1. Oracle 的使用

1.1. SQLPLUS 的命令

初始化表的位置：
set NLS_LANG=american_america.us7ascii (设置编码才可以使用下面脚本)
cd $ORACLE_HOME/rdbms cd demo summit2.sql
****
我们目前使用的是 oralce 9i 9201 版本
select * from v$version;

恢复练习表命令：
sqlplus **/** @summit2.sql //shell 要在这个文件的位置。

登陆 oracle 的命令：
sqlplus 用户名/密码

show user 显示当前登陆的身份.
set pause on
set pause off 分页显示.

oracle 中默认日期和字符是左对齐，数字是右对齐
table or view does not exist ; 表或示图不存在

edit 命令用于自动打开 vi 修改刚修执行过的 sql 的命令。
修改方法二：
l 3 先定位到行 c /旧串/新串

执行出错时，利用错误号来查错误:
!oerr ora 942 (装完系统后会装一个 oerr 工具，用于通过错误号来查看错误的具体信息)

想在 sql 中执行 unix 命令时，把所有的命令前加一个!就可以， 或者 host(用于从 sql 从切换至 unix 环境中去)

/*** 初次使用时注意 ****
运行角本时的命令：
先切换到 unix 环境下，cd $oracle_home cd sqlplus cd demo 下面有两个角本建表语句。

@demobld.sql
sqlplus nanjing/nanjing @demobid.sql 直接运行角本，后面跟当前目录或者是绝对路径

保存刚才的 sql 语句： save 命令 第二次保存时要替换之前的角本 save 文件名 replace
把刚才保的 sql 重新放入 buffer 中

spool on 开启记录
spool off 关闭记录
spool 文件名 此命令会把所有的操作存在某个文件中去

常见缩写：
nls national language support 国家语言支持

1.2. SQL 的结构

|DDL 数据库定义
|DML 数据库管理
SQL－－Commit rollback
|DCL 数据库控制
|grant+revoke 权限管理

表分为：系统表（数据字典），用户表
注：知道数据字典可以更便于使用数据库。

1.3. SQL 语句

1.3.1. 纵向投影操作 select

select * from student;
select name||'('||id||')' EMPLOYEE from employee;
select name,salary*13 from employee;

NVL function
如果原来的数值是 null 的话，由指定数值替代。
select last_name,title,salary*NVL(commission_pct,0)/100 COMM from s_emp;

1.3.2. column 使用

column(col) columnName clear/format/heading/justify format
column salary format $9999999.00 设置数字显示形式
column name fromat a15; 设置字符串显示 15 个字符
column salary justify left/right/center 输出格式
column salary heading text 设置显示的字段名
column clear 清除格式
column last_name;显示该字段名所用的格式
column salary justify left format $99,999.00 (定义工资的显示形式)

1.3.3. order by

ORDER BY 排序 升序和降序 ASC 升序(默认) DESC 降序

```
select * from s_emp order by dept_id , salary desc
```

部门号升序，工资降序
关键字 distinct 也会触发排序操作。

```
select * from employee order by 1; //按第一字段排序
```

NULL 被认为无穷大。order by 可以跟别名。

1.3.4. where 选择操作（横向投影）

where 条件一定是根据某个字段来进行过滤操作.

```
select * from s_emp where dept_id=42; 查看部门号为 42 的所有员工
select * from s_emp where salary>1000 查看工资高于 1000 的所有员工
select salary from s_emp where first_name='Geroge' 找出名字为 Geroge 的员工的工资数
select table_name from user_tables where table_name='S_EMP'; 查某个具体表名时，表名的字符串必须要为大写
或者采用 upper(table_name)
select * from user_talbes where table_name like ‘s\_%’ escape ‘\’;
使用转义字符对关键字进行转义。
```

逻辑运算：
BETWEEN AND 在什么之间

NOT BETWEEN AND 注意区间：[ ]是一个闭区间
IN(LIST) 在某个集合中
NOT IN (list) 空值会有影响 (等于 list 其中任何一个就行，为提高效率常把比例高的放在前面)
LIKE 模糊配置
NOT LIKE 通配比较
IS NULL 是空
AND
OR
NOT
练习 3：(查出 s_emp 表中所有员工的一年的总收入)

```
select first_name , salary*12*( 1+nvl(commission_pct/100 , 0 ) ) " year salary "
from s_emp;
```

nvl 函数 专用于处理空值的影响.

练习 4:(找出表名以 S_开头的所有表)对于一些特殊字符，要用到 escape 转义,并不是一定要用\，escape 后面定义是什么字符为转义字符，那就用哪个字符

```
select table_name from user_tables where table_name like 'S\_%' escape '\';
```

1.3.5. 单行函数

单行函数： (dual 哑表)
字符函数：
lower 转小写 select lower('SQLPLUS') from dual;-->对纯字符串处理的时候
upper 转大写 select upper('sqlplus') from dual;
initcap 首字符大写 select initcap('tarena') from dual;
concat 连接字符串 select concat(first_name ， last_name) from s_emp;等效于||
substr 求子串 select substr('tarenasd0603' ,1,6) from dual; (取前六个字符) select substr('tarenasd0603',-2) from dual; (取后两个字符)
length 求字符长度
select length('tarena') from dual;
nvl 空值函数 两个参数的类型要匹配，统一的，表示：如果有，则返回前面的参数，如果没有就返回后面的参数
eg:select first_name,salary from s_emp where lower(first_name)='george';
select first_name , substr(first_name , -2) from s_emp; (查出 s_emp 表中所有用户名字的最后两个字符)
默认的是从左向右，如果是-2 则表示从右向左数
练 习 5 ： select first_name , salary from s_emp where

lower(first_name)='george';

数值函数：
round 函数(四舍五入) select round(45.935, 2) from dual; 不带参数时默认为 0 位小数
trunc 函数(截取，不管后面的数字) select trunc(45.995, 1) from dual;

日期函数：
oracle 数据库中存放时间格式的数据，是以 oracle 特定的格式存贮的，占 7 个字节，与查询时显示的时间格式无关，具体哪个字节表示什么，我不太清楚，请高手补充。存贮的时间包括年月日时分秒，最小精度为秒，不存贮秒以下的时间单位。因此在一些前台支持毫秒级的程序(如 PB 客户端程序)连接到 oracle 数据库时应注意这点。查询时显示的时间格式，由会话环境决定，或者由用户定义，与数据库无关。

select sysdate from dual; 从伪表查系统时间，以默认格式输出。
sysdate+(5/24/60/60) 在系统时间基础上延迟 5 秒
sysdate+5/24/60 在系统时间基础上延迟 5 分钟

sysdate+5/24 在系统时间基础上延迟 5 小时
sysdate+5 在系统时间基础上延迟 5 天
所以日期计算默认单位是天

内置函数：
months_between（sysdate,addmonth(sysdate,5)） //两个月有多少天。
add_months(sysdate,-5) 在系统时间基础上延迟 5 月
add_months(sysdate,-5*12) 在系统时间基础上延迟 5 年
last_day(sysdate) 一个月最后一天
next_day(sysdate,’Friday’) 下个星期星期几。
round (sysdate,’day’) 不是四除五入了，是过了中午留下，不过的略掉
trunc(sysdate,’month’) 不到一月都省略

