# ORACLE基本语法

* 引号

**Oracle**在创建表的时候**表名或者列名没有用双引号括起**是不区分大小写的，用双引号括起才区分大小写。并且，除了取别名的情况下，都用单引号，当然取别名也可以不用引号

* 外键

**Foreign key(s\_sid) references students(s\_id) [on delete cascade|| set null]**定义了一个外键约束，被student1表的s\_id约束，on delete 子句说明了当父表的行被删除的时候，子表被约束的行的处理办法。cascade是删除子表所有被父表约束的行，set null 是将被约束的行置空。

* 运算顺序

运算符运算顺序，先运算not，and最后or，但是可以用括号来改变运算顺序。

* 运算关系

And运算结果关系

有null参与运算的话，有false结果是false，无false结果是null；

| **值** | **TRUE** | | **FALSE** | **NULL** |
| --- | --- | --- | --- | --- |
| TRUE | | TRUE | FALSE | NULL |
| FALSE | | FALSE | FALSE | FALSE |
| NULL | | NULL | FALSE | NULL |

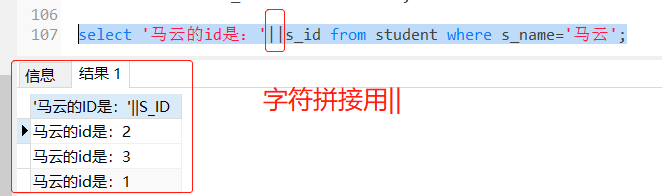
* Or运算结果关系

有null参与运算的话，有true结果是true，无true结果是null；

| **值** | **TRUE** | | **FALSE** | **NULL** |
| --- | --- | --- | --- | --- |
| TRUE | | TRUE | TRUE | TRUE |
| FALSE | | TRUE | FALSE | NULL |
| NULL | | TRUE | NULL | NULL |

* 字符拼接

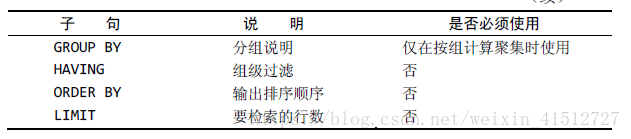
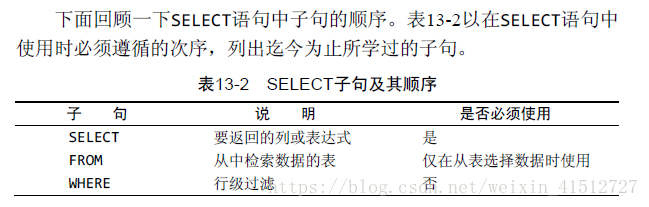
Oracle当中字符拼接用 ||完成；



* Where和having的区别

where中不能有分组函数，例如avg(),sum(),max(),min()等。Having后可以有分组函数。

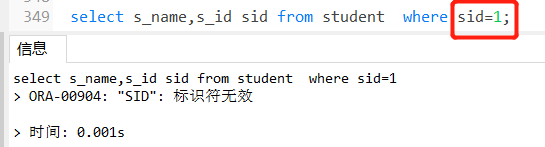
* 子句顺序

Where 之后如果有group by子句，where要放在group by之前，having要放在group by之后。

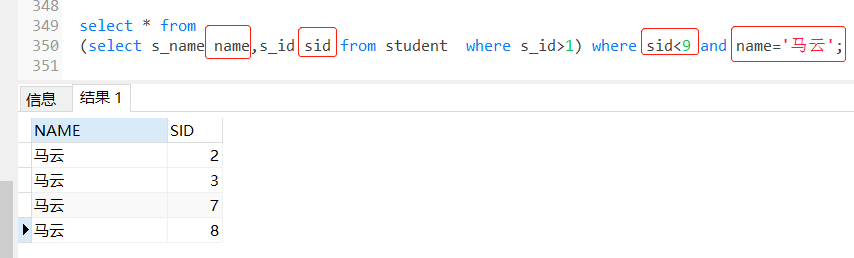
* 别名

在oracle中，条件判断不能用别名，例如:

select s\_name,s\_id sid from student where sid=1;

因为在查询中，where子句会比select先执行，而别名要在select中才会定义，所以在where条件判断的时候，别名会无法识别，所以条件判断中不能使用别名。

这条规则仅限于非子句的别名，例如外层查询要使用内层查询的别名是可以的，因为内层查询的select是比外层的where先执行的。



* End相关

有begin就有end，有if就有end if，有loop就有end loop，有open cursor 就有

Close cursor，

* 参数

游标，函数等的参数，在写在游标或者函数后作为形参时，不能带长度

# 数据定义语言DDL （Data Definition Language）

## 1.创建表语言create table

create table student(

s\_id number(2) primary key,

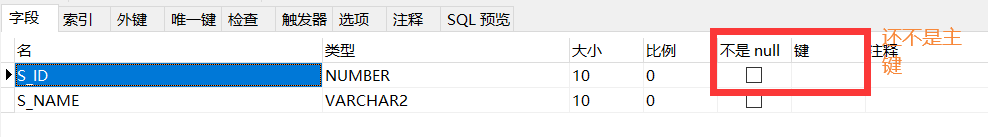
s\_name varchar2(4) ,

s\_sid number(2),

[primary key (s\_id)] 主键约束

[foreign key (s\_sid) references student1(s\_id)] 外键约束

);



create table students as (select s\_id,s\_name from student); 将另一个表查询出来的数据建一个表，不包括索引和主键。

包括：create table \ modify table \ drop table \ create index \ drop index

## 2.更改表结构 alter table

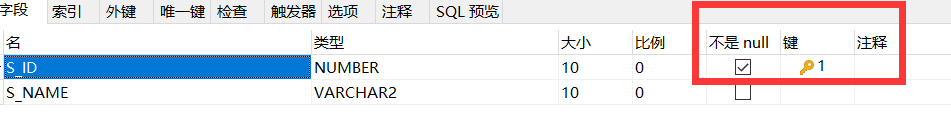
### Add

alter table student add s\_age date;增加属性列

alter table student add primary key(s\_id); 增加主键

alter table students add foreign key(s\_sid) references student(s\_id) on delete cascade; 增加外键

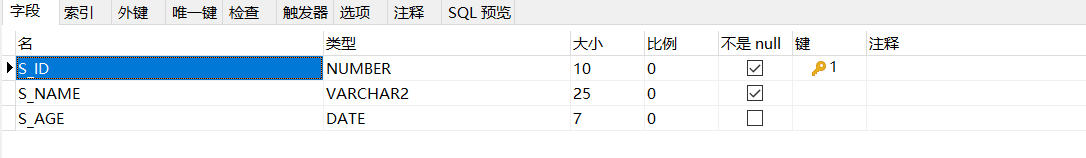
alter table student add stuts number(1) default 1; 增加属性列并给定默认值





