Oracle实战详解

作者：小霸王

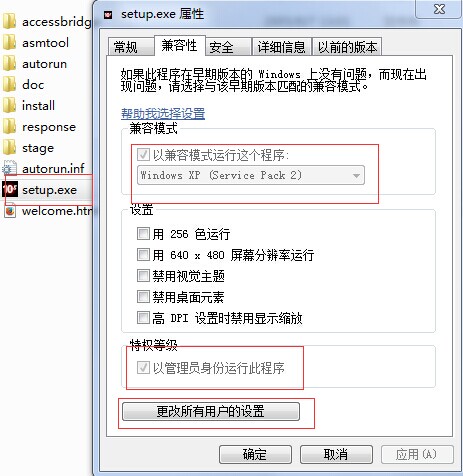
讲师：小霸王

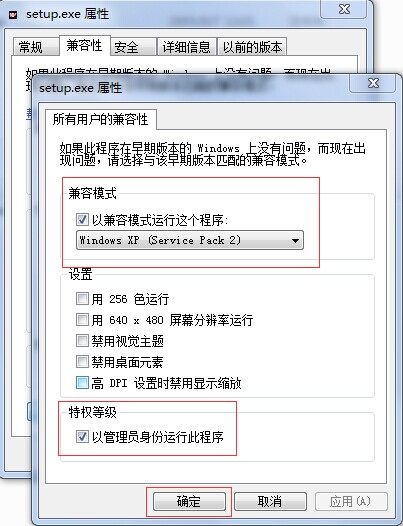
# oracle介绍

ORACLE[数据库系统](http://baike.baidu.com/view/7809.htm" \t "_blank)是美国ORACLE公司（甲骨文）提供的以[分布式数据库](http://baike.baidu.com/view/68389.htm" \t "_blank)为核心的一组软件产品，是目前最流行的客户/服务器(CLIENT/SERVER)或B/S[体系结构](http://baike.baidu.com/view/1188494.htm" \t "_blank)的数据库之一。比如SilverStream就是基于数据库的一种中间件。ORACLE数据库是目前世界上使用最为广泛的[数据库管理系统](http://baike.baidu.com/view/68446.htm" \t "_blank)，作为一个通用的[数据库系统](http://baike.baidu.com/view/7809.htm" \t "_blank)，它具有完整的[数据管理](http://baike.baidu.com/view/14717.htm" \t "_blank)功能；作为一个关系数据库，它是一个完备关系的产品；作为[分布式数据库](http://baike.baidu.com/view/68389.htm" \t "_blank)它实现了[分布式处理](http://baike.baidu.com/view/185352.htm" \t "_blank)功能。但它的所有知识，只要在一种机型上学习了ORACLE知识，便能在各种类型的机器上使用它。

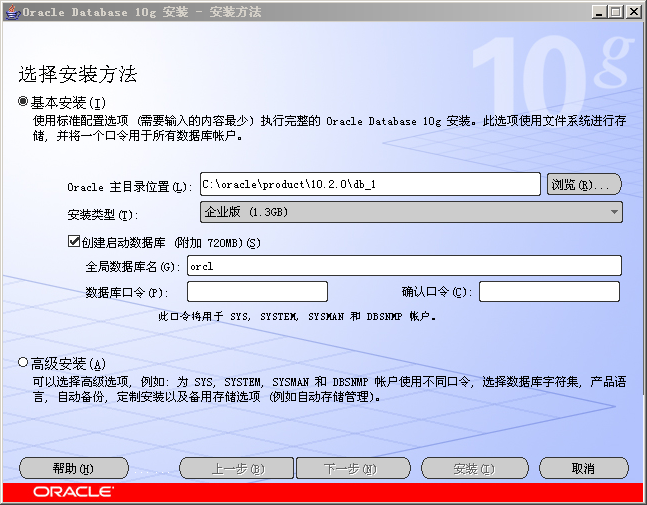
# oracle10g安装

1. 解压oracle数据库安装包，如果是win7或者win8系统右键点击setup.exe选择兼容性，以xp方式，并且以管理员方式运行，以及其他所有用户都按着此规则如图



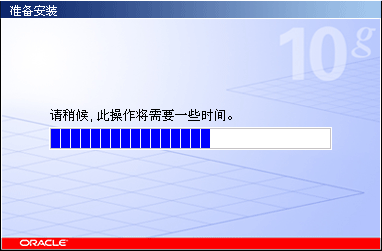


1. 并双击解压目录下的setup.exe，出现安装界面，如下：

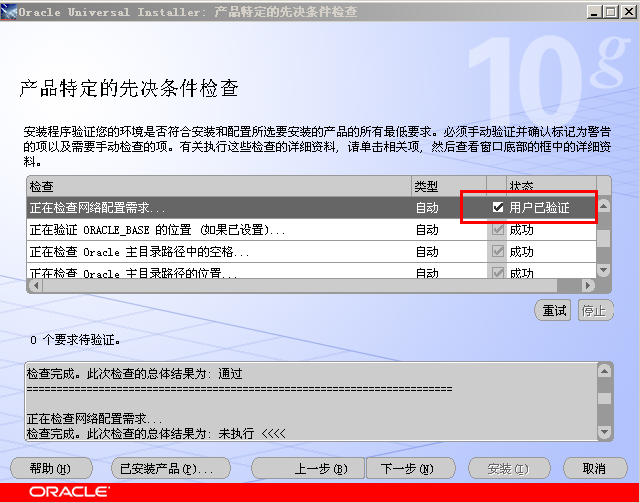


1. 输入口令和确认口令，如：password，点击下一步，出现如下进度条，

注：此口令即是管理员密码。



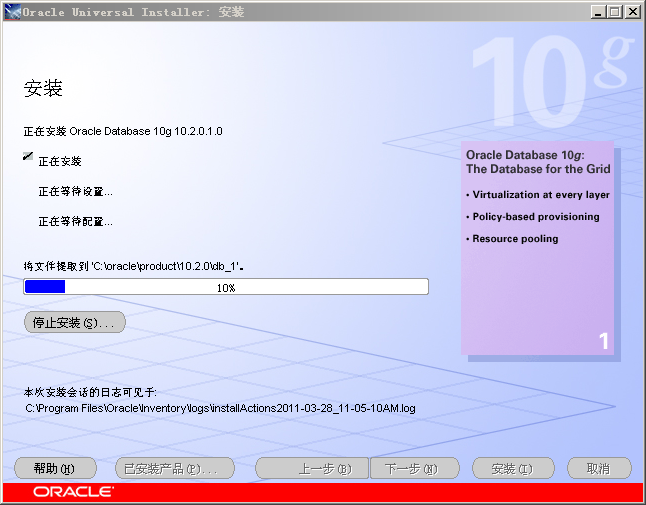
1. 检查先决条件，选中红框所示的选择框，如下图：



1. 点击“下一步”，出现“概要”界面，点击“安装”。



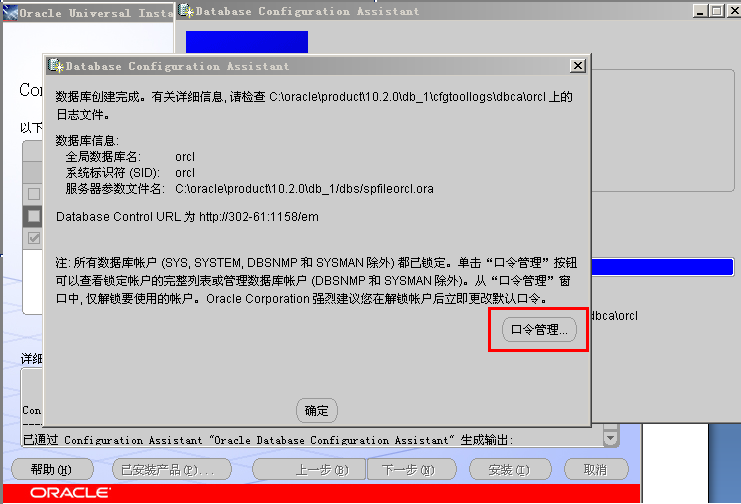
1. 出现安装进度条，等待安装完成，如下图：



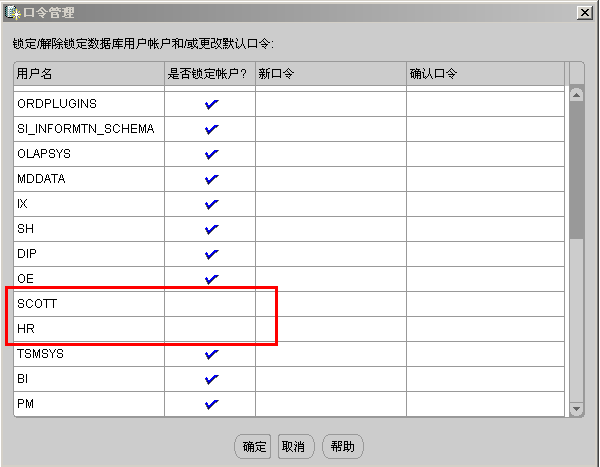
1. 安装完成后，自动运行配置向导，如下图，等待其完成：



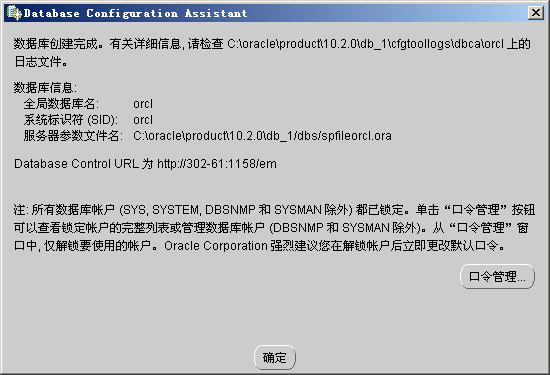
1. 完成后，出现“口令管理”界面，点击“口令管理”，如下图：



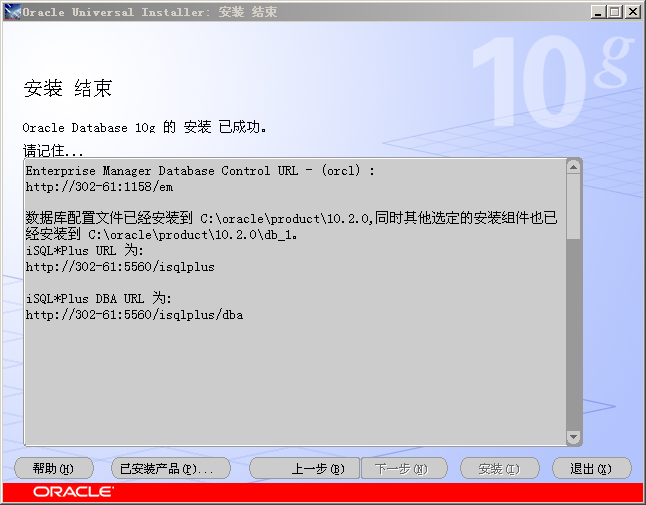
1. 将SCOTT和HR用户的沟去掉（解锁这两个账户），如下图所示，点击“确定”：



1. 回到“口令管理”界面，点击“确定”，如下图：



1. 安装结束，点击“退出”。



# PLSQL Developer安装

傻瓜式安装即可，不建议汉化，最后会提示输入序列号，打开plsql-sn.txt复制序列号，输入即可。重要提示：不要把plsql developer安装在有括号的目录下，否则会出现连不上数据库的情况，安装目录也不要出现中文。



Plsql developer是最好用也是公司里面大量使用的数据库图形化工具，功能十分强大，在开发人员中广受欢迎，基本上是使用oracle数据库的必备神器。Oracle支持命令行方式操作，在实际的开发中很少使用命令行，绝大多数使用图形化的工具Plsql developer

# PowerDesigner安装

傻瓜式安装即可。

PowerDesigner是数据库模型设计的工具，功能十分强大，不仅能设计数据库模型，还能设计各种UML建模的图形。

# oracle基本概念

* **数据库：**

Oracle数据库是数据的物理存储。这就包括（数据文件ORA或者DBF、控制文件、联机日志、参数文件）。其实Oracle数据库的概念和其它数据库不一样，这里的数据库是一个操作系统只有一个库。可以看作是Oracle就只有一个大数据库。

* **实例：**

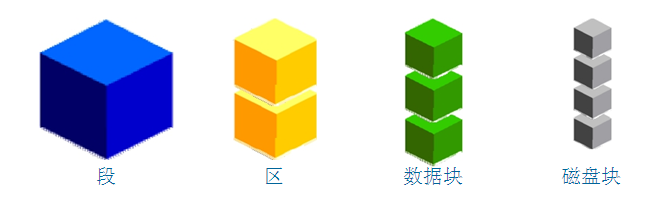
一个Oracle实例（Oracle Instance）有一系列的后台进程（Backguound Processes)和内存结构（Memory Structures)组成。一个数据库可以有n个实例。

* **用户：**

用户是在实例下建立的。不同实例可以建相同名字的用户。

* **表空间：**

表空间是Oracle对物理数据库上相关数据文件（ORA或者DBF文件）的逻辑映射。一个数据库在逻辑上被划分成一到若干个表空间，每个表空间包含了在逻辑上相关联的一组结构。每个数据库至少有一个表空间(称之为system表空间)。  
  
