

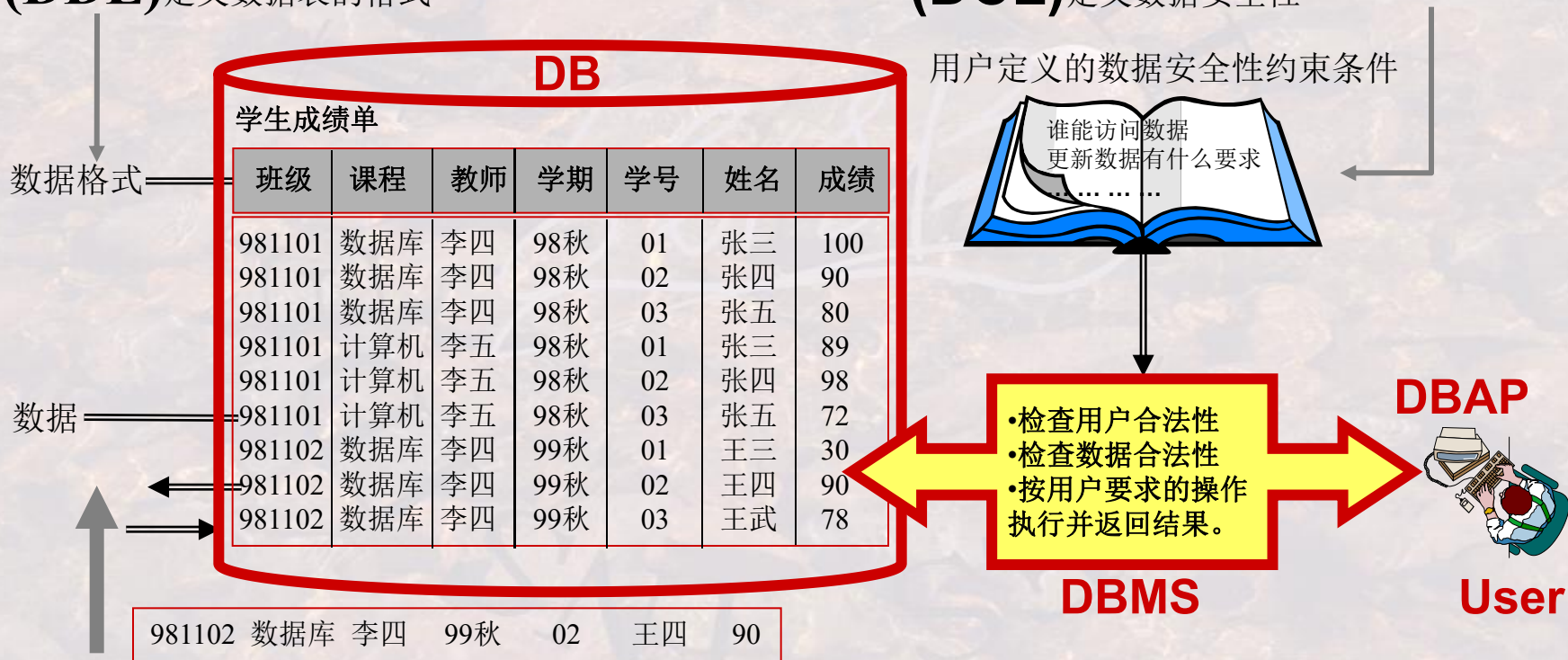
5.2.5 数据库语言--结构化数 据库语言SQL

由关系模型到结构化数据库语言SQL

(1)回顾：数据库系统的概念

阶段1: User/DBAP通过**数据定义语言 (DDL)**定义数据表的格式

User/DBA通过**数据控制语言 (DCL)**定义数据安全性



阶段2: User/DBAP通过**数据操纵语言 (DML)**操纵数据进出数据库

关系运算式

$\Pi_{\text{列名}, \dots, \text{列名}} (\sigma_{\text{检索条件}} (\text{表名1} \times \text{表名2} \times \dots))$

□数据库语言SQL

Select 列名 [[, 列名] ...]

From 表名1 [[, 表名2], ...]

[**Where** 检索条件];

语义：将**From**后面的所有表串接起来，检索出满足“检索条件”的元组，并按给定的列名及顺序进行投影显示。

SQL: Structural Query Language

□SQL语言是数据库系统的标准语言，它可以定义数据库、操纵数据库和进行数据库控制。

□SQL语言主要由以下9个单词引导的操作语句来构成，但每一种语句都能表达复杂的操作请求。

➤ DDL语句引导词：Create(建立), Alter(修改), Drop(撤消)

✓定义Database, Table, View, Index

➤ DML语句引导词：Insert(插入), Update(更新), Delete(删除), Select(查询)

✓各种方式的更新与检索操作

✓各种条件的查询操作，如连接查找，模糊查找，分组查找，嵌套查找等

✓各种聚集操作，求平均、求和、...等，分组聚集，分组过滤等

➤ DCL语句引导词：Grant, Revoke

✓安全性控制---授权和撤消授权

由关系模型到结构化数据库语言SQL

(4)用SQL语言创建数据库并定义表-简介

创建课程学习数据库：SCT

Create Database SCT;



Student					
S#	Sname	Ssex	Sage	D#	Sclass

Course				
C#	Cname	Chours	Credit	T#

➤定义学生表：Student

```
Create Table Student ( S# char(8) not null , Sname char(10),  
                        Ssex char(2), Sage integer, D# char(2), Sclass char(6) );
```

➤定义课程表：Course

```
Create Table Course ( C# char(3) , Cname char(12),  
                      Chours integer, Credit float(1), T# char(3) );
```

➤同学可自己定义其他的表：Dept, Teacher, SC

create table 表名(列名 数据类型 [**not null**] [, 列名 数据类型 , ...]);

由关系模型到结构化数据库语言SQL

(5)用SQL语言在所定义表中增加记录-简介

➤ 追加学生表中的元组

Insert Into Student

Values ('98030101', '张三', '男', 20, '03', '980301');

Insert Into Student (S#, Sname, Ssex, Sage, D# , Sclass)

Values ('98030102', '张四', '女', 20, '03', '980301');

➤追加课程表中的元组

Insert Into Course

/*所有列名省略，须与定义或存储的列名顺序一致

Values ('001', '数据库', 40, 6, '001');

/*如列名未省略，须与语句中列名的顺序一致

Insert Into Course(Cname, C#, Credit, Chours, T#);

Values ('数据库', '001', 6, 20, '001');

Student					
S#	Sname	Ssex	Sage	D#	Sclass
98030101	张三	男	20	03	980301
98030102	张四	女	20	03	980301
98030103	张五	男	19	03	980301
98040201	王三	男	20	04	980402
98040202	王四	男	21	04	980402
98040203	王五	女	19	04	980402

insert into 表名[(列名 [, 列名]...] values (值 [, 值] , ...);

由关系模型到结构化数据库语言SQL

(6)已经建立起的数据库

已经建立好的数据库---供后面学习和训练使用

Student					
S#	Sname	Ssex	Sage	D#	Sclass
98030101	张三	男	20	03	980301
98030102	张四	女	20	03	980301
98030103	张五	男	19	03	980301
98040201	王三	男	20	04	980402
98040202	王四	男	21	04	980402
98040203	王五	女	19	04	980402

SC		
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

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

Dept		
D#	Dname	Dean
01	机电	李三
02	能源	李四
03	计算机	李五
04	自动控制	李六

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

SQL-SELECT之简单使用

SQL-SELECT之简单使用

(1)基本检索操作的表达方法

Select 的简单语法形式为:

Select 列名 [[, 列名] ...]

