第5章 数据查询

数据查询是数据库中使用的最多的操作，数据查询是普通用户访问数据库中所保存资料的主要途径。数据查询可以从数据库中检索出满足条件的数据记录，是数据库应用中最常用的操作，数据查询使用 SELECT 语句完成，该语句功能强大、使用灵活。每当我们上淘宝、京东、支付宝等网络平台时，我们就完成了多个查询操作，数据查询和人们的生活息息相关。

本章主要内容

- 数据查询语句 SELECT
- Oracle 数据库中常用的内置 SQL 函数
- SQL * Plus 查询输出结果格式化
- SQL 脚本文件的创建与执行
- 多表连接查询、子查询、集合查询
- ORACLE 数据表伪列的应用(ROWID,ROWNUM,LEVEL)

5.1 数据查询语句 SELECT

数据查询语句 SELECT 在 Oracle 数据库中是应用频度最高的语句之一。SELECT 语句的作用是让数据库服务器根据客户的要求从数据库中检索出所需要的信息，并且可以按规定的格式进行分类、统计、排序，再把结果回馈给客户。除此之外还可以用 SELECT 语句设置和显示系统信息，为局部变量赋值等。

SELECT 语句具有强大的查询功能，完整的语法非常复杂，掌握了 SELECT 语句就可以轻松地利用数据库来完成自己的工作。

```
SELECT [ ALL|DISTINCT ] select_list
FROM ]schema.]table_name | [ schema.]view_name
[WHERE search_condition ]
[GROUP BY group_by_expression [HAVING search_condition ]]
[ORDER BY order_expression [ ASC | DESC ]]
```

SELECT 语句的含义是：根据 WHERE 子句的条件表达式，从 FROM 子句所指定的表或视图中查找满足条件的记录，再按 select_list 所指定的查询列表项显示结果。还可以根据 GROUP BY 子句给出的分组表达式将查询结果进行分组。ORDER BY 子句的作用是将查询结果进行排序。在 Oracle 中，SELECT 语句必须包含 SELECT 和 FROM 子句，即使有些查询不需要表时，通常也要用 DUAL(DUAL 是一个小表，只有一行一列，数据库

安装完毕后,任何用户都可使用此表)表来补足语法;而其他子句可以根据查询的要求进行选择。下面先介绍 SELECT 子句和 FROM 子句。

5.1.1 SELECT 子句和 FROM 子句

SELECT 子句和 FROM 子句是 SELECT 语句的必选项,也就是说,每个 SELECT 语句都必须包含这两个子句。其语法格式如下:

```
SELECT [ALL|DISTINCT] select_list
FROM [schema.]table_name | [ schema.]view_name
```

其中各参数的意义如下:

- SELECT:用于查询的关键字。
- ALL | DISTINCT:ALL 表示筛选出表中满足条件的所有记录,一般情况下可省略;DISTINCT 表示从查询结果集中去掉重复的行。
- select_list:指定查询的字段,如果要查询所有的字段可以使用星号(*)代替。
- [schema.]table_name:指定查询的数据源的表名称和它的方案名,如果表是当前数据库连接用户方案下的表,则方案名可以省略。
- [schema.] view_name:指定查询的数据源的视图名称和它的方案名,方案名也可以省略。

1. 选择所有列(表中的全部列)

在 SELECT 子句中可以使用星号(*)显示表中所有的列。

例 5.1 以 scott 用户登录数据库,查询 emp 表中的所有列。

```
SELECT * FROM emp;
```

2. 指定部分列(业务操作关心的列)

指定列的语法格式如下:

```
SELECT column_name1 [, column_name2, ...]
FROM [ schema.]table_name| [ schema.]view_name
```

其中,column_name1 [, column_name2,…]是要查询的字段列表,中间用逗号隔开。

例 5.2 以 scott 用户登录数据库,查询 emp 表中每个雇员的 empno、ename、job 的值。

```
SELECT empno,ename ,j ob
FROM scott .emp;
```

要注意的是在数据查询时,列的显示顺序由 SELECT 子句指定,该顺序可以和列定义时顺序不同,这并不影响数据在表中的存储顺序;在查找多列内容时,用逗号将各字段分开。

3. 改变列标题(为列标题起别名)

在默认情况下,查询结果中显示的列标题就是在创建表时使用的字段名,用户可以根据要求在 SELECT 语句中改变列标题,语法格式如下:

```
SELECT column_name1   [ AS] alias, column_name2 [ AS] alias, ...
FROM [ schema.]table_name| [ schema.]view_name
```

其中，column_name 是要查询的字段名；AS 是为字段起别名的关键字，可以省略；alias 是为字段起的别名。

例 5.3 以 scott 用户登录数据库，查询 emp 表中每个雇员的 empno、ename、job 的值，并将结果中各列的标题指定为编号、姓名、工作。

```
SELECT empno AS 编号,ename AS 姓名,job AS 工作
FROM emp;
```

或

```
SELECT empno 编号,ename 姓名,job 工作
FROMemp;
```

要注意的是如果列标题中包含了一些特殊的字符，如空格等，就必须用双引号将列标题括起来，否则认为是非法字符或语法错误。

4. 使用计算列(计算域)

在进行数据查询时，经常需要对表中数据计算后才能得到满意的结果。在查询结果中可以输出对列计算后的值，即 SELECT 子句中可以使用表达式作为查询对象。可以使用各种运算符和函数对字段的值进行计算。函数包括普通函数和统计函数，分别在后面的章节介绍。而运算符通常使用算术运算符和字符串连接运算符(||)。算数运算符包括加(+)、减(-)、乘(*)、除(/)和取模(%)运算。

例 5.4 以 scott 用户登录数据库，查询 emp 表中每个雇员的姓名、工作和工资增加 300 元后的新工资。

```
SELECT ename ||'的工作是'||job AS 雇员,sal + 300 AS 新工资 FROM emp;
```

要注意的是，如果不为计算列指定列标题，系统将直接使用计算表达式作为列标题。对表中列的计算只是影响查询结果，并不改变表中的数据。

5. DISTINCT 关键字

使用 DISTINCT 关键字可以从结果集中消除重复的行，使结果更简洁。其语法格式如下：

```
SELECT DISTINCT column_name1 [, column_name2, ...]
FROM [ schema.]table_name| [ schema.]view_name
```

例 5.5 以 scott 用户登录数据库，查询 emp 表中的 job 和 deptno 字段，要求去除重复的行。

```
SELECT DISTINCT job,deptno FROM emp;
```

5.1.2 WHERE 子句

在实际工作中，大部分查询并不是针对表中所有记录进行查询，就像在淘宝上购买某类产品时全部商品列表几乎无法满足我们尽快找出所购商品的需求，而是要找出满足某些条件的记录，此时我们可以在 SELECT 语句中使用 WHERE 子句，目的是从表中筛选出符合条件的行，WHERE 子句必须紧跟在 FROM 子句之后，其语法格式如下：

```
SELECT [ ALL | DISTINCT ] select_list
FROM [schema.]table_name | [ schema.]view_name
WHERE search_condition
```


其中，search_condition 指定从表中查询记录的筛选条件。筛选条件是指由比较运算符、逻辑运算符、字符串模式匹配符、是否为空运算符等构成的表达式，该表达式的结果是逻辑值真或假。

在使用字符串和日期数据进行比较时，应符合下面的规定：

- 字符串和日期必须用单引号括起来。
- 字符串数据区分大小写。
- 日期数据的格式是敏感的，默认的日期格式是 DD-MON-YY。

1. 比较运算符

WHERE 子句允许使用的比较运算符包括以下几种：=（等于）、<（小于）、>（大于）、<=（小于等于）、>=（大于等于）、<>或!=（不等于）、!>（不大于）、!<（不小于）。

例 5.6 以 scott 用户登录数据库，查询 emp 表中工资大于 2000 的雇员信息。

```
SELECT * FROM emp
WHERE sal > 2000;
```

例 5.7 以 scott 用户登录数据库，查询 emp 表中工作为 SALESMAN 的雇员编号、姓名、工作信息。

```
SELECT empno ,ename, j ob
 FROM emp
WHERE job = 'SALESMAN';
```

注意：表中的字符串常量是区分大小写的，而表名、字段名和 SQL 命令不区分大小写。

2. 逻辑运算符

在 WHERE 子句中可以使用逻辑运算符把若干个查询条件连接起来，从而实现比较复杂的选择查询。可以使用的逻辑运算符包括逻辑与（AND）、逻辑或（OR）和逻辑非（NOT）。

其语法格式如下：

```
SELECT [ ALL | DISTINCT ] select_list
FROM [schema.]table_name | [ schema.]view_name
WHERE [NOT ] search_condition { AND | OR } [NOT] search_condition
```

例 5.8 以 scott 用户登录数据库，查询 emp 表中工资在 2000～3000 元之间的雇员记录。

```
SELECT empno, ename,job, sal FROM emp
WHERE sal > = 2000 AND sal < 3000;
```

例 5.9 以 scott 用户登录数据库，查询 emp 表中工作为 SALESMAN 或 CLERK 的雇员的编号、姓名、工作。

```
SELECT empno,ename, job FROM emp
WHERE job = 'SALESMAN' OR job = 'CLERK';
```

3. 字符串模式匹配符

在前面介绍的查询中，查询条件都是确定的。但在实际应用中，并不是所有的查询条件都是确定的。例如，要查询公司中一个姓张的销售人员，但不知道叫什么名字，此时，精确查询就不管用了，必须使用 LIKE 关键字进行模糊查询。其语法格式如下：

```
SELECT [ ALL | DISTINCT] select_list
FROM [schema.]table_name | [ schema.]view_name
WHERE expression [NOT] LIKE. 'string'
```

其中，string 是匹配字符串，其含义是查找由 expression 指定的表达式与匹配字符串相匹配的记录。匹配字符串可以是一个完整的字符串，也可以使用%和_两种匹配符。%代表字符串中包含零个或多个任意字符；_代表字符串中包含一个任意字符。NOT 关键字是对 LIKE 运算符的否定，表示可以查询那些不匹配的记录。

例 5.10 以 scott 用户登录数据库，查询 emp 表中姓名以 A 开头的雇员信息。

```
SELECT * FROM emp
WHERE ename LIKE 'A%';
```

例 5.11 以 scott 用户登录数据库，查询 emp 表中姓名的倒数第二个字母是 E 的雇员信息。

```
SELECT * FROM emp
WHERE ename LIKE '%E_';
```

