

数据库系统概论

赵刚

北京理工大学

关系型数据库标准语言**SQL**

- ◆ **SQL (Structured Query Language)**
- ◆ **1974年由Boyce chamberlin提出**
- ◆ **1975—1979在system R中实现**
- ◆ **最终成为关系型数据库的标准语言**
- ◆ **1986 ANSI 接受成为美国标准 SQL-86**
- ◆ **1987 ISO 接受成为国际标准**
- ◆ **1989 发布SQL-89标准**
- ◆ **1992 发布SQL-92标准**
- ◆ **新的标准 SQL-99 亦称为SQL-3**
- ◆ **2003 年 SQL2003 简称SQL4**
- ◆ **标准的统一，促进了SQL语言的进一步发展完善**

SQL概述

- ◆ **SQL**是基于关系代数和关系演算之间的结构化查询语言
- ◆ 功能不仅仅是查询，是通用的功能极强的关系数据库操纵语言。
- ◆ 集数据查询，数据操纵，数据定义，数据控制于一体

SQL特点

◆ 综合统一

- ◆ 数据操纵语言统一
- ◆ 数据结构统一

◆ 高度非过程化

- ◆ 无需指明操作路径
- ◆ 只需提出做什么，无需指明怎么做

◆ 面向集合的操作方式

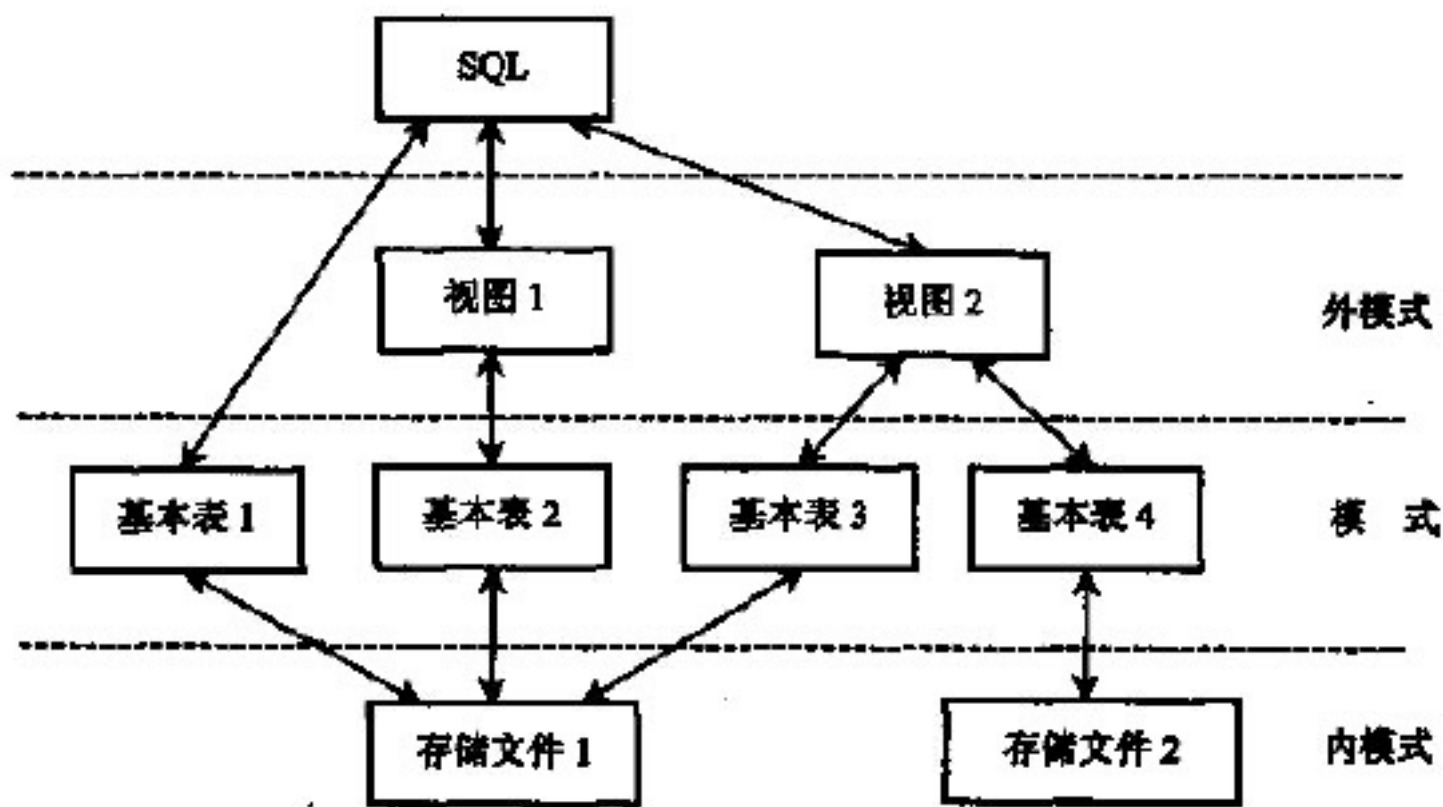
◆ 同一语法两种使用方式

- ◆ 交互方式
- ◆ 嵌入方式

◆语言简洁，易学易用 一共9个动词

SQL 功能	动 词
数据查询	SELECT
数据定义	CREATE, DROP, ALTER
数据操纵	INSERT, UPDATE, DELETE
数据控制	GRANT, REVOKE

SQL语言的系统结构



- ◆ 外模式对应于视图(view)和部分基本表 (Base Table)
- ◆ 模式对应于基本表 (Base Table)
- ◆ 内模式对应于存储文件
- ◆ SQL可以对基表和视图进行操作，都是关系
- ◆ 基表是独立存在的表，一个关系对应一个表
- ◆ 一个或多个基表对应1个存储文件
- ◆ 存储文件构成内模式，对用户透明
- ◆ 视图是1个或多个基表导出的表，不独立储存在数据库中