From 表名

[**Where** 检索条件];

●语义：从<表名>所给出的表中，查询出满足<检索条件>的元组，并按给定的<列名>及顺序进行投影显示

$$\Pi_{\text{列名}, \dots, \text{列名}} (\sigma_{\text{检索条件}} (\text{表名}))$$

SQL-SELECT之简单使用

(2)基本检索条件书写

- 例如：检索学生表中所有学生的信息

```
Select  S#, Sname, Ssex, Sage, Sclass, D#  
From  Student ;
```

```
Select  *  From  Student ;
```

//如投影所有列，则可以用*来简写

- 再如：检索学生表中所有学生的姓名及年龄

```
Select  Sname, Sage  
From  Student ;
```

//投影出某些列

- 再如：检索学生表中所有年龄小于19岁的学生的年龄及姓名

```
Select  Sage, Sname  
From  Student  
Where  Sage <= 19;
```

//投影的列可以重新排定顺序

Student					
S#	Sname	Ssex	Sage	D#	Sclass
98030101	张三	男	20	03	980301
98030102	张四	女	20	03	980301
98030103	张五	男	19	03	980301
98040201	王三	男	20	04	980402
98040202	王四	男	21	04	980402
98040203	王五	女	19	04	980402

SELECT ... FROM ... WHERE ...

SQL-SELECT之简单使用

(2)基本检索条件书写

➤ 例如：求或者学过001号课程, 或者学过002号课程的学生的学号

```
Select S# From SC
Where C# = '001' OR C#='002';
```

➤ 再例如：求既学过001号课程, 又学过002号课程的学生的学号? 如下书写SQL语句会得到正确结果吗? 它能得到什么结果? 怎样正确书写?

```
Select S# From SC
Where C# = '001' AND C#='002';
```

//正确的SQL语句在讲义后面的示例中讲解

对于每一行应用
Where子句的条件

SC		
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

SELECT ... FROM ... WHERE ...

SQL-SELECT之简单使用

(3)如何去掉检索结果的重复元组或记录呢？

检索结果去重复：DISTINCT

➤例如：在选课表中，检索成绩大于80分的所有学号

```
Select S#
```

```
From SC
```

```
Where Score > 80 ; //有重复元组出现，比如一个同学两门以上课程大于80
```

```
Select DISTINCT S#
```

```
From SC
```

```
Where Score > 80; //重复元组被DISTINCT过滤掉，只保留一份
```

SC		
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

表(Table)和关系(Relation)在大部分情况下概念通用，但有细微差别：前者可允许有重复元组，而后者不允许

SQL-SELECT之简单使用

(4)如何对检索结果进行排序呢？

检索结果的排序

Select语句中结果排序是通过增加**order by**子句实现的

order by 列名 [**asc** | **desc**]

➤ 意义为结果按指定列名进行排序，若后跟**asc**或省略，则为升序；若后跟**desc**，则为降序。

➤ 例如：按学号由小到大的顺序显示出所有学生的学号及姓名

```
Select S#, Sname From Student  
Order By S# ASC ;
```

➤ 再如：检索002号课大于80分的所有同学学号并按成绩由高到低顺序显示

```
Select S# From SC Where C# = '002' and Score > 80  
Order By Score DESC ;
```

SELECT ... FROM ... WHERE ... ORDER BY ...

模糊查询

➤比如检索“姓张的学生”，检索“张某某”，这类查询问题，**Select**语句是通过在检索条件中引入运算符**like**来表示的

➤ 含有**like**运算符的表达式

列名 [**not**] **like** “字符串”

➤ 找出匹配给定字符串的字符串。其中给定字符串中可以出现%, _等匹配符.

➤ 匹配规则:

❑ “**%**” 匹配零个或多个字符

❑ “**_**” 匹配任意单个字符

❑ “****” 转义字符，用于去掉一些特殊字符的特定含义，使其被作为普通字符看待，如用 “**\%**”去匹配字符%，用**_** 去匹配字符_

SQL-SELECT之简单使用

(5)如何表达模糊性的查询内容呢？

- 例如：检索所有姓张的学生学号及姓名

```
Select  S#, Sname  From  Student  
Where  Sname Like  '张%';
```

- 再如：检索名字为张某某的所有同学姓名

```
Select  Sname  From  Student  
Where  Sname Like  '张__';
```

- 再如：检索名字不姓张的所有同学姓名

```
Select  Sname  From  Student  
Where  Sname Not Like  '张%';
```

已知如下关系：学生Student(S#, Sname)，课程Course(C#, Cname, T#)，选课SC(S#, C#, Score)，教师T(T#, Tname)。其中S#为学号，Sname为学生命名，C#为课号，Cname为课程名，T#为教师编号，Score为成绩，Tname为教师名。(1)阅读SQL语句

SELECT S# FROM SC WHERE C# = 'C01' OR C# = 'C02';

关于其查询结果，下列说法正确的是_____。

- ☐ A 既学过C01号课程，又学过C02号课程的所有学生的学号；
- ☒ B 或者学过C01号课程，或者学过C02号课程的所有学生的学号
- ☐ C 既未学过C01号课程，又未学过C02号课程的所有学生的学号
- ☐ D 或者未学过C01号课程，或者未学过C02号课程的所有学生的学号；

提交

SQL-SELECT之简单使用

(6)小结

- 例如：检索学生表中所有学生的信息

```
SELECT ... FROM ... WHERE ...
```

- 再如

```
SELECT DISTINCT ... FROM ... WHERE ...
```

- 再如

```
SELECT ... FROM ... WHERE ... ORDER BY ...
```

```
SELECT ... FROM ... WHERE ... LIKE ...
```

```
SELECT ... FROM ... WHERE ...
```

Student

S#	Sname	Ssex	Sage	D#	Sclass
98030101	张三	男	20	03	980301
98030102	张四	女	20	03	980301
98030103	张五	男	19	03	980301
98040201	王三	男	20	04	980402
98040202	王四	男	21	04	980402
98040203	王五	女	19	04	980402

SQL-SELECT之多表联合操作

SQL-SELECT之多表联合操作

(1)多表联合检索的表达方法

➤ **Select** 的多表联合检索语句如下：

Select 列名 [[, 列名] ...]

