# 数据添加

## 数据添加语法

语法1：

**INSERT INTO** 表名 (列名1,列名2,...,列名n)

**VALUES** (值1,值2,...,值n);

注意：

列名顺序没有要求

值要与列一一对应

语法2：不是很推荐

**INSERT INTO** 表名

**VALUES** (值1,...,值n);

注意：

如果所有的字段都显式指定值，那么可以省略字段，直接指定值，和表定义时列的顺序对应

实例：

INSERT INTO t\_student (id,stu\_no,name,sex,age,class\_no)

VALUES (2,10002,'mary',1,15,1);

或者：

INSERT INTO t\_student

VALUES (3,'lily',10003,1,16,1); -- name字段和stu\_no字段的位置

## DML语句和commit

对于DML操作，需要执行COMMIT;提交数据。

COMMT语句是提交事务的语句，COMMIT语句可以用来提交当前事务的所有更改。提交后，其他用户将能够看到您的更改。

COMMIT把整个DML未提交的数据一起都提交了

# 单表查询

## 准备工作

TIPS：

|  |
| --- |
| sysdate:当前系统时间，date类型  查询当前系统时间：  select sysdate from dual; |

date类型时间，精确到秒。

中文默认格式：'05-6月18'

英文默认格式：'05-JUN-18'

修改当前会话日期语言到英文：

alter session set nls\_date\_language=english;

NLS : Native Language Support

查看系统中的各种NLS参数：

|  |
| --- |
| select \* from **v$nls\_parameters;** |

创建用户oace/oace :

CREATE USER oace IDENTIFIED BY oace;

授权 connect, resource

GRANT connect, resource TO oace;

切换到oace用户

conn oaec/oaec

修改会话日期语言为英文

alter session set nls\_date\_language=english;

导入脚本 oaec.sql

@oaec.sql (注意路径)

查询当前用户所有的表名：

select table\_name from user\_tables;

|  |
| --- |
| SQL> select table\_name from user\_tables;  TABLE\_NAME  ------------------------------------------------------------  S\_CUSTOMER  S\_DEPT  S\_EMP  S\_IMAGE  S\_INVENTORY  S\_ITEM  S\_LONGTEXT  S\_ORD  S\_PRODUCT  S\_REGION  S\_TITLE  TABLE\_NAME  ------------------------------------------------------------  S\_WAREHOUSE  已选择12行。 |

表的结构介绍：

查看表的结构：desc <表名>;

表名：s\_region 区域表

id:主键，区域编号

name：区域名

表名：s\_dept 部门表

id:主键，部门编号

name:部门名

region\_id:所在区域的编号，外键

表名：s\_emp 员工表

id:主键，员工编号

last\_name:姓

first\_name:名

userid:工号

start\_date:入职时间

comments:介绍

manager\_id:上级的编号，外键

title:职位

dept\_id:所在部门的编号，外键

salary:薪资

commission\_pct:提成的百分数

表名：s\_customer 客户表

id:主键

name:名字

sales\_rep\_id:对应的销售的编号，外键，s\_emp(id)

region\_id:所在的区域编号,外键，s\_region(id)

## 基本查询语句

基本查询语句格式：

|  |
| --- |
| SELECT 列名,...,列名  FROM 表名; |

1. SELECT…FROM…是查询语句**必须的**两个子句。

DUAL表：伪表，利用这个伪表可以设置或查看序列，或者是调用一些内置的函数，方便操作。它的存在,是为了操作上的方便.因为select 都是要有特定对象的。

|  |
| --- |
| select sysdate from dual;  select 99\*99 from dual; |

1. 基本查询语句的执行流程：

FROM，确定表-->SELECT,确定列（字段）

查询所有的字段，可以用\*。不推荐。

**查询练习：**

**练习1：查询部门表的所有信息**

|  |
| --- |
| SELECT id,name,region\_id  FROM s\_dept; |

**练习2：查询员工的名,姓**

|  |
| --- |
| SELECT first\_name,last\_name  FROM s\_emp; |

查询结果显示，如果列太宽，可以使用col进行调整列宽。仅限于本次会话。

列宽调整的语句：

文本型:col 列名 format a长度; a20:表示20个字符宽度

数值型:col 列名 format 9999; 一个9表示一位

列宽的调整实例：

|  |
| --- |
| col first\_name format a20;  col last\_name format a20;  select first\_name, last\_name from s\_emp; |

1. 针对数值型，日期型的列，可以在查询中进行四则运算。

例1：查询每个员工的月薪(salary),年薪(salary\*12)

|  |
| --- |
| SELECT id,salary,salary\*12  FROM s\_emp; |

空值问题：空值是无效的，未指定的，未知的或不可预知的值。

空值不是空格或0。

如果某个列值是空的,对应的表达式结果也是空的。

NVL函数语法： NVL(eExpression1, eExpression2)

如果 eExpression1 的计算结果为 null 值，则 NVL( ) 返回 eExpression2。如果 eExpression1 的计算结果不是 null 值，则返回 eExpression1。eExpression1 和 eExpression2 可以是任意一种数据类型。如果 eExpression1 与 eExpression2 的结果皆为 null 值，则 NVL( ) 返回 .NULL.。

假设有张表test中有个字段name，可能存在null ：

select NVL(name,”空的”) from test;

如果name是空的，那么显示的值是”空的”。

或者把NVL理解成空值转换函数：NVL(列或者表达式, 默认值)

例2：计算员工的工资，注意有些员工存在提成。

考虑提成，那么员工的工资计算如下：salary\*(1+ commission\_pct/100)

注意：只有部分员工存在提成，没有提成的员工，这个字段是空值。

|  |
| --- |
| SELECT id,salary\*(1+NVL(commission\_pct,0)/100)  FROM s\_emp; |

1. 查询时，可以给表，列，或者表达式取别名。

给列或表达式取别名：

select <字段名或表达式> [别名] from <表>

或者： select <字段名或表达式> [as 别名] from <表>

给表取别名：

select <字段名或表达式> [别名] from <表名> [别名]

表的别名不可以使用as。

例3：查询员工信息，提供员工编号，工号，职位和薪资

|  |
| --- |
| select id "员工编号", userid "工号", title "职位", salary "薪资"  from s\_emp; |

|  |
| --- |
| select a.id "员工编号", a.userid "工号", a.title "职位", a.salary "薪资"  from s\_emp a; |