数据定义

操作对象	操作方式		
	创建	删除	修改
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

◆SQL2 还增加了模式定义的功能

模式定义与删除

CREATE SCHEMA <模式名> AUTHORIZATION <用户名>

DROP SCHEMA <模式名> <CASCADE|RESTRICT>

◆ 为用户**WANG**定义一个学生-课程模式**S-T**

◆ **CREATE SCHEMA “S-T” AUTHORIZATION**

WANG;

定义删除和修改基本表

```
CREATE TABLE <表名> ( <列名> <数据类型>[ 列级完整性约束条件 ]  
    [, <列名> <数据类型>[ 列级完整性约束条件 ] ] ...  
    [, <表级完整性约束条件> ] ) ;
```

建立一个“课程”表**Course**

CREATE TABLE Course

(Cno CHAR(4) PRIMARY KEY,

Cname CHAR(40),

先修课

Cpno CHAR(4),

Ccredit SMALLINT,

FOREIGN KEY (Cpno) REFERENCES Course(Cno)

);

Cpno是外码 被参
照表是**Course** 被
参照列是**Cno**

建立一个学生选课表**SC**

CREATE TABLE SC

(Sno CHAR(9),

Cno CHAR(4),

Grade SMALLINT,

PRIMARY KEY (Sno,Cno),

/ 主码由两个属性构成，必须作为表级完整性进行定义*/*

FOREIGN KEY (Sno) REFERENCES Student(Sno),

/ 表级完整性约束条件，Sno是外码，被参照表是Student */*

FOREIGN KEY (Cno)REFERENCES Course(Cno)

/ 表级完整性约束条件， Cno是外码，被参照表是Course*/*

);

数据类型例子

SMALLINT

半字长二进制整数。

INTEGER 或 INT

全字长二进制整数。

DECIMAL(p[, q])

压缩十进制数，共 p 位，其中小数点后有 q 位。 $0 \leq q \leq p \leq 15$ ， $q=0$ 时可以省略不写。

或 **DEC(p[, q])**

FLOAT

双字长浮点数。

CHARACTER(n) 或 CHAR(n)

长度为 n 的定长字符串。

VARCHAR(n)

最大长度为 n 的变长字符串。

GRAPHIC(n)

长度为 n 的定长图形字符串。

VARGRAPHIC(n)

最大长度为 n 的变长图形字符串。

DATE

日期型，格式为 **YYYY-MM-DD**。

TIME

时间型，格式为 **HH.MM.SS**。

TIMESTAMP

日期加时间。

模式和表

- ◆1 表名显式标识模式名
- ◆2 创建模式同时创建表
- ◆3 设置搜索路径

修改基表

ALTER TABLE <表名>

[ADD[COLUMN] <新列名> <数据类型> [完整性约束]] [ADD <表级完整性约束>]

[DROP [COLUMN] <列名> [CASCADE| RESTRICT]]

[DROP CONSTRAINT<完整性约束名>[RESTRICT | CASCADE]]

[ALTER COLUMN <列名><数据类型>] ;

修改基本表（续）

向**Student**表增加“入学时间”列，其数据类型为日期型

```
ALTER TABLE Student ADD S_entrance DATE;
```

不管基本表中原来是否已有数据，新增加的列一律为空值

修改基本表（续）

将年龄的数据类型由字符型（假设原来的数据类型是 字符型）
改为整数。

```
ALTER TABLE Student ALTER COLUMN Sage INT;
```

增加课程名称必须取唯一值的约束条件。

```
ALTER TABLE Course ADD UNIQUE(Cname);
```

删除基本表

DROP TABLE <表名> [RESTRICT| CASCADE] ;

❖ **RESTRICT:** 删除表是有限制的。

- 欲删除的基本表不能被其他表的约束所引用
- 如果存在依赖该表的对象，则此表不能被删除

❖ **CASCADE:** 删除该表没有限制。

- 在删除基本表的同时，相关的依赖对象一起删除

删除基本表（续）

删除**Student**表

DROP TABLE Student CASCADE;

- 基本表定义被删除，数据被删除
- 表上建立的索引、视图、触发器等一般也将被删除

建立和删除索引

- ◆ 索引是加快查询的手段，用户可以根据自己的查询需要建立若干索引

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>

ON <表名> (<列名>[<次序>][, <列名>[<次序>]]...) ;

CLUSTER 指明索引是聚簇索引：索引项的顺序和表中记录的物理顺序一致的索引组织

一个基表只能建立一个聚簇索引，对于经常更新的索引不宜建立聚簇索引

删除索引 **drop index <索引名>**

为学生-课程数据库中的**Student**，**Course**，**SC**三个 表建索引。**Student**表按学号升序建唯一索引，**Course**表 按课程号升序建唯一索引，**SC**表按学号升序和课程号降序 建唯一索引

```
CREATE UNIQUE INDEX Stusno ON Student(Sno);
```

```
CREATE UNIQUE INDEX Coucno ON Course(Cno);
```

```
CREATE UNIQUE INDEX SCno ON SC(Sno ASC,Cno DESC);
```

❖ **ALTER INDEX** <旧索引名> **RENAME TO** <新索引名>

■ 将SC表的SCno索引名改为SCSno

```
ALTER INDEX SCno RENAME TO SCSno;
```

❖ **DROP INDEX** <索引名>; 删除索引时，系统会从数据字典中删去有关该索引的描述。

删除Student表的Stusname索引

```
DROP INDEX Stusname;
```


查询

❖ 语句格式

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

FROM <表名或视图名>[,<表名或视图名>]... | (**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]]**;

数据查询

- **SELECT**子句：指定要显示的属性列
- **FROM**子句：指定查询对象（基本表或视图）
- **WHERE**子句：指定查询条件
- **GROUP BY**子句：对查询结果按指定列的值分组，该属性列值相等的元组为一个组。通常会在每组中作用聚集函数。
- **HAVING**短语：只有满足指定条件的组才予以输出
- **ORDER BY**子句：对查询结果表按指定列值的升序或降序排序