From 表名1, 表名2, ...

Where 检索条件；

➤ 相当于 $\Pi_{\text{列名}, \dots, \text{列名}} (\sigma_{\text{检索条件}} (\text{表名1} \times \text{表名2} \times \dots))$

➤ 检索条件中要包含连接条件，通过不同的连接条件可以实现各种连接操作。

SQL-SELECT之多表联合操作

(2)多表联合检索之连接条件书写

➤例如：按“001”号课成绩由高到低的顺序显示出所有学生的姓名(二表连接)

```
Select  Sname From Student, SC
Where  Student.S# = SC.S# and SC.C# = '001'
Order By Score DESC;
```

➤当多表连接时，如果两个表的属性名相同，则需采用表名.属性名方式来限定该属性是属于哪一个表

➤再如：按‘数据库’课程成绩由高到低顺序显示所有同学姓名(三表连接)

```
Select  Sname From Student, SC, Course
Where  Student.S# = SC.S# and SC.C# = Course.C#
      and Cname = '数据库'
Order By Score DESC;
```

```
Student(S#,Sname,Ssex,Sage,D#,Sclass)
Course(C#,Cname,Chours,Credit,T#)
SC(S#,C#,Score)
Dept(D#,Dname,Dean)
Teacher(T#,Tname,D#,Salary)
```


SQL-SELECT之多表联合操作

(3)多表联合检索之表与列的别名

➤ 连接运算涉及到重名的问题，如两个表中的属性重名，连接的两个表重名(同一表的连接)等，因此需要使用**别名**以便区分

➤ **select**中采用别名的方式：

Select 列名 **as** 列别名 [[, 列名 **as** 列别名] ...]

From 表名1 **as** 表别名1, 表名2 **as** 表别名2, ...

Where 检索条件；

➤ 上述定义中的**as** 可以省略

➤ 当定义了别名后，在检索条件中可以使用别名来限定属性

SQL-SELECT之多表联合操作

(3)多表联合检索之表与列的别名

➤例如：求有薪水差额的任意两位教师

```
Select  T1.Tname as Teacher1, T2.Tname as Teacher2  
From    Teacher T1, Teacher T2  
Where   T1.Salary > T2.Salary ;
```

➤求年龄有差异的任意两位同学的姓名

```
Select  S1.Sname as Stud1, S2.Sname as Stud2  
From    Student S1, Student S2  
Where   S1.Sage > S2.Sage ;
```

➤请同学书写一下：求 ‘001’号课程有成绩差的任意两位同学

➤有时表名很长时，为书写条件简便，也定义表别名，以简化书写。

```
Student(S#,Sname,Ssex,Sage,D#,Sclass)  
Course(C#,Cname,Chours,Credit,T#)  
SC(S#,C#,Score)  
Dept(D#,Dname,Dean)  
Teacher(T#,Tname,D#,Salary)
```

SQL-SELECT之多表联合操作

(4)多表联合检索之表与自身的连接

➤再如：求既学过“001”号课又学过“002”号课的所有学生的学号(二表连接)
Select SC1.S# From SC SC1, SC SC2

Where SC1.S# = SC2.S# and SC1.C#='C01'
and SC2.C#='C02' ;

➤再如：求“C01”号课成绩比“C02”号课成绩高的所有学生的学号(二表连接)
Select SC1.S# From SC SC1, SC SC2

Where SC1.S# = SC2.S# and SC1.C#='C01'
and SC2.C#='C02' and SC1.Score > SC2.Score;

SC: SC1			SC: SC2		
SC1.S#	SC1.C#	SC1.Score	SC2.S#	SC2.C#	SC2.Score
S01	C01	80	S01	C02	90
S01	C01	80	S01	C03	85
S01	C02	90	S01	C01	80
S01	C02	90	S01	C03	85
S01	C03	85	S01	C01	80
S01	C03	85	S01	C02	90

SQL-SELECT之多表联合操作

(5)多表联合检索之语义之理解

➤ 正确理解汉语表达的查询语义，正确表达为**SQL**语句

➤ **例如：**列出没学过李明老师讲授课程的所有同学的姓名？如下书写**SQL**语句会得到正确结果吗？它能得到什么结果？怎样正确书写？

```
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#;
```

//正确的SQL语句在讲义后面的示例中讲解

SQL-SELECT之多表联合操作

(6)多表联合检索之嵌套子查询

IN子查询。其基本语法为：

表达式 [**not**] **in** (子查询)

➤ 语义：判断某一表达式的值是否在子查询的结果中。

➤再例如：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'  );
```

➤再例如：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' );
```

SQL-SELECT之多表联合操作

(6)多表联合检索之嵌套子查询

再例如：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# );
```


SQL-SELECT之多表联合操作

(7)非相关子查询 vs. 相关子查询

非相关子查询

外层查询

Select Sname

From Student

Where S# not in (

内层查询

Select S#

From SC, Course C, Teacher T

Where T.Tname = '李明' and SC.C# = C.C#
and T.T# = C.T#);

➤ 内层查询独立进行，没有涉及任何外层查询相关信息的子查询被称为非相关子查询。

相关子查询

- 有时，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，这样的子查询称为相关子查询。
- 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
- 例如：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。

现有关系数据库如下： 学生（学号，姓名，性别，专业、奖学金）， 课程（课程号，课程名，学分）， 选课（学号，课程号，分数）， 用SQL语言实现下列小题：

(1)检索没有获得奖学金、同时至少有一门课程成绩在95分以上的学生信息，包括学号、姓名和专业，正确的SQL语句是_____。

A

SELECT 学号, 姓名, 专业 **FROM** 学生, 课程, 选课
WHERE 学生.学号=选课.学号 **AND** 课程.课程号=选课.课程号 **AND** 奖学金<=0 **AND** 分数>95;

B

SELECT 学号, 姓名, 专业 **FROM** 学生, 选课
WHERE 学生.学号=选课.学号 课程.课程号=选课.课程号 奖学金<=0
分数>95;

C

SELECT 学号, 姓名, 专业 **FROM** 学生, 课程, 选课
WHERE 学生.学号=选课.学号 **OR** 课程.课程号=选课.课程号 **AND** 奖学金<=0 **AND** 分数>95;

提交

SQL-SELECT之多表联合操作

(6)小结

➤再如：求既学过“001”号课又学过“002”号课的所有学生的学号(二表连接)

```
Select SC1.S# From SC SC1, SC SC2  
Where SC1.S# = SC2.S# and SC1.C#='C01'
```

SELECT ... FROM 表名1,表名2,... WHERE ...

➤再如：求既学过“001”号课又学过“002”号课且成绩大于80分的所有学生的学号(三表连接)

SELECT ... FROM ... WHERE ... IN (SELECT ... FROM ... WHERE ...)

