FROM dba_objects WHERE object_name = 'TEST'

AND owner = 'COOLYL';

DATA_OBJECT_ID

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

        6907

SQL> SELECT dbms_rowid.rowid_create(1,6907,3,23,0) LOW_RID from

DUAL;

LOW_RID

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

AAABr7AADAAAAAXAAA

SQL> SELECT dbms_rowid.rowid_create(1,6907,3,24,0) HI_RID from

DUAL;

HI_RID

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

AAABr7AADAAAAAYAAA

SQL> CREATE TABLE test_bak AS SELECT /*+ ROWID(A) */ * FROM coolyl.test

A WHERE rowid < 'AAABr7AADAA

AAAXAAA';

表已创建。

SQL>

SQL> INSERT INTO test_bak SELECT /*+ ROWID(A) */ * FROM coolyl.test

A WHERE rowid >= 'AAABr7AADAAAAA

YAAA';

已创建 23812 行。
```

```
SQL> commit;

提交完成。

SQL> select count(*) from test_bak;

  COUNT(*)

----------

    24873

可以看到丢失了 24952-24873=79 条记录。

SQL> connect coolyl/coolyl

已连接。

SQL> select status from dba_indexes where index_name='INDX_TEST';

STATUS

--------

VALID

SQL> drop table test;

表已丢弃。

SQL> create table test as select * from sys.test_bak;

表已创建。

SQL> select status from dba_indexes where index_name='INDX_TEST';

未选定行。

SQL> create index indx_test on test(object_id);

索引已创建。

SQL> select status from dba_indexes where index_name='INDX_TEST';

STATUS

--------

VALID

SQL> select count(*) from test;

  COUNT(*)

----------

    24873

D:\oracle\oradata\orcl>dbv file=test.dbf blocksize=8192
```

```
DBVERIFY: Release 9.2.0.5.0 - Production on 星期一 11月 29 17:58:09
2004
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
DBVERIFY - 验证正在开始 : FILE = test.dbf
DBVERIFY - 验证完成
检查的页总数        : 640
处理的页总数（数据）: 459
失败的页总数（数据）: 0
处理的页总数（索引）: 106
失败的页总数（索引）: 0
处理的页总数（其它）: 10
处理的总页数 (段)  : 0
失败的总页数 (段)  : 0
空的页总数          : 65
标记为损坏的总页数: 0
```

汇入的页总数　　：0

作者简介：

叶梁，网名 coolyl

现任 itpub Oracle 管理版版主

擅长数据库的维护，对于数据库的安装，调整，备份方面有自己独到的经验。同时也给一些国内的大型企业做过 oracle 的培训，有一定的培训经验。

曾做过很多大型项目的数据库维护和支持工作，对 oracle 的维护有相当多的实际经验，善于现场解决问题。

曾任职于国内某大型软件企业做 oracle 数据库的技术支持，客户遍及全国各个行业，尤其是电信，政府行业。

现任职于某外资电信企业华北区分公司，DBA，负责华北区 40 多个数据库系统的维护，对大型数据库管理经验丰富。

《Oracle 数据库 DBA 专题技术精粹》一书的主编及主要作者.

更多可以访问 http://blog.itpub.net/coolyl

关于字符集更改的内部操作

本文作者: eygle (eygle.com@gmail.com)

摘要:

字符集问题对于 Oracle 数据库是极为重要的，了解和正确设置和修改字符集对于 DBA 来说非常重要，本系列文章就此进行了深入讨论。

关于字符集问题的初步探讨，曾经在《Oracle 数据库 DBA 专题技术精粹》一书中发表，本文是后来补充的内容，并未包含在原文当中。

在以前的文章中我们提到，通过修改 props$的方式更改字符集在 Oracle7 之后是一种极其危险的方式，应该尽量避免。

我们又知道，通过 ALTER DATABASE CHARACTER SET 更改字符集虽然安全可靠，但是有严格的子集和超集的约束，实际上我们很少能够用到这种方法。

实际上 Oracle 还存在另外一种更改字符集的方式.

1.1 修改字符集的内部做法

如果你注意过的话，在 Oracle 的 alert<sid>.log 文件中，你可能看到过这样的日志信息:

```
alter database character set INTERNAL_CONVERT ZHS16GBK
Updating character set in controlfile to ZHS16GBK
 SYS.SNAP$ (REL_QUERY) - CLOB representation altered
 SYS.METASTYLESHEET (STYLESHEET) - CLOB representation altered
 SYS.EXTERNAL_TAB$ (PARAM_CLOB) - CLOB representation altered
 XDB.XDB$RESOURCE (SYS_NC00027$) - CLOB representation altered
 ODM.ODM_PMML_DTD (DTD) - CLOB representation altered
 OE.WAREHOUSES (SYS_NC00003$) - CLOB representation altered
```

```
 PM.ONLINE_MEDIA (SYS_NC00042$) - CLOB representation altered

 PM.ONLINE_MEDIA (SYS_NC00062$) - CLOB representation altered

 PM.ONLINE_MEDIA (PRODUCT_TEXT) - CLOB representation altered

 PM.ONLINE_MEDIA (SYS_NC00080$) - CLOB representation altered

 PM.PRINT_MEDIA (AD_SOURCETEXT) - CLOB representation altered

 PM.PRINT_MEDIA (AD_FINALTEXT) - CLOB representation altered

Completed: alter database character set INTERNAL_CONVERT ZHS1
```

在这里面，我们看到这样一条重要的，Oracle 非公开的命令:

```
alter database character set INTERNAL_CONVERT/ INTERNAL_USE ZHS16GBK
```

这个命令是当你选择了使用典型方式创建了种子数据库以后，Oracle 会根据你选择的字符集设置，把当前种子数据库的字符集更改为期望字符集，这就是这条命令的作用.

在使用这个命令时，Oracle 会跳过所有子集及超集的检查，在任意字符集之间进行强制转换，所以，使用这个命令时你必须十分小心，你必须清楚这一操作会带来的风险.

我们之前讲过的内容仍然有效，你可以使用 csscan 扫描整个数据库，如果在转换的字符集之间确认没有严重的数据损坏，或者你可以使用有效的方式更改，你就可以使用这种方式进行转换.

1.1 内部原理及说明

我们来看一下具体的操作过程及 Oracle 的内部操作:

```
SQL> shutdown immediate

Database closed.

Database dismounted.

ORACLE instance shut down.

SQL> startup mount
```

```
ORACLE instance started.

Total System Global Area  135337420 bytes

Fixed Size                 452044 bytes

Variable Size           109051904 bytes

Database Buffers         25165824 bytes

Redo Buffers              667648 bytes

Database mounted.

SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;


System altered.


SQL> ALTER SYSTEM SET JOB_QUEUE_PROCESSES=0;


System altered.


SQL> ALTER SYSTEM SET AQ_TM_PROCESSES=0;


System altered.


SQL> ALTER DATABASE OPEN;

Database altered.

SQL> alter session set events '10046 trace name context forever,level 12';


Session altered.


SQL> alter database character set INTERNAL_USE ZHS16CGB231280


Database altered.


SQL>
```

这是 alert.log 文件中的记录信息:

```
Tue Oct 19 16:26:30 2004
Database Characterset is ZHS16GBK
replication_dependency_tracking turned off (no async multimaster replication
found)
Completed: ALTER DATABASE OPEN
Tue Oct 19 16:27:07 2004
alter database character set INTERNAL_USE ZHS16CGB231280
Updating character set in controlfile to ZHS16CGB231280
Tue Oct 19 16:27:15 2004
Thread 1 advanced to log sequence 118
  Current log# 2 seq# 118 mem# 0: /opt/oracle/oradata/primary/redo02.log
Tue Oct 19 16:27:15 2004
ARC0: Evaluating archive   log 3 thread 1 sequence 117
ARC0: Beginning to archive log 3 thread 1 sequence 117
Creating          archive          destination          LOG_ARCHIVE_DEST_1:
'/opt/oracle/oradata/primary/archive/1_117.dbf'
ARC0: Completed archiving  log 3 thread 1 sequence 117
Tue Oct 19 16:27:20 2004
Completed: alter database character set INTERNAL_USE ZHS16CGB231280
Shutting down instance: further logons disabled
Shutting down instance (immediate)
License high water mark = 1
Tue Oct 19 16:29:06 2004
ALTER DATABASE CLOSE NORMAL
...
```

格式化 10046 跟踪文件,得到以下信息(摘要):

```
alter session set events '10046 trace name context forever,level 12'


alter database character set INTERNAL_USE ZHS16CGB231280


call     count       cpu    elapsed       disk      query    current        rows

------- ------  -------- ---------- ---------- ---------- ----------
```

```
Parse       1      0.00       0.00          0          0          0           0
Execute     1      4.88       6.04        910      16825      18099           0
Fetch       0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------
total       2      4.88       6.04        910      16825      18099           0

