厦门大学计算机科学系本科生课程

《数据库系统原理》

扫一扫访问班级网站
支持手机浏览

第3章 关系数据库标准语言SQL（2016版）

林子雨

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：http://www.cs.xmu.edu.cn/linziyu

提纲

- 3.1 SQL概述
- 3.2 学生-课程数据库
- 3.3 数据定义
- 3.4 数据查询
- 3.5 数据更新
- 3.6 视图

3.1 SQL概述

- **3.1.1 SQL的产生与发展**
- **3.1.2 SQL的特点**
- **3.1.3 SQL的基本概念**

3.1.1 SQL的产生与发展

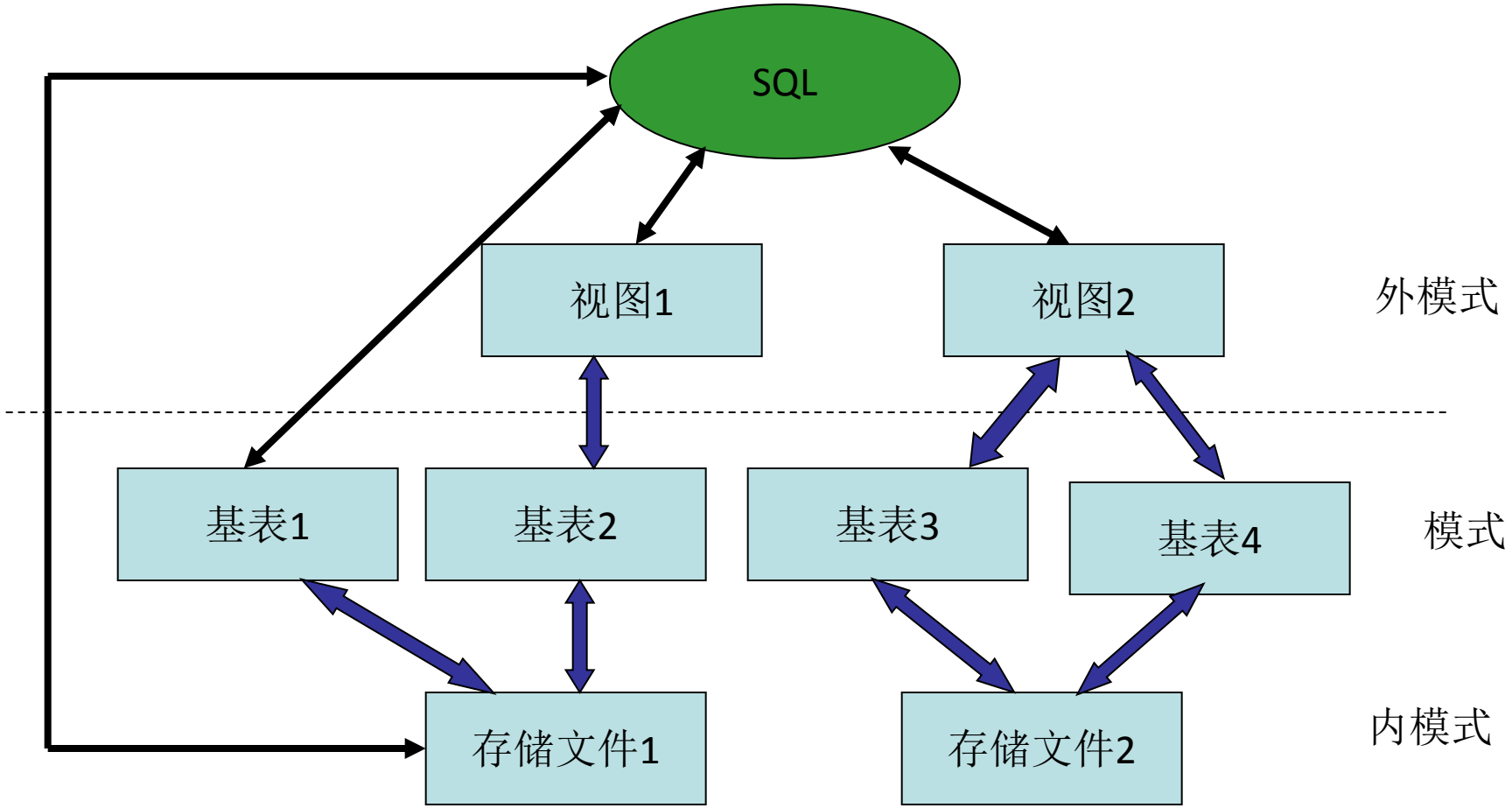
- 1974年，由Boyce和Chamberlin提出，并在IBM公司研制的关系数据库管理系统原型System R 上实现
- 1986年10月，美国国家标准局的数据库委员会X3H2批准了SQL作为关系数据库语言的美国标准，并公布了SQL标准文本
- 1987年，国际标准化组织通过这一标准

3.1.2 SQL的特点

SQL的特点

- 1. 综合统一
- 2. 高度非过程化
- 3. 面向集合的操作方式
- 4. 以同一种语法结构提供两种使用方法
- 5. 语言简洁，易学易用

1、综合统一（操纵三级模式）



5、语言简捷，易学易用

表 3.1 SQL 语言的动词

SQL 功 能	动 词
数 据 定 义	CREATE， DROP， ALTER
数 据 查 询	SELECT
数 据 操 纵	INSERT， UPDATE， DELETE
数 据 控 制	GRANT， REVOKE

第3章 关系数据库标准语言SQL

- 3.1 SQL概述
- 3.2 学生-课程数据库
- 3.3 数据定义
- 3.4 数据查询
- 3.5 数据更新
- 3.6 视图

学生-课程数据库

- 学生表：**Student(Sno，Sname，Ssex，Sage，Sdept)**
- 课程表：**Course(Cno，Cname，Cpno，Ccredit)**
- 学生选课表：**SC(Sno，Cno，Grade)**

3.2 学生-课程数据库

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

Student

(a)

3.2 学生-课程数据库

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

Course

(b)

3.2 学生-课程数据库

学 号	课 程 号	成 绩
Sno	Cno	Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

SC

(c)

提纲

- 3.1 SQL概述
- 3.2 学生-课程数据库
- 3.3 数据定义
- 3.4 数据查询
- 3.5 数据更新
- 3.6 视图

3.3 数据定义

- **3.3.1 模式的定义与删除**
- **3.3.2 基本表的定义、删除与修改**
- **3.3.3 索引的建立与删除**

3.3 数据定义

表 3.2　SQL 的数据定义语句

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	

3.3 数据定义

- **3.3.1 模式的定义与删除**
- **3.3.2 基本表的定义、删除与修改**
- **3.3.3 索引的建立与删除**

3.3.1 模式的定义与删除

一、定义模式

CREATE SCHEMA <模式名> AUTHORIZATION <用户名>

Ps:

1、若没有指定<模式名>，那么<模式名>隐含为<用户名>

2、要创建模式，调用该命令的用户必须具有DBA权限，或者获得了DBA授予的CREATE SCHEMA的权限

例题

[例1] 定义一个学生-课程模式S-T

CREATE SCHEMA “S-T” AUTHORIZATION WANG;

为用户“WANG”定义了一个模式S-T

3.3.1 模式的定义与删除

二、删除模式

DROP SCHEMA <模式名> <CASCADE | RESTRICT>

其中CASCADE和RESTRICT两者必选其一

1、CASCADE（级联），表示在删除模式同时把该模式中所有的数据库对象全部一起删除

2、RESTRICT（限制），表示若该模式下已定义了下属的数据库对象，则拒绝删除

例题

[例4] 删除模式ZHANG

DROP SCHEMA ZHANG CASCADE

3.3 数据定义

- **3.3.1 模式的定义与删除**
- **3.3.2 基本表的定义、删除与修改**
- **3.3.3 索引的建立与删除**

3.3.2 基本表的定义、删除与修改

一、定义基本表

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>] ）；

<表名>：所要定义的基本表的名字

- <列名>：组成该表的各个属性（列）
- <列级完整性约束条件>：涉及相应属性列的完整性约束条件
- <表级完整性约束条件>：涉及一个或多个属性列的完整性约束条件

例题

[例1] 建立一个“学生”表Student，它由学号Sno、姓名Sname、性别Ssex、年龄Sage、所在系Sdept五个属性组成。其中学号是主键，并且姓名取值也唯一。

```
CREATE TABLE Student
      (Sno      CHAR(5)  primary key,
       Sname  CHAR(8)  UNIQUE,
       Ssex      CHAR(2) ,
       Sage      INT,
       Sdept    CHAR(10));
```

