

第3章 SQL之数据查询

大纲

- SQL概述
- 学生-课程数据库
- 数据定义
- **数据查询**
- 数据更新
- 空值的处理
- 视图
- 本章小结

数据查询的一般形式

- 查询语句(SELECT)格式

SELECT [ALL|DISTINCT]<目标列表达式>[,<目标列表达式>] ...

FROM <表名或视图名>[,<表名或视图名>]...| (SELECT 语句) [AS]<别名> --派生表

[WHERE <条件表达式>]

[GROUP BY <列名1>[HAVING <条件表达式>]]

[ORDER BY <列名2>[ASC|DESC]]

[limit n]; --该选项是openGauss支持的语法结构, 表明只显示前面n条记录

数据查询

- **单表查询**
- 连接查询
- 嵌套查询
- 集合查询
- 基于派生表的查询
- Select语句的一般形式

openGauss之数据查询SQL

- 支持标准的SQL92/SQL99/SQL2003/SQL2011规范，支持GBK和UTF-8字符集，支持SQL标准函数与分析函数，支持存储过程。
- openGauss之SQL学习(课下自学)
<https://education.huaweicloud.com/courses/course-v1:HuaweiX+CBUCNXDR006+Self-paced/about>
- 华为贾军锋老师的材料(见补充)

单表查询

- **查询仅涉及一个表**

- 1.选择表中的若干列**

- 2.选择表中的若干元组

- 3.ORDER BY子句

- 4.聚集函数

- 5.GROUP BY子句

■ 查询指定列

[例3.16] 查询全体学生的学号与姓名。

```
SELECT Sno, Sname  
FROM Student;
```

[例3.17] 查询全体学生的姓名、学号、所在系。

```
SELECT Sname, Sno, Sdept  
FROM Student;
```

■ 查询全部列

- 选出所有属性列：
 - 在SELECT关键字后面列出所有列名
 - 将<目标列表达式>指定为 *

[例3.18] 查询全体学生的详细记录。

```
SELECT Sno, Sname, Ssex, Sage, Sdept  
FROM Student;
```

或

```
SELECT *  
FROM Student;
```


■ 查询经过计算的值

- SELECT子句的<目标列表表达式>不仅可以为表中的属性列，也可以是表达式

[例3.19] 查全体学生的姓名及其出生年份。

SELECT Sname, 2019-Sage FROM Student; /*假设当时为2019年*/

Sname	2019-Sage
李勇	1994
刘晨	1995
王敏	1996
张立	1995

[例3.20] 查询全体学生的姓名、出生年份和所在院系，要求用小写字母表示系名。

```
SELECT Sname, 'Year of Birth: ', 2019-Sage, LOWER(Sdept) FROM Student;
```

Sname	'Year of Birth:'	2019-Sage	LOWER(Sdept)
李勇	Year of Birth:	1999	cs
刘晨	Year of Birth:	2000	cs
王敏	Year of Birth:	2001	ma
张立	Year of Birth:	2000	is

注：例题中的 'Year of Birth' 是字面值(literal value)

- 使用列别名改变查询结果的列标题:

```
SELECT Sname NAME, 'Year of Birth:' BIRTH, 2019-Sage BIRTHDAY, LOWER(Sdept) DEPARTMENT  
FROM Student;
```

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1994	cs
刘晨	Year of Birth:	1995	cs
王敏	Year of Birth:	1996	ma
张立	Year of Birth:	1995	is

单表查询

- **查询仅涉及一个表**

- 1.选择表中的若干列

- 2.选择表中的若干元组

- 3.ORDER BY子句

- 4.聚集函数

- 5.GROUP BY子句

■ 消除取值重复的行

- 如果没有指定DISTINCT关键词，则缺省为ALL
- 指定DISTINCT关键词，去掉表中重复的行

[例3.21] 查询选修了课程的学生学号。

SELECT Sno FROM SC; 等价于: SELECT ALL Sno FROM SC;

SELECT DISTINCT Sno FROM SC;

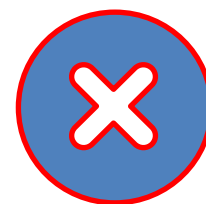
Sno
201215121
201215122



Sno
201215121
201215121
201215121
201215122
201215122

- 注意: **DISTINCT**短语的作用范围是**所有**目标列

```
SELECT DISTINCT Cno, DISTINCT Grade  
FROM SC;
```



```
SELECT DISTINCT Cno, Grade  
FROM SC;
```



■ 查询满足条件的元组

- 通过WHERE子句实现

表3.6 常用的查询条件

查询条件	谓 词
比较	=, >, <, >=, <=, !=, <>, !>, !<; NOT+上述比较运算符
确定范围	BETWEEN AND, NOT BETWEEN AND
确定集合	IN, NOT IN
字符匹配	LIKE, NOT LIKE
空 值	IS NULL, IS NOT NULL
多重条件（逻辑运算）	AND, OR, NOT

