八、多表查询、分组查询、子查询

# 多表查询

在Oracle中，多表查询有多中连接形式：

## 连接类型

常见的连接类型有：

1. **等值连接（Equijoin）**

**等值连接（Equijoin）：**标间有含义相同的关联字段可以比较；

|  |
| --- |
|  |

**注：**

* 可使用AND 操作符增加查询条件；
* 使用表别名可以简化查询；
* 使用表名（表别名）前缀可提高查询效率；
* 为了连接n个表，至少需要n-1个连接条件；

1. **非等值连接（Non-Equijoin）**

**非等值连接（Non-Equijoin）：**标间没有直接相关联的字段，需要通过表中字段的范围进行判断；

|  |
| --- |
|  |

1. **自连接（Self join）**

**自连接（Self join）：**同一张表中有两个数据相同但含义不同的字段；把一张表当做两张表使用，为同一张表两个别名；

|  |
| --- |
|  |

1. **外连接（ Outer join ）**

使用外连接可以看到参与连接的某一方不满足连接条件的记录分为：**左外连接，右外连接，全外连接；**以上属于SQL1992标准，而下面几种连接属于SQL1999标准：

1. **交叉连接（Cross join）**
2. **自然连接（Natural join）**
3. **使用Using子句建立连接**
4. **使用On子句建立连接**

## 多表查询的概念

在查询数据库记录时，如果查询字段包含多张表中的数据，则要用到多表查询。

语法如下：

|  |
| --- |
| **SELECT [DISTNCT] \*| 字段[别名] [字段[别名]…]**  **FROM 表名称1[别名1], [表名称2[别名2]…]**  **[WHERE 条件（s）];** |

**补**：统计一张表中记录的条数，即一张表中的数据量，可以用COUNT函数进行查询；

1. COUNT函数的应用，查询emp表中的记录数；

|  |
| --- |
|  |

**补**：表查询的经验，在查询一个未知的数据库或表时，要先用相关函数查看数据量，然后再查询数据，若直接查询数据，可能因为数据量过大而造成系统异常。例如：在大数据用户sh中，有一张名叫sales 的表，其中的数据记录有90多万条，若直接查询记录，则可能造成系统异常；

|  |
| --- |
|  |

1. 多表查询示例，查询emp,dept表中的记录数；

|  |
| --- |
|  |
|  |
|  |

由上表可知，多表查询的记录数是单独查询两张表的记录数的乘积，这是由于数据库的查询机制造成的，它将emp表中的每条记录都与dept表中的每条记录匹配了一次；

|  |
| --- |
|  |

这种查询机制称为笛卡尔积，相当于两个集合A、B，分别有m、n个元素，从A、B中各取一个元素配对，一共有m\*n种配对方式；

**注**：如何去掉笛卡尔积？

**方法1.**通过两表之间的关联字段进行判断，emp与dept表的关联字段是deptno，通过判断emp表中deptno是否等于dept表中deptno来判断是否显示该记录：

|  |
| --- |
|  |
|  |

**补：**在多表查询中，若多个表中有相同的字段名，则需要注明字段是哪张表的，格式为：

|  |
| --- |
| 表名.字段名 |

若表名很长，则需要先为表起一个别名，方便使用。

使用上述方法只是在显示上去除了笛卡尔积的影响，但实际在数据库中查询记录时，依然采用笛卡尔积匹配。若所查询表中数据很大，则匹配次数也会很大，查询性能将会很差，所需时间很长，可能造成系统异常。例如，在sh用户中，有sales和costs两张表，若通过上述方式查询两张表，则会非常费时；

|  |
| --- |
|  |
|  |
| SELECT \* FROM sales,costs WHERE sales.prod\_id=costs.prod\_id; |