4. 范围比较

在 WHERE 子句中可以使用 BETWEEN 和 AND 关键字对表中某一范围内的数据进行查询，系统将逐行检查表中的数据是否在 BETWEEN 和 AND 关键字设定的范围内，该范围是一个连续的闭区间。如果在其设定的范围内，则取出该行，否则不取该行。其语法格式如下：

```
SELECT [ ALL | DISTINCT ] select_list
FROM [schema.]table_name | [ schema.]view_name
WHERE column_name [NOT] BETWEEN expressionl AND expression2
```

例 5.12 以 scott 用户登录数据库，查询 emp 表中雇佣日期为 1987 年的雇员的记录。

```
SELECT * FROM emp
WHERE hiredate BETWEEN '1-1月-1987' AND '31-12月-1987';
```

与 BETWEEN…AND…相对的 NOT BETWEEN…AND…，用于查询不在某一范围内的数据。

5. 使用查询列表

如果要查询的字段的取值范围不是一个连续的区间，而是一些离散的值，那么可以使用关键字 IN 进行查询，语法格式如下：

```
SELECT [ALL|DISTINCT] select_list
FROM [schema.]table_name | [ schema.]view_name
WHERE column_name [NOT] IN (value1, value2, ...)
```

例 5.13 以 scott 用户登录数据库，查询 emp 表中工作分别为 CLERK、ANALYST、MANAGER 的雇员信息。

```
SELECT * FROM emp
WHERE job IN ('CLERK','ANALYST','MANAGER');
```

该命令与下面命令等价：

```
SELECT * FROM  emp
WHERE job = 'CLERK' OR job = 'ANALYST' OR job = 'MANAGER';
```

与 IN 相对的 NOT IN，用于查询字段值不属于指定集合的记录。

6. 空值的判定

当需要判定一个表达式的值是否为空值时，使用 IS NULL 关键字。空值判定的语法格式如下：

```
SELECT [ALL|DISTINCT] select_list
FROM [schema.]table_name | [ schema.]view_name
WHERE column_name IS [NOT] NULL
```

这里要注意的是，空值 NULL 和任何数运算其结果都是 NULL。如果上面的语句，写成了“WHERE column_name＝NULL”，其结果是非常错误的，因为无论 column_name 的值是什么，其后的判断条件均是 NULL。

例 5.14 以 scott 用户登录数据库，查询 emp 表中经理为空的记录。

```
SELECT * FROM emp WHERE mgr IS NULL;
```

5.1.3 ORDER BY 子句

经常需要对查询结果排序输出，如雇员工资由高到低排列，SELECT 语句通过 ORDER BY 子句对查询结果进行排序显示。其语法格式如下：

```
SELECT [ALL|DISTINCT] select_list
FROM [schema.]table_name | [ schema.]view_name
ORDER BY col_name | expression [ASC | DESC][ , col_name| expression[ASC|DESC]...]
```

其中各参数的意义如下：

- col_name | expression：表示排序时用的字段或表达式。字段或表达式可以是一个也可以是多个。
- ASC：表示按升序排列，可省略。
- DESC：表示按降序排列。

例 5.15 以 scott 用户登录数据库，查询 emp 表中工作是 SALESMAN 的记录，按工资的降序排列。

```
SELECT * FROM emp
WHERE job = 'SALESMAN' ORDER BY sal DESC
```

注意：在默认情况下，ORDER BY 子句按升序进行排序，即默认使用的是 ASC 关键

字。如果特别要求按降序进行排列，必须使用 DESC 关键字。另外，ORDER BY 子句后边的排序字段名可以写成该字段在 SELECT 子句后边位置的数字序号；如果查询命令中为字段起了别名，那么还可以用字段的别名进行排序。如下面两条命令所示：

```
--使用字段的数字序号进行排序
SELECT ename, sal FROM  emp ORDER BY 2 asc;
--使用字段的别名进行排序
SELECT ename,sal AS salary FROM emp
ORDER BY salary asc;
```

当 ORDER BY 子句指定了多个排序列时，系统先将查询结果按照 ORDER BY 子句中第一列指定的顺序排列，当该列出现相同值时，再将这些行按照第二列的顺序排列，依次类推。

例 5.16 以 scott 用户登录数据库，将 emp 表中的数据记录先按工作升序排列，当工作相同时再按工资的降序排列。

```
SELECT empno, ename, job sal FROM emp
ORDER BY job, sal DESC;
```

5.1.4 统计函数

在对数据进行分析统计时常常会对表中的数据进行分类、统计、汇总、求标准差、协方差等操作，如统计雇员的人数，统计某部门的平均工资等，这些操作都可以使用 Oracle 提供的统计函数来实现。统计函数用来处理数值型数据，常用的统计函数如表 5-1 所示。

表 5-1 常用的统计函数

函 数 名	描 述
MAX(expression)	返回表达式集合中的最大值，忽略 NULL 值
MIN(expression)	返回表达式集合中的最小值，忽略 NULL 值
AVG([DISTINCT] expression)	返回表达式集合中的元素的平均值，忽略 NULL 值；如果使用 DISTINCT 选项，则去掉重复值后再求平均值
SUM([DISTINCT] expression)	返回表达式集合中的元素的汇总值，忽略 NULL 值；如果使用 DISTINCT 选项，则去掉重复值后再求汇总值
COUNT([DISTINCT] expression)	返回数据表中记录行数；如果使用 DISTINCT 选项，则去掉重复行后再求记录数
COUNT(*)	返回数据表中记录行数
STDDEV([DISTINCT|ALL] expression)	求标准差，ALL 表示对所有的值求标准差，DISTINCT 表示只对不同的值求标准差
VARIANCE([DISTINCT|ALL] expression)	求协方差，ALL 表示对所有的值求协方差，DISTINCT 表示只对不同的值求协方差
MEDIAN(expression)	求向量的中位数。向量长度为奇数时，中位数等于排序后恰好在中间位置的数值，向量长度为偶数时，中位数等于向量的中间位置相邻 2 个数据的平均值

例 5.17 以 scott 用户登录数据库，求 emp 表中所有雇员的平均工资、最高工资、最低工资、工资的总和。

```
SELECT AVG(sal) AS 平均工资,MAX (sal) AS 最高工资,
       MIN(sal) AS 最低工资,SUM (sal) AS 工资总和
FROM  emp;
```


查询结果如图 5-1 所示。

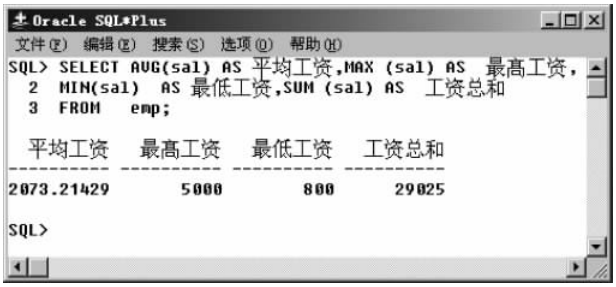


图 5-1　统计函数使用

例 5.18　以 scott 用户登录数据库，统计 emp 表中工作为 SALESMAN 的雇员人数。

```
SELECT COUNT (empno) AS 人数 FROM emp
WHERE job = 'SALESMAN';
```

例 5.19　以 scott 用户登录数据库，统计 emp 表中有多少种不同的工作(只统计数量)。

```
SELECT COUNT (DISTINCT job) AS 总人数 FROM emp;
```

5.1.5　GROUP BY 子句

前面介绍的统计函数都是对表中的所有行或满足 WHERE 条件的部分行进行一次统计运算，返回一个汇总结果。但有时候，需要将表中的数据按照某些字段值分组，然后对每组内的数据进行统计，从而得到多个汇总结果，此时必须使用 GROUP BY 子句。该子句的功能是根据指定的列将表中数据分成多个组，然后进行汇总。其语法格式如下：

```
SELECT [ ALL | DISTINCT ] select_list
FROM [schema.]table_name | [ schema.]view_name
WHERE search_condition
GROUP BY group_by_expression[,...n]
```

其中，group_by_expression 是用于分组的表达式，通常为字段名，可以是一个也可以是多个。如果分组的字段是多个，那么先按照第一个字段值分组，也就是将第一个分组字段值相同的行作为一组，然后在每个组内再按照第二个字段值进行分组，也就是说最终是基于这些列的唯一组合进行分组的，最后在分好的组中进行汇总。

在使用 GROUP BY 子句时，需要注意以下几个原则：

- 使用 GROUP BY 子句时，将分组字段值相同的行作为一组，而且每组只产生一个汇总结果，每个组只返回一行，不返回详细信息。
- 在 SELECT 子句的后面，只能有两种类型的表达式，一种是出现在 GROUP BY 子句后面的字段名或者是它的非统计函数表达式，另一种是其他非分组字段的统计函数表达式(统计函数可参见前面的介绍)。

- 如果在该查询语句中使用了 WHERE 子句，那么先在表中查询满足 WHERE 条件的记录，再将这些记录按照 GROUP BY 子句分组，也就是说 WHERE 子句先生效。
- GROUP BY 子句后面可以出现多个分组字段名，它们用逗号隔开。

例 5.20 以 scott 用户登录数据库，统计 emp 表中各种工作的雇员人数、工资标准差。

```
SELECT job,COUNT ( * ) AS 人数, STDDEV(sal) AS 工资标准差
FROM emp
GROUP BY job;
```

例 5.21 以 scott 用户登录数据库，统计 emp 表中各个部门中的各种工作的雇员人数。

```
SELECT deptno, job, COUNT( * ) AS 人数
FROM emp
GROUP BY deptno, job;
```

5.1.6 HAVING 子句

使用 GROUP BY 子句和统计函数对记录进行分组后，还可以使用 HAVING 子句对分组后的结果进一步筛选。如按工作分组后，求出各组的平均工资，然后在所有的组中查找平均工资大于 2500 的组记录。要注意的是：不可用 WHERE 实现等价的功能。

例 5.22 以 scott 用户登录数据库，统计 emp 表中平均工资大于 2500 的工作。

```
SELECT job,AVG(sal) AS 平均工资
FROM emp
GROUP BY job
HAVING AVG(sal)> 2500;
```

在上面这个查询中如果用 WHERE AVG(sal)＞2500 ，那将是非常错误的！

在 SELECT 语句中，当同时存在 GROUP BY 子句、HAVING 子句和 WHERE 子句时，其执行顺序为：先 WHERE 子句，后 GROUP BY 子句，再 HAVING 子句。即先用 WHERE 子句从数据源中筛选出符合条件的记录，接着用 GROUP BY 子句对选出的记录按指定字段分组、汇总，最后再用 HAVING 子句筛选出符合条件的组。

例 5.23 统计 scott 方案下的 emp 表中 1982 年后参加工作的、雇员人数超过了 2 人的部门编号。

```
SELECT deptno,COUNT( * ) AS  人数
FROM  emp
WHERE hiredate>'1 - 1 月 - 1982'
GROUP BY deptno
HAVING COUNT( * )> 2;
```

