---

1. 分组与聚合

1.1. 聚合函数

名称	含义
MIN	求最小值，计算中忽略空值。
MAX	求最大值，计算中忽略空值。
SUM	求合计，计算中忽略空值。
COUNT	返回个数，计算中包括空值。
AVG	求平均值，计算中忽略空值。
COLLECT_LIST	将分组中的某列转为一个数组返回，计算中忽略空值。

例：

显示所有雇员的人数、薪水总计、薪水平均值、最高薪水和最低薪水。

```SQL
SELECT COUNT(SAL),SUM(SAL),AVG(SAL),MAX(SAL),MIN(SAL) FROM EMP;

C1                  C2                  C3                      C4          C5
------------------------------------------------------------------------------------
12                  24925               2077.0833333333335      5000        800
```

按用户分组，显示每个用户学习过的课程名字。

```SQL
>select username, collect_list(course) from t_course group by username;
tony      'c','c++','c#'
jay      'python','java'
```

1.2. GROUP BY

利用GROUP BY和聚合函数可以实现分组累计。

举例来说，如果要求显示各个部门的薪水合计，可以使用下面的语句。

```SQL
SELECT DEPTNO,SUM(SAL) FROM EMP GROUP BY DEPTNO;

DEPTNO       C2
-----------------------------------
20           6775
10           8750
30           9400
```

需要注意的是，在SELECT子句中作分组统计的所有字段必须出现在GROUP BY子句中(比如上例中GROUP BY的字段是DEPTNO，那么在SELECT后面必须出现DEPTNO)，否则无法查询。

分组集（Grouping Sets）是多个分组的并集，用于在一个查询中，按照不同的分组列对集合进行聚合运算，等价于对单个分组使用“union all”，计算多个结果集的并集。有4种不同的语法：

```SQL
group by a,b
group by rollup(a,b)
group by cube(a,b)
group by grouping sets((),(a),(a,b),rollup(a,b),cube(a,b))
```

创建示例数据

```SQL
CREATE TABLE TEST_OLAP
(Item int not null,
Color varchar(10) not null,
Quantity int not null,
Store int not null);

INSERT INTO TEST_OLAP VALUES (1,'q',1,1),(2,'w',2,2),(3,'e',3,3),(4,'r',4,4),(5,'t',5,5);
```

使用GROUP BY时还可以和rollup、cube联合使用

```SQL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
CUBE(ITEM,COLOR) ORDER BY ITEM,COLOR;

--等价于
SELECT NULL AS ITEM, NULL AS COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM
TEST_OLAP
UNION ALL
SELECT ITEM,NULL AS COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP
GROUP BY ITEM
UNION ALL
SELECT NULL AS ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP
GROUP BY COLOR
UNION ALL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
ITEM,COLOR;
```

```SQL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
ROLLUP(ITEM,COLOR) ORDER BY ITEM,COLOR;

--等价于
SELECT NULL AS ITEM, NULL AS COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM
TEST_OLAP
UNION ALL
SELECT ITEM,NULL AS COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP
GROUP BY ITEM
UNION ALL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
ITEM,COLOR;
```

将整个集合作为一个分组，grouping sets 是()

```SQL
SELECT SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY GROUPING
SETS(());

--等价于
SELECT SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP;
```

grouping sets是(a)

```SQL
SELECT ITEM,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
GROUPING SETS((ITEM)) ORDER BY ITEM;

--等价于
SELECT ITEM,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY ITEM
ORDER BY ITEM;
```

grouping sets 是(a,b)

```SQL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
GROUPING SETS((ITEM,COLOR)) ORDER BY ITEM,COLOR;

--等价于
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
ITEM,COLOR ORDER BY ITEM,COLOR;
```

分组集是：rollup(a,b)，或 grouping sets是),(a),(a,b

```SQL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
GROUPING SETS((),(ITEM),(ITEM,COLOR)) ORDER BY ITEM,COLOR;

--等价于
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
ROLLUP(ITEM,COLOR) ORDER BY ITEM,COLOR;
```

分组集是：cube(a,b)，或grouping sets是(),(a),(b),(a,b)

```SQL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
GROUPING SETS((),(ITEM),(ITEM,COLOR)) ORDER BY ITEM,COLOR;

--等价于
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
ROLLUP(ITEM,COLOR) ORDER BY ITEM,COLOR;
```

分组集是：grouping setsa),(b, rollup(a)等价于 groupinga),(a),(b),(b,a

```SQL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
GROUPING SETS((ITEM),(COLOR),(ITEM,ITEM),(COLOR,ITEM)) ORDER BY ITEM,COLOR;

--等价于
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
GROUPING SETS((ITEM),(COLOR)),ROLLUP(ITEM) ORDER BY ITEM,COLOR;
```