由此可见，多表查询在数据量很大的情况下，性能很差；

1. 多表查询的应用，查询员工的编号、姓名、职位、部门名称、部门位置；

思路：

1. 找出与查询相关的表：emp、dept
2. 找出表间的关联字段：deptno
3. 查询emp表中的记录：

|  |
| --- |
| SELECT e.empno,e.ename,e.job FROM emp e; |

4）添加dept表及其字段：

|  |
| --- |
| SELECT e.empno,e.ename,e.job,d.dname,d.loc FROM emp e,dept d; |

5）去除笛卡尔积：

|  |
| --- |
|  |

**注：**将问题分步进行可以使复杂的问题简单化，分步进行多表查询将会更加简便；

1. **自连接**：查询出每一个雇员的姓名、职位、所属领导的姓名；

|  |
| --- |
|  |

**分析**：每个雇员都有自己的编号以及所属领导的编号，而领导也是雇员，故也属于emp表，这种关联叫自身关联；可以将其看做是两个相同表的多表查询：

1）查询雇员的姓名、职位：

|  |
| --- |
| SELECT e.ename,e.job FROM emp e; |

2）加入领导姓名：

|  |
| --- |
| SELECT e.ename,e.job,m.ename manager FROM emp e,emp m; |

3）设置关联条件：

|  |
| --- |
|  |

**注**：此查询缺少一条记录，即没有领导的那个员工的记录；若要解决此问题，需要用到左、右连接。

1. **等值关联**：查询员工的编号、姓名、职位、工资、所属领导编号、所属部门、部门位置；

分析：

1. 确定所关联的表：emp,dept;
2. 确定关联字段：deptno,empno;
3. 查询员工编号、姓名、职位、工资：

|  |
| --- |
| SELECT e.empno,e.ename,e.job,e.sal FROM emp e; |

1. 增加领导姓名、所属部门、部门位置信息；

|  |
| --- |
| SELECT e.empno,e.ename,e.job,e.sal,m.ename,d.dname,d.loc FROM emp e,emp m,dept d; |

5）设置关联条件

|  |
| --- |
|  |

1. **非等值关联**：查询每个雇员的编号、姓名、工资、所在部门名称、工资等级；

思路：

1. 找出相关联的表：emp、dept、salgrade
2. 找出关联字段：deptno、sal(losal/hisal)
3. 查询emp表中的信息：

|  |
| --- |
| SELECT e.empno,e.ename,e.sal FROM emp e; |

4）添加部门信息和工资等级信息：

|  |
| --- |
| SELECT e.empno,e.ename,e.sal,d.dname,s.grade FROM emp e,dept d,salgrade s; |

5）设置关联条件：

|  |
| --- |
|  |
|  |

1. 查询每个雇员的编号、姓名、工资、所在部门名称、工资等级（工资等级用文字翻译，如‘1’翻译为‘第一等工资’）；

思路：采用DECODE函数进行判断替换；

|  |
| --- |
|  |

## 外连接：左、右连接

问题：在多表查询时，有些数据常常因为不符合关联条件而被遗漏，例如：查询员工信息以及所在部门信息；

|  |
| --- |
|  |

显示的结果中，部门信息只有3个，因为部门40中没有员工，故不符合关联条件而被舍弃；

为解决此问题，Oracle提供了左、右连接机制解决问题；符号是：

* 表名.字段名(+)=表名.字段名：表示右连接，连接右边表中缺失的内容；
* 表名.字段名=表名.字段名(+)：表示左连接，连接左边表中缺失的内容；

故以上问题可以这样解决：

|  |
| --- |
| SELECT \* FROM emp,dept WHERE emp.deptno(+)=dept.deptno; |

**注**：SQL语句中的符号，一律是英文格式符号，若换成中文格式符号会报异常；

1. 查询员工姓名以及其所属领导姓名，并显示无领导的员工；

|  |
| --- |
|  |

**注**：上述多表查询的方法属于Oracle所特有的方法，若想在其他数据库中实现多表查询的语法还需要用到SQL1999提供的标准。

# 第二节 SQL1999多表查询语法

很多时候Oracle数据库中的操作都是它自己特有的，当要在其他数据库中用到相应操作时，就需要借助SQL1999的标准语法来解决问题了。采用SQL1990语法解决多表查询问题的语法：

|  |
| --- |
| **SELECT table1.column , table2.column**  **FROM table1 [CROSS JOIN table 2]|**  **[NATURAL JOIN table2]|**  **[JOIN table2 USING(column\_name)]|**  **[JOIN table2 ON(table1.column\_name=table2.column\_name)]|**  **[LEFT|RIGHT|FULLOUTER JOIN table2 ON(table1.column\_name=table2.column\_name)];** |