例题 （续）

Sno	Sname	Ssex	Sage	Sdept

↑ ↑ ↑ ↑ ↑

字符型 字符型 字符型 整数 字符型

长度为 5 长度为 8 长度为 2 长度为 10

不能为空值

定义基本表（续）

- 常用完整性约束
  - 主码约束：　**PRIMARY KEY**
  - 唯一性约束：**UNIQUE**
  - 非空值约束：**NOT NULL**
  - 参照完整性约束

PRIMARY KEY与 UNIQUE的区别？

例题 （续）

- **[例2] 建立一个“学生选课”表SC，它由学号Sno、课程号Cno，修课成绩Grade组成，其中(Sno, Cno)为主码。**

```
CREATE TABLE SC(
        Sno CHAR(5) ,
        Cno CHAR(3) ,
        Grade   int,
        Primary key (Sno, Cno));
```

二、修改基本表

ALTER TABLE <表名>

[ADD <新列名> <数据类型> [完整性约束]]
[DROP <完整性约束名>]
[ALTER COLUMN <列名> <数据类型>]
[DROP COLUMN <列名> <数据类型>];

- <表名>：要修改的基本表
- ADD子句：增加新列和新的完整性约束条件
- DROP子句：删除指定的完整性约束条件
- ALTER COLUMN子句：用于修改列名和数据类型
- DROP COLUMN子句：用于删除列

例题

[例2]　向Student表增加“入学时间”列，其数据类型为日期型。

ALTER TABLE Student ADD Scome DATETIME;

- 不论基本表中原来是否已有数据，新增加的列一律为空值。

例题

[例3] 将入学日期的数据类型改为字符型。

```
ALTER TABLE Student ALTER COLUMN Scome
char(8);
```

- 注：修改原有的列定义有可能会破坏已有数据

语句格式（续）

- 删除属性列

[例4] 将入学日期列删除掉。

```
ALTER TABLE Student DROP COLUMN Scome
```

三、删除基本表

DROP TABLE <表名>;

基本表删除

数据、表上的索引 自动都删除

例题

[例5] 删除SC表

```
DROP TABLE SC;
```

3.3 数据定义

- **3.3.1 模式的定义与删除**
- **3.3.2 基本表的定义、删除与修改**
- **3.3.3 索引的建立与删除**

3.3.3 索引的建立与删除

- 建立索引是加快查询速度的有效手段
- 建立索引
  - DBA或表的属主（即建立表的人）根据需要建立
  - 有些DBMS自动建立以下列上的索引
    - PRIMARY KEY
    - UNIQUE
- 维护索引
  - DBMS自动完成
- 使用索引
  - DBMS自动选择是否使用索引以及使用哪些索引

一、建立索引

- 语句格式

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
ON <表名>(<列名>[<次序>][,<列名>[<次序>]]…);

- 用<表名>指定要建索引的基本表名字
- 索引可以建立在该表的一列或多列上，各列名之间用逗号分隔
- 用<次序>指定索引值的排列次序，升序：ASC，降序：DESC。缺省值：ASC
- UNIQUE表明此索引的每一个索引值只对应唯一的数据记录
- CLUSTER表示要建立的索引是聚簇索引

建立索引（续）

- 唯一值索引
  - 对于已含重复值的属性列不能建**UNIQUE**索引
  - 对某个列建立**UNIQUE**索引后，插入新记录时**DBMS**会自动检查新记录在该列上是否取了重复值。这相当于增加了一个**UNIQUE**约束

建立索引（续）

- 聚簇索引
  - 建立聚簇索引后，基表中数据也需要按指定的聚簇属性值的升序或降序存放。也即聚簇索引的索引项顺序与表中记录的物理顺序一致

例：

CREATE CLUSTER INDEX Stusname ON Student(Sname)；

在**Student**表的**Sname**（姓名）列上建立一个聚簇索引，而且**Student**表中的记录将按照**Sname**值的升序存放

建立索引（续）

- 在一个基本表上最多只能建立一个聚簇索引
- 聚簇索引的用途：对于某些类型的查询，可以提高查询效率
- 聚簇索引的适用范围
  - 很少对基表进行增删操作
  - 很少对其中的变长列进行修改操作
- 聚簇索引的不适用情形
  - 表记录太少
  - 经常插入、删除、修改的表
  - 数据分布平均的表字段

建立索引（续）

在下列三种情况下，有必要建立簇索引：

(1)查询语句中采用该字段作为排序列

(2)需要返回局部范围的大量数据

(3)表格中某字段内容的重复性比较大例如，student表中dno(系号)一列有大量重复数据，当在dno列上建立了簇索引后，下面的连接查询速度会加快。

建立索引（续）

•多列索引和多个单列索引

考虑两种不同的建立索引方式：

case 1:对c1,c2,c3三列按此顺序添加一个多列索引;

case 2: 对c1,c2,c3分别建立三个单列索引;

问题1:按c1搜索时，哪种索引效率快?

答：case2

问题2:按C2搜索时，哪种索引效率快?

答：case2,并且，case1的索引无效

问题3：按C1,C2,C3搜索哪种效率快?

答：case1

问题4：按C2,C3,C1搜索时哪种效率快?

答：case2,因为没有按多列索引的顺序搜索，case1的索引没有使用到。

覆盖查询简单的说就是所有查询列被所使用的索引覆盖的查询

建立索引（续）

- 如何去建立一个多列索引，最重要的一个问题是如何安排列的顺序是至关重要的
- 比如需要对一个表tb里面的两个字段foo,bar建一个索引，那么索引的顺序是（foo,bar)还是(bar,foo)呢
- 假设tb表有1700条记录，foo字段有750个不同的记录，那么foo字段上的cardinality是750。总规则可以说是cardinality越大的字段应该排在索引的第一位就是说索引的位置是(foo,bar)，因为cardinality越大那么第一次取出来的记录集就越小，再进行第二次查询的次数就越少了。

建立索引（续）

•当在同一表格中建立簇索引和非簇索引时，先建立簇索引后建非簇索引比较好。因为如先建非簇索引的话，当建立簇索引时，SQL Server会自动将非簇索引删除，然后重新建立非簇索引。每个表仅可以有一个簇索引，最多可以有249个非簇索引。它们均允许以一个或多个字段作为索引关键字(Index Key)，但最多只能有16个字段。

•SQL Server只对那些能加快数据查询速度的索引才能被选用。如果利用索引检索还不如顺序扫描速度快，SQL Server仍用扫描方法检索数据。建立不能被采用的索引只会增加系统的负担，降低检索速度。因此，可利用性是建立索引的首要条件。

例题

[例6] 为学生-课程数据库中的**Student，Course，SC**三个表建立索引。其中**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);
```

二、删除索引

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。

[例7] 删除Student表的Stusname索引。

```
DROP INDEX Student.Stusname;
```

第3章 关系数据库标准语言SQL

- 3.1 SQL概述
- 3.2 学生-课程数据库
- 3.3 数据定义
- 3.4 数据查询
- 3.5 数据更新
- 3.6 视图

3.4 查　询

- 3.4.1 单表查询
- 3.4.2 连接查询
- 3.4.3 嵌套查询
- 3.4.4 集合查询
- 3.4.5 SELECT 语句的一般格式

3.4 查　询

- 语句格式

SELECT [ALL|DISTINCT] <目标列表达式>
[，<目标列表达式>] ...
FROM <表名或视图名>[，<表名或视图名>] ...
[WHERE <条件表达式>]
[GROUP BY <列名1> [HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC|DESC]];

3.4 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集函数。
- HAVING短语：筛选出只有满足指定条件的组
- ORDER BY子句：对查询结果表按指定列值的升序或降序排序

3.4 查　询

- 3.4.1 单表查询
- 3.4.2 连接查询
- 3.4.3 嵌套查询
- 3.4.4 集合查询
- 3.4.5 SELECT 语句的一般格式

3.4.1 单表查询

查询仅涉及一个表，是一种最简单的查询操作

一、选择表中的若干列

二、选择表中的若干元组

三、对查询结果排序

四、使用集函数

五、对查询结果分组

查询指定列

[例1] 查询全体学生的学号与姓名。

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

[例2] 查询全体学生的姓名、学号、所在系。

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

查询全部列

[例3] 查询全体学生的详细记录。

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

或