分组集是：grouping setsa,b,rollup(a)

```SQL
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
GROUPING SETS((ITEM,COLOR),(COLOR,ITEM)) ORDER BY ITEM,COLOR;

--等价于
SELECT ITEM,COLOR,SUM(QUANTITY) AS TOTALQUANTITY,COUNT(STORE) AS STORES FROM TEST_OLAP GROUP BY
GROUPING SETS((ITEM,COLOR)),ROLLUP(ITEM) ORDER BY ITEM,COLOR;
```

1.3. WHERE子句和GROUP BY子句

WHERE子句和GROUP BY子句同时使用时，GROUP BY只对符合WHERE限制的数据记录进行分组聚合计算。换言之，在作真正的聚合计算之前，WHERE子句将不符合条件的数据记录剔除了。

例：

部门10和30的合计薪水是多少？

```SQL
SELECT DEPTNO,SUM (SAL) FROM EMP WHERE DEPTNO IN (10,30) GROUP BY DEPTNO;

DEPTNO       C2
-----------------------------------
10           8750
30           9400
```

1.4. GROUP BY和ORDER BY

在GROUP BY后加上ORDER BY，可以使得分组统计按照指定的秩序来显示。

例如，按部门编号顺序显示部门人数、合计薪水、部门最高薪水、部门最底薪水和部门平均薪水，可以使用下面的SQL语句：

```SQL
SELECT DEPTNO
,COUNT (*)
,SUM (SAL)
,MAX (SAL)
,MIN (SAL)
,AVG (SAL)
FROM EMP
GROUP BY DEPTNO
ORDER BY DEPTNO;

DEPTNO       C2                   C3                   C4           C5           C6
-------------------------------------------------------------------------------------------------
-------------
10           3                    8750                 5000         1300
2916.6666666666665
20           3                    6775                 3000         800
2258.3333333333335
30           6                    9400                 2850         950
1566.6666666666667
```

1.5. GROUP BY和HAVING条件限定

HAVING条件子句是和GROUP一起使用的，用来对分组统计的结果进行限定，只返回满足其条件的分组统计结果。

举例来说，按部门编号顺序显示部门人数、合计薪水、部门最高薪水、部门最低薪水和部门平均薪水，条件是只显示部门平均薪水小于2000的部门。

```SQL
SELECT DEPTNO
,COUNT (*)
,SUM (SAL)
,MAX (SAL)
,MIN (SAL)
,AVG (SAL)
FROM EMP
GROUP BY DEPTNO
HAVING AVG (SAL) < 2000;

DEPTNO       C2                   C3                   C4           C5           C6
-------------------------------------------------------------------------------------------------
-------------
30           6                    9400                 2850         950
1566.6666666666667
```

1.6. GROUP BY小结

在进行分组聚合操作时，要特别注意以下各点：

WHERE：用来限定参与分组聚合运算的表的数据记录，只有满足条件的数据记录才会被选中参与分组聚合。

GROUP BY：将符合 WHERE 条件子句的记录进行分组

HAVING：用来限定可以返回的分组聚合的结果

ORDER BY：用来指定结果的输出顺序

1.7. 分页实现

语法：

```SQL
SELECT select_list
    FROM table_expression
    [ group BY ... ]
    [ ORDER BY ... ]
    {LIMIT number,number} | {LIMIT number,OFFSET number}
```

例：

```SQL
create external table time_dim_wr
(
    t_time_sk                 integer               not null,
    t_time_id                 char(16)              not null,
    t_time                    integer                       ,
    t_hour                    integer                       ,
    t_minute                  integer                       ,
    t_second                  integer                       ,
    t_am_pm                   char(2)                       ,
    t_shift                   char(20)                      ,
    t_sub_shift               char(20)                      ,
    t_meal_time               char(20)                      ,
    primary key (t_time_sk)
) location ('hdfs://node1/node3/db/tpcds/tpcds-1t/time_dim') format 'parquet';

SELECT T_TIME_SK,T_TIME_ID,T_MINUTE FROM time_dim_wr GROUP BY T_TIME_SK,T_TIME_ID,T_MINUTE
ORDER BY t_time_sk LIMIT 10 offset 10;

T_TIME_SK   T_TIME_ID       T_MINUTE
-------------------------------------------------
10      AAAAAAAALAAAAAAA    0
11      AAAAAAAAMAAAAAAA    0
12      AAAAAAAANAAAAAAA    0
13      AAAAAAAAOAAAAAAA    0
14      AAAAAAAAPAAAAAAA    0
15      AAAAAAAAABAAAAAA    0
16      AAAAAAAABBAAAAAA    0
17      AAAAAAAACBAAAAAA    0
18      AAAAAAAADBAAAAAA    0
19      AAAAAAAAEBAAAAAA    0

SELECT T_TIME_SK,T_TIME_ID,T_MINUTE FROM time_dim_wr GROUP BY T_TIME_SK,T_TIME_ID,T_MINUTE
ORDER BY t_time_sk LIMIT 10,10;

T_TIME_SK   T_TIME_ID       T_MINUTE
-------------------------------------------------
10      AAAAAAAALAAAAAAA    0
11      AAAAAAAAMAAAAAAA    0
12      AAAAAAAANAAAAAAA    0
13      AAAAAAAAOAAAAAAA    0
14      AAAAAAAAPAAAAAAA    0
15      AAAAAAAAABAAAAAA    0
16      AAAAAAAABBAAAAAA    0
17      AAAAAAAACBAAAAAA    0
18      AAAAAAAADBAAAAAA    0
19      AAAAAAAAEBAAAAAA    0
```

