# Oracle第三天

# 使用DDL语句管理表

## 创建表空间

表空间？ ORACLE数据库的逻辑单元。 数据库---表空间 一个表空间可以与多个数据文件（物理结构）关联一个数据库下可以建立多个表空间，一个表空间可以建立多个用户、一个用户下可以建立多个表。

|  |
| --- |
| create tablespace itcast001  datafile 'c:\itcast001.dbf'  size 100m  autoextend on  next 10m |

itcast 为表空间名称

datafile 指定表空间对应的数据文件

size 后定义的是表空间的初始大小

autoextend on 自动增长 ，当表空间存储都占满时，自动增长

next 后指定的是一次自动增长的大小。

## 用户

1、创建用户

|  |
| --- |
| create user itcastuser  identified by itcast  default tablespace itcast001 |

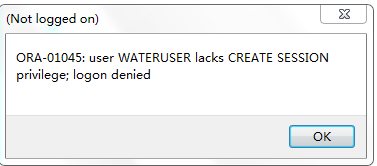
identified by 后边是用户的密码

default tablespace 后边是表空间名称

oracle数据库与其它数据库产品的区别在于，表和其它的数据库对象都是存储在用户下的。

2、用户赋权限

新创建的用户没有任何权限，登陆后会提示



Oracle中已存在三个重要的角色：connect角色，resource角色，dba角色。

CONNECT角色： --是授予最终用户的典型权利，最基本的

ALTER SESSION --修改会话

CREATE CLUSTER --建立聚簇

CREATE DATABASE LINK --建立数据库链接

CREATE SEQUENCE --建立序列

CREATE SESSION --建立会话

CREATE SYNONYM --建立同义词

CREATE VIEW --建立视图

RESOURCE角色： --是授予开发人员的

CREATE CLUSTER --建立聚簇

CREATE PROCEDURE --建立过程

CREATE SEQUENCE --建立序列

CREATE TABLE --建表

CREATE TRIGGER --建立触发器

CREATE TYPE --建立类型

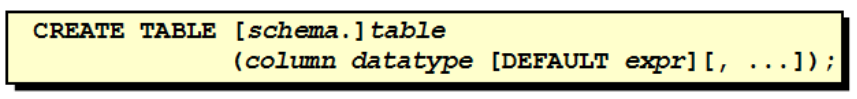
DBA角色：拥有全部特权，是系统最高权限，只有DBA才可以创建数据库结构，并且系统权限也需要DBA授出，且DBA用户可以操作全体用户的任意基表，包括删除

|  |
| --- |
| grant dba to itcastuser |

进入system用户下给用户赋予dba权限，否则无法正常登陆

## 创建表

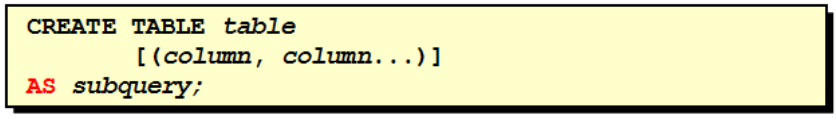
语法：



数据的类型：



使用子查询创建表的语法：



创建表范例：创建person表

create table person(

pid number(10),

name varchar2(10),

gender number(1) default 1,

birthday date

);

insert into person(pid, name, gender, birthday)

values(1, '张三', 1, to\_date('1999-12-22', 'yyyy-MM-dd'));

## 修改表

在sql中使用alter可以修改表

添加语法：ALTER TABLE 表名称 ADD(列名1 类型 [DEFAULT 默认值]，列名1 类型 [DEFAULT 默认值]...)

修改语法：ALTER TABLE 表名称 MODIFY(列名1 类型 [DEFAULT 默认值]，列名1 类型 [DEFAULT 默认值]...)

修改列名: ALTER TABLE 表名称 RENAME COLUMN 列名1 TO 列名2

范例：在person表中增加列address

alter table person add(address varchar2(10));

范例：把person表的address列的长度修改成20长度

alter table person modify(address varchar2(20));

