第三章 关系数据库标准语言SQL

3.1 SQL概述
3.2 创建环境
3.3 数据定义
3.4 数据查询
3.5 数据更新
3.6 空值的处理
3.7 视图
3.8 小结

第三章 关系数据库标准语言SQL

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

- SQL语言(Structured Query Language)
  - 1974年由Boyce和Chamberlin提出
  - 1975年～1979年IBM公司在System R原型系统上实现
  - 是关系数据库的标准语言，是数据库领域中一个主流语言

3.1.1 SQL的产生与发展

- SQL标准
  - SQL-86
    - 第一个SQL标准
    - 由美国国家标准局（American National Standard Institute，简称ANSI）公布
    - 1987年国际标准化组织（International Organization for Standardization，简称ISO）通过
  - SQL-89
  - SQL-92
  - SQL3　　/*面向对象*/

3.1.2 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. 同一种语法结构提供两种使用方式
- 5. 语言简捷，易学易用

2. 综合统一

- SQL语言集数据定义语言DDL、数据操纵语言DML、数据控制语言DCL的功能于一体。

2. 综合统一

- 可以独立完成数据库生命周期中的全部活动，包括：
  - 定义关系模式，插入数据建立数据库
  - 对数据库中的数据进行查询和更新
  - 数据库重构和维护
  - 数据库安全性、完整性控制
  - 等

2. 高度非过程化

- 用户只需提出“做什么”，而不必指明“怎么做”

- 存取路径的选择以及SQL语句的操作过程由系统自动完成。大大减轻了用户负担，而且有利于提高数据独立性。

3. 面向集合的操作方式

- SQL语言采用集合操作方式
  - 操作对象、查找结果可以是元组的集合
  - 一次插入、删除、更新操作的对象可以是元组的集合

4. 同一种语法结构提供两种使用方式

- 自含式语言
  - 能够独立地用于联机交互的使用方式
- 嵌入式语言
  - 能够嵌入到高级语言（例如C，COBOL，FORTRAN，PL/1）程序中，供程序员设计程序时使用。