2. 集合操作

集合操作主要包括：合并操作(UNION)、相交操作(INTERSECT)和求差操作(EXCEPT)。

DB的集合操作与标准ANSI集合操作的不同之处在于返回结果的重复记录处理上。在ANSI标准中集合操作将重复记录自动剔除，而DB增加了ALL关键词，ALL关键词允许保留重复记录。

2.1. 集合操作的定义

相交操作

返回在每一个SELECT语句结果中都存在的记录。INTERSECT用来连接两个SELECT查询，这两个查询所返回的数据集必须具有相同的字段数和数据类型。经相交操作后，只有在两个查询中完全相同的数据行才会返回。举例来说，如果第一个查询返回的记录为A、B、C，第二个查询返回的记录为B、C、D，则相交后的结果为B和C。

合并操作

合并所有SELECT语句的结果，重复记录在结果集中只显示一次。UNION连接两个或更多的查询，虽然每个查询的字段名可以不相同，但必须具有相同的字段数和数据类型。

求差操作

返回第一个SELECT语句结果中除第二个SELECT语句结果中以外的所有记录。同样，两个SELECT查询所返回的字段名可以不同，但数目和数据类型必须一致。在有些数据库系统中，排它操作也称为相减操作(MINUS)。

UNION的基本规则

a. 所有的SELECT语句必须要有同样多的表达式数目。

b. 相关表达式的域必须兼容。

例：

谁是经理7839并且谁为他工作？

```SQL
SELECT ENAME,'employee'
FROM EMP
WHERE MGR == 7839
UNION
SELECT ENAME,'MANAGER'
FROM EMP
WHERE EMPNO == 7839;

ENAME                C2
-------------------------------
JONES                employee
KING                 MANAGER
BLAKE                employee
CLARK                employee
```

2.2. 相交操作

我们通过例子来说明。列出所有的部门编号(不管部门是否有员工) 没有下属)。

```SQL
SELECT DEPTNO
FROM EMP
INTERSECT
SELECT DEPTNO
FROM DEPT;

DEPTNO
-------------
20
10
30
```

2.3. 求差操作

例：

查询工资在1000及以上且非经理的员工编号和姓名。

```SQL
SELECT  EMPNO, ENAME  FROM EMP WHERE  SAL >= 1000
EXCEPT
SELECT  EMPNO, ENAME  FROM EMP WHERE  JOB <> 'MANAGER';

EMPNO       ENAME
----------------------------------
7698        BLAKE
7782        CLARK
7566        JONES
```

2.4. 关于集合操作的补充规则

我们将有关集合操作的一些补充规则列举如下：

a. 在子查询中不能使用集合操作

b. 在定义视图时不能使用集合操作

c. 集合操作的优先级为：INTERSECT 第一，其后分别为 UNION 和 EXCEPT，从左到右。可以使用括号改变优先级。

d. 每一个 SELECT 语句必须有一个 FROM <表名>的子句

e. 每个单独的 SELECT 语句中可以使用 GROUP BY

f. GROUP BY 不能用于或影响整个返回结果集

g. 重复记录将会抛弃，除非使用 ALL 参数

3. 内连接

对于关系数据库系统来说，范式理论是它的基础。多个表按照一定的规定进行组合，可以非常清晰地描述一个企业的业务运行方式。如何灵活地从多个表中查找所需要的信息，很重要的一个手段就是表的连接操作。

对任何关系数据库系统而言，表的连接操作都是非常耗系统资源的。我们常看到一些对某些商用RDBMS颇有经验的人指出，在把系统的逻辑模型转换到物理模型，进行真正的物理实施时，要作不规范化处理(De-Normalise)。一个常用的手段就是将多个表合并成一个表，尽量减少表的连接。之所有这样做的主要原因在于，这些商用RDBMS在处理复杂的多表连接操作时速度太慢，因此将表进行预先的合并，减少表的连接，以提高系统的查询处理速度。这样处理的最大问题是数据冗余量太大，而且将表合并后，会丢失一些实体之间关联的重要信息。