## 删除表

语法：DROP TABLE 表名

## 约束

在数据库开发中，约束是必不可少，使用约束可以更好的保证数据的完整性。在Oracle数据库中，约束的类型包括：

主键约束（Primary Key）

非空约束（Not Null）

唯一约束（Unique）

外键约束（Foreign Key）

检查性约束（Check）

### 1.主键约束

主键约束都是在id上使用，而且本身已经默认了内容不能为空，可以在建表的时候指定。

创建一张表，把pid作为主键

create table person(

pid number(10) primary key,

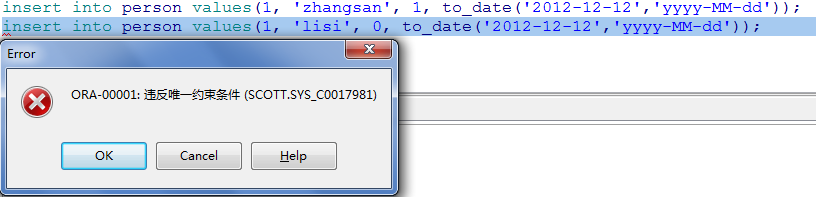
name varchar2(10),

gender number(1) default 1,

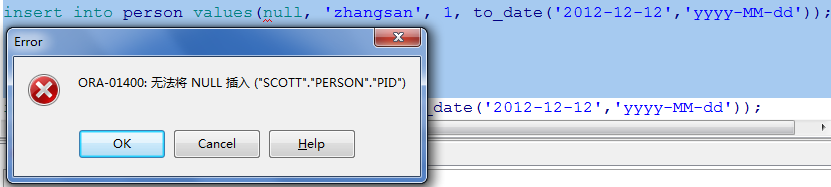
birthday date

);

主键不可重复， SCOTT.SYS\_C0017981是系统自动分配的约束的名字



主键不可为空



我们可以自己来指定主键约束的名字

create table person(

pid number(10),

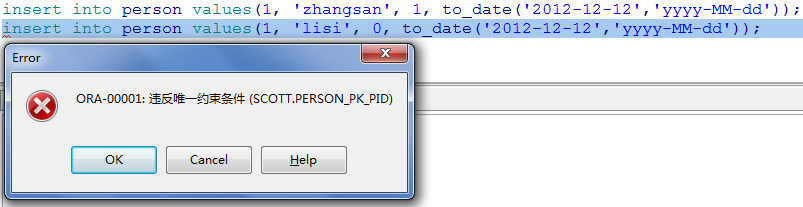
name varchar2(10),

gender number(1) default 1,

birthday date,

constraint person\_pk\_pid primary key(pid)

);



### 2.非空约束

使用非空约束，可以使指定的字段不可以为空。

范例：建立一张pid和name不可以为空的表

create table person(

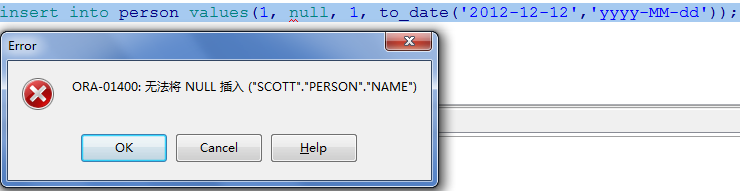
pid number(10) not null,

name varchar2(10) not null,

gender number(1) ,

birthday date,

);



### 3.唯一约束（unique）

表中的一个字段的内容是唯一的

范例：建表一个name是唯一的表

create table person(

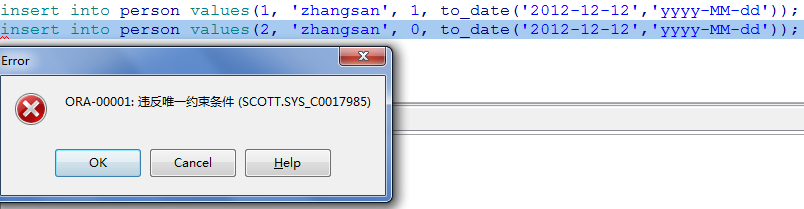
pid number(10) ,

name varchar2(10) unique,

gender number(1) ,

birthday date

);



唯一约束的名字也可以自定义

create table person(

pid number(10) ,

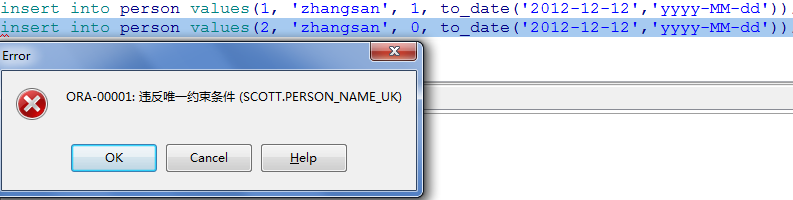
name varchar2(10),

gender number(1) ,

birthday date,

constraint person\_name\_uk unique(name)

);