5.2 Oracle 数据库中常用的内置 SQL 函数

在数据库中，所谓的内置(Build In)函数就是已经在 DBMS 中实现了的系统函数。Oracle 提供了大量内置函数，用户可以利用这些函数完成特定的运算和操作。常用的函数包括以下几种：字符串处理函数，数学计算函数，日期时间函数，转换函数。当调用这些

SQL 函数时如果给其传递一个 NULL 参数，那么被调用函数自动返回 NULL，除 CONCAT，EGEXP_REPLACE，NVL，REPLACE 几个函数外。

5.2.1 字符串处理函数

字符串函数主要用于对字符串数据进行处理。可以在 SELECT 语句中使用字符串函数。常用的字符串函数如下所示：

- CONCAT(string1，string2)：连接两个字符串。
- LENGTH(string)：返回字符串 string 的长度。
- LOWER(string)：将给定字符串 string 的全部字母变成小写。
- UPPER(string)：将给定字符串 string 的全部字母变成大写。
- INITCAP(string)：将给定字符串 string 的首字母变成大写，其余字母不变。
- INSTR(string，value)：查询字符 value 在字符串 string 中出现的位置。
- LPAD(string，length[，padding])：在 string 左侧填充 padding 指定的字符串直到达到 length 指定的长度，若未指定 padding，则默认用空格填充。
- RPAD(string，length[，padding])：在 string 右侧填充 padding 指定的字符串直到达到 length 指定的长度，若未指定 padding，则默认用空格填充。
- LTRIM(string，[trimming_value])：去掉字符串 string 左边的由 trimming_value 指定的字符。
- RTRIM(string，[trimming_value])：去掉字符串 string 右边的由 trimming_value 指定的字符。
- REPLACE (string，string1 [，string2])：替换字符串。在字符串 string 中查找 string1，并用 string2 替换。如果没有指定 string2，则查找到指定的字符串时，删除该字符串。
- SUBSTR(string，start，[count])：获取字符串 string 的子串。返回 string 中从 start 位置开始长度为 count 的子串。

例 5.24 字符串函数举例。

```
SELECT UPPER('abc'),LOWER('ABC'),INITCAP('abc')
FROM DUAL;
SELECT SUBSTR(ename,1,2) ,length(ename)        /* 截取雇员姓名的前两位 */
FROM emp;
```

在上面的查询语句中，使用了 Oracle 系统的一个比较特别的表 DUAL。该表属于 SYS 方案，但所有用户都可以使用 DUAL 名称直接访问它。用 SELECT 计算常量表达式、伪列等值时经常使用该表，因为它只返回一行数据，而使用其他表时可能返回多个数据行。另外这个表还主要用来满足 SELECT 命令的语法要求，因为 Oracle 中要求 SELECT 命令必须包含 FROM 子句。

5.2.2 数值运算函数

数值函数通常对输入的数字参数执行某些特定的数学计算，并返回运算结果。常用的数值函数如下所示：

- ABS(value)：返回给定数字表达式的绝对值。
- CEIL(value)：返回大于或等于 value 的最小整数值。
- FLOOR(value)：返回等于或小于 value 的最大整数值。
- COS (value)：求余弦值。
- COSH (value)：求反余弦值。
- EXP(value)：返回以 e 为底的指数值。
- LN(value)：返回 value 的自然对数。
- POWER (value, exponent)：返回 value 的 exponent 次幂。
- SQRT(value)：返回 value 的平方根。
- ROUND(value, precision)：将 value 按 precision 精度进行四舍五入。
- MOD(value, divisor)：返回 value 除以 divisor 的余数。
- TRUNC(value, precision)：将 value 按 precision 精度进行截取，不进行四舍五入。

例 5.25 数值函数举例。

```
SELECT ROUND(3.567,2),TRUNC (3.567,2), CEIL(3.567), FLOOR(3.567)
FROM DUAL;
```

5.2.3 日期和时间函数

Oracle 提供了丰富的日期时间函数来处理日期型数据，常用的日期时间函数如下所示：

- ADD_MONTHS(date, number)：在指定的日期 date 上增加 number 个月。
- LAST_DAY(date)：返回日期 date 所在月的最后一天。
- MONTHS_BETWEEN(date1, date2)：返回 date1 和 date2 之间隔多少个月。
- NEW_TIME (date, current_zone, future_zone)：将 date 从 current_zone 时区转换为 future_zone 时区。
- NEXT_DAY(date, 'day')：返回指定日期(date)后的星期(day)对应的新日期。
- SYSDATE：返回系统的日期。
- CURRENT_TIMESTAMP：返回当前的日期和时间。
- EXTRACT (c1 from d1)：从日期 d1 中抽取 c1 指定的年、月、日、时、分、秒。

例 5.26 日期函数举例。

```
SELECT LAST_DAY (SYSDATE) FROM DUAL;
SELECT MONTHS_BETWEEN (SYSDATE, '1 - 7 月 - 2017') FROM DUAL;
SELECT NEXT_DAY (SYSDATE, '星期五') FROM DUAL;
SELECT EXTRACT(YEAR FROM SYSDATE) FROM DUAL;
SELECT EXTRACT(month FROM order_date) "Month",
COUNT(order_date) "No. of Orders"
FROM orders
GROUP BY EXTRACT(month FROM order_date)
ORDER BY "No. of Orders" DESC;
```

5.2.4 转换函数

在执行运算的过程中，经常需要把数据从一种数据类型转换为另一种数据类型，这种转

换既可以是隐式转换，也可以是显式转换。隐式转换是在运算过程中系统自动完成的，而显式转换则需要调用相应的转换函数来实现。常用的转换函数如下所示：

- TO_CHAR(date, 'format')：按照 format 的格式将日期型数据转换为字符串。
- TO_NUMBER(char)：将包含了数字的字符串转换为数值型数据。
- TO_DATE(string, 'format')：按照 format 的格式将 string 字符串数据转换为日期型。
- CHARTOROWID(char)：将字符串转换为 ROWID 类型。
- ROWIDTOCHAR(x)：将 ROWID 类型转换为字符串类型。
- NVL(exp1,exp2)：如果 exp1 的值是 NULL，则函数值返回 exp2；否则返回 exp1。

例 5.27 转换函数举例。

```
SELECT TO_CHAR (SYSDATE,'YYYY - MM - DD HH24:MI:SS) FROM DUAL;
SELECT TO_NUMBER ('1234')  FROM DUAL;
SELECT sal + NVL (comm,0) AS income FROM emp;
```

5.3 SQL * Plus 查询输出结果格式化

在 SQL * Plus 环境下，有许多参数可以控制 SQL * Plus 的输出显示格式。利用 SHOW ALL 命令，用户可以知道当前的显示格式的设置。用户可以通过 SET 命令设置各参数来改变当前的工作环境，在视窗环境的 SQL * Plus(sqlplusw. exe)中，也可以通过“环境”对话框来设置。设置的环境变量是临时性的，当用户退出 SQL * Plus 后，设置的参数将全部丢失。

5.3.1 SQL * Plus 环境中的常用格式化选项

SQL * Plus 环境中的常用格式化选项如下：

- arraysize：设置 SQL * Plus 一次从数据库中取出的行数，其取值范围为 1～500。
- autocommit：设置 Oracle 的提交方式。当设置为 On 时，自动提交用户做的更改；当设置为 Off 时，则必须等待用户使用 COMMIT 命令才能提交。
- linesize：设置 SQL * Plue 在一行中能够显示的总字符数，默认为 80。
- null：设置当 SELECT 语句返回 NULL 值时显示的字符串。
- numformat：设置数字的默认显示格式。
- newpage：设置每页打印标题前空的行数，默认值为 1。
- pagesize：设置每页打印的行数，该值包括 newpage 设置的空行数。
- pause：设置 SQL * Plus 输出结果时是否暂停。
- space：设置输出结果中列与列之间的空格数，默认值为 1。
- sqlcase：设置执行 SQL 命令前是否转换大小写。取值可以为 mixed（不转换）、lower（转换为小写）、upper（转换为大写）。
- sqlcontinue：设置 SQL * Plus 的命令提示符，默认为“>”。
- timing：控制是否统计每个 SQL 命令的运行时间。

- underline：设置 SQL * Plus 是否在列标题下面添加分隔符。
- wrap：当要显示的数据比列的宽度长时，该参数可设置是否截断数据项的显示。设置为 Off 时表示截断，设置为 On 时表示超出部分折叠到下一行显示。

除了上面列出的参数外，还有很多参数，可以通过 SHOW ALL 命令查看这些参数。若想查看某一个参数当前的设置值，可用 SHOW 命令加参数名称直接查看。

5.3.2 使用“环境”对话框设置格式化选项的值

使用“环境”对话框设置格式化选项的步骤如下：

(1) 启动视窗版的 SQL * Plus 工具(sqlplusw. exe，Oracle 11 中已废除了此功能，用 sqlplus. exe)。

(2) 选择“Options(选项)”菜单下的“Environment(环境)”选项，弹出如图 5-2 所示的对话框。在此对话框中可以对各个参数设置所需要的值，改变当前的工作环境。从“Set Option(设定选项)”列表中选择某一项后，“值”选项区域变亮，表示可以重新设置，设置完成后，单击“确定”按钮即可。

图 5-2 “环境”对话框

5.3.3 使用命令设置格式化选项的值

1. 使用 SET 命令设置环境参数命令格式如下：

```
SET <option> <value>
```

其中各参数的意义如下：

- option：用来控制当前环境的参数名称，option 包括用 SHOW ALL 命令输出的所有显示格式化参数。
- value：为该参数设置的新值。

例如，命令 SET LINESIZE 指定页宽的大小，默认值为 80；命令 SET PAGESIZE 指定一页的大小，默认值为 14。当页面和行的大小使用默认值时，用 SQL * Plus 登录 scott 用户，查询 emp 表中的记录的效果如图 5-3 所示。很显然这种显示结果是不友好的。