```
Select SC1.S# From SC SC1, SC SC2  
Where SC1.S# = SC2.S# and SC1.C#='C01' and SC2.C#='C02' and SC1.Score > 80
```

SC1.S#	SC1.C#	SC1.Score	SC2.S#	SC2.C#	SC2.Score
S01	C01	80	S01	C02	90
S01	C01	80	S01	C03	85
S01	C02	90	S01	C01	80
S01	C02	90	S01	C03	85
S01	C03	85	S01	C01	80
S01	C03	85	S01	C02	90

SQL-SELECT之分组聚集操作

SQL-SELECT之分组聚集操作

(1)SELECT之结果计算与聚集函数

结果计算与聚集函数

➤ **select**子句可以是一些计算表达式或聚集函数，表明在选择和投影的同时直接进行一些运算，如下所示：

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] ;

➤ 计算表达式可以是常量、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。

➤ 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;

SQL-SELECT之分组聚集操作

(1)SELECT之结果计算与聚集函数

结果计算与聚集函数

➤ **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

SQL-SELECT之分组聚集操作

(1)SELECT之结果计算与聚集函数

结果计算与聚集函数

- 求教师的工资总额

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

- 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#;
```

- 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#;
```

分组计算与聚集

- 为解决同时求解若干个集合的聚集运算问题，引出了分组的概念。
- **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，这一过程就是分组过程。
- 分组可以在基本**Select**语句基础上引入分组子句来完成：

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, ...

SQL-SELECT之分组聚集操作

(2)SELECT之分组计算与聚集

分组计算与聚集

➤ 例如： 求每一个学生的平均成绩

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

➤ 再如： 求每一门课程的平均成绩

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

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

Group by S#

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

Group by C#

SQL-SELECT之分组聚集操作

(3)SELECT之分组过滤

分组过滤----过滤掉分组，而不是元组

➤求不及格课程超过两门的同学的学号，下述写法正确吗？

```
Select S# From SC
```

```
Where Score < 60 and Count(*)>2
```

```
Group by S#;
```

➤若要对分组(集合)进行条件过滤，可使用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 ] ] ;
```

SQL-SELECT之分组聚集操作

(3)SELECT之分组过滤

分组过滤----过滤掉分组，而不是元组

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

SQL-SELECT之分组聚集操作

(3)SELECT之分组过滤

分组过滤----过滤掉分组，而不是元组

- 例如 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;
```

- 再如 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;
```

SQL-SELECT之分组聚集操作

(3)SELECT之分组过滤

分组过滤----过滤掉分组，而不是元组

➤例如：求有两门以上不及格课程的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;
```

➤上述写法正确吗？（课程范围）

➤正确的如下书写，为什么呢？

```
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# ;
```

SQL-SELECT之分组聚集操作

(4)SQL-SELECT之总结

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] { , ... }]
```


5.3.4 数据也是生产力--数据挖掘

由数据库到数据挖掘I-数据挖掘示例之背景与概念

(1)数据也是生产力

数据对超市经营有无帮助呢？



商品购买明细

交易号 T1000 , 日期 04/05/2013 , 时间 10:18 , 收款员 E02
顾客 C01 , 支付方式 MasterCard , 总金额 ¥1400.00

商品号	商品名	数量	单价	金额
200008	汇源果汁	5	200.00	1000.00
200020	哈啤90	1	300.00	300.00
200035	555香烟	1	100.00	100.00

收款员 E02
00.00
金额
1000.00
300.00
100.00

客户购买习惯
商品组合方式及策略
...

营销策略
价格策略
货源组织



由数据库到数据挖掘I-数据挖掘示例之背景与概念

(2)数据运用的前提—数据的聚集与管理

超市数据库



商品购买明细				
交易号 T1000 , 日期 04/05/2013 , 时间 10:18 , 收款员 E02				
顾客 C01 , 支付方式 MasterCard , 总金额 ¥ 1400.00				
商品号	商品名	数量	单价	金额
200008	汇源果汁	5	200.00	1000.00
200020	哈啤90	1	300.00	300.00
200035	555香烟	1	100.00	100.00

商品购买单						
交易号	日期	时间	收款员号	顾客号	支付方式	总金额
T1000	04/05/2013	10:18	E02	C01	MasterCard	1400.00
T1001	04/05/2013	11:10	E01	C03	Visa	1200.00
...
T1101	04/05/2013	15:31	E03	C02	MasterCard	1000.00

商品购买单明细					
交易号	商品号	商品名	数量	单价	金额
T1000	200008	汇源果汁	5	200.00	1000.00
T1000	200020	哈啤90	1	300.00	300.00
T1000	200035	555香烟	1	100.00	100.00
T11001	200020	哈啤90	2	300.00	600.00
T11001	200009	巧克力	2	300.00	600.00
...
T1101	200008	汇源果汁	1	200.00	200.00
T1101	200020	哈啤90	1	300.00	300.00
...

(3)什么是数据挖掘？

数据挖掘，又称为数据库中知识发现，它是一个从大量数据中抽取挖掘出未知的、有价值的模式或规律等知识的复杂过程。简单地讲就是从大量数据中挖掘或抽取出知识。

- 概要归纳
- 关联分析
- 分类与预测
- 聚类分析
- 异类分析
- 演化分析

由数据库到数据挖掘I-数据挖掘示例之背景与概念

(4)怎样挖掘数据--一个例子？

数据挖掘之关联规则挖掘

商品购买明细

交易号 T1000 , 日期 04/05/2013 , 时间 10:18 , 收款员 E02

顾客 C01 , 支付方式 MasterCard , 总金额 ¥1400.00

商品号	商品名	数量	单价	金额
08	汇源果汁	5	200.00	1000.00
20	哈啤90	1	300.00	300.00
200035	555香烟	1	100.00	100.00

商品的关联规则

“由尿布的购买，能够推断出啤酒的购买”

“尿布” \Rightarrow “啤酒” [支持度=2%，置信度
-60%]

支持度2%意味着所分析事务的2%同时购买尿布和啤酒
置信度60%意味着购买尿布的顾客60%也购买啤酒。

是否相信这条规则呢？—让数据说话

(5)概念准备

关联规则挖掘相关的基本概念

1. 项、项集与事务

设 $P = \{p_1, p_2, \dots, p_m\}$ 是所有项 (Item) 的集合。 D 是数据库中所有事务的集合，其中每个事务 T (Transaction) 是项的集合，是 P 的子集，即 $T \subseteq P$ ；每一个事务有一个关键字属性，称作交易号或事务号以区分数据库中的每一个事务。设 A 是一个项集 (ItemSet)，事务 T 包含 A 当且仅当 $A \subseteq T$ 。

2. 关联规则

关联规则是形如 $A \Rightarrow B$ 的蕴涵式，即命题 A (如“项集 A 的购买”) 蕴涵着命题 B (如“项集 B 的购买”)，或者说由命题 A 能够推出命题 B ，其中 $A \subseteq P$ ， $B \subseteq P$ ，并且 $A \cap B = \emptyset$ 。