### Modify

alter table student modify s\_name varchar2(25) not null;修改属性列，包括改类型，非空，约束等。改变属性的时候，要保证所有的数据满足改之后的属性，例如：改变了非空属性，则该字段不能有空数据，否则会改失败。



### Drop

alter table studet drop column s\_age; 删除属性列

或者可以用：alter table student set unused column s\_age; 逻辑删除s\_age 列

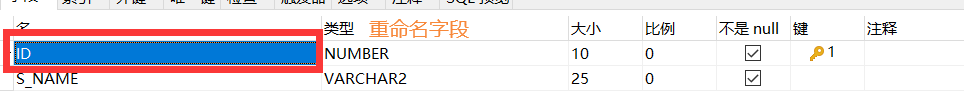
之后再用: alter table student drop unused column; 物理删除s\_age列



### Rename

alter table student rename column s\_id to id; 重命名属性列

alter table students rename to student; 重命名表



## 3.删除表 drop table

drop table student [cascade constraints ]; cascade 子句可不选，若不选，这张表有被别的表外键引用的时候不能删除，若选了，则会删除这个表和引用的那张表的外键约束。



## 4.

# 数据操作语言 （Data Manipulation Language）

包括 insert \ update \ delete

## 1.插入数据 insert into

insert into students(s\_id,s\_name) values(2,’丽丽’); 插入一行数据

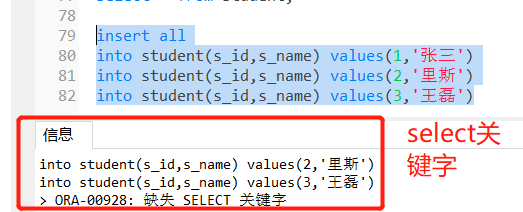
insert all

into student(s\_id,s\_name) values(1,'张三')

into student(s\_id,s\_name) values(2,'里斯')

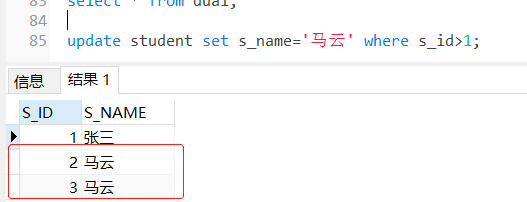
into student(s\_id,s\_name) values(3,'王磊')

select \* from dual; 插入多行数据，最后的select子句不能少，否则会报错。



## 2.更新数据 update

update student set s\_name='马云' where s\_id>1; 更新数据



## 3.删除数据 delete

delete from student where s\_id=1; 删除数据，如果这时候student表被其他表引用了，就会删除失败。如果表定义外键的时候有on delete子句，就会根据子句的 cascade属性或者 set null来处理。

# 数据控制语言 （Data Control Language）

# *数据类型*

## Number（p,s）

P表示精度，是数字中的有效数位，范围为1~38.默认为38.超过会报错。

S表示尺度，取值范围为-84~127，如果取值为正数，表示小数点往右的最大数位，超过这个数位就会被四舍五入。如果为负数，表示这个数要精确到小数点左边的指定数位。默认值为0。

小数点左边有多少位数取决于 p-s,不论s为正还是为负。

Number 1254 1254

Number(5) 1254 1254

Number(3) 1254 报错 有效数位超过3

Number(4,1) 1254.1 报错 有效数位超过4

Number(4,1) 1254.0 报错 小数点左边的数位大于 4-1

Number(4,1) 125.1 125.1

Number(4,1) 125.19 125.2 超过就四舍五入

Number(4,1) 999.99 报错 小数四舍五入之后超过了数的精度

Number(2,3) 0.100 报错 2-3=-1，小数点右边1位不能有数

Number(2,3) 0.0157 0.016 小数点右边超过3位四舍五入

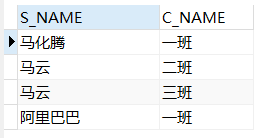
Number(3,-2) 45600 45600 3- -2=5，小数点左边有5位

Number(3,-2) 45690 45700 小数点左边前两位有数就会四舍五入

# 表连接

## 1.内连接 inner join

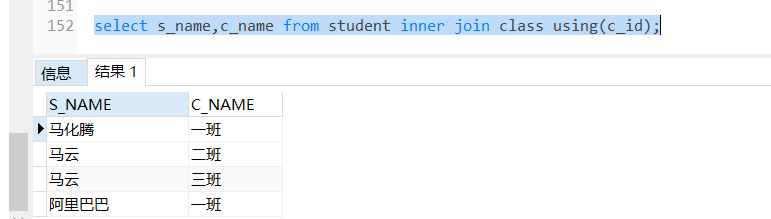
select s\_name,c\_name from student inner join class on student.s\_cid = class.c\_id ;



返回的是两个表中都满足on子句里的条件的行。

on 子句可以换成using子句：

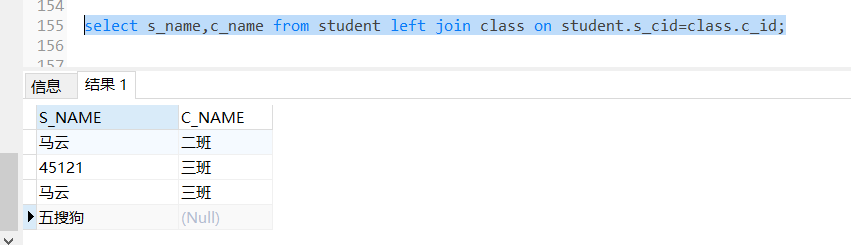
select s\_name,c\_name from student inner join class using(c\_id);



Using子句中出现的列必须要在两张表中都出现，必须要用括号括起。

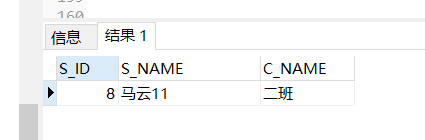
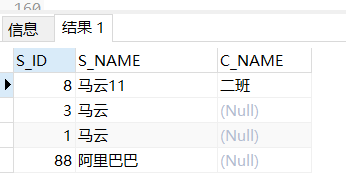
## 2.左连接 left join

select s\_name,c\_name from student left join class on student.s\_cid = class.c\_id;



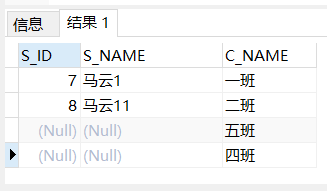
左连接返回的数据是：左表的所有数据和右表匹配的数据，右表中不匹配的行会置空然后显示。左连接也可以将on子句换成using子句，同样也是要字段在两张表中都出现