两种不同使用方式下，SQL语言的语法结构基本一致

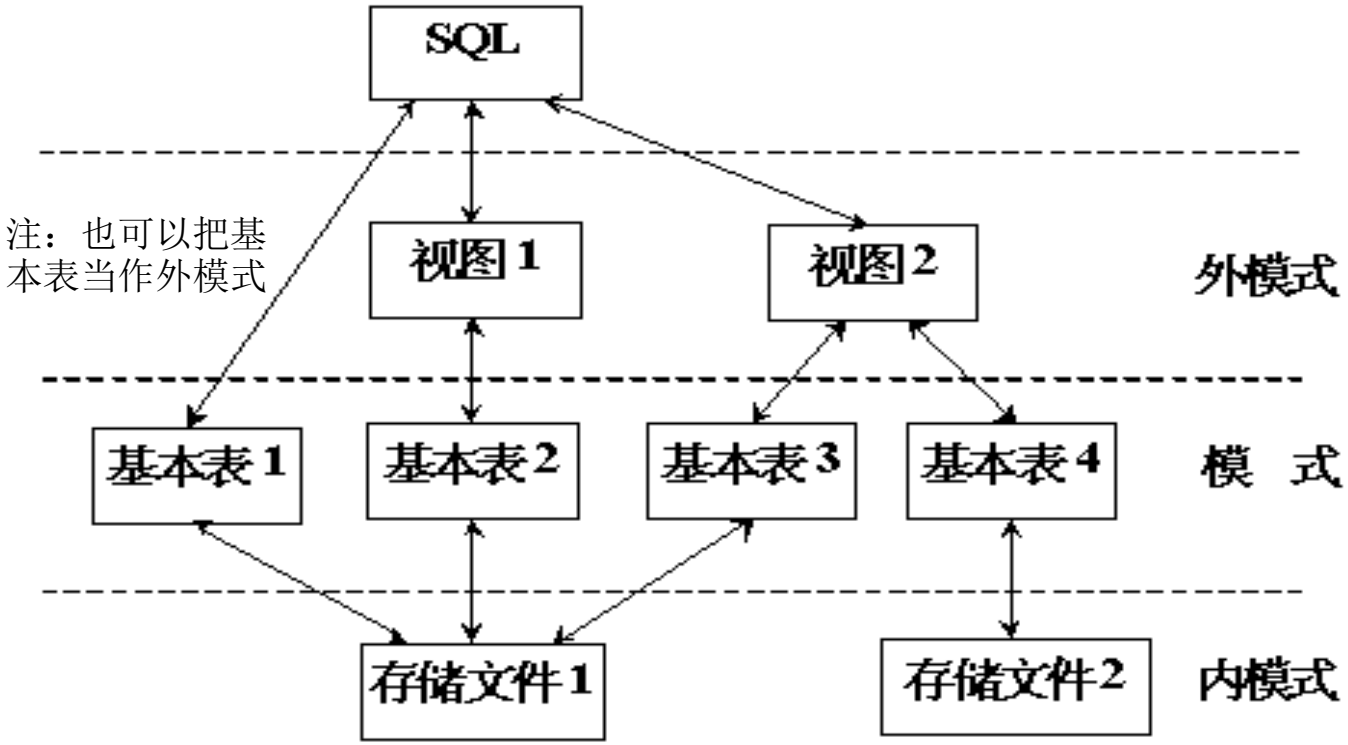
5. 语言简捷，易学易用

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

3.1.3 SQL语言的基本概念

SQL 语言支持关系数据库三级模式结构



SQL语言的基本概念（续）

- 用户用SQL语言对基本表和视图进行操作
- 基本表
  - 本身独立存在的表，一个关系对应一个表
  - 一个（或多个）基本表对应一个存储文件
  - 一个表可以带若干索引，索引也存放在存储文件中
- 存储文件
  - 存储文件的逻辑结构组成了关系数据库的内模式
  - 存储文件的物理结构是任意的，对用户是透明的
- 视图
  - 从一个或几个基本表或视图导出的表
  - 是虚表，只存放视图的定义而不存放对应数据

第三章 关系数据库标准语言SQL

3.1 SQL概述
3.2 创建环境
3.3 数据定义
3.4 数据查询
3.5 数据更新
3.6 空值的处理
3.7 视图
3.8 小结

3.2 创建环境

- 3.2.1 安装SQL Server 数据库系统
- 3.2.2 设计 学生—课程数据库

3.2.1 安装SQL Server 数据库系统

- 本安装过程以安装 SQL server 2005 个人版为例。
- 运行安装程序

启动 SQL Server

- 通过“开始”、“程序”、“SQL Server Management studio”，启动“SQL Server”。

3.2.2 设计 学生—课程数据库

- 在 第二章的 关系代数 一节中曾经设计过一个 学生—课程数据库，现在仍以该数据库为例
- 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)

3.2.2 设计 学生—课程数据库

Student

Sno	Sname	Ssex	Sage	Sdept
200215121	李勇	男	20	CS
200215122	刘晨	女	19	CS
200215123	王敏	女	18	MA
200215125	张立	男	19	IS

3.2.2 设计 学生—课程数据库

Course

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

3.2.2 设计 学生—课程数据库

SC

Sno	Cno	Grade
200215121	1	92
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80

第三章 关系数据库标准语言SQL

3.1 SQL概述
3.2 创建环境
3.3 数据定义
3.4 数据查询
3.5 数据更新
3.6 空值的处理
3.7 视图
3.8 小结

数据定义概述

- SQL的数据定义功能
  - 定义表(模式)
    - 创建表
    - 删除表
    - 修改表定义
  - 定义视图(外模式)
    - 创建视图
    - 删除视图
    - 间接修改视图定义：删除+创建

数据定义概述(续)

- SQL的数据定义功能（续）
  - 定义索引(内模式)
    - 创建索引
    - 删除索引
    - 间接修改索引定义：删除+创建

数据定义概述(续)

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 模式的定义和删除

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

3.3.4 数据字典

3.3 数据定义

3.3.1 模式的定义和删除

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

3.3.4 数据字典

3.3.1 模式的定义和删除

一、定义模式

二、删除模式

三、建立 学生－课程数据库

一、定义模式

- 语句格式

  CREATE SCHEMA [模式名] AUTHORIZATION <用户名>

  – 模式名：数据库名
  – 关于权限

二、模式 与 库

- 创建模式，就是创建了一个数据库
  - 使用CREATE DATABASE 代替CREATE SCHEMA
  - 使用DROP DATABASE 代替DROP SCHEMA

