* SQL(基础查询)

SQL（关联查询）

**1. SQL(基础查询)**

**1.1. 基本查询语句**

**1.1.1. FROM子句**

SQL查询语句的语法如下：

01.SELECT <\*, column [alias], …> FROM table;

其中：SELECT用于指定要查询的列，FROM指定要从哪个表中查询。如果要查询所有列，可以在SELECT后面使用\*号，如果只查询特定的列，可以直接在SELECT后面指定列名，列名之间用逗号隔开。例句如下，查询dept表中的所有记录：

01.SELECT \* FROM dept;

**1.1.2. 使用别名**

在SQL语句中可以通过使用列的别名改变标题的显示样式，或者表示计算结果的含义，使用语法是列的别名跟在列名后，中间可以加或不加一个“AS”关键字。例如：

01.SELECT empno AS id ,ename "Name", sal \* 12 "Annual Salary" FROM emp;

别名可以直接写，不必用双引号引起来。但是如果希望别名中区分大小写字符，或者别名中包含字符或空格，则必须用双引号引起来。

**1.1.3. WHERE子句**

在SELECT语句中，可以在WHERE子句中使用比较操作符限制查询结果，是可选的。

当查询条件中和数字比较，可以使用单引号引起，也可以不用，当和字符及日期类型的数据比较，则必须用单引号引起。例如查询部门10下的员工信息：

01.SELECT \* FROM empWHERE deptno = 10;

查询职员表中职位是’SALESMAN’的职员：

01.SELECT ename, sal, job FROM emp WHERE job = 'SALESMAN';

**1.1.4. SELECT子句**

如果只查询表的部分列，需要在SELECT后指定列名，例如：

01.SELECT empno, ename, sal, job FROM emp;

**1.2. 查询条件**

**1.2.1.** 使用>, <, >=, <=, !=, <>, =

在WHERE子句中的查询条件，可以使用比较运算符来做查询。比如：查询职员表中薪水低于2000元的职员信息：

01.SELECT ename, sal FROM emp WHERE sal< 2000;

查询职员表中不属于部门10的员工信息（!=等价于<>）：

01. SELECT ename, sal, job FROM emp WHERE deptno != 10;

查询职员表中在2002年1月1号以后入职的职员信息，比较日期类型数据：

01. SELECT ename, sal, hiredate FROM emp

02. WHERE hiredate>to\_date('2002-1-1','YYYY-MM-DD');

**1.2.2. 使用AND，OR关键字**

02.WHERE sal> 1000 AND job = 'CLERK';

查询薪水大于1000或者职位是’CLERK’的职员信息：

01.SELECT ename, sal, job FROM emp

02.WHERE sal> 1000 OR job = 'CLERK';

**1.2.3. 使用LIKE条件（模糊查询）**

当用户在执行查询时，不能完全确定某些信息的查询条件，或者只知道信息的一部分，可以借助LIKE来实现模糊查询。LIKE需要借助两个通配符：

•%：表示0到多个字符

•\_：标识单个字符

这两个通配符可以配合使用，构造灵活的匹配条件。例如查询职员姓名中第二个字符是‘A’的员工信息：

01.SELECT ename, job FROM emp WHERE ename LIKE '\_A%';

**1.2.4. 使用IN和NOT IN**

在WHERE子句中可以用比较操作符IN(list)来取出符合列表范围中的数据。其中的参数list表示值列表，当列或表达式匹配于列表中的任何一个值时，条件为TRUE，该条记录则被显示出来。

IN页可以理解为一个范围比较操作符，只不过这个范围是一个指定的值列表，NOT IN(list) 取出不符合此列表中的数据记录。例如查询职位是MANAGER或者CLERK的员工：

01.SELECT ename, job FROM emp WHERE job IN ('MANAGER', 'CLERK');

查询不是部门10或20的员工：

01.SELECT ename, job FROM emp WHERE deptno NOT IN (10, 20);

**1.2.5. BETWEEN…AND…**

BETWEEN…AND…操作符用来查询符合某个值域范围条件的数据，最常见的是使用在数字类型的数据范围上，但对字符类型和日期类型数据也同样适用。例如查询薪水在1500-3000之间的职员信息：

