SQL命令行使用

游标详细的使用的方法:http://blog.csdn.net/liyong199012/article/details/8948952

连接数据库 conn/as sysdba 以数据库管理员身份连接数据库

conn 然后输入用户名 密码

sys 系统管理员 最高权限

system 本地管理员 次高权限

创建新用户方法: conn/as sysdba

create user username(你想要的用户名) identified by password

(你想设置的密码)

给用户授权 ：

grant create session to 用户名;

create table

create tablespace

create create view

create table

drop table

insert、update。

all

创建表空间 example:

create tablespace StudentInfo

datafile ‘S:/oracle\_table\_space/StudentInfo.dbf’

size 10M

autoextend on next 5M maxsize 1000M;

删除表空间 drop tablespace StudentInfo including contents and datafiles

通过名字查看存储过程:

SELECT \* FROM ALL\_SOURCE where TYPE='PROCEDURE' AND NAME ='MYPROCEDURE1';

查询数据库中所有表空间 select tablespace\_name from dba\_tablespaces;

显示某个表的结构: DESC 表名 DESC即describe(描述);

修改表结构 alter table 表名 add/modify (列名 类型 限定);

插入表记录 Insert into 表名 values ( );

删除表记录 delete from 表名 where 条件;

删除表中的所有记录 truncate table 表明;

创建表时指定所属的表空间方法： 在create语句最后的括号外 添加 tablespace 表空间名。

这种方式。

show user; 查看当前登录的用户

spool d:\test.txt; 将屏幕显示及查询结果输入到指定文本文件中

操作~~~

spool off; 此次输出会覆盖之前test中的文本内容.

sqlplus 登录之后,可以使用buff(缓存)来存储/执行/修改要执行的sql语句

这里的buff的特点:

1.buff中只能存储一条sql语句(但是这条sql语句可能有很多行)

2.每次放入新的sql语句,会把之前的覆盖掉

3.每次执行sql语句,都会把这个sql语句放到buff里面

l 查看缓存中的sql语句

然后输入想要定位的行号(数字) 定位到该行 然后就可以执行以下命令:

a 在[定位]的那一行后面追加新的内容

i 在[定位]的那一行下面插入新的一行

c 替换[定位]的那一行中的某些字符串

c/老的字符串/新的字符串

del 删除[定位]的那一行内容

n 后面加内容可以重写这一行

! 后面接终端命令 !clear:清屏 windows中使用$符号 例如:$cls

/ 执行缓存sql命令

clear buffer:清空当前缓存的命令

save test.sql buff中的sql语句保存在test.sql文件中

get test.sql 把test.sql中的内容在加载到buff中,但是没有运行

start test.sql 把test.sql中的内容在加载到buff中并且执行

@test.sql 把test.sql中的内容在加载到buff中并且执行

edit file\_name 使用系统默认编辑器去编辑文件

注意：单引号和双引号内大小写敏感。  **非常重要 ！！！！**

关键字不能缩写，也不能跨行书写，单引号内也不跨行写。

每个子句是一行。

字符串用单引，列别名用双引。

标点符号要用英文标点，中文出错。

column

column是sqlplus里最实用的一个命令，很多时候sql语句输出的列宽度不合适而影响查看，都需要用到这个命令来更改select语句中指定列的宽度和标题。大部分时候，我们可以简写column为col即可，主要有以下两种用法：

修改列宽度

column c1 format a20 --将列c1（字符型）显示最大宽度调整为20个字符

column c1 format 9999999 --将列c1（num型）显示最大宽度调整为7个字符

修改列标题

column c1 heading c2 --将c1的列名输出为c2

column 显示所有对列格式的设置情况

column last\_name 显示对last\_name列显示设置的情况

column last\_name clear 删除对last\_name列格式设置的情况

Clear column 清除所有column的格式设置

Column columName

可以显示该列的格式设置，这里的列并不特定于某个表.

类型转换函数

TO\_CHAR converts a number or date string to a character string.

 TO\_NUMBER converts a character string containing digits to a number.