每个表空间由同一磁盘上的一个或多个文件组成，这些文件叫数据文件(datafile)。一个数据文件只能属于一个表空间。

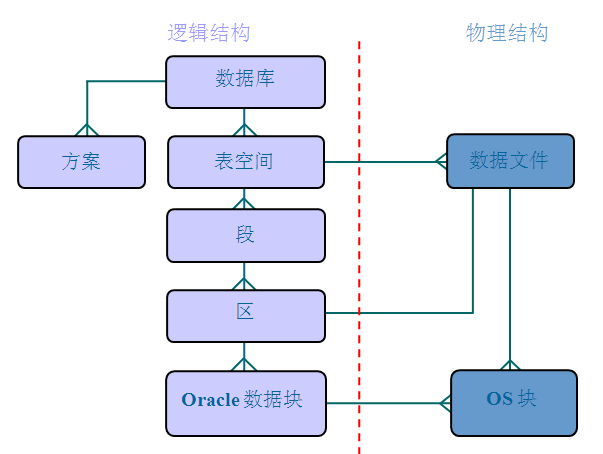


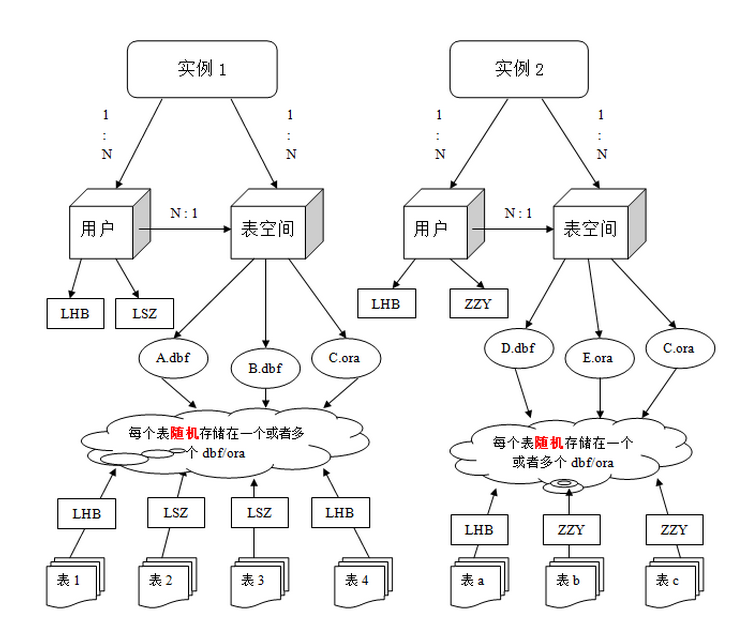
* **数据文件（dbf、ora）:**

数据文件是数据库的物理存储单位。数据库的数据是存储在表空间中的，真正是在某一个或者多个数据文件中。而一个表空间可以由一个或多个数据文件组成，一个数据文件只能属于一个表空间。一旦数据文件被加入到某个表空间后，就不能删除这个文件，如果要删除某个数据文件，只能删除其所属于的表空间才行。

注： 表的数据，是有用户放入某一个表空间的，而这个表空间会随机把这些表数据放到一个或者多个数据文件中。

由于oracle的数据库不是普通的概念，oracle是有用户和表空间对数据进行管理和存放的。但是表不是有表空间去查询的，而是由用户去查的。因为不同用户可以在同一个表空间建立同一个名字的表！这里区分就是用户了！



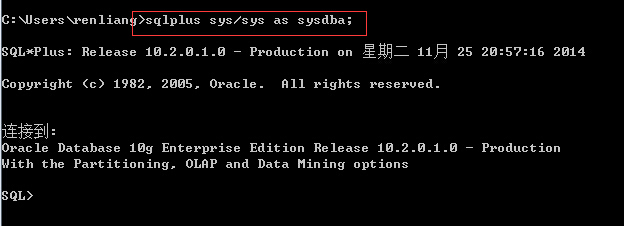


# 命令行常用操作

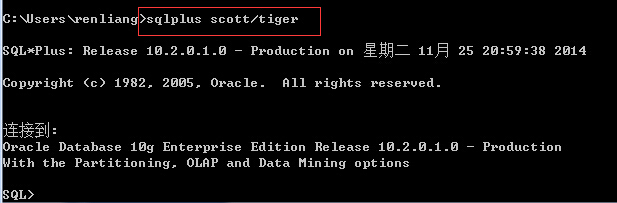
## 登录

运行cmd进入命令行

Sqlplus 用户名/密码 [as sysdba]



如果是超级管理员需要在用户名/密码后面加上 as sysdba,是以系统管理员的身份来登录的，如图。



如果是普通用户不需要as sysdba

## 2． 查看当前连接数据库的用户

使用show user查看当前的用户



## 用户的切换

在登录的状态下输入：conn 用户名/密码 [as sysdba]

如图：

* 切换为超级管理员

C:\Users\renliang\AppData\Roaming\Tencent\Users\491793587\QQ\WinTemp\RichOle\4405G$FR0O@98)R(Q7]{6PA.jpg

* 切换为普通的用户



## 查看用户下的表

为了防止命令行中数据展示表格错乱的情况可以设计行宽和列宽

Set linesize 300; 每行展示300个字符

Col 列名 for a[数字]，某一列占几个字符

在用户登录的情况下输入：select \* from tab;



查看表中的数据

Col ename for a8

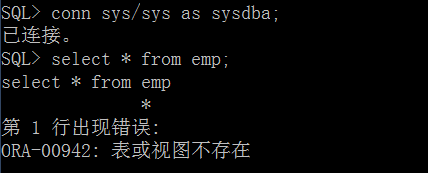
Col mgr for 9999

Col sal for 9999

Select \* from emp;

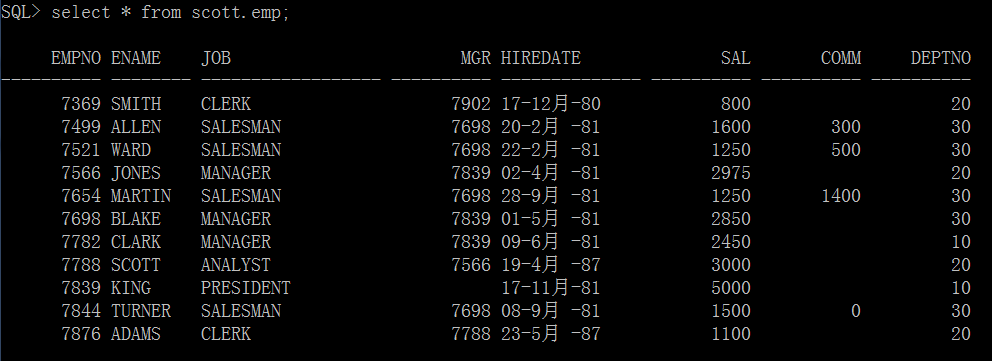


如果使用dba用户登录查询该表无法查看到，因为sys用户下没有这个emp表



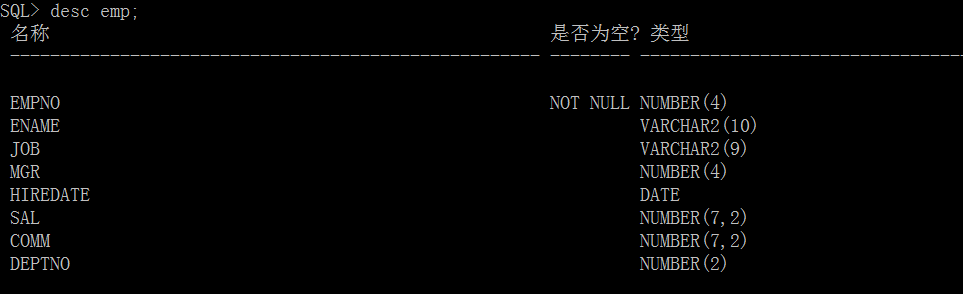
由于超级管理员的权限很大可以查看Scott用户下的表

可以使用select \* from [用户名].[表名] 来查看某用户下的表的数据



## 查看表的结构

Desc 表名



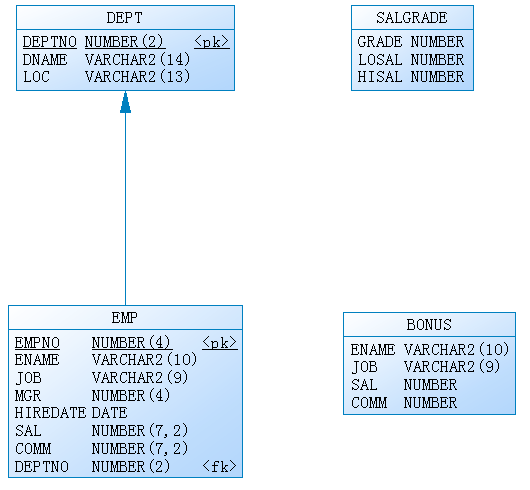
Number(4) 最大长度为4为的数值类型

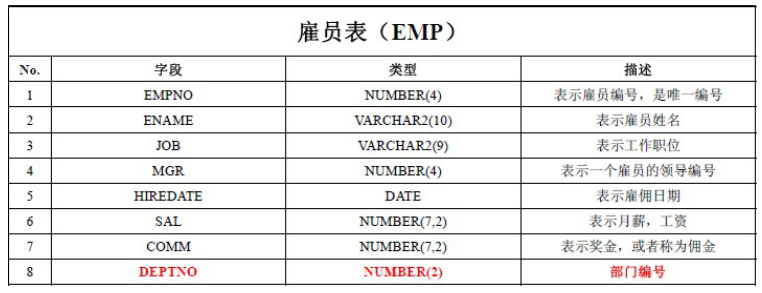
Varchar2(10) 最大长度为10的字符串，varchar2用于存储可变长度的字符串，.varchar2把所有字符都占两字节处理(一般情况下)，varchar只对汉字和全角等字符占两字节，数字，英文字符等都是一个字节，VARCHAR2把空串等同于null处理，而varchar仍按照空串处理；建议在oracle中使用varchar2

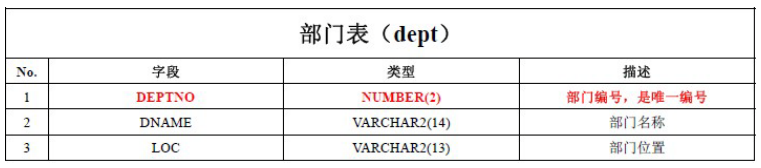
Number(7,2) 数值类型整数位占5位，小数位占2位，一共7位。

Date 时间类型

# 7.Scott用户下的表结构











# 8. sql

## 1.sql简介

结构化查询语言(Structured Query Language)简称SQL(发音：/ˈɛs kjuː ˈɛl/ "S-Q-L")，结构化查询语言是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理[关系数据库系统](http://baike.baidu.com/view/549699.htm" \t "_blank)；同时也是[数据库脚本文件](http://baike.baidu.com/view/3542225.htm" \t "_blank)的扩展名。结构化查询语言是高级的非过程化编程语言，允许用户在高层[数据结构](http://baike.baidu.com/view/9900.htm" \t "_blank)上工作。它不要求用户指定对数据的存放方法，也不需要用户了解具体的数据存放方式，所以具有完全不同底层结构的不同[数据库系统](http://baike.baidu.com/view/7809.htm" \t "_blank), 可以使用相同的结构化查询语言作为数据输入与管理的接口。结构化查询语言语句可以嵌套，这使它具有极大的灵活性和强大的功能。

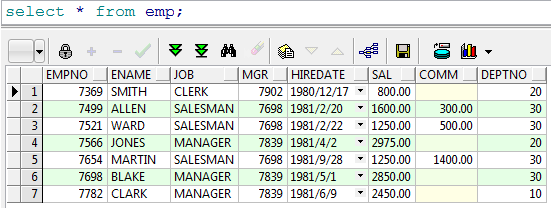
* **DML**(数据库操作语言): 其语句包括动词[INSERT](http://baike.baidu.com/view/270874.htm" \t "_blank)，[UPDATE](http://baike.baidu.com/view/1028155.htm" \t "_blank)和[DELETE](http://baike.baidu.com/view/4794.htm" \t "_blank)。它们分别用于添加，修改和删除表中的行。也称为动作查询语言。
* **DDL**(数据库定义语言): 其语句包括动词CREATE和DROP。在数据库中创建新表或删除表（CREAT TABLE 或 DROP TABLE）；为表加入索引等。DDL包括许多与人[数据库目录](http://baike.baidu.com/view/4369490.htm" \t "_blank)中获得数据有关的保留字。它也是动作查询的一部分。