1. 消除重复的数据

语句： SELECT DISTINCT <列名或表达式…> FROM <表名>

例4：查询部门名字，比较不用DISTINCT和使用DISTINCT，两种查询语句结果的不同。

|  |
| --- |
| SELECT name FROM s\_dept;  SELECT DISTINCT name FROM s\_dept; |

1. 字符串的拼接

Oracle中采用”||”拼接字符串。

例5: 查询员工的编号，姓名和薪资。

|  |
| --- |
| col 姓名 format a30;  select id "编号",first\_name|| ', '||last\_name "姓名",salary "薪资"  from s\_emp; |

注意：', '是单引号，而非双引号。

补充：Oracle中，单引号和双引号的区别。

|  |
| --- |
| 双引号的作用：**关键字，对象名、字段名加双引号，则示意 Oracle将严格区分大小写，否则Oracl都默认大写。** 1.双引号引住关键字 select "sysdate" from dual;  等同于select sysdate from dual; 如下则不会输出时间，只会输出sysdate这个字符串。 select 'sysdate' from dual;  2.双引号引住字段名  select \* from emp where "ENAME" = scott; 因为双引号提示oracle严格区分大小写，如下则不行  select \* from emp where "ename" = scott;  单引号在 Oracle 中有三种身份： 1.引用一个**字符串常量**，也就是界定一个字符串的开始和结束。 例：  select \* from table\_name where id=1123; -- 这种代表查询的是数字  select \* from table\_name where id='1123';-- 这种代表查询的是字符 2. 转义符，对紧随其后出现的字符(单引号)进行转义 3. 表示它本身，也就是它作为一个字符串的一部分而出现在一个字符串常量中，这点与2密不可分。 示例：  select '''' from dual; 分析：表达式 '''' 中第1、4个单引号表示引用一个字符串常量，表达式的剩余部分我们从左往右分析，由于中间是一对单引号，故此对中的第一个是转义符，其对第二个进行转义，使第二个表示单引号本身。结果是只有一个单引号字符。sql 语句的运行结果与我们的分析结果一致。 |

## 查询中的排序

### 排序语句

SELECT <字段或表达式…>

FROM <表名>

ORDER BY <列名1> [ASC|DESC],[列名2] [ASC|DESC],...;

例1: 查询每个员工的id,dept\_id,title,按照部门编号降序排序

|  |
| --- |
| select id "编号", dept\_id "部门编号", title "职位"  from s\_emp  order by "部门编号" desc;  select id "编号", dept\_id "部门编号", title "职位"  from s\_emp  order by dept\_id desc; |

例2: 查询每个员工的id,dept\_id,title,按照部门编号降序,如果部门编号相等,按照职位升序。

|  |
| --- |
| select id "编号", dept\_id "部门编号", title "职位"  from s\_emp  order by dept\_id desc, title asc; |

### 排序中的空值问题：

当排序时，如果排序的列存在空值，就会产生问题。我们可以用nulls first， nulls last来指定null值显示的位置。

例3： 查询员工的提成，按照提成升序排列，没有提成的都置后。

|  |
| --- |
| SELECT id, first\_name, commission\_pct  FROM s\_emp  ORDER BY commission\_pct nulls last; |

## 条件查询

### 使用where语句对结果进行过滤。

使用where的条件查询语句：

SELECT …

FROM …

WHERE <条件表达式>

ORDER BY …;

执行流程 :

FROM,确定表 --> WHERE,确定行,行筛选 --> SELECT,筛选列,结果集确定 --> ORDER BY,对结果集进行排序。

位置:

跟在from之后

例1：查询所有薪资高于1000的员工(id,salary)。

|  |
| --- |
| SELECT id,salary  FROM s\_emp  WHERE salary>1000; |

例2：查询41号部门的所有员工(id,dept\_id,salary),按照薪资升序排序。

|  |
| --- |
| SELECT id,dept\_id,salary  FROM s\_emp  WHERE dept\_id=41  ORDER BY salary ASC; |

### 比较运算符





例1：查询薪资在1100到1550之间的所有员工。

|  |
| --- |
| SELECT id,salary  FROM s\_emp  WHERE salary BETWEEN 1100 AND 1550; |

例2：查询部门编号为41,44,45,50的所有员工。

|  |
| --- |
| SELECT id,dept\_id  FROM s\_emp  WHERE dept\_id IN(41,44,45,50); |

例3：查询所有没有提成的员工。

|  |
| --- |
| SELECT id,commission\_pct  FROM s\_emp  WHERE commission\_pct IS NULL; |

例4：查找姓Smith的员工。

|  |
| --- |
| SELECT id, last\_name  FROM s\_emp  WHERE last\_name= 'Smith'; |

注意：Oracle中，查询条件中的查询条件的值是分大小写的。

Like模糊查询：

两个通配符

\_:代表任意一个字符

%:代表任意多个(0,1,多个)字符

例如：'%java%':包含java的字符串。

例5：查找以字母C开头的员工名。

|  |
| --- |
| SELECT first\_name  FROM s\_emp  WHERE first\_name LIKE 'C%'; |

例6：查找以y结尾的员工名。

|  |
| --- |
| SELECT first\_name  FROM s\_emp  WHERE first\_name LIKE '%y'; |

例7：查找第二个字母是e的员工名。

|  |
| --- |
| SELECT first\_name  FROM s\_emp  WHERE first\_name LIKE '\_e%'; |

例8：查找长度至少是4的员工名。

|  |
| --- |
| SELECT first\_name  FROM s\_emp  WHERE first\_name LIKE '\_\_\_\_%'; |

如果需要通配符当做普通字符进行模糊匹配，需要转义，通过ESCAPE实现。

例9: 查询以\_开头的姓。(了解即可)

|  |
| --- |
| SELECT last\_name  FROM s\_emp  WHERE last\_name LIKE '\\_%' ESCAPE '\'; |

例10：查找所有1991年入职的员工。

|  |
| --- |
| SELECT id, first\_name, start\_date  FROM s\_emp  WHERE start\_date BETWEEN '01-JAN-91' AND '31-DEC-91'; |