■ 比较大小

[例3.22] 查询计算机科学系全体学生的名单。

```
SELECT Sname  
FROM Student  
WHERE Sdept= 'CS' ;
```

[例3.23] 查询所有年龄在20岁以下的学生姓名及其年龄。

```
SELECT Sname,Sage  
FROM Student  
WHERE Sage < 20;
```

[例3.24] 查询考试成绩有不及格的学生的学号。

```
SELECT DISTINCT Sno  
FROM SC  
WHERE Grade<60;
```


- 确定范围:

BETWEEN ... AND ... , NOT BETWEEN ... AND ...

[例3.25] 查询年龄在20~23岁(包括20岁和23岁)之间的学生的姓名、系别和年龄。

```
SELECT Sname, Sdept, Sage
FROM Student
WHERE Sage BETWEEN 20 AND 23;
```

[例3.26] 查询年龄不在20~23岁之间的学生姓名、系别和年龄。

```
SELECT Sname, Sdept, Sage
FROM Student
WHERE Sage NOT BETWEEN 20 AND 23;
```

- 确定集合:

IN <值表>, NOT IN <值表>

[例3.27] 查询计算机科学系(CS)、数学系(MA)和信息系(IS)学生的姓名和性别。

```
SELECT Sname, Ssex  
FROM Student  
WHERE Sdept IN ('CS','MA' , 'IS' );
```

[例3.28] 查询既不是计算机科学系、数学系，也不是信息系的学生姓名和性别。

```
SELECT Sname, Ssex  
FROM Student  
WHERE Sdept NOT IN ('IS','MA' , 'CS' );
```

■ 字符匹配:

[NOT] LIKE '<匹配串>' [ESCAPE '<换码字符>']

– <匹配串>可以是一个完整的字符串，也可以含有通配符%和 _

– % (百分号)代表任意长度(长度可以为0)的字符串。

例如，a%b表示以a开头，以b结尾的任意长度的字符串。

– _ (下划线)代表任意单个字符。

例如，a_b表示以a开头，以b结尾的长度为3的任意字符串。

- 匹配串为固定字符串

[例3.29] 查询学号为201215121的学生的详细情况。

```
SELECT *  
FROM Student  
WHERE Sno LIKE '201215121';
```

等价于：

```
SELECT *  
FROM Student  
WHERE Sno = '201215121';
```

- 匹配串为含通配符的字符串

[例3.30] 查询所有姓刘学生的姓名、学号和性别。

```
SELECT Sname, Sno, Ssex  
FROM Student  
WHERE Sname LIKE '刘%';
```

[例3.31] 查询姓"欧阳"且全名为三个汉字的学生的姓名。

```
SELECT Sname  
FROM Student  
WHERE Sname LIKE '欧阳_';
```

[例3.32] 查询名字中第2个字为"阳"字的学生的姓名和学号。

```
SELECT Sname, Sno  
FROM Student  
WHERE Sname LIKE '__阳%';
```

[例3.33] 查询所有不姓刘的学生姓名、学号和性别。

```
SELECT Sname, Sno, Ssex  
FROM Student  
WHERE Sname NOT LIKE '刘%';
```

- 使用换码字符将通配符转义为普通字符

[例3.34] 查询DB_Design课程的课程号和学分。

```
SELECT Cno, Ccredit  
FROM Course  
WHERE Cname LIKE 'DB\_Design' ESCAPE '\';
```

[例3.35] 查询以"DB_"开头，且倒数第3个字符为i的课程的具体情况。

```
SELECT *  
FROM Course  
WHERE Cname LIKE 'DB\__%i\__' ESCAPE '\';
```

注：ESCAPE '\ ' 表示 '\ ' 为换码字符

- 涉及空值(NULL)的查询:

IS NULL 或 IS NOT NULL

- “IS”不能用“=”代替
- 空值和任何数据之间的比较是没有意义的。如果数学表达式中包含空值，则结果都为NULL。
- 空值与数字0和空字符串所代表的意义不同，它代表的是未定义的值，不占用存储空间的，而0和空格是有意义的且占用存储空间。

[例3.36] 某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩。
查询缺少成绩的学生的学号和相应的课程号。

```
SELECT Sno, Cno  
FROM SC  
WHERE Grade IS NULL;
```

[例3.37] 查询所有有成绩的学生学号和课程号。

```
SELECT Sno, Cno  
FROM SC  
WHERE Grade IS NOT NULL;
```