例子：
上月末的日期：select last_day(add_months(sysdate, -1)) from dual;
本月的最后一秒：select trunc(add_months(sysdate,1),'month') - 1/24/60/60 from dual
本周星期一的日期：select trunc(sysdate,'day')+1 from dual
年初至今的天数：select ceil(sysdate - trunc(sysdate, 'year')) from dual;

格式转换函数：
to_char 显示日期：
从数字转化为 char to_char(date,'格式')
从日期转化为 char to_char(date, 'fmt')

yyyy	2007 年份
year	two thousand seven 年份
mm	03 (格式缩写显示也缩写)
month	march 月份
dy	fri 星期几缩写
day	Friday 星期几
dd	16 一个月第几天
mi	30 分钟
ss	35 秒钟
hh	18 小时
rr	07 年 最近时间
yy	07 年 当前世纪年份

```
select to_char(sysdate, 'yyyy mm dd hh24:mi:ss') from dual;
```

```
select to_char(sysdate, 'fmyyyy mm dd hh24:mi:ss') from dual;
```

查出三月分入职的员工：

```
select first_name,start_date from s_emp where to_char(start_date,'mm')='03';
```

to_date 表达日期：
字符转日期

```
select to_date('2000 11 20', 'yyyy mm dd ') from dual;

select round(to_date('10-OCT-06' ,'dd-mon-RR') ) from dual;
```

to_number
字符转数字

```
select to_number('10') from dual ;
```

函数、表达式、隐式数据类型转换会导致索引用不上，where 条件后面只能放单行函数，它起了一个过滤的的作用。

1.3.6. 组函数

group by 分组子句 对分组后的子句进行过滤还可以用 having 条件对分组后的条件进行过滤 where 是对记录进行过滤

avg(distinct | all)求平均值
count(distinct | all)统计

max(distinct | all) 求最大值
min(distinct | all)求最小值
sum(distinct | all) 求和
(所有组函数会忽略空值 , avg sum 只能作用于数字类型)
求有提成员工的提成的平均值;

```
select avg(nvl(commission_pct ,0 ) ) from s_emp;
```

有多少人有提成:

```
select count( commission_pct ) from s_emp ;
```

count(*) 用于统计记录数:

```
select sum(commission_pct)/ count(*) from s_emp;
```

员工分部在多少个不同的部门:count 默认为作 all 的动作

```
select count(dept_id) from s_emp;
select count(distinct dept_id) from s_emp;
```

求各个部门的平均工资:group by 子句也会触发排序

```
select dept_id , avg(salary) aa from s_emp group by dept_id
order by aa ; //对平均工资排序
select dept_id , avg(salary) aa from s_emp group by dept_id;
```

注意:group by 子句后面跟有条件只能是组函数查询的结果中的字段,所以我们会人为在结果要加入一些 group by 要用的字段,如:dept_id 可能不想要。

```
select region_id , count(*) from s_dept 此句会有错,count(*)是单组分组函数,如果加上 group by region_id 就是找出同地区的部门数量。
select max(region_id) , count(*) from s_dept; (强制语法上可以正确,但是不能保证结果也会正确)
```

求各个部门不同工种的平均工资:

```
select dept_id , title, avg(salary) from s_emp group by dept_id ,
title ;
```

哪些部门的平均工资比 2000 高:

```
select dept_id, avg(salary) aa from s_emp group by (dept_id) having
avg(salary)>2000;
```

除了 42 部门以外的部门的平均工资:

```
select dept_id , avg(salary) from s_emp group by (dept_id ) having
dept_id!=42;
```

```
select dept_id , avg(salary) from s_emp where dept_id!=42
group by (dept_id ) ;(此种 sql 效率要高，先过滤)
再计算）
```

where 单行函数。
having 组函数。
求各个部门的平均工资:
// 这样统计不详细

```
select max(d.name) , avg (s.salary) from s_emp s, s_dept d where
s.dept_id=d.id group by d.name;
```

//****这问题很经典，为了过 oracle sql 语法关而写 max(d.name) ***

```
select max(d.name) , avg(e.salary) , max(r.name) from s_emp e, s_dept
d , s_region r where e.dept_id = d.id and d.region_id=r.id
group by d.id ;
```

1.3.7. 多表连接

多表连接操作：
两表没有任何关联时会产生迪卡尔机：

```
select first_name , name from s_emp , s_dept;
```

1) 等值连接：

练习一:查看员工的姓名和员工部门号:(要考虑到表中实际数据中空值的影响）

```
select first_name , name from s_emp e, s_dept d where e.dept_id=d.id;
```

同时起了别名

```
select first_name , name from s_emp e, s_dept d where e.dept_id=d.id and
e.first_name='George';
```

具体到哪个人所在的部门
表的两边有空值的话，不会显示出来。
练习二：每个员工所在的部门和部门所在的地区

```
select first_name , s_dept.name, s_region.name from s_emp, s_dept, s_region
where
s_emp.dept_id=s_dept.id
and
```

```
s_dept.region_id=s_region.id;
```

等价于

```
select first_name,d.name,r.name
from s_emp e,s_dept d,s_region r
where e.dept_id=d.id and d.region_id=r.id;
```

等值连接：
练习三:找出每个员工和每个员工的工资级别

```
select a.ename , a.sal, b.grade from emp a , salgrade b where a.sal between b.losal and b.hisal;
select a.ename , a.sal, b.grade from emp a , salgrade b where a.sal>=b.losal
and a.sal<=b.hisal;
```

2) 自连接：

当一个表的插入行之间有了关系时就发生了(又名：内连接)
select first_name , manager_id from s_emp;
查出所有员工的部门领导的名称：(这种 sql 会少一条记录，总经理没有被配置上）

```
select e.first_name , m.first_name from s_emp e , s_emp m where
e.manager_id=m.id; //关键是同一张表用不同的别名
```

3) 外连接：

(防止空值忽略，用（+）的一方会模拟一条记录配置另一方)这就称为外连接,不加（+）一个记录都不能少；

```
select e.first_name , m.first_name from s_emp e , s_emp m where
e.manager_id=m.id(+);
```

+号放在哪边就表示在哪边补空，来跟对方来匹配，使得数据一个都不会漏掉，这个例子中的领导有可能会没有(最高领导就再没有领导了，所以就
＋号放在可能出现空值的一边)
标准写法：
显示没有员工的部门

```
select distinct d.name
from
s_emp e,s_dept d
```

```
where
e.dept_id(+)=d.id
and
e.dept_id is null
```

显示有员工的部门：

```
select distinct d.name
from
s_emp e,s_dept d
where
e.dept_id=d.id
```

1.3.8. 子查询

关于子查询: Subqueries
找出所有员工中，工资最低的那个员工：(利用子查询)

```
select first_name, salary from s_emp where salary = ( select min(salary) from s_emp) ;
```

```
select max(first_name), min(salary) from s_emp;
//这样写会出错姓名和工资不一致
```

子查询运行的顺序： 先运行子查询再运行主查询，子查询一般出现在运算符的右边。
单值运算符：运算后面只能跟一个值
多值运算符：可以对两个以上的值进行操作
查询谁跟 Smith 干一样的活：
1．先找出 Smith 干什么工作

```
select title from s_emp where last_name='Smith';
```