❖ 查询指定列

查询全体学生的学号与姓名。

```
SELECT Sno,Sname 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 *

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

◆❖ “虚” 列

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

查全体学生的姓名及其出生年份

```
SELECT Sname,2014-Sage  
FROM Student;
```

Sname	2014-Sage
李勇	1994
刘晨	1995
王敏	1996
张立	1995

查询全体学生的姓名、出生年份和所在的院系，要求用小写字母表示系名。

```
SELECT Sname,'Year of Birth: ',2014-Sage,LOWER(Sdept) FROM  
Student;
```

输出结果:

Sname	'Year of Birth: '	2014-Sage	LOWER(Sdept)
-------	-------------------	-----------	--------------

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
----	---------------------	----

❖ 使用列别名改变查询结果的列标题:

```
SELECT Sname NAME, 'Year of Birth:' BIRTH,  
2014-Sage BIRTHDAY, LOWER(Sdept) DEPARTMENT  
FROM Student;
```

输出结果:

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

❖ 消除取值重复的行 如果没有指定**DISTINCT**

关键词，则缺省为**ALL**

查询选修了课程的学生学号。

SELECT Sno FROM SC;

结果为:

Sno
201215121
201215121
201215121
201215122
201215122

消除取值重复的行（续）

❖ 指定**DISTINCT**关键词，去掉表中重复的行

```
SELECT DISTINCT Sno FROM SC;
```

执行结果：

<u>Sno</u>

201215121

201215122

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

① 比较大小

查询计算机科学系全体学生的名单。

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

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;
```

查询考试成绩有不及格的学生的学号。

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

确定范围

❖ 谓词: **BETWEEN ... AND ...**
NOT BETWEEN ... AND ...

- ◆ 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;

确定集合

❖ 谓词: **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' );
```

确定集合

❖ 谓词: **IN** <值表>, **NOT IN** <值表>

[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的姓名和性别。

```
SELECT Sname, Ssex FROM Student
```

```
WHERE Sdept NOT IN ('IS','MA' , 'CS' );
```

字符匹配

❖ 谓词: **[NOT] LIKE**

<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

■_（下横线） 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

字符匹配（续）

■ 匹配串为固定字符串

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';
```


字符匹配

- 匹配串为含通配符的字符串

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%';
```

字符匹配

- 匹配串为含通配符的字符串

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__';
```

字符匹配（续）

[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%';
```

字符匹配（续）

- 使用换码字符将通配符转义为普通字符

查询DB_Design课程的课程号和学分。

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

⑤ 涉及空值的查询

❖ 谓词: **IS NULL** 或 **IS NOT NULL**

■ “IS” 不能用 “=” 代替

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
```

涉及空值的查询

❖谓词: **IS NULL** 或 **IS NOT NULL**

■ “IS” 不能用 “=” 代替

查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

■ **AND**的优先级高于**OR**

■ 可以用括号改变优先级

[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;
```

■

多重条件查询（续）

❖ 改写

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')
```

可改写为：

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


ORDER 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

ORDER BY子句（续）

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;
```

ORDER BY子句（续）

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;
```

4. 聚集函数

❖ 聚集函数:

■ 统计元组个数

COUNT(*)

■ 统计一列中值的个数

COUNT([DISTINCT|ALL] <列名>)

■ 计算一列值的总和（此列必须为数值型）

SUM([DISTINCT|ALL] <列名>)

■ 计算一列值的平均值（此列必须为数值型）

AVG([DISTINCT|ALL] <列名>)

■ 求一列中的最大值和最小值

MAX([DISTINCT|ALL] <列名>)

MIN([DISTINCT|ALL] <列名>)

聚集函数（续）

查询学生总人数。

```
SELECT COUNT(*)
```

```
FROM Student;
```

查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
```

```
FROM SC;
```

计算1号课程的学生平均成绩。

```
SELECT AVG(Grade) FROM SC
```

```
WHERE Cno= ' 1 ';
```

聚集函数（续）

- ◆ 查询选修1号课程的学生最高分数。
- ◆ `SELECT MAX(Grade) FROM SC`
- ◆ `WHERE Cno='1';`
- ◆ 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;`

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子句（续）

求各个课程号及相应的选课人数。

```
SELECT Cno, COUNT(Sno) FROM SC
```

```
GROUP BY Cno;
```

查询结果可能为：

Cno	COUNT(Sno)
1	22
2	34
3	44
4	33
5	48

GROUP BY子句（续）

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;
```

GROUP BY子句（续）

查询平均成绩大于等于90分的学生学号和平均成绩 下面的语句是不对的：

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

因为**WHERE**子句中是不能用聚集函数作为条件表达式 正确的查询语句应该是：

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

GROUP BY子句（续）

❖ **HAVING**短语与**WHERE**子句的区别：

- 作用对象不同
- **WHERE**子句作用于基表或视图，从中选择满足条件的元组
- **HAVING**短语作用于组，从中选择满足条件的**组**。

数据查询

(连接查询)

连接查询

❖ 不像关系代数中“连接”是用一个特殊符号来表达的，在SQL中“连接”是用“连接条件”来表达的。

❖ 连接条件或连接谓词：用来连接两个表的条件 一般格式：

■ [**<表名1>.**]**<列名1>** **<比较运算符>** [**<表名2>.**]**<列名2>**

❖ 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（续）



◆1.等值与非等值连接查询

◆2.自身连接

◆3.外连接

◆4.多表连接

等值与非等值连接查询

❖ 等值连接：连接运算符为 “=”

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;
```

查询结果:

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

等值与非等值连接查询（续）

❖ 自然连接

❖ 采用在**SELECT**中去掉重复字段的方式实施



对上例用自然连接完成。

SELECT

Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade FROM

Student,SC

WHERE Student.Sno = SC.Sno;

等值与非等值连接查询（续）

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 Student.Sno=SC.Sno
```