■ 多重条件查询

- 逻辑运算符：AND和OR来连接多个查询条件
 - 运算符的优先级： 括号 > AND > OR

[例3.38] 查询计算机系年龄在20岁以下的学生姓名。

```
SELECT Sname  
FROM Student  
WHERE Sdept= 'CS' AND Sage<20;
```

[例3.27] 查询计算机科学系(CS), 数学系(MA)和信息系(IS)学生的姓名和性别。

```
SELECT Sname, Ssex  
FROM Student  
WHERE Sdept='CS' OR Sdept='MA' OR Sdept='IS';
```

单表查询

- **查询仅涉及一个表**

- 1.选择表中的若干列

- 2.选择表中的若干元组

- 3.ORDER BY子句**

- 4.聚集函数

- 5.GROUP BY子句

■ ORDER BY子句

- 可以按一个或多个属性列排序
- 升序：ASC；降序：DESC；缺省值为升序
- 对于空值，排序时显示的次序由具体系统实现来决定

[例3.39] 查询选修了3号课程的学生学号及其成绩，查询结果按分数降序排列。

```
SELECT Sno, Grade FROM SC WHERE Cno='3' ORDER BY Grade DESC;
```

[例3.40] 查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列。

```
SELECT * FROM Student ORDER BY Sdept, Sage DESC;
```

单表查询

- **查询仅涉及一个表**

- 1.选择表中的若干列
- 2.选择表中的若干元组
- 3.ORDER BY子句
- 4.聚集函数**
- 5.GROUP BY子句

■ 聚集函数

函数名	含义	用法
COUNT()	统计元组个数	COUNT(*)
	统计一列中值的个数	COUNT([DISTINCT ALL] <列名>)
SUM()	计算一列值的总和 • 此列必须为数值型	SUM([DISTINCT ALL] <列名>)
AVG()	计算一列值的平均值 • 此列必须为数值型	AVG([DISTINCT ALL] <列名>)
MIN()	求一列中的最小值	MIN([DISTINCT ALL] <列名>)
MAX()	求一列中的最大值	MAX([DISTINCT ALL] <列名>)

- 当聚集函数遇到空值时，除count(*)外，都跳过空值而只处理非空值
- count(*)是对元组进行计数，某个元组的一个或部分列取空值不影响其统计结果

[例3.41] 查询学生总人数。

```
SELECT COUNT(*)  
FROM Student;
```

[例3.42] 查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)  
FROM SC;
```

[例3.43] 计算1号课程的学生平均成绩。

```
SELECT AVG(Grade)  
FROM SC  
WHERE Cno='1';
```

[例3.44] 查询选修1号课程的学生最高分数。

```
SELECT MAX(Grade)
FROM SC
WHERE Cno='1';
```

[例3.45] 查询学生201215012选修课程的总学分数。

```
SELECT SUM(Ccredit)
FROM SC, Course
WHERE Sno='201215012' AND SC.Cno=Course.Cno;
```


单表查询


- **查询仅涉及一个表**

- 1.选择表中的若干列
- 2.选择表中的若干元组
- 3.ORDER BY子句
- 4.聚集函数
- 5.GROUP BY子句**

- **GROUP BY子句分组：细化聚集函数的作用对象**
 - 如果未对查询结果分组，聚集函数将作用于整个查询结果
 - 对查询结果分组后，聚集函数将分别作用于每个组
 - 按指定的一列或多列值分组，值相等的为一组
 - **GROUP BY**后面的子句必须出现在**SELECT**的第一个位置

[例3.46] 求各个课程号及相应的选课人数。


```
SELECT Cno, COUNT(Sno)
FROM SC
GROUP BY Cno;
```



CNO	COUNT(SNO)
1	22
2	34
3	44
4	33
5	48


[例3.47] 查询选修了3门以上课程的学生学号。

```
SELECT Sno  
FROM SC  
GROUP BY Sno  
HAVING COUNT(*)>3;
```



[例3.48] 查询平均成绩大于等于90分的学生学号和平均成绩。

```
SELECT Sno, AVG(Grade)  
FROM SC  
GROUP BY Sno  
HAVING AVG(Grade)>=90;
```



```
SELECT Sno, AVG(Grade)  
FROM SC  
WHERE AVG(Grade)>=90  
GROUP BY Sno;
```

- **HAVING短语与WHERE子句的区别:**

类型	作用对象	含义
WHERE子句	基表或视图	选择满足条件的元组
HAVING短语	组	选择满足条件的组

数据查询

- 单表查询
- **连接查询**
- 嵌套查询
- 集合查询
- 基于派生表的查询
- Select语句的一般形式

连接查询

- 什么是连接查询？

- 同时涉及两个以上表的查询

- 什么是连接条件或连接谓词？

- 用来连接两个表的条件

[<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>

[<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

- 连接字段：连接谓词中的列名称

- 连接条件中的各连接字段类型必须是可比的，但名字不必相同

连接查询

- 等值与非等值连接查询
- 自身连接
- 外连接
- 多表连接

1.等值与非等值连接查询

■ 什么是等值连接查询？

- 连接运算符为 = 的查询

[例 3.49] 查询每个学生及其选修课程的情况。