01. SELECT ename, sal FROM emp

02. WHERE sal BETWEEN 1500 AND 3000;

**1.2.6. 使用IS NULL和IS NOT NULL**

空值NULL是一个特殊的值，比较的时候不能使用”=”号，必须使用IS NULL，否则不能得到正确的结果。例如查询哪些职员的奖金数据为NULL：

01.SELECT ename, sal, comm FROM emp

02.WHERE comm IS NULL;

**1.2.7. 使用ANY和ALL条件**

在比较运算符中，可以出现ALL和ANY，表示“全部”和“任一”，但是ALL和ANY不能单独使用，需要配合单行比较操作符>、>=、<、<=一起使用。其中：

•> ANY : 大于最小

•< ANY：小于最大

•> ALL：大于最大

•< ALL：小于最小

例如，查询薪水比职位是“SALESMAN”的人高的员工信息，比任意一个SALESMAN高都行：

01.SELECT empno, ename, job, sal, deptno

02.FROM emp

03.WHERE sal> ANY (

04.SELECT sal FROM emp WHERE job = 'SALESMAN');

**1.2.8. 查询条件中使用表达式和函数**

当查询需要对选出的字段进行进一步计算，可以在数字列上使用算术表达式(+、-、\*、/)。表达式符合四则运算的默认优先级，如果要改变优先级可以使用括号。

算术运算主要是针对数字类型的数据，对日期类型的数据可以做加减操作，表示在一个日期值上加或减一个天数。

查询条件中使用字符串函数UPPER，将条件中的字符串变大写后再参与比较：

01.SELECT ename, sal, job FROMempWHERE ename = UPPER('rose');

查询条件中使用算数表达式，查询年薪大于10w元的员工记录：

01.SELECT ename, sal, job FROM empWHERE sal \* 12 >100000;

**1.2.9. 使用DISTINCT过滤重复**

数据表中有可能存储相同数据的行，当执行查询操作时，默认情况会显示所有行，不管查询结果是否有重复的数据。当重复数据没有实际意义，经常会需要去掉重复值，使用DISTINCT实现。例如查询员工的部门编码，包含所有重复值：

01.SELECT deptno FROM emp;

查询员工的部门编码，去掉重复值：

01.SELECT DISTINCT deptno FROM emp;

DISTINCT后面的列可以组合查询，下例查询每个部门的职位，去掉重复值。注意是deptno和job联合起来不重复：

01.SELECT DISTINCT deptno, job FROM emp;

**1.3. 排序**

**1.3.1. 使用ORDER BY字句**

对查询出的数据按一定规则进行排序操作，使用ORDER BY子句。语法如下：

注意，ORDER BY必须出现在SELECT中的最后一个子句。下例对职员表按薪水排序：

01.SELECT ename, sal

02.FROM emp

03.ORDER BY sal;

**1.3.2. ASC和DESC**

排序时默认按升序排列，即由小及大，ASC用来指定升序排序，DESC用来指定降序排序。

因为NULL值视作最大，则升序排列时，排在最后，降序排列时，排在最前。如果不写ASC或DESC，默认是ASC，升序排列。例如，按员工的经理升序排序：

01. SELECT empno, ename, mgr FROM emp

02. WHERE deptno = 10 ORDER BY mgr;

降序排列，必须指明，按员工的薪水倒序排序：

01. SELECT ename, sal FROM emp

02. ORDER BY sal DESC;

**1.3.3. 多个列排序**

当以多列作为排序标准时，首先按照第一列进行排序，如果第一列数据相同，再以第二列排序，以此类推。多列排序时，不管正序还是倒序，每个列需要单独设置排序方式。

01. SELECT ename, deptno, sal FROM emp

02. ORDER BY deptno ASC, sal DESC;

**1.4. 聚合函数**

**1.4.1. 什么是聚合函数**

查询时需要做一些数据统计，比如：查询职员表中各部门职员的平均薪水，各部门的员工人数。当需要统计的数据并不能在职员表里直观列出，而是需要根据现有的数据计算得到结果，这种功能可以使用聚合函数来实现，即：将表的全部数据划分为几组数据，每组数据统计出一个结果。