连接谓词

```
AND SC.Cno='2'
```

```
AND SC.Grade>90;
```

选择谓词

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。

连接查询（续）



1.等值与非等值连接查询

2.自身连接

3.外连接

4.多表连接



2. 自身连接

- ❖ 自身连接：一个表与其自己进行连接，是一种特殊的连接
- ❖ 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

查询每一门课的直接先修课的名称

```
SELECT FIRST.Cname , SECOND.Cname
```

```
FROM Course FIRST, Course SECOND
```

```
WHERE FIRST.Cpno = SECOND.Cno;
```

自身连接（续）

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	数据处理导论		2
7	C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	数据处理导论		2
7	C语言	6	4

自身连接（续）

查询结果：

First.Cname	Second.Cname
数据库	数据结构
信息系统	数据库
操作系统	数据结构
数据结构	C语言
C语言	数据处理导论

连接查询（续）

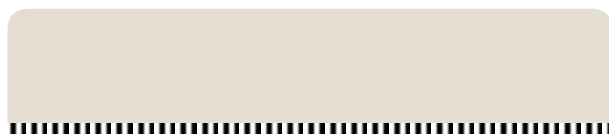
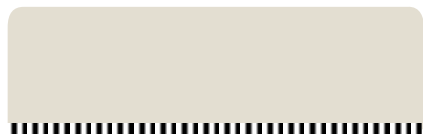


◆1.等值与非等值连接查询

◆2.自身连接

◆3.外连接

◆4.多表连接



3. 外连接

❖ 外连接与普通连接的区别

- 普通连接操作只输出满足连接条件的元组
- 外连接操作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
 - 列出左边关系中所有的元组
- 右外连接
 - 列出右边关系中所有的元组

外连接（续）

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

外连接（续）

执行结果：

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

连接查询（续）



◆1.等值与非等值连接查询

◆2.自身连接

◆3.外连接

◆4.多表连接

4. 多表连接

❖ 多表连接：两个以上的表进行连接

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;
```

数据查询

(嵌套查询)

嵌套查询

❖ 嵌套查询概述

■ 一个**SELECT-FROM-WHERE**语句称为一个**查询块**

■ 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 FROM /*外层查询/父查询*/

Student WHERE Sno IN

(SELECT Sno FROM SC

WHERE Cno= ' 2 ');

/*内层查询/子查询*/

嵌套查询（续）

- 上层的查询块称为外层查询或父查询
- 下层查询块称为内层查询或子查询
- SQL语言允许多层嵌套查询
- 子查询的限制
 - 不能使用ORDER BY子句

嵌套查询求解方法

- ❖ 不相关子查询：子查询的查询条不依赖于父查询
 - 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。

嵌套查询求解方法（续）

❖ 相关子查询：子查询的查询条件依赖于父查询

■ 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



◆1.带有**IN**谓词的子查询

◆2.带有比较运算符的子查询

◆3.带有**ANY**（**SOME**）或**ALL**谓词的子查询

◆4.带有**EXISTS**谓词的子查询



带有**IN**谓词的子查询

查询与“刘晨”在同一个系学习的学生。 此查可以分步完成

① 确定“刘晨”所在系名

```
SELECT Sdept FROM Student
```

```
WHERE Sname= '刘晨 ';
```

结果为: **CS**

② 查找所有在**CS**系学习的学生。

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

结果为:

Sno	Sname	Sdept
201215121	李勇	CS
201215122	刘晨	CS

带有**IN**谓词的子查询（续）

将第一步查询嵌入到第二步查询的条件中

```
SELECT Sno, Sname, Sdept FROM
```

```
Student
```

```
WHERE Sdept    IN
```

```
(SELECT Sdept FROM Student
```

```
WHERE Sname= ' 刘晨 ');
```

此查询为不相关子查询。

带有**IN**谓词的子查询（续）

用自身连接完成查询要求

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

带有IN谓词的子查询（续）

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'
)
);
```

③ 最后在**Student**关系中
取出**Sno**和**Sname**

② 然后在**SC**关系找出
选修了**3**号课程的学生学
号

① 首先在**Course**关系找出
“信息系统”的课程号，为**3**

带有**IN**谓词的子查询（续）

用连接查询实现：



```
SELECT Sno,Sname
```

```
FROM Student,SC,Course
```

```
WHERE Student.Sno = SC.Sno AND
```

```
SC.Cno = Course.Cno AND
```

```
Course.Cname='信息系统';
```


- 
- ◆ 1. 带有**IN**谓词的子查询
 - ◆ 2. 带有比较运算符的子查询
 - ◆ 3. 带有**ANY** (**SOME**) 或**ALL**谓词的子查询
 - ◆ 4. 带有**EXISTS**谓词的子查询
- 

2. 带有比较运算符的子查询

- ❖ 当能确切知道内层查询返回单值时，可用比较运算符（>，<，=，>=，<=，!=或<>）。

```
SELECT Sno,Sname,Sdept FROM Student
WHERE Sdept =      /*由于一个学生只可能在一个系学习，用=代替*/
  (SELECT Sdept
   FROM Student
   WHERE Sname= '刘晨');
```

带有比较运算符的子查询（续）

找出每个学生超过他选修课程平均成绩的课程号。

相关子查询

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

带有比较运算符的子查询（续）

◆❖可能的执行过程

◆■从外层查询中取出**SC**的一个元组**x**，将元组**x**的**Sno**值

◆（**201215121**）传送给内层查询。

◆**SELECT AVG(Grade) FROM SC y**

◆**WHERE y.Sno='201215121' ;**

带有比较运算符的子查询（续）

❖ 可能的执行过程（续）

■ 执行内层查询，得到值**88**（近似值），用该值代替内层查询，得到外层查询：

```
SELECT Sno,Cno FROM SC x  
WHERE Grade >=88;
```

■ 执行该查询，得到
结果：

(201215121,1)

(201215121,3)

带有比较运算符的子查询（续）



❖ 然后外层查询取出下一个元组重复做上述①至③步骤，直到外层的**SC**元组全部处理完毕。

结果为：

(201215121,1)

