# Oarcle

## DDL

### 表空间与用户

//xxx表空间

create tablespace dzjc\_index

logging datafile 'E:\datafileOfOracle\spjc\_index.dbf'

size 32m

autoextend on

next 32m maxsize unlimited

extent management local

segment space management auto;

//临时表空间

create temporary tablespace dzjc\_temp

tempfile 'E:\datafileOfOracle\spjc\_temp.dbf'

size 32m

autoextend on

next 32m maxsize unlimited

extent management local;

//segment space management auto; 临时表空间不支持空间管理 所以这句话需删掉

//导入

imp userid=zz\_spjc/zz\_spjc@orcl file='C:\Users\cui\Desktop\监察系统\郑州市\数据库\spjc\_2012-08-16.18-05-02.dmp' log='E:\oracle\_exp\_imp\_log\aaa.log' full=y

imp u\_website/u\_website@orcl file='C:\Users\cui\Desktop\dataOfDb\ny\_website.dmp' log='E:\oracle\_exp\_imp\_log\website.log' full=y

imp hn\_fgs\_tysp/12345678@192.168.1.114/orcl file=..

//导出

非dba用户导出文件时full的作用探讨

前提是scott是非dba角色用户

\* exp scott/tiger@orcl file='E:\oracle\_exp\_imp\_log\scott1.dmp'

log='E:\oracle\_exp\_imp\_log\aaa1.log' //结果是：成功

\* exp scott/tiger@orcl file='E:\oracle\_exp\_imp\_log\scott2.dmp'

log='E:\oracle\_exp\_imp\_log\aaa2.log' full=y

结果是：不成功 提示 必须是 DBA 才能执行完整数据库或表空间导出操作

\* exp scott/tiger@orcl file='E:\oracle\_exp\_imp\_log\scott3.dmp'

log='E:\oracle\_exp\_imp\_log\aaa3.log' owner=scott

结果是：成功

dba用户导出文件时full的作用探讨

前提是system是dba用户

\* exp system/manager@orcl file='E:\oracle\_exp\_imp\_log\system1.dmp'

log='E:\oracle\_exp\_imp\_log\aaa1.log'

结果是：成功 导出的仅仅是system的对象

\* exp system/manager@orcl file='E:\oracle\_exp\_imp\_log\system2.dmp'

log='E:\oracle\_exp\_imp\_log\aaa2.log' full=y

结果是：成功 但导出的是整个数据库

\* exp system/manager@orcl file='E:\oracle\_exp\_imp\_log\system3.dmp'

log='E:\oracle\_exp\_imp\_log\aaa3.log' owner=(scott,u\_spjc)

结果是：成功 导出的是scott，u\_spjc 用户的对象

总结：只有dba用户导出时才能使用full=y，且导出的是整个数据库，因此在导出时一般不会用到full=y

### 用户

//创建用户

create user zz\_spjc

profile default

identified by zz\_spjc

default tablespace dzjc

temporary tablespace dzjc\_temp;

//修改密码

alter user 用户名 identified by 密码

//赋予权限（赋予权限麻烦，我们直接给他赋予角色）

grant connect,resource,dba to zz\_spjc;

//再赋予其开发中常用的权限

grant create view to zz\_spjc;

grant select any dictionary to lyl;---说明：解决Dynamic Performance Tables not accessible 问题。

grant debug any procedure, debug connect session to lyl--说明：解决ora-0131，解决debug connect session system privilege

用户 表空间 删除创建最省事的方式，

drop user huigu cascade;//表空间我们没有必要创建，因为我们使用系统自带的

create user huigu identified by huigu;//此时使用oracle自带的users 和temp表空间

grant connect,resource to huigu;

create table test(id int, name varchar2(20));

insert into test values(1,'test');

--在创建用户时，可以指定数据表空间，如果不指定，默认表空间为user。

--在建表时，也可以指定表空间，你指定那个表空间，表数据就放在哪个表空间，如果不指定，则放在表所属用户的数据表空间。

### 表相关

drop table t\_about;

create table t\_about

(

id varchar2(32) not null,

name varchar2(100) not null,

about varchar2(32),

sortno varchar2(9) not null,

isvalid char(1) not null

);

alter table t\_about add primary key (id);

comment on column t\_about.id is 'uuid方式生成';

comment on column t\_about.name is '名称';

comment on column t\_about.about is '关于我们，附件id';

comment on column t\_about.sortno is '排序号';

comment on column t\_about.isvalid is '是否有效';

alter table t\_sbsn\_cpxx drop (scenicid); //删除列

alter table tm\_scenic\_ticket add ttype varchar2(20) ; //增加列

alter table t\_scenic\_video rename column video to svideo; //修改列名

alter table t\_scenic\_video modify stuname(varchar2 32)//修改列数据类型

//外键

//默认方式

ADD CONSTRAINT FK\_INVOICE\_ID

FOREIGN KEY(INVOICE\_ID ) REFERENCES T\_INVOICE(ID);

//省略模式

创建外键约束是，外键字段参照父表的主键或Unique约束字段。这种情况下可以不指定外键参照字段名，如下：

ALTER TABLE T\_INVOICE\_DETAIL ADD CONSTRAINT FK\_INVOICE\_ID

FOREIGN KEY(INVOICE\_ID) REFERENCES T\_INVOICE;

//级联模式

ALTER TABLE T\_INVOICE\_DETAIL

ADD CONSTRAINT FK\_INVOICE\_ID

FOREIGN KEY(INVOICE\_ID ) REFERENCES T\_INVOICE(ID)

ON DELETE CASCADE;

外键语法有个选项可以指定级联删除特征。这个特征仅作用于父表的删除语句。使用这个选项，父表的一个删除操作将会自动删除所有相关的子表记录

如果不能级联删除，可设置子表外键字段值为null,使用on delete set null语句(外键字段不能设置not null约束).

ALTER TABLE T\_INVOICE\_DETAIL

ADD CONSTRAINT FK\_INVOICE\_ID

FOREIGN KEY(INVOICE\_ID ) REFERENCES T\_INVOICE(ID)

ON DELETE SET NULL;

索引

CREATE [unique] INDEX indexname  
ON [user.]table (column [ASC | DESC] [,column  
[ASC | DESC] ] ... )

[CLUSTER [scheam.]cluster]  
[INITRANS n]  
[MAXTRANS n]  
[PCTFREE n]  
[STORAGE storage]  
[TABLESPACE tablespace]  
[NO SORT]  
Advanced

CLUSTER 指定一个聚簇（Hash cluster不能建索引）

INITRANS、MAXTRANS 指定初始和最大事务入口数  
Tablespace 表空间名  
STORAGE 存储参数，同create table 中的storage.  
PCTFREE 索引数据块空闲空间的百分比(不能指定pctused)  
NOSORT 不（能）排序（存储时就已按升序，所以指出不再排序）

ALTER [UNIQUE] INDEX [user.]index  
[INITRANS n]  
[MAXTRANS n]   
REBUILD   
[STORAGE n]

REBUILD 是 根据原来的索引结构重新建立索引，实际是删除原来的索引后再重新建立。

DROP INDEX [schema.]indexname

### 视图相关

CREATE OR REPLACE VIEW dept\_sum\_vw

(name,minsal,maxsal,avgsal)

AS

SELECT d.dname,min(e.sal),max(e.sal),avg(e.sal)

FROM emp e,dept d

WHERE e.deptno=d.deptno

GROUP BY d.dname;

## 常量

### sysdate

SQL> select sysdate ,to\_char(sysdate,'hh') from dual;

SYSDATE TO\_CHAR(SYSDATE,'HH')        
  -------------------- ---------------------        
   2003-10-13 19:35:21 07

 SQL> select sysdate ,to\_char(sysdate,'hh24') from dual;        
      
    SYSDATE TO\_CHAR(SYSDATE,'HH24')        
    -------------------- -----------------------        
    2003-10-13 19:35:21 19

## 自定义类型

### 记录类型

|  |
| --- |
| Type typename is record(  Column1 type[lenghth],  Column2 type [length]  ) |
| 举例1  declare    type type\_student IS RECORD(           name VARCHAR2(100),           age  NUMBER(3),           sex  VARCHAR2(1)   );      r\_student type\_student;   v\_name VARCHAR2(100);   i\_age  NUMBER(3);   v\_sex  VARCHAR(1);   i integer; begin   select name, age, sex into r\_student from t\_student where gid = 1;   v\_name := r\_student.name;   i\_age := r\_student.age;   v\_sex := r\_student.sex;   dbms\_output.put\_line(v\_name);   dbms\_output.put\_line(v\_sex);   dbms\_output.put\_line(i\_age); end; |