```
SELECT *
FROM Student;
```

3. 查询经过计算的值

SELECT子句的<目标列表达式>为表达式

- 算术表达式
- 字符串常量
- 函数
- 列别名
- 等

3. 查询经过计算的值

[例4] 查全体学生的姓名及其出生年份。

```
SELECT Sname,2011-Sage
FROM Student;
```

3. 查询经过计算的值

[例5] 查询全体学生的姓名、出生年份和所在系。在出生年份前面增加一个说明，在系名称后面增加一个“系”作为表示

```
SELECT Sname, '出生年份:', 2011-Sage,
Sdept + '系'
FROM Student;
```

[例5.1] 使用列别名改变查询结果的列标题

```
SELECT Sname '姓名', 'Year of Birth: '
   '生日标识',
2011-Sage  '生日',  Sdept+'系'  '系名'
FROM Student;
```

二、选择表中的若干元组

- 消除取值重复的行
- 查询满足条件的元组

1. 消除取值重复的行

–在SELECT子句中使用DISTINCT短语

假设SC表中有下列数据

Sno	Cno	Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

ALL 与 DISTINCT

[例6] 查询选修了课程的学生学号。

(1) SELECT Sno
FROM SC;

或(默认 ALL)

SELECT ALL Sno
FROM SC;

(2) SELECT DISTINCT Sno
FROM SC;

例题（续）

- 注意 **DISTINCT**短语的作用范围是所有目标列

  例：查询选修课程的各种成绩

  错误的写法

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

  正确的写法

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

2.查询满足条件的元组

WHERE子句常用的查询条件

表3.3 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	AND，OR

(1) 比较大小

在WHERE子句的<比较条件>中使用比较运算符

- =，>，<，>=，<=，!= 或 <>，!>，!<，
- 逻辑运算符NOT + 比较运算符

[例8] 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;
```

或

```
SELECT Sname，Sage
    FROM   Student
    WHERE NOT Sage >= 20；
```

(2) 确定范围

- 使用谓词 **BETWEEN … AND …**

  NOT BETWEEN … AND …

[例10] 查询年龄在20~23岁（包括20岁和23岁）之间的学生的姓名、系别和年龄。

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

例题（续）

[例11] 查询年龄不在20~23岁之间的学生姓名、系别和年龄。

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

(3) 确定集合

使用谓词　IN <值表>，NOT IN <值表>

<值表>：用逗号分隔的一组取值

[例12]查询信息系（IS）和计算机科学系（CS）学生的姓名和性别。

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

(3) 确定集合

[例13]查询既不是信息系又不是计算机科学系的学生的姓名和性别。

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

(4) 字符串匹配

- **[NOT] LIKE ‘<匹配串>’ [ESCAPE ‘ <换码字符>’]**

  <匹配串>：指定匹配模板

  匹配模板：固定字符串或含通配符的字符串

  当匹配模板为固定字符串时，

  可以用 = 运算符取代 LIKE 谓词

  用 != 或 < >运算符取代 NOT LIKE 谓词

通配符

- % (百分号) 代表任意长度（长度可以为0）的字符串
  - 例：a%b表示以a开头，以b结尾的任意长度的字符串。如acb，addgb，ab 等都满足该匹配串
- _ (下横线) 代表任意单个字符
  - 例：a_b表示以a开头，以b结尾的长度为3的任意字符串。如acb，afb等都满足该匹配串

ESCAPE 短语：

– 当用户要查询的字符串本身就含有 % 或 _ 时，要使用**ESCAPE '<换码字符>'** 短语对通配符进行转义。

例题

1) 匹配模板为固定字符串

[例14] 查询学号为95001的学生的详细情况。

```
SELECT *
    FROM  Student
    WHERE  Sno LIKE '95001';
```

等价于：

```
SELECT  *
FROM  Student
WHERE Sno = '95001';
```

例题（续）

2) 匹配模板为含通配符的字符串

[例15] 查询所有姓刘学生的姓名、学号和性别。

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

例题（续）

匹配模板为含通配符的字符串（续）

[例16] 查询姓“刘"且全名为三个汉字的学生的姓名。

```
SELECT Sname
    FROM   Student
    WHERE  Sname LIKE '刘__';
```

备注：这里有两个_

关于SQL Server 2008字符集

数据库字符集为ASCII时，一个汉字需要两个_；
当数据库字符集为GBK时，一个汉字只需要一个_

```
SELECT
COLLATIONPROPERTY('Chinese_PRC_Stroke_CI_AI_KS_WS',
'CodePage')
```

查询结果：
936 简体中文GBK
950 繁体中文BIG5
437 美国/加拿大英语
932 日文
949 韩文
866 俄文
65001 unicode UFT-8

例题（续）

匹配模板为含通配符的字符串（续）

[例17] 查询名字中第2个字为“敏”字的学生的姓名和学号。

```
SELECT Sname，Sno
    FROM Student
    WHERE Sname LIKE '_敏%';
```

例题（续）

[例18] 查询所有不姓刘的学生姓名。

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

例题（续）

3) 使用换码字符将通配符转义为普通字符

[例19] 查询课程名称中包含“面向对象_C++课程”的课程号和学分。

```
SELECT Cno，Ccredit
    FROM Course
    WHERE Cname LIKE '%面向对象_C++%'
```

例题（续）

```
SELECT Cno，Ccredit
    FROM Course
    WHERE Cname LIKE '%面向对象\ _C++ %'
                ESCAPE '\'
```

例题（续）

使用换码字符将通配符转义为普通字符(续)

[例20] 查询以"DB_"开头，且倒数第3个字符为 i的课程的详细情况。

```
SELECT *
    FROM   Course
    WHERE  Cname LIKE  'DB\_%i_ _' ESCAPE ' \ ';
```

(5) 涉及空值的查询

- 使用谓词 **IS NULL** 或 **IS NOT NULL**
- **“IS NULL”** 不能用 **“= NULL”** 代替

[例21] 有些课没有先修课程。查询没有先修课程的课程名称。

```
Select Cname
From Course
Where Cpno IS NULL
```

(6) 多重条件查询

用逻辑运算符**AND**和 **OR**来联结多个查询条件

- **AND**的优先级高于**OR**
- 可以用括号改变优先级

可用来实现多种其他谓词

- **[NOT] IN**
- **[NOT] BETWEEN … AND …**

例题

[例23] 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;
```

改写[例12]

[例12] 查询信息系（**IS**）、数学系（**MA**）和计算机科学系（**CS**）学生的姓名和性别。

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

可改写为：

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

改写[例10]

[例10] 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；
```

可改写为：

```
SELECT Sname，Sdept，Sage
FROM Student
WHERE  Sage>=20 AND Sage<=23；
```

三、对查询结果排序

使用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

当排序列含空值时

空值的显示顺序由具体系统实现来决定

例如按升序排，含空值的元组最后显示

按降序排，空值的元组最先显示

各个系统的实现可以不同，只要保持一致即可

对查询结果排序（续）

[例24]　查询选修了3号课程的学生的学号及其成绩，查询结果按分数降序排列。

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

对查询结果排序（续）

[例25] 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;
```

四、使用集函数

5类主要聚集函数

- 计数

  COUNT（[DISTINCT|ALL] *）

  COUNT（[DISTINCT|ALL] <列名>）

- 计算总和

  SUM（[DISTINCT|ALL] <列名>）

- 计算平均值

  AVG（[DISTINCT|ALL] <列名>）

使用集函数（续）

求最大值

MAX（[DISTINCT|ALL] <列名>）

求最小值

MIN（[DISTINCT|ALL] <列名>）

– DISTINCT短语：在计算时要取消指定列中的重复值
– ALL短语：不取消重复值
– ALL为缺省值

使用集函数（续）

[例26] 查询学生总人数。

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

[例27] 查询选修了课程的学生人数。

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

注：用**DISTINCT**以避免重复计算学生人数

使用集函数（续）

[例28] 计算2号课程的学生平均成绩。