因为是多行数据参与运算返回一行结果，也称作分组函数、多行函数、集合函数。用到的关键字：

•GOURP BY 按什么分组

•HAVING 进一步限制分组结果

1.4.2. MAX和MIN

用来取得列或表达式的最大、最小值，可以用来统计任何数据类型，包括数字、字符和日期。例如获取机构下的最高薪水和最低薪水，参数是数字：

01.SELECT MAX(sal) max\_sal, MIN(sal) min\_sal

02.FROM emp;

计算最早和最晚的入职时间，参数是日期：

01.SELECT MAX(hiredate) max\_hire, MIN(hiredate) min\_hire

02.FROM emp;

**1.4.3. AVG和SUM**

AVG和SUM函数用来统计列或表达式的平均值和和值，这两个函数只能操作数字类型，并忽略NULL值。例如获得机构下全部职员的平均薪水和薪水总和：

01.SELECT AVG(sal) avg\_sal, SUM(sal) sum\_sal FROM emp;

**1.4.4. COUNT**

COUNT函数用来计算表中的记录条数，同样忽略NULL值。例如获取职员表中一共有多少名职员记录：

01.SELECT COUNT(\*) total\_num FROM emp;

获得职员表中有多少人是有职位的（忽略没有职位的员工记录）

01. SELECT COUNT(job) total\_job FROM emp;

**1.4.5. 聚合函数对空值的处理**

聚合函数忽略NULL值。即当emp表中的某列有NULL值，比如某新入职员工没有薪水，比较两条语句的结果：

01.SELECT AVG(sal) avg\_sal FROM emp;

02.SELECT AVG(NVL(sal,0)) avg\_sal FROM emp;

**1.5. 分组**

**1.5.1. GROUP BY子句**

上面的例子都是以整个表作为一组。如果希望得到每个部门的平均薪水，而不是整个机构的平均薪水，需要把整个数据表按部门划分成一个个小组，每个小组中包含一行或多行数据，在每个小组中再使用分组函数进行计算，每组返回一个结果。语法如下：

01.SELECT <\*, column [alias], …>

02.FROM table [WHERE condition(s)]

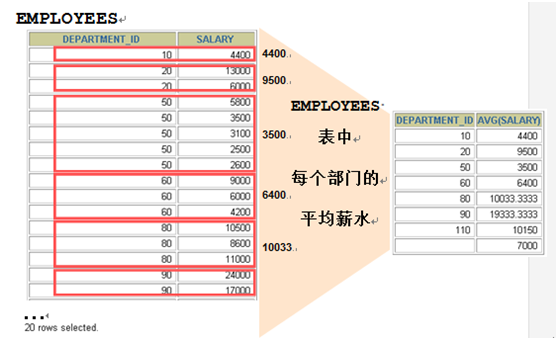
03.[GROUP BY group\_by\_expression]

04.[HAVING group\_condition]

05.[ORDER BY column [ASC | DESC]] ;

其中划分的小组有多少，最终的结果集行数就有多少。

**1.5.2. 分组查询**



**1.5.3. HAVING字句**

HAVING子句用来对分组后的结果进一步限制，比如按部门分组后，得到每个部门的最高薪水，可以继续限制输出结果。必须跟在GROUP BY后面，不能单独存在。例如查询每个部门的最高薪水，只有最高薪水大于4000的记录才被输出显示：

01.SELECT deptno, MAX(sal) max\_sal FROM emp

02.GROUP BY deptno HAVING MAX(sal) >4000;

**1.6. 查询语句的执行顺序**

当一条查询语句中包含所有的子句，执行顺序依下列子句次序：

1.FROM 子句：执行顺序为从后往前、从右到左。数据量较少的表尽量放在后面。

2.WHERE子句：执行顺序为自下而上、从右到左。将能过滤掉最大数量记录的条件写在WHERE 子句的最右。

3.GROUP BY：执行顺序从左往右分组，最好在GROUP BY前使用WHERE将不需要的记录在GROUP BY之前过滤掉。

4.HAVING 子句：消耗资源。尽量避免使用，HAVING 会在检索出所有记录之后才对结果集进行过滤，需要排序等操作。

5.SELECT子句：少用\*号，尽量取字段名称。ORACLE 在解析的过程中, 通过查询数据字典将\*号依次转换成所有的列名, 消耗时间。

6.ORDER BY子句：执行顺序为从左到右排序，消耗资源。

**2.1. 关联基础**

**2.1.1. 关联的概念**

实际应用中所需要的数据，经常会需要查询两个或两个以上的表。这种查询两个或两个以上数据表或视图的查询叫做连接查询，连接查询通常建立在存在相互关系的父子表之间。语法如下：