## CROSS JOIN：交叉连接，用以产生笛卡尔积

Cross join产生了一个笛卡尔集，其效果等同于在两个表进行连接时未使用WHERE子句限定连接条件;

原则：

* 从两个表中选出同名列的值均对应相等的所有行；
* 如果两个表中同名列的数据类型不同，则出错；
* 不允许在参照列上使用表名或者别名作为前缀；

格式：

|  |
| --- |
| **表1 CROSS JOIN 表2** |

1. 采用笛卡尔积查询emp、dept表；

|  |
| --- |
| SELECT \* FROM emp CROSS JOIN dept; |

## NATURAL JOIN:自然连接，自动找到匹配的关联字符，消除笛卡尔积

格式：

|  |
| --- |
| **表1 NATURAL JOIN 表2** |

1. 查询emp,dept表，消除笛卡尔积；

|  |
| --- |
| SELECT \* FROM emp NATURAL JOIN dept; |

## JOIN … USING(column\_name):用户自定义关联字段，并消除笛卡尔积

如果不希望参照被连接表的所有同名列进行等值连接，自然连接将无法满足要求，可以在连接时使用USING子句来设置用于等值连接的列（参照列）名。

**格式：**

|  |
| --- |
| **表1 JOIN 表2 USING(字段名)** |

1. 查询emp,dept表，以deptno为关联字段，消除笛卡尔积；

|  |
| --- |
| SELECT \* FROM emp JOIN dept USING(deptno); |

## JOIN…ON(…):用户自定义关联条件，消除笛卡尔积

如果要参照非同名的列进行等值连接，或想设置任意的连接条件，可以使用ON子句；

格式：

|  |
| --- |
| **表1 JOIN 表2 ON(关联条件)** |

1. 查询emp,dept表，以emp.deptno=dept.deptno为关联条件，消除笛卡尔积；

|  |
| --- |
| SELECT \* FROM emp JOIN dept ON(emp.deptno=dept.deptno); |

## LEFT|RIGHT|FULLOUTER JOIN…ON(…)设置左、右连接，增加丢失数据；

* **内连接（Inner Join）**

在SQL99规范中，内连接只返回满足连接条件的数据。

* **外连接（Outer Join）包括以下三种类型**

1. **左外联接（Left Outer Join ）**两个表在连接过程中除返回满足连接条件的行以外，还返回左表中不满足条件的行，这种连接称为左外联接。
2. **右外联接（Right Outer Join）**两个表在连接过程中除返回满足连接条件的行以外，还返回右表中不满足条件的行，这种连接称为右外联接。
3. **满外联接（Full Outer Join）**Oracle9i开始新增功能，两个表在连接过程中除返回满足连接条件的行以外，还返回两个表中不满足条件的所有行，这种连接称为满外联接。

**格式：**

|  |
| --- |
| **表1 LEFT JOIN 表2 ON(关联条件)：左连接** |
| **表1 RIGHT JOIN 表2 ON(关联条件)：右连接** |
| **表1 FULLOUTER JOIN 表2 ON(关联条件)：全连接** |

1. 查询emp,dept表，以emp.deptno=dept.deptno为关联条件，消除笛卡尔积，并加上丢失数据；

|  |
| --- |
| SELECT \* FROM emp RIGHT JOIN dept ON(emp.deptno=dept.deptno); |

**注**：多表查询的性能不高，且其性能只有在数据量较大的情况下才有所体现，故多表查询能不使用，就尽量不要使用；

SQL1999语法在所有支持它数据库中都是通用的，故可以以不变应万变；

# 第三节 分组函数及分组查询

## 3.1 分组函数

分组函数对一组数据进行运算，针对一组数据（多行记录）只返回一个结果，也称多行函数；不允许在WHERE 子句中使用分组函数，这与子句运行顺序有关；

常用的统计函数有以下几种：

|  |
| --- |
| COUNT()：求出查询记录数 |
| AVG()：求出平均值 |
| SUM()：求和 |
| MIN()：求出最小值 |
| MAX()：求出最大值 |

**补充：**

1. **常用分组函数**

|  |
| --- |
|  |

1. **count()函数**

* count(\*)返回组中总记录数目；
* count(exp)返回表达式exp值非空的记录数目；
* count(distinct(exp))返回表达式exp值不重复的、非空的记录数目。