### 4.检查约束

使用检查约束可以来约束字段值的合法范围。

范例：创建一张表性别只能是1或2

create table person(

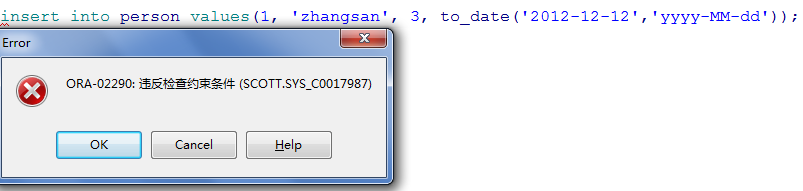
pid number(10) ,

name varchar2(10),

gender number(1) check(gender in (1, 2)),

birthday date

);



检查约束也可以自定义

create table person(

pid number(10) ,

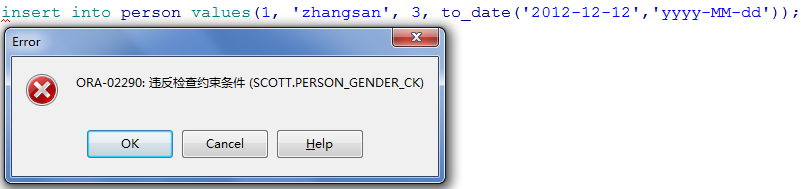
name varchar2(10),

gender number(1),

birthday date,

constraint person\_gender\_ck check(gender in (1,2))

);



### 5.外键约束

之前所讲的都是单表的约束，外键是两张表的约束，可以保证关联数据的完整性。

范例：创建两张表，一张订单表，一张是订单明细表，订单和明细是一对多的关系

create table orders(

order\_id number(10) ,

total\_price number(10,2),

order\_time date,

constraint orders\_order\_id\_pk primary key(order\_id)

);

create table order\_detail(

detail\_id number(10) ,

order\_id number(10),

item\_name varchar2(10),

quantity number(10),

constraint order\_detail\_detail\_id\_pk primary key(detail\_id)

);

insert into orders values(1, 200, to\_date('2015-12-12','yyyy-MM-dd'));

insert into order\_detail values(1, 2, 'java',1);

我们在两张表中插入如上两条数据，我们发现在order\_detail表中插入的order\_id在order表中并不存在，这样在数据库中就产生了脏数据。此时需要外键来约束它。

我们再次建表

create table orders(

order\_id number(10) ,

total\_price number(10,2),

order\_time date,

constraint orders\_order\_id\_pk primary key(order\_id)

);

create table order\_detail(

detail\_id number(10) ,

order\_id number(10),

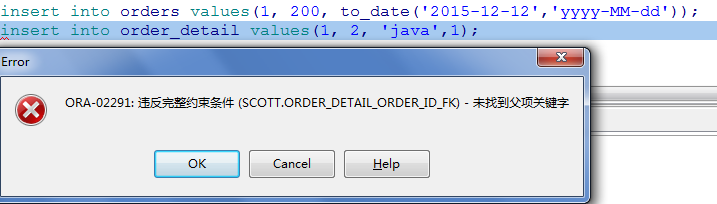
item\_name varchar2(10),

quantity number(10),

constraint order\_detail\_detail\_id\_pk primary key(detail\_id),

constraint order\_detail\_order\_id\_fk foreign key(order\_id) references orders(order\_id)

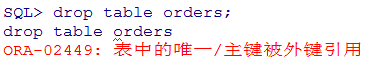
);



外键关联一定注意：

外键一定是主表的主键

删表时一定先删子表再删主表，如果直接删主表会出现由于约束存在无法删除的问题



但是可以强制删除drop table orders cascade constraint;(不建议)

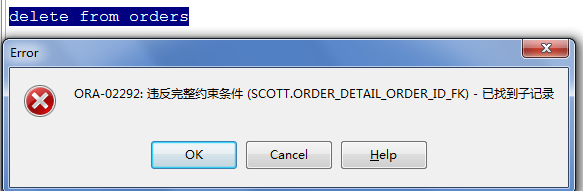
删除主表的数据可以先删除子表的关联数据，再删主表，也可以使用级联删除。

级联删除在外键约束上要加上on delete cascade 如

constraint order\_detail\_order\_id\_fk foreign key(order\_id)

references orders(order\_id) on delete cascade

这样删除主表数据的时候会把字表的关联数据一同删除



# 使用DML语句处理数据

## 插入数据

语法：INSERT INTO表名[(列名1，列名2，...)]VALUES(值1，值2，...)