商品购买明细

交易号 T1000，日期 04/05/2013，时间 10:18，收款员 E02
顾客 C01，支付方式 MasterCard，总金额 ¥1400.00

商品号	商品名	数量	单价	金额
200008	汇源果汁	5	200.00	1000.00
200020	哈啤90	1	300.00	300.00
200035	555香烟	1	100.00	100.00

(5)概念准备

关联规则挖掘相关的基本概念

3. 支持度与置信度

Support ($A \Rightarrow B$) = $P(A \cup B)$ = 包含A和B的事务数 ÷ D中事务总数。

confidence ($A \Rightarrow B$) = $P(B|A)$ = 包含A和B的事务数 ÷ 包含A的事务数。

支持度反映一条规则的实用性，置信度反映规则的“值得信赖性”的程度

4. 强规则

同时满足最小支持度阈值(min_s)和最小置信度阈值(min_c)的规则称作强规则。

5. k-项集与k-频繁项集

项的集合称为**项集**，包含k个项的项集称为**k-项集**。

项集的出现频率是包含项集的事务数，简称为项集的**频率**、**支持计数**或**计数**。如果项集的出现频率大于或等于 min_s 与D中事务总数的乘积，则项集满足最小支持度 min_s 。如果项集满足最小支持度，则称它为**频繁项集**。频繁k-项集的集合通常记作 L_k 。

{面包, 果酱} --- 2-项集

{面包, 果酱, 奶油} --- 3-项集

关联规则挖掘的基本思想

找出所有频繁项集。依定义，这些项集出现的频率至少和预定义的最小出现频率一样。

如何挖掘频繁项集？

Apriori 算法



由频繁项集产生强关联规则。依定义，这些规则必须满足最小支持度和最小置信度。

数据挖掘之关联规则挖掘

{面包, 果酱} --- 2-项集
{面包, 果酱, 奶油} --- 3-项集

商品的关联规则

商品购买明细				
交易号 <u>T1000</u> , 日期 <u>04/05/2013</u> , 时间 <u>10:18</u> , 收款员 <u>E02</u> 顾客 <u>C01</u> , 支付方式 <u>MasterCard</u> , 总金额 <u>¥ 1400.00</u>				
商品号	商品名	数量	单价	金额
08	汇源果汁	5	200.00	1000.00
20	哈啤90	1		300.00
00035				100.00



支持度2%意味着所分析事务的2%同时购买尿布和啤酒
置信度60%意味着购买尿布的顾客60%也购买啤酒。

是否相信这条规则呢?—让数据说话

-数据挖掘示例之计算过程-1

关联规则挖掘的基本思想



找出所有频繁项集。依定义，这些项集出现的频率至少和预定义的最小出现频率一样。

如何挖掘频繁项集？

Apriori 算法

由频繁项集产生强关联规则。依定义，这些规则必须满足最小支持度和最小置信度。

由数据库到数据挖掘II-数据挖掘示例之计算过程-1

(1)频繁项集发现的计算过程

频繁项集挖掘算法计算示例

1.对问题域数据进行抽象

商品购买明细数据库

商品购买明细				
交易号 <u>T1000</u> , 日期 <u>04/05/2013</u> , 时间 <u>10:18</u> , 收款员 <u>E02</u> 顾客 <u>C01</u> , 支付方式 <u>MasterCard</u> , 总金额 <u>¥1400.00</u>				
商品号	商品名	数量	单价	金额
200008	汇源果汁	5	200.00	1000.00
200020	哈啤90	1	300.00	300.00
200035	555香烟	1	100.00	100.00

交易号	一次交易中购买的商品列表		交易号	一次交易中购买的商品列表
T0000	P1, P2, P3, P5		T0050	P1, P3, P5
T1000	P1, P2, P6, P8		T1500	P2, P4, P8
T2000	P2, P3, P7, P8		T2500	P1, P3, P5
T3000	P1, P2, P6		T3500	P2, P3, P7
T4000	P1, P2, P3, P5, P6, P7		T4500	P1, P2, P6, P8
T5000	P1, P3, P5, P6		T5500	P1, P2, P5, P6
T6000	P2, P3, P6		T6500	P1, P2, P5, P6
T7000	P1, P4, P6		T7500	P1, P2, P4, P6
T8000	P2, P3, P4, P5		T8500	P1, P2, P4, P5, P6
T9000	P3, P4, P5		T9500	P1, P2, P4, P5, P6
总交易次数: 20				

(1)频繁项集发现的计算过程

频繁项集挖掘算法计算示例

2.形成候选1-项集，并求出频繁1-项集

候选1项集.

项集	支持度计数
{ P1 }	14
{ P2 }	15
{ P3 }	10
{ P4 }	7
{ P5 }	11
{ P6 }	12
{ P7 }	3
{ P8 }	3

频繁1项集. 支持度计数 \geq 最小支持度计数5
($\text{min_sup}=5/20=25\%$)



项集	支持度计数
{ P1 }	14
{ P2 }	15
{ P3 }	10
{ P4 }	7
{ P5 }	11
{ P6 }	12

由数据库到数据挖掘II-数据挖掘示例之计算过程-1

(1)频繁项集发现的计算过程

频繁项集挖掘算法计算示例

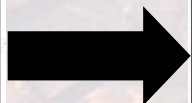
3.形成候选2-项集，并求出频繁2-项集

候选2项集.

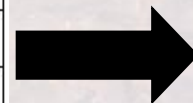
$$C_2 = L_1 \text{ Join } L_1$$

频繁1项集

项集	支持度计数
{ P1 }	14
{ P2 }	15
{ P3 }	10
{ P4 }	7
{ P5 }	11
{ P6 }	12



项集	支持度计数
{ P1, P2 }	10
{ P1, P3 }	5
{ P1, P4 }	4
{ P1, P5 }	9
{ P1, P6 }	11
{ P2, P3 }	6
{ P2, P4 }	5
{ P2, P5 }	7
{ P2, P6 }	10
{ P3, P4 }	2
{ P3, P5 }	7
{ P3, P6 }	3
{ P4, P5 }	4
{ P4, P6 }	3
{ P5, P6 }	6



频繁2项集. 支持度计数 \geq 最小支持度计数5

项集	支持度计数
{ P1, P2 }	10
{ P1, P3 }	5
{ P1, P5 }	9
{ P1, P6 }	11
{ P2, P3 }	6
{ P2, P4 }	5
{ P2, P5 }	7
{ P2, P6 }	10
{ P3, P5 }	7
{ P5, P6 }	6

由数据库到数据挖掘II-数据挖掘示例之计算过程-1

(1)频繁项集发现的计算过程

频繁项集挖掘算法计算示例

4.形成候选3-项集，并剪枝，进一步求出频繁3-项集

候选3项集.
 $C_3=L_2 \text{ Join } L_2$

频繁2项集

项集	支持度计数
{ P1, P2 }	10
{ P1, P3 }	5
{ P1, P5 }	9
{ P1, P6 }	11
{ P2, P3 }	6
{ P2, P4 }	5
{ P2, P5 }	7
{ P2, P6 }	10
{ P3, P5 }	7
{ P5, P6 }	6