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: SYS

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------
  control file sequential read                    4        0.00          0.00
  control file parallel write                     2        0.05          0.08
  log file sync                                   2        0.08          0.08
  SQL*Net message to client                       1        0.00          0.00
  SQL*Net message from client                     1       18.06         18.06
********************************************************************************
....
update col$ set charsetid = :1
where
 charsetform = :2

....

update argument$ set charsetid = :1
where
 charsetform = :2
....
```

```
update collection$ set charsetid = :1
where
 charsetform = :2
....

update attribute$ set charsetid = :1
where
 charsetform = :2
....

update parameter$ set charsetid = :1
where
 charsetform = :2
....

update result$ set charsetid = :1
where
 charsetform = :2

....

update partcol$ set spare1 = :1
where
 charsetform = :2

....

update subpartcol$ set spare1 = :1
where
```

```
 charsetform = :2


....


update props$ set value$ = :1
where
 name = :2


....


update "SYS"."KOTAD$" set SYS_NC_ROWINFO$ = :1
where
 SYS_NC_OID$ = :2
....


update seq$ set increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,
  cache=:7,highwater=:8,audit$=:9,flags=:10
where
 obj#=:1


....


update kopm$ set metadata = :1,  length
  = :2
where
 name='DB_FDO'


....


ALTER DATABASE CLOSE NORMAL
```

此处生成的日志你可以在这里下载(供参考):

http://www.eygle.com/special/primary_ora_13730.zip

http://www.eygle.com/special/primary_ora_13730.tkf.log

我们看到这个过程和之前 ALTER DATABASE CHARACTER SET 操作的内部过程是完全相同的，也就是说 INTERNAL_USE 提供的帮助就是使

Oracle 数据库绕过了子集与超集的校验.

这一方法在某些方面是有用处的，比如测试；应用于产品环境大家应该格外小心，除了你以外，没有人会为此带来的后果负责:

1.1 结语(我们不妨再说一次):

对于 DBA 来说，有一个很重要的原则就是:不要把你的数据库置于危险的境地！

这就要求我们，在进行任何可能对数据库结构发生改变的操作之前，先做有效的备份，很多 DBA 没有备份的操作中得到了惨痛的教训。

更多内容请参考:

关于字符集问题的系列探讨文章:

http://www.eygle.com/index-special.htm

作者简介:

盖国强，网名 eygle

曾任 ITPUB MS 版版主,现任 itpub Oracle 管理版版主.

曾任职于某国家大型企业,服务于烟草行业,开发过基于 Oracle 数据库的大型 ERP 系统,属国家信息产业部重点工程.同时负责 Oracle 数据库管理及优化,并为多家烟草企业提供 Oracle 数据库管理、优化及技术支持.

目前任职于北京某电信增值业务系统提供商企业,首席 DBA,负责数据库业务.管理全国 30 多个数据库系统。项目经验丰富,曾设计规划及支持中国联通增值业务等大型数据库系统.

实践经验丰富，长于数据库诊断、性能调整与 SQL 优化等。对于 Oracle 内部技术具有深入研究。

高级培训讲师,培训经验丰富,曾主讲 itpub dba 培训及 itpub 高级性能调整等主要课程.

《Oracle 数据库 DBA 专题技术精粹》一书的主编及主要作者.

你可以在http://www.eygle.com上找到关于作者的更多信息.

如何处理 Oracle 数据库中的坏块问题

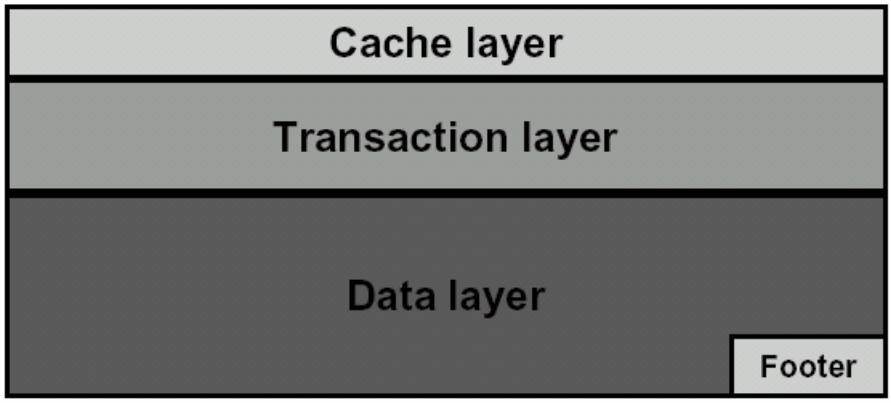
本文作者: coolyl (allan@itpub.net)

摘要:

本文主要介绍如何去处理在 Oracle 数据库中出现坏块的问题，对于坏块产生在不同的对象上，处理的方法会有所不同，本文将大致对这些方法做一些介绍。因为数据库运行时间长了，由于硬件设备的老化，出现坏块的几率会越来越大，因此，做为一个 DBA，怎么去解决数据库出现的坏块问题就成了一个重要的议题了.

1.1 什么是数据库的坏块

首先我们来大概看一下数据库块的格式和结构，如下图所示：



数据库的数据块有固定的格式和结构，分三层：cache layer，transaction layer，data layer。在我们对数据块进行读取写入操作的时候，数据库会对要读写的数据块做一致性的检查，其中包括：数据块的类型、数据块的地址信息、数据块的 SCN 号以及数据块的头部和尾部。如果发现其中有不一致的信息，那数据库就会标记这个数据块为坏块了。数据库的坏块分为两种，逻辑坏块和物理坏块。

1.2 坏块对数据库产生的影响

如果数据库出现坏块，数据库的告警日志文件里面会存在有如下的一些报错信息：Ora-1578 以及 Ora-600 and trace file in bdump directory，其中 Ora-600 错误的第一个参数值的

范围是[2000]-[8000]，不同的值代表着数据块的不同的层出现问题，具体的如下表所示：