对于DB来说，它是针对大型数据的分析和处理而设计的，具有非常好的并行处理能力，因此在执行表的连接这类复杂操作时能表现出非常好的性能。我们可以通过一些实际的练习来体会这一点。表的连接操作分成两种，即内连接(INNER JOIN)和外连接(OUTER JOIN)。本章只讨论内连接操作，外连接留待后面专门介绍。

3.1. 内连接基本介绍

对于内连接来说，答案集是由来自两个或更多表的字段组成，并且各个表参与连接的字段必须匹配。很多情况下都要用到表的连接，如针对试验数据库提出下面的问题：列出所有员工的编号、姓和所在部门的名字。这个问题就要使用到表的内连接操作。

```SQL
SELECT EMP.EMPNO,EMP.ENAME,DEPT.DNAME FROM EMP,DEPT WHERE EMP.DEPTNO == DEPT.DEPTNO;

EMPNO       ENAME                DNAME
----------------------------------------------------------
7782        CLARK                ACCOUNTING
7839        KING                 ACCOUNTING
7844        TURNER               SALES
7900        JAMES                SALES
7902        FORD                 RESEARCH
7934        MILLER               ACCOUNTING
7369        SMITH                RESEARCH
7499        ALLEN                SALES
7521        WARD                 SALES
7566        JONES                RESEARCH
7654        MARTIN               SALES
7698        BLAKE                SALES
```

事实上，如果按照标准的内连接操作语法，上面的SQL语句应该是下面的形式：

```SQL
SELECT EMP.EMPNO,EMP.ENAME,DEPT.DNAME FROM EMP INNER JOIN DEPT ON EMP.DEPTNO == DEPT.DEPTNO;

EMPNO       ENAME                DNAME
----------------------------------------------------------
7782        CLARK                ACCOUNTING
7839        KING                 ACCOUNTING
7844        TURNER               SALES
7900        JAMES                SALES
7902        FORD                 RESEARCH
7934        MILLER               ACCOUNTING
7369        SMITH                RESEARCH
7499        ALLEN                SALES
7521        WARD                 SALES
7566        JONES                RESEARCH
7654        MARTIN               SALES
7698        BLAKE                SALES
```

3.2. 别名的定义和使用

在进行连接操作时，经常要使用别名。例如上节中的SQL语句可以写成下面的形式：

```SQL
SELECT e.EMPNO,e.ENAME,d.DNAME FROM EMP e,DEPT d WHERE e.DEPTNO == d.DEPTNO;

EMPNO       ENAME                DNAME
----------------------------------------------------------
7782        CLARK                ACCOUNTING
7839        KING                 ACCOUNTING
7844        TURNER               SALES
7900        JAMES                SALES
7902        FORD                 RESEARCH
7934        MILLER               ACCOUNTING
7369        SMITH                RESEARCH
7499        ALLEN                SALES
7521        WARD                 SALES
7566        JONES                RESEARCH
7654        MARTIN               SALES
7698        BLAKE                SALES
```

这里EMP的别名是E，而DEPT的别名是D。由此可见，所谓别名，是表或者视图的一个临时名字，它必须在FROM子句中加以定义。对于名字很长的表或者视图，使用别名是很方便的。另外，当一个表与它自己进行连接，即所谓的自连接(SELFJOIN)时，一定要使用别名。一旦定义好别名，在此SQL语句中就必须使用它。如果使用别名，并用ON子句来定义连接条件，那上面的SQL语句可写成下面的形式：

```SQL
SELECT e.EMPNO,e.ENAME,d.DNAME FROM EMP e INNER JOIN DEPT d ON e.DEPTNO == d.DEPTNO;

EMPNO        ENAME                DNAME
-------------------------------------------------------
7782         CLARK                ACCOUNTING
7839         KING                 ACCOUNTING
7844         TURNER               SALES
7900         JAMES                SALES
7902         FORD                 RESEARCH
7934         MILLER               ACCOUNTING
7369         SMITH                RESEARCH
7499         ALLEN                SALES
7521         WARD                 SALES
7566         JONES                RESEARCH
7654         MARTIN               SALES
7698         BLAKE                SALES
```

3.3. 交叉连接(CROSS JOIN)

在上面讨论的内连接操作中，有一个条件特别要注意：那就是要有连接条件，通过连接条件，把参与连接的各表的相应字段进行匹配，从而得到合适的结果。

如果在一个连接中没有连接条件，那这个连接就变成了交叉连接(CROSS JOIN)，也称为乘积连接(PRODUCT JOIN)。如下面的SQL语句就是一个交叉连接：

```SQL
SELECT EMP.EMPNO,EMP.DEPTNO FROM EMP,DEPT WHERE EMP.EMPNO == 7698;

EMPNO        DEPTNO
--------------------------
7698         30
7698         30
7698         30
7698         30
```