项集	
{ P1, P2, P3 }	
{ P1, P2, P5 }	
{ P1, P2, P6 }	
{ P1, P3, P5 }	
{ P1, P3, P6 }	被剪掉,因{P3,P6}
{ P1, P5, P6 }	
{ P2, P3, P4 }	被剪掉,因{P3,P4}
{ P2, P3, P5 }	
{ P2, P3, P6 }	被剪掉,因{P3,P6}
{ P2, P4, P5 }	被剪掉,因{P4,P5}
{ P2, P4, P6 }	被剪掉,因{P4,P6}
{ P2, P5, P6 }	
{ P3, P5, P6 }	被剪掉,因{P3,P6}

频繁3项集

候选3项集的支持度计数

项集	支持度计数
{ P1, P2, P3 }	2
{ P1, P2, P5 }	6
{ P1, P2, P6 }	8
{ P1, P3, P5 }	4
{ P1, P5, P6 }	6
{ P2, P3, P5 }	3
{ P2, P5, P6 }	5

项集	支持度计数
{ P1, P2, P5 }	6
{ P1, P2, P6 }	8
{ P1, P5, P6 }	6
{ P2, P5, P6 }	5

由数据库到数据挖掘II-数据挖掘示例之计算过程-1

(1)频繁项集发现的计算过程

频繁项集挖掘算法计算示例

5.迭代地求出最终结果-频繁项集

频繁3项集

项集	支持度计数
{ P1, P2, P5 }	6
{ P1, P2, P6 }	8
{ P1, P5, P6 }	6
{ P2, P5, P6 }	5



候选4项集---频繁4项集支持度计数 ≥ 5

项集	支持度计数
{ P1, P2, P5, P6 }	5

频繁项集全集 = 频繁1项集 \cup 频繁2项集 \cup 频繁3项集 \cup 频繁4项集

项集	支持度计数
{ P1 }	14
{ P2 }	15
{ P3 }	10
{ P4 }	7
{ P5 }	11
{ P6 }	12
{ P1, P2 }	10
{ P1, P3 }	5
{ P1, P5 }	9
{ P1, P6 }	11
{ P2, P3 }	6
{ P2, P4 }	5
{ P2, P5 }	7
{ P2, P6 }	10
{ P3, P5 }	7
{ P5, P6 }	5
{ P1, P2, P5 }	6
{ P1, P2, P6 }	8
{ P1, P5, P6 }	6
{ P2, P5, P6 }	5
{ P1, P2, P5, P6 }	5

由数据库到数据挖掘II-数据挖掘示例之计算过程-1

(2)频繁项集的发现算法

发现频繁项集的Apriori 算法

输入：事务数据库D；最小支持度计数阈值 \min_s 。

输出：D 中的频繁项集L。

算法：

```
1)  $L_1 = \text{find\_frequent\_1\_itemsets}(D)$ ; //算法始自频繁一项集的产生结果 $L_1$ 
2) for ( $k = 2; L_{k-1} \neq \emptyset; k++$ ) { //自频繁2项集的获取开始迭代( $k=2$ ),直至频繁 $k-1$ 项集为空时停止迭代.
3)    $C_k = \text{apriori\_gen}(L_{k-1}, \min\_s)$ ; //由频繁 $k-1$ 项集 $L_{k-1}$ 产生候选 $k$ 项集 $C_k$ . 由下面的一个子过程实现;
4)   for each transaction  $t \in D$  { //对候选项集进行支持度计数, 即对D中每一个事务进行处理;
5)      $C_t = \text{subset}(C_k, t)$ ; //从 $t$ 的项集中找出是 $C_k$ 中元素的候选子集 $C_t$ 
6)     for each candidate  $c \in C_t$  { //是 $C_t$ 中元素的项集, 则其支持度计数加1
7)        $c.\text{count}++$ ; //支持度计数
8)     }
9)   }
10)   $L_k = \{c \in C_k \mid c.\text{count} \geq \min\_s\}$  //形成频繁 $k$ -项集
11) }
12) return  $L = \bigcup_k L_k$ ;
```

```
procedure apriori_gen( $L_{k-1}$ : frequent (k-1)-itemset) //由频繁 $k-1$ 项集 $L_{k-1}$ , 产生候选 $k$ 项集的子过程
1) for each itemset  $l_1 \in L_{k-1}$  {
2)   for each itemset  $l_2 \in L_{k-1}$  { //对 $L_{k-1}$ 中每一个项集与其中的另一个项集进行组合
3)     if ( $(l_1[1]=l_2[1]) \wedge \dots \wedge (l_1[k-2]=l_2[k-2]) \wedge (l_1[k-1] < l_2[k-2])$ ) then {
4)        $c = l_1 \text{ (Join) } l_2$ ; //如果可连接, 则连接起来, 生产候选 $k$ -项集 $C_k$ 
5)       if has_infrequent_subset( $c, L_{k-1}$ ) then
6)         delete  $c$ ; // 剪枝, 如果 $c$ 包含有不是频繁 $k-1$ 项集的子集, 则删除 $c$ 
7)       else add  $c$  to  $C_k$ ;
8)     }
9) }
10) return  $C_k$ ;

procedure has_infrequent_subset( $c$ : candidate  $k$ -itemset;  $L_{k-1}$ : frequent (k-1)-itemset)
// 判断一个候选 $k$ -项集是否包含有不是频繁 $k-1$ 项集的项集
1) for each (k-1)-subsets of  $c$ 
2)   if  $s \notin L_{k-1}$  then
3)     return TRUE;
4) return FALSE;
```

-数据挖掘示例之计算过程-2

由数据库到数据挖掘II-数据挖掘示例之计算过程-2

(0)回顾：频繁项集的挖掘

关联规则挖掘的基本思想



找出所有频繁项集。依定义，这些项集出现的频率至少和预定义的最小出现频率一样。



由频繁项集产生强关联规则。依定义，这些规则必须满足最小支持度和最小置信度。

项集	支持度计数
{ P1 }	14
{ P2 }	15
{ P3 }	10
{ P4 }	7
{ P5 }	11
{ P6 }	12
{ P1, P2 }	10
{ P1, P3 }	5
{ P1, P5 }	9
{ P1, P6 }	11
{ P2, P3 }	6
{ P2, P4 }	5
{ P2, P5 }	7
{ P2, P6 }	10
{ P3, P5 }	7
{ P5, P6 }	5
{ P1, P2, P5 }	6
{ P1, P2, P6 }	8
{ P1, P5, P6 }	6
{ P2, P5, P6 }	5
{ P1, P2, P5, P6 }	5

由数据库到数据挖掘II-数据挖掘示例之计算过程-2

(1)什么是关联规则？

商品的关联规则

商品购买明细

交易号 T1000 , 日期 04/05/2013 , 时间 10:18 , 收款员 E02
顾客 C01 , 支付方式 MasterCard , 总金额 ¥1400.00

商品号	商品名	数量	单价	金额
08	汇源果汁	5	200.00	1000.00
20	哈啤90	1	300.00	300.00
200035	555香烟	1	100.00	100.00

“由尿布的购买，能够推断出啤酒的购买”

“尿布” \Rightarrow “啤酒” [支持度=2%，置信度
=60%]

支持度2%意味着所分析事务的2%同时购买尿布和啤酒
置信度60%意味着购买尿布的顾客60%也购买啤酒。

(2)关联规则的产生

关联规则的生成计算示例



{P1,P2,P5,P6}可以产生的潜在规则 $A \Rightarrow B$,
其中 $A \cup B = \{P1,P2,P5,P6\}$, $A \cap B = \emptyset$.