标准写法



简单写法（不建议）

INSERT INTO 表名VALUES(值1，值2，...)

insert into **person**

values(1,'张三',1,'9-5月-1981','北京北七家');

注意：使用简单的写法必须按照表中的字段的顺序来插入值，而且如果有为空的字段使用null

insert into person values(2,'李四',1,null,'北京育新');

## 更新数据

全部修改：UPDATE 表名 SET 列名1=值1，列名2=值2，....

局部修改：UPDATE 表名 SET 列名1=值1，列名2=值2，....WHERE 修改条件；

在update中使用子查询：

例如：给 NEW YORK地区的所有员工涨100员工资

|  |
| --- |
| update emp set sal=sal+100 where deptno  in (select deptno from dept where loc='NEW YORK') |

## 删除数据

语法 : DELETE FROM 表名 WHERE 删除条件;

在删除语句中如果不指定删除条件的话就会删除所有的数据

Truncate table实现数据删除

比较truncat与delete实现数据删除？

delete删除的数据可以rollback,也可以闪回

delete删除可能产生碎片，并且不释放空间

truncate是先摧毁表结构，再重构表结构

注意：插入、更新和删除会引起数据的变化。我们就必须考虑数据的完整性。

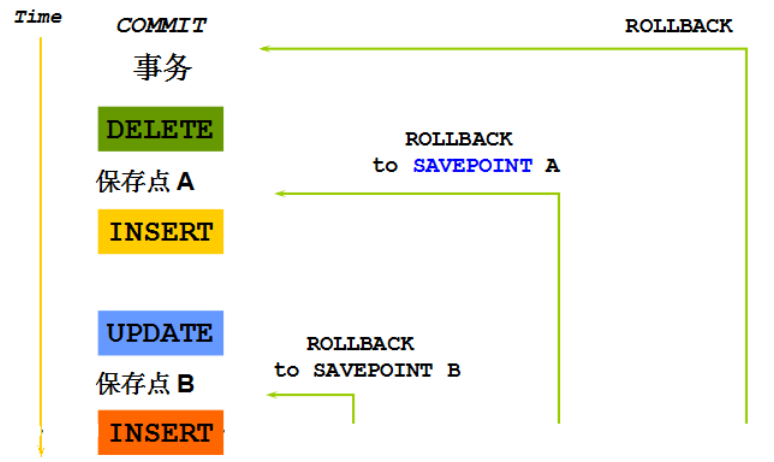
## Oracle中的事务

这是因为oracle的事务对数据库的变更的处理，我们必须做提交事务才能让数据真正的插入到数据库中，在同样在执行完数据库变更的操作后还可以把事务进行回滚，这样就不会插入到数据库。如果事务提交后则不可以再回滚。

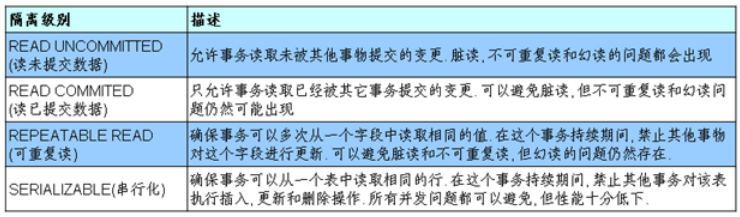
提交：commit

回滚：rollback

Oracle中事务的保存点：



事务的隔离级别和隔离性：



Oracle 支持的 3种事务隔离级别：READ COMMITED, SERIALIZABLE，READ ONLY. Oracle 默认的事务隔离级别为: READ COMMITED