```
Range                block layer

Cache layer           2000 – 4000

Transaction layer      4000 – 6000

Data layer             6000 - 8000
```

坏块产生影响的对象可能是数据字典表、回滚段表、临时段、用户数据表和索引等。不同的对象产生坏块后的处理方法不尽相同。

1.3 坏块产生的原因

Oracle 调用标准 C 的系统函数，对数据块进行读写操作，因此，坏块是有可能由以下几种原因产生：

硬件的 I/O 错误

操作系统的 I/O 错误或缓冲问题

内存或 paging 问题

磁盘修复工具

一个数据文件的一部分正在被覆盖

Oracle 试图访问一个未被格式化的系统块失败

数据文件部分溢出

Oracle 或者操作系统的 bug

1.4 坏块的处理方法

1.4.1 先收集相应的关于坏快的信息

从 AlertSID.log 文件或者从 trace 文件中查找，找到例如以下的一些信息：

```
Ora-1578  file#  (RFN)  block#

Ora-1110  file#  (AFN)  block#

Ora-600   file#  (AFN)  block#
```

其中 RFN 表示的是 relative_fno

AFN 表示的是 file_id

```
   Select file_name,tablespace_name,file_id “AFN”,relative_fno “RFN”
From dba_data_files;
Select file_name,tablespace_name,file_id, relative_fno “RFN”
From dba_temp_files;
```

1.4.2 确定存在坏块的对象是什么：

```
SELECT tablespace_name, segment_type, owner, segment_name, partition_name  FROM
dba_extents WHERE file_id = <AFN> and <BL> between block_id AND block_id + blocks
- 1;
```

通过上面这个查询语句就可以查出当前存在坏块的对象是什么，是什么类型的对象。需要注意的是如果是 temp 文件中出现坏块，是没有记录返回的。

1.4.3 根据上面查询出来的对象类型，确定相应的处理方法

出现坏块的常见对象有：

Sys 用户下的对象

回滚段

临时段

索引或者分区索引

表

常用的处理方法有：

恢复数据文件

只恢复坏的 block（9i 以上版本可用）

通过 ROWID RANGE SCAN 保存数据

使用 DBMS_REPAIR

使用 EVENT

1.4.4 具体处理方法的介绍

1.4.4.1 恢复数据文件方法

如果数据库是归档方式下，并且有完整的物理备份，就可以使用此方法来恢复。

步骤如下：

先 offline 受影响的数据文件，执行以下的语句：

```
ALTER DATABASE DATAFILE 'name_file' OFFLINE;
```

保留有坏块的数据文件，然后拷贝备份的数据文件。如果恢复的数据文件要求路径不同，执行以下的语句：

