数据库系统基础

哈尔滨工业大学

第3章 SQL语言

第3章 SQL语言

3.1 SQL语言概述

---- SQL语言提出和发展

---- SQL语言概揽

---- 本章的目标

3.2 简单的SQL-DDL/DML: 创建数据库

3.3 SQL-DML之查询Select

3.4 SQL-DML之更新Insert/Update/Delete

3.5 SQL-视图及DDL的进一步介绍

3.1 SQL语言简述

---- SQL语言提出和发展

- 1974年，由Boyce和Chamber提出
- 1975-1979年，在System R上首次实现，由IBM的San Jose研究室研制，称为Sequel(Structured English QUEry Language)
- 1986年推出了SQL标准：SQL-86， “数据库语言SQL: Structured Query Language”
- 1989年ANSI / ISO推出了SQL标准: SQL-89, 数据库语言SQL的标准集合
- 1992年进一步推出了SQL标准：SQL-92，也称为SQL2
  - 是SQL-89的超集
  - 增加了许多新特性，如新数据类型，更丰富数据操作，更强完整性支持等
  - 原SQL-89被称为entry-SQL, 扩展的被称为Intermediate级和Full级

3.1 SQL语言简述

---- SQL语言提出和发展(续)

- 1999年进一步推出了SQL标准： SQL-99，也称为SQL3
  - 对面向对象的一些特征予以支持，支持抽象数据类型，支持行对象和列对象等
  - 对递归、触发等复杂操作也予以规范化定义
  - 有些特征，现有数据库厂商尚不能做到完全支持
  - 废弃了SQL2的分级，但定义了core-SQL及扩展的SQL
- SQL 2003； SQL 2006； SQL 2008
- SQL还有一个标准是X/Open标准，主要强调各厂商产品的可移植性，只包含被各厂商广泛认可的操作

3.1 SQL语言简述

---- SQL语言概览

➢SQL语言是集DDL、DML和DCL于一体的数据库语言

➢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)
  - ✓模式的定义和删除，包括定义Database, Table, View, Index,完整性约束条件等，也包括定义对象(RowType行对象, Type列对象)
- ❑ DML语句引导词：Insert, Update, Delete, Select
  - ✓ 各种方式的更新与检索操作，如直接输入记录，从其他Table(由SubQuery建立)输入
  - ✓ 各种复杂条件的检索，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

3.1 SQL语言简述

---- 本章目标

- 全面地完整地掌握SQL语言的各种语法和能力
- 复杂查询请求能够熟练地准确地用SQL语言表达

第3章 SQL语言

3.1 SQL语言概述

3.2 简单的SQL-DDL/DML: 创建数据库

---- 课堂讲义中的示例数据库

---- 创建数据库之Create Table: 定义表

---- 创建数据库之Insert：追加表中的元组

3.SQL-DML之查询Select

3.4 SQL-DML之更新Insert/Update/Delete

3.5 SQL-视图及DDL的进一步介绍

3.2 简单的SQL-DDL/DML: 创建数据库

---- 课堂讲义中使用的数据库

- 学生选课数据库SCT
- 学生：学号S#, 姓名Sname, 性别Ssex, 年龄Sage, 所属系别D#

  Student(S# char(8), Sname char(10), Ssex char(2), Sage integer, D# char(2), Sclass char(6))
- 院系：系别D#，系名Dname, 系主任Dean

  Dept (D# char(2), Dname char(10), Dean char(10))

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

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

3.2 简单的SQL-DDL/DML: 创建数据库

---- 课堂讲义中使用的数据库(续)

- 课程：课程号C#, 课程名CName, 任课教师编号T#, 学时Hours

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

- 教师：教师编号T#，教师名TName, 所属院系D#，工资Salary

  Teacher (T# char(3), Tname char(10), D# char(2), Salary float(2))

- 选课：学生号S#, 课程号C#, 成绩Score

  SC(S# char(8), C# char(3), Score float(1))