三、建立 学生—课程数据库

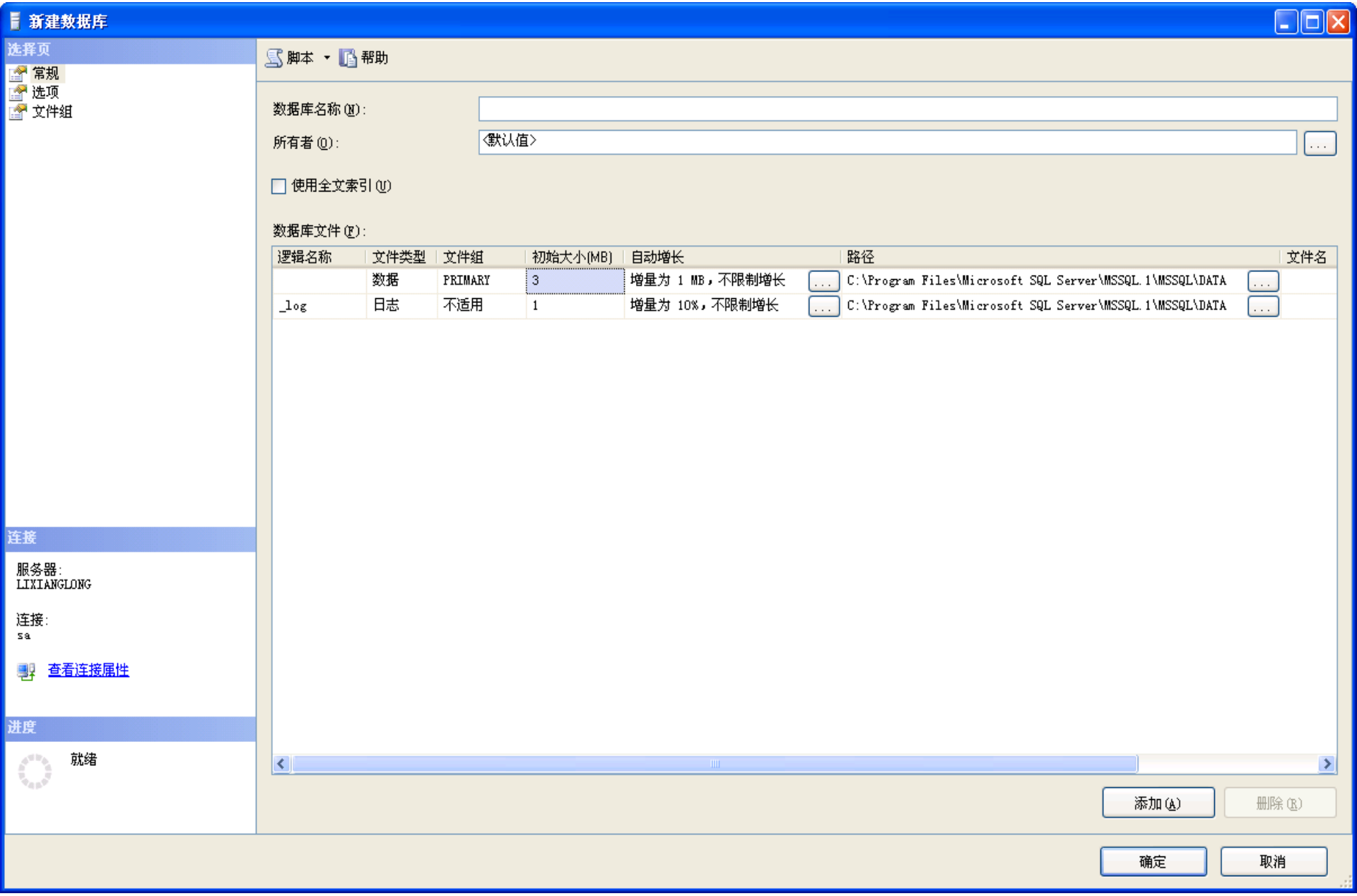
- 通过“对象资源管理器”建库
- 通过“SQL”命令，建立数据库

通过“对象资源管理器”建库

- 运行 SQL SERVER，打开左侧树状视图控件（对象资源管理器），定位到数据库文件夹。点击右键，新建数据库。

通过“对象资源管理器”建库

- 输入数据库的名称 school 。查看 “数据文件” 和 “事务日志” 选项卡，并可以进行相应的参数设置。
- 确定。


新建数据库
选择页
常规
选项
文件组
脚本
帮助
数据库名称(N):
所有者(O):
<默认值>
使用全文索引(U)
数据库文件(F):
逻辑名称
文件类型
文件组
初始大小(MB)
自动增长
路径
文件名
数据
PRIMARY
3
增量为 1 MB，不限制增长
C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA
_log
日志
不适用
1
增量为 10%，不限制增长
C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA
连接
服务器:
LIXIANGLONG
连接:
sa
查看连接属性
进度
就绪
添加(A)
删除(R)
确定
取消

通过“对象资源管理器”建库

- 查看建立的数据库
  - 在“对象资源管理器”中，展开数据库文件夹，可以看到新建立的数据库。（可能需要刷新）
- 删除数据库
  - 在“对象资源管理器”中选择数据库，按“Del”键
  - 启动 SQL查询分析器，在SQL命令窗口中输入命令：

```
USE master
IF EXISTS (SELECT name
      FROM master.dbo.sysdatabases
      WHERE name = 'school')
DROP DATABASE [school]
```

通过“SQL”命令，建立数据库