# 管理其他数据库对象

## 视图

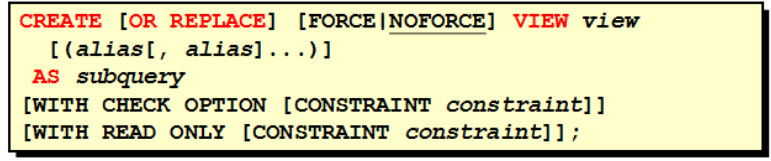
什么是视图：

视图就是封装了一条复杂查询的语句。

视图是一个虚表。

最大的优点就是简化复杂的查询。

创建视图的语法

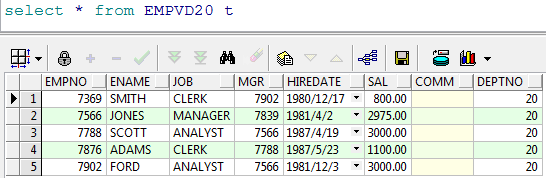


创建视图示例

范例：建立一个视图，此视图包括了20部门的全部员工信息

create view empvd20 as select \* from emp t where t.deptno = 20

视图创建完毕就可以使用视图来查询，查询出来的都是20部门的员工



语法2：CREATE OR REPLACE VIEW 视图名称 AS 子查询

如果视图已经存在我们可以使用语法2来创建视图，这样已有的视图会被覆盖。

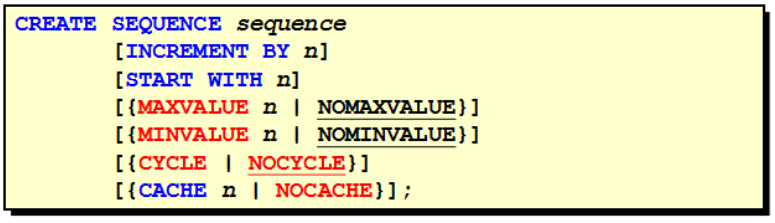
create or replace view empvd20 as select \* from emp t where t.deptno = 20

不建议通过视图对表中的数据进行修改，因为会受到很多的限制。

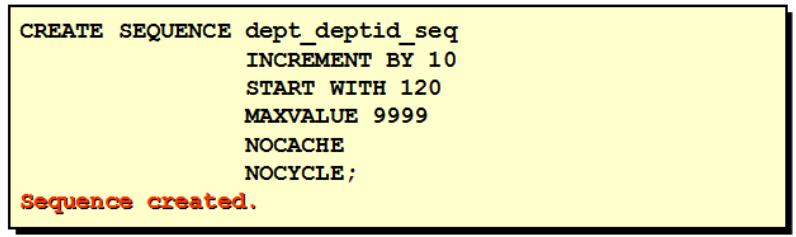
## 序列

在很多数据库中都存在一个自动增长的列,如果现在要想在oracle 中完成自动增长的功能, 则只能依靠序列完成,所有的自动增长操作,需要用户手工完成处理。并且Oracle将序列值装入内存可以提高访问效率。

语法：



范例:

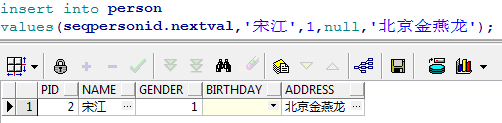


序列创建完成之后,所有的自动增长应该由用户自己处理,所以在序列中提供了以下的两种操作：

nextval :取得序列的下一个内容

currval :取得序列的当前内容

在插入数据时需要自增的主键中可以这样使用



序列可能产生裂缝的原因：

回滚

系统异常

多个表共用一个序列

## 索引

索引是用于加速数据存取的数据对象。合理的使用索引可以大大降低i/o 次数,从而提高数据访问性能。

#### 1． 单列索引

单列索引是基于单个列所建立的索引，比如:

CREATE index 索引名 on 表名(列名)

#### 2． 复合索引

复合索引是基于两个列或多个列的索引。在同一张表上可以有多个索引，但是

要求列的组合必须不同,比如：

Create index emp\_idx1 on emp(ename,job);

Create index emp\_idx1 on emp(job,ename);

范例：给person表的name建立索引

create index pname\_index on person(name);

范例：给person表创建一个name和gender的索引

create index pname\_gender\_index on person(name, gender);

#### 3 . 索引测试

创建一个新表，向表中插入100W或500W条数据，记录查询一条数据所用时间，之后创建索引，后再查询一条数据，比较两条数据查询的时间。

--创建表

create table t\_test(

tid number,

tname varchar2(30)

)

--创建序列

create sequence seq\_test;

--插入数据

begin

for i in 1..5000000

loop

insert into t\_test values(seq\_test.nextval,'测试数据'||i);

end loop;

commit;

end;

随便查询一条数据

select \* from t\_test where tname ='测试数据4789889'; 用时： 1.797s

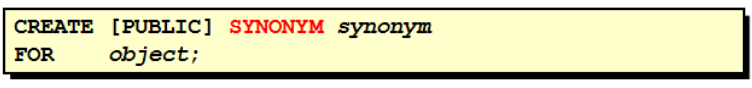
--创建索引

create index index\_test on t\_test(tname)

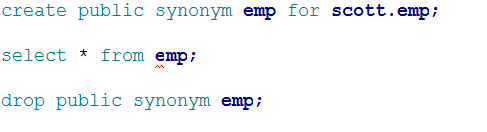
select \* from t\_test where tname ='测试数据4889889';用时: 0.047