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

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

3.2 简单的SQL-DDL/DML: 创建数据库

---- SQL-DDL

- DDL: Data Definition Language
  - 创建数据库(DB)
  - 创建DB中的Table(定义关系模式)
  - 定义Table的约束条件
  - 定义Table中各个属性的约束条件
  - 定义View、Index、Tablespace… …等
  - 上述各种定义的撤消与修正
- DDL通常由DBA来使用，也有经DBA授权后由应用程序员来使用
- 我们先学习Create Database及Create Table的简单形式，其他内容将在后续章节介绍或参阅相关使用手册

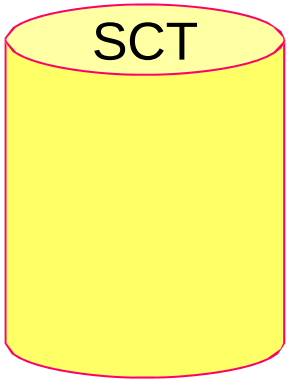
3.2 简单的SQL-DDL/DML: 创建数据库

---- 创建数据库

- 数据库(Database)是若干具有相互关联关系的Table/Relation的集合.
可以简单看作是一个集中存放若干Table的大型文件
- 创建Database，需使用create database语句
- create database的简单语法形式为：

  create database 数据库名;
- 例如：创建我们的课程学习数据库SCT

  create database SCT;
- 接下来就可以在其中定义Table了



3.2 简单的SQL-DDL/DML: 创建数据库

---- 创建Table

- 创建Table，需使用create table语句
- create table简单语法形式为：

  create table 表名(列名 数据类型 [*Primary key |UNIQUE*] [*not null*] [,
  列名 数据类型 [*not null*] , …]) ;

- **PRIMARY KEY:** 主键约束，给定的一列或多列，每个表只能创建一主键约束
- **UNIQUE:** 唯一性约束，候选键，一个表可以有多个UNIQUE 约束
- **Not Null:** 是指该列允许不允许有空值出现，如选择了Not Null表明该列不允许有空值出现。通常主键是不允许有空值的。
- 语法中的数据类型在SQL标准中有定义

3.2 简单的SQL-DDL/DML: 创建数据库

---- 创建Table(续)

- 在SQL-92标准中定义的数据类型
  - Char (n) :固定长度的字符串
  - Varchar (n) :可变长字符串
  - int :整数 // 有时不同系统也写作integer
  - Numeric (p，q) :固定精度数字，小数点左边p位，右边p-q位
  - real :浮点精度数字 // 有时不同系统也写作float(n)，小数点后保留n位
  - Date :日期 (2003-09-12)
  - time ：时间 (23:15:003)
  - …
- 现行商用DBMS的数据类型有时和上面有些差异，请注意;和高级语言的数据类型，总体上是一致的，但也有些差异

3.2 简单的SQL-DDL/DML: 创建数据库

---- 创建Table(续)

- 例如定义学生表：Student

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

- 再例如定义课程表：Course

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

- 同学可自己定义另外三个表： Dept, Teacher, SC

3.2 简单的SQL-DDL/DML: 创建数据库

---- 向表中追加元组的值

- 在表中追加元组的值要使用DML
- DML: Data Manipulation Language
  - 向Table中追加新的元组：Insert
  - 修改Table中某些元组中的某些属性的值: Update
  - 删除Table中的某些元组: Delete
  - 对Table中的数据进行各种条件的检索: Select
- DML通常由用户或应用程序员使用，访问经授权的数据库
- 我们先学习Insert的简单形式，其他内容将在后续章节介绍或参阅相关使用手册

3.2 简单的SQL-DDL/DML: 创建数据库

---- 向表中追加元组的值(续)

- 追加元组，需使用insert into语句
- insert into 简单语法形式为：

  insert into 表名[(列名 [, 列名]…]

  values (值 [, 值] , …) ;

- 语法中的values后值的排列，须与into子句后面的列名排列一致。
- 若表名后的所有列名省略， 则values后的值的排列，须与该表存储中的列名排列一致。

3.2 简单的SQL-DDL/DML: 创建数据库

---- 向表中追加元组的值(续)

- 例如：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');
```

- 再例如：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');
```