```
SELECT AVG(Grade)
  FROM SC
  WHERE Cno= ' 2 ';
```

[例29] 查询选修3号课程的学生最高分数。

```
SELECT MAX(Grade)
  FROM SC
  WHER Cno= ' 3';
```

五、对查询结果分组

使用GROUP BY子句分组

细化聚集函数的作用对象

- 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

使用GROUP BY子句分组

[例30] 求各个课程号及相应的选课人数。

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

使用GROUP BY子句分组

课堂作业

我们拥有下面这个 "Orders" 表：
现在，我们希望查找每个客户的总金额（总订单）。

O_Id	OrderDate	OrderPrice	Customer
1	2008/12/29	1000	Bush
2	2008/11/23	1600	Carter
3	2008/10/05	700	Bush
4	2008/09/28	300	Bush
5	2008/08/06	2000	Adams
6	2008/07/21	100	Carter

语句1：SELECT Customer, SUM(OrderPrice) FROM Orders

语句2：SELECT Customer,SUM(OrderPrice) FROM Orders GROUP BY Customer

使用GROUP BY子句分组

语句1：SELECT Customer, SUM(OrderPrice) FROM Orders

Customer	SUM(OrderPrice)
Bush	5700
Carter	5700
Bush	5700
Bush	5700
Adams	5700
Carter	5700

使用GROUP BY子句分组

语句2：SELECT Customer,SUM(OrderPrice) FROM Orders GROUP BY Customer

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

对查询结果分组 （续）

- **GROUP BY**子句的作用对象是查询的中间结果表
- 分组方法：按指定的一列或多列值分组，值相等的为一组；**group by** 是先排序后分组
- 使用**GROUP BY**子句后，**SELECT**子句的列名列表中只能出现分组属性和聚集函数，或者说，**select** 后面的所有列中,没有使用聚合函数的列,必须出现在 **group by** 后面
- 如果要用到**group by** 一般用到的就是 “每”这个字,例如：每个部门有多少人 就要用到分组的技术

对查询结果分组（续）

select Sno,Cno from SC group by Sno

--将会出现错误
--消息 8120，级别 16，状态 1，第 1 行
选择列表中的列 'SC.Cno' 无效，因为该列没有包含在聚合函数或 GROUP BY 子句中。

这就是我们需要注意的一点，如果在返回集字段中，这些字段要么就要包含在Group By语句的后面，作为分组的依据；要么就要被包含在聚合函数中。

select Sno as '学号',
COUNT(*) as '选课门数' from SC group by Sno

使用HAVING短语筛选最终输出结果

[例31] 查询选修了2门及以上课程的学生学号。

```
SELECT Sno
    FROM  SC
    GROUP BY Sno
    HAVING  COUNT(*) >2;
```

例题

[例32] 查询有2门及以上课程是85分以上的学生的学号及（85分以上的）课程数

```
SELECT  Sno,  COUNT(*)
        FROM   SC
        WHERE Grade>=85
        GROUP BY Sno
        HAVING COUNT(*)>=2;
```

使用HAVING短语筛选最终输出结果

- 只有满足**HAVING**短语指定条件的组才输出
- **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。

3.4 查　询

- 3.4.1 单表查询
- 3.4.2 连接查询
- 3.4.3 嵌套查询
- 3.4.4 集合查询
- 3.4.5 SELECT 语句的一般格式

3.4.2 连接查询

同时涉及多个表的查询称为连接查询

用来连接两个表的条件称为连接条件或连接谓词

一般格式：

- [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>

  比较运算符：=、>、<、>=、<=、!=

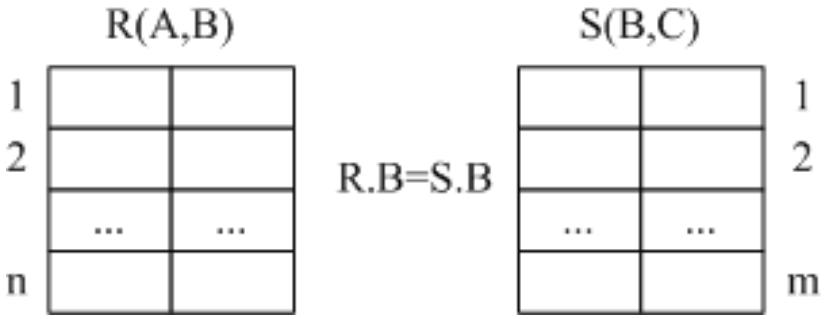
- [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

连接查询（续）

- 连接字段
  - 连接谓词中的列名称为连接字段
  - 连接条件中的各连接字段类型必须是可比的，但不必是相同的

连接操作的执行过程

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



嵌套循环连接

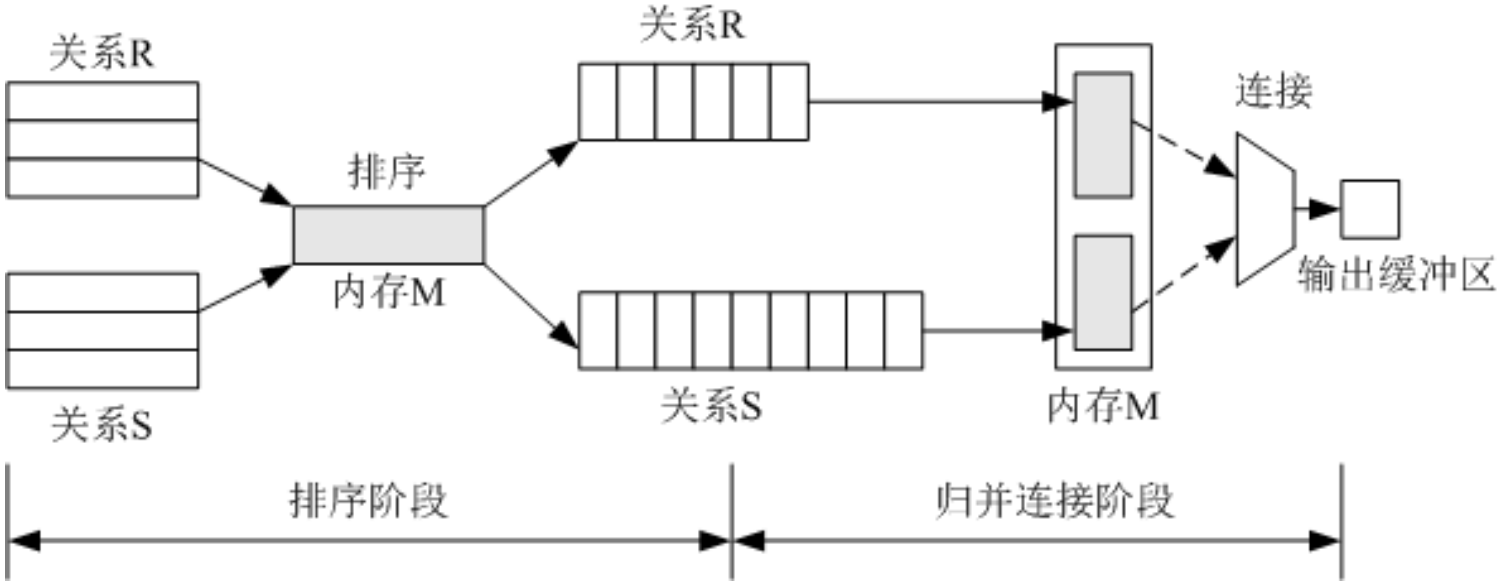
基于元组的嵌套循环连接

```
Result = {};/*初始化结果集合*/
For each tuple s in S
    For each tuple r in R
        If r.B=s.B then /*元组r和元组s满足连接条件*/
            Join r and  s as tuple t;
            Output t into Result;/*输出连接结果元组*/
        Endif
    Endfor
Endfor
Return Result
```

嵌套循环连接

- 上面基于元组的嵌套循环连接算法中，对于循环外层的关系，通常称为外关系，而对循环内层的关系称为内关系。
- 在执行嵌套循环连接时，仅对外关系进行1次读取操作，而对内关系则需要进行反复读取操作。
- 如果不进行优化的话，这种基于元组的执行代价很大，以磁盘IO计算最多可能达到Card(R)*Card(S)。
- 因此，通常对这种算法进行修改，以减少嵌套循环连接的磁盘IO代价。一种方法是使用连接属性上的索引，以减少参与连接元组的数量，称为“索引嵌套循环连接”

排序合并法(SORT-MERGE)



常用于=连接

- 首先按连接属性对表1和表2排序
- 对表1的第一个元组，从头开始扫描表2，顺序查找满足连接条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。当遇到表2中第一条大于表1连接字段值的元组时，对表2的查询不再继续

排序合并法

- 找到表1的第二条元组，然后从刚才的中断点处继续顺序扫描表2，查找满足连接条件的元组，找到后就将表1中的第二个元组与该元组拼接起来，形成结果表中一个元组。直接遇到表2中大于表1连接字段值的元组时，对表2的查询不再继续
- 重复上述操作，直到表1或表2中的全部元组都处理完毕为止

连接查询（续）

SQL中连接查询的主要类型

- 广义笛卡尔积
- 等值连接(含自然连接)
- 非等值连接查询
- 自身连接查询
- 外连接查询
- 复合条件连接查询

一、广义笛卡尔积

- 不带连接谓词的连接
- 很少使用

例：