01.SELECT table1.column, table2.column

02.FROM table1, table2

03.WHERE table1.column1 = table2.column2;

或者：

01.SELECT table1.column, table2.column

02.FROM table1JOIN table2

03.ON(table1.column1 = table2.column2);

**2.1.2. 笛卡尔积**

笛卡尔积指做关联操作的每个表的每一行都和其它表的每一行做组合，假设两个表的记录条数分别是X和Y，笛卡尔积将返回X \* Y条记录。当两个表关联查询时，不写连接条件，得到的结果即是笛卡尔积。例如：

01.SELECT COUNT(\*) FROM emp; --14条记录

02.SELECT COUNT(\*) FROM dept; --4条记录

03.SELECT emp.ename, dept.dnameFROM emp, dept;--56条记录

**2.1.3. 等值连接**

等值连接是连接查询中最常见的一种，通常是在有主外键关联关系的表间建立，并将连接条件设定为有关系的列，使用等号”=”连接相关的表。例如查询职员的姓名、职位以及所在部门的名字和所在城市，使用两个相关的列做等值操作：

01.SELECT e.ename, e.job, d.dname, d.loc

02.FROM emp e, dept d

03.WHERE e.deptno = d.deptno;

**2.2. 关联查询**

**2.2.1. 内连接**

内连接返回两个关联表中所有满足连接条件的记录。例如查询员工的名字和所在部门的名字：

01.SELECT e.ename, d.dname

02.FROM emp e, dept d

03.WHERE e.deptno = d.deptno

上面的语法也可以写为：

01.SELECT e.ename, d.dname

02.FROM emp e JOIN dept d

03.ON(e.deptno = d.deptno);

**2.2.2. 外连接**

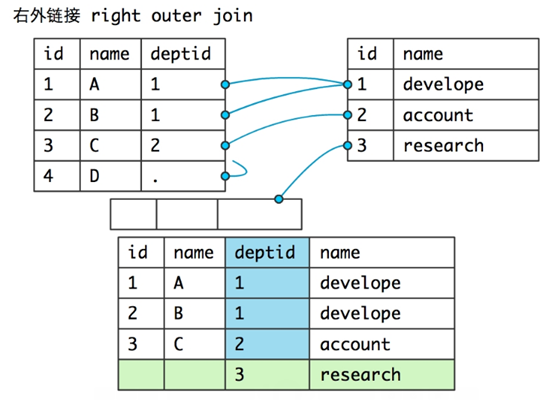
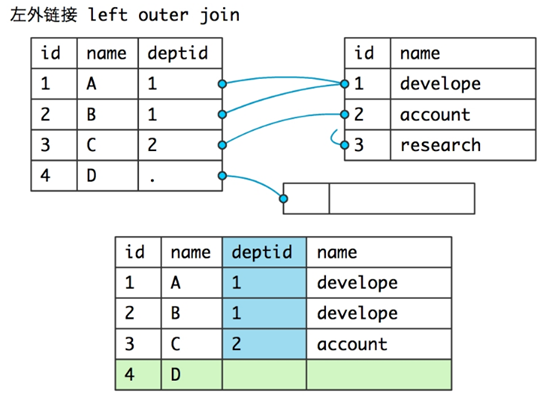
内连接返回两个表中所有满足连接条件的数据记录，在有些情况下，需要返回那些不满足连接条件的记录，需要使用外连接，即不仅返回满足连接条件的记录，还将返回不满足连接条件的记录。比如把没有职员的部门和没有部门的职员查出来。外连接的语法如下：

01.SELECT table1.column, table2.column

02.FROM table1 [LEFT | RIGHT | FULL] JOIN table2

03.ON table1.column1 = table2.column2;

了解驱动表的概念。



* 外连接查询的例子,Emp表做驱动表:

01.SELECT e.ename, d.dname

02.FROM emp e LEFT OUTER JOIN dept d

03.ON e.deptno = d.deptno;

* Dept表做驱动表:

01.SELECT e.ename, d.dname

02.FROM emp e RIGHT OUTER JOIN dept d

03.ON e.deptno = d.deptno;

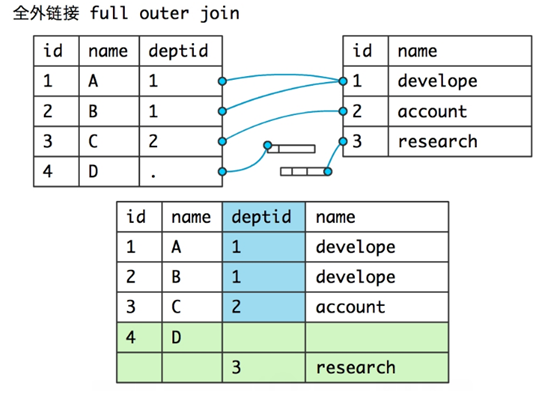
**2.2.3. 全连接**

全外连接是指除了返回两个表中满足连接条件的记录，还会返回不满足连接条件的所有其它行。即是左外连接和右外连接查询结果的总和。例如：

01.SELECT e.ename, d.dname

02.FROM emp e FULL OUTER JOIN dept d

03.ON e.deptno = d.deptno;



**2.2.4. 自连接**

自连接是一种特殊的连接查询，数据的来源是一个表，即关联关系来自于单表中的多个列。表中的列参照同一个表中的其它列的情况称作自参照表。

自连接是通过将表用别名虚拟成两个表的方式实现，可以是等值或不等值连接。例如查出每个职员的经理名字，以及他们的职员编码：

01.SELECT worker.empnow\_empno, worker.enamew\_ename, manager.empnom\_empno, manager.enamem\_ename

02.FROM emp worker join emp manager

03.ON worker.mgr = manager.empno;

* SQL(高级查询)

**1.1. 子查询**

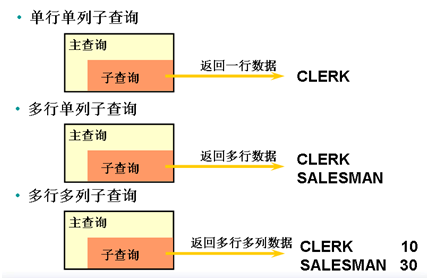
**1.1.1. 子查询在WHERE子句中**

在SELECT查询中，在WHERE查询条件中的限制条件不是一个确定的值，而是来自于另外一个查询的结果。为了给查询提供数据而首先执行的查询语句叫做子查询。

子查询：嵌入在其它SQL语句中的SELECT语句，大部分时候出现在WHERE子句中。子查询嵌入的语句称作主查询或父查询。主查询可以是SELECT语句，也可以是其它类型的语句比如DML或DDL语句。

根据返回结果的不同，子查询可分为单行子查询、多行子查询及多列子查询。

* **图示:**



例如查找和SCOTT同职位的员工：

01.SELECT e.ename, e.job

02.FROM emp e

03.WHERE e.job =

04. (SELECT job FROM emp WHERE ename = 'SCOTT')；

如果子查询返回多行，主查询中要使用多行比较操作符，包括IN、ALL、ANY。其中ALL和ANY不能单独使用，需要配合单行比较操作符>、>=、<、<=一起使用。

例如查询出部门中有SALESMAN但职位不是SALESMAN的员工的信息：

01.SELECT empno, ename, job, sal, deptno

02.FROM emp

03.WHERE deptno IN

04.(SELECT deptno FROM emp WHERE job = 'SALESMAN')

05.AND job <> 'SALESMAN';

在子查询中需要引用到主查询的字段数据，使用EXISTS关键字。EXISTS后边的子查询至少返回一行数据，则整个条件返回TRUE。如果子查询没有结果，则返回FALSE。例如列出来那些有员工的部门信息：