这里虽然定义了WHERE条件，但显然两个表EMPLOYEE和DEPARTMENT之间没有任何进行匹配的操作，即没有连接条件。这就是一个交叉连接。如果写成下面的形式，就更清楚了。

```SQL
SELECT e.EMPNO,d.DEPTNO FROM EMP e CROSS JOIN DEPT d WHERE e.EMPNO == 7698;

EMPNO        DEPTNO
--------------------------
7698         10
7698         20
7698         30
7698         40
```

下面的SQL语句则是由于使用的连接条件不正确而产生的交叉乘积。在这个SQL语句中，连接条件"3==3"和两个表没有任何关系，显然是不正确的。

```SQL
SELECT e.EMPNO,d.DNAME FROM EMP e INNER JOIN DEPT d ON 3 == 3;

EMPNO        DNAME
-----------------------------------
7782         ACCOUNTING
7839         ACCOUNTING
7844         ACCOUNTING
7900         ACCOUNTING
7902         ACCOUNTING
7934         ACCOUNTING
7369         ACCOUNTING
7499         ACCOUNTING
7521         ACCOUNTING
7566         ACCOUNTING
7654         ACCOUNTING
7698         ACCOUNTING
7782         RESEARCH
7839         RESEARCH
7844         RESEARCH
7900         RESEARCH
7902         RESEARCH
7934         RESEARCH
7369         RESEARCH
7499         RESEARCH
7521         RESEARCH
7566         RESEARCH
7654         RESEARCH
7698         RESEARCH
7782         SALES
7839         SALES
7844         SALES
7900         SALES
7902         SALES
7934         SALES
7369         SALES
7499         SALES
7521         SALES
7566         SALES
7654         SALES
7698         SALES
7782         OPERATIONS
7839         OPERATIONS
7844         OPERATIONS
7900         OPERATIONS
7902         OPERATIONS
7934         OPERATIONS
7369         OPERATIONS
7499         OPERATIONS
7521         OPERATIONS
7566         OPERATIONS
7654         OPERATIONS
7698         OPERATIONS
```

完全没有任何限制的交叉连接称为笛卡尔乘积(CARTESIAN PRODUCT)，如下面的SQL语句就是一个笛卡尔乘积。

笛卡尔乘积耗费系统很多的资源，特别是缓冲空间，因为它返回的结果非常庞大。假设一个表有1000条记录，另一个表有2000条记录，这是两个很小的表，但它们进行笛卡尔乘积后，返回的结果是200万行数据！一般情况下，这种交叉连接都没有什么实际的意义，没有必要做深入的研究。

```SQL
SELECT EMP.EMPNO,EMP.DEPTNO FROM EMP CROSS JOIN DEPT;

EMPNO        DEPTNO
-------------------------
7782         10
7782         10
7782         10
7782         10
7839         10
7839         10
7839         10
7839         10
7844         30
7844         30
7844         30
7844         30
7900         30
7900         30
7900         30
7900         30
7902         20
7902         20
7902         20
7902         20
7934         10
7934         10
7934         10
7934         10
7369         20
7369         20
7369         20
7369         20
7499         30
7499         30
7499         30
7499         30
7521         30
7521         30
7521         30
7521         30
7566         20
7566         20
7566         20
7566         20
7654         30
7654         30
7654         30
7654         30
7698         30
7698         30
7698         30
7698         30
```

3.4. 自连接(SELF JOIN)

在有些情况下，要把一个表和它自身进行连接，这就是自连接(SELF JOIN)。

举例来说，要把所有名字带有T的雇员及其经理找出来，可以使用下面的SQL语句。

```SQL
SELECT * FROM EMP e,EMP m WHERE e.MGR == m.EMPNO AND e.ENAME LIKE '%T%';

EMPNO        ENAME                JOB                  MGR          HIREDATE    SAL
COMM         DEPTNO       EMPNO        ENAME                JOB                  MGR
HIREDATE    SAL          COMM         DEPTNO
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------
7844         TURNER               SALESMAN             7698         1981-09-08  1500
0            30           7698         BLAKE                MANAGER              7839
1981-05-01  2850         <NULL>       30
7369         SMITH                CLERK                7902         1980-12-17  800
<NULL>       20           7902         FORD                 ANALYST              7566
1981-12-03  3000         <NULL>       20
7654         MARTIN               SALESMAN             7698         1981-09-28  1250
1400         30           7698         BLAKE                MANAGER              7839
1981-05-01  2850         <NULL>       30
```