2．再在所有数据中找和 Smith 工作匹配的人
//下种写法可能还存在 bug，没有考虑到数据的全面性,有潜在性问题

```
select last_name , title from s_emp where title =( select title from s_emp where last_name='Smith' ) and last_name <> 'Smith' ;
```

//这种写法才考虑的比较全面

```
select last_name , title from s_emp where title in ( select title
from s_emp where last_name='Smith' ) and last_name <> 'Smith' ;
```

使用子查询时应注意： 单行子查询返回多个结果时会有错误 single-row subquery returns more than one value

查出哪些员工的工资比平均工资低：

```
select e.first_name , e.salary
from s_emp e
where
e.salary<
(select avg( nvl( salary,0 ) )
from s_emp)
```

哪些部门的平均工资比所有部门的平均工资要低：

第一步先查出各个部门的平均工资：

```
select min(avg(salary)) from s_emp group by dept_id;
```

第二步再查出哪个部门的工资是最低的:

```
select dept_id, avg(salary) from s_emp group by dept_id having
avg(salary) = (select min(avg(salary) ) from s_emp group by dept_id ) ;
```

哪个部门里没有员工：

```
select deptno from dept where deptno not in ( select
deptno from emp );
```

哪些人是普通员工：（用子查询形式来做）

```
select first_name,last_name from s_emp
where id not in
(select distinct(manager_id) from s_emp where manager_id is not null);
```

1.3.9. 语言环境的设置

```
$ echo $LANG
zh_CN.hp15CN
$ echo $NLS_LANG
simplified chinese_china.zhs16cgbk
```

1.3.10. E--R 图 实体关系图 entity relation

开发流程先进行需求分析，进行系统设计，建表，再进行开发编码,测试最终产品上线试运行。
把软件设计模型转化为数据中的表，设计时要考虑性能的设计

第一范式：最简单的一种建方式，一张表只有一个主键。
第二范式：表的自连接存在原因，一张表，学生表中也有班级的信息。
第三范式：表连接存在的原因，两张表，其中一张表引用其它一张表。

约束：
为了保证数据的一致性，
primary key (pk) 主键约束 不允许有重复和空值(唯一且非空）
foregin key (fk) 外键约束 两张表 parent table child table
unique key (uk) 唯一可以为空
not null
数据库设计时的注意：
索引： 为了提高效率而设计的一种与业务无关的
考虑表点用的物理空间：
考虑表之间的关系：
一对多关系： 利用 FK+PK 实现，多的一方引用外键
一对一关系： 可以利用 FK+UK 实现，
多对多关系： 通过中间增加一个附加表来实现，附加表利用联合主键来实现，联合起来的主键唯一。

1.3.11. DDL 语句：

DDL (DataBase Defind Language)数据库定义语句：
table (表）
view(示图）
sequence(序列号）
index(索引）

创建表语句：
create table [schema].表名 (字段名， 字段类型 约束条件);
schema 默认就是当前用户,严格来访问表名完整的写法是 schema.tablename
数据类型：
表名的命令规则： 首字母为字母，不得超过 30 个字符
char(size) 定长 不管是否达到最大宽度，都会点最大的宽度。
varchar2(size) 可变长 按实际的字节占用空间
number 所有的数字类型都称为 number

number(n, m) n------n 位宽度 m-----小数点后的宽度
number(2,4) 小数点后 4 位，有效位 2 位 values(0.0099) 这样可以 values(0.01)这样出错
LONG 2GB 大文本一个表最我只允许定义一个 LONG 类型(不建议使用）
CLOB 大对象形式存放(在表里只存一个指针)
BLOB 存二进制大对象(声音，图像之类）

default 作用演示：
create table test(c1 number default 10, c2 number);

1.3.12. 表的约束

主键约束的定义：

```
create table test(c number primary key ); 列级约束
create table test(c number , primary key(c) ) ; 表级约束
create table test( c1 number constraints pkc1 primary key );
```

此约束有名字: pkc1

```
create table test(c number , c2 number , primary key (c ,c1) ) ;
```

用表级约束可以实现联合主键

外键约束的定义：(先定义父表，再定义子表）

```
carete table parent(c1 number primary key );
create table child (c number primary key , c2 number references parent(c1)); //列约束
create table child( c number primary key , c2 number , foreign key(c2) references parent(c1)); //表约束
```

```
on delete cascade (及联删除，删除父表时子表也跟着删除）
on delete set null (及联删除父表时子表中引用的字段为 null）
create table child(id number, parent_id number(10) constraint child_fk references parent(id) ) on delete cascade
create table child(id number, parent_id number(10) constraint child_fk references parent(id) ) on delete set null
```

不给约束起名字时，系统给约束起名时的规律为：数据库用户名_数字(约束名也

不能重名）
定义一个约束的两种形式：
列级约束 表级约束

非空约束：
not null (利用 desc 可能看到）primary key 自动具有非空约束的特点

非主键和外键
check 约束：
create table test(c1 number check(c1>1000));
此表中要求 c1 的值必须要大于 1000 才为有效值 .

复制表的语法（仅限于 Oracle）：

```
create table child1 as select * from child;
```

利用已知表建一张新表：只会把非空约束带过来，其它约束要自己添加

```
create table s_emp_42 as select * from s_emp where dept_id = 42;
```

只取要表结构，不想要表中数据的建表方式：

```
create table s_emp_copy as select * from s_emp where 1=2;
```

(这是一个小技巧,利用永不等式提取零条数据，在 JDBC 的学习中会用到 where 1=1 的形式,注意体会）

查看一张表的约束：(查数据字典示图)
desc user_constraints;(这个数据字典中会查到相应的信息）

```
select constraint_name, constraint_type from user_constraints where table_name='S_EMP';
```

P pk
R fk
C check
U UK
V 这种只定义在示图中(with check option 相当于组示图加了一个约束）
O 也是出现在示图中
非空约束和 CHECK 都是用 C 来表示

查看字段约束的方法：

```
desc user_cons_columns;
```

```
select column_name, position from user_cons_columns where
constraint_name='S_EMP_ID_PK' ;
```

position 的含义：联合主键，约束名一样。
user_constraints user_cons_columns 两张表的约束名相等，表名相等，两张表一关联就可以查出所需的信息。

```
select constraint_name , r_constraint_name from user_constraints where
constraint_type='R' and table_name='S_EMP' ;
```

数据库建立时，数据字典就会建好。
user_constraints; 自己拥有的
all_constraints; 你自己拥有的加上你可以访问的
dba_constraints 所有的

查看当前数据库数据字典的字典（这个示图很重要）

```
desc dict;
select table_name form dict where table_name like '%cons%;
```

示图：

```
user_objects; user_tables;
select distinct object_type from user_objects;
```

1.3.13. 介绍事务的概念：

commit 提交，此时说明前面所有语句都成功执行
rollback 回退操作，此时会恢复至上一次提交时的状态。
savepoint 设置保存点

注意 insert into 后面可以跟子查询

```
insert into s_emp_42 select * from s_emp where dept_id =42;
```

UPDATE 修改字段值：

```
update s_emp set dept_id =10 where id =2 ;
update s_emp set commission_pct =10 ;
```

没有 where 条件时说明是改表中所有的值.
注意：如有外键引用时常会出现外键引用值没有找到等错误？

delete 删除记录命令语法：

```
delete from    s_emp   where   dept_id=42;
delete form    s_emp ;          没有 where 条件时说明删除表中所有的值
```