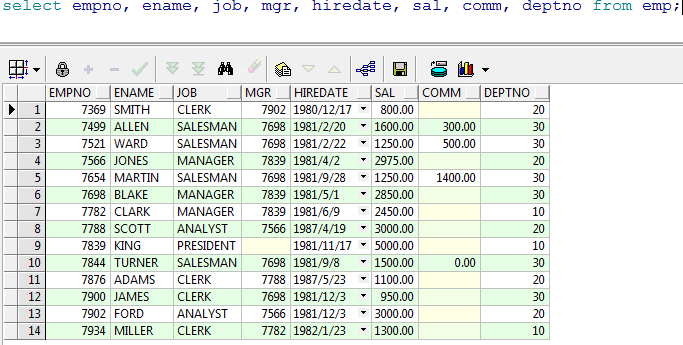
**DCL**(数据库控制语言):它的语句通过GRANT或REVOKE获得许可，确定单个用户和用户 组对[数据库对象](http://baike.baidu.com/view/50913.htm" \t "_blank)的访问。某些RDBMS可用GRANT或REVOKE控制对[表单](http://baike.baidu.com/view/296684.htm" \t "_blank)个列的访问。

## 简单查询

### 1.查询语法

Select \* |列名 from 表名

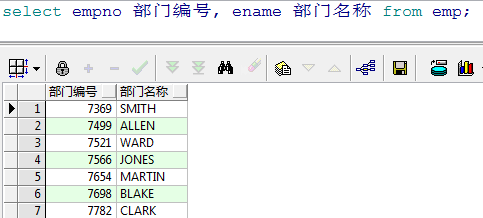




### 2.别名用法

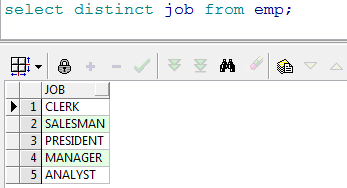
在查询的结果列中可以使用别名

Select 列名 别名，列名别名，... from emp;



### 除重复的数据

Select distinct \*|列名, ... from emp;

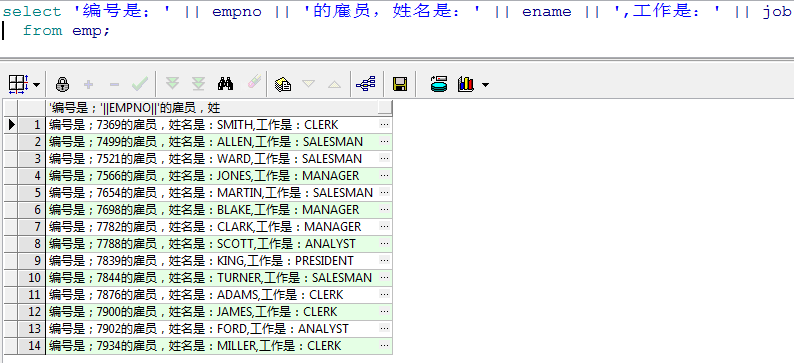


使用distinct可以消除重复的行，如果查询多列的必须保证多列都重复才能去掉重复

### 字符串连接查询

查询雇员编号，姓名，工作

编号是：7369的雇员， 姓名是：smith，工作是：clerk



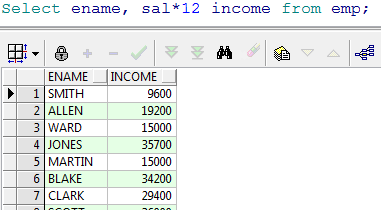
字符串的连接使用‘||’

### 查询中四则运算

查询每个雇员的年薪

Select ename, sal\*12 from emp;

Select ename, sal\*12 income from emp;



Sql中支持四则运算“+，-，\*，/”

## 3. 限定查询

在查询绝大多数都会有条件的限制

语法：select \*|列名 from 表名 where 条件

例如：查询工资大于1500的所有雇员



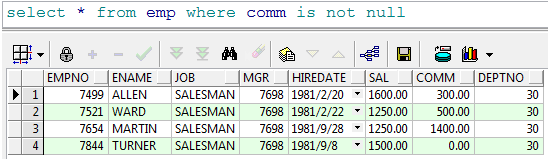
### 非空和空的限制

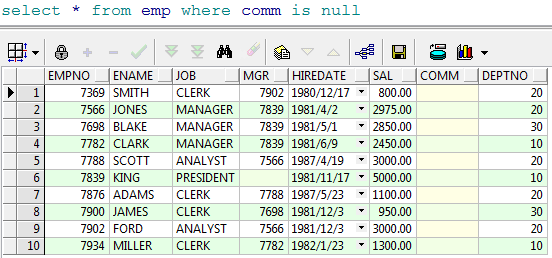
* 示例：查询每月能得到奖金的雇员

分析：只要字段中存在内容表示不为空，如果不存在内容就是null，

语法：列名 IS NOT NULL

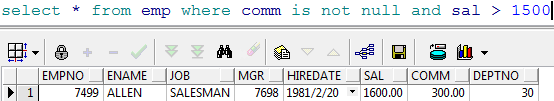
为空 列名 IS NULL





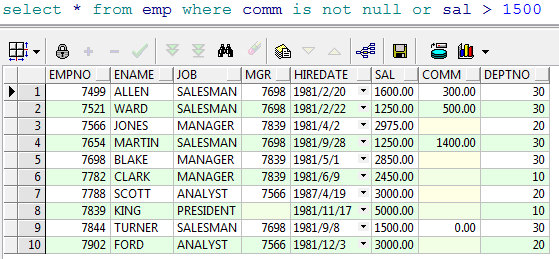
* 范例：查询工资大于1500并且有奖金领取的雇员

分析：多个查询条件同时满足之间使用‘AND’



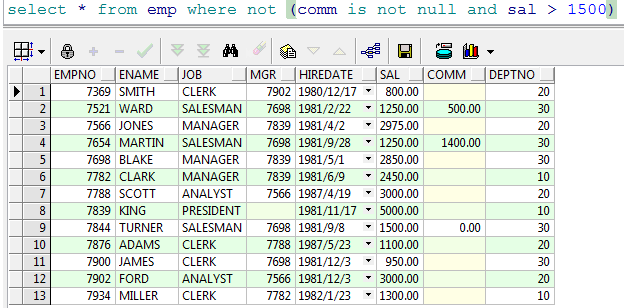
* 范例:查询工资大于1500或者有奖金的雇员

分析：多个查询条件或满足，条件之间使用“OR”



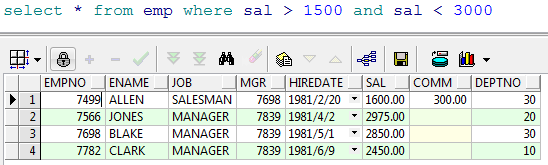
* 范例：查询工资不大于1500和没有奖金的人

语法：NOT(查询条件)

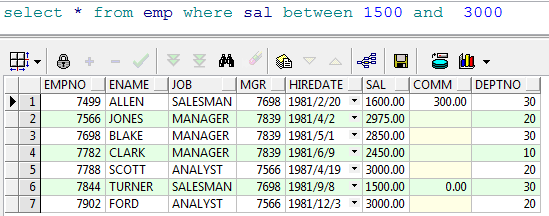


范例：基本工资大于1500但是小于3000的全部雇员

分析：sal>1500, sal<3000

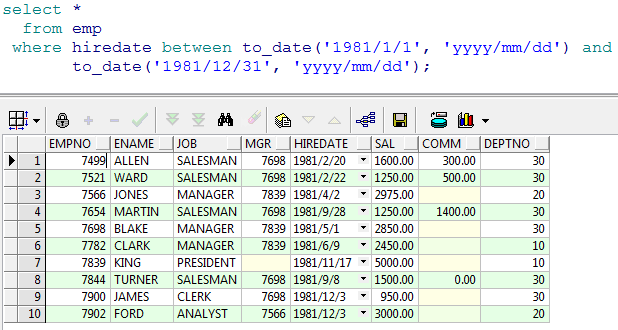


Between and等于 sal > =1500 and sal <= 3000



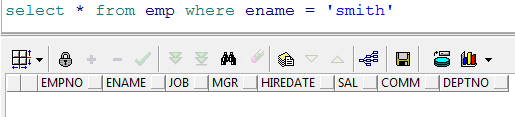
范例：查询1981-1-1到1981-12-31号入职的雇员

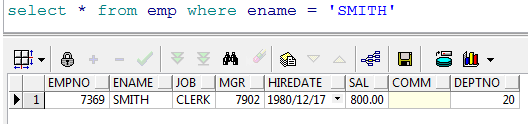
分析：between and 不仅可以使用在数值之间，也可以用在日期的区间



范例：查询雇员名字叫smith的雇员

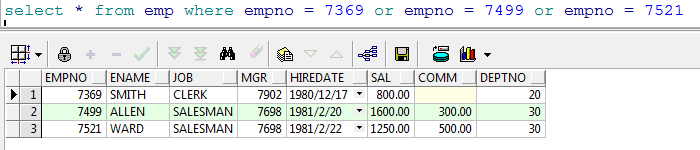
在oracle中的查询条件中查询条件的值是区分大小写的





范例:查询雇员编号是7369,7499,7521的雇员编号的具体信息

如果使用之前的做法可以使用OR关键字

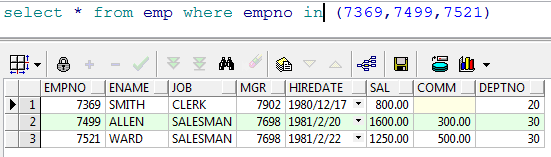


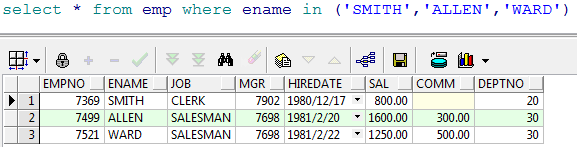
实际上，此时指定了查询范围，那么sql可以使用IN关键字

语法: 列名 IN (值1，值2，....)

列名 NOT IN (值1， 值2，...)

其中的值不仅可以是数值类型也可以是字符串





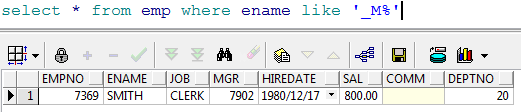
在常用的站点中经常会有模糊查询，即：输入一个关键字，把符合的内容全部的查询出来，在sql中使用LIKE语句完成。

在LIKE中主要使用以下两种通配符

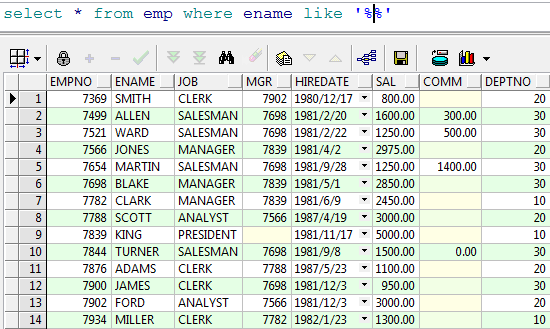
“%”：可以匹配任意长度的内容

“\_”:可以匹配一个长度的内容

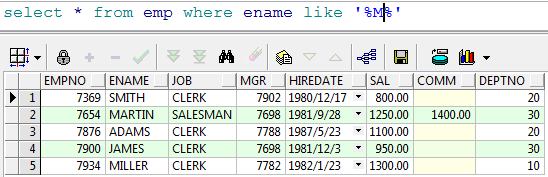
范例：查询出所有雇员姓名中第二个字符包含“M”的雇员



在LIKE中如果没有关键字表示查询全部

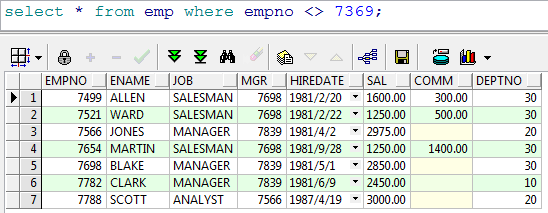


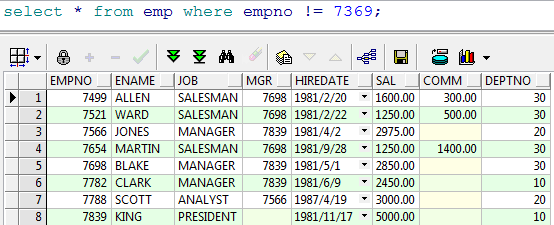
查询名字中带有“M”的雇员



在oracle中不等号的用法可以有两种形式“<>”和“!=”