01.SELECT deptno, dname FROM dept d

02.WHERE EXISTS

03. (SELECT \* FROM emp e

04. WHERE d.deptno = e.deptno);

**1.1.2. 子查询在HAVING部分**

子查询不仅可以出现在WHERE子句中，还可以出现在HAVING部分。

例如查询列出最低薪水高于部门30的最低薪水的部门信息：

01.SELECT deptno, MIN(sal) min\_sal

02.FROM emp

03.GROUP BY deptno

04.HAVING MIN(sal) >

05.(SELECT MIN(sal) FROM emp WHERE deptno = 30);

**1.1.3. 子查询在FROM部分**

在查询语句中，FROM子句用来指定要查询的表。如果要在一个子查询的结果中继续查询，则子查询出现在FROM 子句中，这个子查询也称作行内视图或者匿名视图。这时，把子查询当作视图对待，但视图没有名字，只能在当前的SQL语句中有效。

查询出薪水比本部门平均薪水高的员工信息：

01.SELECT e.deptno, e.ename, e.sal

02.FROM emp e,

03.(SELECT deptno, AVG(sal) avg\_sal FROM emp GROUP BY deptno) x

04.WHERE e.deptno = x.deptno

05.ande.sal>x.avg\_sal

06.ORDER BY e.deptno;

**1.1.4. 子查询在SELECT部分**

把子查询放在SELECT子句部分，可以认为是外连接的另一种表现形式，使用更灵活：

01.SELECT e.ename, e.sal, e.deptno,

02. (SELECT d.deptno FROM dept d

03. WHERE d.deptno = e.deptno) deptno

04.FROM emp e;

**1.2. 分页查询**

**1.2.1. ROWNUM**

ROWNUM被称作伪列，用于返回标识行数据顺序的数字。例如：

01.SELECT ROWNUM, empno, ename, sal

02.FROM emp;

ROWNUM只能从1计数，不能从结果集中直接截取。

下面的查询语句将没有结果：

01.SELECT ROWNUM, empno, ename, sal

02.FROM emp

03.WHERE rownum> 3;

如果利用ROWNUM截取结果集中的部分数据，需要用到行内视图：

01.SELECT \* FROM

02. (SELECT ROWNUM rn , e.\* FROM emp e )

03. WHERE rn BETWEEN 8 AND 10;

也就是将ROWNUM先作为行内视图的一个列，在主查询中就可以使用这个列值作为条件。

**1.2.2. 使用子查询进行分页**

分页策略是指每次只取一页的数据。当每次换页，取下一页的数据。在ORACLE中利用ROWNUM的功能来进行分页。

假设结果集共105条，每20条分为一页，则共6页：

Page1： 1 至 20

Page2： 21 至40

…

PageN: (n - 1) \* pageSize + 1 至 n \* pageSize

**1.2.3. 分页与ORDER BY**

按薪水倒序排列，取出结果集中第8到第10条的记录：

01.SELECT \* FROM

02. (SELECT ROWNUM rn , t.\* FROM

03. (SELECT empno,ename,sal FROM emp

04. ORDER BY sal DESC) t

05. )

06. WHERE rn BETWEEN 8 AND 10;

根据要查看的页数，计算起点值（(n - 1) \* pageSize + 1）和终点值（n \* pageSize），替换掉BETWEEN和AND的参数，即得到当前页的记录。

**1.3. DECODE函数**

**1.3.1. DECODE函数基本语法**

DECODE函数的语法如下：

DECODE (expr, search1, result1[, search2, result2…][, default])

它用于比较参数expr的值，如果匹配到哪一个search条件，就返回对应的result结果，可以有多组search和result的对应关系，如果任何一个search条件都没有匹配到，则返回最后default的值。default参数是可选的，如果没有提供default参数值，当没有匹配到时，将返回NULL。

查询职员表，根据职员的职位计算奖励金额，当职位分别是’MANAGER’、’ANALYST’、’SALESMAN’时，奖励金额分别是薪水的1.2倍、1.1倍、1.05倍，如果不是这三个职位，则奖励金额取薪水值：

01.SELECT ename, job, sal,

02.DECODE(job, 'MANAGER', sal \* 1.2,

03. 'ANALYST', sal \* 1.1,

04. 'SALESMAN', sal \* 1.05,

05.sal

06. ) bonus

07.FROM emp;