- 启动“查询分析器”，输入SQL命令：

```
use master;
create database school on primary     --数据库名为school
   (name = ‘school_data’,              --数据文件名
   filename = ‘f:\lxl\sql\school_data.mdf’,        --数据文件物理位置
   size = 5mb,                         --默认大小
   filegrowth = 10%)                   --使用空间用满后的增长速度
log on (name = ‘school_log’,           --以下为日志文件
   filename = 'f:\lxl\sql\school_log.ldf',
   size = 5mb,
   filegrowth = 10%);
```

*primary:文件组名称

3.3 数据定义

3.3.1 模式的定义和删除

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

3.3.4 数据字典

3.3.2 基本表的定义、删除与修改

一、定义基本表
二、数据类型
三、模式与表
四、修改基本表
五、删除基本表

一、定义基本表

- 关系名（表名）
- 属性名（列名）
- 完整性约束

定义基本表（续）

- 语句格式

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>] ）；

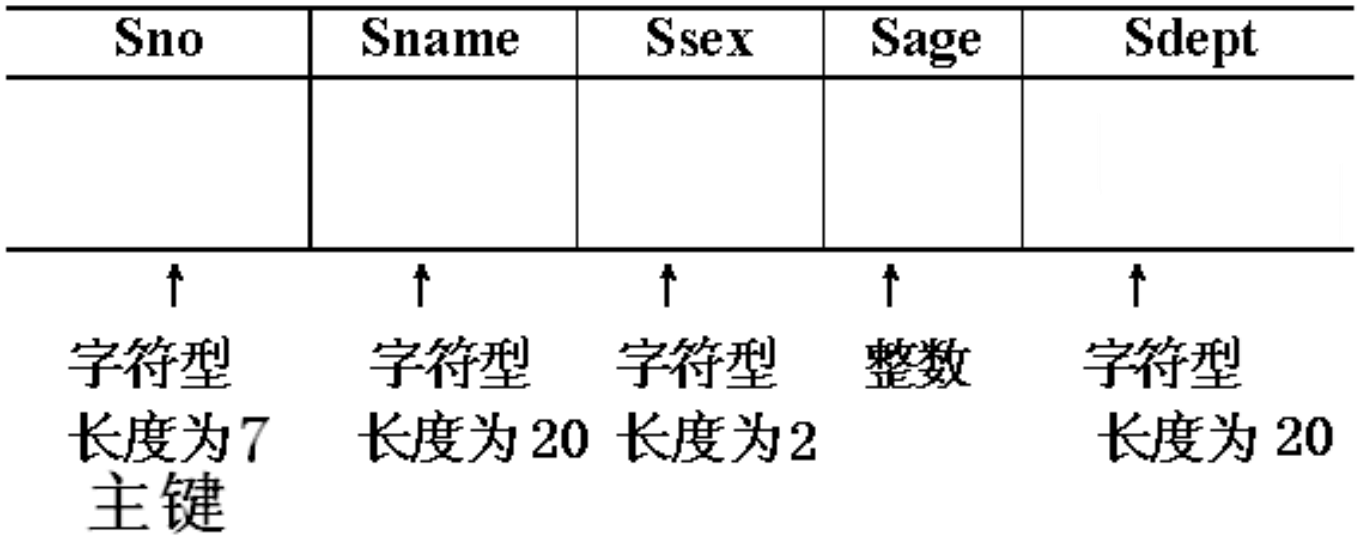
- <表名>：所要定义的基本表的名字
- <列名>：组成该表的各个属性（列）
- <列级完整性约束条件>：涉及相应属性列的完整性约束条件
- <表级完整性约束条件>：涉及一个或多个属性列的完整性约束条件