```
SELECT Student.*, SC.*  
FROM Student, SC  
WHERE Student.Sno = SC.Sno;
```

Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
201215121	李勇	男	20	CS	201215121	1	92
201215121	李勇	男	20	CS	201215121	2	85
201215121	李勇	男	20	CS	201215121	3	88
201215122	刘晨	女	19	CS	201215122	2	90
201215122	刘晨	女	19	CS	201215122	3	80

连接操作的执行过程

■ 嵌套循环法(NESTED-LOOP)

- 首先在表1中找到第一个元组，然后从头开始扫描表2，逐一查找满足连接件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。
- 表2全部查找完后，再找表1中第二个元组，然后再从头开始扫描表2，逐一查找满足连接条件的元组，找到后就将表1中的第二个元组与该元组拼接起来，形成结果表中一个元组。
- 重复上述操作，直到表1中的全部元组都处理完毕。

■ 排序合并法(SORT-MERGE)

- 常用于=连接
- 首先按连接属性对表1和表2排序。
- 对表1的第一个元组，从头开始扫描表2，顺序查找满足连接条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。当遇到表2中第一条大于表1连接字段值的元组时，对表2的查询不再继续。
- 找到表1的第二条元组，然后从刚才的中断点处继续顺序扫描表2，查找满足连接条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。直接遇到表2中大于表1连接字段值的元组时，对表2的查询不再继续。
- 重复上述操作，直到表1或表2中的全部元组都处理完毕为止。

■ 索引连接(Index-Join)

- 对表2按连接字段建立索引
- 对表1中的每个元组，依次根据其连接字段值查询表2的索引，从中找到满足条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组

自然连接

[例 3.50] 对[例 3.49]用自然连接完成。

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade  
FROM   Student, SC  
WHERE  Student.Sno = SC.Sno;
```

- 自然连接后的表头显示顺序：

公共属性	第一张表剩余属性	第二张表剩余属性
------	----------	----------
- 一条SQL语句可以同时完成选择和连接查询，这时WHERE子句是由连接谓词和选择谓词组成的复合条件。

[例 3.51] 查询选修2号课程且成绩在90分以上的所有学生的学号和姓名。

```
SELECT Student.Sno, Sname  
FROM Student, SC  
WHERE SC.Cno='2' AND SC.Grade>90 AND Student.Sno=SC.Sno;
```

- 问题：如何判断一个给定的SQL语句中的连接是等值连接还是自然连接？
 - 两张表有相同的属性名和数据类型，使用natural join关键词表明

连接查询

- 等值与非等值连接查询
- **自身连接**
- 外连接
- 多表连接

2.自身连接

- 什么是自身连接？

- 一个表与其自己进行的连接

- 自身连接的使用方法

- 需要给表起别名以示区别
- 由于所有属性名都是同名属性，因此必须使用别名前缀

[例3.52] 查询每一门课的间接先修课(即先修课的先修课)。

```
SELECT FIRST.Cno, SECOND.Cpno  
FROM Course FIRST, Course SECOND  
WHERE FIRST.Cpno = SECOND.Cno;
```

FIRST表 (Course表)

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4

SECOND表 (Course表)

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4



Cno	Cpno
1	7
3	5
5	6

连接查询

- 等值与非等值连接查询
- 自身连接
- **外连接**
- 多表连接

3.外连接

■ 外连接类型

– (全)外连接、左外连接和右外连接

连接类型	特点
左外连接	<ul style="list-style-type: none">• Left out join• 列出左边关系中所有的元组
右外连接	<ul style="list-style-type: none">• Right out join• 列出右边关系中所有的元组
外连接	<ul style="list-style-type: none">• Outer join, full outer join• 以指定表为连接主体，将主体表中不满足连接条件的元组一并输出• 左外连接与右外连接的并集
(内)连接	<ul style="list-style-type: none">• Inner join, join• 只输出满足连接条件的元组

[例3. 53] 改写[例 3.49]

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade
FROM Student LEFT OUTER JOIN SC ON (Student.Sno=SC.Sno);
```

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
201215121	李勇	男	20	CS	1	92
201215121	李勇	男	20	CS	2	85
201215121	李勇	男	20	CS	3	88
201215122	刘晨	女	19	CS	2	90
201215122	刘晨	女	19	CS	3	80
201215123	王敏	女	18	MA	NULL	NULL
201215125	张立	男	19	IS	NULL	NULL

■ 课堂练习

- 使用右外连接和外连接改写上述语句，并比较这三者结果的异同。
- 找出所有没有选修任何课程的学生姓名。

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade  
FROM Student RIGHT OUTER JOIN SC ON (Student.Sno=SC.Sno);
```