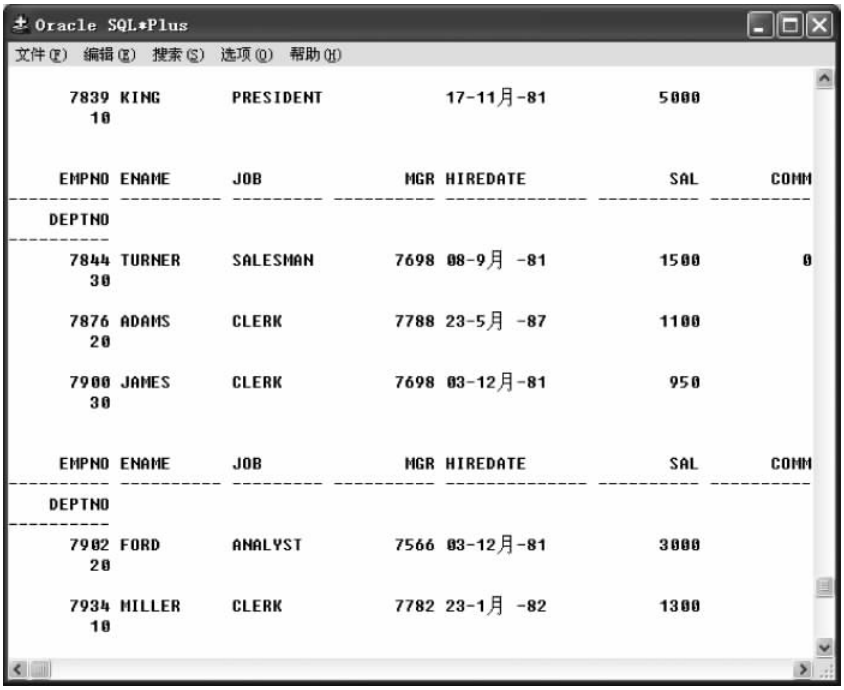


图 5-3　查询结果以默认行、页面大小显示效果

例 5.28　设置页宽和页的大小后登录 scott 用户，查询 emp 表中的记录，结果如图 5-4 所示。

```
SET LINESIZE 120
SET PAGESIZE 30
SELECT * FROM  emp;
```

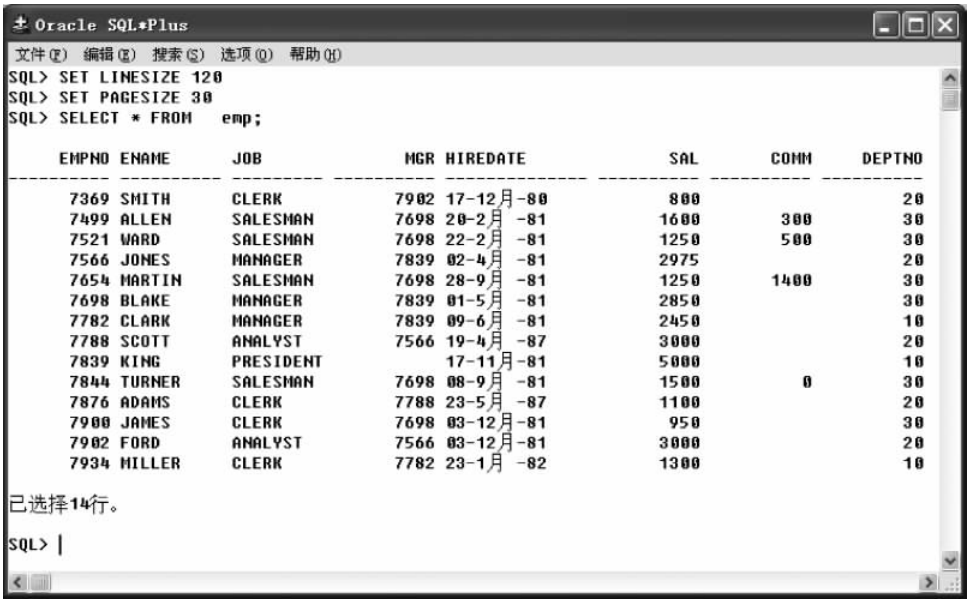


图 5-4　设置页面和行大小后查询效果

2. 使用 COLUMN 命令设置列的显示格式

命令格式如下：

```
COLUMN column_name | expression option
```

其中各参数的意义如下：

- column_name | expression：指定要格式化显示的数据项，可以是一个字段名也可以是一个表达式。
- option：指定数据项的显示属性。包括 FORMAT、HEADING、JUSTIFY、CLEAR 等。其中，FORMAT 用于设置列的显示宽度和格式；HEADING 用于设置列标题；JUSTIFY 用于设置列的对齐方式；CLEAR 用于清除列的属性。

例 5.29 以 scott 用户登录，查询 emp 表中的姓名、工作和工资，要求分别设置 job、sal 的显示格式。

```
SET PAGESIZE 30
COLUMN job FORMAT a8 WRAPPED
COLUMN sal FORMAT 999,999.00
SELECT ename, job, sal FROM emp;
```

执行结果如图 5-5 所示。job 字段使用 a8 指定显示宽度为 8 个字符，如果列值长度超过了定义的显示宽度，回行显示；sal 字段使用 999,999.00 指定显示 6 位数字，2 位小数，用逗号作为分隔符。

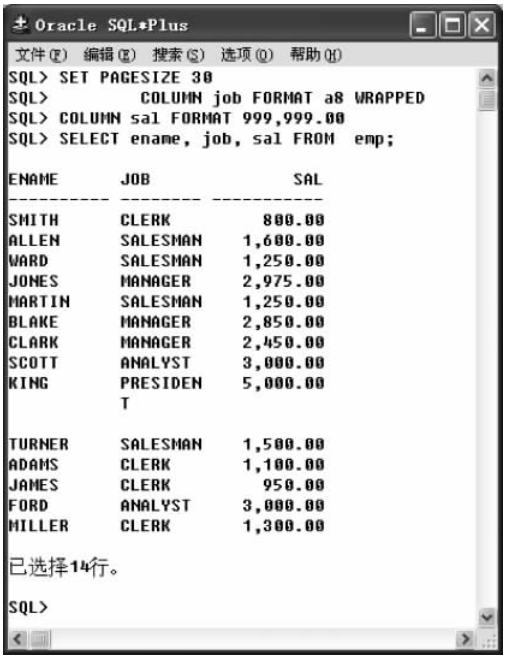


图 5-5 设置列的显示格式后的查询效果

用户如果想查看某列的显示格式，可以使用“COLUMN 字段名”查看，如 COLUMN job；用户可以通过 ON 或 OFF 设置某列的显示属性是否起作用，如 COLUMN job OFF，则

禁用 job 列的显示属性；用户还可以通过 CLEAR 选项清除设置的显示属性，如 COLUMN job CLEAR。

3. 使用 TTITLE 命令和 BTITLE 命令设置页眉页脚

可用 TTITLE 命令设置每页的标题，通常使用的默认设置为：标题文本在行中央，每页上都有日期和页码。可用 BTITLE 命令在每页的底部指定一些信息。例如：

```
SET LINESIZE 100
SET PAGESIZE 13
TTITLE 'SALESMAN 雇员信息'
BTITLE CENTER '--- report1 --- '
SELECT * FROM emp WHERE job = 'SALESMAN';
```

执行结果如图 5-6 所示。

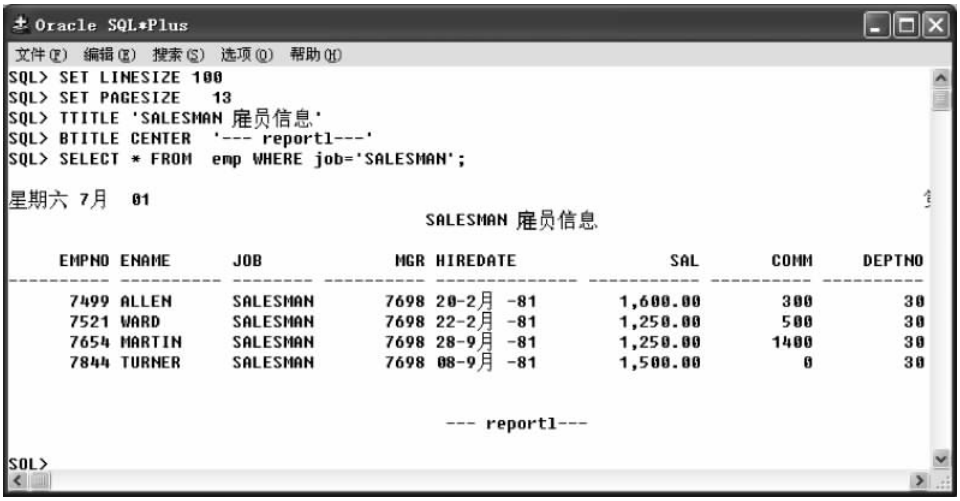


图 5-6　页标题和页底部的设置

5.4　SQL 脚本文件的创建与执行

SQL 脚本文件由 SQL 语句或 PL/SQL 程序组成，是一个可在 SQL * Plus 中执行的文件。用户可以把一个或多个 SQL 命令或 PL/SQL 块存放到 SQL 脚本文件中，可以编辑或执行指定的 SQL 脚本文件。

5.4.1　创建 SQL 脚本文件

SAVE 命令可将用户输入的 SQL 语句或 PL/SQL 程序保存到一个 SQL 脚本文件中，当用户需要时，可直接执行该文件，不需要重新输入。SAVE 命令的语法格式如下：

```
SAVE filename [CREATE | REPLACE | APPEND]
```

其中各参数的意义如下：

- filename：指定 SQL 脚本文件名，如果用户没有提供扩展名，则默认扩展名为. sql。

- CREATE | REPLACE | APPEND：CREATE 选项用于指定如果脚本文件不存在，则创建一个新文件，该选项为默认选项；REPLACE 选项用于指定如果文件不存在，则创建，否则用 SQL * Plus 缓冲区中的内容覆盖文件中的内容；APPEND 选项则把缓冲区中的内容追加到文件的末尾。

例 5.30 保存在 SQL * Plus 中执行的 SQL 语句到 D:\select_emp.sql 脚本文件中。

```
SELECT * FROM  EMP;
SAVE D:\select_emp.sql
```

执行结果如图 5-7 所示。

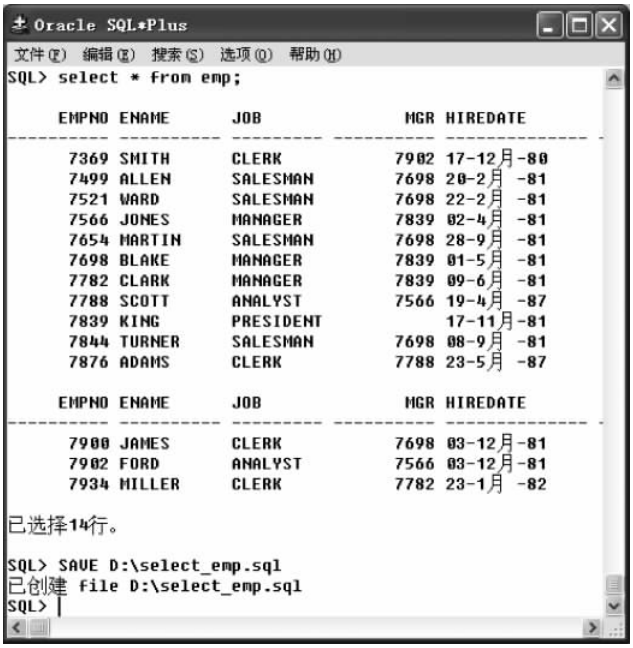


图 5-7 保存 SQL 脚本文件

可以使用 EDIT 命令编辑指定的脚本文件，其语法格式如下：

```
EDIT filename
```

例如，在 SQL * Plus 中执行 EDIT D:\select_emp.sql 命令可以打开文件的编辑窗口。

5.4.2 执行 SQL 脚本文件

在 SQL * Plus 中，可以使用@、@@、START 命令执行脚本文件。命令的语法格式如下：

```
@ |@@|START filename [ arg,...]
```