### 表类型

|  |
| --- |
| Type type\_name is table of element\_type index by [binary\_integer|pls\_integer|varchar2];  说明  element\_type:可以是基本类型(如varchar2, Ingeger, number等)、记录数据类型(即TYPE type\_name IS RECORD定义的类型)、%ROWTYPE。    INDEX BY:该语句的作用是使Number类型的下标自增长，自动初始化，并分配空间，有了该语句，向表记录插入元素时，不需要显示初始化，也不需要通过extend分配空间。Binary\_Integer 与 Pls\_Integer 都是整型类型.Binary\_Integer类型变量值计算是由Oracle来执行，不会出现溢出，但是执行速度较慢，因为它是由Oracle模拟执行。而Pls\_Integer的执行是由硬件即直接由CPU来运算，因而会出现溢出，但其执行速度较前者快许多。  举例  declare    TYPE t\_student\_rowtype IS TABLE OF t\_student%ROWTYPE INDEX BY BINARY\_INTEGER;      v\_name VARCHAR2(100);   i\_age  NUMBER(3);   v\_sex  VARCHAR(1);   i integer; begin  dbms\_output.put\_line('------------表记录类型表记录------------');   select \* into v\_tbl\_rowtype(1) from t\_student where gid = 1;   select \* into v\_tbl\_rowtype(2) from t\_student where gid = 2;   select \* into v\_tbl\_rowtype(3) from t\_student where gid = 3;  dbms\_output.put\_line('name:'||v\_tbl\_rowtype(1).name||', '||'age:'||v\_tbl\_rowtype(1).age||', '||'sex:'||v\_tbl\_rowtype(1).sex||', '||'grade:'||v\_tbl\_rowtype(1).grade);   dbms\_output.put\_line('name:'||v\_tbl\_rowtype(2).name||', '||'age:'||v\_tbl\_rowtype(2).age||', '||'sex:'||v\_tbl\_rowtype(2).sex||', '||'grade:'||v\_tbl\_rowtype(2).grade);  dbms\_output.put\_line('name:'||v\_tbl\_rowtype(3).name||', '||'age:'||v\_tbl\_rowtype(3).age||', '||'sex:'||v\_tbl\_rowtype(3).sex||', '||'grade:'||v\_tbl\_rowtype(3).grade);end; |

### 游标类型

**游标**是SQL的一个内存工作区，由系统或用户以变量的形式定义。游标的作用就是用于临时存储从数据库中提取的数据块。

**Oracle数据库**的Cursor类型:

静态游标：分为显式（explicit）游标和隐式（implicit）游标；

REF游标：是一种引用类型，类似于指针。

#### 静态游标

##### 隐式游标

**隐式游标分类：**

* Select …INTO…语句
* For xx in xxx loop语句
* DML语句（增删改查语句）

**隐式游标使用：**

可以通过隐式Cusor的属性来了解操作的状态和结果。Cursor的属性包含：

* SQL%ROWCOUNT 整型代表DML语句成功执行的数据行数。
* SQL%FOUND  布尔型值为TRUE代表插入、删除、更新或单行查询操作成功。
* SQL%NOTFOUND 布尔型与SQL%FOUND属性返回值相反。
* SQL%ISOPEN 布尔型DML执行过程中为真，结束后为假。

举例：

|  |
| --- |
| declare  begin  update employees set employee\_name='Mike' where employee\_id=1001;  if SQL%FOUND then  dbms\_output.put\_line('Name is updated');  else  dbms\_output.put\_line('Name is not updated');  end if;  end; |
| declare  begin  for tableInfo in (select \* from user\_tables) loop  dbms\_output.put\_line(tableInfo.table\_name);  end loop;  exception  when others then  dbms\_output.put\_line(sqlerrm);  end; |

##### 显式游标

**1) 显式Cursor的属性包含：**

游标的属性   返回值类型   意义

%ROWCOUNT   整型  获得FETCH语句返回的数据行数

%FOUND  布尔型 最近的FETCH语句返回一行数据则为真，否则为假

%NOTFOUND   布尔型 与%FOUND属性返回值相反

%ISOPEN 布尔型 游标已经打开时值为真，否则为假

**2) 对于显式游标的运用分为四个步骤：**

a 定义游标---Cursor  [Cursor Name]  IS;

b 打开游标---Open  [Cursor Name];

c  操作数据---Fetch  [Cursor name]

d  关闭游标---Close [Cursor Name]

|  |
| --- |
| **loop循环的无参游标**  declare  cursor cur is select \* from user\_tables;  tableInfo user\_tables%rowtype;  begin  open cur;  loop  fetch cur into tableInfo;  exit when cur%notfound;  dbms\_output.put\_line(tableInfo.table\_name);  end loop;  exception  when others then  dbms\_output.put\_line(sqlerrm);  close cur;  end; |
| **for循环的无参游标**  declare  cursor cur is select \* from user\_tables;  begin  for tableInfo in cur loop  dbms\_output.put\_line(tableInfo.table\_name);  end loop;  exception  when others then  dbms\_output.put\_line(sqlerrm);  end; |
| **Loop循环的有参游标**  declare  cursor cur(tblName varchar2) is select \* from  user\_constraints where table\_name=tblName;  tableInfo user\_constraints%rowtype;  begin  open cur('EMPLOYEES');  loop  fetch cur into tableInfo;  exit when cur%notfound;  dbms\_output.put\_line(tableInfo.constraint\_name);  end loop;  exception  when others then  dbms\_output.put\_line(sqlerrm);  close cur;  end; |
| **For循环有参游标**  declare  cursor cur(tblName varchar2) is select \* from  user\_constraints where table\_name=tblName;  begin  for tableInfo in cur('EMPLOYEES') loop  dbms\_output.put\_line(tableInfo.constraint\_name);  end loop;  exception  when others then  dbms\_output.put\_line(sqlerrm);  end |

#### REF游标

REF Cursor在运行的时候才能确定游标使用的查询。利用REF CURSOR，可以在程序间传递结果集（一个程序里打开游标变量，在另外的程序里处理数据）。

**REF CURSOR分两种:**

Strong REF CURSOR:指定retrun type，CURSOR变量的类型必须和return type一致。

Weak REF CURSOR:不指定return type，能和任何类型的CURSOR变量匹配。

**Ref cursor的使用:**

1) Type [Cursor type name] is ref cursor

2) Open cursor for...

3) Fetch  [Cursor name]

4) Close Cursor

举例：

|  |
| --- |
| //建包  create or replace package TEST as  type employees\_refcursor\_type is ref cursor return employees%rowtype;  procedure employees\_loop(employees\_cur IN employees\_refcursor\_type);  end TEST; |
| //实现包  //游标在此取值  create or replace package body TEST as  procedure employees\_loop(employees\_cur IN employees\_refcursor\_type) is  emp employees%rowtype;  begin  loop  fetch employees\_cur into emp;  exit when employees\_cur%NOTFOUND;  dbms\_output.put\_line(emp.employee\_id);  end loop;  end employees\_loop;  end TEST; |
| //调用  //游标是在此创建 在此打开 在此关闭  declare  empRefCur TEST.employees\_refcursor\_type;  begin  for i in 10..20 loop  dbms\_output.put\_line('Department ID=' || i);  open empRefCur for select \* from employees where department\_id=i;  TEST.employees\_loop(empRefCur);  end loop;  exception  when others then  dbms\_output.put\_line(sqlerrm);  close empRefCur;  end; |

## 函数

### 自带函数

#### 日期函数

##### to\_date

以2007-11-02   13:45:25:181为例：

Year:

        yy two digits 两位年                显示值:07

        yyy three digits 三位年                显示值:007

        yyyy four digits 四位年                显示值:2007

Month:

        mm    number     两位月              显示值:11

        mon    abbreviated 字符集表示          显示值:11月,若是英文版,显示nov

        month spelled out 字符集表示          显示值:11月,若是英文版,显示november

        Day:

        dd    number         当月第几天        显示值:02

        ddd    number         当年第几天        显示值: 310

        dy    abbreviated 当周第几天简写    显示值:星期五,若是英文版,显示fri

        day    spelled out   当周第几天全写    显示值:星期五,若是英文版,显示friday

        ddspth spelled out, ordinal twelfth

Hour:

              hh    two digits 12小时进制            显示值:01

              hh24 two digits 24小时进制            显示值:13

        Minute:

              mi    two digits 60进制                显示值:45

         Second:

              ss    two digits 60进制                显示值:25

微妙

SSS 显示值:181

##### add\_months

select sysdate from dual;

2008-5-26 23:55:28

select add\_months(sysdate,-2) from dual;

2008-3-26 23:55:28

不存在add\_years、add\_days，那么年、日、时、分、秒、微妙的增减该怎么办？

加1年：add\_months(sysdate,12)

加1天：sysdate+1

加1小时：sysdate+1/24

加1分钟：sysdate+1/24\*60

加1秒钟：sysdate+1/24\*60\*60

加1微妙：sysdate+1/24\*60\*60\*1000

##### next\_day

NEXT\_DAY(d,number)：下一个星期几的日期

星期日 = 1  星期一 = 2  星期二 = 3 星期三 = 4  星期四 = 5  星期五 = 6  星期六 = 7

##### last\_day

Last\_day 返回指定日期对应月份的最后一天

例：last\_day(to\_date('1999.11.29','yyyy.mm.dd'))//1999.11.31

last\_day(to\_date('2000.02.01','yyyy.mm.dd'))//2000.02.29

##### trunc

**select to\_char(sysdate,'yyyy-mm-dd hh24:mi:ss') from dual;**2011-12-29 16:24:34

**select trunc(sysdate-1,'w') from dual;  -- 离当前时间最近的周四，若当天为周四则返回当天，否则返回上周四**2011-12-22

**select trunc(sysdate,'ww') from dual;**2011-12-24//**--截取到上周末（上周周六）**

**select trunc(sysdate,'iw') from dual;**2011-12-26//**--本周第2天，即本周一**

**select trunc(sysdate,'year') from dual;**2011-1-1//截取到年

**select trunc(sysdate,'yyyy') from dual;**2011-1-1//截取到年

**select trunc(sysdate,'month') from dual;**2011-12-1//截取到月

**select trunc(sysdate,'mm') from dual;**2011-12-1//截取到月

**select trunc(sysdate,'day') from dual;**2011-12-25//**--截取到周（本周第一天，即上周日）**

**select trunc(sysdate,'d') from dual;**2011-12-25//**--截取到上周期（本周第一天，即上周日）d相当于day**

**select to\_char(trunc(sysdate,'dd'),'yyyymmdd hh24:mi:ss') from dual;** 20111229 00:00:00//**--截取到日（当日的零点零分零秒）**

**select to\_char(trunc(sysdate),'yyyymmdd hh24:mi:ss') from dual;** 20111229 00:00:00// **--默认截取到日（当日的零点零分零秒）**

**select trunc(sysdate,'hh24') from dual;**2011-12-29 16:00:00//**--截取到小时（当前小时，零分零秒）**

**select trunc(sysdate,'mi') from dual;**2011-12-29 16:24:00//**--截取到分（当前分，零秒）**

**select trunc(sysdate,'mi') from dual;**2011-12-29 16:24:00//**--报错 直接使用sysdate即可。**

##### 获取本周、本月、本季、本年的第一天和最后一天

--本周

select trunc(sysdate, 'd') + 1 from dual;

select trunc(sysdate, 'd') + 7 from dual;

