数据库系统之二
--数据库语言-SQL

战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

第7讲 SQL语言之复杂查询与视图

战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

本讲学习什么?

战德臣 教授

基本内容

1. SQL语言之子查询运用
2. SQL语言之结果计算与聚集函数
3. SQL语言之分组查询与分组过滤
4. 利用SQL语言实现关系代数操作
5. SQL语言之视图及其应用

重点与难点

- SQL-SELECT：IN | NOT IN, θ some, θ all, Exists | NOT Exists
- SQL-SELECT：聚集函数，GROUP BY, HAVING
- 视图及其应用

利用SQL语言表达复杂查询

(NOT) IN子查询

战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

利用SQL语言表达复杂查询—(NOT) IN子查询

(1)子查询

战德臣 教授

为什么需要子查询？

- 现实中，很多情况需要进行下述条件的判断
  - ❑ 集合成员资格
    - ✓某一元素是否是某一个集合的成员
  - ❑ 集合之间的比较
    - ✓某一个集合是否包含另一个集合等
  - ❑ 集合基数的测试
    - ✓测试集合是否为空
    - ✓测试集合是否存在重复元组

这是SQL语言的亮点之一

- 子查询：出现在Where子句中的Select语句被称为子查询(subquery)，子查询返回了一个集合，可以通过与这个集合的比较来确定另一个查询集合。
- 三种类型的子查询：(NOT) IN-子查询；θ-Some/ θ-All子查询；(NOT) EXISTS子查询

(NOT) IN子查询

- 基本语法：

表达式 [*not*] *in* (子查询)

- 语法中，表达式的最简单形式就是列名或常数。
- 语义：判断某一表达式的值是否在子查询的结果中。

示例：列出张三、王三同学的所有信息

```
Select  *  From  Student
Where  Sname  in  (“张三”, “王三”);
```

//此处直接使用了某一子查询的结果集合。如果该集合是已知的固定的，可以如上直接书写

- 上述示例相当于

```
Select  *  From  Student
Where  Sname = “张三”  or  Sname = “王三”;
```

利用SQL语言表达复杂查询—(NOT) IN子查询
(2) IN与NOT IN谓词子查询

战德臣 教授

示例：列出选修了001号课程的学生的学号和姓名

```
Select   S#, Sname   From  Student
Where   S#   in  ( Select    S#   From  SC  Where   C# = '001' ) ;
```

示例：求既学过001号课程，又学过002号课程的学生的学号

```
Select   S#   From  SC
Where   C# = '001'   and
        S#   in  ( Select    S#   From  SC  Where   C# = '002' ) ;
```

还能够怎样书写?

一种查询多种书写方式是好还是不好?

利用SQL语言表达复杂查询—(NOT) IN子查询
(2) IN与NOT IN谓词子查询

战德臣 教授

示例：列出没学过李明老师讲授课程的所有同学的姓名?

```
Select   Sname   From  Student
Where  S#  not  in ( Select   S#   From  SC, Course  C, Teacher  T
                    Where   T.Tname = '李明'  and  SC.C# = C.C#
                            and  T.T# = C.T#  );
```

➢注意下述写法的不正确性！

```
Select   Sname   From  Student  S, SC, Course  C, Teacher  T
Where  T.Tname <> '李明'   and  C.C# = SC.C#
       and  SC.S# = S.S#  and  T.T# = C.T#;
```

为什么不正确?

利用SQL语言表达复杂查询—(NOT) IN子查询
(3) 非相关子查询

战德臣 教授

➢带有子查询的Select语句区分为内层和外层

外层查询

```
Select   Sname
From  Student
Where   S#   not  in ( Select    S#
                      From  SC, Course  C, Teacher  T
                      Where   T.Tname = '李明'  and  SC.C# = C.C#
                                    and  T.T# = C.T# ) ;
```

内层查询

➢**非相关子查询**：内层查询独立进行，没有涉及任何外层查询相关信息的子查询

➢ 前面的子查询示例都是非相关子查询

战德臣 教授

➢**相关子查询**：内层查询需要依靠外层查询的某些参量作为限定条件才能进行的子查询

➢ 外层向内层传递的参量需要使用外层的表名或表别名来限定

示例：求学过001号课程的同学的姓名

```
Select   Sname
From  Student  Stud
Where   S#   in ( Select    S#
                  From  SC
                  Where   S# = Stud.S#   and  C# = '001'  );
```

➢ 注意：相关子查询只能由外层向内层传递参数，而不能反之；这也称为变量的作用域原则。

利用SQL语言表达复杂查询

θ Some与θ All子查询

战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

利用SQL语言表达复杂查询--θ Some与θ All子查询
(1)子查询

战德臣 教授

为什么需要子查询？

- 现实中，很多情况需要进行下述条件的判断
  - ❑ 集合成员资格
    - ✓某一元素是否是某一个集合的成员
  - ❑ 集合之间的比较
    - ✓某一个集合是否包含另一个集合等
  - ❑ 集合基数的测试
    - ✓测试集合是否为空
    - ✓测试集合是否存在重复元组

这是SQL语言的亮点之一

- 子查询：出现在Where子句中的Select语句被称为子查询(subquery)，子查询返回了一个集合，可以通过与这个集合的比较来确定另一个查询集合。
- 三种类型的子查询：(NOT) IN-子查询；θ-Some/ θ-All子查询；(NOT) EXISTS子查询

θ some / θ all子查询

➢基本语法：

表达式 *θ some* (子查询)

表达式 *θ all* (子查询)

与“某一个”比较，还是与“所有”比较？

➢ 语法中， *θ* 是比较运算符： < , > , >= , <= , = , <>。

➢ 语义：将表达式的值与子查询的结果进行比较：

❑ 如果表达式的值至少与子查询结果的某一个值相比较满足*θ*关系，则“表达式 *θ some* (子查询)”的结果便为真；

❑ 如果表达式的值与子查询结果的所有值相比较都满足*θ*关系，则“表达式 *θ all* (子查询)”的结果便为真；

利用SQL语言表达复杂查询--θ Some与θ All子查询
(2) θ some / θ all谓词子查询

战德臣 教授

示例：找出工资最低的教师姓名

```
Select   Tname   From  Teacher
Where   Salary  <=  all  ( Select   Salary  From  Teacher );
```

利用元组演算怎样表达?