注意：如有外键引用时，删除一张表时常会出现不能删除的情况，
原因一　是因为此时正在有人操作表中记录
原因二　此表有其他的表引用，没能设及联删除：
delete 删除一张大表时空间不释放，非常慢是因为占用大量的系统资源，支持回退操作，空间还被这张表占用着。
truncate table 表名 (删除表中记录时释放表空间)

DML 语句：
表级共享锁： 对于操作一张表中的不同记录时，互不影响
行级排它锁：对于一行记录，oracle 会只允许只有一个用户对它在同一时间进行修改操作
wait() 等到行级锁被释放，才进行数据操作
drop 一张表时也会对表加锁，DDL 排它锁,所以在删除一张表时如果当前还有用户操作表时不能删除表

alter table 命令用于修改表的结构(这些命令不会经常用)：
增加约束：

```
alter table   表名 add    constraint    约束名   primary key   (字段）;
```

解除约束：(删除约束)
alter table 表名 drop primary key（对于主键约束可以直接用此方法，因为一张表中只有一个主键约束名，注意如果主键此时还有其它表引用时删除主键时会出错）

```
alter table   father   drop   primary key     cascade ;  (如果有子表引用主键时，要用此语法来删除主键,这时子表还存在只是子表中的外键约束被及联删除了）
```

alter table 表名 drop constraint 约束名;
(怎样取一个约束名：1、人为的违反约束规定根据错误信息获取!
2、查询示图获取约束名!)

alter table 表名 disable from primary key ; (相当于把一个表的主键禁用)
alter table 表名 enable primary key ;（enable 时会自动去检查表的记录是不是符合要求,如果有脏数据时必须要先删除脏数据才可以 enable）

****

增加字段：
alter table 表名 add(字段字 字段类型)
删除字段：
alter table 表名 drop(字段)
alter tbale 表名 drop column 字段 ; (8i 以后才支持）
给列改名:920 才支持

```
alter table 表名 rename column 旧字段名 to 新字段名;
```

修改字段
(此时应注意的问题，更改时要看具体值情况之间的转达换， 改为字符类型时，必须要为空)

```
alter table 表名 modify( 字段，类型)
```

更改表中的字段：
update 表名 set 字段 = 值 where 条件
更改表名
rename 旧表名 to 新表名；
删除表：
trucate table 表名:(表结构还在，数据全部删除，释放表所占的空间，不支持回退,常用删除大表）

1.3.14. oralce 中产生序列(sequence):

create sequence 序列名;
(不带参数时默认为从 1 开始每次递增 1，oracle 中为了提高产生序列的效率一般一次性产生 20 个序列放入当前会话的序列池中备用以加快效率，序列会出现不连续的动作回退操作不会影响序列取值）
sequence 的参数：
increment by n 起始值， start with n 递增量，maxvalue n 最大值，minvalue n 最小值，cycle | no cycle 轮回， cache n 绶存(第一次取时会一次取多少个 id 存起来）
查看 sequence 示图：

```
desc user_sequences ;
select sequence_name , cache_size , last_number from user_sequences
where sequence_name like 's_';
select 序列名.currval from dual 查看当前的序列数
第一次执行这个命令会出错，因为还没有序列号放进序列池中，也就是说
currval 读取的是序列池中的数值。
select 序列名.nextval from dual 查看下一个序列数，它会自动给当前的序列加 1 ，实际上是给序列池中的数值加一，而序列表中加的是缓存的值。
```

清空当前会话的内存：

alter system flush shared_pool;（执行此命令要有 DBA 权限，一般用户执行出错）

修改序列：(此命令不常用，只需了解就行不必深究）
alter sequence 序列名 修改项；
删除序列 sequence
drop sequence 序列名;

1.3.15. 创建示图

creating views（属于了解知识）

```
desc user_views;
select text from user_views where view_name='TEST1_V1' ;
```

示图就相当于一条 select 语句,定义了一个示图就是定义了一个 sql 语句,示图不占空间,使用 view 不会提高性能，但是能简单化 sql 语句
（扩展知识： oracle 8i 以后的新示图）MV 物化视图(占存储空间，把 select 结果存在一个空间，会提高查询视图,增强实时性，但是存在刷新问题， 主要应用在数据仓库中用要用于聚合表）
使用示图的好处：控制数据访问权限.
如何创建一个示图：

```
create or replace views test_vi as select * from test1 where c1=1;
```

此时往表 test1（base table 基表）中插入数据时：表中没能变化，示图中的数据发生改变
从示图中插数据时相对应的表会发生改变:
往示图中插数据时，会直接插进基表中，查看示图中的数据时，相当于就是执行创建时的 select 语句。
简单示图：能进行 DML 操作。
复杂示图：来源于多张表，不能执行 DML 操作。

1.3.16. 关于利用 rownum 分页显示

rownum 有个特点要么等于 1 要么小于某个值， 不能直接等于某个值, 不能大于某个值。rownum 常用于分页显示。
练习：查询出第 5 条数据和第 10 条数据之间：

```
select first_name , rnum from ( select rownum rnum , first_name from s_emp where rownum <=10 ) where rnum between 5 and 10 ;
```

分面显示：

```
SELECT * FROM (SELECT a.*, rownum r FROM S_EMP a ) WHERE r
between 5 AND 10;
```

练习：哪些员工的工资比本部门的平均工资高？

```
select first_name , salary , avgsal from s_emp e , ( select
dept_id , avg (salary ) avgsal from s_emp group by dept_id ) a
where e.dept_id =a.dept_id and e.salary > a.avgsal;
```

在示图上加一个 with check option 就相当于给示图加上了约束

```
create view test_v as select * from test where c =1 with
check option ;
create view test_v as select * from test with read option; 定义在视图上的约束，
只读。
```

同义词:相当于别名的作用(***只需了解***）系统自建的同义词： user_tables
create synonym asd_s_emp for asd_0607.s_emp ;
目的就是为了给 asd_0607_s_emp 表起另一个代替的名称 asd.s_emp;注意这个同义词只能自己使用;
create public synonym p_s_emp fro asd_0607.s_emp; 创建公共的同义词，但是要权限.
删除同义词：
drop synonym 同义词名称

1.3.17. 创建索引

Creating indexes(概念很重要对系统的性能影响非常大）
建索引的目的就是为了加快查询速度，使表中数据在做 insert 语句时就进行排序。
索引是 DML 语句触发的，系统自动运行，也可以强制使用索引。
索引的结构
key | rowid

索引就相于一本的书的目录。索引点系统空间，属于表的附属物。删除一个表时，相对应的索引也会删除。truncate 表时索引结构在，但是数据不存在。
full table scan 全表扫描
用索引就是为了快速定位数据：(理解时就以字典的目录为例）
查看表的 rowid：
select rowid , first_name from s_emp;

rowid 定义的信息有： object block table
每条记录都有自己的 rowid ,rowid 记录了这条记录在数据库中的位置，可以快速读取。
索引由谁创建：用户,建索引后会使 DML 操作效率慢，但是对用户查询会提高效率,这就是我们建索引的最终目的，
创建一个索引：
create index 索引名 on 表名 (字段名）;
create insex testindex on test(c1, c2);
哪些字段应该建索引：
经常要用 where 的子句的地方，所以要用索引.用不用索引，关键要看所查询的数据与所有数据的百分比，表越大，查询的记录越少，索引的效率最高.