**示例：**

|  |
| --- |
| **SELECT COUNT(\*) FROM emp;**  **SELECT COUNT(comm) FROM emp;**  **SELECT COUNT(DISTINCT(deptno)) FROM emp;** |

1. 统计员工的人数、总工资、平均工资、最高工资、最低工资；

|  |
| --- |
|  |

**注：**COUNT() 函数主要负责统计记录数，若在一张表中没有记录数，则COUNT()函数返回 0，即COUNT()函数永远都会返回一个具体的数字；

|  |
| --- |
|  |

但其他函数有可能返回null；

|  |
| --- |
|  |

## 分组统计

讨论问题域：将表中具体的记录按照某种规律（例如按照某字段相同的分为一组）进行分组查询，例如：某公司将员工按照男女分组，统计男女数量；某学校按照年龄分组，18岁以上一组，18岁以下一组，然后统计相关数据；某社团将学生按照地区分组，上海人一组，北京人一组。 分组在什么情况下应用？当数据中存在重复时，运用分组才有意义，否则一个人也可以是一组，但没有意义；

分组查询的语法：

|  |
| --- |
| **SELECT [DISTNCT] \*| 分组字段1[别名] [,分组字段2[别名]…] |统计函数**  **FROM 表名称[别名],[表名称[别名]…]**  **[WHERE 条件（s）]**  **[GROUP BY 分组字段1 [,分组字段2,…]]**  **[ORDER BY 排序字段 ASC|DESC[，排序字段 ASC|DESC…]];** |

1. 按部门分组统计每个部门的员工人数，平均工资；

|  |
| --- |
|  |

1. 按照职位分组，求出每个职位的最高工资和最低工资；

|  |
| --- |
|  |

1. 分别统计有奖金和没有奖金的雇员人数和平均工资；

|  |
| --- |
|  |

**注**：

* 统计函数可以单独使用，但不能同其他字段同时使用，例如：

|  |  |
| --- | --- |
| 正确用法 | SELECT COUNT(empno) FROM emp; |
| 错误用法 |  |

* 若要分组查询的话，则SELECT 子句后面只能跟分组字段和统计函数，不能跟其他字段；例如：

|  |  |
| --- | --- |
| 正确用法 | SELECT job,COUNT(empno),AVG(sal) FROM emp GROUP BY job; |
| 错误用法 |  |

* 统计函数允许嵌套，但嵌套之后的分组查询中不允许有其他字段；例如：按职务分组，查询每个职务平均工资的最大值；

|  |  |
| --- | --- |
| 正确用法 | SELECT MAX(AVG(sal)) FROM emp GROUP BY job; |
| 错误用法 |  |

1. 查询每个部门的名称、部门人数、平均工资；

思路：

1）确定相关联的表：dept,emp;

2）确定关联字段：deptno;

3）查询部门名称、员工编号、员工薪水：

|  |
| --- |
| SELECT d.dname,e.empno,e.sal FROM emp e,dept d WHERE e.deptno=d.deptno; |

4）由于部门名称出现重复，故可以按照部门名称分组进行统计：

|  |
| --- |
| SELECT d.dname,COUNT(e.empno),AVG(e.sal) FROM emp e,dept d  WHERE e.deptno=d.deptno  GROUP BY d.dname; |

5）加上有关联并替换空数据：

|  |
| --- |
| SELECT d.dname,COUNT(e.empno),NVL(AVG(e.sal),0) FROM emp e,dept d  WHERE e.deptno(+)=d.deptno  GROUP BY d.dname; |
|  |

除了根据但字段分组外，分组查询还可以对多字段分组，

1. 查询部门编号、部门名称、部门位置、部门人数、部门平均薪水；

思路：

1. 查找关联表：emp,dept;
2. 查找关联字段：deptno;
3. 查询统计所需要的字段：deptno,dname,loc,empno,sal

|  |
| --- |
|  |

4）从上表可以看出，在字段deptno,dname,loc都有重复，且相对应，故可以对这三个字段分组；

|  |
| --- |
| SELECT d.deptno,d.dname,d.loc,COUNT(e.empno),AVG(e.sal) FROM emp e,dept d  WHERE e.deptno=d.deptno  GROUP BY d.deptno,d.dname,d.loc; |
|  |