```
ALTER DATABASE RENAME FILE 'old_name' TO 'new_name';
```

恢复数据文件，执行以下语句：

```
RECOVER DATAFILE 'name_of_file';
```

Online 恢复后的数据文件，执行以下的语句：

```
ALTER DATABASE DATAFILE 'name_of_file' ONLINE;
```

1.4.4.2 只恢复坏的 block（9i 以上版本可用）

使用这种方法要求数据库版本是 9.2.0 以上，要求配置了 Rman 的 catalog 数据库，数据库为归档方式，并且有完整的物理备份。

步骤如下：

使用 RMAN 的 BLOCKRECOVER 命令 ：

```
Rman>run{blockrecover datafile 5 block 11,16;}
```

也可以强制使用某个 SCN 号之前的备份，恢复数据块。

```
Rman>run{blockrecover datafile 5 block 11,16 restore until sequence 8505;}
```

1.4.4.3 通过 ROWID RANGE SCAN 保存数据

先取得坏块中 ROW ID 的最小值，执行以下的语句：

```
SELECT dbms_rowid.rowid_create(1,<OBJ_ID>,<RFN>,<BL>,0) from DUAL;
```

2）取得坏块中的 ROW ID 的最大值，执行以下的语句：

```
SELECT dbms_rowid.rowid_create(1,<OBJ_ID>,<RFN>,<BL>+1,0) from DUAL;
```

3）建立一个临时表存储那些没有坏块的数据，执行以下的语句：

```
CREATE TABLE salvage_table AS SELECT  *  FROM  corrupt_tab Where 1=2;
```

4）保存那些不存在坏块的数据到临时表中，执行以下的语句：

```
INSERT INTO salvage_table SELECT /*+ ROWID(A) */ * FROM <owner.tablename> A
WHERE rowid < '<low_rid>';
```

```
INSERT INTO salvage_table SELECT /*+ ROWID(A) */ * FROM <owner.tablename> A  WHERE rowid >= '<hi_rid>';
```

5) 根据临时表中的数据重建表，重建表上的索引，限制。

1.4.4.4 使用 10231 诊断事件，在做全表扫描的时候跳过坏块

可以在 session 级别设定：

```
ALTER SESSION SET EVENTS '10231 TRACE NAME CONTEXT FOREVER, LEVEL 10';
```

也可以在数据库级别上设定，在初始化参数中加入：event="10231 trace name context forever, level 10" ，然后重启数据库。

然后从存在坏块的表中取出不存在坏块的数据，执行以下的语句：

```
CREATE TABLE salvage_emp AS SELECT * FROM corrupt_table;
```

最后 rename 生成的 corrupt_table 为原来表的名字，并重建表上的索引和限制。

1.4.4.5 使用 dbms_repair 包进行恢复

使用 dbms_repair 标记有坏块的表，在做全表扫描的时候跳过坏块，执行以下的语句：

```
  Execute DBMS_REPAIR.SKIP_CORRUPT_BLOCKS('<schema>','<tablename>');
```

然后使用 exp 工具或者 createtable as select 的方法取出没有坏块数据，然后重建表，表上的索引和限制。

1.5 坏块的预先发现的方法

1.5.1 利用 exp 工具

如果要检测数据库中所有的表，可以利用 exp 工具导出整个数据库可以检测坏块。不过这个工具有一些缺陷，对以下情况的坏块是检测不出来的：

HWM 以上的坏块是不会发现的

索引中存在的坏块是不会发现的

数据字典中的坏块是不会发现的

1.5.2 利用 ANALYZE TABLE tablename VALIDATE STRUCTURE CASCADE 的方法

如果只是对数据库中比较重要的表进行坏块检查，可以使用 ANALYZE TABLE tablename VALIDATE STRUCTURE CASCADE 的方法来检测坏块，它执行坏块的检查，但是不会标记坏块为 corrupt，检测的结果保存在 USER_DUMP_DEST 目录下的用户 trace 文件

中。

1.5.3 利用 dbv 工具

使用 Oracle 的专门工具 dbv 来检查坏块，具体的语法如下：

```
关键字      说明                  (默认)

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

FILE        要验证的文件              (无)

START       起始块                  (文件的第一个块)

END         结束块                  (文件的最后一个块)

BLOCKSIZE   逻辑块大小            (2048)

LOGFILE     输出日志                  (无)

FEEDBACK    显示进度              (0)

PARFILE     参数文件                (无)

USERID      用户名/口令             (无)