左连接中注意where条件，将限制条件写在where中，只会显示满足条件的行，将限制条件写在on中，会返回所有行，不满足条件的行会置null显示。



## 3.右连接 right join

select s\_id,s\_name,c\_name from student right join class on student.s\_cid = class.c\_id;



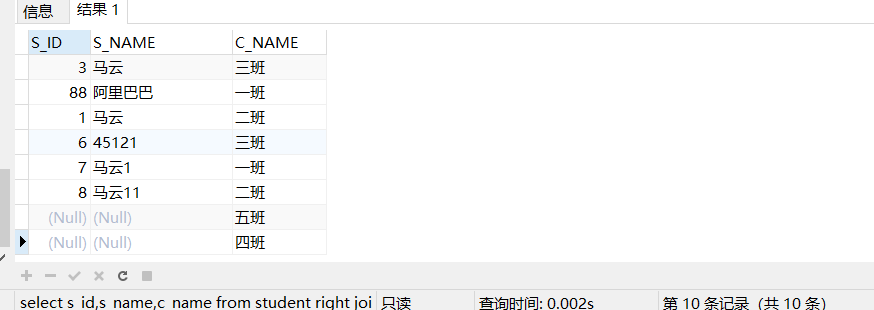
右连接返回的是右表的所有行，左表不匹配的行会置空然后展示。

同理右连接也可以有using子句和需要注意where子句里的条件。

## 4.全连接 full join

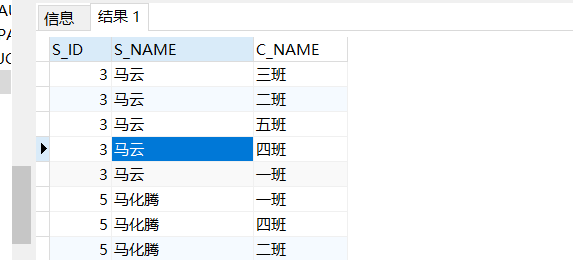
select student.s\_name,student.s\_id,class.c\_name from student full join class on student.s\_cid=class.c\_id;

全连接其实就是左连接和右连接的合集



## 5.交叉连接 cross join（笛卡尔连接）

**select s\_id,s\_name,c\_name from student cross join class order by s\_id;**



# 操作符

## 1.all操作符

操作符之前必须有运算符：=，!=,>，>=，<，<=

select s\_id,s\_name,age from student where age>all(0,2,5);查找大于5的值

select c from c where c1>all(c1,c2,c3);

all表示满足所有查询结果，等价于：

Select c from c where c1>c1 and c1>c2 and c1>c3;

多个条件之间用and连接

All 操作符和各个运算符的关系：col 和all(list) 的关系

= :col = all(list) col 与list里的每一个值完全匹配 为true

!= : col 与list里的所有值都不匹配时为true

> : col 大于list的最大值时为true

>=: col 大于等于list 的最大值时为true

< : col 小于 list的最小值时为true

<= : col 小于等于 list的最小值时为true

## 2.any和some操作符

操作符之前必须有运算符：=，!=,>，>=，<，<=

Any 和 some的行为相同，所以他们完全可以互相替换。

select s\_id,s\_name,age from student where age>any(0,2,5);查找大于0的值

select c from c where c1>any(c1,c2,c3);

any/some表示满足至少一个查询结果，等价于：

Select c from c where c1>c1 or c1>c2 or c1>c3;

多个条件之间用or连接

Any/some 操作符和各个运算符的关系：col 和all(list) 的关系

= :col = all(list) col 与list里的至少一个值匹配 为true

!= : col 与list里的至少一个值不匹配时为true

> : col 大于list的最小值时为true

>=: col 大于等于list 的最小值时为true

< : col 小于 list的最大值时为true

<= : col 小于等于 list的最大值时为true

## 3.exists和not exists 操作符

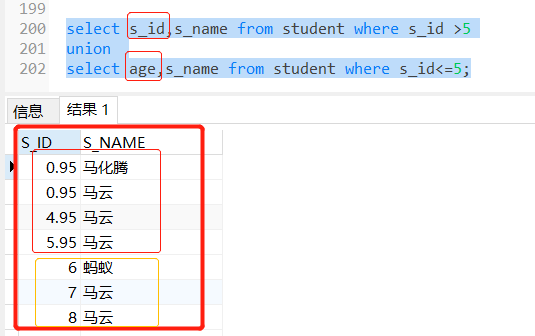
## 4.union 操作符

select s\_id,s\_name from student where s\_id >5

union

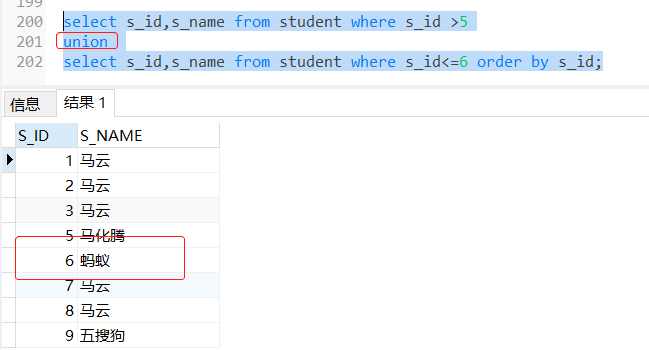
select s\_id,s\_name from student where s\_id<=5;

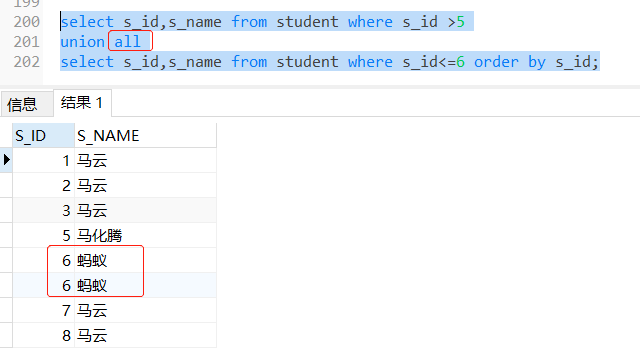
union将两个或多个select语句联合在一起，所有select语句必须返回相同的列数，对应列的类型也要相同。如果l