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade  
FROM Student OUTER JOIN SC ON (Student.Sno=SC.Sno);
```

```
Select sname  
from student LEFT OUTER JOIN SC ON (student.sno=sc.sno)  
where Cno IS NULL;
```

连接查询

- 等值与非等值连接查询
- 自身连接
- 外连接
- **多表连接**

4.多表连接

- 什么是多表连接?
 - 两个以上表的连接

[例3.54] 查询每个学生的学号、姓名、选修的课程名及成绩。

```
SELECT Student.Sno, Sname, Cname, Grade  
FROM   Student, SC, Course /*多表连接*/  
WHERE  Student.Sno=SC.Sno AND SC.Cno=Course.Cno;
```

openGauss连接示例

- 官网:

<https://www.opengauss.org/zh/docs/3.0.0/docs/BriefTutorial/JOIN.html>

数据查询

- 单表查询
- 连接查询
- **嵌套查询**
- 集合查询
- 基于派生表的查询
- Select语句的一般形式

嵌套查询

- 查询块(query block): 构造嵌套查询的基本单元
 - SELECT...FROM...WHERE...(S-F-W)称为一个查询块
- 什么是嵌套查询(Embedded query)
 - 也称子查询(subquery), 将一个查询块嵌套在另一个查询块的WHERE子句或HAVING短语的条件中的查询称为嵌套查询

```
SELECT Sname /*外层查询/父查询*/  
FROM Student  
WHERE Sno IN  
        (SELECT Sno /*内层查询/子查询*/  
         FROM SC  
         WHERE Cno='2');
```

■ 构造子查询应遵循的几个规则：

- 子查询必须用括号括起来。
- 子查询的SELECT后面只能有一列。
- 子查询返回多行，只能与多值运算符一起使用，如IN运算符。
- 子查询不能用Order by子句
 - order by只能在主查询中使用，且至多1次

■ 子查询分类

- 不相关子查询：子查询的查询条件不依赖于父查询
- 相关子查询：子查询的查询条件依赖于父查询

子查询的求解算法

■ 不相关子查询的求解方法

- 由里向外逐层处理。即每个子查询在上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。

■ 相关子查询的求解方法

- 首先取外层查询中表的第一个元组，根据它与内层查询相关的属性值处理内层查询，若WHERE子句返回值为真，则取此元组放入结果表。
- 然后再取外层表的下一个元组。
- 重复这一过程，直至外层表全部检查完为止。

嵌套查询

- **带有IN谓词的子查询**
- 带有比较运算符的子查询
- 带有ANY(SOME)或ALL谓词的子查询
- 带有EXISTS谓词的子查询

[例3.55] 查询与“刘晨”在同一个系学习的学生。

```
SELECT Sno, Sname, Sdept
FROM Student
WHERE Sdept IN ( SELECT Sdept
                  FROM Student
                  WHERE Sname='刘晨');
```

```
SELECT S1.Sno, S1.Sname, S1.Sdept
FROM Student S1, Student S2
WHERE S2.Sname='刘晨' AND S1.Sdept=S2.Sdept;
```

[例3.56] 查询选修了课程名为“信息系统”的学生学号和姓名。

```
SELECT Sno,Sname
FROM Student
WHERE Sno IN
      (SELECT Sno
       FROM SC
       WHERE Cno IN
            (SELECT Cno
             FROM Course
             WHERE Cname= '信息系统'));
```

```
SELECT Sno, Sname
FROM Student, SC, Course
WHERE Course.Cname='信息系统' AND Student.Sno=SC.Sno AND SC.Cno=Course.Cno;
```

嵌套查询

- 带有IN谓词的子查询
- **带有比较运算符的子查询**
- 带有ANY(SOME)或ALL谓词的子查询
- 带有EXISTS谓词的子查询

- 当能确切知道内层查询返回单值时，可用比较运算符(>, <, =, >=, <=, !=或<>)
- 在[例 3.55]中，由于一个学生只可能在一个系学习，则可以用 = 代替IN.