5）加上右关联并替换空数据：

|  |
| --- |
| SELECT d.deptno,d.dname,d.loc,COUNT(e.empno),NVL(AVG(e.sal),0) FROM emp e,dept d  WHERE e.deptno(+)=d.deptno  GROUP BY d.deptno,d.dname,d.loc; |
|  |

1. 查询部门编号、部门名称、部门位置、部门人数、部门平均薪水，要求平均薪水高于2000；

|  |
| --- |
| SELECT d.deptno,d.dname,d.loc,COUNT(e.empno),NVL(AVG(e.sal),0) FROM emp e,dept d  WHERE e.deptno(+)=d.deptno AND AVG(e.sal)>2000  GROUP BY d.deptno,d.dname,d.loc; |
|  |

由上图可知在WHERE子句中不能使用分组函数，这是由WHERE子句本身的功能决定的，WHERE子句的功能就是从全部记录中寻找符合条件的记录，而不能从分组中查找；其原理如下：

|  |
| --- |
|  |

## HAVING子句

由上图可知，要想实现在分组数据中再次过滤查找，就要用到HAVING子句，其语法如下：

|  |
| --- |
| **SELECT [DISTNCT] \*| 分组字段1[别名] [,分组字段2[别名]…] |统计函数**  **FROM 表名称[别名],[表名称[别名]…]**  **[WHERE 条件（s）]**  **[GROUP BY 分组字段1 [,分组字段2,…]]**  **[HAVING 分组后的过滤条件（可以使用统计函数）]**  **[ORDER BY 排序字段 ASC|DESC[，排序字段 ASC|DESC…]]** |

故例20可以通过如下语句解决：

|  |
| --- |
| SELECT d.deptno,d.dname,d.loc,COUNT(e.empno),NVL(AVG(e.sal),0) FROM emp e,dept d  WHERE e.deptno(+)=d.deptno  GROUP BY d.deptno,d.dname,d.loc  HAVING AVG(e.sal)>2000; |
|  |

**注：**区别WHERE子句和HAVING子句：

1. WHERE用于分组操作之前的过滤，是从全部数据中筛选出符合条件的数据，WHERE子句中不能用统计函数；
2. HAVING子句用于分组之后的过滤，是从已分组数据中过滤出符合条件的数据，HAVING子句中能够使用统计函数；
3. 综合运用，显示非销售人员工作名称以及从事同一工作雇员的月工资的总和，并且要满足从事同一工作的雇员的月工资合计大于$5000，输出结果按月工资的合计升序排列；

思路：

1. 找出相关联的表：emp
2. 查询相关字段：job,sal，并加入条件判断：

|  |
| --- |
| SELECT job,sal FROM emp WHERE NOT job='SALESMAN'; |

**注**：Oracle中SQL语句中的关键字不区分大小写，但Oracle中的数据区分大小写，所以’SALESMAN’不等于’salesman’；

3）由于job字段有重复数据，故可按照job分组统计，并加入统计函数：

|  |
| --- |
| SELECT job,SUM(sal) FROM emp WHERE NOT job='SALESMAN'  GROUP BY job; |

4）加入对分组后数据的条件判断并排序：

|  |
| --- |
| SELECT job,SUM(sal) FROM emp WHERE NOT job='SALESMAN'  GROUP BY job  HAVING SUM(sal)>5000  ORDER BY SUM(sal) ASC; |
|  |

# 第四节 子查询

子查询是简单查询、限定查询、多表查询、分组查询的综合体，子查询在一定条件下能够替代性能较差的多表查询，故其应用很广；子查询指的是多个查询语句嵌套使用进行数据查询；子查询在主查询前执行一次，主查询使用子查询的结果；

使用子查询注意事项：

* 在查询是基于未知值时应考虑使用子查询；
* 子查询必须包含在括号内；
* 建议将子查询放在比较运算符的右侧，以增强可读性；
* 除非进行Top-N 分析，否则不要在子查询中使用ORDER