示例：找出001号课成绩不是最高的所有学生的学号

```
Select   S#   From  SC
Where   C# = “001”  and
        Score <  some ( Select   Score  From  SC  Where   C# = “001” );
```

利用SQL语言表达复杂查询--θ Some与θ All子查询
(2) θ some / θ all谓词子查询

战德臣 教授

示例：找出所有课程都不及格的学生姓名(相关子查询)

```
Select   Sname   From  Student
Where   60 > all  ( Select   Score  From  SC
                    Where   S# = Student.S# );
```

请同学们书写满足下述条件的查询语句

- 找出001号课成绩最高的所有学生的学号
- 找出98030101号同学成绩最低的课程号
- 找出张三同学成绩最低的课程号

暂停视频
书写一下...

利用SQL语言表达复杂查询--θ Some与θ All子查询
(2) θ some / θ all谓词子查询

战德臣 教授

请同学们书写满足下述条件的查询语句

- **找出001号课成绩最高的所有学生的学号**

```
Select   S#   From  SC
Where   C# = “001”  and
        Score >= all ( Select   Score  From  SC  Where   C# = “001” );
```

- **找出98030101号同学成绩最低的课程号**

非相关子查询

```
Select   C#   From  SC
Where   S# = “98030101”  and
        Score <= all ( Select   Score  From SC Where S# = “98030101” );
```

- **找出张三同学成绩最低的课程号**

相关子查询

```
Select   C#   From  SC, Student S
Where   Sname = “张三”  and S.S#=SC.S# and
        Score <= all ( Select   Score  From  SC
                       Where  S#=S.S# );
```

战德臣 教授

为什么不用 θ any？

➢在SQL标准中，也有θ any 谓词，但由于其语义的模糊性：any, “任一”是指所有呢？还是指某一个？不清楚，所以被 θ some替代以求更明晰。

示例：求工资小于任一教师的教师姓名？ 下面哪一种写法正确呢？

```
Select   Tname   From  Teacher
Where   Salary  < any ( Select   Salary  From  Teacher  );
```

怎么理解呢?

```
Select   Tname   From  Teacher
Where   Salary  < some  ( Select   Salary  From  Teacher  );
```

```
Select   Tname   From  Teacher
Where   Salary  < all  ( Select   Salary  From  Teacher  );
```

等价性变换需要注意

- 如下两种表达方式含义是相同的

  表达式 = *some* (子查询)

  表达式 *in* (子查询)

示例

```
Select   Sname   From  Student  S
Where   S#   in ( Select    S#   From  SC
                    Where   S# = S.S#   and  C# = '001'  ) ;
```

```
Select   Sname   From  Student  S
Where   S# = some  ( Select    S#   From  SC
                    Where   S# = S.S#   and  C# = '001'  ) ;
```

<> Some是否等价于 not in?

利用SQL语言表达复杂查询--θ Some与θ All子查询
(3) 需要注意的

➢如下两种表达方式含义却是不同的，请注意

表达式 ***not in*** (子查询)

表达式 ***<> some*** (子查询)

与*not in*等价的是

表达式 ***<> all*** (子查询)

示例

```
Select   Sname   From  Student  S
Where   S#   not in ( Select    S#   From  SC
                  Where   S# = S.S#   and  C# = '001'  ) ;
```

```
Select   Sname   From  Student  S
Where   S# <>some  ( Select    S#   From  SC
                  Where   S# = S.S#   and  C# = '001'  ) ;
```

结果是什么呢?

```
Select   Sname   From  Student  S
Where   S# <> all  ( Select    S#   From  SC
                  Where   S# = S.S#   and  C# = '001'  ) ;
```

利用SQL语言表达复杂查询
--(NOT) EXISTS子查询

战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

利用SQL语言表达复杂查询--(NOT) EXISTS子查询
(1)子查询

战德臣 教授

为什么需要子查询？

- 现实中，很多情况需要进行下述条件的判断
  - 集合成员资格
    - ✓某一元素是否是某一个集合的成员
  - 集合之间的比较
    - ✓某一个集合是否包含另一个集合等
  - 集合基数的测试
    - ✓测试集合是否为空
    - ✓测试集合是否存在重复元组

这是SQL语言的亮点之一

- 子查询：出现在Where子句中的Select语句被称为子查询(subquery)，子查询返回了一个集合，可以通过与这个集合的比较来确定另一个查询集合。
- 三种类型的子查询：(NOT) IN-子查询；θ-Some/ θ-All子查询；(NOT) EXISTS子查询

战德臣 教授

(NOT) EXISTS 子查询

➢基本语法：

[not] Exists (子查询)

➢语义：子查询结果中有无元组存在

示例：检索选修了赵三老师主讲课程的所有同学的姓名

```
Select  DISTINCT  Sname   From  Student
Where  exists ( Select   *   From  SC, Course, Teacher
                    Where   SC.C# = Course.C#   and SC. S# = Student.S#
                        and  Course.T# = Teacher.T#  and Tname = '赵三' ) ;
```

➢不加not形式的Exists谓词可以不用，比如上面例子就可以直接写成：

```
Select  DISTINCT  Sname   From  Student, SC, Course, Teacher
Where   SC.C# = Course.C#   and   SC.S# = Student.S#
        and  Course.T# = Teacher.T#  and Tname = '赵三' ) ;
```

利用SQL语言表达复杂查询--(NOT) EXISTS子查询
(2) EXISTS 与NOT EXISTS谓词子查询

战德臣 教授

➢然而not Exists却可以实现很多新功能

示例：检索学过001号教师主讲的所有课程的所有同学的姓名

```
Select  Sname  From  Student
Where  not  exists                                          //不存在
        ( Select  *  From  Course                           //有一门001教师主讲课程
          Where Course.T# = '001'  and not  exists          //该同学没学过
                ( Select  *  From  SC
                  Where  S# = Student.S# and C# = Course.C# ) );
```

➢上述语句的意思：不存在有一门001号教师主讲的课程该同学没学过

利用元组演算怎样表达?