--本月

select trunc(sysdate, 'mm') from dual;

select last\_day(trunc(sysdate)) from dual;

--本季

select trunc(sysdate, 'Q') from dual;

select add\_months(trunc(sysdate, 'Q'), 3) - 1 from dual;

--本年

select trunc(sysdate, 'yyyy') from dual;

select add\_months(trunc(sysdate, 'yyyy'), 12) - 1 from dual;

#### 字符串函数

##### **SUBSTR**

**语法: SUBSTR（string,a[,b]）**

**功能:** 返回从**a位置**开始的**b个字符长**的string的一个子字符串. **第一个字符的位置是1**. 如 果a是0,那么它就被认为从第一个字符开始.如果是正数,返回字符是从左边向右边进行计算的.如果b是负数,那么返回的字符是从string的末尾开始从 右向左进行计算的.如果b不存在,那么它将缺省的设置为整个字符串.如果b小于1,那么将返回NULL.如果a或b使用了浮点数,那么该数值将在处理进行 以前首先被转为一个整数.

substr(字符串,截取开始位置,截取长度) //返回截取的字

substr('Hello World',0,1) //返回结果为 'H'  \*从字符串第一个字符开始截取长度为1的字符串

substr('Hello World',1,1) //返回结果为 'H'  \*0和1都是表示截取的开始位置为第一个字符

substr('Hello World',2,4) //返回结果为 'ello'

substr('Hello World',-3,3)//返回结果为 'rld' \*负数(-i)表示截取的开始位置为字符串右端向左数第i个字符

##### **INSTR**

**语法: INSTR（string1, string2[a,b]）**

**功能:** 得到在**string1**中包含string2的位置. string1时从左边开始检查的,开始的位置为a,如果a是一个负数,那么string1是从右边开始进行扫描的.第b次出现的位置将被返回. a和b都缺省设置为1,这将会返回在string1中第一次出现string2的位置.如果string2在a和b的规定下没有找到,那么返回0.位置的 计算是相对于string1的开始位置的,不管a和b的取值是多少.

SELECT instr('syranmo','s') FROM dual; -- 返回 1

SELECT instr('syranmo','ra') FROM dual;  -- 返回 3

SELECT instr('syranmo','a',1,2) FROM dual;  -- 返回 0

SELECT instr('syranmo','an',-1,1) FROM dual;  -- 返回 4

|  |
| --- |
| 如果我有一份资料，上面都是一些员工的工号（字段：CODE），可是我现在要查询出他们的所有员工情况，例如名字，部门，职业等等，这里举例是两个员工，工号分别是’A10001′，’A10002′，其中假设staff是员工表，那正常的做法就如下：  　　SELECT  code , name , dept, occupation FROM staff  WHERE code IN ('A10001','A10002');  　　或者：  　　SELECT  code , name , dept, occupation FROM staff  WHERE code = 'A10001' OR code = 'A10002';  　　有时候员工比较多，我们对于那个’觉得比较麻烦，于是就想，可以一次性导出来么？这时候你就可以用instr函数，如下：  　　SELECT  code , name , dept, occupation FROM staff  WHERE instr('A10001,A10002',code)>0;  查询出来结果一样，这样前后只用到两次单引号，相对方便点。  　　还有一个用法，如下：  　　SELECT code, name, dept, occupation  FROM staff  WHERE instr(code, '001') > 0;  　　等同于  　　SELECT code, name, dept, occupation  FROM staff  WHERE code LIKE '%001%' ; |

##### **TRANSLATE**

**语法: TRANSLATE(string,from\_str,to\_str)**

**功能:** 返回将所出现的from\_str 中的每个字符替换为to\_str中的相应字符以后的string. TRANSLATE是REPLACE所提供的功能的一个超集.如果from\_str比to\_str长,那么在from\_str中而不在to\_str中而外 的字符将从string中被删除,因为它们没有相应的替换字符. to\_str不能为空.Oracle把空字符串认为是NULL,并且如果TRANSLATE中的任何参数为NULL,那么结果也是NULL.

##### **REPLACE**

**语法：REPLACE（**string，search\_str[,replace\_str]**）**

**功能：**把string中的所有的子字符串search\_str用可选的replace\_str替换，如果没有指定replace\_str，所有的string中的子字符串search\_str都将被删除。**REPLACE**是**TRANSLATE**所提供的功能的一个子集。

##### **LENGTH/LENGTHB**

length(string)计算string所占的字符长度.如果string是NULL,返回结果是NULL,而不是0.

lengthb(string)计算string所占的字节长度.如果string是NULL,返回结果是NULL,而不是0.

select length('我') from dual --返回1

select lengthb('我') from dual --返回2

select length('AB') from dual --返回2

select lengthb('AB') from dual --返回2

通过这个示例，我们可以看出来，Length 和 Lengthb 函数的一个重要用处，就是用来判断记录值里是否有中文内容。

如果有中文，那么Length() != Lengthb()

如果没有中文，那么Length() == Lengthb()

##### **UPPER/LOWER**

**UPPER:** 返回大写的string.不是字母的字符不变.如果string是CHAR数据类型的,那么结果也是CHAR类型的.如果string是VARCHAR2类型的,那么结果也是VARCHAR2类型的.

**LOWER:** 返回小写的string.不是字母的字符不变.如果string是CHAR数据类型的,那么结果也是CHAR类型的.如果string是VARCHAR2类型的,那么结果也是VARCHAR2类型的.

##### **CONCAT**

**语法：** CONCAT（string1,string2）

**功能：**返回string1，并且在后面连接string2。

##### **CHR /ASCII**

chr()函数将ASCII码转换为字符：字符 –》 ASCII码；

ascii()函数将字符转换为ASCII码：ASCII码 –》 字符；

SQL> select chr(65) from dual;//A

SQL> select ASCII(A) from dual;//65

#### 数学函数

##### FLOOR/CEIL

Floor向下取整（floor 意为地板；数学中的地板是无穷小 天花板是无穷大）

select floor(3.1415927) from dual;//3

select floor(-3.1415927) from dual;//-4

ceil向上取整（ceil意为天花板；数学中的地板是无穷小 天花板是无穷大）

select ceil(3.1415927) from dual;//4

select ceil(-3.1415927) from dual;//-3

##### ROUND

ROUND( number[, decimal\_places] )

number : 需四舍五入处理的数值

decimal\_places : 四舍五入 , 小数取几位 ( 预设为 0 )

select round(123.456, 0) from dual; 回传 123

select round(123.456, 1) from dual; 回传 123.5

select round(123.456, 2) from dual; 回传 123.46

select round(123.456, 3) from dual; 回传 123.456

select round(-123.456, 2) from dual; 回传 -123.46

##### ABS

求绝对值

##### MOD

求整数相除的余数

select mod(3,4) from dual; //3

select mod(8,4) from dual; //0

select mod(9,4) from dual; //1

##### Trunc

trunc(x [,y]),其中如果没有指定y,则对x在0位小数进行截断,

如果y是负数, 则对x在小数点左边的第|y|位处进行截断。

select trunc(5.75) from dual //5

select trunc(5.75,1) from dual//5.7

select trunc(5.75,-1) from dual//0

为什么select trunc(5.75,-1) from dual ==0？

5.75看成 0005.75.000 从小数点左边第一位往左边截,不是0吗；切记小数点是第0位，

#### 分析函数

##### COUNT

|  |
| --- |
| Select count(\*) from emp; //14  Select count(mgr) from emp;//13  结论：忽略了其中的空值行。  Max/min/sum/avg 同样如此 |
| Select count(deptno) from emp;  将会得到错误的结果。因为实际上有很多重复的值也被计算在内。  为了找到正确的答案，你应该这样写：  Select count(distinct deptno) from emp; |

##### MAX

##### MIN

##### SUM

##### AVG

##### row\_number

row\_number() over(order by column)按column排序，最好返回每条记录返回一个序列号；

row\_number()over(partition by col1 order by col2)表示根据col1分组，在分组内部根据col2排序，最好返回每条记录返回一个序列号；另：partition是隔开、分开、区块的意思。

序列号从1开始。

在使用over等开窗函数时，over里头的分组及排序的执行晚于“where，group by，order by”的执行。

举例：

ID NAME                           DATE1  
---------- ------------------------------ ------------------  
       101 aaa                                  09-SEP-13

       101 bbb                                 10-SEP-13

       101 ccc                                  11-SEP-13

       102 ddd                                  08-SEP-13

       102 eee                                  11-SEP-13

SELECT ID,NAME,DATE1,ROW\_NUMBER() OVER(partition by ID order by DATE1 desc) as RN FROM T1;