 TO\_DATE converts a character string of a date to a date value.

select round(10/3) from dual; 四舍五入保留整数 3

select round(10/3,1) from dual 四舍五入保留一位 3.3

select round(10/3,2) from dual 四舍五入保留两位小数 3.33

to\_char 把日期转换为字符

例如:

把当前日期按照指定格式转换为字符串

select to\_char(sysdate,'yyyy')

from dual;

日期格式：

yyyy：四位数的年份

rrrr：四位数的年份

yy：两位数的年份

rr：两位数的年份

mm：两位数的月份(数字)

D：一周的星期几

DD：一月的第几天

DDD ：一年的第几天

YEAR：英文的年份

MONTH：英文全称的月份

mon：英文简写的月份

ddsp：英文的第几天(一个月的)

ddspth：英文序列数的第几天(一个月的)

DAY：全英文的星期

DY：简写的英文星期

hh：小时

mi：分钟

ss：秒

例如:

测试常见的一些日期数据转换为字符串的格式

select to\_char(sysdate,'yyyy MM D DD DDD YEAR MONTH ddsp ddspth DAY DY') from dual;

select to\_char(sysdate,'dd-mm-yy')

from dual;

select to\_char(sysdate,'yy-mm-dd')

from dual;

select to\_char(sysdate,'dd-mm-yy HH24:MI:SS AM')

from dual;

类型转换函数

TO\_CHAR converts a number or date string to a character string.

 TO\_NUMBER converts a character string containing digits to a number.

 TO\_DATE converts a character string of a date to a date value.

to\_char 把日期转换为字符

例如:

把当前日期按照指定格式转换为字符串

select to\_char(sysdate,'yyyy')

from dual;

日期格式：

yyyy：四位数的年份

rrrr：四位数的年份

yy：两位数的年份

rr：两位数的年份

mm：两位数的月份(数字)

D：一周的星期几

DD：一月的第几天

DDD ：一年的第几天

YEAR：英文的年份

MONTH：英文全称的月份

mon：英文简写的月份

ddsp：英文的第几天(一个月的)

ddspth：英文序列数的第几天(一个月的)

DAY：全英文的星期

DY：简写的英文星期

hh：小时

mi：分钟

ss：秒

例如:

测试常见的一些日期数据转换为字符串的格式

select to\_char(sysdate,'yyyy MM D DD DDD YEAR MONTH ddsp ddspth DAY DY') from dual;

select to\_char(sysdate,'dd-mm-yy')

from dual;

select to\_char(sysdate,'yy-mm-dd')

from dual;

select to\_char(sysdate,'dd-mm-yy HH24:MI:SS AM')

from dual;

select to\_char(sysdate,’yyyy mm dd’) from dual;

不等值连接

假设数据库中还有一张工资等级表:salgrade

工资等级表salgrade:

gradeName列表示等级名称

losal 列表示这个级别的最低工资数

hisal 列表示这个级别的最高工资数

表中的数据类似于下面内容:

id salgrade losal hisal

1 初级程序员 2000 4000

2 中级程序员 4000 6000

例如:

查询出员工的名字、职位、工资、工资等级名称9

SELECT e.last\_name, e.title, e.salray, s.gradeName

FROM s\_emp e, salgrade s

WHERE e.salray BETWEEN s.losal AND s.hisal

左外连接 以左为主 全部显示，即便右边的表无匹配

select \*

from a **left outer join** b

where a.~=b.~

右外连接 相反 right outer join

全连接 full outer join 顾名思义

自连接

执行多条连续插入的语句.

insert all into class values('CS0701','计算机科学与技术07-01班','信息管理学院',2007,null)

into class values('CS0702','计算机科学与技术07-02班','信息管理学院',2007,null)

into class values('IS0701','信息管理与信息系统07-01班','信息管理学院',2007,null)

into class values('IS0801','信息管理与信息系统08-01班','信息管理学院',2008,null)

into class values('CP0801','注册会计08\_01班','会计学院',2008,null)

into class values('CP0802','注册会计08\_02班','会计学院',2008,null)

into class values('CP0803','注册会计08\_03班','会计学院',2008,null)

into class values('ER0701','金融管理07-01班','金融学院',2007,null)

into class values('CS0801','计算机科学与技术08-01班','信息管理学院',2008,null)

select \* from dual;

**修改表的某列的结构的时候:**

alter table tablename modify (列名 新基础属性);

例如 alter table Class modify( classname varchar(100));

//在表中添加一个新的列

alter table t\_user

add birthday date;

//删除表的某列

alter table t\_user

drop column birthday;

//给表中的列添加约束

//这个约束相当于之前的表级约束

alter table t\_user

add constraint user\_name\_un

unique(name);

//测试刚添加的唯一约束是否生效

insert into t\_user(id,name) values(1,'zs');

insert into t\_user(id,name) values(2,'zs');

//删除表中的约束

alter table t\_user

drop constraint user\_name\_un;