范例：查询雇员编号不是7369的雇员信息





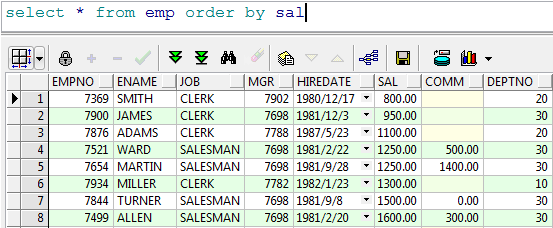
## 4.对结果集排序

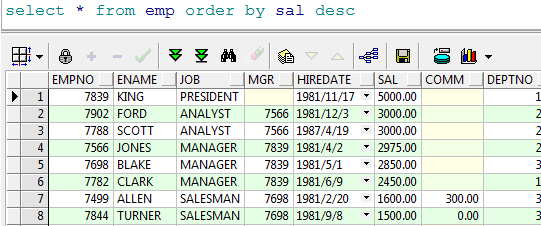
在sql中可以使用ORDER BY对查询结果进行排序

语法：SELECT \* |列名 FROM 表名 {WEHRE 查询条件} ORDER BY 列名1 ASC|DESC，列名2...ASC|DESC

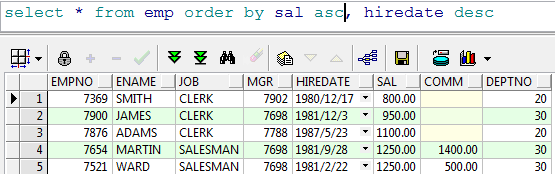
范例：查询雇员的工资从低到高

分析：ORDER BY 列名 默认的排序规则是升序排列，可以不指定ASC，如果按着降序排列必须指定DESC





如果存在多个排序字段可以用逗号分隔



注意ORDER BY语句要放在sql的最后执行。

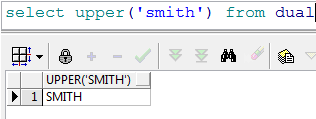
## 5.单行函数

### 1.字符函数

**接收字符输入返回字符或者数值，dual是伪表**

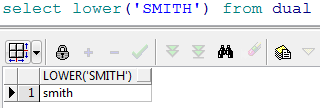
1. 把小写的字符转换成大小的字符

upper('smith')



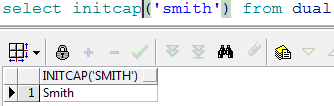
1. 把大写字符变成小写字符

lower('SMITH')



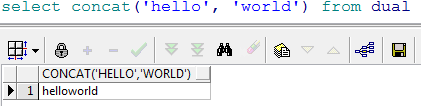
1. 把首字符大写

initcap('smith')



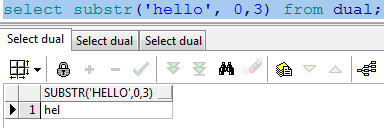
1. 字符串的连接可以使用concat可以使用“||”建议使用“||”

concat('hello', 'world')



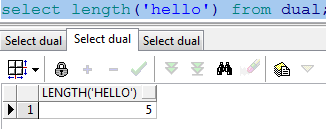
1. 字符串的截取，使用substr，第一个参数是源字符串，第二个参数是开始索引，第三个参数结束的索引，开始的索引使用1和0效果相同

substr('hello', 1,3)



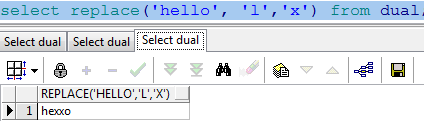
1. 获取字符串的长度

length('hello')



1. 字符串替换，第一个参数是源字符串，第二个参数被替换的字符串，第三个是替换字符串

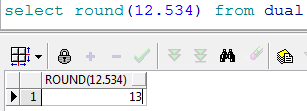
replace('hello', 'l','x')

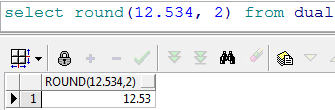


### 数值函数

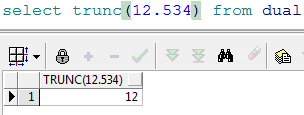
1. 四舍五入函数：ROUND()

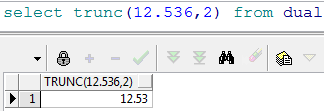
默认情况下ROUND四舍五入取整，可以自己指定保留的位数。



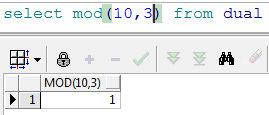


1. 取整：TRUNC()，默认全部去掉小数，也可以指定保留的位数





1. 取余数MOD()



### 3.日期函数

Oracle中提供了很多和日期相关的函数，包括日期的加减，在日期加减时有一些规律

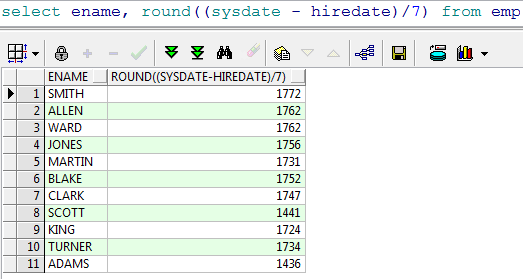
日期 – 数字 = 日期

日期 + 数字 = 日期

日期 – 日期 = 数字

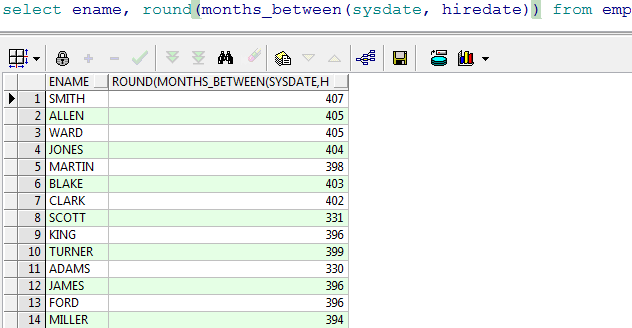
1. 范例：查询雇员的进入公司的周数。

分析：查询雇员进入公司的天数(sysdate – 入职日期)/7就是周数



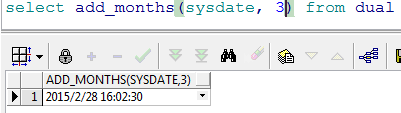
1. 获得两个时间段中的月数：MONTHS\_BETWEEN()

范例：查询所有雇员进入公司的月数



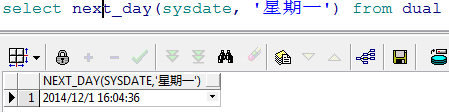
1. 获得几个月后的日期：ADD\_MONTH()

范例：求出三个月后的日期



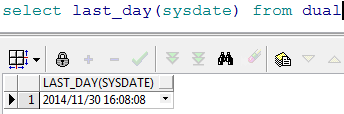
1. 指定给出下次某日期数NEXT\_DATE()

范例：求出下一个星期一是什么日期



1. 求出一个日期的最后一天

范例：求出本月的最后一天是几号



### 4.转换函数

1. TO\_CHAR:字符串转换函数

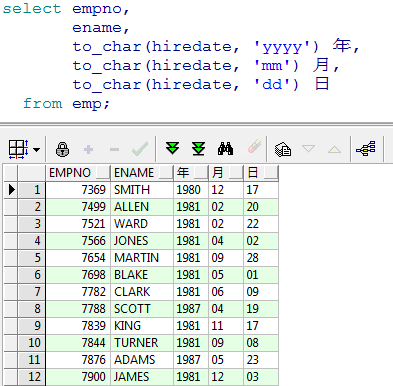
范例：查询所有的雇员将将年月日分开，此时可以使用TO\_CHAR函数来拆分

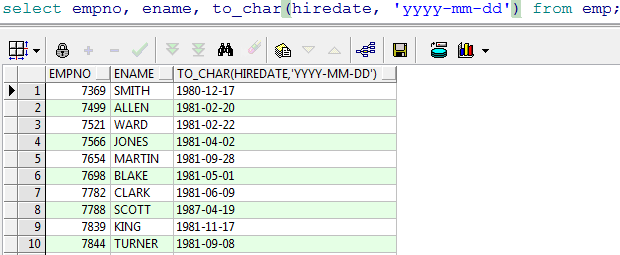
拆分时需要使用通配符

年：y, 年是四位使用yyyy

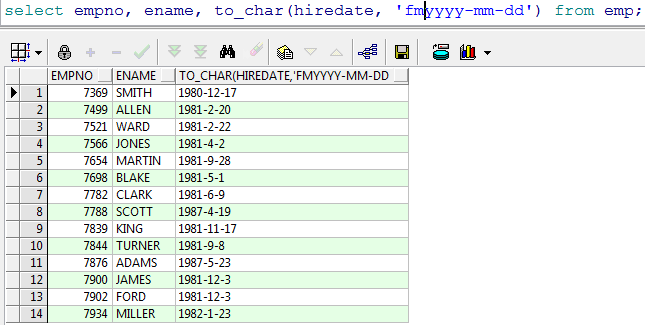
月：m, 月是两位使用mm

日：d, 日是两位使用dd



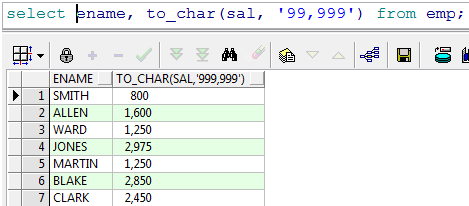


在结果中10以下的月前面被被补了前导零，可以使用fm去掉前导零

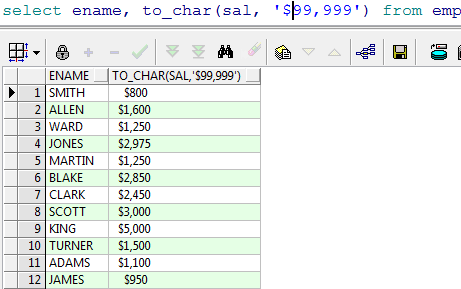


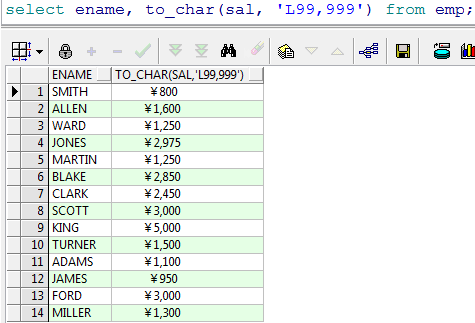
TO\_CHAR还可以给数字做格式化

范例：把雇员的工资按三位用“,”分隔，在oracle中“9”代表一位数字



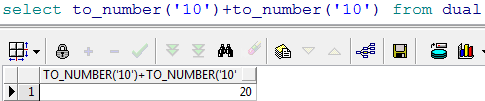
如果在钱的前面加上国家的符号可以使用“$”代表是美元，如果要使用本地的钱的单位使用“L”





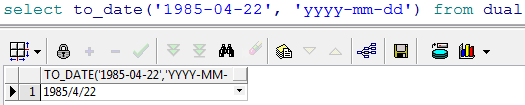
2. TO\_NUMBER:数值转换函数

TO\_NUMBER可以把字符串转换成数值



3. TO\_DATE:日期转换函数

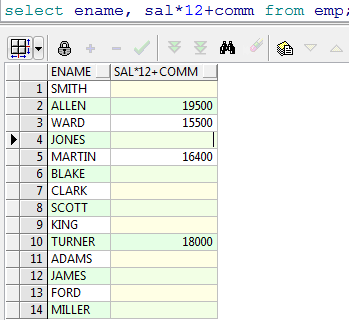
TO\_DATE可以把字符串的数据转换成日期类型



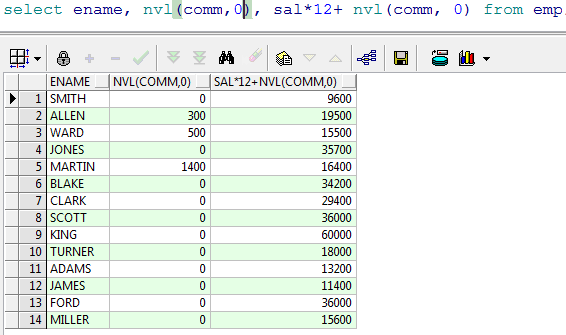
### 5.通用函数

1．空值处理nvl

范例：查询所有的雇员的年薪



我们发现很多员工的年薪是空的，原因是很多员工的奖金是null，null和任何数值计算都是null，这时我们可以使用nvl来处理。



2.Decode函数

该函数类似if....else if...esle

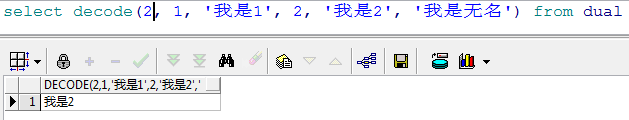
语法：DECODE(col/expression, [search1,result1],[search2, result2]....[default])

Col/expression:列名或表达式

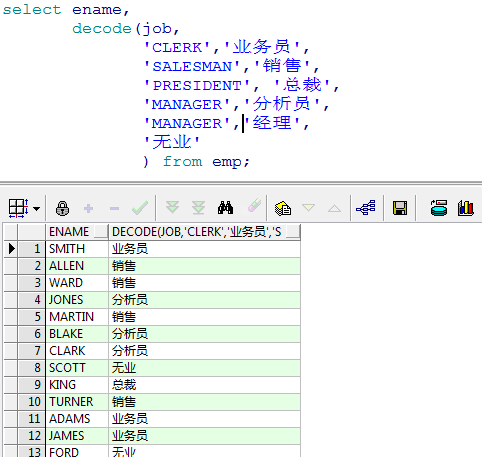
Search1，search2...:用于比较的条件

Result1, result2...:返回值

如果col/expression和Searchi匹配就返回resulti,否则返回default的默认值



范例：查询出所有雇员的职位的中文名



**3.case when**

CASE expr WHEN comparison\_expr1 THEN return\_expr1

[WHEN comparison\_expr2 THEN return\_expr2

WHEN comparison\_exprn THEN return\_exprn

ELSE else\_expr]