利用关系代数怎样表达?

利用SQL语言表达复杂查询--(NOT) EXISTS子查询
(2) EXISTS 与NOT EXISTS谓词子查询

战德臣 教授

示例：列出没学过李明老师讲授任何一门课程的所有同学的姓名

```
Select   Sname  From  Student
Where  not  exists                                          //不存在
        ( Select   *   From   Course, SC, Teacher           //学过一门课程
          Where Tname='李明' and Course.T# =Teacher.T#
                and Course.C# = SC.C#  and S# = Student.S# );
```

利用SQL语言表达复杂查询--(NOT) EXISTS子查询
(2) EXISTS 与NOT EXISTS谓词子查询

战德臣 教授

示例：列出至少学过98030101号同学学过所有课程的同学的学号

```
Select   DISTINC  S#  From  SC  SC1
Where  not  exists                                          //不存在
        ( Select   *   From   SC  SC2                       //有一门课程
          Where SC2.S# = '98030101' and   not  exists       //该同学没学过
                ( Select   *   From   SC
                  Where   C# = SC2.C#  and   S# = SC1.S# ) );
```

利用元组演算怎样表达?

利用关系代数怎样表达?

$\pi_{S\#,\ C\#}(SC) \div \pi_{C\#}(\sigma_{S\#='98030101'}(SC))$

$\{ t[S\#] \mid t \in SC \wedge \forall(u \in SC \wedge u[S\#]='98030101')$

$(\exists(w \in SC)(w[S\#]=t[S\#] \wedge w[C\#]=u[C\#])) \}$

利用SQL语言表达复杂查询--(NOT) EXISTS子查询
(2) EXISTS 与NOT EXISTS谓词子查询

战德臣 教授

示例：已知SPJ(Sno, Pno, Jno, Qty), 其中Sno供应商号，Pno零件号，Jno工程号，Qty数量，列出至少用了供应商S1供应的全部零件的工程号

```
Select   DISTINCT  Jno  From  SPJ   SPJ1
Where  not  exists                                          //不存在
        ( Select   *  From   SPJ   SPJ2                     //有一种S1的零件
          Where SPJ2.Sno = 'S1' and   not  exists           //该工程没用过
                ( Select   *   From   SPJ  SPJ3
                  Where   SPJ3.Pno = SPJ2.Pno
                          and   SPJ3.Jno = SPJ1.Jno ) );
```

利用SQL语言进行结果计算与聚集计算

战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

结果计算

➢Select-From-Where语句中，Select子句后面不仅可是列名，而且可是一些计算表达式或聚集函数，表明在投影的同时直接进行一些运算

Select 列名 | **expr** | **agfunc(列名)** [[, 列名 | **expr** | **agfunc(列名)**] …]

From 表名**1** [, 表名**2** …]

[**Where** 检索条件] ;

➢expr可以是常量、列名、或由常量、列名、特殊函数及算术运算符构成的算术运算式。特殊函数的使用需结合各自DBMS的说明书

➢agfunc()是一些聚集函数

利用SQL语言进行结果计算与聚集计算
(1)结果计算

战德臣 教授

示例：求有差额(差额>0)的任意两位教师的薪水差额

```
Select   T1.Tname as TR1, T2.Tname  as TR2, T1.Salary – T2.Salary
From  Teacher  T1,  Teacher  T2
Where   T1.Salary > T2.Salary;
```

示例：依据学生年龄求学生的出生年份，当前是2015年

```
Select   S.S#, S.Sname, 2015 – S.Sage+1  as Syear
From  Student S;
```

聚集函数

- SQL提供了五个作用在简单列值集合上的内置聚集函数agfunc, 分别是：COUNT、SUM、AVG、MAX、MIN
- SQL聚集函数的参数类型、结果类型与作用如下：

Name	Argument type	Result type	Description
Count	any (can be *)	numeric	count of occurrences
sum	numeric	numeric	sum of arguments
avg	numeric	numeric	average of arguments
max	char or numeric	same as arg	maximum value
min	char or numeric	same as arg	minimum value

求个数　求 和　求平均　求最大　求最小

示例：求教师的工资总额

```
Select  Sum(Salary)  From  Teacher;
```

示例：求计算机系教师的工资总额

```
Select  Sum(Salary)  From  Teacher  T, Dept
Where  Dept.Dname = '计算机' and Dept.D# = T.D#;
```

示例：求数据库课程的平均成绩

```
Select  AVG(Score)  From  Course  C, SC
Where  C.Cname = '数据库' and C.C# = SC.C#;
```

怎样表达求每门课程的平均成绩?