(201215121,3)

(201215122,2)

- 
- ◆ 1. 带有**IN**谓词的子查询
 - ◆ 2. 带有比较运算符的子查询
 - ◆ 3. 带有**ANY**或**ALL**谓词的子查询
 - ◆ 4. 带有**EXISTS**谓词的子查询
- 

带有**ANY**（**SOME**）或**ALL**谓词的子查询（续）

使用**ANY**或**ALL**谓词时必须同时使用比较运算

语义为：

> 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
>= ANY	大于等于子查询结果中的某个
>= ALL	值 大于等于子查询结果中的 所有值



带有**ANY** (**SOME**) 或**ALL**谓词的子查询 (续)

查询非计算机科学系中比计算机科学系任意一个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 */
```

结果:

Sname	Sage
王敏	18
张立	19

执行过程:

(1) 首先处理子查询, 找出**CS**系中所有学生的年龄, 构成一个集合 (**20,19**)

(2) 处理父查询, 找所有不是**CS**系且年龄小于**20 或 19**的学生

带有**ANY** (**SOME**) 或**ALL**谓词的子查询 (续)

❖ 用聚集函数实现

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

带有**ANY**（**SOME**）或**ALL**谓词的子查询（续）

查询非计算机科学系中比计算机科学系**所有**学 生年龄都小的学生姓名及年龄。

方法一：用**ALL**谓词

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

带有**ANY**（**SOME**）或**ALL**谓词的子查询（续）

方法二：用聚集函数 **SELECT**

Sname,Sage FROM Student

WHERE Sage <

(SELECT MIN(Sage)



FROM Student WHERE Sdept= ' CS ')

AND Sdept <>' CS ';

带有**ANY**（**SOME**）或**ALL**谓词的子查询（续）

ANY（或**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

- 
- ◆ 1. 带有 **IN** 谓词的子查询
 - ◆ 2. 带有比较运算符的子查询
 - ◆ 3. 带有 **ANY** (**SOME**) 或 **ALL** 谓词的子查询
 - ◆ 4. 带有 **EXISTS** 谓词的子查询
- 

带有**EXISTS**谓词的子查询

❖ **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。

带有**EXISTS**谓词的子查询（续）

❖ **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

带有**EXISTS**谓词的子查询（续）

查询所有选修了1号课程的学生姓名。

思路分析：

- 本查询涉及**Student**和**SC**关系
- 在**Student**中依次取每个元组的**Sno**值，用此值去检查**SC**表
- 若**SC**中存在这样的元组，其**Sno**值等于此**Student.Sno**值, 并且其**Cno**= '1'，则取此**Student.Sname**送入结果表

带有**EXISTS**谓词的子查询（续）

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

带有**EXISTS**谓词的子查询（续）

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 Sno = Student.Sno AND Cno='1');
```

带有**EXISTS**谓词的子查询（续）

❖ 不同形式的查询间的替换

- 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

- 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：
$$(\forall x) P \equiv \neg (\exists x (\neg P))$$

数据查询

(集合查询)

集合查询

❖ 集合操作的种类

- 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

集合查询（续）

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

集合查询（续）

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 ';
```

集合查询（续）

◆ 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
```

集合查询（续）

实际上就是查询计算机科学系中年龄不大于**19**岁的学生。

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

集合查询（续）

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';
```

集合查询（续）

也可以表示为：

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



集合查询（续）

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;
```

集合查询（续）

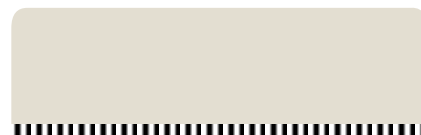
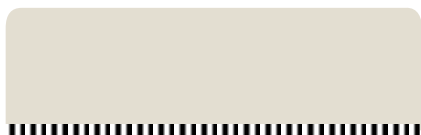
实际上是查询计算机科学系中年龄大于**19**岁的学生

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





数据更新





插入数据

修改数据

删除数据





❖ 两种插入数据方式

- 插入元组

- 插入子查询结果

- 可以一次插入多个元组



1. 插入元组

❖ 语句格式

INSERT

INTO <表名> [(<属性列1>[,<属性列2 >...])]

VALUES (<常量1> [,<常量2>]...);

❖ 功能

- 将新元组插入指定表中

插入元组（续）

❖ INTO子句

- 指定要插入数据的表名及属性列
- 属性列的顺序可与表定义中的顺序不一致
- 没有指定属性列：表示要插入的是一条完整的元组，且属性列属性与表定义中的顺序一致
- 指定部分属性列：插入的元组在其余属性列上取空值

插入元组（续）

❖ **VALUES**子句

■提供的值必须与**INTO**子句匹配

- 值的个数
- 值的类型

插入元组（续）

将一个新学生元组（学号：**201215128**；姓名：陈冬；性别：男；所在系：**IS**；年龄：**18**岁）插入到**Student**表中。

INSERT

INTO Student (Sno,Sname,Ssex,Sdept,Sage) VALUES
('201215128','陈冬','男','IS',18);

插入元组（续）

插入一条选课记录（ '200215128','1 ' ）。

```
INSERT INTO SC(Sno,Cno)
```

```
VALUES ('201215128 ',' 1 ');
```

关系数据库管理系统将在新插入记录的**Grade**列上自动地赋空值。

或者：

```
INSERT INTO SC
```

```
VALUES (' 201215128 ',' 1 ',NULL);
```

插入元组（续）

将学生张成民的信息插入到**Student**表中。

```
INSERT INTO Student
```

```
VALUES ('201215126','张成民','男' ,18,'CS');
```


2. 插入子查询结果

❖ 语句格式

INSERT INTO <表名> [(<属性列1> [,<属性列2>...)
子查询;

■ INTO子句

■ 子查询

● SELECT子句目标列必须与INTO子句匹配

- 值的个数
- 值的类型

插入子查询结果（续）

对每一个系，求学生的平均年龄，并把结果存入数据库

第一步：建表

```
CREATE TABLE Dept_age
( Sdept CHAR(15)          /*系名*/
  Avg_age SMALLINT);      /*学生平均年龄*/
```