END

select t.empno,

t.ename,

case

when t.job = 'CLERK' then

'业务员'

when t.job = 'MANAGER' then

'经理'

when t.job = 'ANALYST' then

'分析员'

when t.job = 'PRESIDENT' then

'总裁'

when t.job = 'SALESMAN' then

'销售'

else

'无业'

end

from emp t



## 多表查询

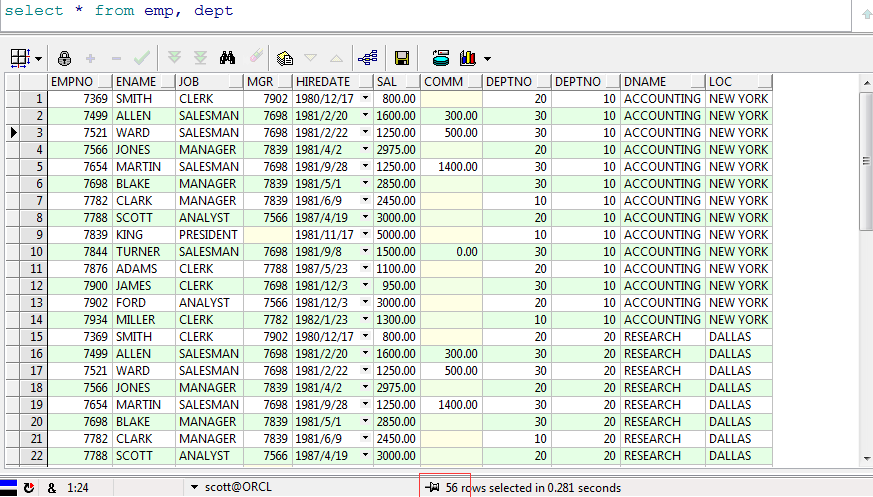
### 1.多表连接基本查询

使用一张以上的表做查询就是多表查询

语法： SELECT {DISTINCT} \*|列名.. FROM 表名 别名，表名1 别名

{WHERE 限制条件 ORDER BY 排序字段 ASC|DESC...}

范例：查询员工表和部门表



我们发现产生的记录数是56条，我们还会发现emp表是16条，dept表是4条，56正是emp表和dept表的记录数的乘积，我们称其为笛卡尔积。

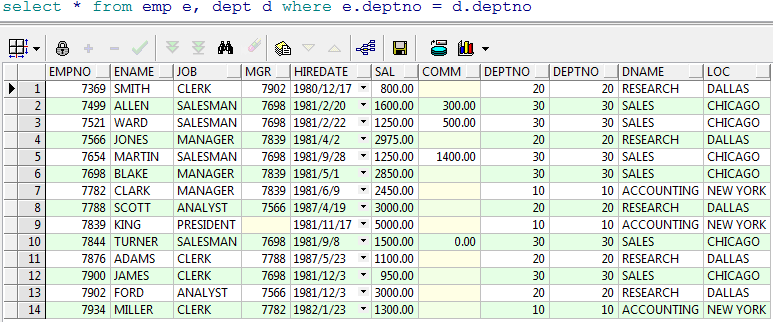
如果多张表进行一起查询而且每张表的数据很大的话笛卡尔积就会变得非常大，对性能造成影响，想要去掉笛卡尔积我们需要关联查询。

在两张表中我们发现有一个共同的字段是depno，depno就是两张表的关联的字段，我们可以使用这个字段来做限制条件，两张表的关联查询字段一般是其中一张表的主键，另一张表的外键。

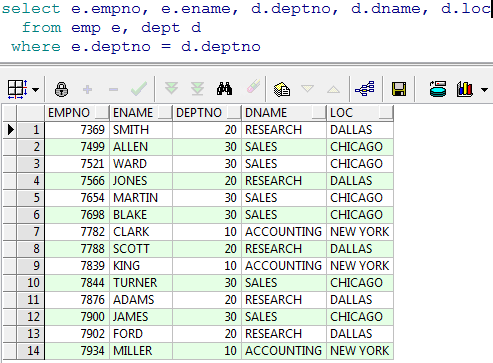


关联之后我们发现数据条数是14条，不在是56条。

多表查询我们可以为每一张表起一个别名

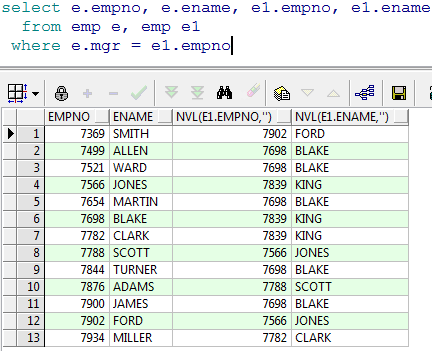


范例：查询出雇员的编号，姓名，部门的编号和名称，地址



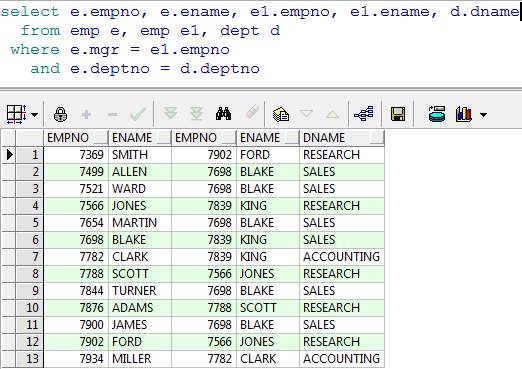
范例：查询出每个员工的上级领导

分析：emp表中的mgr字段是当前雇员的上级领导的编号，所以该字段对emp表产生了自身关联，可以使用mgr字段和empno来关联



范例:在上一个例子的基础上查询该员工的部门名称

分析：只要在上一个例子基础上再加一张表的关联，使用deptno来做关联字段即可



范例：查询出每个员工编号，姓名，部门名称，工资等级和他的上级领导的姓名，工资等级

select e.empno,

e.ename,

decode(s.grade,

1,'一级',

2,'二级',

3,'三级',

4,'四级',

5,'五级') grade,

d.dname,

e1.empno,

e1.ename,

decode(s1.grade,

1,'一级',

2,'二级',

3,'三级',

4,'四级',

5,'五级') grade

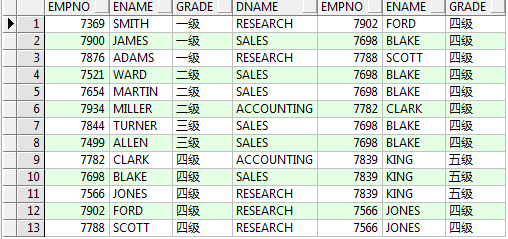
from emp e, emp e1, dept d, salgrade s, salgrade s1

where e.mgr = e1.empno

and e.deptno = d.deptno

and e.sal between s.losal and s.hisal

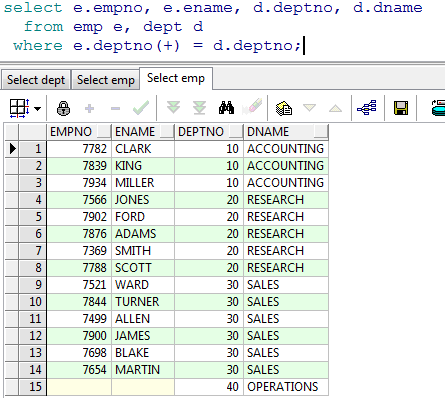
and e1.sal between s1.losal and s1.hisal



### 外连接（左右连接）

1. 右连接

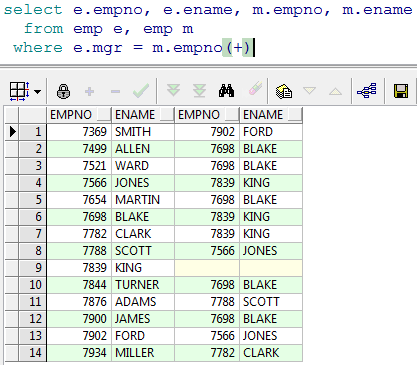
当我们在做基本连接查询的时候，查询出所有的部门下的员工，我们发现编号为40的部门下没有员工，但是要求把该部门也展示出来，我们发现上面的基本查询是办不到的



使用(+)表示左连接或者右连接，当(+)在左边表的关联条件字段上时是左连接，如果是在右边表的关联条件字段上就是右连接。

范例：查询出所有员工的上级领导

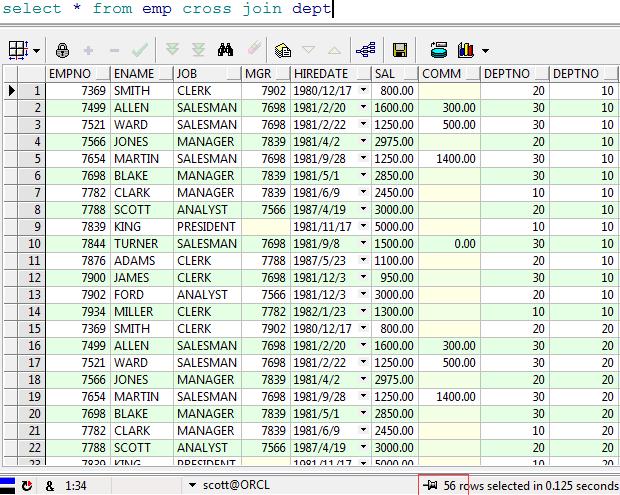
分析：我们发现使用我们以前的做法发现KING的上级领导没有被展示，我们需要使用左右连接把他查询出来



### 3.sql1999对SQL的支持

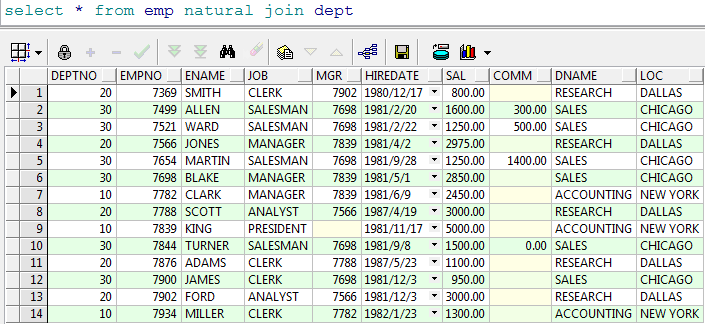
#### 1.交叉连接CROSS JOIN（了解）

交叉连接会产生笛卡尔积

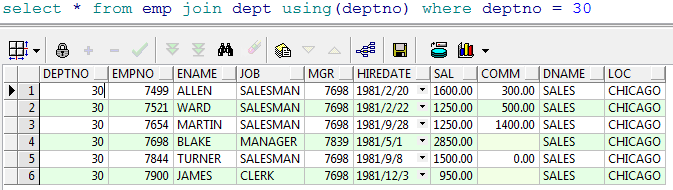


#### 自然连接NATURAL JOIN（了解）

自然连接会自动的分析管理条件进行连接，去掉笛卡尔积。

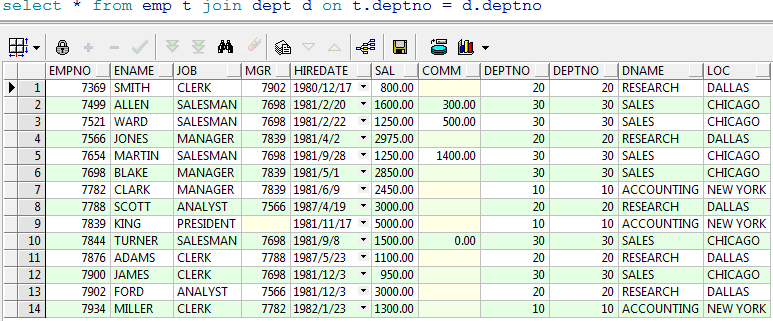


#### USING子句，直接管理操作（了解）

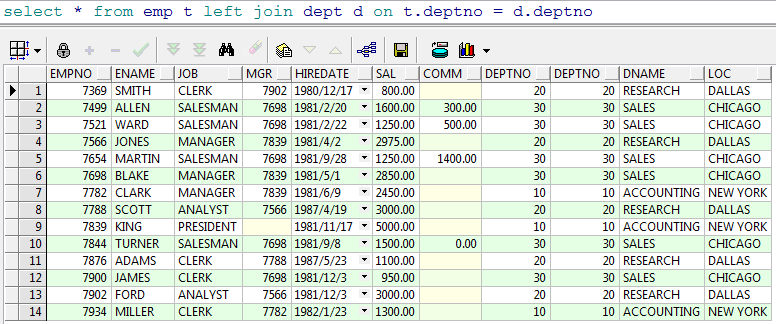


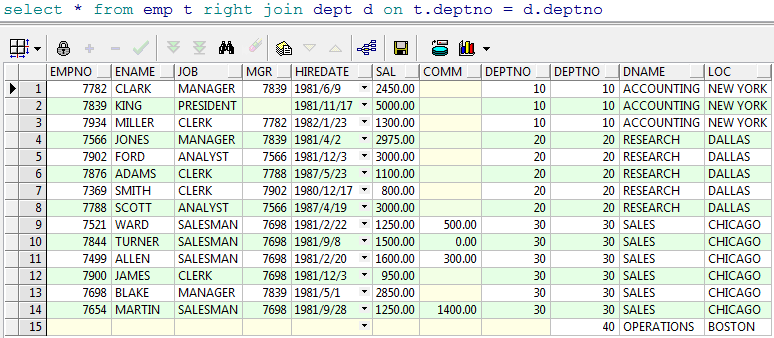
#### ON子句，自己编写连接条件（重要）

On相当于where



#### 左连接和右连接LEFT JOIN和RIGHT JOIN（重要）

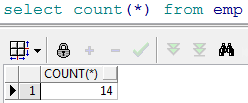




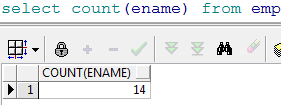
### 4.分组函数

#### 1.统计记录数count()

范例：查询出所有员工的记录数

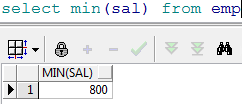


不建议使用count(\*)，可以使用一个具体的列以免影响性能。



#### 2.最小值查询min()

范例：查询出来员工最低工资



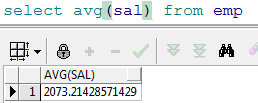
#### 3.最大值查询max()