利用SQL语言进行分组查询与分组过滤

战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

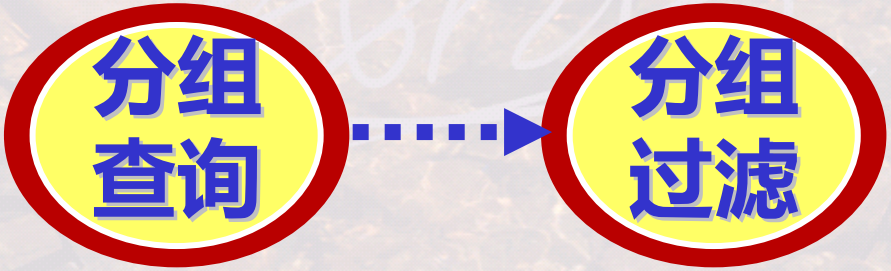
利用SQL语言进行分组查询与分组过滤

(1) 问题提出

战德臣 教授

下面的查询请求该如何表达呢？

- **求每一门课程的平均成绩**
- **求每一个学生的平均成绩**



战德臣 教授

分组：SQL可以将检索到的元组按照某一条件进行分类，具有相同条件值的元组划到一个组或一个集合中，同时处理多个组或集合的聚集运算。

- 分组的基本语法：

  Select 列名 | expr | agfunc(列名) [[, 列名 | expr | agfunc(列名)] …]

  From 表名1 [, 表名2 …]

  [***Where*** 检索条件]

  [***Group by*** 分组条件] ;

- 分组条件可以是

  列名1, 列名2, …

这是SQL语言的亮点之一

利用SQL语言进行分组查询与分组过滤
(3) 分组查询示例

战德臣 教授

示例： 求每一个学生的平均成绩

```
Select  S#, AVG(Score)  From  SC
Group  by  S#;
```

➢上例是按学号进行分组，即学号相同的元组划到一个组中并求平均值

S#	C#	Score
98030101	001	92
98030101	002	85
98030101	003	88
98040202	002	90
98040202	003	80
98040202	001	55
98040203	003	56
98030102	001	54
98030102	002	85
98030102	003	48

示例：求每一门课程的平均成绩

```
Select  C#, AVG(Score)  From  SC
Group  by  C#;
```

➢上例是按课号进行分组，即课号相同的元组划到一个组中并求平均值

S#	C#	Score
98030101	001	92
98040202	001	55
98030102	001	54
98030101	002	85
98030102	002	85
98040202	002	90
98040202	003	80
98030101	003	88
98040203	003	56
98030102	003	48

战德臣 教授

示例：求不及格课程超过两门的同学的学号

```
Select   S#  From  SC
Where  Score < 60  and  Count(*)>2
Group by  S#;
```

//正确的SQL语句在讲义后面的示例中讲解

利用SQL语言进行分组查询与分组过滤

(4) 分组过滤

战德臣 教授

➢聚集函数是不允许用于Where子句中的：Where子句是对每一元组进行条件过滤，而不是对集合进行条件过滤

分组过滤：若要对集合(即分组)进行条件过滤，即满足条件的集合/分组留下，不满足条件的集合/分组剔除。

➢**Having子句，又称分组过滤子句**。需要有Group by子句支持，换句话说，没有Group by子句，便不能有Having子句。

Select 列名 | **expr** | **agfunc(列名)** [[, 列名 | **expr** | **agfunc(列名)**] …]
From 表名1 [, 表名2 …]
[**Where** 检索条件]
[**Group by** 分组条件 [**Having** 分组过滤条件]] ;

又有扩展了…

战德臣 教授

示例：求不及格课程超过两门的同学的学号

```
Select   S#  From  SC
Where  Score < 60
Group by   S#   Having  Count(*)>2;
```

示例：求有10人以上不及格的课程号

```
Select   C#  From  SC
Where  Score < 60
Group by   C#   Having  Count(*)>10;
```

战德臣 教授

HAVING子句与WHERE子句表达条件的区别

每一分组检查满足与否的条件要用 **Having 子句表达**。

注意:不是每一行都检查，所以使用Having子句一定要有Group by子句

S#	C#	Score
98030101	001	92
98030101	002	85
98030101	003	88
98040202	002	90
98040202	003	80
98040202	001	55
98040203	003	56
98030102	001	54
98030102	002	85
98030102	003	48

每一行都要检查满足与否的条件要用 **WHERE子句表达**

利用SQL语言进行分组查询与分组过滤
(6) 分组过滤示例

战德臣 教授

➢**分组查询仍需要注意语义问题**

示例：求有两门以上不及格课程同学的学号及其平均成绩

```
Select   S#, Avg(Score)  From  SC
Where  Score < 60
Group by   S#   Having  Count(*)>2;
```

//正确的SQL语句在讲义后面的示例中讲解

为什么不正确呢?

它的结果是什么呢...

利用SQL语言进行分组查询与分组过滤
(6) 分组过滤示例

战德臣 教授

➢**分组查询仍需要注意语义问题**

示例：求有两门以上不及格课程同学的学号及其平均成绩

```
Select   S#, Avg(Score)  From  SC
Where  Score < 60
Group by   S#   Having  Count(*)>2;
```

//正确的SQL语句在讲义后面的示例中讲解

➢上例SQL语句求出的是“该同学那几门不及格课程的平均成绩”，而不是“该同学所有课程的平均成绩”。因此正确写法为：

```
Select   S#, Avg(Score)  From  SC
Where  S#  in
          ( Select   S#  From  SC
            Where  Score < 60
            Group by   S#   Having  Count(*)>2 )
Group by S# ;
```

利用SQL语言实现关系代数操作

战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

利用SQL语言实现关系代数操作

(1) 并-交-差的处理

战德臣 教授

- SQL语言：并运算 UNION, 交运算INTERSECT, 差运算EXCEPT。
- 基本语法形式：

  子查询 { Union [ALL] | Intersect [ALL] | Except [ALL] 子查询 }

- 通常情况下自动删除重复元组：不带ALL。若要保留重复的元组，则要带ALL。

假设子查询1的一个元组出现m次，子查询2的一个元组出现n次，则该元组在：

- ✓子查询1 Union ALL 子查询2 ， 出现m + n次
- ✓子查询1 Intersect ALL 子查询2 ， 出现min(m,n)次
- ✓子查询1 Except ALL 子查询2 ， 出现max(0, m – n)次

注意带ALL和不带ALL的差别

SQL并运算

示例：求学过002号课的同学或学过003号课的同学学号

```
Select  S#  From  SC  Where  C# = '002'
UNION
Select  S#  From  SC  Where  C# = '003';
```

- 上述语句也可采用如下不用UNION的方式来进行

```
Select  S#  From  SC  Where  C# = '002'  OR  C# = '003';
```