Union 和union all的区别：

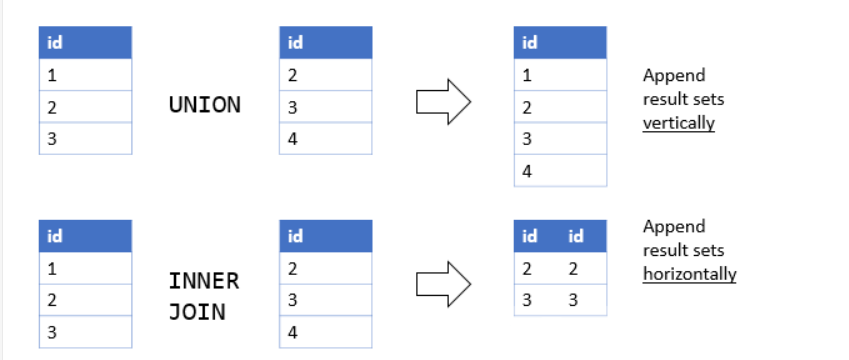
Union会将不同查询返回的相同行过滤，而union all会全部显示。





Union和join的区别，

Union将返回的结果纵向组合，而join将结果横向组合。



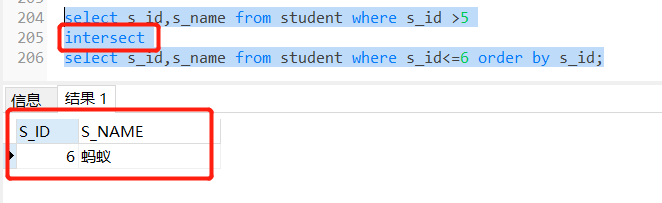
## 5.intersect 操作符

和union操作符类似，intersect操作也要求返回相同的列数，对应列也要相同类型，不同于union的是，intersect返回的是两个或以上查询相同的行。

select s\_id,s\_name from student where s\_id >5

intersect

select s\_id,s\_name from student where s\_id<=6;



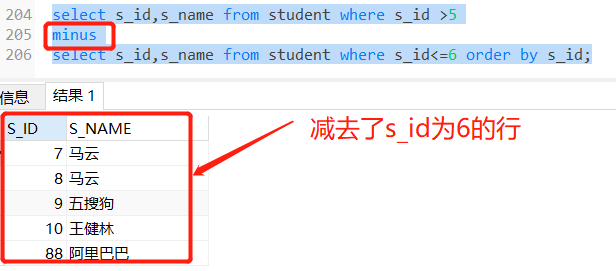
## 6.minus 操作符

和union、intersect操作符类似，minus也要求列数相同和类型相同，不同的是minus返回的是第一个查询减去第二个查询的记录，

select s\_id,s\_name from student where s\_id >5

minus

select s\_id,s\_name from student where s\_id<=6;



# 循环

有三种循环方式：while循环，exit循环，for循环每种循环里面都有loop。

## While循环：

declare

i number(2):=1;

begin

while i<11 loop 当i<11的时候进入循环

dbms\_output.put\_line(i); 循环体

i:=i+1; 循环进行方式，i+1

end loop; 有loop就有end loop

end;

## exit循环： 常用循环，重点

declare

i number(2):=1;

begin

loop 上来就循环

exit when i>10; 提出条件

dbms\_output.put\_line(i); 循环体

i:=i+1; 循环进行

end loop; end loop不能忘

end;

## for循环

declare

begin

for i in 1..10 loop i在1到10 的时候循环

dbms\_output.put\_line(i);

end loop;

end;

# If判断

declare

i number(3) := &ii; sql plus里可以用&符号让用户从键盘输入数据，

begin &后面带的字符可以随便取

if i<18 then

dbms\_output.put\_line('少年');

elsif i<30 then

dbms\_output.put\_line('青年');

elsif i<50 then

dbms\_output.put\_line('壮年');

else

dbms\_output.put\_line('老年');

end if;

end;

有if就有end if，if、 elsif、 else、end if是if判断相关的，注意elsif 中间少了一个e。在写判断条件的时候，第一个elsif不需要写i>18 and i<30，因为之有不满足第一个判断才会进入到第二个判断。

# 函数

分为单行函数和多行函数；

单行函数：作用于一行，返回一个值。

多行函数：作用于多行，返回一个值。

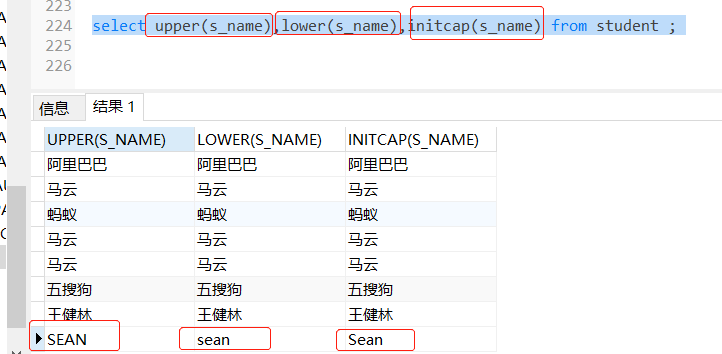
## 单行函数

### 1.字符函数：

#### 大小写转换

upper，lower全部大写或者小写，initcap首字母大写

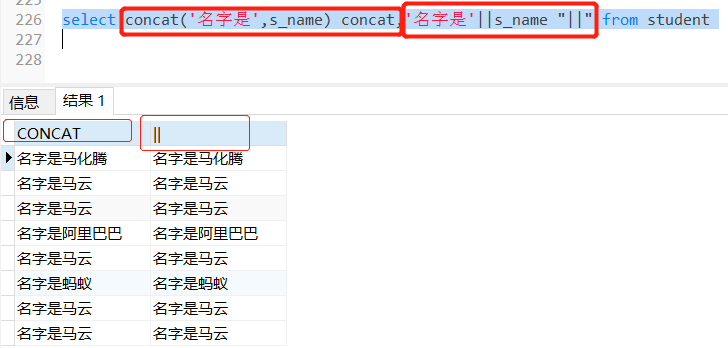
select upper(s\_name),lower(s\_name) ,initcap(s\_name) from student ;



#### 字符串拼接函数

concat, || 两种拼接方式是一样的；

select concat('名字是',s\_name) concat, '名字是'||s\_name "||" from student;



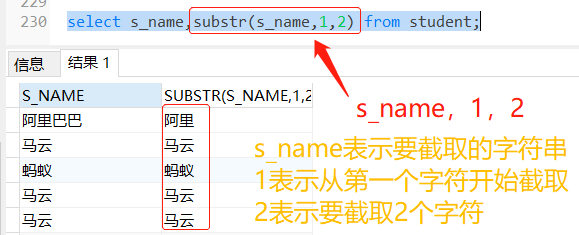
#### 字符串截取

substr 截取一定长度的字符串

select s\_name,substr(s\_name,1,2) from student;

substr(string,start,length)