```
SELECT  Student.* ,  SC.*
FROM    Student, SC
```

二、等值与非等值连接查询

等值连接、自然连接、非等值连接

[例32] 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；
```

等值连接

- 连接运算符为 = 的连接操作
  - [<表名1>.]<列名1> = [<表名2>.]<列名2>
  - 任何子句中引用表1和表2中同名属性时，都必须加表名前缀。引用唯一属性名时可以加也可以省略表名前缀。

自然连接

- 等值连接的一种特殊情况，把目标列中重复的属性列去掉。

[例33] 对[例32]用自然连接完成。

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

非等值连接查询

连接运算符 不是 = 的连接操作

[<表名1>.]<列名1><比较运算符>[<表名2>.]<列名2>

比较运算符：>、<、>=、<=、!=

[<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2>

AND [<表名2>.]<列名3>

三、自身连接

- 一个表与其自己进行连接，称为表的**自身连接**
- 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

自身连接（续）

[例34]　查询每一门课的间接先修课（即先修课的先修课）

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

四、外连接（Outer Join）

- 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

外连接（续）

[例 33] 查询每个学生及其选修课程的情况

包括没有选修课程的学生----用外连接操作

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

五、复合条件连接

WHERE子句中含多个连接条件时，称为复合条件连接

[例35]查询选修2号课程且成绩在86分以上的所有学生的学号、姓名

```
SELECT Student.Sno, student.Sname
FROM   Student, SC
WHERE Student.Sno = SC.Sno   /* 连接谓词*/
AND  SC.Cno= ' 2 ' /* 其他限定条件 */
   AND  SC.Grade > 86;   /* 其他限定条件 */
```

多表连接

[例36] 查询每个学生的姓名、选修课程名及成绩。

```
SELECT Sname，Cname，Grade
  FROM   Student，SC，Course
  WHERE Student.Sno = SC.Sno
                and SC.Cno = Course.Cno；
```

3.4 查　询

- 3.4.1 单表查询
- 3.4.2 连接查询
- 3.4.3 嵌套查询
- 3.4.4 集合查询
- 3.4.5 SELECT 语句的一般格式

3.4.3 嵌套查询

- 嵌套查询概述
- 嵌套查询分类
- 嵌套查询求解方法
- 引出子查询的谓词

嵌套查询(续)

- 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

嵌套查询(续)

[例] 查询选修2号课程的学生信息。

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

嵌套查询(续)

- 子查询的限制
  - 不能使用**ORDER BY**子句
- 层层嵌套方式反映了 **SQL**语言的结构化
- 有些嵌套查询可以用连接运算替代

嵌套查询分类

- 不相关子查询

  子查询的查询条件不依赖于父查询

- 相关子查询

  子查询的查询条件依赖于父查询

嵌套查询求解方法

- 不相关子查询

  是由里向外逐层处理。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
  - 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。

引出子查询的谓词

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

一、带有IN谓词的子查询

[例37] 查询与“刘晨”在同一个系学习的学生。

此查询要求可以分步来完成

① 确定“刘晨”所在系名

```
SELECT  Sdept
FROM    Student
WHERE   Sname=‘刘晨’
```

结果为：

Sdept
IS

带有IN谓词的子查询（续）

② 查找所有在IS系学习的学生。

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

结果为：

Sno	Sname	Sdept
95001	刘晨	IS
95004	张立	IS

构造嵌套查询

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。DBMS求解该查询时也是分步去做的。

带有IN谓词的子查询（续）

用自身连接完成本查询要求

```
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）

父查询和子查询中的表均可以定义别名

```
SELECT Sno，Sname，Sdept
FROM   Student  S1
WHERE  S1.Sdept  IN
      (SELECT Sdept
       FROM    Student  S2
       WHERE S2.Sname=‘刘晨’);
```

带有IN谓词的子查询（续）

[例38]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关系中找出选修了3号课程的学生学号

① 首先在Course关系中找出“信息系统”的课程号，结果为3号

带有IN谓词的子查询（续）

- 用连接查询

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

二、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<>）。
- 与ANY或ALL谓词配合使用

带有比较运算符的子查询（续）

例：假设一个学生只可能在一个系学习，并且必须属于一个系，则在[例37]可以用 = 代替IN：

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

带有比较运算符的子查询（续）

相关子查询

例：找出每个学生超过他自己选修课程平均成绩的课程号

```
Select Sno, Cno
From SC  x
Where Grade >= (select avg(Grade)
                from  SC  y
          where  y.Sno=x.Sno);
```

三、带有ANY或ALL谓词的子查询

谓词语义

- **ANY**：任意一个值
- **ALL**：所有值

带有ANY或ALL谓词的子查询（续）

需要配合使用比较运算符

> 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值
>= ALL	大于等于子查询结果中的所有值
<= ANY	小于等于子查询结果中的某个值
<= ALL	小于等于子查询结果中的所有值
= ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值（通常没有实际意义）
!=（或<>）ANY	不等于子查询结果中的某个值
!=（或<>）ALL	不等于子查询结果中的任何一个值

带有ANY或ALL谓词的子查询（续）

[例39] 查询其他系中比信息系某一个(其中某一个)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= ' IS ')
                AND Sdept <> ' IS ' ;
                /* 注意这是父查询块中的条件 */
```

带有ANY或ALL谓词的子查询（续）

结果

Sname	Sage
王敏	18

执行过程

1.DBMS执行此查询时，首先处理子查询，找出IS系中所有学生的年龄，构成一个集合(19，18)

2. 处理父查询，找所有不是IS系且年龄小于19 或 18的学生

带有ANY或ALL谓词的子查询（续）

- **ANY和ALL谓词有时可以用集函数实现**
  - **ANY与ALL与集函数的对应关系**

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

带有ANY或ALL谓词的子查询（续）

- 用集函数实现子查询通常比直接用**ANY**或**ALL**查询效率要高，因为前者通常能够减少比较次数

带有ANY或ALL谓词的子查询（续）

[例40] 查询其他系中比信息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= ' IS ')
       AND Sdept <> ' IS ’;
```

带有ANY或ALL谓词的子查询（续）

方法二：用集函数

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

补充：分页语句

利用NOT IN 和SELECT TOP实现分页

```
select TOP 页大小 *
FROM
Student
where (Sno NOT IN( select top 页大小*（页数-1） Sno from Student order by Sno))
order by Sno
```

假设每页大小为2，显示第3页，可以采用以下SQL语句：

```
select TOP 2 *
FROM
Student
where (Sno NOT IN( select top 4 Sno from
Student order by Sno))
order by Sno
```

补充：分页语句

```
select  top 页大小  *
from Student
where (Sno >
(select MAX(Sno) from  (Select top 页大小*（页数-1）  Sno from student
order by sno) as T )
)
order by sno
```

```
select top 2 *
from Student
where (Sno >
(select MAX(Sno) from  (Select top 4 Sno from
student order by sno) as T )
)
order by sno
```

四、带有EXISTS谓词的子查询

1. **EXISTS谓词**
2. **NOT EXISTS谓词**
3. **不同形式的查询间的替换**
4. **相关子查询的效率**
5. **用EXISTS/NOT EXISTS实现全称量词**
6. **用EXISTS/NOT EXISTS实现逻辑蕴函**

带有EXISTS谓词的子查询(续）

- 1. EXISTS谓词
  - 存在量词∃
  - 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，则返回真值
    - 若内层查询结果为空，则返回假值
  - 由EXISTS引出的子查询，其目标列表达式通常都用*，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
- 2. NOT EXISTS谓词

带有EXISTS谓词的子查询(续）

［例41］ 查询所有选修了2号课程的学生姓名。

带有EXISTS谓词的子查询(续）

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关系。
- 若SC中存在这样的元组，其Sno值等于此Student.Sno值，并且其Cno= ‘2’，则取此Student.Sname送入结果关系。

带有EXISTS谓词的子查询(续）

- 用嵌套查询