项集 A	项集 A 支持度 计数(支持度)	项集 B	项集 A∪B 的支持 度计数(支持度)	置信度 = 项集(A∪B)的支 持度÷项集 A 的支持度
{ P1, P2, P5 }	6 (30%)	{ P6 }	5 (25%)	5/6=83.33%
{ P1, P2, P6 }	8 (40%)	{ P5 }	5 (25%)	5/8=62.50%
{ P2, P5, P6 }	5 (25%)	{ P1 }	5 (25%)	5/5=100.00%
{ P1, P5, P6 }	6 (30%)	{ P2 }	5 (25%)	5/6=83.33%
{ P1, P2 }	10 (50%)	{ P5, P6 }	5 (25%)	5/10=50.00%
{ P1, P5 }	9 (45%)	{ P2, P6 }	5 (25%)	5/9=55.55%
{ P1, P6 }	11 (55%)	{ P2, P5 }	5 (25%)	5/11=45.45%
{ P2, P5 }	7 (35%)	{ P1, P6 }	5 (25%)	5/7=71.42
{ P2, P6 }	10 (50%)	{ P1, P5 }	5 (25%)	5/10=50.00%
{ P5, P6 }	6 (30%)	{ P1, P2 }	5 (25%)	5/6=83.33%
{ P1 }	14 (70%)	{ P2, P5, P6 }	5 (25%)	5/14=35.71%
{ P2 }	15 (75%)	{ P1, P5, P6 }	5 (25%)	5/15=33.33%
{ P5 }	11 (55%)	{ P1, P2, P6 }	5 (25%)	5/11=45.45%
{ P6 }	12 (60%)	{ P1, P2, P5 }	5 (25%)	5/12=41.66

项集	支持度计数
{ P1 }	14
{ P2 }	15
{ P3 }	10
{ P4 }	7
{ P5 }	11
{ P6 }	12
{ P1, P2 }	10
{ P1, P3 }	5
{ P1, P5 }	9
{ P1, P6 }	11
{ P2, P3 }	6
{ P2, P4 }	5
{ P2, P5 }	7
{ P2, P6 }	10
{ P3, P5 }	7
{ P5, P6 }	5
{ P1, P2, P5 }	6
{ P1, P2, P6 }	8
{ P1, P5, P6 }	6
{ P2, P5, P6 }	5
{ P1, P2, P5, P6 }	5



(2)关联规则的产生

关联规则的生成计算示例

输出的规则表， $A \cap B = \emptyset$ ，“购买A能够推出购买B”。置信度 $\geq 70\%$ 的规则。

项集 A	项集 A 支持度 计数(支持度)	项集 B	项集 $A \cup B$ 的支持 度计数(支持度)	置信度 = 项集($A \cup B$)的支 持度 ÷ 项集 A 的支持度
{ P1, P2, P5 }	6 (30%)	{ P6 }	5 (25%)	$5/6=83.33\%$
{ P2, P5, P6 }	5 (25%)	{ P1 }	5 (25%)	$5/5=100.00\%$
{ P1, P5, P6 }	6 (30%)	{ P2 }	5 (25%)	$5/6=83.33\%$
{ P2, P5 }	7 (35%)	{ P1, P6 }	5 (25%)	$5/7=71.42$
{ P5, P6 }	6 (30%)	{ P1, P2 }	5 (25%)	$5/6=83.33\%$

由数据库到数据挖掘II-数据挖掘示例之计算过程-2

(2)关联规则的产生

关联规则的生成计算示例

组合形成规则表，
频繁3项集能推出
哪些频繁项集？
置信度标记红色为
置信度 $\geq 70\%$ 的
规则。支持度标记
蓝色的为满足置信
度前提下的支持度
 $\geq 50\%$ 的规则

项集 A	项集 A 支持度 计数(支持度)	项集 B	项集 A \cup B 的支持 度计数(支持度)	置信度=项集(A \cup B)的支 持度÷项集 A 的支持度
{P1, P2, P5}	6 (30%)	{P6}	5 (25%)	5/6=83.33%
{P1, P2, P6}	8 (40%)	{P5}	5 (25%)	5/8=62.50%
{P2, P5, P6}	5 (25%)	{P1}	5 (25%)	5/5=100.00%
{P1, P5, P6}	6 (30%)	{P2}	5 (25%)	5/6=83.33%
{P1, P2}	10 (50%)	{P5, P6}	5 (25%)	5/10=50.00%
{P1, P2}	10 (50%)	{P5}	6 (30%)	6/10=60.00%
{P1, P2}	10 (50%)	{P6}	8 (40%)	8/10=80.00%
{P1, P5}	9 (45%)	{P2, P6}	5 (25%)	5/9=55.55%
{P1, P5}	9 (45%)	{P6}	6 (30%)	6/9=66.66%
{P1, P5}	9 (45%)	{P2}	6 (30%)	6/9=66.66%
{P1, P6}	11 (55%)	{P2, P5}	5 (25%)	5/11=45.45%
{P1, P6}	11 (55%)	{P2}	8 (40%)	8/11=72.72%
{P1, P6}	11 (55%)	{P5}	6 (30%)	6/11=54.54%
{P2, P5}	7 (35%)	{P1, P6}	5 (25%)	5/7=71.42
{P2, P5}	7 (35%)	{P1}	6 (30%)	6/7=85.71%
{P2, P5}	7 (35%)	{P6}	5 (25%)	5/7=71.42%
{P2, P6}	10 (50%)	{P1, P5}	5 (25%)	5/10=50.00%
{P2, P6}	10 (50%)	{P1}	8 (40%)	8/10=80.00%
{P2, P6}	10 (50%)	{P5}	5 (25%)	5/10=50.00%
{P5, P6}	6 (30%)	{P1, P2}	5 (25%)	5/6=83.33%
{P5, P6}	6 (30%)	{P1}	6 (30%)	6/6=100.00%
{P5, P6}	6 (30%)	{P2}	5 (25%)	5/6=83.33%
{P1}	14 (70%)	{P2, P5, P6}	5 (25%)	5/14=35.71%
{P1}	14 (70%)	{P2, P5}	6 (30%)	6/14=42.85%
{P1}	14 (70%)	{P2, P6}	8 (40%)	8/14=57.14%
{P1}	14 (70%)	{P5, P6}	6 (30%)	6/14=42.85%
{P1}	14 (70%)	{P2}	10 (50%)	10/14=71.42%
{P1}	14 (70%)	{P5}	9 (45%)	9/14=64.28%
{P1}	14 (70%)	{P6}	11 (55%)	11/14=78.57%
A 为 {P2}, {P5}, {P6} 能推出哪些 B。可类同 A 为 {P1} 时那样处理。此处略				