其中各参数的意义如下：

- fiename：指定要执行的脚本文件名。
- arg：为脚本文件中的参数提供的值。

例 5.31 执行例 5.30 创建的脚本文件 select_emp.sql。

```
START D:\select_emp.sql
```

或

```
@ D:\select_emp.sql
```

例 5.32 创建带参数的查询，保存到 D:\select_empl.sql 中，并执行此脚本文件。

```
SELECT * FROM emp WHERE sal > &1;
SAVE D:\select_empl. sql
@ D:\select_empl .sql 3000
```

在该例中出现了符号“&”，它表示定义替代变量，它后面的名字是替代变量的名字，可以是任意符合命名规范的标识符。在执行包含替代变量的命令时，要求用户为替代变量输入值。但是，如果在脚本文件的命令中包含替代变量，那么该替代变量的形式必须为 &[1-9] 格式，否则在执行脚本文件的命令时指定的实参无法传给命令中的替代变量。

@@命令也可以执行 SQL 脚本文件，但与@稍有不同。当主调用脚本文件和被调用脚本文件在同一目录时，命令可以直接以文件名执行被调用脚本文件，而@命令必须要求给出被调用脚本文件的目录。参见下例。

例 5.33 使用@@命令调用同一目录下的脚本文件。

(1) 在 D:\Oracle 目录下创建 3 个脚本文件，分别是 a.sql、b.sql、c.sql。

(2) 脚本文件 b.sql 中的命令是：

```
SELECT * FROM emp WHERE job = 'SALESMAN';
```

脚本文件 c.sql 中的命令是：

```
SELECT * FROM  emp WHERE job = 'CLERK';
```

(3) 在脚本文件 a.sql 中调用 b.sql 和 c.sql 文件，因此 a.sql 中的命令是：

```
@@ b.sql;
@@ c.sql;
```

(4) 在 SQL * Plus 中执行 a.sql 文件。

```
@ d:\oracle\a.sql
```

本例中，步骤(3)中的命令还可以写成如下形式：

```
@@ D:\Oracle\b. sql;
@@ D:\Oracle\c. sql;
```

或

```
@ D:\Oracle\b. sql;
@ D:\Oracle\c. sql;
```

5.5 多表连接查询

在 SELECT 语句的 WHERE 子句中提供一个连接条件，以过滤无意义的信息。多表连接查询传统的基本语法格式如下：

```
SELECT column_list
FROM [schema.]table_name1 | [ schema . ] view_name1,
      [schema. ] table_name2 | [schema.] view_name2[,...]
WHERE {connection_condition AND | OR search_condition }
```

其中各参数的意义如下：

- column_list：表示连接查询可以选择的字段列表，这些字段可以是 FROM 后面指定的所有表中包含的任意列。
- connection_condition：表示两表之间的连接条件，通常由两表的公共字段和关系运算符组成。
- search_condition：表示查询条件。

例 5.34 查询选修了课程并且成绩及格的学生的学号、姓名、课程号、课程名、成绩。

```
SELECT student.studentID,sname,score.courselD,cname,grade
FROM student,course,score
WHERE student.studentID = score.studentID AND course.courseID = score.courseID
     AND grade > = 60;
```

其中 student.studentID＝score.studentID AND course.courseID＝score.courseID 是三张表的连接条件，grade＞＝60 是查询条件。

表与表之间的这种连接查询是最传统的多表连接查询方式，可以把多个表连接起来，以满足生成复杂报告的需要。根据是否包含相关联的表中的匹配行和非匹配行，查询中的连接条件又分为内连接、外连接、自然连接等。除了传统的连接方式外，SQL92 后的标准还支持使用关键字 JOIN 的连接。使用 JOIN 连接的语法格式如下：

```
SELECT column_list
FROM table_name1 [join_type] JOIN table_nam2 ON connection_condition
    [[join_type] JOIN table_name3 ON connection_condition[...]
WHERE search_condition
```

其中，[join_type] JOIN 表示连接类型。包括：

- 内连接：INNER JOIN。
- 外连接：[LEFT|RIGHT|FULL] OUTER JOIN。
- 交叉连接：CROSS JOIN。

ON 关键字后面是连接条件，与简单连接查询中写在 WHERE 关键字后面的连接条件相同。

5.5.1 内连接查询

内连接是最常用的连接查询，一般使用 INNER JOIN 关键字来指定内连接，INNER 可

以省略。所谓内连接是指查询结果集中只包含满足连接条件的记录。当未指明连接类型时，默认为内连接。使用 JOIN 关键字的内连接查询与例 5.34 中类似的查询方式等价，即查询结果中只包含两个表中相匹配的行。内连接又可以分为等值内连接、不等值内连接和自然连接。

1. 等值内连接

等值内连接是在 ON 后面给出的连接条件中使用等号(=)运算符比较被连接的两张表的公共字段，其查询结果中只包含两表的公共字段值相等的行，列可以是两表中的任意列。

例 5.35 以 scott 用户登录数据库，基于 emp 表和 dept 表，查询雇员工资大于 2000 的雇员编号、姓名、工资、所在部门编号、部门名称。

```
SELECT empno, ename, sal, e. deptno, dname
FROM emp e INNER JOIN dept d ON e.deptno = d.deptno
WHERE SAL > 2000;
```

例 5.36 查询选修了数据库原理课程且成绩在 80 分以上的学生的学号、姓名、课程名和成绩。

```
SELECT s.studentID, sname, cname, grade
FROM student s JOIN score sc ON s. studentID = sc. studentID
    JOIN course c on sc. courselD = c.courselD
WHERE cname = '数据库原理' AND grade > 80;
```

2. 不等值内连接

不等值内连接是在连接条件中使用除“=”运算符以外的其他比较运算符比较被连接的公共字段。这些运算符包括>、> =、<=、<、!>、!<和<>。不等值内连接查询在实际应用中使用得较少。

3. 自然连接

自然连接(NATURAL JOIN)是一种特殊的等值内连接，它是由系统根据两表的同名字段自动作等值比较的内连接，因此不需要用 ON 关键字指定连接条件。在使用自然连接时需要注意两表的同名字段不能(也没有必要)用表名进行限制。因为进行的是等值比较，查询的结果集中同名字段的值是完全一样的，所以如果在 SELECT 后面使用“*”号，那么在查询结果集中系统只包含一列同名字段和它的值。

例 5.37 将例 5.35 改为自然连接。

```
SELECT empno, ename, sal, deptno, dname
FROM emp NATURAL JOIN dept
WHERE SAL > 2000;
```

要注意的是在该例中两表的同名字段 deptno 前面是不能加表名进行限制的。

5.5.2 外连接查询

内连接查询是保证查询结果集中的所有行都要满足连接条件，而使用外连接查询时，它返回的查询结果集中不仅包含符合连接条件的行，而且还包含连接运算符左边的表(简称左表，左外连接时)或右边的表(简称右表，右外连接时)，或两个连接表(完全外连接时)中的不

符合连接条件的行。

外连接分为：左外连接、右外连接和完全外连接。

1. 左外连接(LEFTJOIN 或 LEFT OUTER JOIN)

左外连接的结果集中包括两表连接后满足 ON 后面指定的连接条件的行(也就是内连接的结果集)和 LEFT OUTER JOIN 子句中指定的左表中不满足条件的行。也就是说左表中所有的行都会出现在查询的结果集中。如果左表的某行在右表中没有匹配行(即不满足比较条件的行),则在这些相关联的结果集中右表的所有选择列均为 NULL。

例 5.38 查询每个部门包括的雇员,如某部门没有雇员,也要显示其情况。要求显示部门名称、雇员名字。

```
SELECT dname, ename
FROM dept LEFT JOIN emp ON dept.deptno = emp.deptno;
```

在本例中,题目要求显示所有部门的名称,如果使用左外连接,那么部门信息表(dept 表)应放在关键字 LEFT JOIN 左边。

Oracle 数据库中使用特有的传统方法也可以实现两个表的左外连接,格式如下:

```
FROM 表 1,表 2
WHERE 表 1.公共字段 = 表 2.公共字段( + )
```

注意,左外连接中(+)符号要在等号的右边,此时会将等号左边表中的所有行都显示出来,等号右边表中只显示满足连接条件的行。将上面的例题改为如下形式:

```
SELECT dname,ename
FROM dept,emp
WHERE dept.deptno = emp.deptno( + );
```

2. 右外连接(RIGHT JOIN 或 RIGHT OUTER JOIN)

右外连接是左外连接的反向连接,将返回两表内连接的结果集和右表中不匹配的行。也就是说返回 RIGHT OUTER JOIN 关键字右边表中的所有行。如果右表的某行在左表中没有匹配行,则将为左表返回 NULL。

例 5.39 将例 5.38 中的左外连接改为右外连接。

```
SELECT dname, ename
FROM emp RIGHT OUTER JOIN dept ON dept.deptno = emp.deptno;
```

如果要显示 dept 表中所有行,则应将 dept 表放到 RIGHT OUTER JOIN 关键字的右边。

Oracle 数据库中使用特有的传统方法也可以实现两个表的右外连接,格式如下:

```
FROM 表 1,表 2
WHERE 表 1.公共字段( + ) = 表 2 .公共字段
```

注意,右外连接中(+)符号要在等号的左边,此时会将等号右边表中的所有行都显示出来,等号左边表中只显示满足连接条件的行。也可以将上面的例题改为如下形式:

```
SELECT enarne, dname
FROM emp,dept
```

```
WHERE emp.deptno( + ) = dept.deptno;
```

3. 完全外连接(FULL JOIN 或 FULL OUTER JOIN)

完全外连接查询的结果集包括两表内连接的结果集和左表与右表中不满足条件的行。当某行在另一表中没有匹配行时,则另一个表的选择列为 NULL。即两个表的所有行都将被返回。

例 5.40 使用 scott 方案下的 emp 表和 dept 表执行完全外连接查询。

```
SELECT ename,dname
FROM emp FULL OUTER JOIN dept
ON dept.deptno = emp.deptno ;
```