```
SELECT Sno, Sname, Sdept
FROM Student
WHERE Sdept = ( SELECT Sdept
                  FROM Student
                  WHERE Sname= '刘晨');
```


[例3.57] 找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT Sno, Cno  
FROM SC x  
WHERE Grade >= ( SELECT AVG(Grade)  
                  FROM SC y  
                  WHERE y.Sno=x.Sno);
```

相关子查询



```
(201215121,1)  
(201215121,3)  
(201215122,2)
```

课堂练习

- 设有表book(类编号, 图书名, 出版社, 价格), 要求查询book表中大于该类图书价格平均值的图书名称。

```
SELECT 图书名  
FROM book a  
WHERE 价格 > (SELECT avg(价格)  
               FROM book b  
               WHERE a.类编号=b.类编号);
```

嵌套查询

- 带有IN谓词的子查询
- 带有比较运算符的子查询
- **带有ANY(SOME)或ALL谓词的子查询**
- 带有EXISTS谓词的子查询


- 当使用ANY或ALL谓词时必须同时使用比较运算。

运算符	语义
> ANY	大于子查询结果中的某个值
> ALL	大于子查询结果中的所有值
< ANY	小于子查询结果中的某个值
< ALL	小于子查询结果中的所有值
!=或 <>ANY	不等于子查询结果中的某个值
!=或 <>ALL	不等于子查询结果中的任何一个值

运算符	语义
>=ANY	大于等于子查询结果中的某个值
>= ALL	大于等于子查询结果中的所有值
<=ANY	小于等于子查询结果中的某个值
<=ALL	小于等于子查询结果中的所有值
=ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值, 通常无实际意义

[例3.58] 查询非计算机科学系中比计算机科学系任意学生年龄小的学生姓名和年龄。

```
SELECT Sname, Sage
FROM Student
WHERE Sage < ANY(SELECT Sage
                  FROM Student
                  WHERE Sdept='CS') AND Sdept <> 'CS';
/*父查询块中的条件 */
```



Sname	Sage
王敏	18
张立	19

```
SELECT Sname,Sage
FROM Student
WHERE Sage < (SELECT MAX(Sage)
              FROM Student
              WHERE Sdept = 'CS') AND Sdept <> 'CS';
```



用聚集函数实现

[例3.59] 查询非计算机科学系中比计算机科学系所有学生年龄小的学生姓名和年龄。

```
SELECT Sname, Sage
FROM Student
WHERE Sage < ALL(SELECT Sage
                  FROM Student
                  WHERE Sdept='CS') AND Sdept <>'CS';
/*父查询块中的条件 */
```



Sname	Sage
王敏	18

```
SELECT Sname,Sage
FROM Student
WHERE Sage < (SELECT MIN(Sage)
              FROM Student
              WHERE Sdept = 'CS') AND Sdept <>'CS';
```



用聚集函数实现

表3.7 ANY/SOME/ALL谓词与聚集函数、 IN谓词的等价转换关系

	=	<>或!=	<	<=	>	>=
ANY	IN	--	<MAX	<=MAX	>MIN	>= MIN
ALL	--	NOT IN	<MIN	<= MIN	>MAX	>= MAX

嵌套查询

- 带有IN谓词的子查询
- 带有比较运算符的子查询
- 带有ANY(SOME)或ALL谓词的子查询
- **带有EXISTS谓词的子查询**

■ 测试空关系可使用带EXISTS谓词的子查询实现

■ EXISTS谓词

- 存在量词 \exists
- 带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值 “true” 或逻辑假值 “false” 。
 - 若内层查询结果非空，则外层的WHERE子句返回真值；
 - 若内层查询结果为空，则外层的WHERE子句返回假值。
- 由EXISTS引出的子查询，其目标列表表达式通常都用 *，因为带EXISTS的子查询只返回真值或假值，给出列名无实际意义。

■ NOT EXISTS谓词

- 若内层查询结果非空，则外层的WHERE子句返回假值；
- 若内层查询结果为空，则外层的WHERE子句返回真值。

[例 3.60] 查询所有选修了1号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE EXISTS (SELECT *
               FROM SC
               WHERE Cno='1' AND Sno=Student.Sno);
```

是否为相关子查询?

[例 3.61] 查询没有选修1号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS (SELECT *
                  FROM SC
                  WHERE Cno='1' AND Sno=Student.Sno);
```

- 带有EXISTS谓词的子查询
- 不同形式查询间的替换
 - 一些带EXISTS或NOT EXISTS谓词的子查询不能被其他形式的子查询等价替换
 - 所有带IN谓词、比较运算符、ANY和ALL谓词的子查询都能用带EXISTS谓词的子查询等价替换
- 用EXISTS/NOT EXISTS实现全称量词(难点)
 - SQL语言中没有全称量词 \forall (For all)
 - 可以把带有全称量词的谓词转换为等价的带有存在量词的谓词:
$$(\forall x)P \equiv \neg(\exists x(\neg P))$$

[例3.55] 查询与“刘晨”在同一个系学习的学生。

```
SELECT Sno, Sname, Sdept
FROM Student S1
WHERE EXISTS ( SELECT *
                FROM Student S2
                WHERE S2.Sdept = S1.Sdept AND S2.Sname = '刘晨');
```