ID NAME               DATE1               RN  
---------- ------------------------------ ------------------ ----------  
       101 ccc                  11-SEP-13              1  
       101 bbb                  10-SEP-13              2  
       101 aaa                  09-SEP-13              3  
       102 eee                  11-SEP-13              1  
       102 ddd                  08-SEP-13              2

#### 判断函数

##### Decode

DECODE(value,if1,then1,if2,then2,if3,then3,...,else)，表示如果value等于if1时，DECODE函数的结果返回then1,..., 如果不等于任何一个if值，则返回else。

##### case when

简单Case函数

CASE sex

WHEN '1' THEN '男'

WHEN '2' THEN '女'

ELSE '其他' END

--Case搜索函数

CASE WHEN sex = '1' THEN '男'

WHEN sex = '2' THEN '女'

ELSE '其他' END

简单Case函数的写法相对比较简洁，但是和Case搜索函数相比，功能方面会有些限制，比如写判断式。还有一个需要注意的问题，Case函数只返回第一个符合条件的值，剩下的Case部分将会被自动忽略。

--比如说，下面这段SQL，你永远无法得到“第二类”这个结果

CASE WHEN col\_1 IN ( 'a', 'b') THEN '第一类'

WHEN col\_1 IN ('a')  THEN '第二类'

ELSE'其他' END

##### nvl

**格式：NVL(expr1,expr2)**

含义是：如果oracle第一个参数为空那么显示第二个参数的值，如果第一个参数的值不为空，则显示第一个参数本来的值。

##### **NULLIF**

**格式：NULLIF(exp1,expr2)**

含义是：**如果exp1和exp2相等则返回空(NULL)，否则返回第一个值。**

#### raise\_application\_error

raise\_application\_error(error\_number,message[,[truefalse]])

自定义不正确消息

error\_number用于定义不正确号，该不正确号必须在-20000到-20999之间的负整数；

message用于指定不正确消息，并且该消息的长度无法超过2048字节；

true，则该不正确会被放在先前不正确堆栈中

false（默认值）则会替代先前所有不正确

### 自定义函数

我们进入scott用户执行下列语句

select username from user\_users;

--输出 scott

#### 无参函数的创建

|  |
| --- |
| create or replace function test1  return varchar2  is  name varchar2(50);  begin  select username into name from user\_users;  return name;  end;  我们在来看看在plsql中执行我们的函数：  执行下列语句：  declare  username varchar2(50);  begin  username:=test1;  dbms\_output.put\_line(username);  end;  -- 输出：scott！  执行下列语句  declare  username varchar2(50);  begin  username:=test1();  dbms\_output.put\_line(username);  end;  -- 依旧输出：scott！ |
| 我们修改上面的函数  create or replace function test2()  return varchar2  is  name varchar2(50);  begin  select username into name from user\_users;  return name;  end;  则报错，错误信息如下  Compilation errors for FUNCTION SCOTT.TEST2  Error: PLS-00103: 出现符号 ")"在需要下列之一时：  <an identifier>  <a double-quoted delimited-identifier> current。  总结：无参函数在函数名之后不能加“()”,切记！ |

#### 有参函数

|  |
| --- |
| create or replace function test1(num number)  return varchar2  is  name varchar2(50);  begin  select dname into name from dept where deptno=num;  return name;  exception  when no\_data\_found then  raise\_application\_error(-2001,'你输入的id无效！');  end;  则提示Compilation errors for FUNCTION SCOTT.TEST2  Error: PL/SQL: ORA-00920: 无效的关系运算符。 |
| 我们修改上面的函数如下：  create or replace function test2(num number)  return varchar2  is  name varchar2(50);  begin  select dname into name from dept where deptno:=num;  return name;  exception  when no\_data\_found then  //--错误号 是我们自己指定的，一般指定为20000以上。  raise\_application\_error(-20001,'你输入的id无效！');  end;    我们在来看看在plsql中执行我们的函数：  执行下列语句：  declare  num number(2):=11;  name varchar2(50);  begin  name:=test2(num);  dbms\_output.put\_line(name);  end; |

## 存储过程

### 基本语法

|  |
| --- |
| create or replace procedure proc\_name[(pi\_name1 in type , po\_name2 out type)]  as  v\_name1 type[length];  v\_name2 type[length]:=value2;  v\_name3 tablename.columnname%type;  v\_name4 tablename.columnname%type:=value4;  v\_name5 tablename%rowtype;  begin  …  exception  when … then …;  when others then …;  end; |

### 判断

#### If

if condition1 then  
  …  
elsif condition2 then //可无此分支  
  …  
elsif condition3 then //可无此分支  
  …

else //没有then 可无此分支

…

end if;

### 循环

#### for

|  |
| --- |
| for element in array loop  …  End loop; |
| 举例1  for i in 1..5 loop    dbms\_output.put\_line(i);  end loop;  举例2  userRow t\_user%rowtype;  cursor userRows is  select \* from t\_user;  begin  for userRow in userRows loop      dbms\_output.put\_line(userRow.Id||','||userRow.Name||','||userRows%rowcount);  end loop; |

#### while

|  |
| --- |
| While condition loop  …  End loop |

#### loop

|  |
| --- |
| loop  …  exit when condition;  …  end loop; |

## 序列

|  |
| --- |
| CREATE SEQUENCE seqTest INCREMENT BY **1** -- 每次加1 START WITH **1** -- 从1开始计数 NOMAXVALUE/MAXVALUE **99999999** -- 不设置最大值/设置最大值 NOCYCLE/CYCLE] -- 不循环/循环 NOCACHE/CACHE **10**; --不缓存/缓存；设置缓存cache个序列，如果系统down掉了或者其它情况将会导致序列不连续 |
| CurrVal：返回 sequence的当前值  NextVal：增加sequence的值，然后返回 增加后sequence值 |
| 注意：   * 第一次NEXTVAL返回的是初始值；随后的NEXTVAL会自动增加你定义的INCREMENT BY值，然后返回增加后的值。 * 一次NEXTVAL会增加一次 SEQUENCE的值，所以如果你在同一个语句里面使用多个NEXTVAL，其值就是不一样的。 * **nextval**调用之后才能使用**currval**，否则会出错。 |

## Job

|  |
| --- |
| 查看job queue 后台进程  select name,description from v$bgprocess; |
| 'sys.dbms\_job'包在'all objects'-->'packages'你可以找到该包，在该包下有很多子过程（存储过程），submit存储过程就是用来创建job的。  sys.dbms\_job.submit  使用Submit()存储过程，工作被正常地计划好。  　这个存储过程有五个参数：job、what、next\_date、interval、no\_parse。  　 PROCEDURE submit ( job　　　 OUT binary\_ineger,  　　　　　　　　　　　What　　　IN　varchar2,  　　　　　　　　　　　next\_date IN　date,  　　　　　　　　　　　interval　 IN　varchar2,  　　　　　　　　　　　no\_parse　 IN　booean:=FALSE)  　　job参数是由Submit()过程返回的binary\_ineger。这个值用来唯一标识一个工作。  　　what参数是将被执行的PL/SQL代码块。  　　next\_date参数指识何时将运行这个工作。  　　interval参数何时这个工作将被重执行。  　　no\_parse参数指示此工作在提交时或执行时是否应进行语法分析——TRUE指示此PL/SQL代码在它第一次执行时应进行语法分析，而FALSE指示本PL/SQL代码应立即进行语法分析。  sys.dbms\_job.broken(  job in binary\_integer,  broken in boolen,  next\_date in date default sysdate);  broken值为true时，表示禁用job；false时，表示启用job（也可用run存储过程）。 |
| --建立job   --建立job后默认是执行的   declare  test\_job\_really number;   begin    dbms\_job.submit(test\_job\_really,'test\_jobproce;',sysdate,'sysdate+1/1440');   commit;   end;  或者  declare  job number;  begin  sys.dbms\_job.submit(job => :job,  what => 'PACK\_XZXK\_EVERYDAY.PROC\_DEAL\_XZXK\_EVERYDAY;',  next\_date => to\_date('15-06-2012 01:00:00', 'dd-mm-yyyy hh24:mi:ss'),  interval => 'trunc(sysdate)+1+1/24');  commit;  end; |
| ---停止job  25是建立的job test\_job\_really   begin    dbms\_job.broken(**25**,true);    commit;   end; |
| --启动job  begin  dbms\_job.run(**25**);  commit;  end; |
| --删除job   begin     dbms\_job.remove(**25**);    commit;   end; |

## 触发器

### **触发器类型**

触发器在数据库里以独立的对象存储，它与存储过程和函数不同的是，存储过程与函数需要用户显示调用才执行，而触发器是由一个事件来启动运行。即触发器是当某个事件发生时自动地隐式运行。并且，触发器不能接收参数。 所以运行触发器就叫触发或点火（firing）。ORACLE事件指的是对数据库的表进行的INSERT、UPDATE及DELETE操作或对视图进行类 似的操作。ORACLE将触发器的功能扩展到了触发ORACLE，如数据库的启动与关闭等。所以触发器常用来完成由数据库的完整性约束难以完成的复杂业务 规则的约束，或用来监视对数据库的各种操作，实现审计的功能。