- 但有时也不能完全转换成不用UNION的方式

示例：已知两个表

Customers(CID, Cname, City, Discnt)

Agents(AID, Aname, City, Percent)

求客户所在的或者代理商所在的城市

```
Select  City  From  Customers
UNION
Select  City  From  Agents ;
```

战德臣 教授

SQL交运算

示例：求既学过002号课，又学过003号课的同学学号

```
Select  S#  From  SC  Where  C# = '002'
INTERSECT
Select  S#  From  SC  Where  C# = '003';
```

- 上述语句也可采用如下不用INTERSECT的方式来进行

```
Select  S#  From  SC  Where  C# = '002'  and  S#  IN
        (Select  S#  From  SC  Where C# = '003');
```

- 交运算符Intersect并没有增强SQL的表达能力，没有Intersect， SQL也可以用其他方式表达同样的查询需求。只是有了Intersect更容易表达一些，但增加了SQL语言的不唯一性。

SQL差运算

示例： 假定所有学生都有选课，求没学过002号课程的学生学号

- 不能写成如下形式：

```
Select  S#  From  SC  Where  C# <> '002'
```

- 可写成如下形式：所有学生 减掉 学过002号课的学生

```
Select  DISTINCT  S#  From  SC
EXCEPT
Select  S#  From  SC Where  C# = '002';
```

利用SQL语言实现关系代数操作
(1) 并-交-差的处理

战德臣 教授

- 前述语句也可不用EXCEPT的方式来进行

```
Select  DISTINCT  S#  From  SC  SC1
Where  not exists ( Select  *  From  SC
            Where  C# = '002'  and  S# = SC1.S#) ;
```

- 差运算符Except也没有增强SQL的表达能力，没有Except， SQL也可以用其他方式表达同样的查询需求。只是有了Except更容易表达一些，但增加了SQL语言的不唯一性。

利用SQL语言实现关系代数操作
(1) 并-交-差的处理

- UNION运算符是Entry-SQL92的一部分
- INTERSECT、EXCEPT运算符是Full-SQL92的一部分
- 它们都是Core-SQL99的一部分，但有些DBMS并不支持这些运算，使用时要注意。

利用SQL语言实现关系代数操作
(2) 空值的处理

战德臣 教授

- 空值是其值不知道、不确定、不存在的值
- 数据库中有了空值，会影响许多方面，如影响聚集函数运算的正确性，不能参与算术、比较或逻辑运算等
- 例如：右下图所示表SC, 如果有某一记录为空值，则求001号课程的平均成绩？会是多少呢？
- 以前，很多DBMS将空值按默认值处理，如字符串类型则以空格来表示，而如数值类型则以0来表示，这也将会引起统计、计算上的不正确性。

空值究竟影响哪些方面?

SC

S#	C#	Score
98030101	001	92
98040202	001	55
98030102	001	?

(2) 空值的处理

- 在SQL标准中和许多现流行的DBMS中，空值被用一种特殊的符号Null来标记，使用特殊的空值检测函数来获得某列的值是否为空值。
- 空值检测

is [*not*] *null*

测试指定列的值是否为空值

示例：找出年龄值为空的学生姓名

```
Select Sname From Student
Where Sage is null;
```

- 注意：上例条件不能写为**Where Sage = null;** 空值是不能进行运算的

为什么要写成IS NULL而不能写成= NULL?

利用SQL语言实现关系代数操作
(2) 空值的处理

战德臣 教授

- 现行DBMS的空值处理小结
  - 除is [not] null之外，空值不满足任何查找条件
  - 如果null参与算术运算，则该算术表达式的值为null
  - 如果null参与比较运算，则结果可视为false。在SQL-92中可看成unknown
  - 如果null参与聚集运算，则除count(*)之外其它聚集函数都忽略null

示例：

```
Select  AVG(Score)  From  SC;
```

上例的值(参考右图)为**73.5 = (92 + 55)/2**。

示例：

```
Select  COUNT(*)  From  SC;
```

上例的值为**3**。

SC

S#	C#	Score
98030101	001	92
98040202	001	55
98030102	001	?

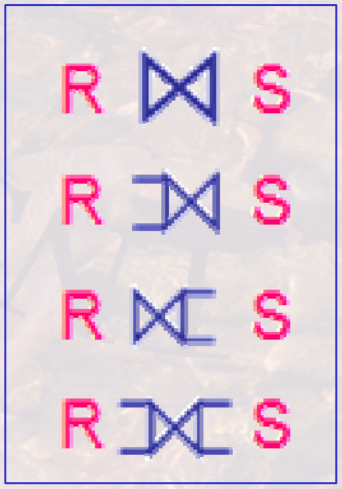
利用SQL语言实现关系代数操作
(3) 内连接、外连接

战德臣 教授

- 关系代数运算中，有连接运算，又分为θ连接和外连接
- 标准SQL语言中连接运算通常是采用

```
Select  列名  [ [, 列名] ... ]
From    表名1, 表名2,  ...
Where   检索条件 ;
```

- 即相当于采用$\Pi_{列名,\ldots,列名}(\sigma_{检索条件}(表名1 \times 表名2 \times \ldots))$。见前面的介绍。



利用SQL语言实现关系代数操作
(3) 内连接、外连接

战德臣 教授

➢SQL的高级语法中引入了内连接与外连接运算，具体形式：

Select 列名 [[, 列名] …]
From 表名1 [NATURAL]
[INNER | { LEFT | RIGHT | FULL} [OUTER]] JOIN 表名2
{ ON 连接条件 | Using (Colname {, Colname …}) }
[*Where* 检索条件] … ;

➢ 上例的连接运算由两部分构成：连接类型和连接条件

连接类型(四者选一)
inner join left outer join right outer join full outer join

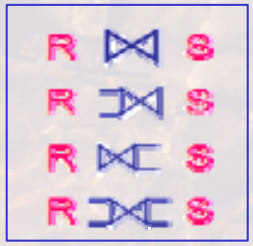
连接条件(三者选一)
natural on <连接条件> using (Col_1, Col_2, ..., Col_n)