第3章 SQL语言

3.1 SQL语言概述

3.2 简单的SQL-DDL/DML: 创建数据库

3.3 SQL-DML之查询Select

---- 基本的检索操作

---- 多表联合检索

---- 子查询

---- 结果计算与聚集函数

---- 分组查询与分组过滤

---- 并、交、差的处理

---- 空值处理

---- 内连接、外连接

---- SQL的完整语法

3.3 SQL-DML之查询Select

---- 基本的检索操作

- SQL提供了结构形式一致但, 功能多样化的检索语句Select
- 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，相当于$\Pi_{列名,\ldots,列名}(\sigma_{检索条件}(表名))$
- Select语句中的select … , from… , where…, 等被称为子句，在以上基本形式基础上会增加许多构成要素，也会增加许多新的子句，满足不同的需求，下面将逐步介绍。

3.3 SQL-DML之查询Select

---- 基本的检索操作

- 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                 //投影出某些列
From  Student;
```

- 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                  //投影的列可以重新排定顺序
  From  Student
Where   Sage <= 19;
```

3.3 SQL-DML之查询Select

---- 基本的检索操作

- 检索条件的书写，与选择运算$\sigma_{con}(R)$的条件con书写是一样的，只是其逻辑运算符用 and , or, not 来表示, 同时也要注意运算符的优先次序及括弧的使用。更要注意对汉语检索条件的正确理解。
- 例如：检索教师表中所有工资少于1500元或者工资大于2000元并且是03系的教师姓名？如何理解与书写检索条件？

```
Select   Tname
  From  Teacher
Where   Salary < 1500  or  Salary  > 2000  and  D# = '03';

Select   Tname
  From  Teacher
Where   (Salary < 1500  or  Salary  > 2000)  and  D# = '03';
```

3.3 SQL-DML之查询Select

---- 基本的检索操作

- 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

3.3 SQL-DML之查询Select

---- 基本的检索操作

- 结果唯一性问题。尽管关系模型要求无重复元组出现在数据库中，但现实DBMS操作中，是允许检索结果出现重复元组的，但也允许无重复元组。
- 在关系(存储的Table)中要求无重复元组是通过定义主键或UNIQUE来保证的，在检索结果中要求无重复元组, 是通过DISTINCT保留字的使用来实现的。
- 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
```

3.3 SQL-DML之查询Select

---- 基本的检索操作

- 结果排序问题。DBMS可以对检索结果进行排序，可以升序排列，也可以降序排列。
- 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 ;
```

3.3 SQL-DML之查询Select

---- 基本的检索操作

- 模糊查询问题。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_

3.3 SQL-DML之查询Select

---- 基本的检索操作

- 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%';
```

3.3 SQL-DML之查询Select

---- 多表联合检索

- 多表联合检索可以通过连接运算来完成，而连接运算又可以通过广义笛卡尔积后再进行选择运算来实现。因此,
- 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$\Pi_{列名,\ldots,列名}(\sigma_{检索条件}(表名1 \times 表名2 \times \ldots))$
- 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等值连接、不等值连接及各种θ-连接。

3.3 SQL-DML之查询Select

---- 多表联合检索

- θ-连接之等值连接示例
- 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;
```

3.3 SQL-DML之查询Select

---- 多表联合检索

- 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

3.3 SQL-DML之查询Select

---- 多表联合检索

- θ-连接之不等值连接示例
- 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 ;
```

- 请写：求 ‘001’号课程有成绩差的任意两位同学的学号
- 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。

3.3 SQL-DML之查询Select

---- 多表联合检索

➢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   S1.S #   From  SC S1, SC S2
Where   S1.S# = S2.S#  and  S1.C#='001'
          and  S2.C#='002' ;
```

➢ 再如：求 “001”号课成绩比 “002”号课成绩高的所有学生的学号(二表连接)

```
Select   S1.S #   From  SC   S1, SC   S2
Where   S1.S# = S2.S#  and  S1.C#='001'
          and  S2.C#='002  and  S1.Score > S2.Score;
```