**比较运算补充：**

**NOT BETWEEN...AND... -- 不在闭区间**

**NOT IN -- 不在值列表中**

**IS NOT NULL -- 不是空**

**NOT LIKE -- 不以匹配的**

### 逻辑运算

NOT : 非

AND : 逻辑 与，并且

OR : 逻辑 或，或者

三者的运算优先级： NOT > AND > OR

例1：查询薪资在1100到1550之间的所有员工，使用逻辑运算。

|  |
| --- |
| SELECT id,salary  FROM s\_emp  WHERE salary>=1100 AND salary<=1550; |

例2：查询包含字母e的名字(不区分大小写)。

|  |
| --- |
| SELECT first\_name  FROM s\_emp  WHERE first\_name LIKE '%e%'  OR first\_name LIKE '%E%'; |

例3：查询在41号或者44号部门,并且薪资高于1000的员工。

|  |
| --- |
| SELECT id,dept\_id,salary  FROM s\_emp  WHERE dept\_id=41 OR dept\_id=44 AND salary>1000;  SELECT id,dept\_id,salary  FROM s\_emp  WHERE (dept\_id=41 OR dept\_id=44)  AND salary>1000; |

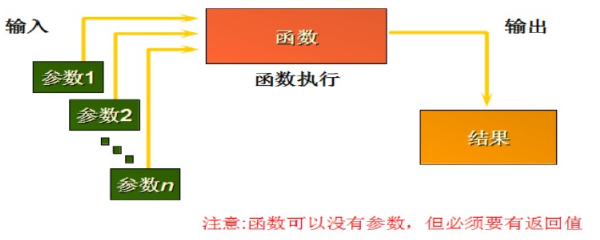
# 单行函数

## 什么是SQL函数？

SQL 拥有很多可用于计数和计算的内建函数。

内建函数的语法如下：

|  |
| --- |
| SELECT function(列) FROM 表 |

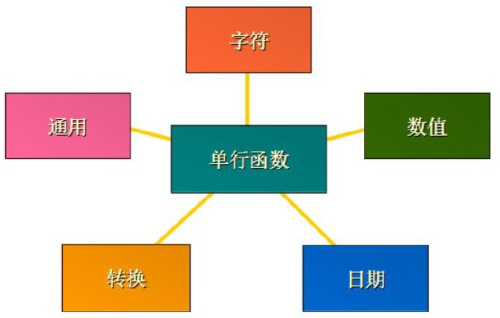


## 函数的类型

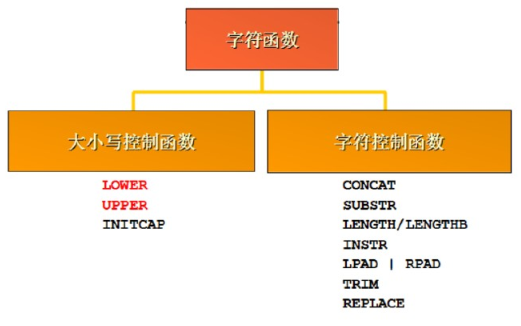
单行函数(Scalar 函数):一行得到一个结果

多行函数(Aggregate 函数):多行(一组)得到一个结果

## 单行函数的分类



### 字符函数



例1： 字符串的连接，可以使用concat函数，也可以使用||，推荐使用||。

连接’hello’和’world’。

|  |
| --- |
| SELECT 'hello '|| 'world ' FROM dual;  SELECT concat('hello', 'world') FROM dual; |

例2：字符串的截取substr函数。第一个参数，源字符串，第二个参数，开始的索引，第三个参数是子串长度。注意：开始的索引，使用0和1是一样的。

截取’Hello’的前三个字符。

|  |
| --- |
| SELECT substr('Hello',1,3) FROM dual; |

例3：获取字符串的长度，length函数。

获取字符串’Hello’的长度。

|  |
| --- |
| SELECT length('Hello') FROM dual; |

例4：字符串替换函数replace，第一个参数是源字符串，第二个参数是被替换的字符串，第三个参数是替换字符串。

把字符串’Hello’中的’l’替换成’x’。

|  |
| --- |
| SELECT replace('Hello', 'l', 'x') FROM dual; |

例5：在员工表中查询包含字母e的名字(不区分大小写)。

|  |
| --- |
| SELECT first\_name  FROM s\_emp  WHERE LOWER(first\_name) LIKE '%e%'; |

### 数值函数

ROUND() 四舍五入

TRUNC() 截取

MOD() 取余

CEIL() 天花板数,向上取整

FLOOR() 地板数,向下取整

例1：请写出下面语句的输出：

|  |
| --- |
| SELECT ROUND(345.926) FROM dual; -- 精确到个位  SELECT ROUND(345.926, 2) FROM dual; -- 精确到小数点后两位  SELECT ROUND(345.926, 0) FROM dual; -- 精确到个位  SELECT ROUND(345.926, -2) FROM dual; -- 精确到百位  SELECT TRUNC(345.926) FROM dual; -- 345  SELECT TRUNC(345.926, 2) FROM dual; -- 345.92  SELECT TRUNC(345.926, 0) FROM dual; -- 345  SELECT TRUNC(345.926, -2) FROM dual; -- 300  SELECT MOD(1600, 300) FROM dual;  SELECT MOD(11,3) FROM dual;  SELECT CEIL(3.2) FROM dual;  SELECT FLOOR(3.2) FROM dual; |

### 日期函数

ORACLE中的日期型数据实际包含两个值：日期和时间。

默认的日期格式是： DD-MON-YY。

日期的数学运算，是指在日期上加上或减去一个数字，结果仍然是日期。

两个日期相减，返回的是两个日期相差的天数。

MONTHS\_BETWEEN(date1,date2)：两个时间的月份差,包含小数。

ADD\_MONTHS(date,n) ： 增加月份

增加年:ADD\_MONTHS(date,n\*12)

增加天: sysdate+n

增加小时: sysdate+n/24

NEXT\_DAY(date,< '星期'>):距离date最近的下一个星期几

LAST\_DAY(date):date所在月份的最后一天

ROUND(date,'精确度')

精确度:

年:year,yyyy,看月份,满7进1

月:month,mm,看日,满16进1

日:dd,看时,满12进1

TRUNC(date,'截取的部分')

例1：2018年7月1日到现在经历了多少个月？

注意如下两个查询语句的区别：

|  |
| --- |
| SELECT FLOOR(MONTHS\_BETWEEN(sysdate,'01-7月-18')) FROM dual;  SELECT FLOOR(MONTHS\_BETWEEN(sysdate,'01-JUL-18')) FROM dual; |

例2：3个月后的今天？2天后的今天？

|  |
| --- |
| SELECT ADD\_MONTHS(sysdate,3) FROM dual;  SELECT sysdate+2 FROM dual; |

例3：下一个星期六的日期是？

|  |
| --- |
| SELECT NEXT\_DAY(sysdate,'星期六') FROM dual;  SELECT NEXT\_DAY(sysdate,'Saturday') FROM dual;  SELECT NEXT\_DAY(sysdate,'Sat') FROM dual; |

例4： 本月最后一天的日期是？

|  |
| --- |
| SELECT LAST\_DAY(sysdate) FROM dual; |

例5：下列语句的输出是什么？

|  |
| --- |
| SELECT ROUND(sysdate,'year') FROM dual;  SELECT ROUND(sysdate,'mm') FROM dual;  SELECT ROUND(sysdate,'dd') FROM dual; |

例6：今年的起始日期？

|  |
| --- |
| SELECT TRUNC(sysdate,'yyyy') FROM dual; |

例7: 本周的星期一日期是？

|  |
| --- |
| SELECT NEXT\_DAY(sysdate,'Mon')-7 FROM dual; |

例8：在员工表中查询员工进入公司的周数。

分析：查询员工进入公司的天数/7就是周数。

员工进入公司的天数 = sysdate – start\_date

|  |
| --- |
| SELECT id, round((sysdate-start\_date)/7) weeks FROM s\_emp; |

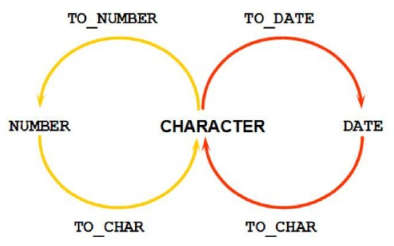
例9：查询所有员工进入公司的月数。

|  |
| --- |
| SELECT id, round(months\_between(sysdate, start\_date)) months FROM s\_emp; |

**TIPS – 常见的时间加减法**

|  |
| --- |
| 加法  select sysdate,add\_months(sysdate,12) from dual; --加1年  select sysdate,add\_months(sysdate,1) from dual; --加1月  select sysdate,to\_char(sysdate+7,'yyyy-mm-dd HH24:MI:SS') from dual; --加1星期  select sysdate,to\_char(sysdate+1,'yyyy-mm-dd HH24:MI:SS') from dual; --加1天  select sysdate,to\_char(sysdate+1/24,'yyyy-mm-dd HH24:MI:SS') from dual; --加1小时  select sysdate,to\_char(sysdate+1/24/60,'yyyy-mm-dd HH24:MI:SS') from dual; --加1分钟  select sysdate,to\_char(sysdate+1/24/60/60,'yyyy-mm-dd HH24:MI:SS') from dual; --加1秒  减法  select sysdate,add\_months(sysdate,-12) from dual; --减1年  select sysdate,add\_months(sysdate,-1) from dual; --减1月  select sysdate,to\_char(sysdate-7,'yyyy-mm-dd HH24:MI:SS') from dual; --减1星期  select sysdate,to\_char(sysdate-1,'yyyy-mm-dd HH24:MI:SS') from dual; --减1天  select sysdate,to\_char(sysdate-1/24,'yyyy-mm-dd HH24:MI:SS') from dual; --减1小时  select sysdate,to\_char(sysdate-1/24/60,'yyyy-mm-dd HH24:MI:SS') from dual; --减1分钟  select sysdate,to\_char(sysdate-1/24/60/60,'yyyy-mm-dd HH24:MI:SS') from dual; --减1秒 |

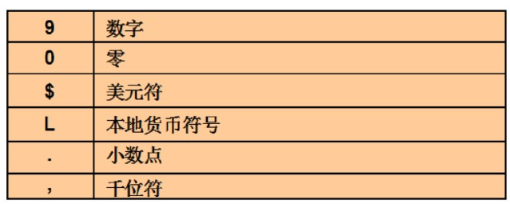
### 转换函数



#### 数值和字符串转换:

**函数说明： to\_char(数值,'格式')**

格式：



SELECT to\_char(1234567.892,'00,000,000.0000') FROM dual;

说明： 模式用0占位，不足的位用0补充。

SELECT to\_char(1234567.892,'99,999,999.9999') FROM dual;

说明：模式用9占位，整数位不足的空着,小数位不足的位用0补充。

增加货币单位:

SELECT to\_char(1234567.892,'$99,999,999.9999') FROM dual;

SELECT to\_char(1234567.892,'L99,999,999.9999') FROM dual;

**to\_number(字符串[,'格式']): 字符串-->数值**

SELECT to\_number('10')+to\_number('10') FROM dual;

#### 时间和字符串转换

to\_char(date,'模式'):把时间转换成字符串

to\_date(字符串,'模式'):把字符串转换成时间

模式符号:

YYYY:四位数字的年

YEAR:英文的年

MM:月

DD:日

HH24:24进制的时

HH:十二进制的时

MI:分

SS:秒

两位的年,尽量不要使用,容易误导

RR:两位的年,0~49表示本世纪,50~99表示上世纪

YY:两位的年,都表示本世纪

把当前时间按照年月日，时分秒的方式显示：

SELECT to\_char(sysdate,'yyyy-mm-dd hh24:mi:ss') FROM dual;

可以很方便的获取某一个时间信息:比如年

SELECT to\_char(sysdate,'yyyy') FROM dual;