替换变量：用&符号来定义替换变量支持交互性提示,对于字符性的数字，一定要写在单引号之间
set verify on
set verify off;
相当于开关变量，用于控制是否显示新旧的 sql 语句
select id ,last_name ,salary from s_emp where title='&job_title';
更改交互的提示信息：
accept p_dname prompt ' 提示信息';
定义变量：
define p_dname='abc';

分页的实现语句：(可以正常运行)
select * from (select rownum rnum , a.* from (select * from s_emp) a) where rnum between 5 and 10 ;

1.4. SQL 语句使用技巧

1．删除表空间

```
DROP TABLESPACE TableSpaceName [INCLUDING CONTENTS [AND DATAFILES]]
```

2．删除用户

drop user XXXX cascade;

3．删除表的注意事项

在删除一个表中的全部数据时，须使用 TRUNCATE TABLE 表名;因为用 DROP TABLE，DELETE * FROM 表名时，TABLESPACE 表空间该表的占用空间并未释放，反复几次 DROP，DELETE 操作后，该 TABLESPACE 上百兆的空间就被耗光了。

4．having 子句的用法

having 子句对 group by 子句所确定的行组进行控制，having 子句条件中只允许涉及常量,聚组函数或 group by 子句中的列。

5．外部联接"+"的用法

外部联接"+"按其在"="的左边或右边分左联接和右联接.若不带"+"运算符的表中的一个行不直接匹配于带"+"预算符的表中的任何行,则前者的行与后者中的一个空行相匹配并被返回.若二者均不带'+',则二者中无法匹配的均被返回.利用外部联接"+",可以替代效率十分低下的 not in 运算,大大提高运行速度.例如,下面这条命令执行起来很慢
用外联接提高表连接的查询速度
在作表连接(常用于视图)时，常使用以下方法来查询数据:

```
SELECT PAY_NO, PROJECT_NAME
FROM A
WHERE A.PAY_NO NOT IN (SELECT PAY_
NO FROM B WHERE VALUE >=120000);
```

---- 但是若表 A 有 10000 条记录，表 B 有 10000 条记录，则要用掉 30 分钟才能查完，主要因为 NOT IN 要进 行一条一条的比较，共需要 10000*10000 次比较后，才能得到结果。该用外联接后，可以缩短到 1 分左右的时间:

```
SELECT PAY_NO,PROJECT_NAME
FROM A,B
WHERE A.PAY_NO=B.PAY_NO(+)
AND B.PAY_NO IS NULL
AND B.VALUE >=12000;
```

6．set transaction 命令的用法

在执行大事务时,有时 oracle 会报出如下的错误:
ORA-01555:snapshot too old (rollback segment too small)
这说明 oracle 给此事务随机分配的回滚段太小了,这时可以为它指定一个足

够大的回滚段,以确保这个事务的成功执行.例如

```
set transaction use rollback segment roll_abc;
delete from table_name where ...
commit;
```

回滚段 roll_abc 被指定给这个 delete 事务,commit 命令则在事务结束之后取消了回滚段的指定.

7．数据库重建应注意的问题

在利用 import 进行数据库重建过程中,有些视图可能会带来问题,因为结构输入的顺序可能造成视图的输入先于它低层次表的输入,这样建立视图就会失败.要解决这一问题,可采取分两步走的方法:首先输入结构,然后输入数据.命令举例如下 (uesrname:jfcl,password:hfjf,host sting:ora1,数据文件:expdata.dmp):

```
imp jfcl/hfjf@ora1 file=empdata.dmp rows=N
imp jfcl/hfjf@ora1 file=empdata.dmp full=Y buffer=64000
commit=Y ignore=Y
```

第一条命令输入所有数据库结构,但无记录.第二次输入结构和数据,64000 字节提交一次.ignore=Y 选项保证第二次输入既使对象存在的情况下也能成功.

```
select a.empno from emp a where a.empno not in (select empno from emp1 where
job='SALE');
```

倘若利用外部联接,改写命令如下:

```
select a.empno from emp a ,emp1 b
where a.empno=b.empno(+)
and b.empno is null
and b.job='SALE';
```

可以发现,运行速度明显提高.

8．从已知表新建另一个表：

```
CREATE TABLE b
AS SELECT * (可以是表 a 中的几列)
FROM a
WHERE a.column = ...;
```

9．查找、删除重复记录:

法一: 用 Group by 语句 此查找很快的

```
select count(num), max(name) from student --查找表中 num 列重复的，列出重复的记录数，并列出他的 name 属性
group by num
having count(num) >1 --按 num 分组后找出表中 num 列重复，即出现次数大于一次
delete from student(上面 Select 的)
```

这样的话就把所有重复的都删除了。-----慎重
法二:当表比较大(例如 10 万条以上)时,这个方法的效率之差令人无法忍受,需要另想办法:
---- 执行下面 SQL 语句后就可以显示所有 DRAWING 和 DSNO 相同且重复的记录

```
SELECT * FROM EM5_PIPE_PREFAB
WHERE ROWID!=(SELECT MAX(ROWID) FROM EM5_PIPE_PREFAB D --D 相当于 First,Second
WHERE EM5_PIPE_PREFAB.DRAWING=D.DRAWING AND
EM5_PIPE_PREFAB.DSNO=D.DSNO);
```

---- 执行下面 SQL 语句后就可以删除所有 DRAWING 和 DSNO 相同且重复的记录

```
DELETE FROM EM5_PIPE_PREFAB
WHERE ROWID!=(SELECT MAX(ROWID) FROM EM5_PIPE_PREFAB D
WHERE EM5_PIPE_PREF
AB.DRAWING=D.DRAWING AND
EM5_PIPE_PREFAB.DSNO=D.DSNO);
```

10．返回表中[N，M]条记录：

取得某列中第 N 大的行

```
select column_name from
(select table_name.*,dense_rank() over (order by column desc) rank from
table_name)
where rank = &N;
```

假如要返回前 5 条记录：

```
select * from tablename where rownum<6;(或是 rownum <= 5 或是 rownum != 6)
```

假如要返回第 5-9 条记录：

```
select * from tablename
where …
```

```
and rownum<10
minus
select * from tablename
where …
and rownum<5
order by name
```

选出结果后用 name 排序显示结果。(先选再排序)
注意：只能用以上符号(<、<=、!=)。

```
select * from tablename where rownum != 10;返回的是前 9 条记录。
```

不能用：>,>=,=,Between...and。由于 rownum 是一个总是从 1 开始的伪列，Oracle 认为这种条件 不成立，查不到记录.
另外，这个方法更快：

```
select * from (
select rownum r,a from yourtable
where rownum <= 20
order by name )
where r > 10
```

这样取出第 11-20 条记录!(先选再排序再选)
要先排序再选则须用 select 嵌套：内层排序外层选。
rownum 是随着结果集生成的，一旦生成，就不会变化了；同时,生成的结果是依次递加的，没有 1 就永远不会有 2!
rownum 是在 查询集合产生的过程中产生的伪列，并且如果 where 条件中存在 rownum 条件的话，则:
1： 假如 判定条件是常量，则：
只能 rownum = 1, <= 大于 1 的自然数， = 大于 1 的数是没有结果的， 大于一个数也是没有结果的
即 当出现一个 rownum 不满足条件的时候则 查询结束 this is stop key!
2: 当判定值不是常量的时候
若条件是 = var , 则只有当 var 为 1 的时候才满足条件，这个时候不存在 stop key ,必须进行 full scan ,对每个满足其他 where 条件的数据进行判定
选出一行后才能去选 rownum=2 的行……