#### **DML触发器**

    ORACLE可以在DML语句进行触发，可以在DML操作前或操作后进行触发，并且可以对每个行或语句操作上进行触发。

#### **替代触发器**

    由于在ORACLE里，不能直接对由两个以上的表建立的视图进行操作。所以给出了替代触发器。它就是ORACLE 8专门为进行视图操作的一种处理方法。

#### **系统触发器**

ORACLE 8i 提供了第三种类型的触发器叫系统触发器。它可以在ORACLE数据库系统的事件中进行触发，如ORACLE系统的启动与关闭等。

### 触发器注意事项

* 一个表上最多可有12个触发器，但同一时间、同一事件、同一类型的触发器只能有一个。并各触发器之间不能有矛盾。
* 在一个表上的触发器越多，对在该表上的DML操作的性能影响就越大。
* 触发器最大为32KB。若确实需要，可以先建立过程，然后在触发器中用CALL语句进行调用。
* **在触发器的执行部分只能用DML语句（SELECT、INSERT、UPDATE、DELETE），不能使用DDL语句（CREATE、ALTER、DROP）**。
* 触发器中不能包含事务控制语句(COMMIT，ROLLBACK，SAVEPOINT)。因为触发器是触发语句的一部分，触发语句被提交、回退时，触发器也被提交、回退了。
* 在触发器主体中调用的任何过程、函数，都不能使用事务控制语句。
* 在触发器主体中不能申明任何Long和blob变量。新值new和旧值old也不能是表中的任何long和blob列。
* 不同类型的触发器(如DML触发器、INSTEAD OF触发器、系统触发器)的语法格式和作用有较大区别。

### 触发器语法

BEFORE 和AFTER指出触发器的触发时序分别为前触发和后触发方式，前触发是在执行触发事件之前触发当前所创建的触发器，后触发是在执行触发事件之后触发当前所创建的触发器。

 FOR EACH ROW选项说明触发器为行触发器。行触发器和语句触发器的区别表现在：行触发器要求当一个DML语句操作影响数据库中的多行数据时，对于其中的每个数据行，只要它们符合触发约束条件，均激活一次触发器；而语句触发器将整个语句操作作为触发事件，当它符合约束条件时，激活一次触发器。当省略FOR EACH ROW 选项时，BEFORE 和AFTER 触发器为语句触发器，而INSTEAD OF 触发器则只能为行触发器。

  REFERENCING 子句说明相关名称，在行触发器的PL/SQL块和WHEN 子句中可以使用相关名称参照当前的新、旧列值，默认的相关名称分别为OLD和NEW。触发器的PL/SQL块中应用相关名称时，必须在它们之前加冒号 (:)，但在WHEN子句中则不能加冒号。

WHEN 子句说明触发约束条件。Condition 为一个逻辑表达时，其中必须包含相关名称，而不能包含查询语句，也不能调用PL/SQL 函数。WHEN 子句指定的触发约束条件只能用在BEFORE 和AFTER 行触发器中，不能用在INSTEAD OF 行触发器和其它类型的触发器中。

 当一个基表被修改( INSERT, UPDATE, DELETE)时要执行的存储过程，执行时根据其所依附的基表改动而自动触发，因此与应用程序无关，用数据库触发器可以保证数据的一致性和完整性。

每张表最多可建立12 种类型的触发器，它们是:

BEFORE INSERT

BEFORE INSERT FOR EACH ROW

AFTER INSERT

AFTER INSERT FOR EACH ROW

BEFORE UPDATE

BEFORE UPDATE FOR EACH ROW

AFTER UPDATE

AFTER UPDATE FOR EACH ROW

BEFORE DELETE

BEFORE DELETE FOR EACH ROW

AFTER DELETE

AFTER DELETE FOR EACH ROW

1.       执行 BEFORE语句级触发器;

2.       对与受语句影响的每一行：

* 执行 BEFORE行级触发器
* 执行 DML语句
* 执行 AFTER行级触发器

3.       执行 AFTER语句级触发器

### DML触发器

**语法：**

CREATE [OR REPLACE] TRIGGER trigger\_name

{BEFORE | AFTER

{

{INSERT | DELETE | UPDATE [OF column [, column …]]}

|

{INSERT | DELETE | UPDATE [OF column [, column …]]}

}

ON [schema.]table\_name | [schema.]view\_name

}

[REFERENCING {OLD [AS] old | NEW [AS] new| PARENT as parent}]

[FOR EACH ROW ]

[WHEN condition]

PL/SQL\_BLOCK | CALL procedure\_name;

DML触发器基本要点

* 触发时机：指定触发器的触发时间。如果指定为BEFORE，则表示在执行DML操作之前触发，以便防止某些错误操作发生或实现某些业务规则；如果指定为AFTER，则表示在执行DML操作之后触发，以便记录该操作或做某些事后处理。
* 触发事件：引起触发器被触发的事件，即DML操作（INSERT、UPDATE、DELETE）。既可以是单个触发事件，也可以是多个触发事件的组合（只能使用OR逻辑组合，不能使用AND逻辑组合）。
* 条件谓词：当在触发器中包含多个触发事件（INSERT、UPDATE、DELETE）的组合时，为了分别针对不同的事件进行不同的处理，需要使用ORACLE提供的如下条件谓词。

1）。INSERTING：当触发事件是INSERT时，取值为TRUE，否则为FALSE。

2）。UPDATING [（column\_1,column\_2,…,column\_x）]：当触发事件是UPDATE      时，如果修改了column\_x列，则取值为TRUE，否则为FALSE。其中column\_x是可选的。

3）。DELETING：当触发事件是DELETE时，则取值为TRUE，否则为FALSE。

* 触发对象：指定触发器是创建在哪个表、视图上。

触发类型：是语句级还是行级触发器。

触发条件：由WHEN子句指定一个逻辑表达式，只允许在行级触发器上指定触发条件，指定UPDATING后面的列的列表。

问题：当触发器被触发时，要使用被插入、更新或删除的记录中的列值，有时要使用操作前、后列的值.

实现:  :NEW 修饰符访问操作完成后列的值

        :OLD 修饰符访问操作完成前列的值

特性INSERT UPDATE DELETE

OLD NULL 实际值 实际值

NEW 实际值 实际值 NULL

|  |
| --- |
| **例1:** 建立一个触发器, 当职工表 emp 表被删除一条记录时，把被删除记录写到职工表删除日志表中去  CREATE OR REPLACE TRIGGER tr\_del\_emp     BEFORE DELETE --指定触发时机为删除操作前触发    ON scott.emp     FOR EACH ROW   --说明创建的是行级触发器  BEGIN    --将修改前数据插入到日志记录表 del\_emp ,以供监督使用。    INSERT INTO emp\_his(deptno , empno, ename , job ,mgr , sal , comm , hiredate )        VALUES( :old.deptno, :old.empno, :old.ename , :old.job,:old.mgr, :old.sal, :old.comm, :old.hiredate ); END; |
| **例2：**限制对Departments表修改（包括INSERT,DELETE,UPDATE）的时间范围，即不允许在非工作时间修改departments表。  CREATE OR REPLACE TRIGGER tr\_dept\_time BEFORE INSERT OR DELETE OR UPDATE  ON departments BEGIN  IF (TO\_CHAR(sysdate,'DAY') IN ('星期六', '星期日'))  OR (TO\_CHAR(sysdate, 'HH24:MI') NOT BETWEEN '08:30' AND '18:00') THEN      RAISE\_APPLICATION\_ERROR(-**20001**, '不是上班时间，不能修改departments表');  END IF; END; |
| **例3：**当部门号为80且（修改其薪资或评价或删除）记录时触发器操作。  CREATE OR REPLACE TRIGGER tr\_emp\_sal\_comm BEFORE UPDATE OF salary, commission\_pct        OR DELETE ON HR.employees FOR EACH ROW WHEN (old.department\_id = **80**) BEGIN  CASE      WHEN UPDATING ('salary') THEN         IF :NEW.salary < :old.salary THEN            RAISE\_APPLICATION\_ERROR(-**20001**, '部门80的人员的工资不能降');         END IF;      WHEN UPDATING ('commission\_pct') THEN          IF :NEW.commission\_pct < :old.commission\_pct THEN            RAISE\_APPLICATION\_ERROR(-**20002**, '部门80的人员的奖金不能降');         END IF;      WHEN DELETING THEN           RAISE\_APPLICATION\_ERROR(-**20003**, '不能删除部门80的人员记录');      END CASE; END; |
| **例4：**利用行触发器实现级联更新。在修改了主表regions中的region\_id之后（AFTER），级联的、自动的更新子表countries表中原来在该地区的国家的region\_id。  CREATE OR REPLACE TRIGGER tr\_reg\_cou AFTER update OF region\_id ON regions FOR EACH ROW BEGIN  DBMS\_OUTPUT.PUT\_LINE('旧的region\_id值是'||:old.region\_id                   ||'、新的region\_id值是'||:new.region\_id);  UPDATE countries SET region\_id = :new.region\_id  WHERE region\_id = :old.region\_id; END; |
| **例5：**在触发器中调用过程。  CREATE OR REPLACE PROCEDURE add\_job\_history  ( p\_emp\_id          job\_history.employee\_id%type    , p\_start\_date      job\_history.start\_date%type   , p\_end\_date        job\_history.end\_date%type    , p\_job\_id          job\_history.job\_id%type    , p\_department\_id   job\_history.department\_id%type    ) IS BEGIN  INSERT INTO job\_history (employee\_id, start\_date, end\_date,                            job\_id, department\_id)   VALUES(p\_emp\_id, p\_start\_date, p\_end\_date, p\_job\_id, p\_department\_id); END add\_job\_history;  --创建触发器调用存储过程... CREATE OR REPLACE TRIGGER update\_job\_history  AFTER UPDATE OF job\_id, department\_id ON employees  FOR EACH ROW BEGIN  add\_job\_history(:old.employee\_id, :old.hire\_date, sysdate,                   :old.job\_id, :old.department\_id); END; |