3.3 SQL-DML之查询Select

---- 多表联合检索

- 注意正确理解用汉语表达的查询的语义，并正确表达
- 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  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

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之更新

- 元组新增Insert：新增一个或一些元组到数据库的Table中
- 元组更新Update: 对某些元组中的某些属性值进行重新设定
- 元组删除Delete：删除某些元组

- SQL-DML既能单一记录操作，也能对记录集合进行批更新操作
- SQL-DML之更新操作需要利用前面介绍的子查询(Subquery)的概念，以便处理“一些”、“某些”等。

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组新增Insert

- 元组新增Insert命令有两种形式
- 单一元组新增命令形式：插入一条指定元组值的元组

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

- 批数据新增命令形式：插入子查询结果中的若干条元组。待插入的元组由子查询给出。

```
insert into 表名 [(列名[，列名]...)]
子查询;
```

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组新增Insert(续)

- 单一元组新增Insert示例

```
Insert Into Teacher (T#, Tname, D#, Salary)
 Values (“005”, “阮小七” , “03”, “1250”);

Insert Into Teacher
 Values (“006”, “李小虎” , “03”, “950”);
```

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组新增Insert(续)

- 批元组新增Insert示例
- 如果我们新建立了一个Table: St(S#, Sname, avgScore), 然后将检索到的同学的平均成绩新增到该表中

```
Insert Into St (S#, Sname, avgScore)
    Select S#, Sname, Avg(Score) From Student, SC
    Where Student.S# = SC.S#
    Group by Student.S# ;
```

- 当有元组新增操作时，DBMS会检查用户定义的完整性约束条件等，如不符合完整性约束条件，则将不会执行新增动作。我们将在后面介绍。

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组删除Delete

- 元组删除Delete命令：删除满足指定条件的元组

  Delete From 表名 [Where 条件表达式] ;

- 如果Where条件省略，则删除所有的元组。

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组删除Delete(续)

- 元组删除Delete示例
- 删除SC表中所有元组

```
Delete From SC;
```

- 删除98030101号同学所选的所有课程

```
Delete From SC Where S# = '98030101';
```

- 删除自动控制系的所有同学

```
Delete From Student Where D# in
        (Select D# From Dept Where Dname = '自动控制');
```

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组删除Delete(续)

- 删除有四门不及格课程的所有同学

```
Delete From Student Where S# in
    ( Select S# From SC Where Score < 60
      Group by S# Having Count(*)>= 4 );
```

- 当有删除时，DBMS会检查用户定义的完整性约束条件等，如不符合完整性约束条件，则将不会执行删除动作。我们将在后面介绍。

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组更新Update

- 元组更新Update命令：用指定要求的值更新指定表中满足指定条件的元组的指定的列的值

```
Update  表名
Set  列名 = 表达式 | (子查询)
     [ [ , 列名 = 表达式 | (子查询) ] … ]
[ Where  条件表达式] ;
```

- 如果Where条件省略，则更新所有的元组。

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组更新Update(续)

- 元组更新Update示例
- 将所有教师工资上调5%

```
Update  Teacher
Set   Salary  = Salary * 1.05  ;
```

- 将所有计算机系的教师工资上调10%

```
Update  Teacher
Set   Salary  = Salary * 1.1
Where  D#  in
        ( Select   D#  From  Dept   Where  Dname = '计算机'  );
```

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组更新Update(续)

- 当某同学001号课的成绩低于该课程平均成绩时，将该同学该门课成绩提高5%

```
Update  SC
Set   Score  = Score * 1.05
Where   C# = '001'  and   Score < some
                ( Select   AVG(Score)  From SC
                  Where  C# = '001' ) ;
```

3.4 SQL-DML之更新Insert/Update/Delete

---- SQL-DML之元组更新Update(续)