哈哈 ~ 效果还是很明显的嘛~

## 同义词



例如：



#### 使用同义词的作用？

可以很方便的访问其它用户的数据库对象

缩短了对象名字的长度

# 数据的导入导出

当我们使用一个数据库时，总希望数据库的内容是可靠的、正确的，但由于计算机系统的故障（硬件故障、软件故障、网络故障、进程故障和系统故障）影响数据库系统的操作，影响数据库中数据的正确性，甚至破坏数据库，使数据库中全部或部分数据丢失。因此当发生上述故障后，希望能重构这个完整的数据库该处理称为数据库恢复，而要进行数据库的恢复必须要有数据库的备份工作。

## 使用cmd命令整库导出与导入

在安装了oracle的电脑上执行

整库导出命令

|  |
| --- |
| exp system/itcast full=y |

添加参数full=y 表示整库导出

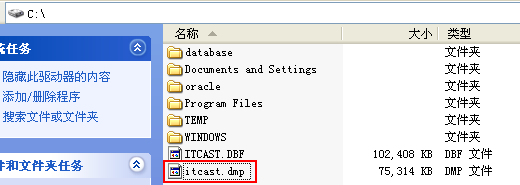


执行命令后会在当前目录下生成一个叫EXPDAT.DMP，此文件为备份文件。



如果想指定备份文件的名称，则添加file参数即可，命令如下

|  |
| --- |
| exp system/itcast file=C:\ itcast.dmp full=y |



整库导入命令

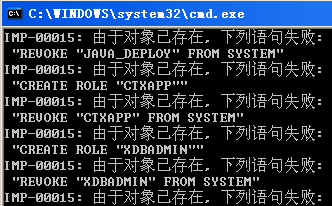
|  |
| --- |
| imp system/itcast full=y |

此命令如果不指定file参数，则默认用备份文件EXPDAT.DMP 进行导入

如果指定file参数，则按照file指定的备份文件进行恢复

|  |
| --- |
| imp system/itcast full=y file= C:\ itcast.dmp |

执行导入命令前需确保oracle数据库中无即将导入的对象，否则将报以下提示：



## 使用cmd命令按用户导出与导入

按用户导出

|  |
| --- |
| exp system/itcast owner=itcast001 file= c:\itcast.dmp |

按用户导入

|  |
| --- |
| imp system/itcast file= c:\itcast.dmp fromuser= itcast001 touser=itcast002 |

## 使用cmd命令按表导出与导入

按表导出

|  |
| --- |
| exp itcast001 /itcast file= itcast001.dmp tables=t\_person,t\_student |

用tables参数指定需要导出的表，如果有多个表用逗号分割即可

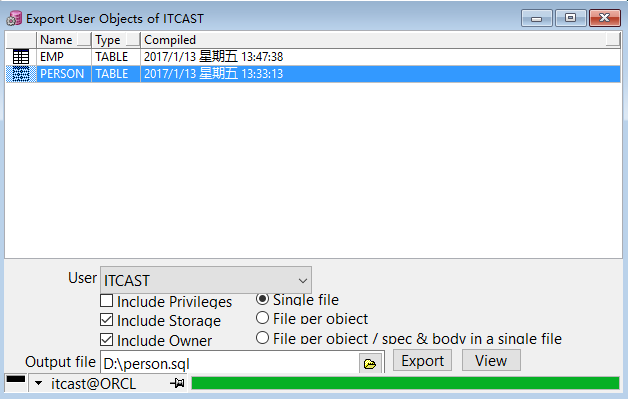
按表导入

|  |
| --- |
| imp itcast001/itcast file= itcast001.dmp tables= t\_ person,t\_student |

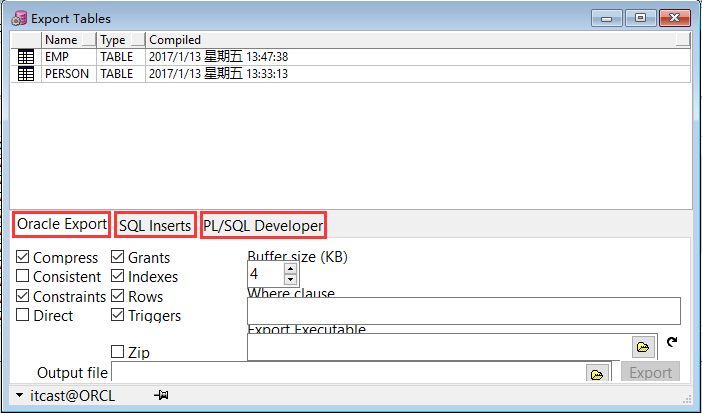
## 使用PLSQL Developer导出数据

1：Tools→Export User Objects...选项，导出.sql文件。

说明：此操作导出的是建表语句



2：Tools→Export Tables...导出表结构及数据



PL/SQL工具包含三种方式导出Oracle表结构及数据，三种方式分别为：Oracle Export 、SQL Inserts、PL/SQL Developer,下面分别简单介绍下区别：