可以方便的把一个时间格式字符串转换成时间,这样要表达一个时间就不受系统格式限制了:

|  |
| --- |
| SELECT to\_date('2018-08-01','yyyy-mm-dd') FROM dual;  SELECT to\_char(to\_date('18-08-01','yy-mm-dd'),'yyyy') FROM dual;  SELECT to\_char(to\_date('18-08-01','rr-mm-dd'),'yyyy') FROM dual;  SELECT to\_char(to\_date('89-08-01','yy-mm-dd'),'yyyy') FROM dual;  SELECT to\_char(to\_date('89-08-01','rr-mm-dd'),'yyyy') FROM dual; |

### 条件表达式

实现IF-THEN-ELSE的逻辑。

问题的提出：在前面的员工表中，我们需要依据员工的薪资计算税率。如果税率按照如下方式计算：

不超过800的，税率为0。

800 – 900, 税率为0.09。

900 - 1000，税率为0.20，

1000 - 1100，税率为0.30，

1100 – 1200，税率0.40，

1200 – 1300，税率0.42，

1300 – 1400，税率0.44，

其他，税率0.45。

现在需要查询每个员工的薪资，以及各自的税率。（不用考虑提成）

思路：

1. 可以考虑新增一张税率表。相对较复杂点。如果使用的频率不高，那么就需要额外考虑。
2. 可以采用条件表达式IF-THEN-ELSE来实现。

在ORACLE中实现IF-THEN-ELSE条件表达式，可以有两种方式：一种是SQL通用的CASE表达式，另外一种是ORACLE特有的DECODE函数。

#### CASE表达式

语法1：

**case 字段   
when 条件1 then 表达式1   
when 条件2 then 表达式2   
else 表达式n**

语法2：

**CASE WHEN 条件 THEN 表达式1 ELSE 表达式2 END;**

例1：运行下列条件表达式，看看最后的输出是什么？

|  |
| --- |
| SELECT  CASE 1 WHEN 1 THEN 'one'  WHEN 2 THEN 'two'  ELSE 'more'  END result  FROM dual; |

|  |
| --- |
| SELECT  CASE  WHEN 1>0 THEN 'true'  ELSE 'false'  END result  FROM dual; |

解答 员工税率 的问题：

|  |
| --- |
| SELECT id, salary,  CASE CEIL((salary-800)/100)  WHEN 0 THEN 0  WHEN 1 THEN 0.09  WHEN 2 THEN 0.20  WHEN 3 THEN 0.30  WHEN 4 THEN 0.40  WHEN 5 THEN 0.42  WHEN 6 THEN 0.44  ELSE 0.45  END tax\_rate  FROM s\_emp; |

#### DECODE函数

语法:

**DECODE(字段或字段表达式,**

**值1, 表达式1,**

**值2, 表达式2,**

**……**

**值n, 表达式n)**

例2：如果员工薪资高于1250，那么标记为高薪，否则标记为低薪。

|  |
| --- |
| SELECT id, salary,  decode(sign(salary-1250), 1, '高薪', 0, '低薪', -1, '低薪') grade  FROM s\_emp;  SELECT id, salary,  decode(sign(salary-1250), 1, '高薪','低薪') grade  FROM s\_emp; |

使用decode解答 员工税率 的问题：

|  |
| --- |
| SELECT id, salary,  decode(CEIL((salary-800)/100),  0,0,  1,0.09,  2,0.20,  3,0.30,  4,0.40,  5,0.42,  6,0.44,  0.45) tax\_rate  FROM s\_emp; |

# 多表查询

## 什么是笛卡尔积？

问题的提出：查询每个员工名，员工所在的部门编号和部门名。

问题所在：数据来源与多张表（s\_emp, s\_dept），表存在关联关系。

是否能够多张表查询？

多张表查询，基本语法：

SELECT <列>…<列>

FROM <表1>, <表2>…

WHERE 条件

注意：列可能来源于多张表。

利用基本语法解决提出的问题，编写如下查询语句：

SELECT s\_emp.id, s\_dept.id, s\_dept.name

FROM s\_emp, s\_dept;

注意：上述SQL语句中，两张表都有id，需要用哪张表的id，需要用表名来限定。

上述查询的结果，是表s\_emp中的数据条数\*s\_dept的数据条数。

结果集 = 表1记录数 \* 表2记录数

这就是笛卡尔积。

用昨天的学生表，科目表和成绩表来做更进一步的说明。

数据准备：

1. 在学生表中，增加3个学生。(t\_student)
2. 在科目表中，增加3个科目。(t\_subject)
3. 在成绩表，给每个学生的每个科目准备一个成绩。(t\_score)

基于上述三张表，查询学生id，学生姓名，学习的科目id，科目名称和对应的成绩。

SELECT t\_student.id, t\_student.name, t\_subject.id, t\_subject.name, t\_score.score

FROM t\_student, t\_subject, t\_score;

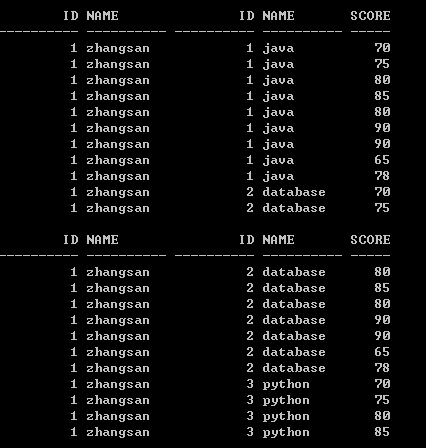
查询结果81行。是三张表的笛卡尔积：3\*3\*9

查询结果中，显然存在大量的冗余，不正确结果。部分结果集显示如下图。对于学生”zhangsan”的科目1 “java”，显然只有一个成绩，而不是结果集中的”学生1的科目1有9个成绩”。

## 多表关联查询

想要避免结果集是笛卡尔集，需要多表关联查询，即对结果集根据实际情况增加查询条件。

仍然以前面的查询为了，学生1的科目1的成绩只能确定一个，确定这个成绩的依据是：成绩表中的学生id，和学生表中的id要相等，这样说明这个成绩是对应于哪个学生的。成绩表中的科目id，要等于科目表中的id，这样表名这个成绩是属于哪个科目的。