利用SQL语言实现关系代数操作
(3) 内连接、外连接

战德臣 教授

- **Inner Join:** 即关系代数中的θ-连接运算
- **Left Outer Join, Right Outer Join, Full Outer Join:** 即关系代数中的外连接运算
  - 如“表1 Left Outer Join 表2”，则连接后，表1的任何元组t都会出现在结果表中，如表2中有满足连接条件的元组s, 则t与s连接；否则t与空值元组连接；
  - 如“表1 Right Outer Join 表2”，则连接后，表2的任何元组s都会出现在结果表中，如表1中有满足连接条件的元组t, 则t与s连接；否则s与空值元组连接；
  - 如“表1 Full Outer Join 表2”，是前两者的并。

连接类型(四者选一)
inner join left outer join right outer join full outer join



利用SQL语言实现关系代数操作
(3) 内连接、外连接

战德臣 教授

- 连接中使用 **natural**
  - 出现在结果关系中的两个连接关系的元组在公共属性上取值相等，且公共属性只出现一次
- 连接中使用 **on <连接条件>**
  - 出现在结果关系中的两个连接关系的元组取值满足连接条件，且公共属性出现两次
- 连接中使用 **using (Col_1, Col_2, ..., Col_n)**
  - (Col_1, Col_2, ..., Col_n)是两个连接关系的公共属性的子集，元组在(Col_1, Col_2, ..., Col_n)上取值相等，且(Col_1, Col_2, ..., Col_n)只出现一次

战德臣 教授

Inner Join

示例: 求所有教师的任课情况并按教师号排序(没有任课的教师也需列在表中)

```
Select   Teacher.T#, Tname, Cname
From  Teacher  Inner  Join  Course
          ON  Teacher.T# = Course.T#
Order by Teacher.T#  ASC;
```

Course

C#	Cname	Chours	Credit	T#
001	数据库	40	6	001
003	数据结构	40	6	003
004	编译原理	40	6	001
005	C 语言	30	4.5	003
002	高等数学	80	12	004

Teacher

T#	Tname	D#	Salary
001	赵三	01	1200.00
002	赵四	03	1400.00
003	赵五	03	1000.00
004	赵六	04	1100.00

任课情况(内连接)

T#	Tname	Cname
001	赵三	数据库
001	赵三	编译原理
003	赵五	数据结构
003	赵五	C 语言
004	赵六	高等数学

Outer Join

示例: 求所有教师的任课情况(没有任课的教师也需列在表中)

```
Select Teacher. T#, Tname, Cname
 From  Teacher  Left  Outer  Join  Course
          ON  Teacher.T# = Course.T#
Order by Teacher.T#  ASC ;
```

Course

C#	Cname	Chours	Credit	T#
001	数据库	40	6	001
003	数据结构	40	6	003
004	编译原理	40	6	001
005	C 语言	30	4.5	003
002	高等数学	80	12	004

Teacher

T#	Tname	D#	Salary
001	赵三	01	1200.00
002	赵四	03	1400.00
003	赵五	03	1000.00
004	赵六	04	1100.00

任课情况(左外连接)

T#	Tname	Cname
001	赵三	数据库
001	赵三	编译原理
002	赵四	
003	赵五	数据结构
003	赵五	C 语言
004	赵六	高等数学

回顾SQL-SELECT

战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

战德臣 教授

SQL：结构化查询语言

子查询

- SELECT ..FROM.. WHERE ..(子查询)
- [NOT] IN
- θ some/ θ all
- [NOT] EXISTS

关系代数的实现

- UNION
- UNION ALL
- EXCEPT
- EXCEPT ALL
- INTERSECT
- INTERSECT ALL
- FROM R, S → FROM R JOIN S
- INNER JOIN
  OUTER JOIN

分组聚集计算

- 聚集函数
- GROUP BY
- GROUP BY..HAVING..

战德臣 教授

SQL-SELECT的完整语法

Subquery ::==

SELECT [ALL | DISTINCT] { * | expr [[AS] c_alias] {, … } }
FROM tableref {, … }
[WHERE search_condition]
[GROUP BY column {, … } [HAVING search_condition]]
| subquery [UNION [ALL] | INTERSECT [ALL] | EXCEPT [ALL]]
[CORRESPONDING [BY] (colname {, … })] subquery;

Tableref ::== tablename [corr_name]

Select statement ::= =

Subquery [ORDER BY result_column [ASC | DESC] { , … }]