我们也可以用ON子句来定义连接条件，如：

```SQL
SELECT * FROM EMP e INNER JOIN EMP m ON e.MGR == m.EMPNO WHERE e.ENAME LIKE '%T%';

EMPNO        ENAME                JOB                  MGR          HIREDATE    SAL
COMM         DEPTNO       EMPNO        ENAME                JOB                  MGR
HIREDATE    SAL          COMM         DEPTNO
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------
7844         TURNER               SALESMAN             7698         1981-09-08  1500
0            30           7698         BLAKE                MANAGER              7839
1981-05-01  2850         <NULL>       30
7369         SMITH                CLERK                7902         1980-12-17  800
<NULL>       20           7902         FORD                 ANALYST              7566
1981-12-03  3000         <NULL>       20
7654         MARTIN               SALESMAN             7698         1981-09-28  1250
1400         30           7698         BLAKE                MANAGER              7839
1981-05-01  2850         <NULL>       30
```

3.5. 子查询(SUBQUERY)与表的连接

在有些情况下，表的连接操作也可以通过子查询来实现。例如，下面采用表连接的SQL语句：

```SQL
SELECT * FROM EMP INNER JOIN DEPT ON EMP.DEPTNO == DEPT.DEPTNO WHERE DEPT.DNAME LIKE
'%RESEARCH%';

EMPNO        ENAME                JOB                  MGR          HIREDATE    SAL
COMM         DEPTNO       DEPTNO       DNAME                LOC
-------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------
7902         FORD                 ANALYST              7566         1981-12-03  3000
<NULL>       20           20           RESEARCH             DALLAS
7369         SMITH                CLERK                7902         1980-12-17  800
<NULL>       20           20           RESEARCH             DALLAS
7566         JONES                MANAGER              7839         1981-04-02  2975
<NULL>       20           20           RESEARCH             DALLAS
```

和下面采用子查询的SQL语句实现的是同样的功能。

```SQL
SELECT * FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE DNAME LIKE '%RESEARCH%');

EMPNO       ENAME                JOB                  MGR         HIREDATE    SAL
COMM        DEPTNO
-------------------------------------------------------------------------------------------------
-------------------------
7902        FORD                 ANALYST              7566        1981-12-03  3000
<NULL>      20
7369        SMITH                CLERK                7902        1980-12-17  800
<NULL>      20
7566        JONES                MANAGER              7839        1981-04-02  2975
<NULL>      20
```

在子查询中，返回的结果集必须来自一个表，而另一个表只是对前一个表的数据作一些限制，看哪些是满足条件的，哪些不满足。如果返回的结果来自两个或更多的表，则必须使用连接操作。至于哪些数据可以返回，必须看连接条件的定义，只有匹配连接条件的数据记录才会返回。

严格说来，连接条件应该在ON子句中加以定义，其它限制条件则在WHERE子句中加以定义。表的连接操作是在缓冲区中完成的，WHERE子句则进一步对连接后的结果进行限制，把那些符合WHERE条件子句的数据记录返回给前端。

4. 多个表的外连接操作

根据前面的讨论，外连接操作中表的先后顺序很重要。因此，它既不能交换，也不能结合。当两个以上表进行外连接时，必须仔细定义连接的秩序，因此ON子句的正确位置就显得很重要。考虑雇员、部门和工作三个表，如果要知道所有的雇员信息，而不考虑该雇员是否有有效的部门编号或工作代码，可以使用下面这种形式的SQL语句:

```SQL
SELECT e.ENAME
,d.DNAME
,b.JOB
FROM DEPT D
RIGHT OUTER JOIN EMP e
ON d.DEPTNO == e.DEPTNO
LEFT OUTER JOIN EMP1 b
ON e.JOB == b.JOB;

ENAME                DNAME                JOB
-------------------------------------------------------------------
CLARK                ACCOUNTING           MANAGER
CLARK                ACCOUNTING           MANAGER
CLARK                ACCOUNTING           MANAGER
KING                 ACCOUNTING           PRESIDENT
TURNER               SALES                SALESMAN
TURNER               SALES                SALESMAN
TURNER               SALES                SALESMAN
TURNER               SALES                SALESMAN
JAMES                SALES                CLERK
JAMES                SALES                CLERK
JAMES                SALES                CLERK
FORD                 RESEARCH             ANALYST
MILLER               ACCOUNTING           CLERK
MILLER               ACCOUNTING           CLERK
MILLER               ACCOUNTING           CLERK
SMITH                RESEARCH             CLERK
SMITH                RESEARCH             CLERK
SMITH                RESEARCH             CLERK
ALLEN                SALES                SALESMAN
ALLEN                SALES                SALESMAN
ALLEN                SALES                SALESMAN
ALLEN                SALES                SALESMAN
WARD                 SALES                SALESMAN
WARD                 SALES                SALESMAN
WARD                 SALES                SALESMAN
WARD                 SALES                SALESMAN
JONES                RESEARCH             MANAGER
JONES                RESEARCH             MANAGER
JONES                RESEARCH             MANAGER
MARTIN               SALES                SALESMAN
MARTIN               SALES                SALESMAN
MARTIN               SALES                SALESMAN
MARTIN               SALES                SALESMAN
BLAKE                SALES                MANAGER
BLAKE                SALES                MANAGER
BLAKE                SALES                MANAGER
```