```
SELECT Sname
FROM Student
WHERE EXISTS
   (SELECT *
    FROM SC                    /*相关子查询*/
    WHERE Sno=Student.Sno AND Cno= '2');
```

带有EXISTS谓词的子查询(续）

- 用连接运算

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

带有EXISTS谓词的子查询(续）

［例42］ 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）

3. 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。

带有EXISTS谓词的子查询(续）

例：[例37]查询与“刘晨”在同一个系学习的学生。可以用带EXISTS谓词的子查询替换：

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

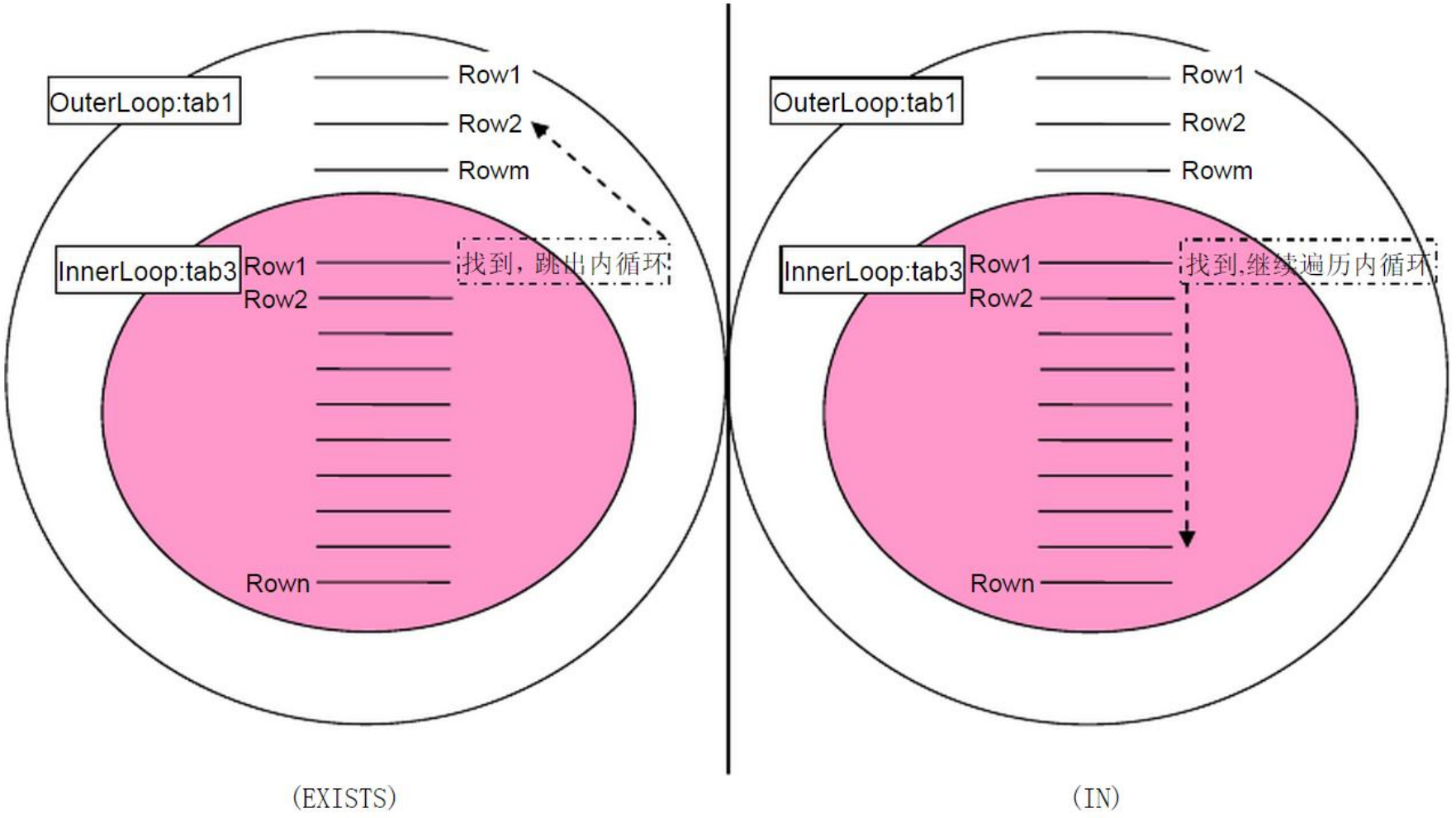
带有EXISTS谓词的子查询(续）

例：[例37]查询与“刘晨”在同一个系学习的学生。可以用带EXISTS谓词的子查询替换：

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

带有EXISTS谓词的子查询(续）

为什么在嵌套连接中，EXISTS会比IN效率高？



带有EXISTS谓词的子查询(续）

5. 用EXISTS/NOT EXISTS实现全称量词

- SQL语言中没有全称量词∀ （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：

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

带有EXISTS谓词的子查询(续）

[例43] 查询选修了全部课程的学生姓名。

SQL语句写法：查询这样的学生，没有一门课程是他不选修的

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

带有EXISTS谓词的子查询(续）

select * from sc where cno = course.cno and sno=student.sno是查询出所有已经选择过课程的学生及相应课程

select * from course where not exists 则是所有没有被选择的课程

在这个基础上的 select sname from student where not exists 则是选取所有没有未选择课程的学生，即选择了所有课程的学员名称

带有EXISTS谓词的子查询(续）

方法二：

```
Select Sname
From student
Where Sno IN
(select Sno
from SC
Group by Sno
Having count(*) = (select count(*)
from course))
```

带有EXISTS谓词的子查询(续）

6. 用EXISTS/NOT EXISTS实现逻辑蕴函

- **SQL语言中没有蕴函(Implication)逻辑运算**
- 可以利用谓词演算将逻辑蕴函谓词等价转换为：

$$p \rightarrow q \equiv \neg p \vee q$$

带有EXISTS谓词的子查询(续）

［例44］ 查询至少选修了学生95002选修的全部课程的学生号码。

解题思路：

- 用逻辑蕴函表达：查询学号为x的学生，对所有的课程y，只要95002学生选修了课程y，则x也选修了y。
- 形式化表示：

  用P表示谓词 “学生95002选修了课程y”

  用q表示谓词 “学生x选修了课程y”

  则上述查询为：$(\forall y)\ p \rightarrow q$

带有EXISTS谓词的子查询(续）

- 等价变换：

$$(\forall y)p \rightarrow q \equiv \neg (\exists y (\neg(p \rightarrow q)))$$
$$\equiv \neg (\exists y (\neg(\neg p \vee q))$$
$$\equiv \neg \exists y(p \wedge \neg q)$$

- 变换后语义：不存在这样的课程y，学生95002选修了y，而学生x没有选。

带有EXISTS谓词的子查询(续）

- 用NOT EXISTS谓词表示：

```
 SELECT * from Student
where not exists
( select * from SC SC1
  where SC1.Sno='95002' and not exists
  (select * from SC SC2
   where SC1.Cno=SC2.Cno and
   SC2.Sno=Student.Sno))
```

课堂作业

题目：查询选修了“以数据库作为先行课”的课程的学生姓名和学号

要求：

（1）第一种方法：使用多表连接

（2）第二种方法：使用嵌套查询

课堂作业

（1）第一种方法：使用多表连接

```
select Student.Sno,student.sname
from Course first, Course second,SC,student
where first.cpno=second.cno
and second.Cname='数据库'
and SC.Cno=first.Cno
and SC.Sno=Student.sno
```

课堂作业

（2）第二种方法：使用嵌套查询

```
select Student.Sno,Student.Sname
from Student
where Student.Sno in
(
select SC.Sno
from SC
where SC.Cno in
(
select first.Cno
from Course first, Course second
where first.cpno=second.cno
and second.Cname='数据库'
)
)
```

3.4 查　询

- 3.4.1 单表查询
- 3.4.2 连接查询
- 3.4.3 嵌套查询
- 3.4.4 集合查询
- 3.4.5 SELECT 语句的一般格式

3.4.4 集合查询

集合操作种类

- 并操作(UNION)
- 交操作(INTERSECT)
- 差操作(EXCEPT)

1. 并操作

- 形式

  <查询块>

  UNION

  <查询块>

  – 参加**UNION**操作的各结果表的列数必须相同；对应项的数据类型也必须相同

并操作（续）

[例45] 查询计算机科学系的学生及年龄不大于19岁的学生。

方法一：

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

并操作（续）

方法二：

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

并操作（续）

[例46] 查询选修了课程1或者选修了课程2的学生号码。

并操作（续）

方法一：

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

方法二：

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

并操作（续）

[例47] 查询学校中所有师生的姓名。