SEGMENT_ID  段 ID (tsn.relfile.block) (无)
```

例如：

```
Dbv file=system01.dbf blocksize=8192

DBVERIFY: Release 9.2.0.5.0 - Production on 星期六 11 月 27 15:29:13 2004

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

DBVERIFY - 验证正在开始 : FILE = system01.dbf

DBVERIFY - 验证完成

检查的页总数        : 32000

处理的页总数（数据）: 13261

失败的页总数（数据）: 0

处理的页总数（索引）: 2184

失败的页总数（索引）: 0

处理的页总数（其它）: 1369

处理的总页数 (段)  : 0

失败的总页数 (段)  : 0

空的页总数          : 15186

标记为损坏的总页数: 0
```

```
汇入的页总数              : 0
```

注：因为 dbv 要求 file 后面跟的必须是一个文件扩展名，所以如果用裸设备存储的，就必须使用 ln 链接裸设备到一个文件，然后再用 dbv 对这个链接文件进行检查。

作者简介：

叶梁，网名 coolyl

现任 itpub Oracle 管理版版主

擅长数据库的维护，对于数据库的安装，调整，备份方面有自己独到的经验。同时也给一些国内的大型企业做过 oracle 的培训，有一定的培训经验。

曾做过很多大型项目的数据库维护和支持工作，对 oracle 的维护有相当多的实际经验，善于现场解决问题。

曾任职于国内某大型软件企业做 oracle 数据库的技术支持，客户遍及全国各个行业，尤其是电信，政府行业。

现任职于某外资电信企业华北区分公司，DBA，负责华北区 40 多个数据库系统的维护，对大型数据库管理经验丰富。

《Oracle 数据库 DBA 专题技术精粹》一书的主编及主要作者.

更多可以访问 http://blog.itpub.net/coolyl

CSDN 第二书店好书推荐:

Oracle JDeveloper 10g 与 J2EE 实战演练 新书

链接地址：http://www.dearbook.com.cn/book/viewbook.aspx?pno=TS0027476

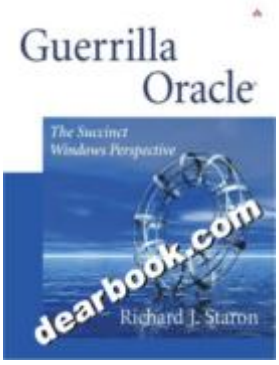
作　　者：何致亿
出　版　社：电子工业出版社
ISBN 书号：7-121-00501-8

本书系统地介绍了 Oracle JDeveloper 10g 提出的革命性的 J2EE 开发框架 —— Oracle Application Development Framework（ADF）。Oracle ADF 是以 J2EE 设计模式为基础的，不仅可以帮助程序员开发更健壮的 J2EE 应用系统，也可以大幅提高程序撰写效率，缩短开发时间。本书由 Oracle 资深技术顾问何致亿先生撰写，涵盖全面的 JDeveloper 10g 与 J2EE 开发技术，内容包括：采用 Oracle ADF 架构的优点，安装 Oracle JDeveloper 10g 与 OC4J 10g，安装 Oracle 10g 数据库服务器，JDeveloper 10g 集成开发环境与项目管理方式，使用 JDeveloper 调试器与性能调校工具，利用 JDBC 存取 Oracle 10g 数据库，管理 Oracle 数据库对象，PL/SQL 存储过程的开发与调试，创建与测试 Oracle ADF 业务组件（Business Components），利用 Oracle ADF 业务组件开发 JClient 应用程序，开发 Servlet 与 JSP，JDeveloper 与 Struts 集成应用，创建、测试与部署 EJB，Web Services 的开发与部署方式等。

本书内容全面，讲解详细易懂，由浅入深，既可作为初学者的入门指导，也可提高高级程序员的开发能力。

Guerrilla Oracle: The Succinct Windows Perspective 原版进口

链接地址：http://www.dearbook.com.cn/book/viewbook.aspx?pno=TS0027014



作　　者：Richard Staron
出 版 社：Pearson Education
ISBN 书号：0201750775

Are you frustrated by your attempts to learn Oracle or improve your Oracle skills because of the sheer amount of technical documentation you have to wade through? This concise tutorial walks you step-by-step through the process, showing you exactly what you need to know to install, create, and support a successful Oracle 8i or 9i environment with Web capabilities.

It presents clear explanations of database, SQL, and Oracle fundamentals. Using a real-world, large-scale example to demonstrate essential tasks, the book follows the Oracle DBMS life cycle from business idea to functioning database.