11．快速编译所有视图

---- 当在把数据库倒入到新的服务器上后(数据库重建)，需要将视图重新编译一遍，因为该表空间视图到其它表空间的表的连接会出现问题，可以利用 PL/SQL 的语言特性，快速编译。

```
SQL >SPOOL ON.SQL
SQL >SELECT ‘ALTER VIEW ‘||TNAME||’
```

```
COMPILE;’ FROM TAB;
SQL >SPOOL OFF
```

然后执行 ON.SQL 即可。
SQL >@ON.SQL
当然，授权和创建同义词也可以快速进行，如：

```
SQL >SELECT ‘GRANT SELECT ON ’
||TNAME||’ TO USERNAME;’ FROM TAB;
SQL >SELECT ‘CREATE SYNONYM
‘||TNAME||’ FOR USERNAME.’||TNAME||’;’ FROM TAB;
```

12．读写文本型操作系统文件

---- 在 PL/SQL 3.3 以上的版本中，UTL_FILE 包允许用户通过 PL/SQL 读写操作系统文件。如下：

```
DECALRE
FILE_HANDLE UTL_FILE.FILE_TYPE;
BEGIN
FILE_HANDLE:=UTL_FILE.FOPEN(
‘C:\’,’TEST.TXT’,’A’);
UTL_FILE.PUT_LINE(FILE_HANDLE,’
HELLO,IT’S A TEST TXT FILE’);
UTL_FILE.FCLOSE(FILE_HANDLE);
END;
```

13．在数据库触发器中使用列的新值与旧值

---- 在数据库触发器中几乎总是要使用触发器基表的列值，如果某条语句需要某列修改前的值，使用:OLD 就可以了，使用某列修改后的新值，用:NEW 就可以了。如:OLD.DEPT_NO,:NEW.DEPT_NO。

14．数据库文件的移动方法

当想将数据库文件移动到另外一个目录下时，可以用 ALTER DATABASE 命令来移动(比 ALTER TABLESPACE 适用性强)：
1. 使用 SERVER MANAGER 关闭实例.

```
SVRMGR > connect internal;
SVRMGR > shutdown;
SVRMGR >exit;
```

2. 使用操作系统命令来移动数据库文件位置(假设这里操作系统为 SOLARIS

2.6). 在 UNIX 中用 mv 命令可以把文件移动到新的位置，

```
#mv /ora13/orarun/document.dbf /ora12/orarun
```

3. 装载数据库并用 alter database 命令来改变数据库中的文件名.

```
SVRMGR > connect internal;
SVRMGR > startup mount RUN73;
SVRMGR > alter database rename file
> ‘/ ora13/orarun/document.dbf’
> ‘/ ora12/orarun/document.dbf’;
```

4. 启动实例.

```
SVRMGR > alter database open;
```

15．连接查询结果：

表 a 列 a1 a2
记录 1 a
1 b
2 x
2 y
2 z
用 select 能选成以下结果:
1 ab
2 xyz
下面有两个例子：
1.使用 pl/sql 代码实现，但要求你组合后的长度不能超出 oracle varchar2 长度的限制

```
create or replace type strings_table is table of varchar2(20);
/
create or replace function merge (pv in strings_table) return varchar2
is
ls varchar2(4000);
begin
for i in 1..pv.count loop
ls := ls || pv(i);
end loop;
return ls;
end;
/
create table t (id number,name varchar2(10));
insert into t values(1,'Joan');
insert into t values(1,'Jack');
```

```
insert into t values(1,'Tom');
insert into t values(2,'Rose');
insert into t values(2,'Jenny');
column names format a80;
select t0.id,merge(cast(multiset(select name from t where t.id = t0.id) as
strings_table)) names
from (select distinct id from t) t0;
drop type strings_table;
drop function merge;
drop table t;
```

2.纯粹用 sql：
表 dept, emp
要得到如下结果
deptno, dname, employees

10, accounting, clark;king;miller
20, research, smith;adams;ford;scott;jones
30, sales, allen;blake;martin;james;turners
每个 dept 的 employee 串起来作为一条记录返回

```
This example uses a max of 6, and would need more cut n pasting to do more than
that：
SQL> select deptno, dname, emps
2 from (
3 select d.deptno, d.dname, rtrim(e.ename ||', '||
4 lead(e.ename,1) over (partition by d.deptno
5 order by e.ename) ||', '||
6 lead(e.ename,2) over (partition by d.deptno
7 order by e.ename) ||', '||
8 lead(e.ename,3) over (partition by d.deptno
9 order by e.ename) ||', '||
10 lead(e.ename,4) over (partition by d.deptno
11 order by e.ename) ||', '||
12 lead(e.ename,5) over (partition by d.deptno
13 order by e.ename),', ') emps,
14 row_number () over (partition by d.deptno
15 order by e.ename) x
16 from emp e, dept d
17 where d.deptno = e.deptno
18 )
19 where x = 1
20 /

DEPTNO DNAME EMPS
```

------- ---------- --
10 ACCOUNTING CLARK, KING, MILLER
20 RESEARCH ADAMS, FORD, JONES, ROONEY, SCOTT, SMITH
30 SALES ALLEN, BLAKE, JAMES, MARTIN, TURNER, WARD

16．在 Oracle 中建一个编号会自动增加的字段,以利于查询

1、建立序列：

```
CREATE SEQUENCE checkup_no_seq
NOCYCLE
MAXVALUE 9999999999
START WITH 2;
```

2、建立触发器：

```
CREATE OR REPLACE TRIGGER set_checkup_no
BEFORE INSERT ON checkup_history
FOR EACH ROW
DECLARE
next_checkup_no NUMBER;
BEGIN
--Get the next checkup number from the sequence
SELECT checkup_no_seq.NEXTVAL
INTO next_checkup_no
FROM dual;
```

```
--use the sequence number as the primary key
--for the record being inserted
:new.checkup_no := next_checkup_no;
END;
```

17．查看对象的依赖关系(比如视图与表的引用)

查看视图：dba_dependencies 记录了相关的依赖关系
查东西不知道要查看哪个视图时，可以在 DBA_Objects 里看，

```
select object_name from dba_objects where object_name like '%ROLE%'(假如查看 ROLE 相关)
```

然后 DESC 一下就大体上知道了。

18．要找到某月中所有周五的具体日期

```
select to_char(t.d,'YY-MM-DD') from (
select trunc(sysdate, 'MM')+rownum-1 as d
from dba_objects
where rownum < 32) t
where to_char(t.d, 'MM') = to_char(sysdate, 'MM') --找出当前月份的周五的日期
and trim(to_char(t.d, 'Day')) = '星期五'
```

```
--------
03-05-02
03-05-09
03-05-16
03-05-23
03-05-30
```

如果把 where to_char(t.d, 'MM') = to_char(sysdate, 'MM')改成 sysdate-90，即为查找当前月份的前三个月中的每周五的日期。