第一种方式导出.dmp格式的文件，.dmp是二进制文件，可跨平台，还能包含权限，效率不错，用的最为广泛。

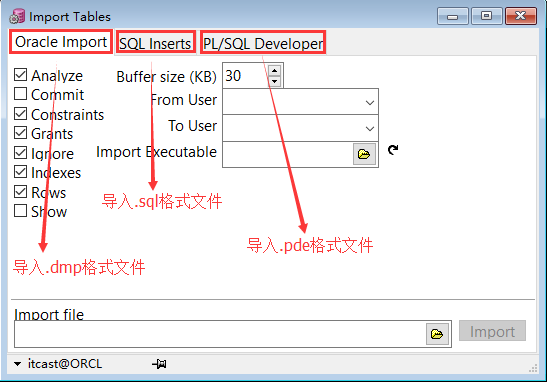
第二种方式导出.sql格式的文件，可用文本编辑器查看，通用性比较好，效率不如第一种，适合小数据量导入导出。尤其注意的是表中不能有大字段（blob,clob,long），如果有，会提示不能导出(提示如下： table contains one or more LONG columns cannot export in sql format,user Pl/sql developer format instead)。

第三种方式导出.pde格式的文件，.pde为PL/SQL Developer自有的文件格式，只能用PL/SQL Developer工具导入导出，不能用文本编辑器查看。

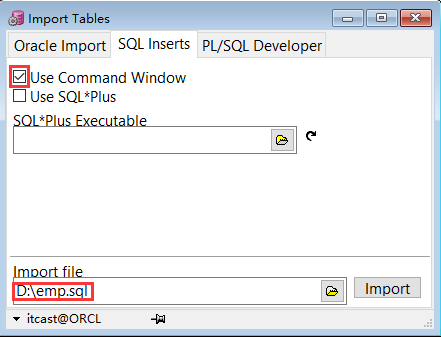
## 使用PLSQL Developer导入数据

导入数据之前最好把以前的表删掉，当然导入另外的数据库数据除外。

1：**Tools→Import Tables...**



2：根据对应格式，在不同界面选择即将导入的文件。



# PL/SQL编程语言

## 什么是PL/SQL?

PL/SQL（Procedure Language/SQL）

PLSQL是Oracle对sql语言的过程化扩展，指在SQL命令语言中增加了过程处理语句（如分支、循环等），使SQL语言具有过程处理能力。把SQL语言的数据操纵能力与过程语言的数据处理能力结合起来，使得PLSQL面向过程但比过程语言简单、高效、灵活和实用。

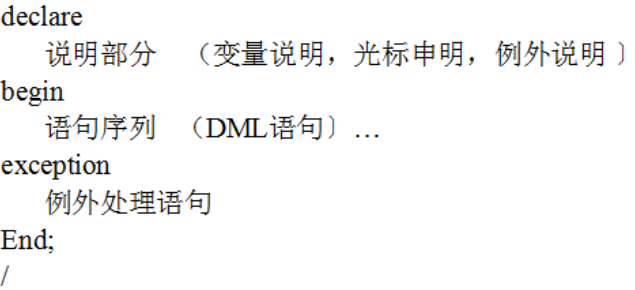
范例1：为职工涨工资，每人涨10％的工资。

update emp set sal=sal\*1.1

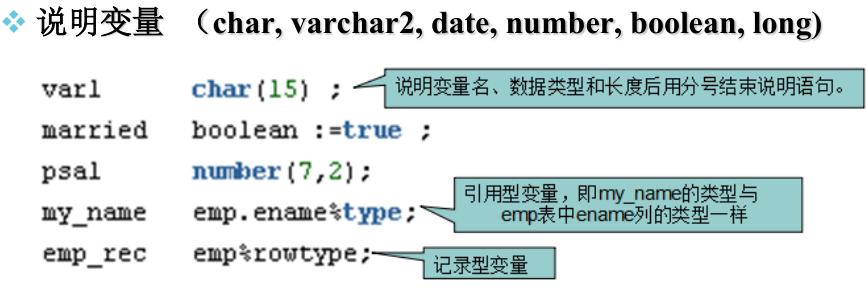
范例2：例按职工的职称长工资,总裁涨1000元,经理涨800元，其他人员涨400元。