战德臣 教授

面向对象/对象关系数据库的查询SQL→OQL

- 基本的SQL：另一SELECT-FROM-WHERE只能出现在WHERE子句；
- 新标准：引入对象概念，可以在能使用聚集(collection)的任何位置使用SELECT-FROM-WEHRE
  - FROM子句中使用
  - SELECT子句中使用
  - Where子句中使用
- 注意：本课程要求基本的SQL

```
select dept_name,
    (select count(*)
     from instructor
     where department.dept_name = instructor.dept_name)
     as num_instructors
from department;
```

```
select name, salary, avg_salary
from instructor I1, lateral (select avg(salary) as avg_salary
                             from instructor I2
                             where I2.dept_name = I1.dept_name);
```

```
select dept_name, avg_salary
from (select dept_name, avg (salary)
      from instructor
      group by dept_name)
      as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

SQL语言之视图及其应用

战德臣

哈尔滨工业大学　教授.博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

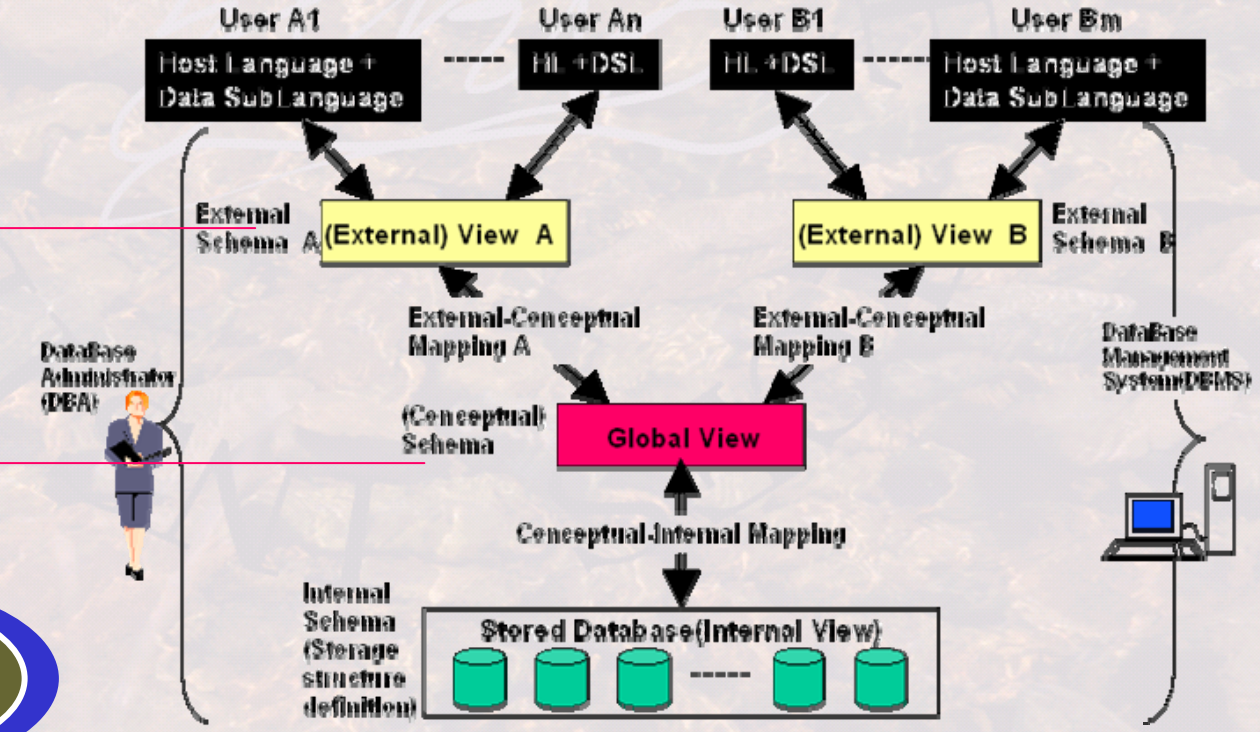
战德臣 教授

回顾：三级模式两层映像结构

- 对应概念模式的数据在SQL中被称为**基本表(Table)**，而对应外模式的数据称为**视图(View)**。**视图不仅包含外模式，而且包含其E-C映像。**

View

Base Table

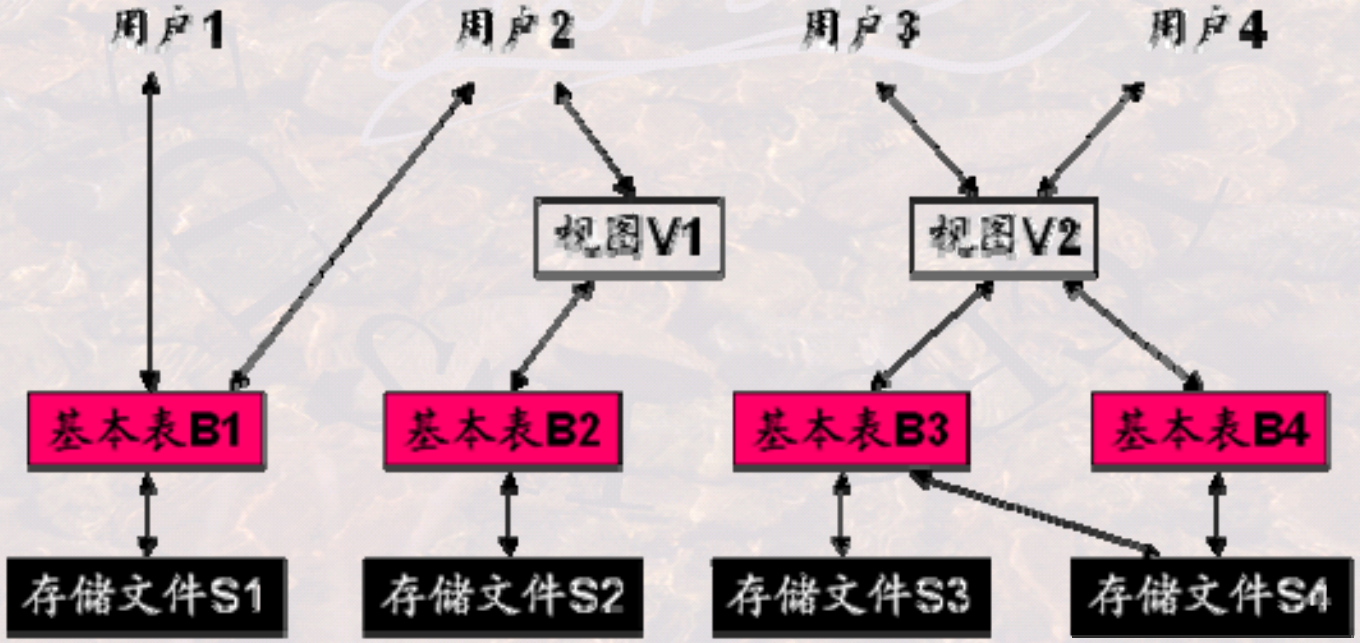


外模式应该怎样实现呢？

战德臣 教授

SQL数据库结构

- 基本表是实际存储于存储文件中的表，基本表中的数据是需要存储的
- 视图在SQL中只存储其由基本表导出视图所需要的公式，即由基本表产生视图的映像信息，其数据并不存储，而是在运行过程中动态产生与维护的
- 对视图数据的更改最终要反映在对基本表的更改上。



➢视图需要“先定义，再使用”

定义视图

create view **view_name [(列名[，列名] …)]**

as 子查询 **[*with check option*]**

➢如果视图的属性名缺省，则默认为子查询结果中的属性名；也可以显式指明其所拥有的列名。

➢with check option指明当对视图进行insert，update，delete时，要检查进行insert/update/delete的元组是否满足视图定义中子查询中定义的条件表达式

示例：定义一个视图 CompStud 为计算机系的学生，通过该视图可以将Student表中其他系的学生屏蔽掉