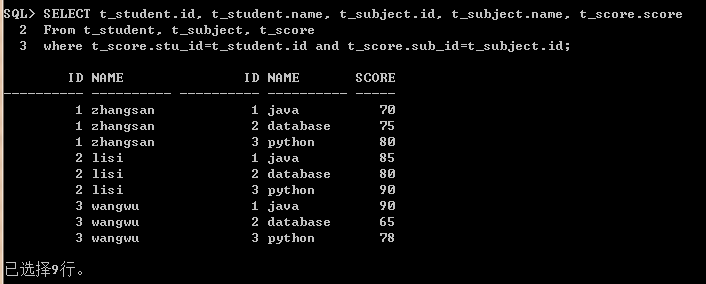
多表关联查询：

SELECT t\_student.id, t\_student.name, t\_subject.id, t\_subject.name, t\_score.score

FROM t\_student, t\_subject, t\_score

WHERE t\_score.stu\_id=t\_student.id AND t\_score.sub\_id=t\_subject.id;

查询结果如下：



多表关联查询，我们可以为每张表取一个别名，这样是SQL语句更简洁：

|  |
| --- |
| SELECT t.id, t.name, u.id, u.name, c.score  FROM t\_student t, t\_subject u, t\_score c  WHERE c.stu\_id=t.id AND c.sub\_id=u.id; |

**总结：**

|  |
| --- |
| **两张以上的表关联查询，需要增加筛选条件以避免笛卡尔积。**  **通常：外键和关联的列相等**  **两张表需要1个筛选条件**  **三张表需要2个筛选条件**  **如果出现重名的列，需要做区分， 通过表名来区分。**  **在多表联查时，建议给每个字段都加前缀。**  **表名太长，可以取别名来代替表名。**  **一旦取了别名，就只能使用别名。** |

切换用户oaec/oace：

例1：利用前面的员工表，部门表，查询每个员工id,部门编号，所在的部门名。

|  |
| --- |
| SELECT s\_emp.id,dept\_id,name  FROM s\_emp,s\_dept  WHERE s\_emp.dept\_id=s\_dept.id;  SELECT e.id,e.dept\_id,d.name  FROM s\_emp e,s\_dept d  WHERE e.dept\_id=d.id; |

例2：查询每个部门名字，及所在的区域名。

|  |
| --- |
| SELECT d.name, r.name  FROM s\_dept d, s\_region r  WHERE d.region\_id=r.id; |

例3：查询每个客户名，及对应的销售名。

|  |
| --- |
| SELECT c.name, e.last\_name||', '||e.first\_name saler  FROM s\_customer c, s\_emp e  WHERE c.sales\_rep\_id = e.id; |

例4：查询每个员工名，及所在的部门名，及所在的区域名。

|  |
| --- |
| SELECT e.last\_name||', '||e.first\_name employee, d.name dept, r.name region  FROM s\_emp e, s\_dept d, s\_region r  WHERE e.dept\_id=d.id and d.region\_id=r.id; |

## 多表查询和条件查询

用前面的学生/科目/成绩表，来练习如下需求：

练习：查询所有学生姓名 – 科目名 – 成绩。成绩我们采用ABCD制：

成绩低于60，成绩等级为E。

成绩不超过70，成绩等级为D。

成绩不超过80，成绩等级为E。

成绩不超过90，成绩等级为B。

其他，成绩等级为A。

|  |
| --- |
| select t.name student, u.name subject,c.score,  (case  when c.score<60 then 'E'  when c.score<70 then 'D'  when c.score<80 then 'C'  when c.score<90 then 'B'  else 'A'  end) grade  from t\_student t, t\_subject u, t\_score c  where c.stu\_id=t.id and c.sub\_id=u.id;  select t.name student, u.name subject,c.score,  decode(sign(c.score-90),1,'A',0,'A',-1,  decode(sign(c.score-80),1,'B',0,'B',-1,  decode(sign(c.score-70),1,'C',0,'C',-1,  decode(sign(c.score-60),1,'D',0,'D',-1,'E')))) grade  from t\_student t, t\_subject u, t\_score c  where c.stu\_id=t.id and c.sub\_id=u.id; |

## 多表查询连接方式

等价连接：筛选条件使用=

不等价连接：筛选条件不使用=

例1：查询员工编号，员工姓名，员工所在部门编号，员工所在部门名称，部门所在区域编号，区域名称。

分析：需要查询的表有员工表s\_emp, 部门表s\_dept，区域表s\_region

这三张表的关联：员工表中的部门id=部门表的id，部门表的region\_id=区域表的id。

|  |
| --- |
| SELECT e.id eid, e.first\_name||' '||e.last\_name name,  d.id did, d.name department,  r.id rid, r.name region  FROM s\_emp e, s\_dept d, s\_region r  WHERE e.dept\_id = d.id and d.region\_id=r.id; |

表的自身关联：同一张表中，字段产生关联。

例2：查询每个员工的编号，姓名，以及他的领导的员工编号，姓名。

分析：员工表中，有个字段manager\_id，记录的是这个员工的领导的员工id。

最顶级的员工的manager\_id的字段为空，因为他就是最大的领导了。

查询的表：s\_emp，这张表中产生了自身关联，及manager\_id关联到了id。

|  |
| --- |
| SELECT e1.id eid, e1.first\_name||' '||e1.last\_name name,  e2.id lid, e2.first\_name||' '||e2.last\_name name  FROM s\_emp e1, s\_emp e2  WHERE e1.manager\_id = e2.id; |

不等价的例子，先准备相关的数据。创建一张新表，记录薪资的等级：

|  |
| --- |
| CREATE TABLE s\_salgrade(  losal number(6),  hisal number(6),  grade char(1)  ); |

在表中增加工资等级数据：

|  |
| --- |
| INSERT INTO s\_salgrade VALUES(0,999,'D');  INSERT INTO s\_salgrade VALUES(1000,1499,'C');  INSERT INTO s\_salgrade VALUES(1500,1999,'B');  INSERT INTO s\_salgrade VALUES(2000,2500,'A');  COMMIT; |

例3：查询每个员工的编号，薪资和薪资等级。

|  |
| --- |
| SELECT e.id eid, e.salary salary, g.grade grade  FROM s\_emp e, s\_salgrade g  WHERE e.salary BETWEEN g.losal AND g.hisal; |