### 替代触发器

**语法是:**

CREATE [OR REPLACE] TRIGGER trigger\_name  
INSTEAD OF  
{INSERT | DELETE | UPDATE [OF column [, column …]]}  
[OR {INSERT | DELETE | UPDATE [OF column [, column …]]}...]  
ON [schema.] view\_name --只能定义在视图上  
[REFERENCING {OLD [AS] old | NEW [AS] new| PARENT as parent}]  
[FOR EACH ROW ] --因为INSTEAD OF触发器只能在行级上触发,所以没有必要指定  
[WHEN condition]  
PL/SQL\_block | CALL procedure\_name;

只能对视图和对象视图建立INSTEAD OF触发器，而不能对表、模式和数据库建立INSTEAD OF 触发器。

|  |
| --- |
| **例1：**  CREATE OR REPLACE VIEW emp\_view AS  SELECT deptno, count(\*) total\_employeer, sum(sal) total\_salary  FROM emp GROUP BY deptno;  在此视图中直接删除是非法：  SQL>DELETE FROM emp\_view WHERE deptno=**10**; DELETE FROM emp\_view WHERE deptno=**10**  ERROR 位于第 1 行:  ORA-01732: 此视图的数据操纵操作非法  但是我们可以创建INSTEAD\_OF触发器来为 DELETE 操作执行所需的处理，即删除EMP表中所有基准行：  **CREATE OR REPLACE TRIGGER emp\_view\_delete    INSTEAD OF DELETE ON emp\_view FOR EACH ROW BEGIN    DELETE FROM emp WHERE deptno= :old.deptno; END emp\_view\_delete;**  **DELETE FROM emp\_view WHERE deptno=10;** |
| **例2：**创建复杂视图，针对INSERT操作创建INSTEAD OF触发器，向复杂视图插入数据。  创建视图：  CREATE OR REPLACE FORCE VIEW "HR"."V\_REG\_COU" ("R\_ID", "R\_NAME", "C\_ID", "C\_NAME") AS  SELECT r.region\_id,     r.region\_name,     c.country\_id,     c.country\_name  FROM regions r,     countries c  WHERE r.region\_id = c.region\_id;  创建触发器：  **CREATE OR REPLACE TRIGGER "HR"."TR\_I\_O\_REG\_COU" INSTEAD OF  INSERT ON v\_reg\_cou FOR EACH ROW DECLARE v\_count NUMBER; BEGIN  SELECT COUNT(\*) INTO v\_count FROM regions WHERE region\_id = :new.r\_id;  IF v\_count = 0 THEN     INSERT INTO regions       (region\_id, region\_name       ) VALUES       (:new.r\_id, :new.r\_name       );  END IF;   SELECT COUNT(\*) INTO v\_count FROM countries WHERE country\_id = :new.c\_id;  IF v\_count = 0 THEN     INSERT     INTO countries       (         country\_id,         country\_name,         region\_id       )       VALUES       (         :new.c\_id,         :new.c\_name,         :new.r\_id       );  END IF; END;** |

### 系统触发器

ORACLE10G提供的系统事件触发器可以在DDL或数据库系统上被触发。DDL指的是数据定义语言，如CREATE 、ALTER及DROP 等。而数据库系统事件包括数据库服务器的启动或关闭，用户的登录与退出、数据库服务错误等。创建系统触发器的语法如下：

**创建触发器的一般语法是:**

CREATE OR REPLACE TRIGGER [sachema.]trigger\_name  
{BEFORE|AFTER}   
{ddl\_event\_list | database\_event\_list}  
ON { DATABASE | [schema.]SCHEMA }  
[WHEN condition]  
PL/SQL\_block | CALL procedure\_name;

其中: ddl\_event\_list：一个或多个DDL 事件，事件间用 OR 分开；

database\_event\_list：一个或多个数据库事件，事件间用 OR 分开；

下面给出系统触发器的种类和事件出现的时机（前或后）：

|  |  |  |
| --- | --- | --- |
| **事件** | **允许的时机** | **说明** |
| STARTUP | AFTER | 启动数据库实例之后触发 |
| SHUTDOWN | BEFORE | 关闭数据库实例之前触发（非正常关闭不触发） |
| SERVERERROR | AFTER | 数据库服务器发生错误之后触发 |
| LOGON | AFTER | 成功登录连接到数据库后触发 |
| LOGOFF | BEFORE | 开始断开数据库连接之前触发 |
| CREATE | BEFORE，AFTER | 在执行CREATE语句创建数据库对象之前、之后触发 |
| DROP | BEFORE，AFTER | 在执行DROP语句删除数据库对象之前、之后触发 |
| ALTER | BEFORE，AFTER | 在执行ALTER语句更新数据库对象之前、之后触发 |
| DDL | BEFORE，AFTER | 在执行大多数DDL语句之前、之后触发 |
| GRANT | BEFORE，AFTER | 执行GRANT语句授予权限之前、之后触发 |
| REVOKE | BEFORE，AFTER | 执行REVOKE语句收权限之前、之后触犯发 |
| RENAME | BEFORE，AFTER | 执行RENAME语句更改数据库对象名称之前、之后触犯发 |
| AUDIT NOAUDIT | BEFORE，AFTER | 执行AUDIT**或**NOAUDIT进行审计或停止审计之前、之后触发 |

|  |
| --- |
| **例1：**创建ddl触发器，存放有关事件信息。  --创建用于记录事件用的表  CREATE TABLE ddl\_event (crt\_date timestamp PRIMARY KEY,  event\_name VARCHAR2(**20**),   user\_name VARCHAR2(**10**),  obj\_type VARCHAR2(**20**),  obj\_name VARCHAR2(**20**));  --创建触犯发器 CREATE OR REPLACE TRIGGER tr\_ddl AFTER DDL ON SCHEMA BEGIN    INSERT INTO ddl\_event VALUES    (systimestamp,ora\_sysevent, ora\_login\_user,      ora\_dict\_obj\_type, ora\_dict\_obj\_name); END tr\_ddl; |
| **例2：**创建登录、退出触发器  CREATE TABLE log\_event (user\_name VARCHAR2(**10**),  address VARCHAR2(**20**),   logon\_date timestamp,  logoff\_date timestamp);   --创建登录触发器 CREATE OR REPLACE TRIGGER tr\_logon AFTER LOGON ON DATABASE BEGIN    INSERT INTO log\_event (user\_name, address, logon\_date)    VALUES (ora\_login\_user, ora\_client\_ip\_address, systimestamp); END tr\_logon; --创建退出触发器 CREATE OR REPLACE TRIGGER tr\_logoff BEFORE LOGOFF ON DATABASE BEGIN    INSERT INTO log\_event (user\_name, address, logoff\_date)    VALUES (ora\_login\_user, ora\_client\_ip\_address, systimestamp); END tr\_logoff; |

### **触发器谓词**

begin   
if **inserting** then   
-----   
elsif updating then   
-----   
elsif deleting then   
------   
end if;

## 其他

### Having和where

WHERE 搜索条件在进行分组操作之前应用；而 HAVING 搜索条件在进行分组操作之后应用。

HAVING 语法与 WHERE 语法类似但 HAVING 可以包含聚合函数。HAVING 子句可以引用选择列表中显示的任意项。

|  |
| --- |
| 举例1:  按产品 ID 对 SalesOrderDetail 进行了分组，并且只包含那些订单合计大于 $1,000,000 且其平均订单数量小于 3 的产品组：  SELECT ProductID, AVG(OrderQty) AS AverageQuantity, SUM(LineTotal) AS Total  FROM Sales.SalesOrderDetail  GROUP BY ProductID  HAVING SUM(LineTotal) > $1000000.00  AND AVG(OrderQty) < 3 ; |

### 删除重复记录

|  |
| --- |
| 查找表中多余的重复记录，重复记录是根据单个字段（Id）来判断  select \* from table where id in(select id from table group by id having cout(id)>1);  或  select t1.\* from table t1,table t2 where t1.id=t2.id and t1.rowid<t2.rowid  删除表中的重复记录，重复记录是根据单个字段（Id）来判断，只留有rowid最小的记录  DELETE from 表 WHERE (id) IN ( SELECT id FROM 表 GROUP BY id HAVING COUNT(id) > 1) AND ROWID NOT IN (SELECT MIN(ROWID) FROM 表 GROUP BY id HAVING COUNT(\*) > 1);  查找表中多余的重复记录，重复记录是根据多个字段判断  select from table where (col1,col2 )in(select col1,col2 from table  group by co1,co2 having cout(co1,co2)>1);  删除表中多余的重复记录，重复记录是根据多个字段判断，只留有rowid最小的记录  delete from 表 a where (a.Id,a.seq) in (select Id,seq from 表 group by Id,seq having count(\*) > 1) and rowid not in (select min(rowid) from 表 group by Id,seq having count(\*)>1) |