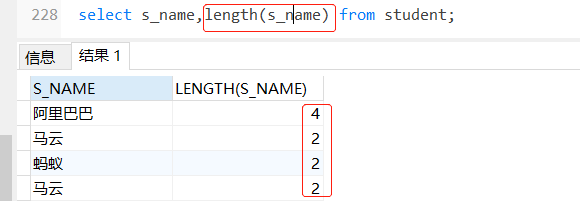
string 表示源字符串，start表示开始截取的位置，length表示截取的长度



#### 字符串长度

length 返回字符串长度

select s\_name,length(s\_name) from student;

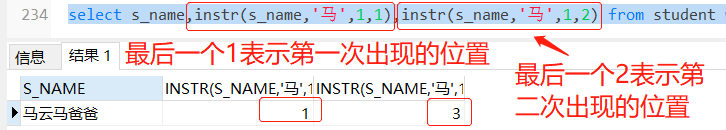


查找字符串位置：instr返回字符串出现位置

select s\_name,instr(s\_name,'马',1,1),instr(s\_name,'马',1,2) from student

instr(c1,c2,start,times)

c1：源字符串c2：要查找的字符串start：开始查找的位置times：第几次出现



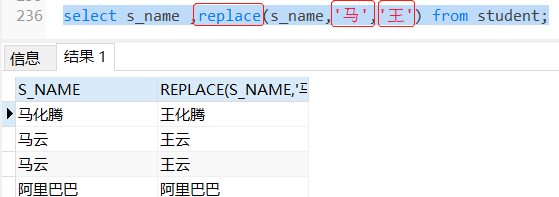
#### 字符串替换

replace 替换字符串为指定字符串

select s\_name ,replace(s\_name,'马','王') from student;

replace(string,s1,s2)

string：源字符串 s1：要替换掉的字符串s2：用来替换的字符串



### 2.日期函数

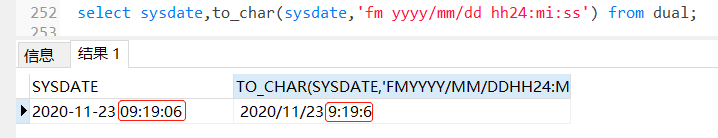
#### 日期转换函数

to\_char,to\_date

##### To\_char:

select to\_char(sysdate,'fm yyyy/mm/dd hh24:mi:ss') from dual;

将日期类型转为字符类型



Fm:将数值前的0去掉

hh后加24表示是24小时制

可以用这个方式格式化日期类型，改变间隔符号等，符号可以多种，包括空格。上图两个输出虽然看起来一样，但是一个是日期类型，一个是字符类型。

##### to\_date:

select to\_date('2020/10/07 11:56:54','fm yyyy-mm-dd hh:mi:ss') from dual;

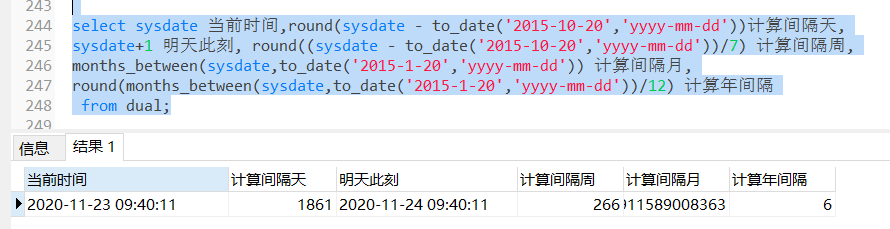
将字符串类型转为日期类型



日期格式的输出只有一种格式 就是yyyy-mm-dd hh24:mi:ss

如果hh后没有加24，那么前面的字符串不能写24制小时。

#### 日期的运算：



Sysdate系统时间，时间可以直接加减

sysdate+1 明天此刻

round(sysdate - to\_date('2015-10-20','yyyy-mm-dd'))计算间隔天,

round((sysdate - to\_date('2015-10-20','yyyy-mm-dd')) /7 ) 计算间隔周

months\_between(sysdate,to\_date('2015-1-20','yyyy-mm-dd')) 计算间隔月

只有月有months\_between函数，计算两个时间的月间隔

round(months\_between(sysdate,to\_date('2015-1-20','yyyy-mm-dd')) /12 ) 计算年间隔 一年一定有12个月

#### 日期截断函数:

[Trunc()](https://blog.csdn.net/uotail/article/details/83176241)和round都有截断时间的作用，不过round会四舍五入的取最近的时间，trunc只会直接截取。

#### 时间的各种运算：

-- 本季度第一天

SELECT to\_char(TRUNC(SYSDATE, 'Q'), 'YYYY-MM-DD') FROM dual;

-- 上个季度最后一天（可以用本季度第一天减去1得到）

SELECT to\_char(TRUNC(SYSDATE, 'Q') - 1, 'YYYY-MM-DD') FROM dual;

-- 上个季度第一天(本季度第一天 减 3个月)

SELECT to\_char(add\_months(TRUNC(SYSDATE, 'Q'), -3), 'YYYY-MM-DD')

FROM dual;

-- 上个季度最后一天(本季度减一个月之后的月份的最后一天)

select to\_char(last\_day(add\_months(trunc(sysdate, 'Q'), -1)), 'yyyy-mm-dd')from dual;

--№1: 取得当前日期是本月的第几周

select to\_char(sysdate,'YYYY-MM-DD W HH24:MI:SS') from dual;

select to\_char(sysdate, 'W') from dual;

--本周的第一天

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

--№2:取得当前日期是一个星期中的第几天,注意星期日是第一天

select sysdate, to\_char(sysdate, 'd') from dual;

--类似:

select to\_char(sysdate, 'yyyy') from dual; --当前年

select to\_char(sysdate, 'Q' ) from dual; --当前季度

select to\_char(sysdate, 'mm') from dual; --当前月

select to\_char(sysdate, 'dd') from dual; --当前日

select to\_char(sysdate, 'day') from dual; --中文星期几

--№7:得到明天凌晨0点0分0秒的日期

select trunc(sysdate + 1) from dual;

select trunc(sysdate) + 1 from dual;

--№8: 本月一日的日期

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

--№9:得到下月一日的日期

select add\_months(trunc(sysdate, 'mm'), 1) from dual;

select trunc(add\_months(sysdate, 1), 'mm') from dual;

-- №10:返回当前月的最后一天?

select last\_day(sysdate) from dual; --最后一天的当前时刻

select last\_day(trunc(sysdate)) from dual;--最后一天的零点

select trunc(last\_day(sysdate)) from dual;--零点

select trunc(add\_months(sysdate, 1), 'mm') - 1 from dual;--零点

--№11: 得到一年的第一天

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

--№11: 得到一年的每一天 输出每一天

select trunc(sysdate, 'yyyy') + rn-1 date0

from (select rownum rn from all\_objects where rownum < 366) t;