第二步：插入数据

```
INSERT INTO Dept_age(Sdept,Avg_age)
  SELECT Sdept, AVG(Sage)
FROM Student
GROUP BY Sdept;
```

插入子查询结果（续）

❖ 关系数据库管理系统在执行插入语句时会检查所插元组是否破坏表上已定义的完整性规则

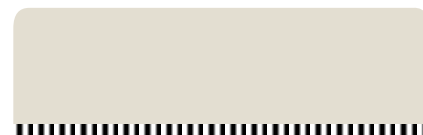
- 实体完整性
- 参照完整性
- 用户定义的完整性
 - **NOT NULL**约束
 - **UNIQUE**约束
 - 值域约束



插入数据

修改数据

删除数据



修改数据

❖ 语句格式

UPDATE <表名>

SET <列名>=<表达式>[,<列名>=<表达式>]...

[WHERE <条件>];

❖ 功能

- 修改指定表中满足**WHERE**子句条件的元组
- **SET**子句给出<表达式>的值用于取代相应的属性列
- 如果省略**WHERE**子句，表示要修改表中的所有元组

修改数据（续）

❖ 三种修改方式

- 修改某一个元组的值
- 修改多个元组的值
- 带子查询的修改语句

修改某一个元组的值

将学生**201215121**的年龄改为**22**岁

```
UPDATE Student SET Sage=22
```

```
WHERE Sno=' 201215121 ';
```

修改多个元组的值

将所有学生的年龄增加1岁。

UPDATE Student

SET Sage= Sage+1;

带子查询的修改语句

将计算机科学系全体学生的成绩置零。

UPDATE SC

SET Grade=0 WHERE Sno IN

(SELETE Sno

FROM Student WHERE Sdept= 'CS');

修改数据（续）

❖ 关系数据库管理系统在执行修改语句时会检查修改操作是否破坏表上已定义的完整性规则

- 实体完整性
- 主码不允许修改
- 用户定义的完整性
 - **NOT NULL**约束
 - **UNIQUE**约束
 - 值域约束

删除数据

❖ 语句格式

DELETE FROM <表名>
[WHERE <条件>];

❖ 功能

- 删除指定表中满足**WHERE**子句条件的元组

❖ **WHERE**子句

- 指定要删除的元组
- 无该子句将会删除表中的全部元组

删除数据（续）

❖ 三种删除方式

- 删除某一个元组的值
- 删除多个元组的值
- 带子查询的删除语句

删除某一个元组的值

删除学号为**201215128**的学生记录。

```
DELETEFROM Student WHERE Sno= 201215128 ;
```

2. 删除多个元组的值

删除所有的学生选课记录。

DELETE FROM SC;

3. 带子查询的删除语句

删除计算机科学系所有学生的选课记录。

DELETE FROM SC

WHERE Sno IN

(SELETE Sno

FROM Student WHERE Sdept= 'CS') ;

视图 (view)



❖ 视图的特点

- 虚表，是从一个或几个基本表（或视图）导出的表
- 只存放视图的定义，不存放视图对应的数据
- 基表中的数据发生变化，从视图中查询出的数据也随之改变

建立视图

❖ 语句格式

CREATE **VIEW**

<视图名> [(<列名> [,<列名>]...)]

AS <子查询>

[WITH CHECK OPTION];

建立视图（续）

❖ **WITH CHECK OPTION**

■对视图进行**UPDATE**，**INSERT**和**DELETE**操作时要保证更新、插入或删除的行满足视图定义中的谓词条件

（即子查询中的条件表达式）

❖子查询可以是任意的**SELECT**语句，是否可以含有**ORDER BY**子句和**DISTINCT**短语，则决定具体系统的实现。

建立视图（续）

- ◆ 组成视图的属性列名：全部省略或全部指定
- ◆ 全部省略：
 - ◆ 由子查询中**SELECT**目标列中的诸字段组成
- ◆ 明确指定视图的所有列名：
 - ◆ 某个目标列是聚集函数或列表达式
 - ◆ 多表连接时选出了几个同名列作为视图的字段
 - ◆ 需要在视图中为某个列启用新的更合适的名字

建立视图（续）

- ❖ 关系数据库管理系统执行**CREATE VIEW**语句时
只是把视图定义存入数据字典，并不执行其中的
SELECT语句。
- ❖ 在对视图查询时，按视图的定义从基本表中将数据查出。

建立视图（续）

建立信息系学生的视图。

```
CREATE VIEW IS_Student AS  
SELECT Sno,Sname,Sage  
FROM Student  
WHERE Sdept= 'IS';
```

建立视图（续）

建立信息系学生的视图，并要求进行修改和插入 操作时
仍需保证该视图只有信息系的学生 。

```
CREATE VIEW IS_Student AS  
SELECT Sno,Sname,Sage FROM Student  
WHERE      Sdept= 'IS'  WITH CHECK OPTION;
```

建立视图（续）

- ❖ 定义 **IS_Student** 视图时加上了 **WITH CHECK OPTION** 子句，对该视图进行插入、修改和删除操作时，**RDBMS** 会自动加上 **Sdept='IS'** 的条件。
- ❖ 若一个视图是从单个基本表导出的，并且只是去掉了基本表的某些行和某些列，但保留了主码，我们称这类视图为 **行列子集视图**。
- **IS_Student** 视图就是一个行列子集视图。

建立视图（续）

❖ 基于多个表的视图

建立信息系选修了1号课程的学生视图（包括 学号、姓名、成绩）。

```
CREATE VIEW IS_S1(Sno,Sname,Grade) AS  
SELECT Student.Sno,Sname,Grade FROM Student,SC  
WHERE Sdept= 'IS' AND  
Student.Sno=SC.Sno AND SC.Cno= '1';
```