```
SELECT Sname
FROM   Student
UNION
SELECT  Tname
FROM   Teacher;
```

2. 交操作

标准SQL中没有提供集合交操作，但可用其他方法间接实现。

2. 交操作

[例48] 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

本例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;
```

交操作（续）

[例49] 查询选修课程1的学生集合与选修课程2的学生集合的交集

本例实际上是查询既选修了课程1又选修了课程2的学生

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

交操作（续）

[例50] 查询学生姓名与教师姓名的交集

本例实际上是查询学校中与教师同名的学生姓名

```
SELECT DISTINCT Sname
FROM Student
WHERE Sname IN
    (SELECT Tname
     FROM Teacher);
```

3. 差操作

标准SQL中没有提供集合差操作，但可用其他方法间接实现。

3. 差操作

[例51] 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。

本例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;
```

差操作（续）

[例52] 查询学生姓名与教师姓名的差集

本例实际上是查询学校中未与教师同名的学生姓名

```
SELECT DISTINCT Sname
FROM Student
WHERE Sname NOT  IN
        (SELECT Tname
         FROM Teacher);
```

4. 对集合操作结果的排序

- **ORDER BY**子句只能用于对最终查询结果排序，不能对中间结果排序
- 任何情况下，**ORDER BY**子句只能出现在最后
- 对集合操作结果排序时，**ORDER BY**子句中用数字指定排序属性

对集合操作结果的排序（续）

[例53] 错误写法

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

对集合操作结果的排序（续）

正确写法

```
SELECT *
FROM Student
WHERE Sdept= 'CS'
UNION
SELECT *
FROM Student
WHERE Sage<=19
ORDER BY 1;
```

3.4 查　询

- 3.4.1 单表查询
- 3.4.2 连接查询
- 3.4.3 嵌套查询
- 3.4.4 集合查询
- 3.4.5 SELECT 语句的一般格式

3.4.5 SELECT语句的一般格式

SELECT [ALL|DISTINCT]

<目标列表达式> [别名] [，<目标列表达式> [别名]] …

FROM <表名或视图名> [别名]

[，<表名或视图名> [别名]] …

[WHERE <条件表达式>]

[GROUP BY <列名1>[，<列名1’>] ...

[HAVING <条件表达式>]]

[ORDER BY <列名2> [ASC|DESC] [，<列名2’>

[ASC|DESC]] …];

第3章 关系数据库标准语言SQL

- 3.1 SQL概述
- 3.2 学生-课程数据库
- 3.3 数据定义
- 3.4 查询
- 3.5 数据更新
- 3.6 视图

3.5 数 据 更 新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据

3.5.1 插入数据

- 两种插入数据方式
  - 插入单个元组
  - 插入子查询结果

1. 插入单个元组

- 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。

插入单个元组（续）

[例1] 将一个新学生记录（学号：95020；姓名：陈冬；性别：男；所在系：IS；年龄：18岁）插入到Student表中。

```
INSERT
    INTO Student
    VALUES ('95020'，'陈冬'，'男'，'IS'，18)；
```

插入单个元组（续）

[例2] 插入一条选课记录('95020'，'1 ')。

```
INSERT
    INTO SC(Sno，Cno)
    VALUES (' 95020 '，' 1 ');
```

新插入的记录在**Grade**列上取空值

插入单个元组（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
- **VALUES子句**
  - 提供的值必须与**INTO**子句匹配
    - > 值的个数
    - > 值的类型

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;

- 功能

  将子查询结果插入指定表中

插入子查询结果（续）

[例3] 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 Deptage
    (Sdept CHAR(15)          /* 系名*/
    Avgage INT)；   /*学生平均年龄*/
```

插入子查询结果（续）

第二步：插入数据

```
INSERT
INTO  Deptage(Sdept，Avgage)
    SELECT  Sdept，AVG(Sage)
    FROM  Student
    GROUP BY Sdept；
```

插入子查询结果（续）

- **INTO**子句(与插入单条元组类似)
  - 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
  - 指定部分属性列：插入的元组在其余属性列上取空值
- 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）

DBMS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
  - 对于有**NOT NULL**约束的属性列是否提供了非空值
  - 对于有**UNIQUE**约束的属性列是否提供了非重复值
  - 对于有值域约束的属性列所提供的属性值是否在值域范围内

3.5 数 据 更 新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据

3.5.2 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

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

1. 修改某一个元组的值

[例4] 将学生95001的年龄改为22岁。

```
UPDATE  Student
        SET Sage=22
        WHERE  Sno=' 95001 ';
```

2. 修改多个元组的值

[例5] 将所有学生的年龄增加1岁。

```
UPDATE Student
    SET Sage= Sage+1
```

修改多个元组的值(续)

[例6] 将信息系所有学生的年龄增加1岁。

```
UPDATE Student
    SET Sage= Sage+1
    WHERE Sdept=' IS ';
```

3. 带子查询的修改语句

[例7] 将计算机科学系全体学生的成绩置零

```
UPDATE SC
    SET  Grade=0
    WHERE  'CS'=
        (SELECT Sdept
         FROM  Student
         WHERE  Student.Sno = SC.Sno);
```

修改数据（续）

– **SET子句**

指定修改方式

要修改的列

修改后取值

– **WHERE子句**

指定要修改的元组

缺省表示要修改表中的所有元组

修改数据（续）

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

3.5 数 据 更 新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据

3.5.3 删除数据

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
  - 缺省表示要删除表中的所有元组

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

1. 删除某一个元组的值

[例8] 删除学号为95003的学生记录。

```
DELETE
    FROM Student
    WHERE Sno='95003';
```

2. 删除多个元组的值

[例9] 删除2号课程的所有选课记录。

```
DELETE
    FROM SC;
    WHERE Cno='2';
```

[例10] 删除所有的学生选课记录。

```
DELETE
    FROM SC;
```

3. 带子查询的删除语句

[例11] 删除信息系所有学生的选课记录。

```
DELETE
    FROM SC
    WHERE ‘IS'=
        (SELETE Sdept
         FROM Student
         WHERE Student.Sno=SC.Sno);
```

删除数据(续)

DBMS在执行删除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

– 参照完整性

- 不允许删除
- 级联删除

更新数据与数据一致性

DBMS在执行插入、删除、更新语句时必须保证数据库一致性

- 必须有事务的概念和原子性
- 完整性检查和保证

第3章 关系数据库标准语言SQL

- 3.1 SQL概述
- 3.2 学生-课程数据库
- 3.3 数据定义
- 3.4 查询
- 3.5 数据更新
- 3.6 视图

3.6 视　图

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，不会出现数据冗余
- 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

3.6 视　图

基于视图的操作

- 查询
- 删除
- 受限更新
- 定义基于该视图的新视图

3.6 视　图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用

1. 建立视图

- 语句格式

CREATE VIEW

<视图名> [(<列名> [，<列名>]…)]

AS <子查询>

[WITH CHECK OPTION];

行列子集视图

[例1] 建立信息系学生的视图。

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

从单个基本表导出

只是去掉了基本表的某些行和某些列保留了码

建立视图（续）

DBMS执行CREATE VIEW语句时只是把视图的定义存入数据字典，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。

插入子查询结果（续）

[例] 对每一个系，求学生的平均年龄，并把结果存入数据库。

1.创建表，存入数据

2.通过视图

创建基表存数据

```
CREATE  TABLE  Deptage
    (Sdept  CHAR(15)          /* 系名*/
    Avgage INT)；   /*学生平均年龄*/

INSERT INTO  Deptage(Sdept，vgage)
 SELECT  Sdept，AVG(Sage)
 FROM  Student   GROUP BY Sdept；
```

直接创建视图

Create view VSA(Sdept，vgage)
as SELECT Sdept，AVG(Sage)
FROM Student GROUP BY Sdept；

组成视图的属性列名

全部省略或全部指定

– 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:

(1) 某个目标列是集函数或列表达式

(2) 多表连接时选出了几个同名列作为视图的字段

(3) 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）

- **WITH CHECK OPTION**

透过视图进行增删改操作时，不得破坏视图定义中的谓词条件

（即子查询中的条件表达式）

WITH CHECK OPTION的视图

[例2] 建立信息系学生的视图，并要求透过该视图进行的更新操作只涉及信息系学生。

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

对IS_Student视图的更新操作

- 修改操作：**DBMS自动加上Sdept= 'IS'的条件**
- 插入操作：**DBMS自动检查Sdept属性值是否为'IS'**
  - 如果不是，则拒绝该插入操作
- 删除操作：**DBMS自动加上Sdept= 'IS'的条件**