5. 子查询

5.1. 基本子查询

假设需要查询所有部门经理的姓，一种方法是先手工查询部门表中所有经理 的员工代码，记住这些数据，再通过查询获得经理的姓。

如果有几千条记录，这种方式肯定不行。所以引出了子查询的概念。子查询是利用另一条SQL语句中的 SELECT语句来获得中间结果，如下所示：

```SQL
SELECT * FROM EMP WHERE EMPNO IN (SELECT EMPNO FROM EMP WHERE JOB == 'MANAGER');

EMPNO        ENAME                JOB                  MGR          HIREDATE    SAL
COMM         DEPTNO
-------------------------------------------------------------------------------------------------
--------------------------
7782         CLARK                MANAGER              7839         1981-06-09  2450
<NULL>       10
7566         JONES                MANAGER              7839         1981-04-02  2975
<NULL>       20
7698         BLAKE                MANAGER              7839         1981-05-01  2850
<NULL>       30
```

5.2. 复杂子查询

例：

查询工作是ACCOUNTING的员工信息。

```SQL
SELECT ENAME,DEPTNO FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE DNAME LIKE
'%ACCOUNTING%');

ENAME                DEPTNO
-----------------------------------
CLARK                10
KING                 10
MILLER               10
```

例：

查询被分配管理同一部门的部门级经理。

```SQL
SELECT EMPNO,ENAME FROM EMP WHERE (DEPTNO, EMPNO) IN (SELECT DEPTNO,MGR FROM emp);

EMPNO        ENAME
-----------------------------------
7782         CLARK
7839         KING
7902         FORD
7566         JONES
7698         BLAKE
```

5.3. EXISTS在子查询中的使用

EXISTS可以使用在子查询中，用来表示查询至少返回一行。如果前面加上否 定词NOT，则表示查询时无记录存在。EXISTS可以代替IN，而NOT EXISTS可以 代替NOT IN。

```SQL
SELECT * FROM EMP e WHERE EXISTS (SELECT * FROM DEPT d WHERE e.DEPTNO==d.DEPTNO);

EMPNO       ENAME                JOB                  MGR          HIREDATE    SAL
COMM        DEPTNO
-------------------------------------------------------------------------------------------------
-------------------------
7782        CLARK                MANAGER              7839         1981-06-09  2450
<NULL>      10
7839        KING                 PRESIDENT            <NULL>       1981-11-17  5000
<NULL>      10
7844        TURNER               SALESMAN             7698         1981-09-08  1500
0           30
7900        JAMES                CLERK                7698         1981-12-03  950
<NULL>      30
7902        FORD                 ANALYST              7566         1981-12-03  3000
<NULL>      20
7934        MILLER               CLERK                7782         1982-01-23  1300
<NULL>      10
7369        SMITH                CLERK                7902         1980-12-17  800
<NULL>      20
7499        ALLEN                SALESMAN             7698         1981-02-20  1600
300         30
7521        WARD                 SALESMAN             7698         1981-02-22  1250
500         30
7566        JONES                MANAGER              7839         1981-04-02  2975
<NULL>      20
7654        MARTIN               SALESMAN             7698         1981-09-28  1250
1400        30
7698        BLAKE                MANAGER              7839         1981-05-01  2850
<NULL>      30


SELECT * FROM EMP e WHERE  NOT EXISTS (SELECT * FROM DEPT d WHERE e.DEPTNO==d.DEPTNO);

EMPNO       ENAME                JOB                  MGR          HIREDATE    SAL
COMM        DEPTNO
-------------------------------------------------------------------------------------------------
-------------------------
```

5.4. 关于子查询的一些基本规则

a. 子查询必须用括号括起来

b. 子查询可以是 IN 或 NOT IN 子句的操作目标

c. 也可以是 EXISTS 或 NOT EXISTS 子句的操作目标

d. 支持 LIKE 或 NOT LIKE

e. ORDER BY 不能用于子查询内

6. 相关子查询和派生表

6.1. 相关子查询（Correlated Subqueries）

子查询是指在SQL语句限定子句(WHERE)中嵌套的查询语句。子查询分为两种类型：独立子查询和相关子查询。独立子查询是单独执行，其结果参与外层的查询，整个查询中子查询只执行一次，执行完后再执行外层查询；相关子查询是指子查询(内层查询)中引用了外层查询所引用表的字段，因此外层查询处理每一条记录时都必须执行一次子查询，因为子查询中引用的字段的值发生了变化。