--№12:今天是今年的第N天

SELECT TO\_CHAR(SYSDATE, 'DDD') FROM DUAL;-- 年的第几天

SELECT TO\_CHAR(SYSDATE, 'D') FROM DUAL;--周的第几天

SELECT TO\_CHAR(SYSDATE, 'DD') FROM DUAL;--月的第几天

--№13:如何在给现有的日期加上2年

select add\_months(sysdate, 24) from dual;

##### --№14:判断某一日子所在年分是否为润年

select last\_day(trunc(sysdate, 'yyyy') + 31) from dual -- 通过一年的第一天加上31天到2月,然后取2月最后一天

select decode(to\_char(last\_day(trunc(sysdate, 'y') + 31), 'dd'),

'29',

'闰年',

'平年')

from dual;

select

case to\_char(last\_day(add\_months(trunc(sysdate,'y'),12)+31),'dd')

when '29' then '闰年'

else '平年'

end

from dual;

--№15:判断两年后是否为润年

select decode(to\_char(last\_day(trunc(add\_months(sysdate, 24), 'y') + 31), 'dd'),

'29',

'闰年',

'平年')

from dual;

--№16:得到日期的季度

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

select to\_number(to\_char(sysdate,'q')) from dual;

select ceil(to\_number(to\_char(sysdate,'mm'))/3) from dual;

select to\_char(sysdate, 'Q') from dual;

--No17:返回某个月的最后一天

select to\_char(last\_day(sysdate), 'dd') from dual

--本周的第一天和最后一天，考虑到中国人习惯是周一为每周的第一天（国际惯例是周日为第一天，oracle也是这样）

select trunc(sysdate,'d')+1,trunc(sysdate,'d')+7 from dual where to\_char(sysdate,'day')<>'星期日'

union all

select trunc(sysdate,'d')+1-7,trunc(sysdate,'d')+7-7 from dual where to\_char(sysdate,'day')='星期日'

--当前季度的第一天

select to\_char(trunc(sysdate,'Q'),'yyyy-mm-dd') from dual;

--当前季度的最后一天

SELECT to\_char(add\_months(TRUNC(SYSDATE, 'Q'), +3)-1, 'YYYY-MM-DD') FROM dual;

select to\_char(trunc(add\_months(last\_day(sysdate), -1) + 1), 'yyyy-mm-dd') "本月第一天",

trunc(sysdate,'mm'),to\_char(last\_day(sysdate), 'yyyy-mm-dd') "本月最后一天" FROM dual;

--取得当前日期是本月的第几周

select to\_char(sysdate,'yyyymmdd w hh24:mi:ss') from dual;

select to\_char(sysdate, 'ww') from dual;--当年第几周

select to\_char(sysdate,'w') from dual ;--本月第几周

select trunc(sysdate,'w') from dual; --本周第一天

~~select trunc(sysdate,'ww') from dual;--前一周时间~~

### 3.数值函数

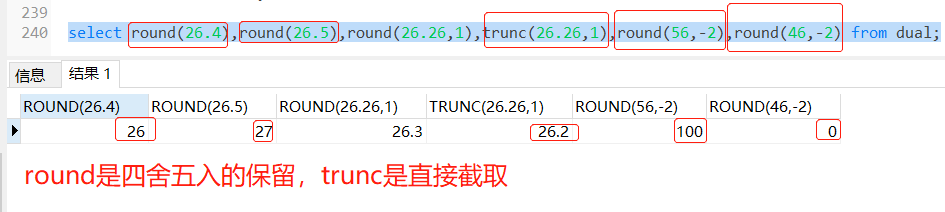
#### 保留小数

round，trunc不同的取精度方式

select round(26.4),round(26.5),round(26.26,1),trunc(26.26,1),round(56,-2),round(46,-2) from dual;

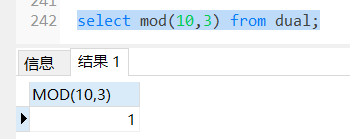
round（n1，n2）：n1：源数值，n2：保留的位数 四舍五入

trunc（n1，n2）：n1：源数值，n2：保留的位数 直接截取，不考虑四舍五入



#### 求余

mod



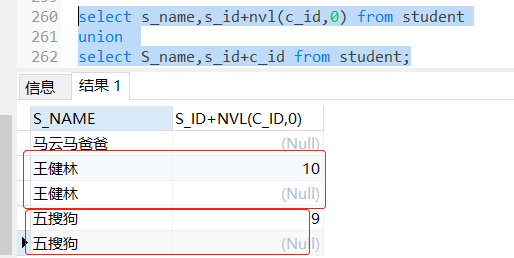
### 4.通用函数

空值处理函数：nvl

Null值与任何值计算都为null值

select s\_name,s\_id+nvl(c\_id,0) from student

若c\_id为空则用0参与运算



多行函数

又叫聚合函数，分组函数共有5个，sum，avg，min，max，count

select count(1) from student; 计数

select sum(s\_id) from student; 总和

select max(s\_id) from student; 最大值

select min(s\_id) from student; 最小值

select avg(s\_id) from student; 平均值

## 条件表达式

**常用的写法是oracle和mysql通用写法，推荐使用**

select s\_id,s\_name,case [s\_id]

when s\_id<5 then '小老弟'

when s\_id<8 then '中老弟'

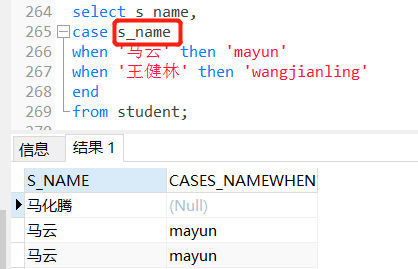
when s\_id<20 then '大老弟'

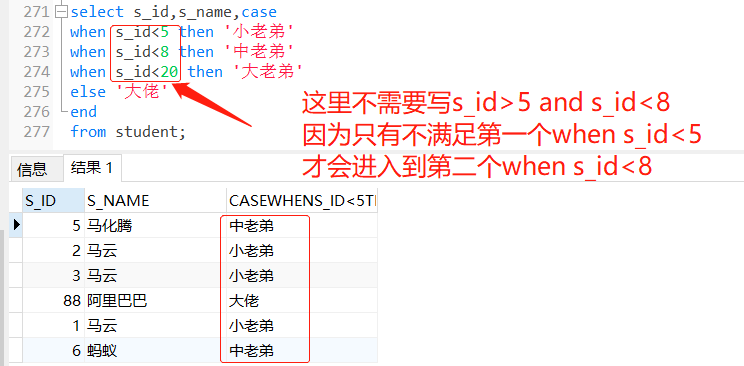
else '大佬'

end

from student;