BY 子句；

* 对单行子查询使用单行运算符；
* 对多行子查询使用多行运算符；

子查询按照返回数据的多少可以分为单行子查询和多行子查询

1. **单行子查询**

* 单行子查询只返回一行记录；
* 对单行子查询可使用单行记录比较运算符；

**注：**子查询空值/多值问题

* 如果子查询未返回任何行，则主查询也不会返回任何结果；
* 如果子查询返回单行结果，则为单行子查询，可以在主查询中对其使用相应的单行记录比较运算符；
* 如果子查询返回多行结果，则为多行子查询，此时不允许对其使用单行记录比较运算符；

**2) 多行子查询**

* 多行子查询返回多行记录;
* 对多行子查询只能使用多行记录比较运算符;

|  |
| --- |
|  |

**3）TopN查询分析**：查询前n条记录

在ORACLE中通常采用子查询的方式来实现TOP N查询

**语法格式：**

|  |
| --- |
| **SELECT 字段列表**  **FROM (*SELECT* 字段列表*FROM table ORDER BY* 排序字段*)***  **WHERE ROWNUM <= n；** |

**示例：**

|  |
| --- |
| **select \***  **from (*select \* from emp order by sal desc*)**  **where rownum <= 5;** |

**子查询语法格式：**

|  |
| --- |
| **SELECT [DISTNCT] \*| 分组字段1[别名] [,分组字段2[别名]…] |统计函数|**  **[ SELECT [DISTNCT] \*| 分组字段1[别名] [,分组字段2[别名]…] |统计函数**  **FROM 表名称[别名],[表名称[别名]…]**  **[WHERE 条件（s）]**  **[GROUP BY 分组字段1 [,分组字段2,…]]**  **[HAVING 分组后的过滤条件（可以使用统计函数）]**  **[ORDER BY 排序字段 ASC|DESC[，排序字段 ASC|DESC…]]]**  **FROM 表名称[别名],[表名称[别名]…]**  **[, SELECT [DISTNCT] \*| 分组字段1[别名] [,分组字段2[别名]…] |统计函数**  **FROM 表名称[别名],[表名称[别名]…]**  **[WHERE 条件（s）]**  **[GROUP BY 分组字段1 [,分组字段2,…]]**  **[HAVING 分组后的过滤条件（可以使用统计函数）]**  **[ORDER BY 排序字段 ASC|DESC[，排序字段 ASC|DESC…]]]**  **[WHERE 条件（s）**  **[,SELECT [DISTNCT] \*| 分组字段1[别名] [,分组字段2[别名]…] |统计函数**  **FROM 表名称[别名],[表名称[别名]…]**  **[WHERE 条件（s）]**  **[GROUP BY 分组字段1 [,分组字段2,…]]**  **[HAVING 分组后的过滤条件（可以使用统计函数）]**  **[ORDER BY 排序字段 ASC|DESC[，排序字段 ASC|DESC…]]]]**  **[GROUP BY 分组字段1 [,分组字段2,…]]**  **[HAVING 分组后的过滤条件（可以使用统计函数）]**  **[ORDER BY 排序字段 ASC|DESC[，排序字段 ASC|DESC…]]** |

理论上讲，子查询可以放在查询语句的任意位置，但在通常运用中，放在FROM子句和WHERE子句中的情况比较多。

### 经验总结：

1. 子查询放在WHERE子句中，一般返回单行单列数据、多行单列数据、或单行多列数据；
2. 子查询放在FROM子句中，则返回多行多列数据，当做临时表使用；

## 4.1 WHERE子句中的子查询

### 4.1.1单行子查询：单行单列数据

1. 查询出工资比SMITH高的全部雇员信息；

思路：

1）先查询出SMITH的工资：

|  |
| --- |
| SELECT sal FROM emp WHERE ename='SMITH'; |

2）将上述查询作为判断条件放入WHERE子句：

|  |
| --- |
| SELECT \* FROM emp  WHERE sal>(  SELECT sal FROM emp  WHERE ename='SMITH'); |
|  |

1. 查询出高于公司雇员平均工资的全部雇员信息；

思路：

1）查询出公司雇员的平均工资：

|  |
| --- |
| SELECT AVG(sal) FROM emp; |

2）将上述查询作为查询条件放入WHERE子句：

|  |
| --- |
| SELECT \* FROM emp  WHERE sal>(  SELECT AVG(sal) FROM emp); |
|  |