- 将张三同学001号课的成绩置为其班级该门课的平均成绩

```
Update  SC
Set   Score  = ( Select   AVG(SC2.Score)
                  From  SC  SC1, Student S1, SC SC2, Student  S2
                  Where  S1.Sclass = S2.Sclass  and  SC1.S# = S1.S#
                         and SC2.S# = S2.S# and S1.Sname='张三'
                         and SC1.C# = '001'  and  SC1.C# = SC2.C#  )
Where   C# = '001'  and   S#  in ( Select   S#  From  Student
                  Where  Sname = '张三'   ) ;
```

第3章 SQL语言

3.1 SQL语言概述

3.2 简单的SQL-DDL/DML: 创建数据库

3.3 SQL-DML之查询Select

3.4 SQL-DML之更新Insert/Update/Delete

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的概念与结构

---- 视图的定义与使用

---- 视图更新问题

---- SQL-DDL的进一步介绍: 撤消语句的使用

3.5 SQL-视图及DDL的进一步介绍

---- SQL-DDL的撤消与修改语句

- 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

- 例如，撤消学生表Student

  Drop Table Student;

- 再例如，撤消教师表Teacher

  Drop Table Teacher;

3.5 SQL-视图及DDL的进一步介绍

---- SQL-DDL的撤消与修改语句(续)

- 注意，SQL-delete语句只是删除表中的元组, 而撤消基本表drop table的操作是撤消包含表格式、表中所有元组、由该表导出的视图、等相关的所有内容，所以使用要特别注意。
- 当表定义完成后，通常不是撤消表，而可能要修正表的定义，此时可使用Alter table语句
- 修正基本表的定义

```
alter  table  tablename
[add  {colname  datatype, …}]          增加新列
[drop {完整性约束名}]                  删除完整性约束
[modify {colname  datatype, …}]        修改列定义
```

3.5 SQL-视图及DDL的进一步介绍

---- SQL-DDL的撤消与修改语句(续)

➢例如，在学生表Student(S#,Sname,Ssex,Sage,D#,Sclass)基础上增加二列Saddr, PID

```
Alter  Table  Student
Add   Saddr  char[40],  PID  char[18] ;
```

➢ 再上例将表中Sname列的数据类型增加两个字符

```
Alter  Table  Student
Modify  Sname char(10) ;
```

➢ 删除学生姓名必须取唯一值的约束

```
Alter  Table  Student
Drop  Unique( Sname );
```

3.5 SQL-视图及DDL的进一步介绍

---- SQL-DDL的撤消与修改语句(续)

- 撤消数据库

  drop database 数据库名;

- 例如撤消SCT数据库

  Drop database SCT;

3.5 SQL-视图及DDL的进一步介绍

---- SQL-DDL的数据库指定与关闭命令

- 有些DBMS提供了操作多个数据库的能力，此时在进行数据库操作时需要指定待操作数据库与关闭数据库的功能。
- 指定当前数据库

  use 数据库名;
- 关闭当前数据库

  close 数据库名;

第3章 SQL语言

3.1 SQL语言概述

3.2 简单的SQL-DDL/DML: 创建数据库

3.3 SQL-DML之查询Select

子查询

3.3 SQL-DML之查询Select

---- 子查询

- 在现实中，很多情况需要进行下述条件的判断
  - 集合成员资格
    - ✓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
- 这时，有可能需要子查询来解决问题
- 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。

3.3 SQL-DML之查询Select

---- 子查询之类型1：IN谓词

- 第1种类型的子查询：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。语法中，表达式的最简单形式就是列名或常数。

- 例如：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";
```

3.3 SQL-DML之查询Select

---- 子查询之类型1：IN谓词(续)

➢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' );
```

3.3 SQL-DML之查询Select

---- 子查询之类型1：IN谓词(续)

- 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# );
```

3.3 SQL-DML之查询Select

---- 非相关子查询

- 如下所示，将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

- 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。
- 前面的子查询示例都是非相关子查询

3.3 SQL-DML之查询Select

---- 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。

3.3 SQL-DML之查询Select

---- 子查询之类型2：θ some / θ all谓词

- 第2种类型的子查询： θ some / θ all子查询。其基本语法为：

  表达式 *θsome* (子查询)

  表达式 *θall* (子查询)

- 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 *θsome* (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；

3.3 SQL-DML之查询Select

---- 子查询之类型2：θ some / θ all谓词(续)

- 例如：找出工资最低的教师姓名

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

- 再例如：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”);
```