定义基本表（续）

- 表级完整性约束与列级完整性约束
  - 只涉及单个列的完整性约束可以定义为列级完整性约束
- 常用完整性约束
  - 主码约束：　PRIMARY KEY
  - 参照完整性约束：　FOREIGN key
  - 唯一性约束：UNIQUE
  - 非空值约束：NOT NULL
  - 取值约束：CHECK

例题

- [例5] 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。其中学号为主码，并且姓名取值唯一。



例题（续）

```
CREATE TABLE Student
       (Sno  CHAR(7)  PRIMARY KEY,
        Sname  CHAR(20) UNIQUE,
        Ssex  CHAR(2),
        Sage  SMALLINT,
        Sdept  CHAR(20)
       );
```

例题（续）

- [例6] 建立一个“课程”表Course，它由课程号Cno、课程名Cname、先修课号Cpno，学分Ceredit组成。其中Cno为主码，Cpno是外码。

```
CREATE TABLE Course
      (Cno CHAR(4) PRIMARY KEY ,
       Cname CHAR(40) ,
       Cpno   CHAR(4),
       Ceredit SMALLINT,
        FOREIGN  key (Cpno)  REFERENCES  Course(Cno));
```

例题 （续）

- [例7] 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，Sno、Cno是外码。

```
CREATE TABLE SC
        (Sno CHAR(7) ,
         Cno CHAR(4) ,
         Grade   SMALLINT,
         Primary key (Sno, Cno) ,
        FOREIGN key (Sno) REFERENCES Student(Sno) ,
         FOREIGN key (Cno) REFERENCES Course(Cno)
        );
```

二、数据类型

- 数据类型
  - SQL中域的概念用数据类型来实现，不同的关系数据库系统支持的数据类型不完全相同
  - SQL提供了一些主要的数据类型

二、数据类型

数据类型	含义
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
INT	长整数
SMALLINT	短整数
NUMERIC(p,d)	定点数，由p位数字（不包括符号、小数点）组成，小数点后面有d位数字
DECIMAL(p,d)同NUMERIC	
REAL	取决于机器精度的浮点数
DOUBLE PRECISON	取决于机器精度的双精度浮点数
FLOAT(n)	浮点数，精度至少为n位数字
DATE	日期
TIME	时间
等	

三、模式（数据库）与表

- 每一个基本表属于某一个数据库，一个数据库包含多个基本表。
- 如何定义一个基本表所属的数据库
  - 不同的DBMS，使用的方法不同。SQL SERVER，通过切换（打开）具体的数据库，在其中创建数据表。

四、修改基本表

- 语句格式
  - ALTER TABLE <表名>
    - [ADD <新列名> <数据类型> [完整性约束]]
    - [ADD < constraint 完整性约束名><约束内容>]
    - [DROP <constraint 完整性约束名>]
    - [DROP COLUMN 列名]
    - [ALTER 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

例题

- [例8] 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 ;

  – 不论基本表中原来是否已有数据，新增加的列一律为空值。
  – 如果基本表中原来已有数据，新增列不可有NOT NULL约束 及 UNIQUE约束

例题

- [例9] 将年龄的数据类型改为整数。

```
ALTER TABLE Student alter column Sage  int;
```

  – 注：修改原有的列定义有可能会破坏已有数据。

例题

- [例10] 增加课程名称必须取唯一值的的约束条件。

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

例 删除列

```
ALTER TABLE Student DROP COLUMN S_entrance
```

五、删除基本表

- 语句格式

  DROP TABLE <表名> [RESTRICT | CASCADE];