基于多个基表的视图

[例4] 建立选修了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 Student.Sno=SC.Sno AND
            SC.Cno= '1'；
```

基于视图的视图

[例5] 建立选修了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；
```

带表达式的视图

[例6] 定义一个反映学生出生年份的视图（包括学号、姓名、出生年份）。

```
CREATE  VIEW BT_S(Sno，Sname，Sbirth)
        AS
        SELECT Sno，Sname，2005-Sage    FROM  Student
```

设置一些派生属性列，也称为虚拟列--Sbirth

带表达式的视图必须明确定义组成视图的各个属性列名

建立分组视图

[例7] 将学生的学号及他的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；
```

建立视图（续）

- 一类不易扩充的视图
  - 以 **SELECT *** 方式创建的视图可扩充性差，应尽可能避免

建立视图（续）

[例8]将Student表中所有女生记录定义为一个视图

```
CREATE VIEW
        F_Student1(stdnum，name，sex，age，dept)
AS  SELECT *
FROM  Student
 WHERE Ssex='女';
```

建立视图（续）

缺点：修改基表**Student**的结构后，

Student表与**F_Student1**视图的映象关系被破坏，导致该视图不能正确工作。

建立视图（续）

```
CREATE VIEW
        F_Student2 (stdnum， name， sex， age，
dept)
  AS SELECT Sno，Sname，Ssex，Sage，Sdept
      FROM Student
      WHERE Ssex='女'；
```

为基表Student增加属性列不会破坏Student表与F_Student2视图的映象关系。

常见的视图形式

- 行列子集视图
- **WITH CHECK OPTION的视图**
- 基于多个基表的视图
- 基于视图的视图
- 带表达式的视图
- 分组视图

2. 删除视图

- **DROP VIEW <视图名>;**
  - 该语句从数据字典中删除指定的视图定义
  - 由该视图导出的其他视图定义仍在数据字典中，但已不能使用，必须显式删除
  - 删除基表时，由该基表导出的所有视图定义都必须显式删除

删除视图(续）

[例9] 删除视图IS_S1

```
DROP VIEW IS_S1;
```

3.6 视　图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用

3.6.2 查询视图

- 从用户角度：查询视图与查询基本表相同
- **DBMS实现视图查询的方法**
  - 实体化视图（**View Materialization**）
    - 有效性检查：检查所查询的视图是否存在
    - 执行视图定义，将视图临时实体化，生成临时表
    - 查询视图转换为查询临时表
    - 查询完毕删除被实体化的视图(临时表)

查询视图（续）

– 视图消解法（View Resolution）

- 进行有效性检查，检查查询的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义
- 把视图定义中的子查询与用户的查询结合起来，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）

[例1] 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；
```

IS_Student视图的定义 (视图定义例1)：

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

查询视图（续）

– 视图实体化法

– 视图消解法

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）

[例2] 查询信息系选修了1号课程的学生

```
SELECT  Sno，Sname
FROM    IS_Student，SC
WHERE  IS_Student.Sno =SC.Sno AND
          SC.Cno= '1';
```

查询视图（续）

- 视图消解法的局限
  - 有些情况下，视图消解法不能生成正确查询。采用视图消解法的DBMS会限制这类查询。

查询视图（续）

[例3]在S_G视图中查询平均成绩在90分以上的学生学号和平均成绩

```
SELECT *
FROM   S_G
WHERE  Gavg>=90;
```

S_G视图定义：

```
CREATE VIEW S_G (Sno，Gavg)
  AS
    SELECT Sno，AVG(Grade)
    FROM SC
    GROUP BY Sno;
```

查询转换

错误：

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

正确：

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

3.6 视　图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用

3.6.3 更新视图

- 用户角度：更新视图与更新基本表相同
- **DBMS实现视图更新的方法**
  - 视图实体化法（**View Materialization**）
  - 视图消解法（**View Resolution**）
- 指定**WITH CHECK OPTION**子句后

DBMS在更新视图时会进行检查，防止用户通过视图对不属于视图范围内的基本表数据进行更新

更新视图（续）

[例1] 将信息系学生视图IS_Student中学号95002的学生姓名改为“刘辰”。

```
UPDATE  IS_Student  SET  Sname= '刘辰'
WHERE  Sno= '95002';
```

转换后的语句：

```
UPDATE  Student  SET Sname= '刘辰'
WHERE Sno= '95002' AND Sdept= 'IS';
```

更新视图（续）

[例2] 向信息系学生视图IS_S中插入一个新的学生记录：95029，赵新，20岁

```
INSERT
INTO IS_Student
VALUES(‘95029’，‘赵新’，20)；
```

转换为对基本表的更新：

```
INSERT
INTO  Student(Sno，Sname，Sage，Sdept)
VALUES('95029'，'赵新'，20，'IS' )；
```

更新视图（续）

[例3] 删除视图CS_S中学号为95029的记录

```
DELETE
FROM IS_Student
WHERE Sno= '95029';
```

转换为对基本表的更新：

```
DELETE
FROM Student
WHERE Sno= '95029' AND Sdept= 'IS';
```

更新视图的限制

- 一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新(对两类方法均如此)

例：视图S_G为不可更新视图。

```
CREATE VIEW S_G (Sno，Gavg)
    AS
SELECT  Sno，AVG(Grade)
FROM    SC
GROUP BY Sno；
```

更新视图（续）

对于如下更新语句：

```
UPDATE  S_G
SET     Gavg=90
WHERE  Sno= '95001';
```

无论实体化法还是消解法都无法将其转换成对基本表SC的更新

3.6 视　图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用

1. 视图能够简化用户的操作

当视图中数据不是直接来自基本表时，定义视图能够简化用户的操作

- 基于多张表连接形成的视图
- 基于复杂嵌套查询的视图
- 含导出属性的视图

2. 视图使用户能以多种角度看待同一数据

- 视图机制能使不同用户以不同方式看待同一数据，适应数据库共享的需要

例：数据库逻辑结构发生改变

学生关系Student(Sno，Sname，Ssex，Sage，Sdept)

“垂直”地分成两个基本表：

SX(Sno，Sname，Sage)

SY(Sno，Ssex，Sdept)

通过建立一个视图**Student**：

```
CREATE VIEW  Student(Sno，Sname，Ssex，Sage，
    Sdept)
AS
SELECT  SX.Sno，SX.Sname，SY.Ssex，SX.Sage，
    SY.Sdept
FROM  SX，SY
WHERE  SX.Sno=SY.Sno；
```

使用户的外模式保持不变，从而对原**Student**表的查询程序不必修改

3. 视图对重构数据库提供了一定程度的逻辑独立性

- 视图在一定程度上保证了数据的逻辑独立性
- 视图只能在一定程度上提供数据的逻辑独立性
  - 由于对视图的更新是有条件的，因此应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。
- 对不同用户定义不同视图，使每个用户只能看到他有权看到的数据

作业

- 教材130页
- 第4、5题

附录：主讲教师

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），现为厦门大学计算机科学系助理教授（讲师），曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度厦门大学奖教金获得者。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。编著出版中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》并成为畅销书籍，编著并免费网络发布40余万字中国高校第一本闪存数据库研究专著《闪存数据库概念与技术》；主讲厦门大学计算机系本科生课程《数据库系统原理》和研究生课程《分布式数据库》《大数据技术基础》。具有丰富的政府和企业信息化培训经验，曾先后给中国移动通信集团公司、福州马尾区政府、福建省物联网科学研究院、石狮市物流协会、厦门市物流协会、福建龙岩卷烟厂等多家单位和企业开展信息化培训，累计培训人数达2000人以上。

附录：课程助教

助教：谢荣东

单位：厦门大学计算机科学系数据库实验室2014级硕士研究生
E-mail: xrdxmu@sina.com

助教：薛倩

单位：厦门大学计算机科学系数据库实验室2015级硕士研究生
E-mail: xueqian_victoria@163.com

附录：班级网站

林子雨主讲《数据库系统原理》2016班级主页

http://dblab.xmu.edu.cn/post/5550/

扫一扫访问班级网站
支持手机浏览


Thank You!