(2)关联规则的产生

关联规则的生成计算示例

最终输出的规则表

项集 A	项集 A 支持度 计数(支持度)	项集 B	项集 A∪B 的支持 度计数(支持度)	置信度 = 项集(A∪B)的支 持度 ÷ 项集 A 的支持度
{ P1, P2 }	10 (50%)	{ P6 }	8 (40%)	8/10=80.00%
{ P1, P6 }	11 (55%)	{ P2 }	8 (40%)	8/11=72.72%
{ P2, P6 }	10 (50%)	{ P1 }	8 (40%)	8/10=80.00%
{ P1 }	14 (70%)	{ P2 }	10 (50%)	10/14=71.42%
{ P1 }	14 (70%)	{ P6 }	11 (55%)	11/14=78.57%
A 为{P2},{P5},{P6}能推出哪些 B, 可类同 A 为{P1}时那样处理, 此处略.				

"P1,P2"⇒"P6"[支持度=50%，置信度=80%]
"P1,P6"⇒"P2"[支持度=55%，置信度=72.72%]
"P2,P6"⇒"P1"[支持度=50%，置信度=80%]
"P1"⇒"P2"[支持度=70%，置信度=71.42%]
"P1"⇒"P6"[支持度=70%，置信度=78.57%]

(3)还能挖掘什么？

关联规则挖掘

单维度单层次规则

$buys(X, "面包") \Rightarrow buys(X, "果酱")$ X代表顾客

多维度多层次规则

$age(X, "30...39") \wedge income(X, "42K...48K") \Rightarrow buys(X, "high_resolution_TV")$

由数据库到数据挖掘II-数据挖掘示例之计算过程-2

(3)还能挖掘什么？

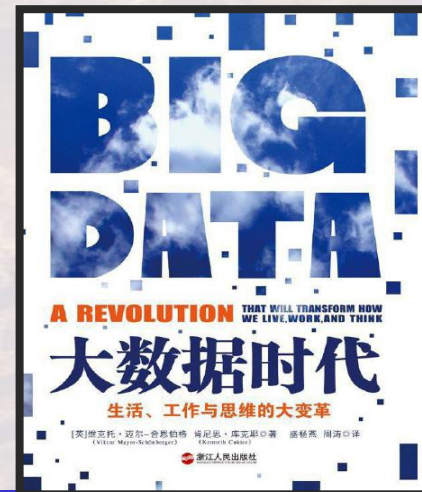
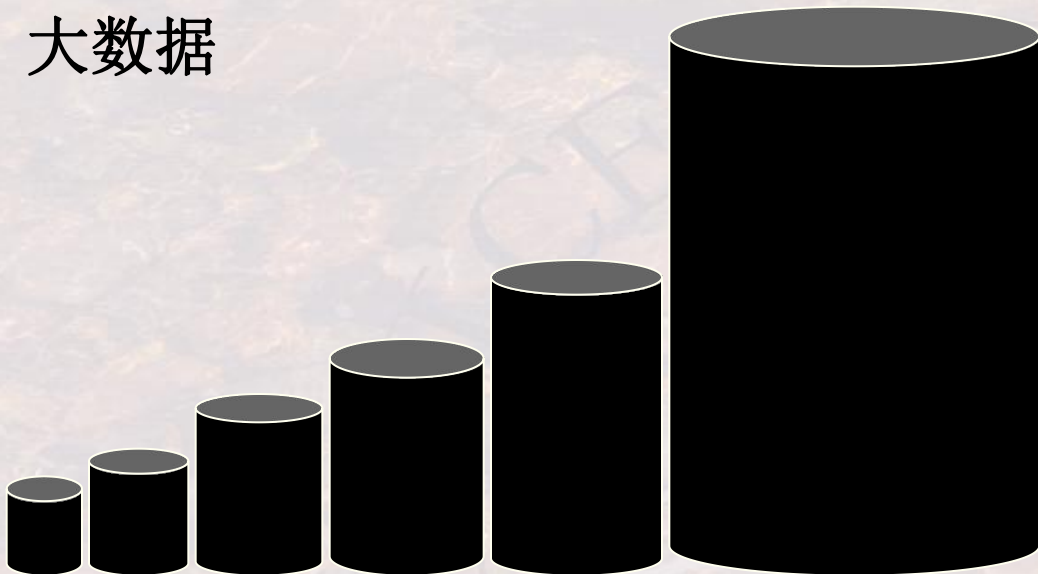
Web数据挖掘

	“微博”挖掘	“超市数据”挖掘
数据基本组织形式	文本---非结构化数据	“表”---结构化数据
被挖掘数据D的集合	众多人众多次：发表的微博	众多人众多次：购买的商品
事务数据T的涵义	一次发表的“微博”可以看作是“若干词汇”的集合	一次购买的商品可以看作是“若干商品”的集合
项的集合	“词汇”的集合	“商品”的集合
频繁项集	频繁使用的“词汇”集合	频繁购买的“商品”集合
规则“A \Rightarrow B”	使用了“词汇A”也使用了“词汇”B	购买了“商品A”也购买了“商品B”
规则挖掘的意义	通过分析，可发现“可以组合在一起的关键词汇”，进而进行主题词设置、读者兴趣引导，以提高某主题的关注度、粉丝的聚集度等	通过分析，可发现“可被组合在一起的商品”进而进行位置、政策等的调整，以提高客户的购买兴趣等

由数据库到数据挖掘II-数据挖掘示例之计算过程-2

(5)小结

大数据



只求关系，不求因果

不要相信经验，一切以数据说话

大数据环境下什么不能发生呢？

bit & Byte

1KB(Kilobyte) = 2^{10} 字节

1MB(Megabyte) = 2^{10} KB

1GB(Gigabyte) = 2^{10} MB

1TB(Trillionbyte) = 2^{10} GB = 2^{20} MB

1PB(Petabyte) = 2^{10} TB = 2^{30} MB

1EB(Exabyte) = 2^{10} PB = 2^{40} MB

1ZB(Zettabyte) = 2^{10} EB = 2^{50} MB

1YB(Yottabyte) = 2^{10} ZB = 2^{60} MB

1BB(Brontobyte) = 2^{10} YB = 2^{70} MB