### lyl项目中使用到的函数和存储过程脚本

|  |
| --- |
| --1、函数  --1.1、返回某日期 年月日字符串  create or replace function func\_getYearmonthday(rq in DATE) return varchar2 is  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  用途：返回某日期 年月日字符串  作者：崔红涛  创建日期：2014-11-28  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  yearandmonthandday varchar2(8);  BEGIN  select to\_char(rq,'yyyyMMdd') into yearandmonthandday from dual ;  return(yearandmonthandday);  end func\_getYearmonthday;  --1.2、计算类似'aaa,bbb,ccc'是否包含'bbb'这样的子串  create or replace function func\_existsubstr(p\_string in varchar2,p\_delimiter in varchar2,p\_target in varchar2)  return number is  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  用途：计算类似'aaa,bbb,ccc'是否包含'bbb'这样的子串  作者：崔红涛  创建日期：2014-11-28  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  v\_pos1 number := 1;--上一次出现p\_delimiter的位置  v\_pos2 number := 1;--本次出现p\_delimiter的位置  v\_substr varchar2(32);--分割出的子串  BEGIN  if p\_string=null then  return 0;  end if;  if p\_delimiter=null then  return 0;  end if;  if p\_target=null then  return 0;  end if;    --针对'a'中是否含有'a'的情况  if v\_pos1= length(p\_string) then  if p\_target=p\_string then  return 1;  end if;  return 0;  end if;    while v\_pos1 < length(p\_string) loop  v\_pos2:=instr(p\_string, p\_delimiter, v\_pos1, 1);--p\_string中 从第v\_pos1个字符开始，第一次（1）出现p\_delimiter的位置  if v\_pos2=0 then --0表示未出现，表示是最后一个分割符后的子串  v\_substr:=SUBSTR(p\_string, v\_pos1);  if p\_target=v\_substr then  return 1;  end if;  return 0;  end if;  v\_substr:=substr(p\_string, v\_pos1, v\_pos2 - v\_pos1);--从串p\_string的v\_pos1开始截取 v\_pos2 - v\_pos1个长度  if(p\_target=v\_substr) then  return 1;  end if;  v\_pos1:=v\_pos2+1;  end loop;    end func\_existsubstr;  --2、存储过程  --2.1、包创建  create or replace package pck\_today is  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：计算每日每个上线景区每种门票的优惠价  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/    /\*\*\*\*\*\*\*\*\*门票价格封装\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  type recorder\_price is record(  oprice t\_scenic\_ticket.oprice%type,--门票门市价  dprice t\_scenic\_ticket.dprice%type,--门票折价  benefitdprice t\_scenic\_ticket.dprice%type ,--计算后的优惠价  startrq t\_scenic\_benefit.startrq%type,--优惠开始日期  endrq t\_scenic\_benefit.endrq%type,--优惠结束日期  state char(2) default '0', -- 0:未计算  -- 1：今日没有优惠（有效，使用benefitdprice=dprice）  -- 2：今日有一个优惠记录（有效，计算benefitdprice）  -- -1：今日有两个或两个以上记录（无效）  -- -2：其他异常  message varchar2(500)--计算状态信息  );    /\*\*\*\*\*\*\*\*\*门票价格数组封装\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  type array\_price is table of recorder\_price index by t\_scenic\_ticket.id%type;    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：查询每日上线景区  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  procedure proc\_today\_scenic;      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：1、根据入参 scenicid 查询today门票  2、根据ticketid 执行proc\_today\_benefit 获取门票的dprice  3、组装门票信息inesrt 到t\_today\_ticket  4、取每个景区的最小drpice，更新t\_scenic表  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  procedure proc\_today\_ticket(p\_scenicid in varchar2, --景区id  p\_scenicname in varchar2); --景区name      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：1、根据入参 ticketid 计算今日的折后价  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  procedure proc\_today\_benefit(p\_ticketid in varchar2, --景区id  p\_recorder\_price in out recorder\_price);---门票价格封装      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：1、取今日门票最小值  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  procedure proc\_today\_min(p\_array\_price in array\_price, --门票集合  p\_oprice in out number,--最小折扣价对应的门票原始价  p\_dprice in out number,--最小折扣价  p\_isshow in out char );--是否web显示        end pck\_today;  --2.2、包实现  create or replace package body pck\_today is  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：查询每日上线景区  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  procedure proc\_today\_scenic  is  c\_online t\_scenic.isonline%type:=1;--上线  begin  for rec in(select  id scenicid ,  name scenicname  from t\_scenic  where isonline =c\_online)loop  proc\_today\_ticket(rec.scenicid,rec.scenicname);  end loop;  end proc\_today\_scenic;        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：1、根据入参 scenicid 查询today门票  2、根据ticketid 执行proc\_today\_benefit 获取门票的dprice  3、组装门票信息inesrt 到t\_today\_ticket  4、取每个景区的最小drpice，更新t\_scenic表  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  procedure proc\_today\_ticket(p\_scenicid in varchar2,--景区id  p\_scenicname in varchar2)--景区name    is        v\_arrays\_price array\_price;--门票价格数组变量声明  v\_arrays\_index t\_scenic\_ticket.id%type;--门票价格数组下标    v\_minoprice t\_scenic.oprice%type:=0;--最小折价对应得门票价  v\_mindprice t\_scenic.dprice%type:=0;--最小折价  v\_isshow t\_scenic.isshow%type:=0;--是否web显示  begin    --1、  for rec in(select  t.id ticketid,--门票id  t.name ticketname, --门票名称  t.startrq,--门票有效期  t.endrq, --门票有效期  t.oprice,--门票门市价  t.dprice,--给王道乐途的价格  t.ptype,--支付方式  t.fkqx,--在线支付付款期限  t.qpqx,--取票期限  t.maxdaysell,--单日最大允许销售量  t.maxdaypersonsell,--同一用户单日最大允许订票数  t.maxsingledeal ,--同一订单最大允许订票数  t.isgd,--是否允许改单(0,1)  t.wfdpj,--web返点评奖  t.mfdpj,--手机返点评奖  t.wfxjq,--web返现金券  t.mfxjq,--手机返现金券  t.mdxjq,--最大抵现价券  t.summary,--简述  t.tnotice --须知  from t\_scenic\_ticket t  where scenicid =p\_scenicid  and sysdate between t.startrq and t.endrq  )loop    v\_arrays\_price(rec.ticketid).oprice:=rec.oprice;  v\_arrays\_price(rec.ticketid).dprice:=rec.dprice;  v\_arrays\_price(rec.ticketid).startrq:=rec.startrq;  v\_arrays\_price(rec.ticketid).endrq:=rec.endrq;  proc\_today\_benefit(rec.ticketid,v\_arrays\_price(rec.ticketid));    begin  update t\_today\_ticket t  set  t.oprice=v\_arrays\_price(rec.ticketid).oprice,  t.dprice=v\_arrays\_price(rec.ticketid).benefitdprice,  t.startrq=v\_arrays\_price(rec.ticketid).startrq,  t.endrq=v\_arrays\_price(rec.ticketid).endrq,  t.exstate=v\_arrays\_price(rec.ticketid).state,  t.extime=sysdate,  t.exmessage=v\_arrays\_price(rec.ticketid).message  where t.ticketid=rec.ticketid  and t.today= func\_getYearmonthday(sysdate);  if sql%notfound then  --插入记录  insert into t\_today\_ticket (  ticketid,today,scenicid,scenicname,ticketname,  startrq,endrq,oprice,dprice,  ptype,fkqx,qpqx,maxdaysell,maxdaypersonsell,maxsingledeal,  isgd,wfdpj,mfdpj,wfxjq,mfxjq,mdxjq,  summary,tnotice,exstate,extime,exmessage  )values(  rec.ticketid,func\_getYearmonthday(sysdate),p\_scenicid,p\_scenicname,rec.ticketname,  v\_arrays\_price(rec.ticketid).startrq,v\_arrays\_price(rec.ticketid).endrq,v\_arrays\_price(rec.ticketid).oprice,v\_arrays\_price(rec.ticketid).benefitdprice,  rec.ptype,rec.fkqx,rec.qpqx,rec.maxdaysell,rec.maxdaypersonsell,rec.maxsingledeal,  rec.isgd,rec.wfdpj,rec.mfdpj,rec.wfxjq,rec.mfxjq,rec.mdxjq,  rec.summary,rec.tnotice,v\_arrays\_price(rec.ticketid).state,sysdate,v\_arrays\_price(rec.ticketid).message  );  end if;  end;  end loop;      --2、  proc\_today\_min(v\_arrays\_price,v\_minoprice,v\_mindprice,v\_isshow);    --dbms\_output.put\_line(isshow);    update t\_scenic t set  t.oprice=v\_minoprice,  t.dprice=v\_mindprice,  t.isshow=v\_isshow  where t.id=p\_scenicid;      commit;      --dbms\_output.put\_line('景区\_'||p\_scenicid||'今日门票信息开始');  --dbms\_output.put\_line('门票类型数目为--->'||arrays.count);    --arrays\_index:=arrays.first;  --while (arrays.exists(arrays\_index)) LOOP  --dbms\_output.put\_line('门票类型\_'||arrays\_index||'价格-->'||arrays(arrays\_index).oprice||';'||arrays(arrays\_index).dprice);  --arrays\_index:=arrays.next(arrays\_index) ;  --end loop;  --dbms\_output.put\_line('景区\_'||p\_scenicid||'今日门票信息结束');  --dbms\_output.put\_line('');  -- dbms\_output.put\_line('');        end proc\_today\_ticket;    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：1、根据入参 ticketid 计算今日的折后价  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  procedure proc\_today\_benefit(p\_ticketid in varchar2, --门票id  p\_recorder\_price in out recorder\_price)---门票价格封装    is      v\_count number:=0;    --err\_code varchar(20);--sql错误码  --err\_text varchar(100);--sql错误信息    v\_benefitname t\_scenic\_benefit.name%type;  v\_benefitstartrq t\_scenic\_benefit.startrq%type;  v\_benefitendrq t\_scenic\_benefit.endrq%type;  v\_benefittype t\_scenic\_benefit.type%type;  v\_benefitvalue t\_scenic\_benefit.value%type;  begin    select count(1) into v\_count from t\_scenic\_benefit t where 1=func\_existsubstr(t.ticketids,',',p\_ticketid)  and sysdate between t.startrq and t.endrq;    --1、  if v\_count >1 then  p\_recorder\_price.state:='-1';  p\_recorder\_price.benefitdprice:=p\_recorder\_price.dprice;--使用原折价  p\_recorder\_price.message:='该门票在<'||func\_getyearmonthday(sysdate)||'>有<'||v\_count||'>条优惠记录';  return;  end if;    --2、  if v\_count =0 then  p\_recorder\_price.state:='1';  p\_recorder\_price.benefitdprice:=p\_recorder\_price.dprice;  p\_recorder\_price.message:='该门票在<'||func\_getyearmonthday(sysdate)||'>没有优惠记录';  return;  end if;    --3、  select t.name,t.startrq,t.endrq,t.type, t.value  into v\_benefitname,v\_benefitstartrq,v\_benefitendrq,v\_benefittype,v\_benefitvalue  from t\_scenic\_benefit t  where 1=func\_existsubstr(t.ticketids,',',p\_ticketid)  and sysdate between t.startrq and t.endrq;  p\_recorder\_price.state:='2';  p\_recorder\_price.startrq:=v\_benefitstartrq;  p\_recorder\_price.endrq:=v\_benefitendrq;  p\_recorder\_price.message:='该门票在<'||func\_getyearmonthday(sysdate)||'>有一条优惠记录';  if 1=v\_benefittype then--百分比  p\_recorder\_price.benefitdprice:=p\_recorder\_price.dprice\*(1-v\_benefitvalue);  elsif 2=v\_benefittype then--定值  p\_recorder\_price.benefitdprice:=p\_recorder\_price.dprice-v\_benefitvalue;  end if;  return;    --4、异常  exception when others then  --err\_code := sqlcode;  --err\_text := sqlerrm;  p\_recorder\_price.state:='-2';  p\_recorder\_price.benefitdprice:=p\_recorder\_price.dprice;  p\_recorder\_price.message:='计算benefit出错<'||func\_getyearmonthday(sysdate)||'><'||sqlerrm||'>';        end proc\_today\_benefit;    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  作用：1、取今日门票最小值  作者：崔红涛  创建日期：2014-11-28    修改历史：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  procedure proc\_today\_min(p\_array\_price in array\_price, --门票集合  p\_oprice in out number,--最小折扣价对应的门票原始价  p\_dprice in out number,--最小折扣价  p\_isshow in out char )--是否web显示    is  pos t\_scenic\_ticket.id%type;--数组下标    begin  pos:=p\_array\_price.first;  if p\_array\_price.exists(pos) then  p\_oprice:=p\_array\_price(pos).oprice;  p\_dprice:=p\_array\_price(pos).benefitdprice;  p\_isshow:='1';  pos:=p\_array\_price.next(pos) ;  end if;  while (p\_array\_price.exists(pos)) loop  if(p\_dprice>p\_array\_price(pos).benefitdprice) then  p\_oprice:=p\_array\_price(pos).oprice;  p\_dprice:=p\_array\_price(pos).benefitdprice;  p\_isshow:='1';  end if;  pos:=p\_array\_price.next(pos) ;  end loop;      end proc\_today\_min;      end pck\_today;  --3、job  begin  sys.dbms\_job.submit(job => :job,  what => 'pck\_today.proc\_today\_scenic;',  next\_date => to\_date('29-11-2014 16:30:00', 'dd-mm-yyyy hh24:mi:ss'),  interval => 'TRUNC(SYSDATE+1)+1/24');  commit;  end;  / |