和DECODE函数功能相似的有CASE语句，实现类似于if-else的操作。

01.SELECT ename, job, sal,

02. CASE job WHEN 'MANAGER' THEN sal \* 1.2

03. WHEN 'ANALYST' THEN sal \* 1.1

04. WHEN 'SALESMAN' THEN sal \* 1.05

05. ELSE sal END

06.bonus

07.FROM emp;

**1.3.2. DECODE函数在分组查询中的应用**

DECODE函数可以按字段内容分组，例如：计算职位的人数，analyst/manager职位属于vip，其余是普通员工operation,这种功能无法用GROUP BY简单实现。用decode的实现方式：

01.SELECT DECODE(job, 'ANALYST', 'VIP',

02. 'MANAGER', 'VIP',

03. 'OPERATION') job,

04.COUNT(1) job\_cnt

05.FROM emp

06.GROUP BY DECODE(job, 'ANALYST', 'VIP', 'MANAGER', 'VIP', 'OPERATION');

DECODE函数也可以按字段内容排序，例如：Dept表中按”研发部”、“市场部”、“销售部”排序，用普通的select语句，无法按照字面数据排序：

01.SELECT deptno, dname, loc

02.FROM dept

03.ORDER BY

04. DECODE(dname, '研发部',1,'市场部',2,'销售部',3), loc;

**1.4. 排序函数**

**1.4.1. ROW\_NUMBER**

ROW\_NUMBER语法如下：

01.ROW\_NUMBER() OVER(

02. PARTITION BY col1 ORDER BY col2)

表示根据col1分组，在分组内部根据col2排序。此函数计算的值就表示每组内部排序后的顺序编号，组内连续且唯一。

ROWNUM是伪列， ROW\_NUMBER功能更强，可以直接从结果集中取出子集。

* 场景：按照部门编码分组显示，每组内按职员编码排序，并赋予组内编码

01.SELECT deptno, ename, empno,

02. ROW\_NUMBER()OVER

03. (PARTITION BY deptno ORDER BY empno) AS emp\_id

04. FROM emp;

**1.4.2. RANK**

RANK函数的语法如下：

01.RANK() OVER(

02. PARTITION BY col1 ORDER BY col2)

表示根据col1分组，在分组内部根据col2给予等级标识，即排名，相同的数据返回相同排名。特点是跳跃排序，如果有相同数据，则排名相同，比如并列第二，则两行数据都标记为2，但下一位将是第四名。

和ROW\_NUMBER的区别是有结果有重复值，而ROW\_NUMBER没有。

* 场景：按照部门编码分组，同组内按薪水倒序排序，相同薪水则按奖金数正序排序，并给予组内等级，用Rank\_ID表示

01.SELECT deptno, ename, sal, comm,

02. RANK() OVER (PARTITION BY deptno

03. ORDER BY sal DESC, comm) "Rank\_ID"

04. FROM emp；

**1.4.3. DENSE\_RANK**

DENSE\_RANK函数的语法如下：

01.DENSE\_RANK() OVER(

02. PARTITION BY col1 ORDER BY col2)

表示根据col1分组，在分组内部根据col2给予等级标识，即排名，相同的数据返回相同排名。特点是连续排序，如果有并列第二，下一个排序将是三，这一点是和RANK的不同，RANK是跳跃排序。

* 场景：关联emp和dept表，按照部门编码分组，每组内按照员工薪水排序，列出员工的部门名字、姓名和薪水：

01.SELECT d.dname, e.ename, e.sal,

02. DENSE\_RANK()OVER

03. (PARTITION BY e.deptno ORDER BY e.sal)

04. AS drank

05. FROM emp e join dept d

06.one.deptno = d.deptno;

**1.5. 高级分组函数**

**1.5.1. ROLLUP**

ROLLUP、CUBE 和 GROUPING SETS 运算符是 GROUP BY 子句的扩展，可以生成与使用 UNION ALL 来组合单个分组查询时相同的结果集，用来简化和高效的实现统计查询。语法形式如下：

GROUP BY ROLLUP(a,b,c)

等同于

GROUP BY a,b,c

UNION ALL

GROUP BY a,b

UNION

GROUP BY a

UNION ALL

全表

查看每天，每月，每年以及所有营业额？

SELECT year\_id,month\_id,

day\_id,SUM(sales\_value)