```
Create  View  CompStud  AS
        ( Select   *   From  Student
          Where  D#  in ( Select   D#  From  Dept
                           Where  Dname = '计算机' ) );
```

示例：定义一个视图Teach为教师任课的情况，把Teacher表中的个人隐私方面的信息，如工资等屏蔽掉，仅反映其教哪门课及其学分等。

```
Create  View  Teach  AS
        ( Select   T.Tname, C.Cname, Credit
            From  Teacher  T, Course  C
          Where  T.T# = C.T# ) ;
```

战德臣 教授

使用视图：定义好的视图，可以像Table一样，在SQL各种语句中使用

示例：检索主讲数据库课程的教师姓名，我们可使用Teach

```
Select  T.Tname  From  Teach  T
Where   T.Cname = '数据库' ;
```

示例：检索计算机系的所有学生，我们可使用CompStud

```
Select  *  From  CompStud;
```

示例：检索计算机系的年龄小于20的所有学生，我们可使用CompStud

```
Select  *  From  CompStud
Where   Sage < 20 ;
```

如何执行对视图的查询呢？

```
Create View Teach AS
        ( Select  T.Tname, C.Cname, Credit
          From Teacher T, Course C
          Where T.T# = C.T# ) ;
```

```
Create View CompStud AS
        ( Select * From Student
          Where D# in ( Select  D# From Dept
                        Where Dname = '计算机' ) ;
```

- 定义视图，有时可方便用户进行检索操作。

示例：定义视图StudStat, 描述学生的平均成绩、最高成绩，最低成绩等

```
Create View StudStat(S#, Sname, AvgS, MinS, MaxS, CNT)
as ( Select S#, Sname, AVG(Score), MIN(Score), Max(Score), Count(*)
        From Student S, SC Where S.S# = SC.S#
      Group by S.S# ) ;
```

示例：基于视图StudStat检索某一学生平均成绩

```
Select Sname, AvgS From StudStat Where Sname = '张三' ;
```

视图究竟有什么作用?

战德臣 教授

SQL视图更新：是比较复杂的问题，因视图不保存数据，对视图的更新最终要反映到对基本表的更新上，而有时，视图定义的映射不是可逆的。

示例：

```
create  view  S_G(S#, Savg )
            as  ( select   S#, AVG(Score)
                    from   SC  group  by   S# );
```

如要进行下述更新操作？

```
update   S_G
set   Savg  = 85
where  S# = '98030101' ;
```

能否由视图S_G的更新，而更新SC呢?

哪些类型的视图是不可更新的?

哪些类型的视图又是可更新的?

示例

```
create  view  ClassStud(Sname, Sclass)
        as  ( select   Sname, Sclass
              from   Student );
```

- 如要进行下述更新操作？

```
Insert  into   ClassStud
Values  ( '张三', '980301' );
```

- 能否由视图ClassStud的更新，而更新Student呢?
- 回答是：不能，因为缺少S#，而S#是Student的主键。

SQL视图更新的可执行性

- 如果视图的select目标列包含聚集函数，则不能更新
- 如果视图的select子句使用了unique或distinct，则不能更新
- 如果视图中包括了group by子句，则不能更新
- 如果视图中包括经算术表达式计算出来的列，则不能更新
- 如果视图是由单个表的列构成，但并没有包括主键，则不能更新

➢ 对于由单一Table子集构成的视图，即如果视图是从单个基本表使用选择、投影操作导出的，并且包含了基本表的主键，则可以更新

可更新SQL视图示例：

```
create  view  CStud(S#, Sname, Sclass)
        as  ( select   S#, Sname, Sclass
              from   Student  where  D# ='03');
```

- 上例是可以更新的

```
Insert  into   CStud
Values  ( "98030104", "张三丰", "980301" );
```

```
insert  into   CStud
values ("98030104", "张三丰", "980301" )
```

转换为

```
insert  into  Student
values ("98030104", "张三丰", Null, Null, "03" , "980301" )
```

- 已经定义的视图也可以撤消

撤消视图

```
Drop View view_name
```

示例：撤消视图Teach

```
Drop View Teach;
```

示例：撤消视图CompStud

```
Drop View CompStud;
```

战德臣 教授

- 不仅视图可以撤消，基本表、数据库等都可以撤消

撤消基本表

Drop Table 表名

示例：撤消学生表Student

```
Drop Table Student;
```

示例：撤消教师表Teacher

```
Drop Table Teacher;
```

回顾本讲学了什么?

战德臣

哈尔滨工业大学　教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

回顾本讲学习了什么?

战德臣 教授

SQL：结构化查询语言

子查询

- SELECT ..FROM..
  WHERE ..(子查询)
- [NOT] IN
- θ some/ θ all
- [NOT] EXISTS

关系代数的实现

- UNION
- UNION ALL
- EXCEPT
- EXCEPT ALL
- INTERSECT
- INTERSECT ALL
- FROM R, S → FROM R JOIN S
- INNER JOIN
  OUTER JOIN

分组聚集计算

- 聚集函数
- GROUP BY
- GROUP BY..HAVING..

视图

- CREATE VIEW
- 视图更新问题
- …FROM [view]…