3.3 SQL-DML之查询Select

---- 子查询之类型2：θ some / θ all谓词(续)

- 再例如：找出所有课程都不及格的学生姓名

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

- 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

3.3 SQL-DML之查询Select

---- 子查询之类型2：θ some / θ all谓词(续)

- θ any 被θ some替换的原由
- 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。
- 例如：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  < some  ( Select   Salary  From  Teacher  );

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

3.3 SQL-DML之查询Select

---- 子查询之类型2：θ some / θ all谓词(续)

- 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)

- 例如：

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

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

3.3 SQL-DML之查询Select

---- 子查询之类型2：θ some / θ all谓词(续)

- 然而下面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)

3.3 SQL-DML之查询Select

---- 子查询之类型3：Exists谓词

- 第3种类型的子查询： Exists子查询。其基本语法为：

  [not] Exists (子查询)

- 语义：子查询结果中有无元组存在。
- 例如：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' );
```

3.3 SQL-DML之查询Select

---- 子查询之类型3：Exists谓词(续)

- 再例如：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# );
```

3.3 SQL-DML之查询Select

---- 子查询之类型3：Exists谓词(续)

- 然而not Exists却可以实现很多新功能
- 例如：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# ) );
```

- 上述语句的意思是：不存在有一门001号教师主讲的课程该同学没学过

3.3 SQL-DML之查询Select

---- 子查询之类型3：Exists谓词(续)

- 再例如：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# ) );
```

3.3 SQL-DML之查询Select

---- 子查询之类型3：Exists谓词(续)

- 再例如：
- 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 ) );
```

3.3 SQL-DML之查询Select

---- 结果计算与聚集函数

- 前面介绍的select-from-where语句中，select子句后面不仅可以是列名，而且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。特殊函数的使用需结合各自DBMS的说明书。

3.3 SQL-DML之查询Select

---- 结果计算与聚集函数(续)

- 例如，有计算表达式投影列的示例:
- 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;
```

3.3 SQL-DML之查询Select

---- 结果计算与聚集函数(续)

- 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

3.3 SQL-DML之查询Select

---- 结果计算与聚集函数(续)

- 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#;
```

3.3 SQL-DML之查询Select

---- 结果计算与聚集函数(续)

- 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
- 以上问题，将在分组查询中解决

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤

- 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, …

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤(续)

- 例如前面例子： 求每一个学生的平均成绩

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

- 上述例子是按学号进行分组，即学号相同的元组划到一个组中并求平均值。
- 改变分组条件便形成了另外的结果：求每一门课程的平均成绩

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

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤(续)

➢不同分组条件分组示例

Group by S#

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 C#

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

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤(续)

- 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#;
```

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

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤(续)

- 若要对集合(即分组)进行条件过滤，可使用Having子句
- Having子句，又称为分组过滤子句。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]] ;

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤(续)

- 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;
```

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤(续)

- HAVING子句与WHERE子句表达条件的区别如下图示例

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

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤(续)

- 分组查询仍需要注意语义问题
- 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？

3.3 SQL-DML之查询Select

---- 分组查询与分组过滤(续)

➢ 前述写法是不正确的，它求出的是那两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，后者是题目要求的。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# ;
```

3.3 SQL-DML之查询Select

---- 并、交、差的处理

- 关系代数中有并运算、交运算和差运算，SQL语言也能实现其运算
- SQL语言中为并运算提供了运算符UNION, 为交运算提供了运算符INTERSECT, 为差运算提供了运算符EXCEPT，其基本语法形式为：

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

- 上述操作，在通常情况下是自动删除重复元组。但有时为了保留重复的元组，则需使用ALL保留字，ALL保留字是允许重复元组出现，具体使用如下：
  - 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

3.3 SQL-DML之查询Select

---- 并、交、差的处理(续)

- 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';
```

3.3 SQL-DML之查询Select

---- 并、交、差的处理(续)

- 有时只能用UNION的方式
- 例如：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 ;
```