Oracle 数据库中使用传统方法不支持实现两个表的完全外连接,因为一个关系运算符最多有一个"(+)"符号。

5.5.3 交叉连接

交叉连接(CROSS JOIN)是用左表中的每一行与右表中的每行进行连接,不能使用 ON 关键字。因此,结果集中的行数是左表的行数乘以右表的行数,该连接查询的全集就是两个表的"笛卡儿乘积"。

例 5.41 用 scott 方案下的 emp 表和 dept 表进行交叉连接。

```
SELECT *
FROM emp CROSS JOIN dept;
```

注意:交叉连接没有 ON 关键字,但可以有 WHERE 子句。当带有 WHERE 子句时,则返回笛卡儿积中满足 WHERE 条件的所有行。

5.6 查询中的集合操作

SELECT 语句中的集合操作就是将两个或多个 SQL 查询结果集合并到一起的复合查询语句,可用这样的操作完成复杂的任务。集合操作主要由集合运算符实现,集合运算符包括:UNION(并集)、INTERSECT(交集)和 MINUS(差集)。

5.6.1 UNION 集合运算

UNION 运算符可以将多个查询结果集合并,形成一个结果集。多个查询的列的数量必须相同,数据类型必须兼容,且顺序必须一致,其语法格式如下:

```
SELECT_statement1
UNION [ALL] SELECT_statement2
UNION [ALL] SELECT_statement3 [...n]
```

其中,SELECT_statement1 等都是 SELECT 查询语句。在这个语句中 UNION ALL 是实现集合操作时不合并重复的行;而 UNION 则要合并重复的行。

例 5.42 使用 UNION 将工资大于 2000 的雇员信息与工作为 MANAGER 的雇员信息合并。

```
SELECT empno,ename, job, sal FROM emp
WHERE sal > 2000
UNION
SELECT empno,ename, job,sal FROM emp
WHERE job = 'MANAGER'
```

本例可以使用关键字 ALL,将保留结果集中的所有的行,包括重复行,查询结果集中的列标题来自第一个 SELECT 语句:

```
SELECT empno, enamef job, sal FROM emp
WHERE sal > 2000
UNION ALL
SELECT empno,ename, job,sal FROM emp
WHERE job = 'MANAGER';
```

当能确保不出现重复行的情况下,运用 UNION ALL 集合运算符的查询效率要比 UNION 查询的效率高。

5.6.2 INTERSECT 集合运算

与 UNION 类似,INTERSECT 也是对两个 SQL 语句所产生的结果进行处理。但与 UNION 不同,INTERSECT 集合运算是取两个结果集的交集。当使用该操作符时,只会显示同时存在于两个结果集中的数据,其语法格式如下:

```
SELECT_statement1
INTERSECT SELECT_statement2
INTERSECT SELECT_statement3 [...n]
```

其中,SELECT_statement1 等都是 SELECT 查询语句。

例 5.43 通过 INTERSECT 集合运算,查询工资大于 2000,并且工作为 MANAGER 的雇员信息。

```
SELECT empno, ename, job, sal FROM  emp
WHERE sal > 2000
INTERSECT
SELECT empno,ename, job, sal FROM emp
WHERE job = 'MANAGER';
```

5.6.3 MINUS 集合运算

MINUS 集合运算可以找到多个查询结果集的差异,即 MINUS 集合运算的结果包含在第一个结果集中但不在第二个结果集中的行,也就是两结果的差集。其语法格式如下:

```
SELECT_statement1
MINUS SELECT_statement2
MINUS SELECT_statement3 [...n]
```

例 5.44 在 emp 表中查询工资大于 2000,但不是经理(MANAGER)的雇员信息。

```
SELECT empno,ename, job, sal FROM emp
```

```
WHERE sal > 2000
MINUS
SELECT empno,ename, job, sal FROM emp
WHERE job = 'MANAGER';
```

当然，一些集合运算的功能完全可以改写为用 AND、OR、NOT 逻辑运算符来实现，读者可以将本节中的例子使用逻辑运算符实现。

5.7 子 查 询

子查询是实现复杂查询的途径之一。一般而言在一个查询条件中，可以嵌套另一个查询，即在一个 SELECT 查询内再嵌入一个 SELECT 查询。外层的 SELECT 语句叫外部查询，内层的 SELECT 语句叫子查询。子查询可以嵌套多层，但每层嵌套需要用圆括号()括起来。子查询除了可以用在 SELECT 语句中，还可以用在 INSERT、UPDATE 和 DELETE 语句中。子查询是一个完整的 SELECT 语句，只不过它作为其他 SQL 命令的一部分。大部分子查询是放在 SELECT 语句中的 WHERE 子句中实现的，也可以放到 FROM 子句中当作虚拟表。根据子查询返回的结果情况可将子查询分为单行子查询、多行子查询和多列子查询。

5.7.1 单行子查询

单行子查询是指子查询只返回单列单行数据，即只返回一个值，也可称为单值子查询。可以使用比较运算符，包括等于(=)、不等于(<>)、小于(<)、大于(>)、小于等于(<=)和大于等于(>=)。单行子查询应用最广泛，经常用在 SELECT、UPDATE、DELETE 语句的 WHERE 子句中充当查询、修改或删除的条件。

例 5.45 利用 scott 方案下的 emp 表和 dept 表查询在 SALES 部门工作的雇员姓名。

```
SELECT ename
FROM  emp
WHERE deptno = (SELECT deptno FROM dept WHERE dname = 'SALES');
```

该查询语句的执行过程为：首先对子查询求值，求出 SALES 的部门编号，然后把子查询的结果代入外部查询，执行外部查询。外部查询依赖于子查询的结果。

例 5.46 利用 scott 方案下的 emp 表查询工资低于平均工资的雇员信息。

```
SELECT * FROM emp
WHERE sal < (SELECT avg (sal) FROM emp)
```

一些使用子查询实现的功能，也可以用表之间的连接查询实现。如例 5.45 中的查询用连接查询实现，代码如下：

```
SELECT ename,dname
FROM emp e INNER JOIN dept d ON e.deptno = d.deptno
WHERE dname = 'SALES';
```

该查询语句也可以获得例 5.45 中的查询结果，但两种方式有不同之处，那就是连接查

询中的 SELECT 关键字的后面可以查询出 dept 表中的数据，但是在例 5.45 中的结果集中是不能输出的。

例 5.47 将 emp 表中雇员编号是 7369 的员工的工资改为平均工资的 1.5 倍。

```
UPDATE emp
SET sal = 1.5 * (SELECT avg (sal) FROM emp)
WHERE empno = 7369;
```

例 5.48 利用 emp 表和 dept 表，删除部门是 SALES 的员工信息。

```
DELETE FROM emp
WHERE deptno = (SELECT deptno FROM dept WHERE dname = 'SALES');
```

5.7.2 多行子查询

多行子查询是指子查询返回单列多行数据，即一组数据。当子查询是单列多行子查询时，必须使用多行比较运算符，包括 IN、NOT IN、ANY、ALL、SOME。IN 和 NOT IN 可以独立使用，表示用来比较表达式的值是否在子查询的结果集中。但是 ANY 和 ALL 必须与单行比较运算符组合起来使用，如下面的情况：

- <ANY：表示小于任何一个，即小于最大值即可。
- =ANY：表示等于任何一个，与 IN 类似。
- >ANY：表示大于任何一个，即大于最小值即可。
- <ALL：表示小于所有值，即小于最小值。
- >ALL：表示大于所有值，即大于最大值。
- =ALL：无意义。

SOME 和 ANY 类似。

例 5.49 利用 scott 方案下的 emp 表和 deot 表，查询所有部门名称是 SALES 和 RESEARCH 的员工编号、姓名、工资和工作。

```
SELECT empno 编号, ename 姓名,sal 工资,deptno 部门编号
FROM emp
WHERE deptno IN (SELECT deptno FROM dept
                 WHERE dname = 'SALES' OR dname = 'RESEARCH')
ORDER BY deptno;
```

该查询的执行顺序为：首先执行括号内的子查询得到结果，然后再利用该结果当作条件执行外部查询。对于复杂的查询，可以使用子查询的嵌套。

例 5.50 在学生选课系统中查询选修了课程名为数据库原理的学生信息。

```
SELECT *
FROM student
WHERE studentID IN (SELECT student ID FROM score
                    WHERE courseID = (SELECT courseID FROM course
                                      WHERE cname = '数据库原理'));
```

例 5.51 利用 scott 方案下的 emp 表查询每个部门的最低工资的雇员信息。

```
SELECT ename, deptno, sal FROM emp
WHERE sal IN (SELECT MIN(sal) FROM emp
              GROUP BY deptno);
```

例 5.52 利用 scott 方案下的 emp 表查询比工作在 SALESMAN 的所有员工工作早的那些雇员的信息。

```
SELECT *
FROM emp
WHERE hiredate < ALL (SELECT hiredate FROM emp WHRRE job = 'SALESMAN');
```

例 5.53 利用 scott 方案下的 emp 表查询工作是 CLERK 并且工资不低于工作是 SALESMAN 的最低工资的雇员信息。

```
SELECT * FROM emp
WHERE job = 'CLERK' AND sal > ANY (SELECT sal FROM emp
                                   WHERE job = 'SALESMAN');
```

5.7.3 多列子查询

单行子查询和多行子查询获得的都是单列数据，但是多列子查询获得的是多列任意行数据。当多列子查询返回单行数据时，在 WHERE 子句中可以使用单行比较符(=，>，<，>=，<=，<>)来进行比较；而返回多行数据时，在 WHERE 子句中必须使用多行比较符(IN、ANY、ALL 和 SOME)来进行比较。

例 5.54 利用 emp 表查询与编号为 7369 的雇员的部门和工作岗位完全相同的所有雇员。

```
SELECT ename, job, sal, deptno
FROM emp
WHERE (deptno, job) = (SELECT deptno, job FROM emp WHERE empno = 7369);
```

使用子查询比较多列数据时，既可以使用成对比较，也可以使用非成对比较。其中，成对比较要求多个列的数据必须同时匹配，而非成对比较则不要求多个列的数据必须同时匹配，此时是单独写的查询条件，各个条件之间是彼此独立的。

例 5.55 利用 emp 表查询工资和奖金与部门编号为 30 的雇员的工资和奖金完全相同的雇员信息。

```
SELECT ename, sal, comm, deptno
FROM emp
WHERE (sal, NVL(comm, -1)) IN (SELECT sal, NVL (comm, -1)
                               FROM emp WHERE deptno = 30);
```