范例：查询出员工的最高工资



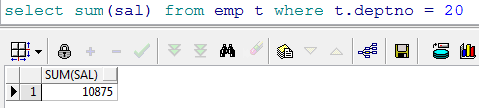
#### 4.查询平均值avg()

范例：查询出员工的平均工资



#### 5.求和函数sum()

范例：查询出20部门的员工的工资总和

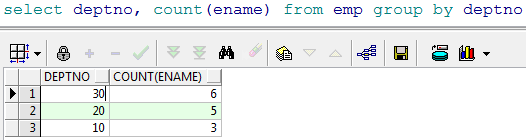


### 5.分组统计

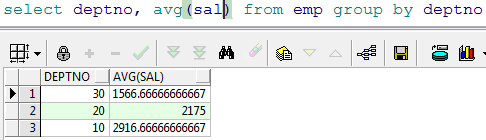
分组统计需要使用GROUP BY来分组

语法：语法：SELECT \* |列名 FROM 表名 {WEHRE 查询条件} {GROUP BY 分组字段} ORDER BY 列名1 ASC|DESC，列名2...ASC|DESC

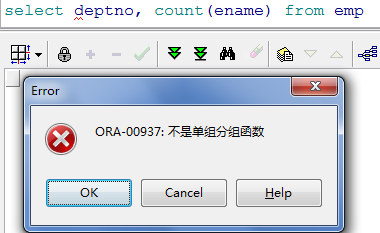
范例：查询每个部门的人数



范例：查询出每个部门的平均工资



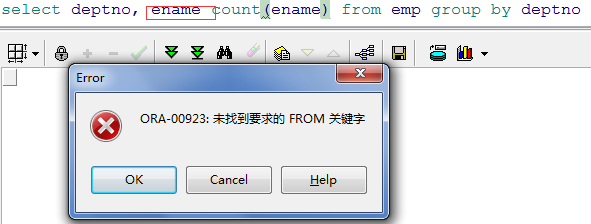
如果我们想查询出来部门编号，和部门下的人数



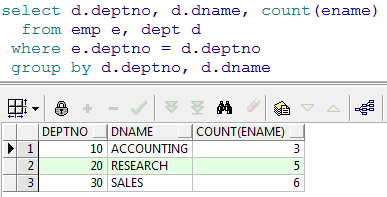
我们发现报了一个ORA-00937的错误

**注意：**

1. **如果使用分组函数，SQL只可以把GOURP BY分组条件字段和分组函数查询出来，不能有其他字段。**
2. **如果使用分组函数，不使用GROUP BY 只可以查询出来分组函数的值**

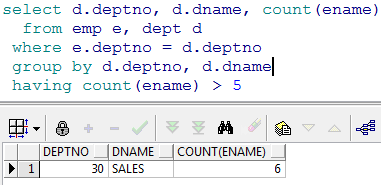
****

范例：按部门分组，查询出部门名称和部门的员工数量

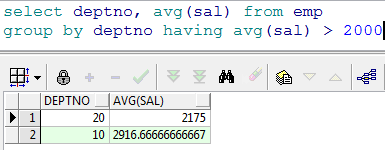


范例：查询出部门人数大于5人的部门

分析：需要给count(ename)加条件，此时在本查询中不能使用where，可以使用HAVING



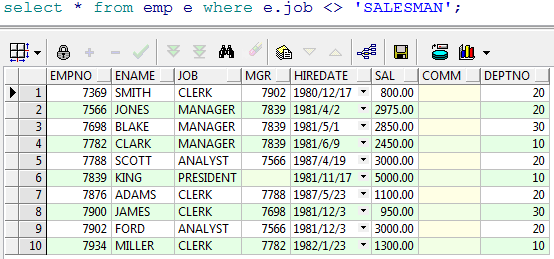
范例：查询出部门平均工资大于2000的部门



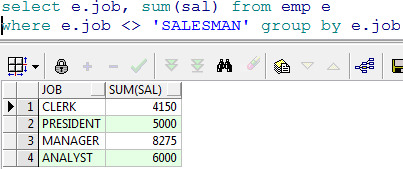
范例：显示非销售人员工作名称以及从事同一工作的员工的月工资的总和，并且要满足从事同一工作月工资总和大于5000，结果按月工资总和的升序排列。

分析：

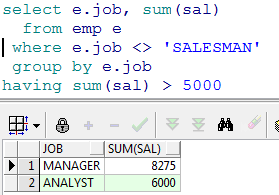
1. 查询出非销售人员



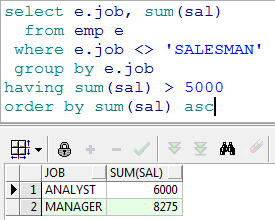
1. 以步骤1为基础按着工作分组求工资的总和



1. 以步骤2为基础查询出月工资总和大于5000的工作



1. 按着月工资的总和的升序排列



**注意：只有分组条件在结果集中是重复的分组才有意义。**

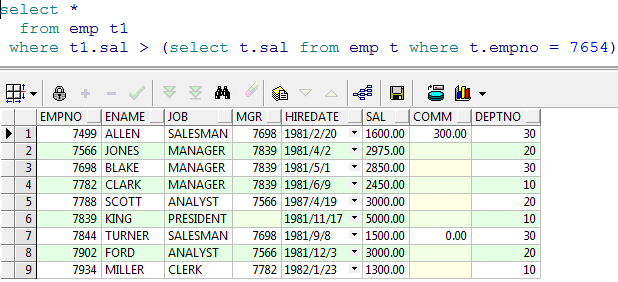
## 7.子查询

子查询：在一个查询的内部还包括另一个查询，则此查询称为子查询。

Sql的任何位置都可以加入子查询。

范例：查询比7654工资高的雇员

分析：查询出7654员工的工资是多少，把它作为条件



所有的子查询必须在“（）”中编写

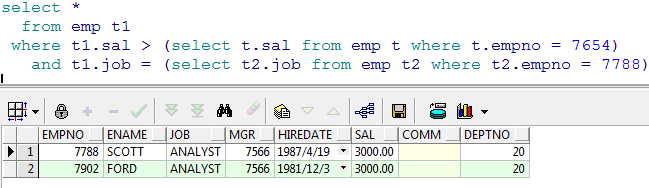
子查询在操作中有三类：

单列子查询：返回的结果是一列的一个内容

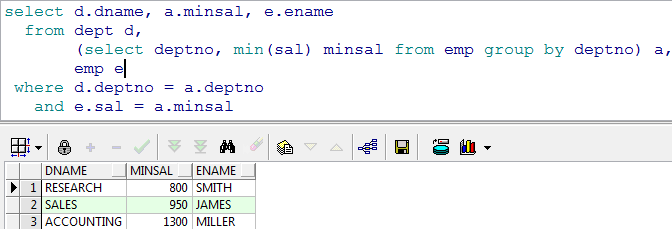
单行子查询：返回多个列，有可能是一个完整的记录

多行子查询：返回多条记录

范例：查询出比雇员7654的工资高，同时从事和7788的工作一样的员工

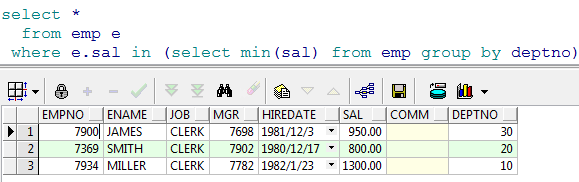


范例：要求查询每个部门的最低工资和最低工资的雇员和部门名称



在返回多条记录的子查询可以把它的结果集当做一张表，给起个别名， 如图中的a。

范例：查询出来每个部门最低工资的员工

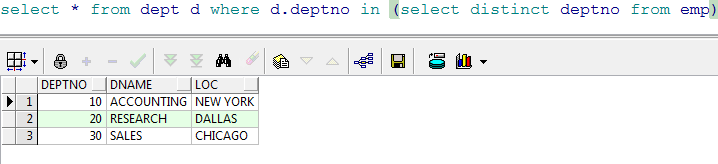


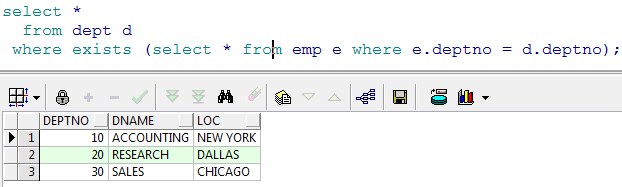
Exists和not exists关键字的用法：

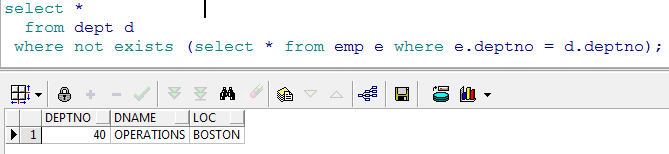
exists （sql 返回结果集为真）

not exists (sql 不返回结果集为真）

范例：查询出有员工的部门有哪些？

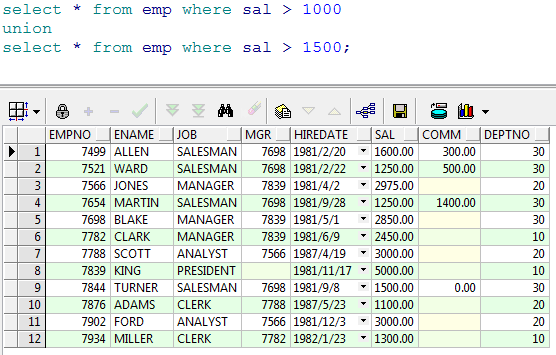


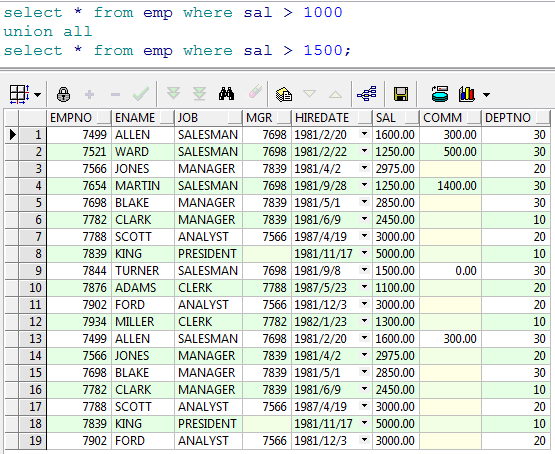




Union和Union All的用法

Union：对两个结果集进行并集操作，不包括重复行。

  
Union All：对两个结果集进行并集操作，包括重复行。



## 8数据库变更

为了保存原始emp的信息保存，我们复制一份emp表

Create table myemp as select \* from emp;

### Insert（增加）

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 myemp values(7789, '张三', '开发', 7839, to\_date('1992-10-22', 'yyyy-MM-dd'), 2000, 200, 10);

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

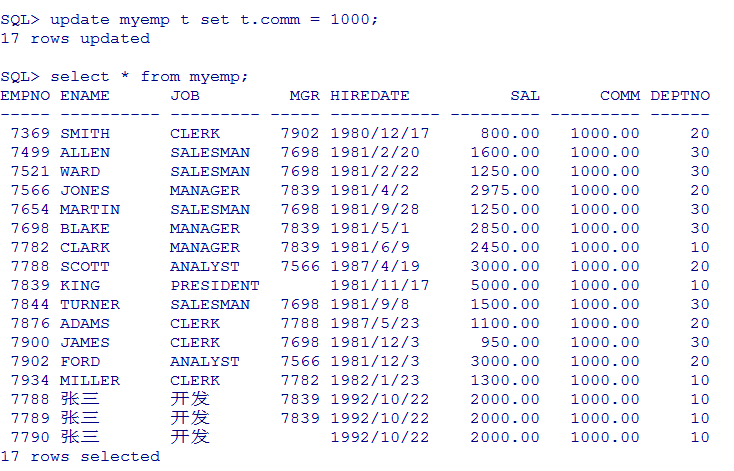
insert into myemp values(7790, '张三', '开发', null, to\_date('1992-10-22', 'yyyy-MM-dd'), 2000, null, 10);

### update（修改）

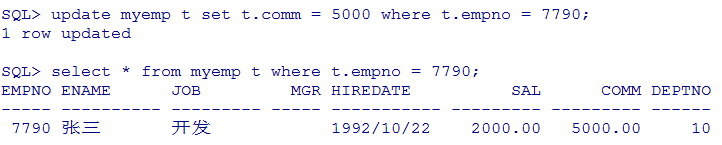
全部修改：UPDATE 表名 SET 列名1=值1，列名2=值2，....