3.3 SQL-DML之查询Select

---- 并、交、差的处理(续)

- 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，如上例所述。只是有了Intersect更容易表达一些，但增加了SQL语言的不唯一性。

3.3 SQL-DML之查询Select

---- 并、交、差的处理(续)

- 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
- 上例不能写成如下形式：

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

- 上例可写成如下形式：所有学生 – 学过002号课的学生

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

3.3 SQL-DML之查询Select

---- 并、交、差的处理(续)

- 前述语句也可采用如下不用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，如上例所述。只是有了Except更容易表达一些，但增加了SQL语言的不唯一性。

3.3 SQL-DML之查询Select

---- 并、交、差的处理(续)

- 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。

3.3 SQL-DML之查询Select

---- 空值处理

- 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。

SC

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

3.3 SQL-DML之查询Select

---- 空值处理(续)

- 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
- 例如：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

3.3 SQL-DML之查询Select

---- 空值处理(续)

- 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

3.3 SQL-DML之查询Select

---- 空值处理(续)

- 例如：

  Select AVG(Score) From SC;

  上例的值(参考右图)为73.5 = (92 + 55)/2。

- 再例如：

  Select COUNT(*) From SC;

  上例的值为3。

SC

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

3.3 SQL-DML之查询Select

---- 内连接、外连接

- 在SQL的高级语法中引入了内连接与外连接运算，具体形式如下：

```
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)

3.3 SQL-DML之查询Select

---- 内连接、外连接(续)

- **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。

3.3 SQL-DML之查询Select

---- 内连接、外连接(续)

- 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

3.3 SQL-DML之查询Select

---- 内连接、外连接(续)

➢ **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;
```

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

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

3.3 SQL-DML之查询Select

---- 内连接、外连接(续)