# sql技巧

## 纵横表转换

### 纵表变横表

|  |
| --- |
| 姓名 科目 分数 张三 语文 80 张三 数学 98 张三 英语 65 李四 语文 70 李四 数学 80 李四 英语 90  **期望查询结果：**  姓名 语文 数学 英语 张三 80 98 65 李四 70 80 90 |
| http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gifselect tname as '姓名' ,  http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gifmax(case ttype when '语文' then tscor else **0** end) '语文',  http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gifmax(case ttype when '数学' then tscor else **0** end) '数学',  http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gifmax(case ttype when '英语' then tscor else **0** end) '英语'  http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.giffrom testScore  http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gifgroup by tname |

### 横表变纵表

|  |
| --- |
| 姓名    语文        数学       英语  张三     80         90         70  李四     90         85         95  王五     88         75         90  **期望查询结果：**  姓名     科目     成绩  张三     语文     80  张三     数学     90  张三     英语     70  李四     语文     90  李四     数学     80  李四     英语     99  王五     语文     85  王五     数学     96  王五     英语     88  SELECT  **姓名**,'语文'  AS     科目,语文 AS   成绩   FROM   TableA  UNION   ALL  SELECT  **姓名**,'数学' AS    科目,数学 AS   成绩   FROM   TableA  UNION   ALL  SELECT   姓名,'英语'AS    科目,英语 AS   成绩   FROM   TableA  ORDER BY 姓名,科目 DESC; |

## 左右链接

### 左连接

Select t1.\*,t2.\* from tabel1 t1 left join table2 t2 on t1.id=t2.t1id

或者

Select t1.\*,t2.\* from tabel1 t1, table2 t2 where t1.id=t2.t1id(+)

特点：右侧不匹配null显示

### 右链接

Select t1.\*,t2.\* from tabel1 t1 right join table2 t2 on t1.id=t2.t1id

或者

Select t1.\*,t2.\* from tabel1 t1, table2 t2 where t1.id(+)=t2.t1id

特点：左侧不匹配null显示

### 分页查询

|  |
| --- |
| Oracle 分页查询  Select \* from(  Select t1.rownum rn,t1. \* from  (Select \* from table where …) t1  where t1.rownum<maxrows  )t2  where t2.rn>minrows |
| Mysql 分页查询  SELECT \* FROM table LIMIT [offset,] rows | rows OFFSET offset  LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是 1)  mysql> SELECT \* FROM table LIMIT 5,10; // 检索记录行 6-15  //为了检索从某一个偏移量到记录集的结束所有的记录行，可以指定第二个参数为 -1：  mysql> SELECT \* FROM table LIMIT 95,-1; //检索记录行 96-last.  //如果只给定一个参数，它表示返回最大的记录行数目：  mysql> SELECT \* FROM table LIMIT 5; //检索前 5 个记录行 |

# sql 优化

1. 尽量sql语句用大写的；

因为oracle总是先解析sql语句，把小写的字母转换成大写的再执行

1. 尽量用TRUNCATE替代DELETE：

当删除表中的记录时,在通常情况下, 回滚段(rollback segments ) 用来存放可以被恢复的信息. 如果你没有COMMIT事务,ORACLE会将数据恢复到删除之前的状态(准确地说是恢复到执行删除命令之前的状况) 而当运用TRUNCATE时, 回滚段不再存放任何可被恢复的信息.当命令运行后,数据不能被恢复.因此很少的资源被调用,执行时间也会很短. (译者按: TRUNCATE只在删除全表适用,TRUNCATE是DDL不是DML)

1. 选择最有效率的表名顺序(只在基于规则的优化器中有效)：   
   ORACLE 的解析器按照从右到左的顺序处理FROM子句中的表名，FROM子句中写在最后的表(基础表 driving table)将被最先处理，在FROM子句中包含多个表的情况下,你必须选择记录条数最少的表作为基础表。如果有3个以上的表连接查询, 那就需要选择交叉表(intersection table)作为基础表, 交叉表是指那个被其他表所引用的表.
2. SELECT子句中避免使用 ‘ \* ‘：   
   ORACLE在解析的过程中, 会将'\*' 依次转换成所有的列名, 这个工作是通过查询数据字典完成的, 这意味着将耗费更多的时间
3. 尽量使用表的别名(Alias)：

当在SQL语句中连接多个表时, 请使用表的别名并把别名前缀于每个Column上.这样一来,就可以减少解析的时间

1. WHERE子句中的连接顺序：   
   ORACLE采用自下而上的顺序解析WHERE子句,根据这个原理,表之间的连接必须写在其他WHERE条件之前, 那些可以过滤掉最大数量记录的条件必须写在WHERE子句的末尾.
2. 尽量用EXISTS替代IN、用NOT EXISTS替代NOT IN：

在许多基于基础表的查询中,为了满足一个条件,往往需要对另一个表进行联接.在这种情况下, 使用EXISTS(或NOT EXISTS)通常将提高查询的效率. 在子查询中,NOT IN子句将执行一个内部的排序和合并. 无论在哪种情况下,NOT IN都是最低效的 (因为它对子查询中的表执行了一个全表遍历).

1. 尽量用UNION-ALL 替换UNION
2. 避免在索引列上使用NOT，
3. 避免在索引列上使用计算．
4. 避免在索引列上使用IS NULL和IS NOT NULL
5. 避免在索引列上使用‘!='，记住, 索引只能告诉你什么存在于表中, 而不能告诉你什么不存在于表中.
6. 避免在索引列上使用‘||'是字符连接函数. 就象其他函数那样, 停用了索引.
7. 避免在索引列上使用like操作
8. 用UNION替换OR (适用于索引列)