建立视图（续）

❖ 基于视图的视图

建立信息系选修了1号课程且成绩在90分以上的 学生的视图。

```
CREATE VIEW IS_S2 AS  
SELECT Sno,Sname,Grade FROM IS_S1  
WHERE Grade>=90;
```

建立视图（续）

❖带表达式的视图

定义一个反映学生出生年份的视图。

```
CREATE VIEW BT_S(Sno,Sname,Sbirth) AS  
SELECT Sno,Sname,2014-Sage FROM Student;
```

建立视图（续）

❖ 分组视图

将学生的学号及平均成绩定义为一个视图

```
CREAT VIEW S_G(Sno, Gavg)
```

```
AS
```

```
SELECT Sno, AVG(Grade) FROM SC
```

```
GROUP BY Sno;
```

2. 删除视图

❖ 语句的格式:

DROP VIEW <视图名>[CASCADE];

- 该语句从数据字典中删除指定的视图定义
- 如果该视图上还导出了其他视图，使用**CASCADE**级联删除语句，把该视图和由它导出的所有视图一起删除
- 删除基表时，由该基表导出的所有视图定义都必须显式地使用**DROP VIEW**语句删除

删除视图（续）

删除视图BT_S和IS_S1

```
DROP VIEW BT_S;    /*成功执行*/
```

```
DROP VIEW IS_S1;  /*拒绝执行*/
```

要删除IS_S1，需使用级联删除：

```
DROP VIEW IS_S1 CASCADE;
```

查询视图

❖ 用户角度：查询视图与查询基本表相同

❖ 关系数据库管理系统实现视图查询的方法

■ 视图消解法（**View Resolution**）

- 进行有效性检查
- 转换成等价的对基本表的查询
- 执行修正后的查询

查询视图（续）

在信息系学生的视图中找出年龄小于20岁的学生。

```
SELECT Sno,Sage  
FROM IS_Student  
WHERE Sage<20;
```

视图消解转换后的查询语句为：

```
SELECT Sno,Sage  
FROM Student  
WHERE Sdept= 'IS' AND Sage<20;
```



查询视图（续）

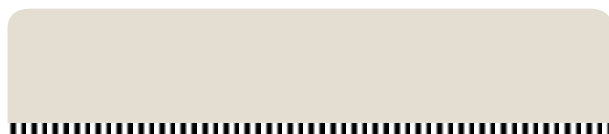
查询选修了1号课程的信息系学生

```
SELECT IS_Student.Sno,Sname FROM
```

```
IS_Student,SC
```

```
WHERE IS_Student.Sno =SC.Sno AND SC.Cno= '1';
```

- 
- ❖视图能够简化用户的操作
 - ❖视图使用户能以多种角度看待同一数据
 - ❖视图对重构数据库提供了一定程度的逻辑独立性
 - ❖视图能够对机密数据提供安全保护
 - ❖适当的利用视图可以更清晰的表达查询



数据控制

- ◆数据库系统可以由**DBMS**统一提供授权控制
 - ◆事务管理功能—事务，提交，回滚
 - ◆数据保护功能—完整性约束，安全控制功能
- ◆安全控制功能
 - ◆什么人对什么数据有什么样的访问能力—不是技术问题，是安全策略问题
 - ◆**DBMS**需要对保障安全策略的实施

DBMS需要具备的安全功能

- ◆授权决定的设置—SQL中的**GRANT** 和 **REVOKE**
- ◆授权结果保存在数据字典
- ◆当用户提出数据访问请求时候，根据数据字典决定是否执行操作请求

授权

◆Grant

```
GRANT <权限>[, <权限>]...  
    [ON <对象类型> <对象名>]  
    TO <用户>[, <用户>]...  
    [WITH GRANT OPTION];
```

对象和授权对照表

对 象	对 象 类 型	操 作 权 限
属性列	TABLE	SELECT, INSERT, UPDATE, DELETE , ALL PRIVILEGES
视 图	TABLE	SELECT, INSERT, UPDATE, DELETE , ALL PRIVILEGES
基本表	TABLE	SELECT, INSERT, UPDATE, DELETE , ALTER, INDEX, ALL PRIVILEGES
数据库	DATABASE	CREATETAB

收回授权

◆ Revoke

```
REVOKE <权限>[ , <权限>]...  
[ON <对象类型> <对象名>]  
FROM <用户>[ , <用户>]...;
```

- ◆ 授权收回会级联下去
- ◆ 用户对自己建立的表拥有全部权限，并且可以授权

嵌入式SQL

- ◆交互的一面向集合的一非过程性
- ◆嵌入式—将**SQL**语言嵌入到其他高级语言中
- ◆该高级语言称为主语言或者宿主语言
- ◆宿主形式和交互形式在细节上会有所不同—虽然语法形式相同。

嵌入式**SQL**的一般形式

- ◆预编译—**DBMS**对源程序进行预扫描—识别出**SQL**语句，转换成主语言的调用—主语言编译程序转换成目标码—这种形式用的比较多
- ◆扩充主语言
- ◆一般形式 **SQL**语句前加**EXEC SQL** 字头，后面跟 **END—EXEC** 或者主语言的分割符