➢ **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 ;
```

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

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

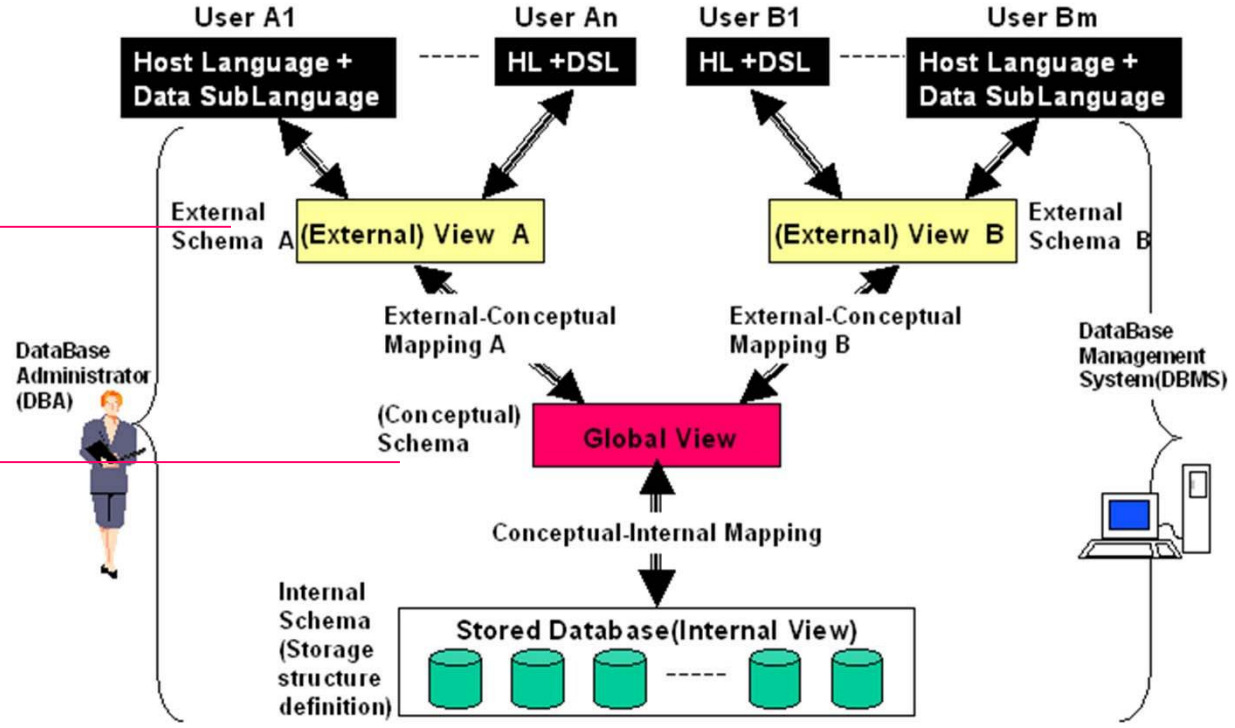
3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的概念和结构

- 回顾我们学习过的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)

View

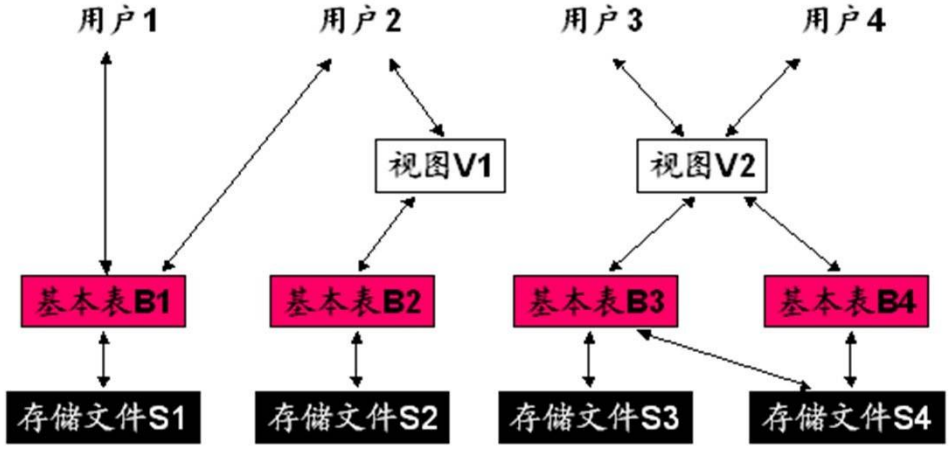
Base Table



3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的概念和结构(续)

- 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。



3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的定义与使用

- 使用视图需要定义，定义视图采用Creat View语句
- 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

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的定义与使用(续)

- 视图定义举例：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' ) ;
```

- 再例如定义一个视图**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。

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

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的定义与使用(续)

- 当定义好视图后，视图可以像Table一样参与SQL的各种操作语句中使用。
- 例如，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'   ;
```

- 再例如，检索计算机系的所有学生，我们可使用CompStud

```
Select  *  From  CompStud;
```

- 再例如，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 ;
```

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的定义与使用(续)

- 定义视图，有时可方便用户进行检索操作。
- 例如，我们可定义一个视图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# ) ;
```

- 在定义了视图StudStat后，我们再检索某一学生平均成绩就很简单了：

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

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的更新

➢ SQL视图更新操作是一个比较复杂的问题，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。例如：

```
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?

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的更新

- 再例如：

```
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。

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的更新

- 因此，SQL视图更新操作受到很大的约束，很多情况是不能进行视图更新的。
  - 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

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的更新

- 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" )
```

3.5 SQL-视图及DDL的进一步介绍

---- SQL视图的撤消(续)

- 已经定义的视图也可以撤消
- 撤消视图

  drop view view_name

- 例如，撤消视图Teach

  Drop View Teach;

- 再例如，撤消视图CompStud

  Drop View CompStud;

3.7 小结

---- 本章我们学习了以下一些概念

- SQL- DML中的查询语句Select
  - 基本的查询操作
  - 多表联合查询
  - 结果计算与聚集函数
  - 分组查询与分组过滤
  - Select的完整语法
- SQL- DML中的操作语句
  - 元组新增操作：Insert
  - 元组更新操作：Update
  - 元组删除操作：Delete