例4：查看每个员工编号，姓名，部门名称，薪资，薪资等级，他的上级的编号，名称，薪资，薪资等级。薪资等级不用ABCD，而是转换成“一级”，“二级”，“三级”，“四级”。

分析：查询的表有：s\_emp, s\_dept, s\_salgrade

取得的薪资等级，通过条件查询转换，建议使用decode函数。

|  |
| --- |
| SELECT e1.id eid, e1.first\_name||' '||e1.last\_name ename,  d.name dname,  e1.salary, decode(g1.grade,  'A','一级',  'B','二级',  'C','三级',  'D','四级',  '五级') sgrade,  e2.id lid, e2.first\_name||' '||e2.last\_name lname,  e2.salary, decode(g2.grade,  'A','一级',  'B','二级',  'C','三级',  'D','四级',  '五级') lgrade  FROM s\_emp e1, s\_emp e2, s\_dept d, s\_salgrade g1, s\_salgrade g2  WHERE e1.manager\_id=e2.id AND  e1.dept\_id=d.id AND  e1.salary BETWEEN g1.losal AND g1.hisal AND  e2.salary BETWEEN g2.losal AND g2.hisal; |

例5：查询入职时间早于其领导的员工。

|  |
| --- |
| select e1.id eid, e1.first\_name||' '||e1.last\_name employee, e1.start\_date,  e2.id lid, e2.first\_name||' '||e2.last\_name leader, e2.start\_date  from s\_emp e1, s\_emp e2  where e1.manager\_id=e2.id  and e1.start\_date<e2.start\_date; |

## 外连接

SQL提供了多种类型的连接方式，它们之间的区别在于：从相互交叠的不同数据集合中选择用于连接的行时所采用的方法不同。

|  |  |
| --- | --- |
| 连接类型 | 定义 |
| 内连接 | 只连接匹配的行 |
| 左外连接 | 包含左边表的全部行（不管右边的表中是否存在与它们匹配的行），以及右边表中全部匹配的行 |
| 右外连接 | 包含右边表的全部行（不管左边的表中是否存在与它们匹配的行），以及左边表中全部匹配的行 |
| 全外连接 | 包含左、右两个表的全部行，不管另外一边的表中是否存在与它们匹配的行。 |

准备三张简单的表：

表TESTA，两个字段A，B

|  |  |
| --- | --- |
| A | B |
| 001 | 10A |
| 002 | 20A |

表TESTB，两个字段A，B

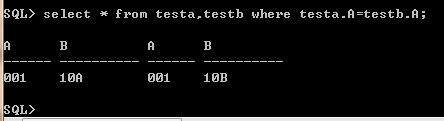
|  |  |
| --- | --- |
| A | B |
| 001 | 10B |
| 003 | 30B |

表TESTC，两个字段A，B

|  |  |
| --- | --- |
| A | B |
| 001 | 10C |
| 004 | 40C |

**内连接：**

|  |
| --- |
| select \* from testa,testb where testa.A=testb.A; |



**左外连接:**

左外连接就是在等值连接的基础上加上主表中的未匹配数据。

|  |
| --- |
| select \*  from testa  **left outer join** testb  **on** testa.A=testb.A; |

|  |
| --- |
| select \*  from testa  **left join** testb  **on** testa.A=testb.A; |

**ORACLE特有的：**

|  |
| --- |
| select \*  from testa,testb  where testa.A=testb.A**(+)**; |



**对于(+)的语法：**

select \*

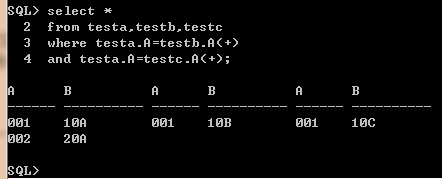
from testa,testb

where testa.A=testb.A(+); --(+)在谁的后面，决定了以谁的数据查询为补充。

**三张表做左外连接：**

|  |
| --- |
| select \*  from testa  **left outer join** testb  **on** testa.A=testb.A  **left outer join** testc  **on** testa.A=testc.A; |

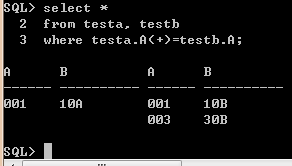
|  |
| --- |
| select \*  from testa,testb,testc  where testa.A=testb.A(+)  and testa.A=testc.A(+); |



**右连接：**

右外连接是在等值连接的基础上加上被连接表的不匹配数据。

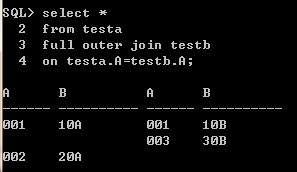
|  |
| --- |
| select \*  from testa  **right outer join** testb  **on** testa.A=testb.A;  select \*  from testa, testb  where testa.A**(+)**=testb.A; |



**全外连接：**

全外连接是在等值连接的基础上将左表和右表的未匹配数据都加上。

|  |
| --- |
| select \*  from testa  **full outer join** testb  **on** testa.A=testb.A; |



例1：查询每个客户及对应的销售。

分析：需要查询的表有s\_customer, s\_emp

要求每个客户，所有需要把所有的客户都显示，以客户表为主查询，连接员工表。

左外连接查询：

|  |
| --- |
| select c.name name, e.first\_name||' '||e.last\_name employee  from s\_customer c  left join s\_emp e  on c.sales\_rep\_id=e.id;  select c.name name, e.first\_name||' '||e.last\_name employee  from s\_customer c, s\_emp e  where c.sales\_rep\_id=e.id(+); |

右外连接查询：

|  |
| --- |
| select c.name name, e.first\_name||' '||e.last\_name employee  from s\_emp e  right join s\_customer c  on c.sales\_rep\_id=e.id;  select c.name name, e.first\_name||' '||e.last\_name employee  from s\_customer c, s\_emp e  where e.id(+)=c.sales\_rep\_id; |