[例3.62] 查询选修了全部课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS (SELECT *
                  FROM Course
                  WHERE NOT EXISTS (SELECT *
                                    FROM SC
                                    WHERE Sno=Student.Sno AND Cno= Course.Cno));
```

■ 用EXISTS/NOT EXISTS实现逻辑蕴涵(难点)

- SQL语言中没有蕴涵(Implication)逻辑运算
- 可以利用谓词演算将逻辑蕴涵谓词等价转换为: $p \rightarrow q \equiv \neg p \vee q$

[例3.63]查询至少选修了学生201215122选修的全部课程的学生号码。

解题思路:

用逻辑蕴涵表达: 查询学号为x的学生, 对所有的课程y, 只要201215122学生选修了课程y, 则x也选修了y。

形式化表示:

用P表示谓词 “学生201215122选修了课程y”

用q表示谓词 “学生x选修了课程y”

则上述查询为: $(\forall y) p \rightarrow q$

- 等价变换：

$$\begin{aligned}(\forall y)p \rightarrow q &\equiv \neg(\exists y(\neg(p \rightarrow q))) \\ &\equiv \neg(\exists y(\neg(\neg p \vee q))) \\ &\equiv \neg\exists y(p \wedge \neg q)\end{aligned}$$

- 变换后语义：不存在这样的课程y，学生201215122选修了y，而学生x没有选。

- 用NOT EXISTS谓词表示:

SELECT DISTINCT Sno

FROM SC SCX

WHERE NOT EXISTS (SELECT *

FROM SC SCY

WHERE SCY.Sno = '201215122' AND

NOT EXISTS (SELECT *

FROM SC SCZ

WHERE SCZ.Sno=SCX.Sno AND SCZ.Cno=SCY.Cno));

--不存在

--有一门课程

--该同学没有学过

数据查询

- 单表查询
- 连接查询
- 嵌套查询
- **集合查询**
- 基于派生表的查询
- Select语句的一般形式

集合查询

- 集合操作是SQL的一个重要特点。
- 集合操作分类
 - 并操作UNION
 - 交操作INTERSECT
 - 差操作EXCEPT
- 参加集合操作的各查询结果的列数必须相同，对应项的数据类型也必须相同。
- 所有的集合操作具有相同的优先级，除非碰到括号。

[例 3.64] 查询计算机科学系的学生及年龄不大于19岁的学生。

```
SELECT *  
FROM Student  
WHERE Sdept= 'CS'  
UNION  
SELECT *  
FROM Student  
WHERE Sage<=19;
```

- UNION：将多个查询结果合并起来时，系统自动去掉重复元组
- UNION ALL：将多个查询结果合并起来时，保留重复元组

[例 3.65] 查询选修了课程1或者选修了课程2的学生。

```
SELECT Sno  
FROM SC  
WHERE Cno='1'  
UNION  
SELECT Sno  
FROM SC  
WHERE Cno='2';
```

[例 3.66] 查询计算机科学系的学生与年龄不大于19岁的学生的交集。

```
SELECT *  
FROM Student  
WHERE Sdept= 'CS'  
INTERSECT  
SELECT *  
FROM Student  
WHERE Sage<=19;
```



```
SELECT *  
FROM Student  
WHERE Sdept= 'CS' AND Sage<=19;
```

[例 3.67] 查询既选修了课程1又选修了课程2的学生。

```
SELECT Sno  
FROM SC  
WHERE Cno='1'  
INTERSECT  
SELECT Sno  
FROM SC  
WHERE Cno='2';
```



```
SELECT Sno  
FROM SC  
WHERE Cno='1' AND Sno IN  
      (SELECT Sno  
       FROM SC  
       WHERE Cno='2');
```

[例 3.68] 查询计算机科学系的学生与年龄不大于19岁的学生的差集。

```
SELECT *  
FROM Student  
WHERE Sdept='CS'  
EXCEPT  
SELECT *  
FROM Student  
WHERE Sage<=19;
```



```
SELECT *  
FROM Student  
WHERE Sdept='CS' AND Sage>19;
```

- openGauss、oracle用到的差集关键词为MINUS，而不是EXCEPT

数据查询

- 单表查询
- 连接查询
- 嵌套查询
- 集合查询
- **基于派生表的查询**
- Select语句的一般形式

基于派生表的查询

- 复杂查询

- 很难或根本不可能用一个SQL查询块或几个SQL查询块的并、交、差来解决的查询。

- 基于派生表的查询

- 子查询出现在FROM子句中的查询
- 此时子查询生成的临时派生表(Derived Table)成为主查询的查询对象
- 是表达复杂查询的一种方法

- 通常，FROM <子查询>表达的查询结果可用新的关系进行命名，包括属性的重新命名

- 用as子句实现

[例3.57] 找出每个学生超过他自己选修课程平均成绩的课程号。