  – 系统从数据字典中删去:
    - 该基本表的描述
    - 该基本表上的所有索引的描述
    - 该基本表上建立的视图
  – 系统从文件中删去表中的数据
  – 有的系统保留表上的视图，但无法引用

例题

- [例11] 删除Student表。

  DROP TABLE Student CASCADE;

- 不同的dbms删除表的处理方式不同（P87）

3.3 数据定义

3.3.1 模式的定义和删除

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

3.3.4 数据字典

3.3.3 索引的建立删除

- 建立索引是加快查询速度的有效手段
- 建立索引
  - DBMS自动建立
    - PRIMARY KEY
    - UNIQUE
  - DBA或表的属主（即建立表的人）根据需要建立
- 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，相当于添加了UNIQUE约束。
  – 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。即聚簇索引的索引项顺序与表中记录的物理顺序一致。
  - [例13]：

    CREATE CLUSTERed 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
  - /*注：小心使用，否则会导致相反的结果*/

例题

- [例14]　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

- 索引由系统使用和维护，无需用户干预。如果数据增删改频繁，系统就会花许多时间维护索引降低查询效率，可以删除一些不必要的索引。
- 语句格式

  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。

例题

- [例13] 删除Student表的Stusname索引。

  DROP INDEX Stusname;

  DROP INDEX student.Stusno

3.3.4 数据字典

- 数据字典是关系数据库管理系统内部的一组系统表，它记录了数据库中所有的定义信息，包括关系模式定义、视图定义、索引定义、完整性约束定义、各类用户对数据库的操作权限、统计信息等。
- SQL SERVER中的master数据库就是数据字典。

第三章 关系数据库标准语言SQL

3.1 SQL概述
3.2 创建环境
3.3 数据定义
3.4 数据查询
3.5 数据更新
3.6 空值的处理
3.7 视图
3.8 小结

3.4 查　询

- 语句格式

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> [, <列名>] …
[HAVING <条件表达式>]]
[ORDER BY <列名> [, <列名>] … [ASC|DESC]];

语句格式

- 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

示例数据库

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)

3.4 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 基于派生表的查询
- 3.4.6 SELECT 语句的一般格式

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 基于派生表的查询
- 3.4.6 SELECT 语句的一般格式

3.4.1 单表查询

- 单表查询

  查询仅涉及一个表，是一种最简单的查询操作

  – 选择表中的若干列
  – 选择表中的若干元组
  – 对查询结果排序
  – 使用集函数
  – 对查询结果分组

一、选择表中的若干列

- 属投影运算
  - 不消除重复行
- 变化方式主要表现在SELECT子句的<目标表达式>上
  - 查询指定列
  - 查询全部列
  - 查询经过计算的值

1. 查询指定列

- 方法
  - 在SELECT子句的<目标列表达式>中指定要查询的属性
  - <目标列表达式>中各个列的先后顺序可以与表中的逻辑顺序不一致。即用户可以根据应用的需要改变列的显示顺序

例题

[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;
```

2. 查询全部列

- 方法
  - 在SELECT关键字后面列出所有列名
  - 当列的显示顺序与其在基表中的顺序相同时，也可以简单地将<目标列表达式>指定为 *

例题

[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

- 方法
  - 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

例题

[例4] 查全体学生的姓名及其出生年份。

```
SELECT Sname, year(getdate())-sage
FROM Student;
```

输出结果：

Sname	（无列名）
李勇	1984
刘晨	1985
王敏	1986
张立	1985

例题（续）

[例5] 查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名。

```
SELECT Sname,'Year of Birth: ', 2016-Sage, LOWER(Sdept)
FROM Student;
```

例题（续）

输出结果：

Sname	'Year of Birth:'	2016-Sage	LOWER(Sdept)
李勇	Year of Birth:	1984	cs
刘晨	Year of Birth:	1985	cs
王名	Year of Birth:	1986	ma
张立	Year of Birth:	1985	is

例题（续）

[例5.1] 使用列别名改变查询结果的列标题