例：

找出每个部门中薪水最高的雇员，显示姓和薪水。

```SQL
SELECT ENAME
,SAL
FROM EMP ee
WHERE SAL ==
(SELECT MAX (SAL)
FROM EMP em
WHERE ee.DEPTNO == em.DEPTNO);

ENAME                SAL
-----------------------------------
KING                 5000
FORD                 3000
BLAKE                2850
```

再举一个例子：查找薪水高于部门平均薪水的所有员工，显示姓和薪水。

```SQL
SELECT ENAME
,SAL
FROM EMP ee
WHERE SAL >
(SELECT AVG (SAL)
FROM EMP em
WHERE ee.DEPTNO == em.DEPTNO);

ENAME                SAL
-----------------------------------
KING                 5000
FORD                 3000
ALLEN                1600
JONES                2975
BLAKE                2850
```

利用相关子查询可以完成较为复杂的功能，下面是在IN和EXISTS表达式中使用相关子查询的例子：哪些部门没有雇员？

```SQL
SELECT DEPTNO,DNAME
FROM DEPT
WHERE DEPTNO NOT IN
(SELECT DEPTNO
FROM emp) ;

ENAME                SAL
-----------------------------------
KING                 5000
FORD                 3000
ALLEN                1600
JONES                2975
BLAKE                2850
```

如果使用NOT EXISTS表达式，可以写成：

```SQL
SELECT DEPTNO,DNAME
FROM DEPT
WHERE NOT EXISTS
(SELECT *
FROM EMP
WHERE EMP.DEPTNO == DEPT.DEPTNO);

DEPTNO       DNAME
-----------------------------------
40           OPERATIONS
```

6.2. 相关子查询和连接

显示在所属部门薪水最高的经理的雇员编、姓名、部门编号和薪水。

```SQL
SELECT e.ENAME,e.MGR
,d.DEPTNO
,e.SAL
FROM DEPT d
INNER JOIN EMP e
ON e.DEPTNO == d.DEPTNO
WHERE e.SAL ==
(SELECT MAX(SAL)FROM EMP);

ENAME                MGR          DEPTNO       SAL
-------------------------------------------------------------
KING                 <NULL>       10           5000
```

6.3. 派生表

中间表需要用单独的SQL语句来创建和维护，我们也可以使用派生表代替中间 表。事实上，派生表就是一个命名的中间表，它在整个SQL语句中自动创建和删 除。在定义派生表的整个SQL语句中都可以通过派生表名来引用它。

语法为：

FROM (子查询) [AS] <派生表名称> [(列表)]

例:

找出所有薪水高于公司平均薪水的员工。

```SQL
SELECT ENAME
,SAL
,AVGSAL
FROM (SELECT AVG (SAL)
FROM EMP) TEMP_TABLE (AVGSAL)
,emp
WHERE SAL > AVGSAL;

ENAME                SAL          C1
-------------------------------------------------------------
CLARK                2450         2077.0833333333335
KING                 5000         2077.0833333333335
FORD                 3000         2077.0833333333335
JONES                2975         2077.0833333333335
BLAKE                2850         2077.0833333333335
```

这里定义了名为TEMP_TABLE的派生表，它只有一个字段即平均薪水。这个派生表在整 个SQL语句中都可以引用，如在SELECT部分引用了派生表的字段，在WHERE条 件子句中也引用了派生表的字段。

研究这个SQL查询，可以得出这样的结论： 派生表实际上还是一种中间表，如果把中间表的操作集成在一条 SQL 语句 中，就是派生表。 派生表在整个 SQL 语句范围内有效。 完成指定的 SQL 操作后派生表将自动被删除。

在派生表中使用分组(GROUP) 在WHERE子句中不能直接使用聚集函数，我们可以通过派生表实现WHERE子句中对聚集函数的引用。

例:

显示所有薪水高于部门平均薪水的雇员信息。

```SQL
SELECT ENAME
,SAL
,DEPTNO
,AVGSAL
FROM (SELECT AVG (SAL)
,DEPTNO
FROM EMP
GROUP BY DEPTNO)
TEMP_TABLE (AVGSAL, DEPTNO)
,EMP ee
WHERE SAL > AVGSAL;

ENAME                SAL          DEPTNO       C1
-------------------------------------------------------------------------
CLARK                2450         20           2258.3333333333335
CLARK                2450         30           1566.6666666666667
KING                 5000         20           2258.3333333333335
KING                 5000         10           2916.6666666666665
KING                 5000         30           1566.6666666666667
FORD                 3000         20           2258.3333333333335
FORD                 3000         10           2916.6666666666665
FORD                 3000         30           1566.6666666666667
ALLEN                1600         30           1566.6666666666667
JONES                2975         20           2258.3333333333335
JONES                2975         10           2916.6666666666665
JONES                2975         30           1566.6666666666667
BLAKE                2850         20           2258.3333333333335
BLAKE                2850         30           1566.6666666666667
```