1.4.1. 建立用户

用户名: oracle
密码: oracle
sqlplus '/as sysdba'

echo $ORACLE_SID

```
create user nanjing identified by nanjing;
grant connect,resource,create session,create table to nanjing;
```

2. 数据字典的使用

1．表：

```
select * from cat; //显示所有用户对象。
select * from tab; //显示所有用户表。
select table_name from user_tables; //显示所有用户表
```

2．视图：

```
select text from user_views where view_name=upper('&view_name');
```

//显示某个视图

3．索引：

```
select index_name,table_owner,table_name,tablespace_name,status from
user_indexes order by table_name;
```

//显示用户索引

4．触发器：

```
select trigger_name,trigger_type,table_owner,table_name,status from user_triggers;
```

//显示用户触发器

5．快照：

```
    select owner,name,master,table_name,last_refresh,next from user_snapshots
order by owner,next;
```

6．同义词：

```
    select * from syn;
```

7．序列：

```
    select * from seq;
```

8．数据库链路：

```
    select * from user_db_links;
```

9．约束限制：

```
select
TABLE_NAME,CONSTRAINT_NAME,SEARCH_CONDITION,STATUS
from user_constraints WHERE TABLE_name=upper('&TABLE_Name');
```

10．本用户读取其他用户对象的权限：

```
select * from user_tab_privs;
```

11．本用户所拥有的系统权限：

```
select * from user_sys_privs;
```

12．用户：

```
select * from all_users order by user_id;
```

13．表空间剩余自由空间情况：

```
select tablespace_name,sum(bytes) 总字节数,max(bytes),count(*) from
dba_free_space group by tablespace_name;
```

14．数据字典：

select table_name from dict order by table_name;
可以通过数据字典表查询 其他系统表
select table_name from dict where table_name = ‘ ***%’order by table_name

15．锁及资源信息：

```
select * from v$lock;不包括 DDL 锁
```

16．数据库字符集：

select name,value$ from props$ where name='NLS_CHARACTERSET';
simplified chiniese_china.zhs16GBK 简体中午编码

17．inin.ora 参数：

```
select name,value from v$parameter order by name;
```

18．SQL 共享池：

```
select sql_text from v$sqlarea;
```

19．数据库：

```
select * from v$database
```

20．控制文件：

```
select * from V$controlfile;
```

21．重做日志文件信息：

```
select * from V$logfile;
```

22．来自控制文件中的日志文件信息：

```
select * from V$log;
```

23．来自控制文件中的数据文件信息：

```
select * from V$datafile;
```

24．NLS 参数当前值：

```
select * from V$nls_parameters;
```

25．ORACLE 版本信息：

```
select * from v$version;
```

26．描述后台进程：

```
select * from v$bgprocess;
```

27．查看版本信息：

```
select * from product_component_version;
```

28．查询 oracle 中所有用户信息

```
select   * from dba_users;
```

29．只查询用户和密码

```
select username,password from dba_users;
```

30．查询当前用户信息

```
select * from dba_ustats;
```

3. PL/SQL 语言

Procedure Language Structure Qurrey Language－－PL/SQL

3.1. Oracle 的变量类型

数据类型
　　数字，字符，布尔，日期
组合类型
　　Record(不同数据集合),table（一张表）,varray（数组）
参考型
　　Ref cursor, ref object_type

数字类型
　　Binary_integer,Dec,Float
　　Number
　　Number(3)
　　Number(4,3)　前面的总长，后面是小数的长度
字符类型
　　Char(不可变长), varchar（过时） , varchar2（现在使用） , string
　　Boolean，date

```
Declare
    V_firstName varchar(20);
Declare
    V_firstName  students.first_name%type;   －－和 students.first_name 的大小一致
*显示输出－－－set serveroutput on
```

3.2. 代码注释

多行　/*　　*/
单行　--

3.3. Type 关键字：

Example 1：

```
Declare
    V_id students.id%type:=1;
    V_name students.name%type:=’mudi’;
    V_age students.age%type:=24;
Begin
    Dbms_ouput.put_line(v_id);
    Dbms_output.put_line(v_name);
    Dbms_output.put_line(v_age);
End;
```

```
DECLARE
    v_LastName S_EMP.LAST_NAME%TYPE;
    v_FirstName S_EMP.FIRST_NAME%TYPE;
    v_ID S_EMP.ID%TYPE;

Example 2：
BEGIN
    select ID,LAST_NAME,FIRST_NAME INTO v_ID,v_LastName,v_FirstName
FROM S_EMP WHERE rownum = 1;
    DBMS_OUTPUT.PUT_LINE(v_ID);
    DBMS_OUTPUT.PUT_LINE(v_LastName);
    DBMS_OUTPUT.PUT_LINE(v_FirstName);
END;

多个 TYPE 类型组合使用
DECLARE
    v_LastName S_EMP.LAST_NAME%TYPE;
    v_FirstName S_EMP.FIRST_NAME%TYPE;
    v_ID S_EMP.ID%TYPE;
BEGIN
    select ID,LAST_NAME,FIRST_NAME INTO v_ID,v_LastName,v_FirstName
FROM S_EMP WHERE rownum = 1;
    DBMS_OUTPUT.PUT_LINE(v_ID);
    DBMS_OUTPUT.PUT_LINE(v_LastName);
    DBMS_OUTPUT.PUT_LINE(v_FirstName);
END;
```

3.4. record 组合类型（相当于 C 中的结构体）

```
Example 1：
Type class is record(
    Id students.id%type;
    Name students.name%type;
    Age studnets.age%type;
);
Sd0510 class;
Begin
    Select * into sd0510 from classes where class_name=’sd0510’;
    Dbms_output.put_line(sd0510.id);
End;
Example 2:
Declare
    Type emp_type is record(
        Emp_id s_emp.id%type,
```

```
        Emp_first_name s_emp.first_name%type,
        Emp_last_name s_emp.last_name%type
)
    Emp emp_type;
Begin
        Select * into emp from s_emp where id=1;
        Dbms_output.put_line(emp.emp_id);
End;
```

3.5. Rowtype 类型

Example 1:

```
declare
Emp s_emp%rowtype;
Begin
    Select * into emp from s_emp where id=1;
End;
```

Table 类型

Example 1:

```
Declare
    Type t_emp is table of s_emp%rowtype index by binary_integer;
    Emp t_emp;
Begin
    Select * into emp(100) from s_emp where id = 100;
    dbms_output.put_line('id===>'||emp(100).id);
    dbms_output.put_line('first_name===>'||emp(100).first_name);
    dbms_output.put_line('last_name===>'||emp(100).last_name);
end;
```

3.6. TABLE 类型

```
TYPE Info IS TABLE OF S_EMP%ROWTYPE INDEX BY BINARY_INTEGER;
    v_info Info;
```

3.7. 变量的作用域与可见性

如同 JAVA

3.8. 控制语句

IF 判断语句
JAVA
If(logic expression){}
Else if(logic expression){}
Else{}
PL/SQL
If logic expression then ….
Elsif logic expression then
Endif
Example 1:

```
declare
        i number;
begin
        select count(*) into i from s_emp;
        if i=0 then
            dbms_output.put_line('no elements in s_emp');
        elsif i<10 then
            dbms_output.put_line('number < 10');
        else
            dbms_output.put_line(i);
        end if;
end;
```