此查询为成对比较。

例 5.56 利用 emp 表查询工资匹配于部门 30 的工资列表、奖金匹配于部门 30 的奖金列表的所有雇员。

```
SELECT ename, sal, comm, deptno
```

```
FROM emp
WHERE sal IN (SELECT sal FROM emp WHERE deptno = 30)
        AND NVL (comm, -1) IN (SELECT NVL (comm, -1)
                               FROM emp WHERE deptno = 30);
```

此查询为非成对比较。

5.7.4 相关子查询

相关子查询是指需要引用外查询表列的子查询语句，是通过 EXISTS 运算符实现的查询。EXISTS 用于测试子查询的结果是否为空，如子查询的结果集不为空，则 EXISTS 返回 TRUE，否则返回 FALSE。EXISTS 还可以与 NOT 合用，即 NOT EXISTS，其返回值与 EXISTS 恰好相反。

例 5.57 在 emp 表中查询工作在 NEW YORK 的所有雇员信息。

```
SELECT ename, job, sal,deptno
FROM emp
WHERE EXISTS (SELECT 'x' FROM dept
             WHERE deptno = emp.deptno AND loc = 'NEW YORK');
```

在该查询语句中，外层 SELECT 语句返回的每一行数据都要根据子查询来评估，如果 EXISTS 关键字中指定的条件为真，查询结果就包含这一行，否则不包含这一行。

本例的执行过程是：首先查找外层查询中 emp 表的第 1 行，根据该行的 deptno 值处理子查询，若结果不为空，则条件为真，就把该行的信息取出作为结果集的一行。然后继续查找 emp 表的第 2 行，第 3 行……重复上面的处理过程直到 emp 表的所有行都查找完为止。本例也说明了相关子查询的效率要高于 in 方式的子查询，在实际应用中要尽可能地利用相关子查询解决问题、少用 in 方式的子查询。

5.7.5 子查询在 FROM 子句中运用

前面介绍的子查询都是用在 WHERE 子句，子查询还可以用在 FROM 子句，该子查询会被作为视图对待，因此也被称为内联视图(Oracle 数据库把子查询也称为内联视图 Inline Views)。

例 5.58 利用 emp 表查询高于部门平均工资的雇员信息。

```
SELECT ename, job, sal
FROM emp e, (SELECT deptno, AVG (sal) AS avg_sal
             FROM emp GROUP BY deptno) d
WHERE e.deptno = d.deptno AND e.sal > d.avg__sal;
```

本查询将子查询(SELECT deptno，AVG (sal) AS avg_sal FROM emp GROUP BY deptno)的结果作为一个视图对待，把该子查询的结果集当成视图 d，让 emp 表和 d 视图进行连接查询，因而得名为内联视图。

例 5.59 查询 scott 方案中平均工资最高的部门的部门编号。

```
SELECT deptno
FROM (SELECT deptno, AVG(sal) avg_sal FROM  emp GROUP BY deptno) A
```

```
WHERE A.avg_sal = (SELECT MAX (B.avg_sal)
            FROM (SELECT dcptno,AVG (sal) avg_sal
                  FROM emp
                  GROUP BY deptno) B);
```


5.8 伪列在查询中的应用

Oracle 数据库中，一个伪列所扮演的角色就像一个表列，但它实际上并不存储在表中。在 SQL 语句中可以从伪列中取值，但不能插入、更新或删除其值。伪列通常为每行返回一个不同的值。在 Oracle 中，最常用的伪列是 ROWID、ROWNUM 和 LEVEL。

5.8.1 ROWID 伪列

对于数据库中的每一行，ROWID 伪列返回数据库表中每一行的物理地址信息，它能唯一地表示一行数据，无论数据行是否重复，其 ROWID 值是绝对唯一的。

ROWID 的值有几个重要用途：

- 它是访问单个行数据的最快方式，通过它可以很快地定位到要访问的数据。
- 它可以告诉用户表中的行是如何存储的。
- 它是表中行的唯一标识符。

可以使用函数 SUBSTR 将 ROWID 中的数据分解成其组件。例如，使用 SUBSTR 函数将扩展 ROWID 伪列分解为其四个组件(数据库对象、文件、块和行)：

```
SELECT ROWID,SUBSTR(ROWID,1,6) "OBJECT",SUBSTR(ROWID,7,3) "FIL",
          SUBSTR(ROWID,10,6) "BLOCK",SUBSTR(ROWID,16,3) "ROW"
FROM emp;
```

例 5.60 ①查询 emp 表中每行的 ROWID；②利用 ROWID 删除数据表中的重复行。

```
SELECT ROWID,empno,ename,job
FROM emp
WHEREdeptno = 20;
```

这个查询通过 ROWID 返回每行的物理地址。

```
DELETE from Test A
where ROWID!= (select MAX(ROWID)
               from Test B where B.col1 = A.col1 and B.col2 = A.col2);
```

在这个案例中，Test 表有两列 col1 和 col2，表中有多行数据重复，可以在子查询中求出重复行数据的最大 ROWID，将重复行中其 ROWID 不等于最大 ROWID 的行都删除，这样对于这些重复行来说，只留下一行数据了，从而达到删除重复行数据的目的。

5.8.2 ROWNUM 伪列

Oracle 查询数据时，将满足条件的数据提取出后，就在结果集中为每一行增加了一个序号，这个序号就是 ROWNUM，它总是从 1 开始的。ROWNUM 也是 Oracle 从数据文件中读取满足条件的记录时，按先后给每行编的顺序号，读出的第 1 行编号为 1。给数据的排

序是在已有结果集上进行的，按指定字段排序不会影响 Oracle 分配给每行的 ROWNUM。

在实际应用中可以使用 ROWNUM 来限制查询返回的行数，例如：

```
SELECT rownum,empno,ename,job FROM emp
WHERE ROWNUM <= 10;
```

查询结果如图 5-8 所示。

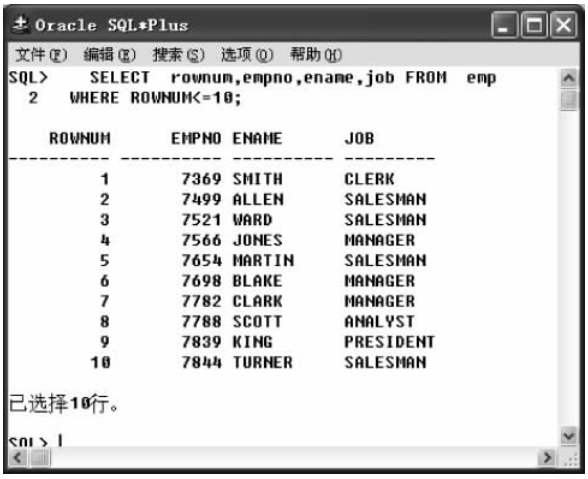


图 5-8　利用 ROWNUM 限制返回记录行数

同样的例子，在其上增加了 ORDER BY 子句：

```
SELECT rownum,empno,ename,job FROM emp
WHERE ROWNUM <= 10
ORDER BY ENAME;
```

查询结果如图 5-9 所示，可见排序不会影响 ROWNUM 本次分配给记录行的序号。

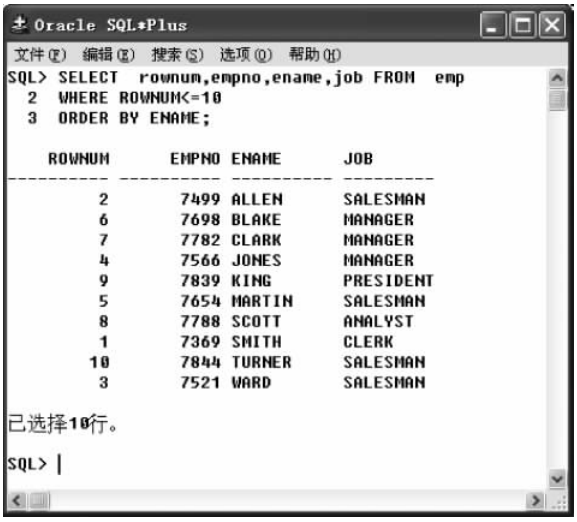


图 5-9　ORDER BY 后的 ROWNUM

5.8.3 Oracle 中的树形查询

在 Oracle 数据库中有一个强大的遍历树形结构的功能，那就是层次查询。层次查询遍历树形结构的数据集合，来获取树的层次数据结构关系表现的数据。其语法格式如下：

```
SELECT [LEVEL],column,expr...
FROM table_name
[WHERE conditions]
[START WITH conditions
[CONNECT BY PRIOR conditions];
```

如图 5-10 所示为一个树形数据结构图，它反映了 scott 方案下 emp 表的数据关系，在 emp 表中 empno 是雇员编号、mgr 是雇员的直接上级编号。Oracle 的树形结构查询可非常高效地遍历这样的树形结构图。

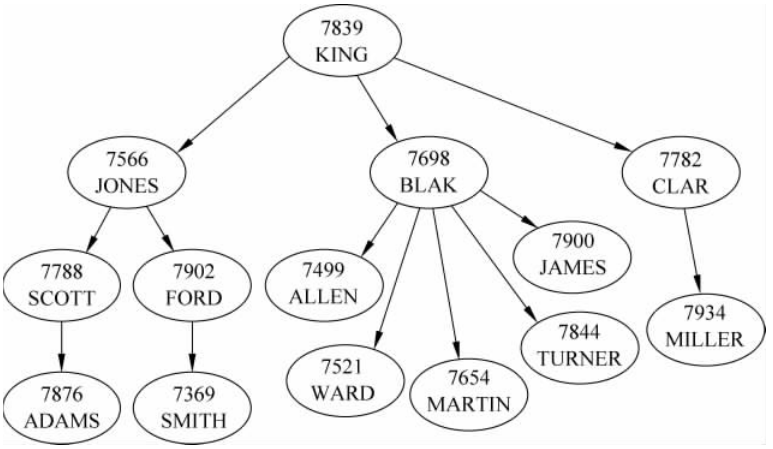


图 5-10　emp 表树形数据结构图

要实现层次查询必须通过 START WITH 子句和 CONNECT BY 子句来实现。层次查询中相关子句及选项解释如下：

① LEVEL 是伪列，表示本次遍历时数据记录的层次号。在具有树结构的表中，每一行数据都是树结构中的一个节点，由于节点所处的层次位置不同，所以每行记录都可以有一个层号。层号根据节点与根节点的距离确定。不论从哪个节点开始，该起始根节点的层号始终为 1，根节点的子节点为 2，依此类推。

② FROM 后面只能是一个表或视图，对于 FROM 是视图的，那么这个视图不能包含连接。

③ WHERE 条件限制了查询返回的行，但是不影响层次关系，属于将节点截断，但是这个被截断的节点的下层子树不受影响。

④ PRIOR 是个形容词，是前一个节点的意思，可以在 CONNECT BY 等号的前后，列之前。

⑤ 彻底剪枝条件应放在 CONNECT BY 子句中；单点剪掉条件应放在 WHERE 子句。但是，CONNECT BY 的优先级要高于 WHERE，也就是 SQL 引擎先执行 CONNECT BY，