```
SELECT Sname NAME, 'Year of Birth: ' BIRTH,
                2016-Sage BIRTHDAY,
                 LOWER(Sdept) DEPARTMENT
FROM Student;
```

输出结果：

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1984	cs
刘晨	Year of Birth:	1985	cs
王名	Year of Birth:	1986	ma
张立	Year of Birth:	1985	is

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

- 方法
  - 在SELECT子句中使用DISTINCT短语

例题（续）

[例6] 查询选修了课程的学生学号。

(1) SELECT Sno FROM SC;

或

SELECT ALL Sno FROM SC;

结果：

```
Sno
---------
200215121
200215121
200215121
200215122
200215122
```

例题（续）

(2)

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

结果：

```
    Sno
-----------
200215121
200215122
```

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

- 属选择运算

- 通过WHERE子句实现
  - 比较大小
  - 确定范围
  - 确定集合
  - 字符串匹配
  - 涉及空值的查询
  - 多重条件查询

查询满足条件的元组（续）

WHERE子句常用的查询条件

表3.4 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, NOT

(1) 比较大小

- 方法
  - 在WHERE子句的<比较条件>中使用比较运算符
    - =，>，<，>=，<=，!=或<>，!>，!<，
    - 逻辑运算符NOT + 含上述比较运算符的表达式

例题

[例7] 查询计算机系全体学生的名单。

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

例题

[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;
```

例题（续）

[例9] 查询考试成绩有不及格的学生的学号。

```
SELECT   DISTINCT Sno
FROM     SC
WHERE    Grade < 60
```

(2) 确定范围

- 方法
  - 使用谓词　BETWEEN … AND …

    NOT BETWEEN … AND …
    - BETWEEN后：范围的下限（即低值）
    - AND后：范围的上限（即高值）
  - 用多重条件查询实现

例题

[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

- 方法
  - 使用谓词　　IN <值表>

    NOT IN <值表>
    - <值表>：用逗号分隔的一组取值
  - 用多重条件查询实现

例题

[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' );
```

例题

[例13] 查询既不是信息系、数学系，也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' , 'MA' , 'CS' );
```

(4) 字符串匹配

- 方法
  - –使用谓词LIKE或NOT LIKE

    [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

字符串匹配(续)

- 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。

– 下划线出现在中间表示具体字符；出现在结尾可以为空。

字符串匹配(续)

– 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200215121的学生的详细情况。

```
SELECT *
FROM  Student
WHERE  Sno LIKE '200215121';
```

等价于：

```
SELECT  *
FROM  Student
WHERE Sno = '200215121';
```

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“欧阳”且全名为三个汉字的学生的姓名。（sql server 中，最后的_，可以为空。）

```
SELECT Sname
FROM   Student
WHERE  Sname LIKE '欧阳_';
```

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"阳"字的学生的姓名和学号。

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

*汉字使用一个 “_”表示

例题（续）

匹配模板为含通配符的字符串（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DB_Design课程的课程号和学分。

```
SELECT Cno , Ccredit
FROM Course
WHERE Cname LIKE 'DB_Design'
```

例题（续）

使用换码字符将通配符转义为普通字符(续)

[例19] (续)

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

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

例题

[例21] 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;
```

例题(续)

[例22] 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;
```

(6) 多重条件查询

- 方法
  - 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
```

例题（续）

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';
```

例题（续）

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

- 方法
  - 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（不同的DBMS处理方式不同）
    - ASC：排序列为空值的元组最后显示

    /*注：可以认为空值是无穷大*/
    - DESC：排序列为空值的元组最先显示

对查询结果排序（续）

- 例题

  [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

- 方法
  - 5类主要集函数
    - 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）

- 5类主要集函数(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）

- 例题

[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;
```

用DISTINCT以避免重复计算学生人数

注意：在不同的列进行Count计算，返回的值可能不同，因为count计算的为非空的值得个数。

使用集函数（续）

[例28] 计算1号课程的学生平均成绩。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno= '1';
```

[例29] 查询选修1号课程的学生最高分数。

```
SELECT MAX(Grade)
FROM SC
WHERE Cno= '1';
```

使用集函数（续）

[例30] 查询学生200215121选修课程的总学分数。

```
SELECT SUM(Ccredit)
FROM SC,Course
WHERE Sno= '200215121' AND SC.Cno= Course.Cno;
```