//修改表的名字:

rename t\_user to mytest;

rename mytest to t\_user;

//修改表中某列的类型

alter table t\_user

modify (name varchar2(500));

//让约束失效:必须知道约束的名字

alter table t\_user

disable constraint user\_id\_pk cascade;

//测试是否设置成功

insert into t\_user(id,name) values(1,'zs1');

insert into t\_user(id,name) values(1,'zs2');

//让失效的约束再次生效

alter table t\_user

enable constraint user\_id\_pk;

//截断表中的数据(删除),不需要提交,默认已经提交,并且不能回滚

truncate table t\_user;

相当于:

delete from t\_user;

commit;

//给表添加注释

comment on table t\_user is '很好';

//给列添加注释

comment on column t\_user.name is 'good';

//查看表中注释

select \* from user\_tab\_comments where table\_name=upper('t\_user');

//查看列中的注释

select \* from user\_col\_comments

where

comments is not null

and

table\_name=upper('t\_user');

运行时参数

sql语句中的值,我们可以使用一个参数来代替,然后每次运行的时候都可以重新输入这个值

例如:

select last\_name,salary,dept\_id

from s\_emp

where id=&id;

select last\_name,salary,dept\_id

from s\_emp

where last\_name=&name;

select last\_name,salary,dept\_id

from s\_emp

where last\_name='&name';

select last\_name,salary,dept\_id

from s\_emp

where

salary>&a

and

salary<&b;

select last\_name,salary,dept\_id

from s\_emp

where &con;

select last\_name,salary,dept\_id

from s\_emp

&con;

注意:&变量名 表示使用这个oracle定义的变量,如果这个变量之前没有定义过,那么这个时候会让你重写输入这个变量的值.如果之前有定义过,那么就是要之前定义过的值

define undefined accept prompt命令

例如:

命令def可以定义一个变量

定义变量: def A=s\_emp

查看定义的变量: def

取消定义的变量: undef A

然后变量A之前定义过,那么这里会之前替换为之前定义的值

select id,last\_name,salary

from &A;

accept命令也可以定义一个变量,而且变量的值需要用户再次输入

例如:

accept A

回车之前需要用户再次输入变量A的值

之后可以使用def命令来查看刚刚定义的A变量

prompt可以在用户输入的时候显示一个提示信息:

例如:

accept name prompt '请输入name变量的值: '

hide可以隐藏用户的输入内容不被看见

例如:

accept name prompt '请输入name变量的值: ' hide

注意:这些定义的都是临时变量,sqlplus退出后重新登录进来就没有了

序列的简单使用方法:

1.创建表

create table test(

id number(3) primary key,

name varchar2(30) not null

)

#查看约束

select \* from user\_constraints;

2.创建序列

create sequence test\_seq

start with 1

increment by 1;

#查看序列

select \*

from user\_sequences;

3.创建触发器

create or replace trigger test\_trigger

before insert on test

for each row

when(new.id is null)

begin

select t\_user\_id\_seq.nextval into:NEW.ID from dual;

end;

#查看触发器

select \* from user\_triggers;

#测试

insert into test values(null,’lyf’);

commit;

select \* from test;

select a.\* ,b.\*

from table1 a,table2 b

此条选择语句会造成笛卡尔积的产生，即如果a表有50条，B表有50条。这种没有where连接的语句得出的表会是左右相乘的2500条。 三张表的话以此类推， 所以才需要where语句来连接。避免笛卡尔积的产生。

常见的数据字典(它们都是视图)

USER开头的视图里面存放着用户自己拥有的对象

ALL开头的视图存放着用户有权限查看的对象

DBA开头的视图存放着数据库所有的对象

V$开头的视图存放数据库运行的一些性能属性数据

//查询用户对象表,找出对象类型的类型都有哪些

select distinct object\_type

from user\_objects;

**视图**

视图的作用:

1.隐藏表中的重要数据

2.代替一些比较长的sql语句

视图分为俩类:

简单视图:

视图所代表的sql中如果**没有group by语句,没有组函数**,查询的只有一张表,那么这样的视图就是简单视图.

复杂视图

视图所代表的sql中**如果有group by语句,或者有组函数,或者查询的是多张表,**那么这样的视图就是复杂视图.