例2：查询每个部门名以及对应的区域名。

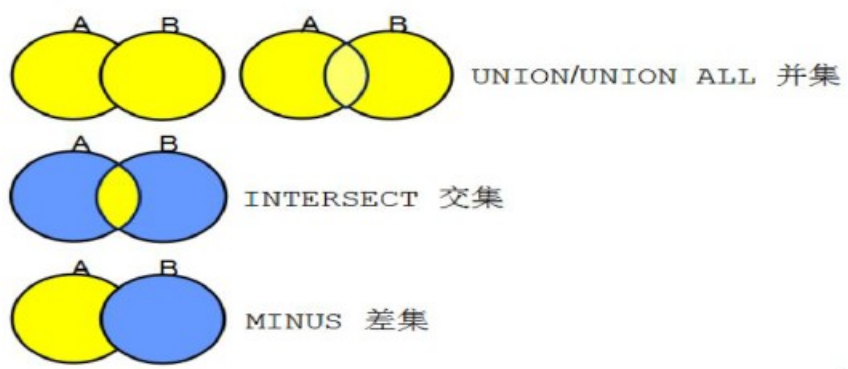
|  |
| --- |
| select d.name department, r.name region  from s\_dept d  left join s\_region r  on d.region\_id=r.id;  select d.name department, r.name region  from s\_dept d, s\_region\_id  where d.region\_id=r.id(+); |

例3：查询每个员工的名字，以及对应的领导的名字。

|  |
| --- |
| select e1.id eid, e1.first\_name employee, e2.id lid, e2.first\_name leader  from s\_emp e1  left join s\_emp e2  on e1.manager\_id=e2.id;  select e1.id eid, e1.first\_name employee, e2.id lid, e2.first\_name leader  from s\_emp e2  right join s\_emp e1  on e1.manager\_id=e2.id; |

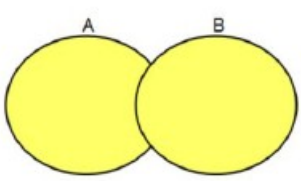
# 集合运算

## 什么是集合运算？

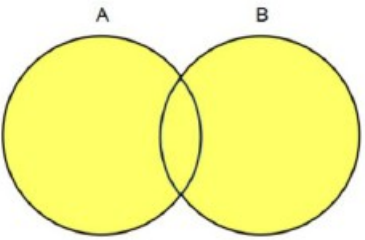


## 并集UNION

UNION运算符返回两个集合去掉重复元素后的所有记录。



UNION ALL返回两个集合的所有元素，包括重复的。



查询所有员工编号大于10的员工。

|  |
| --- |
| SELECT first\_name,id  FROM s\_emp  WHERE id>10; |

查询所有员工编号小于20的员工

|  |
| --- |
| SELECT first\_name,id  FROM s\_emp  WHERE id<20; |

查询所有员工编号大于10**或者**小于20的员工。

|  |
| --- |
| SELECT first\_name,id  FROM s\_emp  WHERE id>10  UNION  SELECT first\_name,id  FROM s\_emp  WHERE id<20; |

如果是UNION ALL，对比前面的结果：

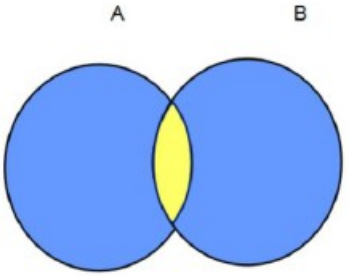
|  |
| --- |
| SELECT first\_name,id  FROM s\_emp  WHERE id>10  UNION ALL  SELECT first\_name,id  FROM s\_emp  WHERE id<20; |

例1：查询所有工资大于1000，**或者**是42号部门的员工。

|  |
| --- |
| select id, salary, dept\_id  from s\_emp  where salary > 1000  UNION  select id, salary, dept\_id  from s\_emp  where dept\_id = 42; |

## 交集INTERSECT

交集INTERSECT运算符返回同时属于两个集合的记录。



例2：查询所有员工编号大于10并且小于20的员工。

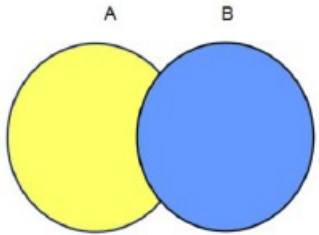
|  |
| --- |
| SELECT first\_name,id  FROM s\_emp  WHERE id>10  INTERSECT  SELECT first\_name,id  FROM s\_emp  WHERE id<20; |

例3：查询所有工资大于1000，并且是42号部门的员工。

|  |  |
| --- | --- |
| select id, salary, dept\_id  from s\_emp  where salary > 1000  INTERSECT  select id, salary, dept\_id  from s\_emp  where dept\_id = 42; |  |

## 差集MINUS

差集MINUS返回属于第一个集合，但不属于第二个集合的记录。



例4：下面的语句的输出结果是什么？

|  |
| --- |
| SELECT first\_name,id  FROM s\_emp  WHERE id>10 --11~25  MINUS  SELECT first\_name,id  FROM s\_emp  WHERE id<20;  select id, salary, dept\_id  from s\_emp  where salary > 1000  MINUS  select id, salary, dept\_id  from s\_emp  where dept\_id = 42; |

# ROWNUM

ROWNUM : 表示行号，是一个“伪列”，代表结果集中的行号。

ROWNUM行号，从1开始的连续整数。

ROWNUM可以如同普通的列一样使用，是关键字。

ROWNUM作用: 可以实现分页。

注意下面语句中的rownum：

|  |
| --- |
| SELECT id,first\_name,rownum  FROM s\_emp;  SELECT id,first\_name,rownum  FROM s\_emp  WHERE id>10; |

例1：查询员工表中的前10行，查询员工编号，名字。

|  |
| --- |
| SELECT id,first\_name,rownum  FROM s\_emp  WHERE rownum<11; |

例1：查询员工表中的第11行~20行，查询员工编号，名字。

|  |
| --- |
| SELECT id,first\_name,rownum  FROM s\_emp  WHERE rownum<=20  MINUS  SELECT id,first\_name,rownum  FROM s\_emp  WHERE rownum<=10; |

想想：上面语句能否使用between…and…，或者使用>=? 为什么？

例2：查询前10人的id，名字，薪资，rownum，并按照薪资升序排列。

|  |
| --- |
| SELECT id,first\_name,salary,rownum  FROM s\_emp  WHERE rownum<=10  ORDER BY salary; |

注意：FROM-WHERE-SELECT，结果集确定，行号就确定，ORDER BY--调整顺序，行号不变。