CONNECT BY 确定树的遍历的方向是从根到叶还是从叶到根。

⑥ START WITH 确定遍历从哪个节点开始，其后的表达式可以有子查询，但是 CONNECT BY 中不能有子查询。

• 从叶子到根进行遍历(通过子节点向根节点追溯)

先由叶子节点开始然后遍历到根节点。Parent_key 表示父节点 key，Child_key 表示子节点 key。在这种情况下 CONNECT BY 的设置如下：

CONNECT BY PRIOR Parent_key＝Child_key 表示上一条记录的父 Key 是本记录的子 Key，等同于 CONNECT BYChild_key＝PRIOR　Parent_key。

例如执行下列查询后，查询结果如图 5-11 所示。可理解为树叶在上，树根在下结构。

```
SELECT LEVEL empno,mgr,ename,job
FROM emp
START WITH mgr = 7839
CONNECT BY PRIOR MGR = EMPNO -- 上一条记录的 mgr 是本记录的 empno
```

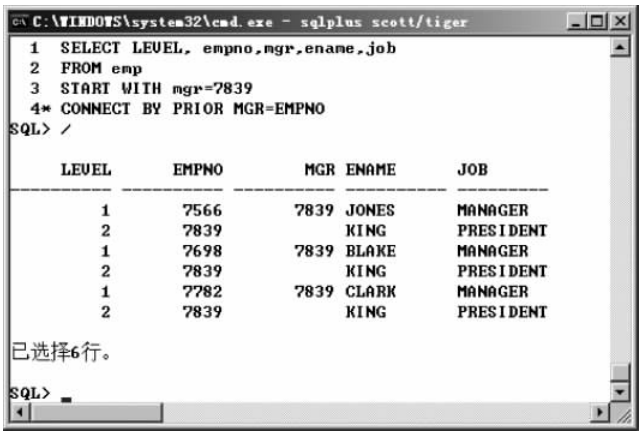


图 5-11　自顶向下遍历结果图

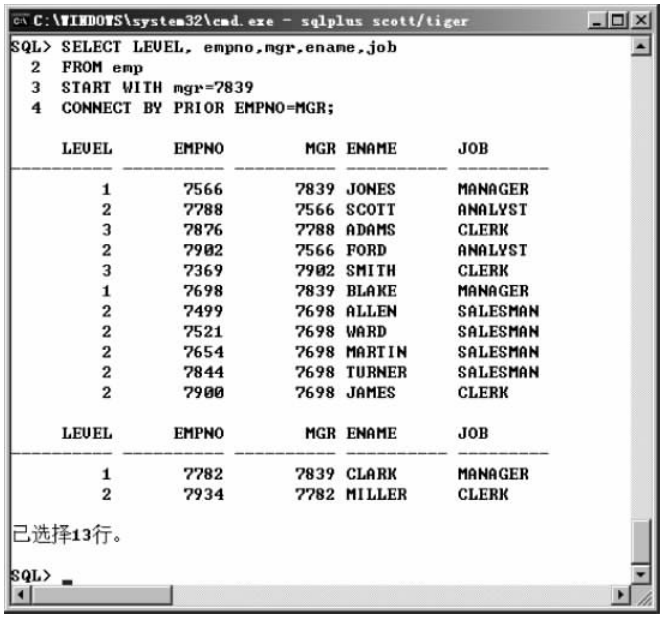
• 从根到叶子遍历(通过根节点遍历子节点)

先由根节点开始遍历一直找到叶子节点。CONNECT BY 之后不能有子查询，但是可以加其他条件，比如 and id !＝2 等。这句话则会截断树枝，如果 id＝2 的这个节点下面有很多子孙后代，则全部截断不显示。如图 5-12 所示为自底向上遍历结果图。查询语句如下：

```
SELECT LEVEL empno,mgr,ename,job
FROM emp
START WITH mgr = 7839
CONNECT BY PRIOR EMPNO = MGR; -- 上一条记录的 empno 是本记录的 mgr
```

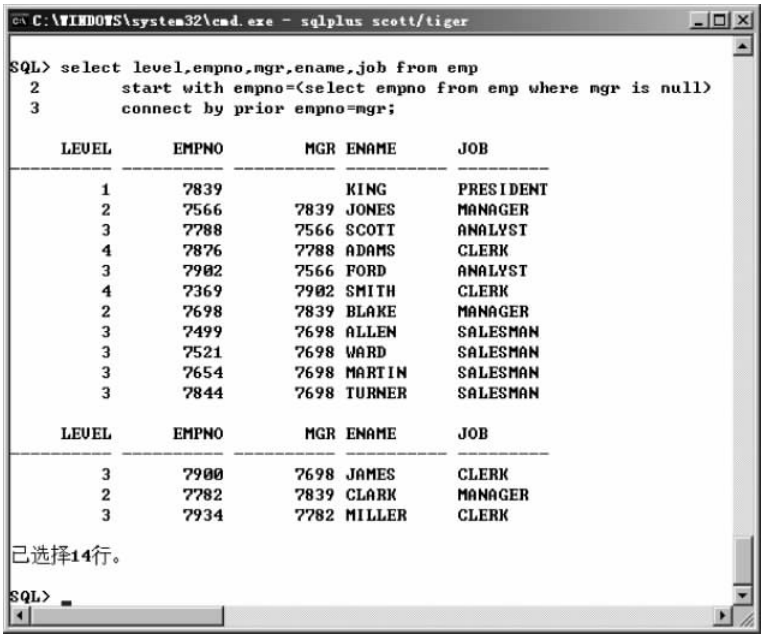
在树形查询中，START WITH 子句中可以带有子查询，用来确定开始遍历的起始节点，如下查询，它的执行结果如图 5-13 所示。

```
SELECT LEVEL,empno,mgr,ename,job FROM emp
START WITH empno = (SELECT empno FROM emp WHERE mgr IS NULL)
CONNECT BY PRIOR empno = mgr;
```



```
C:\WINDOWS\system32\cmd.exe - sqlplus scott/tiger
SQL> SELECT LEVEL, empno,mgr,ename,job
  2  FROM emp
  3  START WITH mgr=7839
  4  CONNECT BY PRIOR EMPNO=MGR;

     LEVEL      EMPNO        MGR ENAME      JOB
---------- ---------- ---------- ---------- ---------
         1       7566       7839 JONES      MANAGER
         2       7788       7566 SCOTT      ANALYST
         3       7876       7788 ADAMS      CLERK
         2       7902       7566 FORD       ANALYST
         3       7369       7902 SMITH      CLERK
         1       7698       7839 BLAKE      MANAGER
         2       7499       7698 ALLEN      SALESMAN
         2       7521       7698 WARD       SALESMAN
         2       7654       7698 MARTIN     SALESMAN
         2       7844       7698 TURNER     SALESMAN
         2       7900       7698 JAMES      CLERK

     LEVEL      EMPNO        MGR ENAME      JOB
---------- ---------- ---------- ---------- ---------
         1       7782       7839 CLARK      MANAGER
         2       7934       7782 MILLER     CLERK

已选择13行。

SQL>
```

图 5-12　自底向上遍历结果图



```
C:\WINDOWS\system32\cmd.exe - sqlplus scott/tiger
SQL> select level,empno,mgr,ename,job from emp
  2        start with empno=(select empno from emp where mgr is null)
  3        connect by prior empno=mgr;

     LEVEL      EMPNO        MGR ENAME      JOB
---------- ---------- ---------- ---------- ---------
         1       7839            KING       PRESIDENT
         2       7566       7839 JONES      MANAGER
         3       7788       7566 SCOTT      ANALYST
         4       7876       7788 ADAMS      CLERK
         3       7902       7566 FORD       ANALYST
         4       7369       7902 SMITH      CLERK
         2       7698       7839 BLAKE      MANAGER
         3       7499       7698 ALLEN      SALESMAN
         3       7521       7698 WARD       SALESMAN
         3       7654       7698 MARTIN     SALESMAN
         3       7844       7698 TURNER     SALESMAN

     LEVEL      EMPNO        MGR ENAME      JOB
---------- ---------- ---------- ---------- ---------
         3       7900       7698 JAMES      CLERK
         2       7782       7839 CLARK      MANAGER
         3       7934       7782 MILLER     CLERK

已选择14行。

SQL>
```

图 5-13　START WITH 中的子查询确定遍历开始节点

5.9　习　　题

根据 Oracle 数据库中的 scott 方案下的 emp 和 dept 表，写出实现下列需求的 SQL 语句。

(1) 查询所有工种为 CLERK 的员工的姓名及其部门名称。

(2) 查询所有部门及其员工信息,包括那些没有员工的部门。

(3) 查询所有员工及其部门信息,包括那些还不属于任何部门的员工。

(4) 查询在 SALES 部门工作的员工的姓名信息。

(5) 查询所有员工的姓名及其直接上级的姓名。

(6) 查询入职日期早于其上级领导的所有员工信息。

(7) 查询从事同一种工作但不属于同一部门的员工信息。

(8) 查询 10 号部门员工及其领导的信息。

(9) 使用 UNION 将工资大于 2500 的雇员信息与工作为 ANALYST 的雇员信息合并。

(10) 通过 INTERSECT 集合运算,查询工资大于 2500,并且工作为 ANALYST 的雇员信息。

(11) 使用 MINUS 集合查询工资大于 2500,但工作不是 ANALYST 的雇员信息。

(12) 查询工资高于公司平均工资的所有员工信息。

(13) 查询与 SMITH 员工从事相同工作的所有员工信息。

(14) 查询工资比 SMITH 员工工资高的所有员工信息。

(15) 查询比所有在 30 号部门中工作的员工的工资都高的员工姓名和工资。

(16) 查询部门人数大于 5 的部门的员工信息。

(17) 查询所有员工工资都大于 2000 的部门的信息。

(18) 查询人数最多的部门信息。

(19) 查询至少有一个员工的部门信息。

(20) 查询工资高于本部门平均工资的员工信息。

(21) 查询工资高于本部门平均工资的员工信息及其部门的平均工资。

(22) 查询每个员工的领导所在部门的信息。

(23) 查询平均工资低于 2000 的部门及其员工信息。

(24) ROWNUM 和 ROWID 两个伪列有何不同?

(25) UNION 和 UNION ALL 在集合操作时有何区别?

(26) 根据树形结构查询的原理,编写 SQL 将 scott 方案下的 emp 表中同一个级次的雇员列在一起。