简单视图和复杂视图的区别:

通过简单视图可以修改原来表中的数据,通过复杂视图是不能修改原来的数据的。

with read only语句

特点:只能通过视图进行查询数据,不能修改

with check option语句

特点:通过视图进行的修改 那么也必须可以通过这个视图能够显示出来,要不然就操作失败

以上字句如果都不添加的话，那么用户就可以通过视图任意修改表了。

**数据库事务**

数据库事务

1.**DML语句执行的时候,如果当前有事务,那么就使用这个事务,如果当前没有事务,这个执行的DML语句就会产生一个新的事务。**

2.只有**DML语句才会产生事务**,其他语句不会产生事务。

3.**commit/rollback/DDL语句都可以把当前事务给结束掉**

4.**commit和DDL语句结束事务的方式是把这个事务给提交了**

5.rollback结束事务的方式是把这个事务给回滚了

注:

提交事务是指让这个事务里面的所有操作都生效到数据库中

回滚事务是指让这个事务里面的所有操作都撤销

测试用的表:

create table t\_customer(

id number,

name varchar2(20) constraint customer\_name\_nn not null,

constraint customer\_id\_pk primary key(id)

);

drop table t\_customer;

测试: 使用俩个终端窗口,同一个账号登录到数据库中,观察事务是否提交对用户查看数据的影响

注:一个用户对A表做了DML操作,但是没有提交事务,这时候别的用户是不能对A表再做其他的DML操作。(为了保证数据的安全和一致性)

例如1:

insert ....产生事务A

update ... 这个操作是事务A中的操作

insert .. 这个操作是事务A中的操作

commit; 让事务A里面的三个操作生效、事务A结束

delete ... 产生新的事务B

insert .. 这个操作是事务B中的操作

insert .. 这个操作是事务B中的操作

insert .. 这个操作是事务B中的操作

rollback; 让事务B中的四个操作都撤销,事务B结束

例如2:

insert ....产生事务A

update ... 这个操作是事务A中的操作

insert .. 这个操作是事务A中的操作

DDL语句; 事务A会被提交,事务A结束

rollback; 这时候回滚已经对事务A不起作用,因为事务A以及被提交了,当前已经没有事务了

注**:create语句 drop语句 alter语句等都属于DDL语句**

事务特征ACID：

原子性：Atomicity

同时成功或者同时失败

一致性：Consistency

事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。

隔离性：Isolation

事务操作应该相互独立

持久性：Durability

事务所做的影响 ，在事务结束之后应该能够是持久的。

isolation 事务隔离级别

事务中产生的问题:

1.**脏读** 主要针对update操作。 一个事务A读到另一个事务B中修改过但是还没有提交的数据

2.**不可重复读** 主要针对update操作。 一个事务A在第一次读数据和第二次读数据之间,有另一个事务B把这个数据更改并提交了,所以就出现了事务A里面读一个数据俩次,但是读到的结果是不同的。

3.**幻读** 主要针对的是insert/delete操作。事务A第一次用where条件筛选出了10条数据,事务A第二次用通样的where条件筛选出的却是11条数据,因为事务B在事务A的第一次和第二次查询之间进行了插入操作,并且插入的这个数据满足事务A的where筛选条件.

事务隔离级别有:

**read-uncommitted** 不提交也能读

**read-committed** 提交之后才能读 解决了脏读

**repeatable-read** 解决了脏读和不可重复读

**serializable** 三个问题都解决了

级别越高解决的问题越多但是效率越低。

注意:并不是所有数据库都支持这四种事务隔离级别,比如oracle就只支持第二种和第四种这俩种,比如mysql就四种全支持。

oracle里面默认的事务隔离级别是第二种:read-committed

oralce里面设置事务隔离级别:

Set Transaction Isolation Level Read Uncommitted

Set Transaction Isolation Level Read Committed

Set Transaction Isolation Level Read Repeatable

Set Transaction Isolation Level Serializable

回滚点/保存点 savepoint

例如:

DML语句1

savepoint A

DML语句2

savepoint B

DML语句3

rollback to A/B

这个时候可以通过这个回滚点让事务回滚到指定的位置,如果不指定回滚点而是直接rollback,那么事务会一下子回滚完.

【特别注意】:rollback到回滚点之后,这个事务并没结束,这个时候还可以接着回滚或者commit提交事务。

create table t\_user(

id number primary key,

name varchar2(100),

salary number

);

drop table t\_user;

例如:

insert into t\_user values(1,'tom',1000);

savepoint A;

insert into t\_user(id,name) values(2,'zs');

savepoint B;

delete from t\_user;

rollback to B;

然后查询看结果

select \* from t\_user;