### 4.1.2单行子查询：单行多列数据

以上两例为子查询返回单行单列（即一条记录的一个字段），也可以返回单行多列数据（即一条记录），但比较少见；

1. 子查询返回单行多列数据；

|  |
| --- |
| SELECT \* FROM emp  WHERE (job,sal)=(  SELECT job,sal  FROM emp  WHERE ename='ALLEN'); |
|  |

若子查询返回数据为多行多列数据，则需要用到判断符进行判断：IN,ANY,ALL

### 4.1.3 IN操作符：用于指定一个子查询的范围判断

格式：该用法与IN(范围)用法相似，只是范围是由子查询语句指定；

|  |
| --- |
| **IN(子查询语句)** |

1. 查询所有职务为经理的员工记录；

|  |
| --- |
| SELECT \* FROM emp WHERE sal IN(SELECT sal FROM emp WHERE job='MANAGER'); |
|  |

**问题：**在IN中可以有null值，但当NOT IN中出现null值时，却查询不到数据？

**原因：**NOT IN中出现null值，表示查询全部记录，但如果数据量很大时，系统有可能出项异常，处于安全考虑，Oracle设置NOT IN中出现null值，表示不查询任何数据；

### 4.1.4 ANY操作符：与每一个数据匹配，有三种匹配形式

* =ANY：与IN完全等效；

|  |
| --- |
| SELECT \* FROM emp WHERE sal =ANY(SELECT sal FROM emp WHERE job='MANAGER'); |
|  |

* >ANY：比子查询返回值中最小值大的记录；有时可用MIN()函数替代；

|  |
| --- |
| SELECT \* FROM emp WHERE sal >ANY(SELECT sal FROM emp WHERE job='MANAGER'); |
|  |

* <ANY：比子查询返回值中最大值小的记录；有时可以用MAX()替代；

|  |
| --- |
| SELECT \* FROM emp WHERE sal <ANY(SELECT sal FROM emp WHERE job='MANAGER'); |
|  |

### 4.1.5 ALL操作符：与每个数据匹配，有两种形式

* >ALL:比子查询中返回的最大值还大；有时可以用MAX()替代；

|  |
| --- |
| SELECT \* FROM emp WHERE sal >ALL(SELECT sal FROM emp WHERE job='MANAGER'); |
|  |

* <ALL:比子查询中返回的最小值还小；有时可用MIN()函数替代；

|  |
| --- |
| SELECT \* FROM emp WHERE sal <ALL(SELECT sal FROM emp WHERE job='MANAGER'); |
|  |

## 4.2 FROM子句中的子查询

FROM子句中的子查询一般都返回多行多列数据，作为一张临时表去使用；

1. 查询部门编号、名称、位置、部门人数、平均工资；

方法一：采用统计函数进行分组查询：

|  |
| --- |
| SELECT d.deptno,d.dname,d.loc,COUNT(e.empno),AVG(e.sal)  FROM dept d,emp e  WHERE d.deptno=e.deptno(+)  GROUP BY d.deptno,d.dname,d.loc; |
|  |

方法二：采用子查询：由于统计函数不能与其他字段共同使用，故将统计函数放在子查询中：

|  |
| --- |
| SELECT dept.deptno,dept.dname,dept.loc,temp.count,temp.avg  FROM dept,  (SELECT deptno,COUNT(emp.empno) count,AVG(emp.sal) avg  FROM emp GROUP BY deptno) temp  WHERE dept.deptno=temp.deptno(+); |
|  |

对比上述两种方法，在方法一中，运用多表查询，匹配次数是：4\*14=56次；在方法二中，运用子查询，匹配次数是：14+4\*3=26次（**注**：分组查询也是逐个匹配）；由此可见，对于相同的问题，运用子查询要比运用多表查询性能更好，尤其在大数据量情况下，性能差更加突出；

### 经验总结：

如果最终查询结果之中需要出现SELECT子句，但是又不能直接使用统计函数的时候，就在子查询中统计信息，即：又复杂统计的地方大部分需要子查询；

## Oracle的分页查询

### 4.3.1方法一：通过ROWNUM分页

|  |
| --- |
| **SELECT temp.\*,ROWNUM row\_num**  **FROM(SELECT \* FROM emp) temp**  **WHERE ROWNUM<=10;** |
|  |