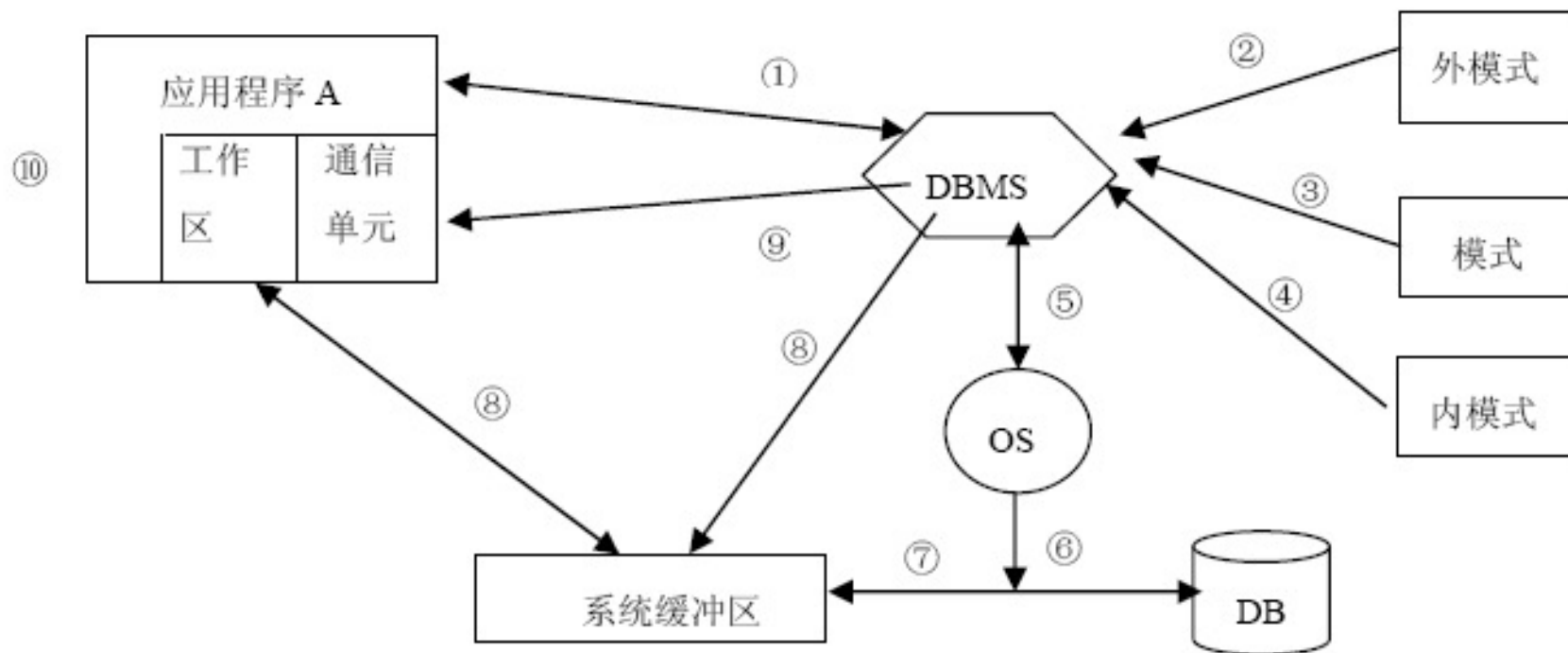
嵌入式**SQL**和主语言之间的通讯

- ◆**SQL**语言的执行状态信息，通过**SQL**通讯区 **SQLCA**实现，以便主语言根据执行情况控制流程
- ◆主语言向**SQL**提供参数，主要通过主变量实现
- ◆**SQL**语句查询结果通过主变量或者游标形式传递

SQL通讯区SQLCA

- ◆SQLCA实际是一个数据结构，通过 **exec SQL INCLUDE SQLCA**来定义
- ◆SQL语句执行后，执行情况 and 状态会保存在这个数据结构中
- ◆应用程序在执行结束后应该检测SQLCA中的状态以便决定下一步处理的流程

数据库系统的工作过程



主变量

- ◆ **SQL**语句中使用的主语言的变量称为主变量
- ◆ 输入主变量—应用程序赋值，**SQL**语句引用
- ◆ 输出主变量—**SQL**语句赋值，反馈给主程序
- ◆ 主变量可以附带一个指示变量用以标明主变量的状态。
- ◆ 主变量的声明必须在 **BEGIN DECLARE SECTION** 和 **END DECLARE SECTION** 之间，在**SQL**语句中需要前缀：以区别数据库对象名称，指示变量也需要前面加前缀(:)并且在主变量之后
- ◆ 在主语言中，主变量可以随意引用，不必加:

游标

- ◆ **SQL**语句是面向集合的
- ◆ 主语言是面向记录的
- ◆ 游标是系统开设的一个缓冲区，存储**SQL**语句的执行结果，以便主语言顺次访问记录
- ◆ 每个游标都有一个名字

不使用游标的**SQL**语句

- ◆说明语句
- ◆数据定义语句
- ◆数据控制语句
- ◆查询结果为单值的**select**语句（**into**子句扩充）
- ◆**INSERT**语句
- ◆非**CURRENT**形式的**UPDATE**和**DELETE**语句

使用游标的**SQL**语句

- ◆ 查询结果为多条记录的**select**语句
- ◆ **Current**形式的**update**和**delete**

使用游标的过程

◆说明游标

◆ EXEC SQL DECLARE <游标名> CURSOR FOR <SELECT 语句>;

◆打开游标

EXEC SQL OPEN <游标名>;

◆推进游标指针并返回当前记录值

EXEC SQL FETCH <游标名>

INTO <主变量>[<指示变量>][, <主变量>[<指示变量>]]...;

◆关闭游标 EXEC SQL CLOSE <游标名>;

Current形式的update和delete

- ◆声明游标 在select语句中使用子句
 - ◆For update of<列名>
- ◆Open打开游标
- ◆Fetch移动游标并返回值到主变量
- ◆检查当前记录并执行相应操作 使用
 - ◆Where current of <游标名>
- ◆关闭游标

嵌入式**SQL**例子

动态SQL简介

- ◆ 静态SQL语句—内嵌SQL语句的变量个数和数据类型在预编译之前是确定的
- ◆ 为了使内嵌SQL语句更加灵活，有些DBMS支持动态的SQL技术
 - ◆ SQL语句正文
 - ◆ 主变量的个数
 - ◆ 主变量的数据类型
 - ◆ SQL语句引用的数据库对象

小节

- ◆ 关系数据库的基本概念和SQL
- ◆ 在SQL中的
 - ◆ 数据定义
 - ◆ 数据查询
 - ◆ 数据更新
 - ◆ 数据控制
- ◆ 视图的概念
- ◆ SQL语言的使用方式
 - ◆ 交互
 - ◆ 内嵌