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

全部更新



局部更新

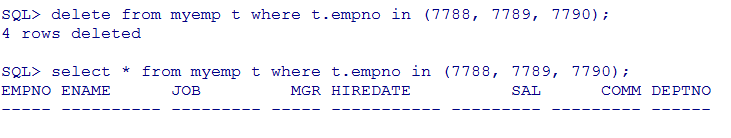


把张三的领导取消



### delete（删除）

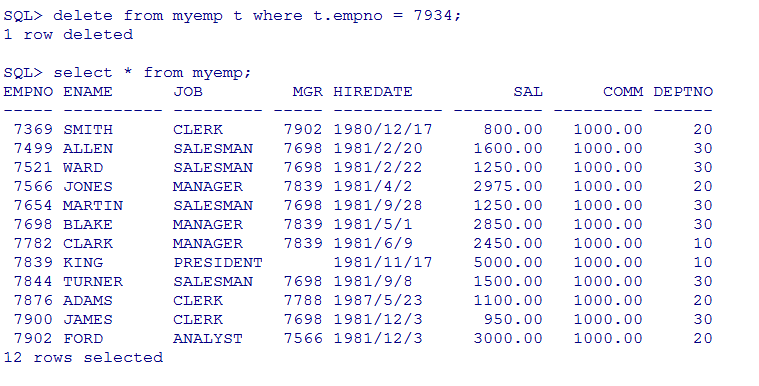
语法 : DELETE FROM 表名 WHERE 删除条件;



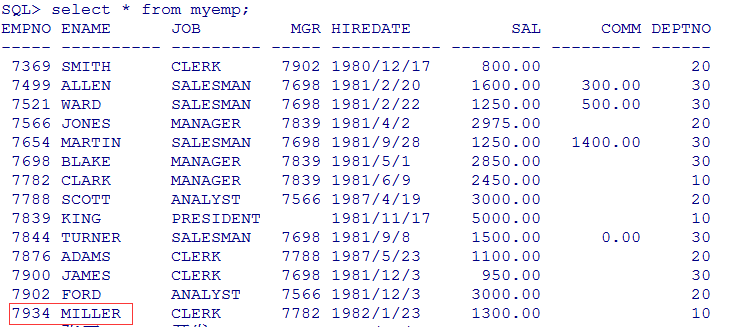
在删除语句中如果不指定删除条件的话就会删除所有的数据

## 9.事物处理

范例：删除员工7934；



从结果上看似乎数据已经删除，但是我们再打开另一个窗口查看发现7934的员工还在



事务处理：所谓的事务管理就是要保证数据的完整性，要么同时成功，要么同时失败

当我们执行完delete语句时，我们去数据库中去查看数据，发现并没有我们删除数据，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

在oracle中会数据库变更会发生锁的情况（此处应用可以解决项目多线程并发带来的数据安全问题）

当两个数据库的连接同时来修改同一数据时，一定会有一连接先修改，另一个连接就会等待直到第一个连接修改完毕再修改

## 10.表的管理

### 1.常用的数据类型

|  |  |  |
| --- | --- | --- |
| No | 数据类型 | 描述 |
| 1 | Varchar， varchar2 | 表示一个字符串 |
| 2 | NUMBER | NUMBER(n)表示一个整数，长度是n |
| NUMBER(m,n):表示一个小数，总长度是m，小数是n，整数是m-n |
| 3 | DATA | 表示日期类型 |
| 4 | CLOB | 大对象，表示大文本数据类型，可存4G |
| 5 | BLOB | 大对象，表示二进制数据，可存4G |

### 2建表

语法：

Create table 表名（

字段1 数据类型 [default 默认值],

字段2 数据类型 [default 默认值],

...

字段n 数据类型 [default 默认值]

）;

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'));

### 2.表删除

语法：DROP TABLE 表名

### 3.表的修改

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]...)

注意修改时如果原有某列的数据长度为200，则不可以把该列修改成小于200的列

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));

### 截断表

在person表中使用delete语句删除数据，则可以使用rollback来回滚，如果想要清空一张表的数据，同时不想回滚可以立即释放资源需要使用截断表的语法

语法：TRUNCATE TABLE 表名

范例：截断person表

truncate table person;

## 11.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。

### 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,

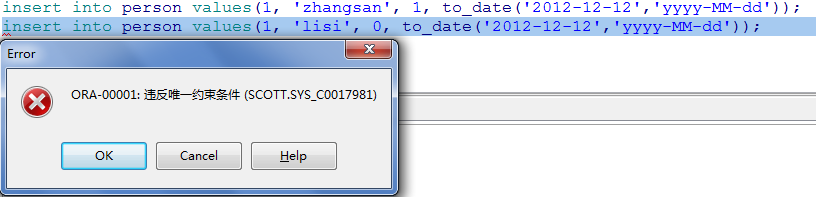
name varchar2(10),

gender number(1) default 1,

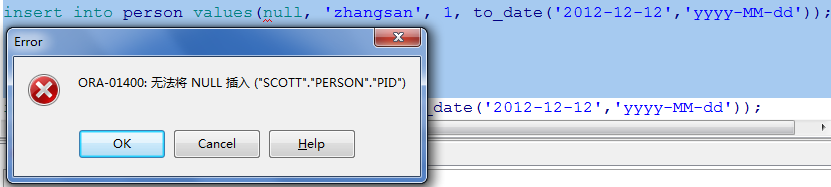
birthday date

);

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),

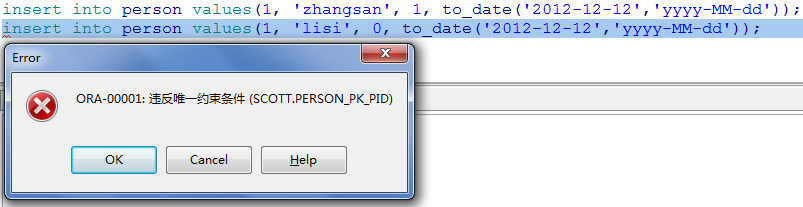
name varchar2(10),

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(

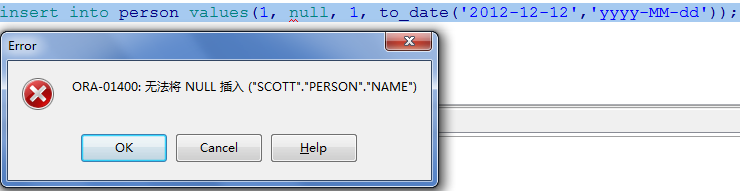
pid number(10) not null,

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(

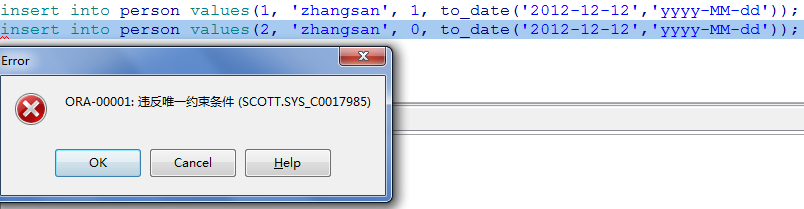
pid number(10) ,

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) ,

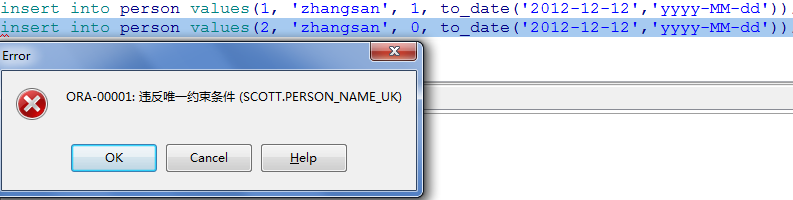
name varchar2(10),

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(

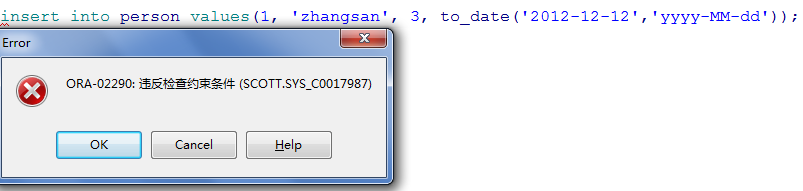
pid number(10) ,

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) ,

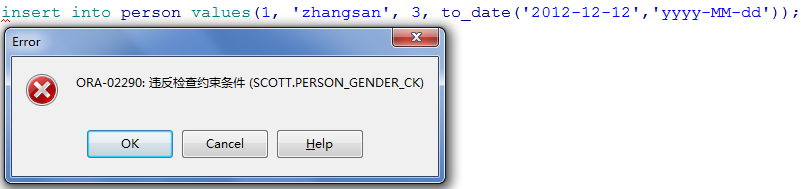
name varchar2(10),

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),

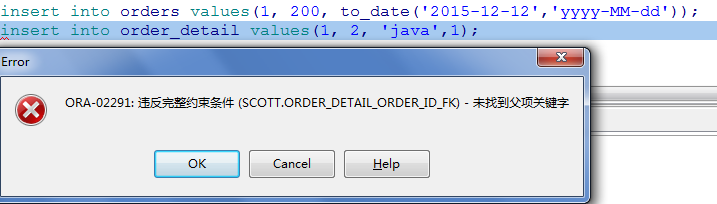
item\_name varchar2(10),

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)

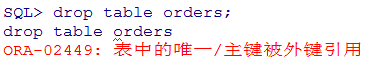
);



外键关联一定注意：

外键一定是主表的主键

删表时一定先删字表再删主表，如果直接删主表会出现由于约束存在无法删除的问题



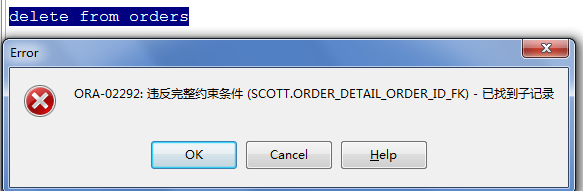
但是可以强制删除drop table orders cascade constraint;(不建议)

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



## rownum

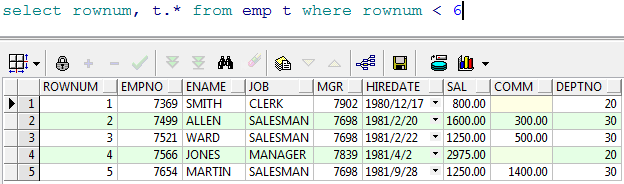
ROWNUM:表示行号，实际上此是一个列,但是这个列是一个伪列,此列可以在每张表中出现。

范例：查询emp表带有rownum列

select rownum, t.\* from emp t



我们可以根据rownum来取结果集的前几行，比如前5行



但是我们不能取到中间几行，因为rownum不支持大于号，只支持小于号，如果想 实现我们的需求怎么办呢？答案是使用子查询，也正是oracle分页的做法。

select \*

from (select rownum rm, a.\* from (select \* from emp) a where rownum < 11) b where b.rm > 5

## 视图

视图就是封装了一条复杂查询的语句。

语法1.：CREATE VIEW 视图名称 AS 子查询

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

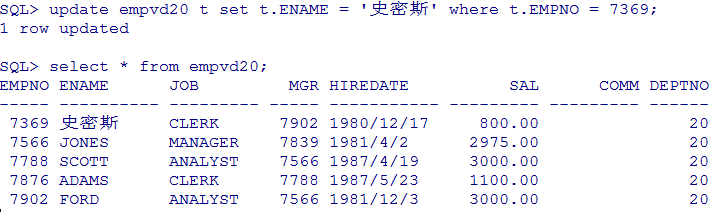


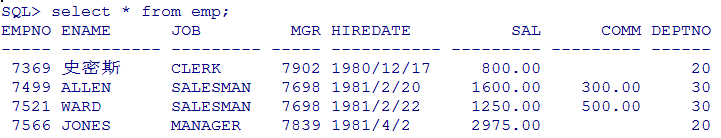
语法2：CREATE OR REPLACE VIEW 视图名称 AS 子查询

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

那么视图可以修改吗？



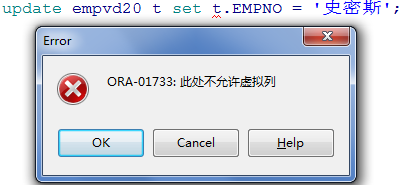


我们尝试着修改视图但是发现是视图所查询的表的字段值被修改了。所以我们一般不会去修改视图。

我们可以设置视图为只读。

语法3：CREATE OR REPLACE VIEW 视图名称 AS 子查询 WITH READ ONLY

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



## 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。

语法：CREATE SEQUENCE 序列名

[INCREMENT BY n]

[START WITH n]