Loop 表达式

```
Loop
    If logic_expression then exit;
End loop;
```

While 表达式

```
While logic
    …
End loop;
```

For 表达式(与 JAVA 的特点是)

```
    For j in low..high loop
        ….
    End loop;
```

GOTO 语句
特点：只能从内部跳到外部
NULL 语句
解决一些空语句的问题
Example 1:

```
If … then
…
```

```
Else
Null;
End if;
    Example 2:
    If … then
    …
    Goto goto_end;
    End;
<<goto_end>>
    End;
```

3.9. SQL in PL/SQL

Data Manipulation language
Data Definition language
Transation Control
System comment
ESQL
在 PL/SQL 只能使用 DML，Transation Control

3.10. Cursor 的使用

```
声明   Cursor s is select * from s_emp;
打开   Open s;
提取数据  FETCH S INTO emp;
          EXIT WHEN S%NOTFOUND;
Cursor 的关键字
    Found
    Notfound
    Isopen
    Rowcount

在 loop 中使用 cursor
declare
    cursor emp_cur is select * from s_emp;
    emp s_emp%rowtype;
begin
    loop
        fetch emp_cur into emp;
        if emp_cur%notfound then
            exit;
        end if;
```

```
    end loop;
    close emp_cur;
end;
/
```

在 while 中使用 cursor

```
Open emp_cur;
While emp_cur%found loop
    Fetch emp_cur into emp;
    Dbms_output.put_line(emp.id);
End loop;
Close emp_cur;
```

在 for 中使用 cursor

```
open emp_cur;
for emp in emp_cur loop
    dbms_output.put_line(emp.id);
end loop;
close emp_cur;
```

3.11. 异常的使用

异常分为：

- 系统异常
  - Dup_val_on_index
  - Timeout_on_resource
  - ……
- 自定义异常

Example 1:

```
DECLARE
    myException      EXCEPTION;
    F_NAME    S_EMP.FIRST_NAME%TYPE;
    F1_NAME   S_EMP.FIRST_NAME%TYPE;
    CURSOR S IS SELECT FIRST_NAME FROM S_EMP;
BEGIN
    OPEN S;
    LOOP
        FETCH S INTO F_NAME;
        EXIT WHEN S%NOTFOUND;
        IF F_NAME IS NULL THEN
            RAISE myException;
        ELSE
            DBMS_OUTPUT.PUT_LINE(' F_Name : ' || F_NAME);
        END IF;
```

```
    END LOOP;
    CLOSE S;

        select first_name into f1_name from s_emp;
    DBMS_OUTPUT.PUT_LINE(' F1_Name : ' || F1_NAME);

EXCEPTION
    WHEN myException THEN
        DBMS_OUTPUT.PUT_LINE(' Captured !');
        CLOSE S;
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(' Captured NO DATA FOUND!');

END;
```

注意：for emp in emp_cur loop …..end loop;自动关闭 cursor

3.12. 使用 PROCEDURE

行参模式分为 in , out , int out 默认为 in

```
CREATE OR REPLACE PROCEDURE ModeTest(
    p_In IN NUMBER,
    p_Out OUT NUMBER,
    p_inout IN OUT NUMBER)
IS
    v_Local Number;
BEGIN
    v_Local := p_In;
    p_Out := 7;
    p_InOut := 7;
END ModeTest;
/
```

注意： 如果在 SQLPLUS 里看详细的信息，show errors
In 模式形参可以有默认值 DEFAULT 或者 :=

两种命名方法
名字标识法：

```
Test_procedure(outPara=>a,intPara=>b,inoutPara=>c);
```

outPara , intPara , inoutPara 为行参名字
位置标识法：

```
Test_procedure(a,b,c);
```

注意：要想直接在 sqlplus 运行 procedure

Variable a number;
Variable b number;
Exec test_procedure(inpara=>100,outpara=>2);
输出结果
Print
以上命令只能用在 SQLPLUS 的 CONSOLE
匿名块每次都编译，有名块第一次编译，以后不用。

删除 PROCEDURE : drop procedure_name

3.13. 使用 FUNCTION

3.13.1. 程序分两种：

1．通过 create or replace 编译的存在数据库里
2．没有 create or replace 不存在数据库里
且只能被调用的父过程调用。

```
CREATE OR REPLACE FUNCTION lessthen_test(
    src IN NUMBER,
    ref IN NUMBER := 400)
RETURN BOOLEAN AS
BEGIN
    IF src > ref THEN RETURN FALSE;
    ELSE RETURN TRUE;
    END IF;
END;
```

3.14. PACKAGE

3.14.1. 包的声明

```
CREATE OR REPLACE PACKAGE MyPackages IS
    PROCEDURE AddStudent(
        p_ID IN S_EMP.ID%TYPE,
        p_FNAME IN S_EMP.FIRST_NAME%TYPE := '',
        p_LNAME IN S_EMP.LAST_NAME%TYPE DEFAULT ''
    );

    FUNCTION DeleteStudent(
        p_ID IN S_EMP.ID%TYPE
```

```
    ) RETURN BOOLEAN;

END MyPackages;
```

3.14.2. 包的定义

```
CREATE OR REPLACE PACKAGE BODY MyPackages AS

    PROCEDURE AddStudent(
        p_ID IN S_EMP.ID%TYPE,
        p_FNAME IN S_EMP.FIRST_NAME%TYPE := '',
        p_LNAME IN S_EMP.LAST_NAME%TYPE DEFAULT ''
    ) IS
        Duplicate EXCEPTION;
        i NUMBER := 0;
    BEGIN
        SELECT COUNT(*) INTO i FROM S_EMP WHERE ID = p_ID AND
FIRST_NAME = p_FNAME AND LAST_NAME = p_LNAME;
        IF i = 0 THEN
            INSERT INTO S_EMP (ID,FIRST_NAME,LAST_NAME)
VALUES(p_ID,p_FNAME,p_LNAME);
            COMMIT;
        ELSE
            RAISE Duplicate;
        END IF;
    EXCEPTION
        WHEN Duplicate THEN
            DBMS_OUTPUT.PUT_LINE('Duplicate Record');
    END AddStudent;

    FUNCTION DeleteStudent(
        p_ID IN S_EMP.ID%TYPE
    ) RETURN BOOLEAN IS
        i NUMBER := 0;
    BEGIN
        SELECT COUNT(*) INTO i FROM S_EMP WHERE ID = p_ID;
        IF i > 0 THEN
            DELETE FROM S_EMP WHERE ID = p_ID;
            COMMIT;
        ELSE
            NULL;
        END IF;
        RETURN TRUE;
    END DeleteStudent;
```

END;

3.14.3. 包的初始化代码

3.15. 触发器

CREATE OR REPLACE TRIGGER qiuTrigger after DELETE OR INSERT OR UPDATE ON S_EMP
FOR EACH ROW(行触发器，不写为语句触发器)
DECLARE
    i NUMBER := 0;
BEGIN
    SELECT COUNT(*) INTO i FROM S_EMP
    DBMS_OUTPUT.PUT_LINE(' Only ' || i || ' Student(s) left.');
END;

三种触发条件
两种触发时间
两种触发规则 （每行或每语句）

不能有事务语句
不能在正在修改的表
不可以修改任何变化表（定义触发器的表）;

3.16. Pl/sql 中执行动态 SQL DDL 语句

Execute immediate test_sql ;

使用占位符合
（：1）
Using id;

* clear screem