有case就有end 如果是等值判断，将条件写在case后面，case后面写了，when后就不能写





**Oracle专用条件表达式写法 考虑到数据库重用，不推荐用**

select s\_name,

decode(

s\_name,

'马云' , 'mayun' ,

'王健林' ,'wangjianling'

)

from student;

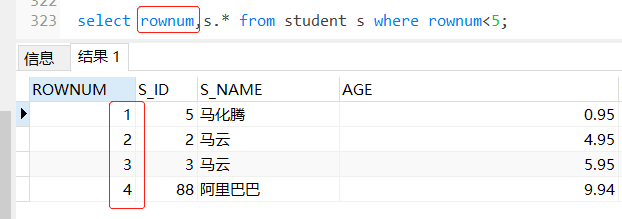
条件都用，逗号间隔

# 分页查询

## Rownum行号

在查询的时候，每查询一条记录就会给该行加上一个行号，行号从1开始，每次加一，不能跳着走。行号不属于任何一张表，行号前不能加表名；

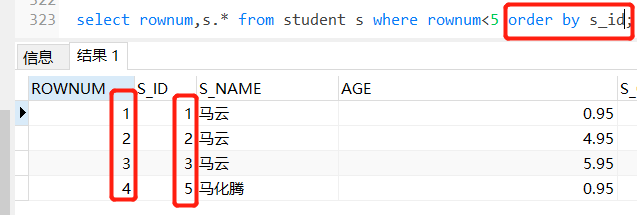
select rownum,s.\* from student s where rownum<5;



## rownum与order by的关系

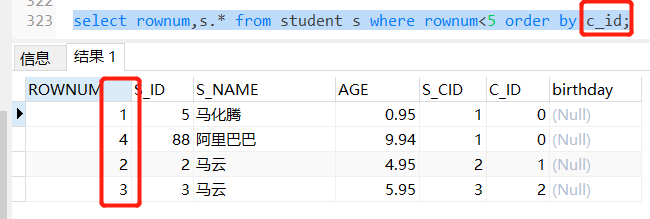
select rownum,s.\* from student s where rownum<5 order by s\_id;

如果order by 的列是主键列，那么会先排序，再加行号，所以行号不会乱。



select rownum,s.\* from student s where rownum<5 order by c\_id;

如果order by的列不是主键列，那么会先赋予行号，再排序，所以行号会乱。

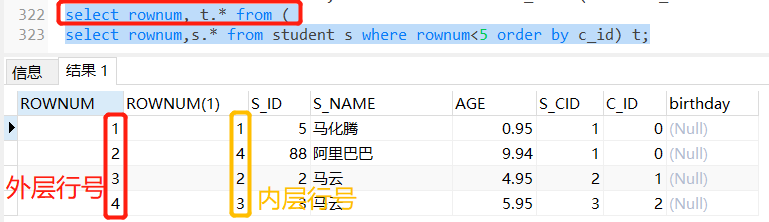


## 排序后赋予行号

如果要用到排序，也要用到行号，可以在外面套一层查询，再赋一次行号。

select rownum, t.\* from (

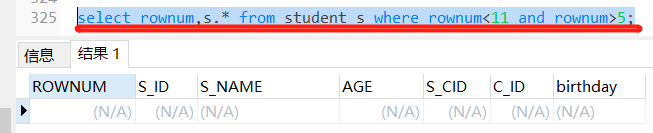
select rownum,s.\* from student s where rownum<5 order by c\_id) t;



## 隔页查询

问题：如果要查第二页的数据，每页5行数据，也就是查行号6-10的数据。

select rownum,s.\* from student s where rownum<11 and rownum>5;



没有查到数据的原因是rownum>5这个条件，因为，查数据先要满足where里的条件才能查到数据。查第一条数据的时候，rownum<11 这个条件是满足的，而rownum>5; 不满足，因为rownum是从1开始的，第一条数据的时候rownum肯定小于5.所以查询结果为空。

结论：不能写rownum大于某个正数。

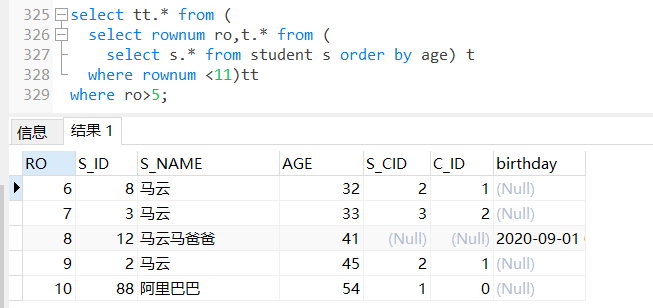
那么要解决上面的问题，就可以再嵌套一个查询。

select tt.\* from (

select rownum ro,t.\* from (

select s.\* from student s order by age) t

where rownum <11)tt

where ro>5; 

# 视图

## 视图的作用：

视图是建立在单表或者多表上的一张虚拟表，表内不存储数据，其数据来自组成视图的基表，视图仅仅是个窗口。

视图可以屏蔽掉基表的一些敏感字段，精简表，只展示需要的一些字段。

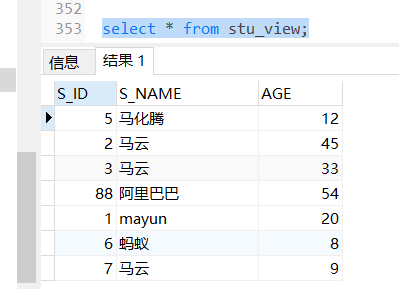
可以保证不同的人访问到的数据相同，只要基表的数据变了，通过视图查询的数据就会发生改变，不需要进行数据统一。

## 创建视图

create view stu\_view as select s\_id,s\_name,age from student;

## 查询视图

select \* from stu\_view;



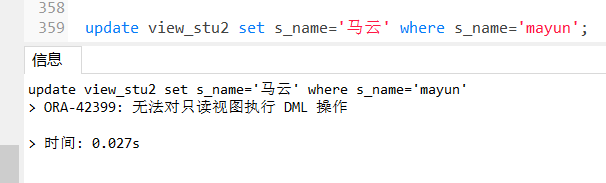
## 操作视图

视图中的数据都是来自于基表的，所以对视图的操作其实就是对基表的操作。视图的修改和表的修改是一样的。但是修改视图容易对基表造成混乱，不建议修改视图。

## 只读视图

只需要在普通视图创建时加上with read only就可

create view view\_stu2 as select s\_name,age from student with read only;