[{MAXVALUE/ MINVALUE n|NOMAXVALUE}]

[{CYCLE|NOCYCLE}]

[{CACHE n|NOCACHE}];

范例:创建一个seqpersonid的序列,验证自动增长的操作

CREATE SEQUENCE seqpersonid;

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：

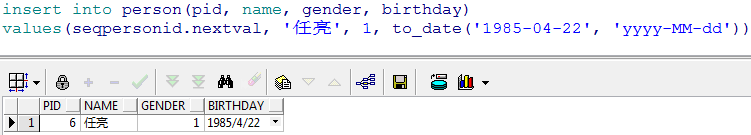
NextVal :取得序列的下一个内容

CurrVal :取得序列的当前内容

select seqpersonid.nextval from dual;

select seqpersonid.currval from dual;

在插入数据时需要自增的主键中可以这样使用



在实际项目中每一张表会配一个序列，但是表和序列是没有必然的联系的，一个序列被哪一张表使用都可以，但是我们一般都是一张表用一个序列。

序列的管理一般使用工具来管理。

## 15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。索引有很多种我们主要介绍常用的几种:

为什么添加了索引之后，会加快查询速度呢？

**图书馆**：如果杂乱地放书的话检索起来就非常困难,所以将书分类，然后再建一个箱子，箱

子里面放卡片，卡片里面可以按类查询,按姓名查或者类别查,这样的话速度会快很多很多，

这个就有点像索引。索引的好处就是提高你找到书的速度，但是正是因为你建了索引，就应

该有人专门来维护索引，维护索引是要有时间精力的开销的，也就是说索引是不能乱建的，

所以建索引有个原则：如果有一个字段如果不经常查询，就不要去建索引。现在把书变成我

们的表，把卡片变成我们的索引，就知道为什么索引会快，为什么会有开销。

创建索引的语法：

创建索引：

1． 单例索引

单例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);

索引的使用原则：

* 在大表上建立索引才有意义
* 在where子句后面或者是连接条件上建立索引
* 索引的层次不要超过4层

# 9.pl/sql

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：为职工长工资，每人长10％的工资。

Update emp set sal=sal\*1.1

范例2：例2: 按职工的职称长工资,总裁长1000元,经理长800元，其他人员长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程序语法

### 1.程序语法：

declare

说明部分 （变量说明，游标申明，例外说明 〕

begin

语句序列 （DML语句〕…

exception

例外处理语句

End;

### 2.常量和变量定义

在程序的声明阶段可以来定义常量和变量。

* 变量的基本类型就是oracle中的建表时字段的变量如char, varchar2, date, number, boolean, long)，

定义语法：varl char(15);

Psal number(9,2);

说明变量名、数据类型和长度后用分号结束说明语句。

常量定义：married boolean:=true

* 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;

### 3. if分支

语法1：

**IF 条件 THEN 语句1;**

**语句2;**

**END IF;**

语法2：

**IF 条件 THEN 语句序列1；**

**ELSE 语句序列 2；**

**END IF；**

语法3：

**IF 条件 THEN 语句;**

**ELSIF 语句 THEN 语句;**

**ELSE 语句;**

**END IF;**

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;

### 4.LOOP循环语句

其中语法2比较常用

语法1：

**WHILE total <= 25000 LOOP**

**.. .**

**total : = total + salary;**

**END LOOP;**

语法2：

**Loop**

**EXIT [when 条件];**

**……**

**End loop**

语法3：

**FOR I IN 1 . . 3 LOOP**

**语句序列 ;**

**END LOOP ;**

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;

### 5.游标Cursor

在写java程序中有集合的概念，那么在pl/sql中也会用到多条记录，这时候我们就要用到游标，游标可以存储查询返回的多条数据。

语法：

CURSOR 游标名 [ (参数名 数据类型,参数名 数据类型,...)] IS SELECT 语句;

例如：cursor c1 is select ename from emp;

游标的使用步骤：

* 打开游标： open c1; (打开游标执行查询)
* 取一行游标的值：fetch c1 into pjob; (取一行到变量中)
* 关闭游标： close c1;(关闭游标释放资源)
* 游标的结束方式 exit when c1%notfound
* 注意： 上面的pjob必须与emp表中的job列类型一致：

定义：pjob emp.empjob%type;

范例1：使用游标方式输出emp表中的员工编号和姓名

declare

cursor pc is

select \* from emp;

pemp emp%rowtype;

begin

open pc;

loop

fetch pc

into pemp;

exit when pc%notfound;

dbms\_output.put\_line(pemp.empno || ' ' || pemp.ename);

end loop;

close pc;

end;

范例2：按员工的工种长工资,总裁1000元，经理长800元其，他人员长400元。

declare

cursor pc is

select \* from myemp;

addsal myemp.sal%type;

pemp myemp%rowtype;

begin

open pc;

loop

fetch pc

into pemp;

exit when pc%notfound;

if pemp.job = 'PRESIDENT' then

addsal := 1000;

elsif pemp.job = 'MANAGER' then

addsal := 800;

else

addsal := 400;

end if;

update myemp t set t.sal = t.sal + addsal where t.empno = pemp.empno;

end loop;

close pc;

end;

范例3：写一段PL/SQL程序，为部门号为10的员工涨工资。

declare

cursor pc(dno myemp.deptno%type) is

select empno from myemp where deptno = dno;

pno myemp.empno%type;

begin

open pc(20);

loop

fetch pc

into pno;

exit when pc%notfound;

update myemp t set t.sal = t.sal + 1000 where t.empno = pno;

end loop;

close pc;

end;

### 6.例外

例外是程序设计语言提供的一种功能，用来增强程序的健壮性和容错性。

系统定义例外

no\_data\_found (没有找到数据)

too\_many\_rows (select …into语句匹配多个行)

zero\_divide ( 被零除)

value\_error (算术或转换错误)

timeout\_on\_resource (在等待资源时发生超时)

范例1：写出被0除的例外的plsql程序

declare

pnum number;

begin

pnum := 1 / 0;

exception

when zero\_divide then

dbms\_output.put\_line('被0除');

when value\_error then

dbms\_output.put\_line('数值转换错误');

when others then

dbms\_output.put\_line('其他错误');

end;

用户也可以自定义例外，在声明中来定义例外

DECLARE

My\_job char(10);

v\_sal emp.sal%type;

No\_data exception;

cursor c1 is select distinct job from emp order by job;

如果遇到异常我们要抛出raise no\_data;

范例：查询部门编号是50的员工

declare

no\_emp\_found exception;

cursor pemp is

select t.ename from emp t where t.deptno = 50;

pename emp.ename%type;

begin

open pemp;

fetch pemp

into pename;

if pemp%notfound then

raise no\_emp\_found;

end if;

close pemp;

exception

when no\_emp\_found then

dbms\_output.put\_line('没有找到员工');

when others then

dbms\_output.put\_line('其他错误');

end;

## 2.存储过程

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的SQL 语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。

创建存储过程语法：

**create [or replace] PROCEDURE 过程名[(参数名 in/out 数据类型)]**

**AS**

**begin**

**PLSQL子程序体；**

**End;**

**或者**

**create [or replace] PROCEDURE 过程名[(参数名 in/out 数据类型)]**

**is**

**begin**

**PLSQL子程序体；**

**End 过程名;**

范例：创建一个输出helloword的存储过程

create or replace procedure helloworld is

begin

dbms\_output.put\_line('helloworld');

end helloworld;

调用存储过程

在plsql中调用存储过程

begin

*-- Call the procedure*

helloworld;

end;

范例2：给指定的员工涨100工资，并打印出涨前和涨后的工资

分析：我们需要使用带有参数的存储过程

create or replace procedure addSal1(eno in number) is

pemp myemp%rowtype;

begin

select \* into pemp from myemp where empno = eno;

update myemp set sal = sal + 100 where empno = eno;

dbms\_output.put\_line('涨工资前' || pemp.sal || '涨工资后' || (pemp.sal + 100));

end addSal1;

调用

begin

*-- Call the procedure*

addsal1(eno => 7902);

commit;

end;

## 3.存储函数

**create or replace function 函数名(Name in type, Name in type, ...) return 数据类型 is**

**结果变量 数据类型;**

**begin**

**return(结果变量);**

**end函数名;**

存储过程和存储函数的区别

一般来讲，过程和函数的区别在于函数可以有一个返回值；而过程没有返回值。

但过程和函数都可以通过out指定一个或多个输出参数。我们可以利用out参数，在过程和函数中实现返回多个值。

范例：使用存储函数来查询指定员工的年薪

create or replace function empincome(eno in emp.empno%type) return number is

psal emp.sal%type;

pcomm emp.comm%type;

begin

select t.sal into psal from emp t where t.empno = eno;

return psal \* 12 + nvl(pcomm, 0);

end;

使用存储过程来替换上面的例子

create or replace procedure empincomep(eno in emp.empno%type, income out number) is

psal emp.sal%type;

pcomm emp.comm%type;

begin

select t.sal, t.comm into psal, pcomm from emp t where t.empno = eno;

income := psal\*12+nvl(pcomm,0);

end empincomep;

调用：

declare

income number;

begin

empincomep(7369, income);

dbms\_output.put\_line(income);

end;

使用java代码调用存储过程和函数

|  |
| --- |
| /\*Class.forName("oracle.jdbc.OracleDriver");  Connection conn = null;  conn = DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:orcl",  "scott", "tiger");  CallableStatement call = conn.prepareCall("{call countyearsal(?,?)}");  call.setInt(1, 7369);    call.registerOutParameter(2, OracleTypes.NUMBER);  call.execute();  int sum = call.getInt(2);  System.out.println(sum);\*/    Class.*forName*("oracle.jdbc.OracleDriver");  Connection conn = **null**;  conn = DriverManager.*getConnection*("jdbc:oracle:thin:@127.0.0.1:1521:orcl",  "scott", "tiger");  CallableStatement call = conn.prepareCall("{?= call countyearsal1(?)}");  call.registerOutParameter(1, OracleTypes.*NUMBER*);  call.setInt(2, 7369);    call.execute();  **int** sum = call.getInt(1);  System.*out*.println(sum); |

## 4.触发器

数据库触发器是一个与表相关联的、存储的PL/SQL程序。每当一个特定的数据操作语句(Insert,update,delete)在指定的表上发出时，Oracle自动地执行触发器中定义的语句序列。

**触发器可用于**

数据确认

实施复杂的安全性检查

做审计，跟踪表上所做的数据操作等

数据的备份和同步

**触发器的类型**

语句级触发器 ：在指定的操作语句操作之前或之后执行一次，不管这条语句影响 了多少行 。

行级触发器（FOR EACH ROW） ：触发语句作用的每一条记录都被触发。在行级触 发器中使用old和new伪记录变量, 识别值的状态。

语法：

**CREATE [or REPLACE] TRIGGER 触发器名**

**{BEFORE | AFTER}**

**{DELETE | INSERT | UPDATE [OF 列名]}**

**ON 表名**

**[FOR EACH ROW [WHEN(条件) ] ]**

**begin**

**PLSQL 块**

**End 触发器名**

范例：插入员工后打印一句话“一个新员工插入成功”

create or replace trigger testTrigger

after insert on person

declare

*-- local variables here*

begin

dbms\_output.put\_line('一个员工被插入');

end testTrigger;

范例：不能在休息时间插入员工

create or replace trigger validInsertPerson

before insert on person

declare

weekend varchar2(10);

begin

select to\_char(sysdate, 'day') into weekend from dual;

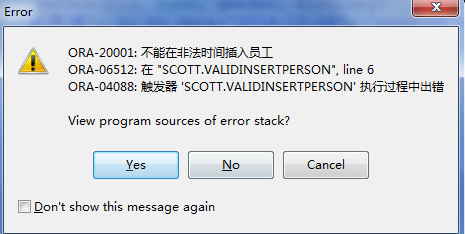
if weekend in ('星期一') then

raise\_application\_error(-20001, '不能在非法时间插入员工');

end if;

end validInsertPerson;

当执行插入时会报错



在触发器中触发语句与伪记录变量的值

|  |  |  |
| --- | --- | --- |
| **触发语句** | **:old** | **:new** |
| Insert | 所有字段都是空(null) | 将要插入的数据 |
| Update | 更新以前该行的值 | 更新后的值 |
| delete | 删除以前该行的值 | 所有字段都是空(null) |

范例：判断员工涨工资之后的工资的值一定要大于涨工资之前的工资

create or replace trigger addsal4p

before update of sal on myemp

for each row

begin

if :old.sal >= :new.sal then

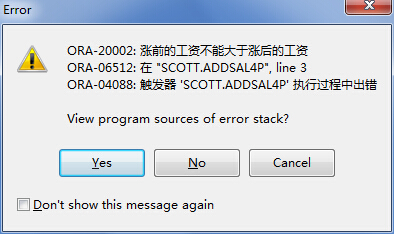
raise\_application\_error(-20002, '涨前的工资不能大于涨后的工资');

end if;

end;

调用

update myemp t set t.sal = t.sal - 1;



create tablespace my0108

logging

datafile 'D:\oracle\product\10.2.0\oradata\orcl\my0108.dbf'

size 32m

autoextend on

next 32m maxsize 2048m

extent management local;