这样的需求我们就无法使用一条SQL来实现，需要借助其他程序来帮助完成，也可以使用pl/sql。

## PL/SQL的语法



## 常量和变量的定义



引用变量

Myname emp.ename%type;

引用型变量，即my\_name的类型与emp表中ename列的类型一样

在sql中使用into来赋值

declare

emprec emp.ename%type;

begin

select t.ename into emprec from emp t where t.empno = 7369;

dbms\_output.put\_line(emprec);

end;

记录型变量

Emprec emp%rowtype

记录变量分量的引用

emp\_rec.ename:='ADAMS';

declare

p emp%rowtype;

begin

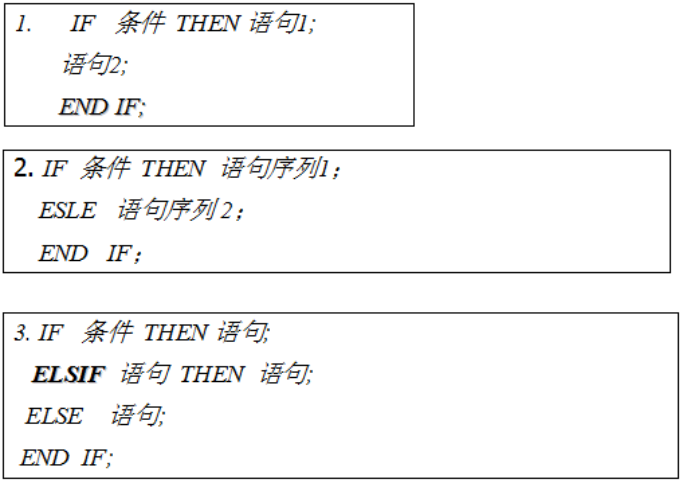
select \* into p from emp t where t.empno = 7369;

dbms\_output.put\_line(p.ename || ' ' || p.sal);

end;

## If语句

语法：



范例1：如果从控制台输入1则输出我是1

declare

pnum number := &num;

begin

if pnum = 1 then

dbms\_output.put\_line('我是1');

end if;

end;

范例2：如果从控制台输入1则输出我是1否则输出我不是1

declare

mynum number := &num;

begin

if mynum = 1 then

dbms\_output.put\_line('我是1');

else

dbms\_output.put\_line('我不是1');

end if;

end;

范例3:判断人的不同年龄段18岁以下是未成年人，18岁以上40以下是成年人，40以上是老年人

declare

mynum number := &num;

begin

if mynum < 18 then

dbms\_output.put\_line('未成年人');

elsif mynum >= 18 and mynum < 40 then

dbms\_output.put\_line('中年人');

elsif mynum >= 40 then

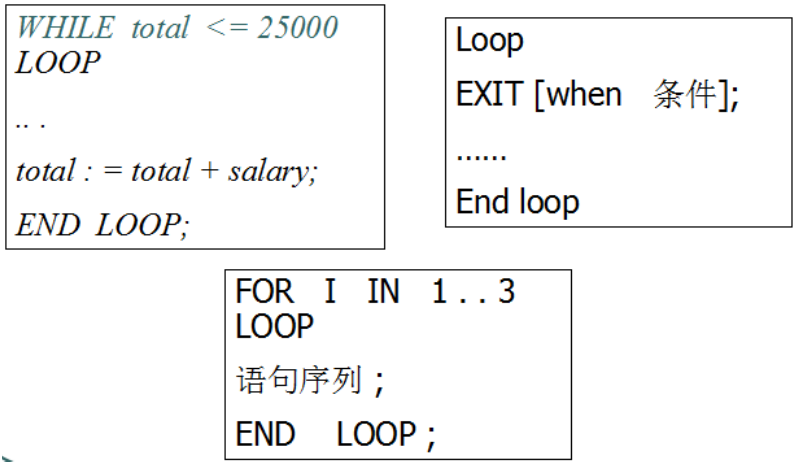
dbms\_output.put\_line('老年人');

end if;

end;

## 循环

语法：



范例:使用语法1输出1到10的数字

declare

step number := 1;

begin

while step <= 10 loop

dbms\_output.put\_line(step);

step := step + 1;

end loop;

end;

范例:使用语法2输出1到10的数字

declare

step number := 1;

begin

loop

exit when step > 10;

dbms\_output.put\_line(step);

step := step + 1;

end loop;

end;

范例:使用语法3输出1到10的数字

declare

step number := 1;

begin

for step in 1 .. 10 loop

dbms\_output.put\_line(step);

end loop;

end;