# 索引

在表的列上建立一个二叉树，达到大幅提升查询效率的目的，但是索引会影响增删改的效率。

索引分为单列索引和复合索引

## 单列索引

### 创建索引

create index idx\_ename on student(s\_name);

创建索引 指定索引名 指定在那张表的哪个字段上建立索引

### 单列索引触发规则

查询的条件必须是索引列中的原始值。

select \* from student where s\_name='马云';

单行函数、模糊查询等都不能触发索引

## 复合索引

### 创建复合索引

create index idx\_snameage on student(s\_name,age);

### 复合索引触发规则

复合索引中的第一列是优先检索列，要触发索引必须包含优先检索列中的原始数据，

select \* from student where s\_name='马云' and age='25'; 触发复合索引

如果复合索引的优先检索列又是一个单列索引，那么会优先触发单列索引

select \* from student where s\_name='马云'; 触发单列索引

如果是or连接条件，则相当于两个查询语句，其中一句触发索引，一句不触发索引，最终不触发索引。

select \* from student where s\_name='马云' or age='25'; 不触发索引

# 触发器

## DML触发器

只在增删改的时候才会触发 ，满足条件自动触发，不需要调用，不能接受参数

语句级触发器和行级触发器，行级触发器包含for each row 。

他们的区别是：行级触发器在每一行满足条件的行记录上都会触发一次，语句级触发器只会在满足条件的情况下触发一次。

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。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）。

### :new 和 :old

:new 访问修改之后的列值

:old 访问修改之前的列值

:new和:old在各个操作时能访问到的值

|  |  |  |  |
| --- | --- | --- | --- |
| 特性 | INSERT | UPDATE | DELETE |
| OLD | NULL | 实际值 | 实际值 |
| NEW | 实际值 | 实际值 | NULL |

### 触发器触发次序

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

### 语句级触发器

不包含 for each row 的触发器

create or replace trigger t1 or replace 表示可修改

after insert 指明了是在插入之后触发

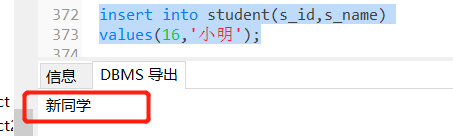
on student on表示在那张表上建立触发器

declare pl/sql块

begin

dbms\_output.put\_line('新同学'); 要执行的操作

end;



### 行级触发器

加了for each row 的触发器，加for each row 是为了使用：old或者：new对象或一行记录。

create or replace trigger t2 创建触发器

before update 在更新之后触发

on student 在student表上建立触发器

for each row 行级触发器

declare

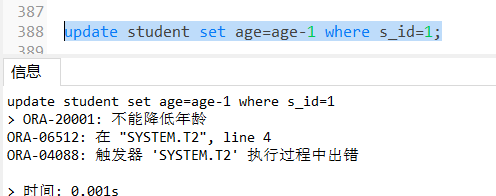
begin

if :old.age>:new.age then 触发之后的操作

raise\_application\_error(-20001,'不能降低年龄');

end if;

end;



### 使用触发器让主键自增

create or replace trigger autokey

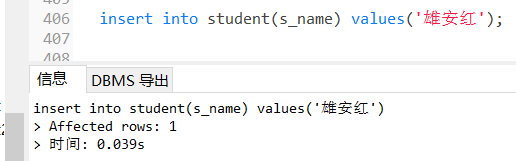
before insert on student

for each row

begin

select max(s\_id)+1 into :old.s\_id from student;

end; 用最大的id+1赋值给id,来使主键自增



## 删除触发器

drop trigger t2;

## 禁用触发器

Alter trigger t2 disable;

## 启用触发器

Alter trigger t2 enable;

## 禁用表内所有触发器

Alter table table\_name disable all triggers;

## 启用表内所有触发器

Alter table table\_name enable all triggers;

# 游标

游标相当于java中的集合，可以存放多个对象，多行记录。

## 不带参游标

declare

cursor c1 is select \* from student;

stu student%rowtype;

begin

open c1; open了游标就要close游标

loop

fetch c1 into stu; 用fetch来获取游标中的值

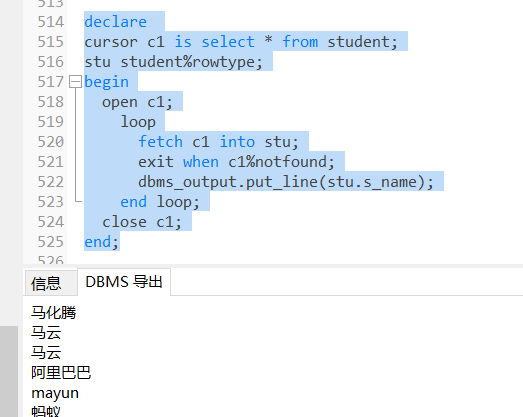
exit when c1%notfound;

dbms\_output.put\_line(stu.s\_name);

end loop;

close c1;

end;



## 带参游标

declare

cursor c2(ages student.age%type) is select s\_id from student where age>ages; ages是游标的参数名，后面的是类型，参数不能带长度

sid student.s\_id%type;

begin

open c2(20); 仅在打开游标的时候，需要给游标的参数赋值

loop

fetch c2 into sid;

exit when c2%notfound; 不要忘记循环退出条件

update student set age =age +1 where s\_id=sid;

end loop;

close c2;

end;

# 存储过程

存储过程是一段已经编译好的，存储在数据库端的pl/sql代码块，可以被直接调用。即使关掉数据库，存储过程也还在。

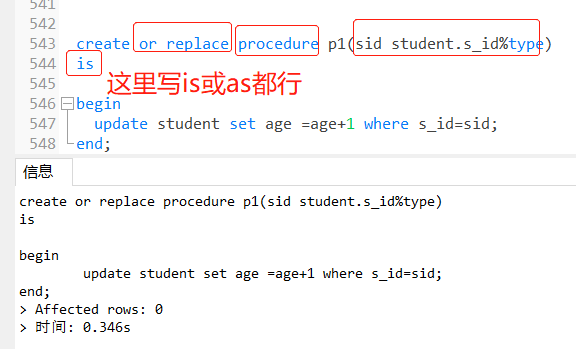
create or replace procedure p1(sid student.s\_id%type)

is or replace表示可以修改，参数sid不能带长度

begin

update student set age =age+1 where s\_id=sid;

end;



# 存储函数

存储函数和存储过程很相似，但是存储函数多了一个返回值。

存储函数在调用的时候，返回值需要接收。

create or replace function f1(sid student.s\_id%type) return number

as 这里的return后面的参数不能带长度

nextyear student.age%type;

begin

select age+1 into nextyear from student where s\_id=sid;

return nextyear;

end;