```
SELECT Sno, Cno
FROM SC, (SELECT Sno, Avg(Grade) FROM SC GROUP BY Sno)
        AS Avg_sc(avg_sno, avg_grade)
WHERE SC.Sno=Avg_sc.avg_sno AND SC.Grade >=Avg_sc.avg_grade;
```

[课堂练习] 找出每个同学其选课平均成绩超过80分的同学姓名。

```
SELECT Sname
FROM Student, (SELECT Sno, Avg(Grade) FROM SC GROUP BY Sno) AS
S (AVG_sno, avg_grade)
WHERE Student.sno=S.avg_sno AND S.avg_grade>80;
```

小结：派生查询的典型应用场景：涉及聚集操作；对聚集结果再查询

- 如果子查询中没有聚集函数，派生表可以不指定属性列，子查询SELECT子句后面的列名为其缺省属性。

[例3.60] 查询所有选修了1号课程的学生姓名，可以用如下查询完成。

```
SELECT Sname  
FROM Student, (SELECT Sno FROM SC WHERE Cno='1') AS SC1  
WHERE Student.Sno = SC1.Sno;
```

数据查询

- 单表查询
- 连接查询
- 嵌套查询
- 集合查询
- 基于派生表的查询
- **SELECT语句的一般形式**

SELECT语句的一般格式

SELECT [ALL|DISTINCT] <目标列表达式> [别名] [, <目标列表达式> [别名]] ...

FROM <表名或视图名> [别名] [, <表名或视图名> [别名]] ...

|(<SELECT语句>)[AS]<别名>

[**WHERE** <条件表达式>]

[**GROUP BY** <列名1> [**HAVING** <条件表达式>]]

[**ORDER BY** <列名2> [ASC|DESC]];

目标列表表达式的可选格式

- 目标列表表达式格式:

- (1) *

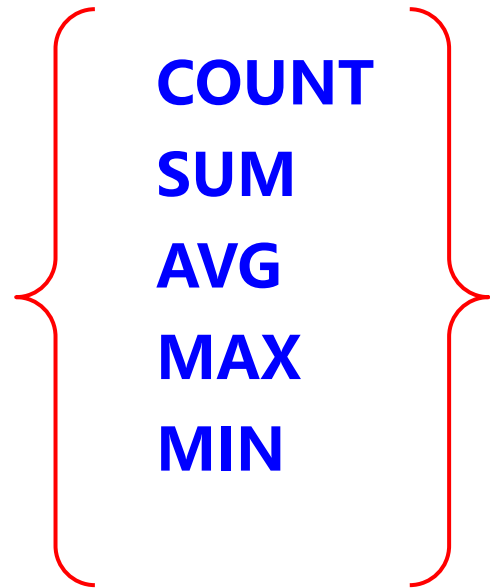
- (2) <表名>.*

- (3) COUNT([DISTINCT|ALL]*)

- (4) [<表名>.]<属性列名表达式>[,<表名>.]<属性列名表达式>]...

- 其中<属性列名表达式>可以是由属性列、作用于属性列的聚集函数和常量的任意算术运算 (+, -, *, /) 组成的运算公式

聚集函数的一般格式

**COUNT**
SUM
AVG
MAX
MIN

([DISTINCT|ALL] <列名>)

WHERE子句的条件表达式的可选格式

(1)

<属性列名> θ $\left\{ \begin{array}{l} \text{<属性列名>} \\ \text{<常量>} \\ \text{[ANY|ALL] (SELECT语句)} \end{array} \right\}$

(2)

<属性列名> [NOT] BETWEEN $\left\{ \begin{array}{l} \text{<属性列名>} \\ \text{<常量>} \\ \text{(SELECT语句)} \end{array} \right\}$ AND $\left\{ \begin{array}{l} \text{<属性列名>} \\ \text{<常量>} \\ \text{(SELECT语句)} \end{array} \right\}$

(3) <属性列名> [NOT] IN { (<值1>[, <值2>] ...) }
(SELECT语句)

(4) <属性列名> [NOT] LIKE <匹配串>

(5) <属性列名> IS [NOT] NULL

(6) [NOT] EXISTS (SELECT语句)

(7) <条件表达式> { AND } <条件表达式> { AND } <条件表达> ...
OR

课堂练习

- 关系R包含A, B, C三个属性:

A	B	C
10	NULL	20
20	30	NULL

- 写出对查询语句SELECT * FROM R WHERE X;当X为下列条件的查询结果
 - A IS NULL
 - $A > 8$ AND $B < 20$
 - $C + 10 > 25$
 - EXISTS (SELECT B FROM R WHERE $A = 10$)
 - C IN (SELECT B FROM R)