FROM sales\_wang

GROUP BY

ROLLUP(year\_id,month\_id,day\_id)

**1.5.2. CUBE**

CUBE函数的语法形式：

01.GROUP BY CUBE(a, b, c)

对cube的每个参数，都可以理解为取值为参与分组和不参与分组两个值的一个维度，所有维度取值组合的集合就是分组后的集合。对于n个参数的cube，有2^n次分组。

如果GROUP BY CUBE(a,b,c)，首先对(a,b,c)进行GROUP BY，然后依次是(a,b)，(a,c)，(a)，(b,c)，(b)，(c)，最后对全表进行GROUP BY操作，所以一共是2^3=8次分组。

01.SELECT a,b,c,SUM(d) FROM test GROUP BY CUBE(a,b,c);

在sales\_value表中使用cube函数：

01.SELECT year\_id, month\_id,

02.SUM(sales\_value) AS sales\_value

03.FROM sales\_tab

04.GROUP BY CUBE (year\_id, month\_id)

05.ORDER BY year\_id, month\_id;

**1.5.3. GROUPING SETS**

GROUPING SETS 运算符可以生成与使用单个 GROUP BY、ROLLUP 或 CUBE 运算符所生成的结果集相同的结果集，但是使用更灵活。

如果不需要获得由完备的 ROLLUP 或 CUBE 运算符生成的全部分组，则可以使用 GROUPING SETS 仅指定所需的分组。GROUPING SETS 列表可以包含重复的分组。

其中分组方式示例如下：

•使用GROUP BY GROUPING SETS(a,b,c)，则对(a),(b),(c)进行GROUP BY

•使用GROUP BY GROUPING SETS((a,b),c), 则对(a,b),(c)进行GROUP BY

•GROUPING BY GROUPING SET(a,a) , 则对(a)进行2次GROUP BY, GROUPING SETS的参数允许重复

**1.6. 集合操作**

**1.6.1. UNION、UNION ALL**

为了合并多个SELECT语句的结果，可以使用集合操作符，实现集合的并、交、差。

集合操作符包括UNION、UNION ALL、INTERSECT和MINUS。多条作集合操作的SELECT语句的列的个数和数据类型必须匹配。

ORDER BY子句只能放在最后的一个查询语句中。

集合操作的语法如下：

01.SELECT statement1

02.[UNION | UNION ALL | INTERSECT | MINUS]

03.SELECT statement2;

UNION和UNION ALL用来获取两个或两个以上结果集的并集：

•UNION操作符会自动去掉合并后的重复记录。

•UNION ALL返回两个结果集中的所有行，包括重复的行。

•UNION操作符对查询结果排序，UNION ALL不排序。

例如，合并职位是’MANAGER’的员工和薪水大于2500的员工集合，查看两种方式的结果差别：

01.--Union

02.SELECT ename, job, sal FROM emp

03.WHERE job = 'MANAGER'

04.SELECT ename, job, sal FROM emp

05.WHERE sal> 2500;

06.

07.--Union all

08.SELECT ename, job, sal FROM emp

09.WHERE job = 'MANAGER'

10.SELECT ename, job, sal FROM emp

11.WHERE sal> 2500;

**1.6.2. INTERSECT**

INTERSECT函数获得两个结果集的交集，只有同时存在于两个结果集中的数据，才被显示输出。使用INTERSECT操作符后的结果集会以第一列的数据作升序排列。

例如：显示职位是’MANAGER’的员工和薪水大于2500的员工的交集：

01.SELECT ename, job, sal FROM emp

02.WHERE job = 'MANAGER'

03.INTERSECT

04.SELECT ename, job, sal FROM emp

05.WHERE sal> 2500;

**1.6.3. MINUS**

MINUS函数获取两个结果集的差集。只有在第一个结果集中存在，在第二个结果集中不存在的数据，才能够被显示出来。也就是结果集一减去结果集二的结果。

例如：列出职位是MANAGER但薪水低于2500的员工记录：

01.SELECT ename, job, sal FROM emp

02.WHERE job = 'MANAGER'

03.MINUS

04.SELECT ename, job, sal FROM emp

05.WHERE sal> 2500;