**注**：oracle规定ROWNUM只能用一次，若上述语句中再加一个关于ROWNUM的条件，则无法查询；

|  |
| --- |
|  |

若要限制上述范围，还需再用一次子查询；

|  |
| --- |
| **SELECT \***  **FROM(SELECT temp.\*,ROWNUM AS row\_num**  **FROM(SELECT \* FROM emp) temp**  **WHERE ROWNUM<=10)**  **WHERE row\_num>=6;** |
|  |

**注**：

* 上例中， ROWNUM在内层语句中，不能使用别名，但可以定义别名，此别名可以在外层查询中使用；
* 若要修改显示的字段，只需要修改最里层子查询的字段即可；

|  |
| --- |
| **SELECT \***  **FROM(SELECT temp.\*,ROWNUM AS row\_num**  **FROM(SELECT empno,ename,sal FROM emp) temp**  **WHERE ROWNUM<=10)**  **WHERE row\_num>=6;** |
|  |

* 也可以用ORDER BY子句为数据排序；

### 4.3.2方法二：通过ROWID

|  |
| --- |
| **SELECT \* FROM emp**  **WHERE ROWID**  **IN**  **(SELECT rid**  **FROM**  **(SELECT ROWNUM rn,rid**  **FROM**  **(SELECT ROWID rid,empno**  **FROM emp**  **ORDER BY empno DESC)**  **WHERE ROWNUM<=10)**  **WHERE rn>=6)**  **ORDER BY empno DESC;** |
|  |

分析：

* 查询表的ROWID和empno

|  |
| --- |
|  |

* 查询前10条记录的ROWID和ROWNUM

|  |
| --- |
| **SELECT ROWNUM rn,rid**  **FROM**  **(SELECT ROWID rid,empno**  **FROM emp**  **ORDER BY empno DESC)**  **WHERE ROWNUM<=10;** |
|  |

* 查询第6到第10条记录

|  |
| --- |
| **SELECT rid**  **FROM**  **(SELECT ROWNUM rn,rid**  **FROM**  **(SELECT ROWID rid,empno**  **FROM emp**  **ORDER BY empno DESC)**  **WHERE ROWNUM<=10)**  **WHERE rn>=6;** |
|  |

* 通过ROWID查询记录；

|  |
| --- |
| **SELECT \* FROM emp**  **WHERE ROWID**  **IN**  **(SELECT rid**  **FROM**  **(SELECT ROWNUM rn,rid**  **FROM**  **(SELECT ROWID rid,empno**  **FROM emp**  **ORDER BY empno DESC)**  **WHERE ROWNUM<=10)**  **WHERE rn>=6)**  **ORDER BY empno DESC;** |

**注**：此方法的速度是最快的，因为查询ROWID是记录的索引，先找出相应的索引，然后根据索引查找数据，这比通过子查询直接查找数据的速度快得多；ROWNUM用来确定记录的范围；

### 4.3.3方法三：通过分析函数

|  |
| --- |
| **SELECT \***  **FROM**  **(SELECT e.\* ,ROW\_NUMBER() OVER(ORDER BY empno DESC) rk**  **FROM emp e)**  **WHERE rk<=10**  **AND rk>=6;** |
|  |

**注**：此方法的速度最慢；

## 通过子查询结果创建表

|  |
| --- |
| **CREATE TABLE myemp(empno,ename,sal,job)**  **AS (SELECT empno,ename,sal,job FROM emp);** |
|  |

通过这种方式不仅可以创建表结构，还可以实现数据的转移；

## 在更新语句中使用子查询

在更新语句中，同样可以使用子查询更新数据：语法如下：

|  |
| --- |
| **UPDATE SET(字段1，字段2，…)=(子查询)**  **WHERE (查询条件);** |

1. 更新用户名为‘SCOTT’的数据，让其薪水、工作、奖金同‘SMITH’一样；

|  |
| --- |
| **UPDATE emp SET(job,sal,comm)=(**  **SELECT job,sal,comm**  **FROM emp**  **WHERE ename='SMITH')**  **WHERE ename='SCOTT';** |