五、对查询结果分组

- 用途
  - 细化集函数的作用对象
    - 未对查询结果分组，集函数将作用于整个查询结果
    - 对查询结果分组后，集函数将分别作用于每个组

对查询结果分组 （续）

- 方法
  - 使用GROUP BY子句分组
    - 分组方法：按指定的一列或多列值分组，值相等的为一组
    - 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集函数

对查询结果分组（续）

- 方法　(续)
  - 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。

例题

[例31] 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

例题

[例32] 查询选修了3门以上课程的学生学号。

```
SELECT Sno, COUNT(Sno)  人数
FROM SC
GROUP BY Sno
HAVING COUNT(*) >3;
```

例题

[例]　查询有3门以上课程在90分以上的学生的学号及90分以上的课程数。

```
SELECT Sno, COUNT(*)
 FROM SC
 WHERE Grade>=90
 GROUP BY Sno
 HAVING COUNT(*)>=3;
```

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 基于派生表的查询
- 3.4.6 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
- 连接条件
  - 用来连接两个表的条件称为连接条件
  - 常用格式
    - [<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
  - 连接条件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

连接查询（续）

一、等值连接查询

二、自身连接查询

三、外连接查询

四、复合条件连接查询

一、等值连接查询

- 等值连接
- 自然连接

一、等值与非等值连接查询

- 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。

等值与非等值连接查询（续）

[例33] 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;
```

等值与非等值连接查询（续）

结果表

Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
200215121	李勇	男	20	CS	200215121	1	92
200215121	李勇	男	20	CS	200215121	2	85
200215121	李勇	男	20	CS	200215121	3	88
200215122	刘晨	女	19	IS	200215122	2	90
200215122	刘晨	女	19	IS	200215122	3	80

等值与非等值连接查询（续）

- 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。
    - <表名1>.<列名1> = <表名2>.<列名2>
  - SELECT语句不能直接实现自然连接

[例34] 对[例33]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）

- 非等值连接
  - 连接运算符不为 = 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>

二、自身连接

- 一个表与其自己进行连接，称为表的自身连接

- 表示方法
  - 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35] 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
      and SECOND.Cpno is not null;
```

自身连接（续）

结果

FIRST表（Course表）

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

自身连接（续）

SECOND表（Course表）

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

自身连接（续）

查询结果

cno	cpno
1	7
3	5
5	6

三、外连接（Outer Join）

- 外连接与普通连接（内连接）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）

- 外连接操作的种类
  - 左外连接（主要用法）
    - 左表显示全部数据
  - 右外连接（没有意义）
    - 右表显示全部数据

外连接（续）

[例36]：用外连接操作改写[例33]

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

外连接（续）

结果：

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
200215121	李勇	男	20	CS	1	92
200215121	李勇	男	20	CS	2	85
200215121	李勇	男	20	CS	3	88
200215122	刘晨	女	19	IS	2	90
200215122	刘晨	女	19	IS	3	80
200215123	王敏	女	18	MA	null	null
200215124	张立	男	19	IS	null	null

四、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

- 复合条件连接的类型
  - 两表按多个属性连接
  - 自身按多个属性连接
  - 多表连接

复合条件连接（续）

[例37] 查询选修2号课程且成绩在90分以上的所有学生的学号，姓名。

```
SELECT Student.Sno, Sname
 FROM   Student,SC
 WHERE  Student.Sno = SC.Sno   /* 连接条件 */
    AND    SC.Cno= '2'
    AND    SC.Grade>90        /* 其他限定条件 */
```

复合条件连接（续）

[例38] 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
WHERE Student.Sno = SC.Sno
      and SC.Cno =Course.Cno;
```

结果：

Student.Sno	Sname	Cname	Grade
200215121	李勇	数据库	92
200215121	李勇	数学	85
200215121	李勇	信息系统	88
200215122	刘晨	数学	90
200215122	刘晨	信息系统	80

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 基于派生表的查询
- 3.4.6 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

```
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(续)

- 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(续)

- 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(续)

- 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。

嵌套查询(续)

- 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
CS

带有IN谓词的子查询（续）

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
200215121	李勇	CS
200215122	刘晨	CS

带有IN谓词的子查询（续）

- 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40]查询选修了课程名为“信息系统”的学生学号和姓名

– 嵌套查询

```
SELECT Sno, Sname              --③ 最后在Student关系中
 FROM Student                        --取出Sno和Sname
 WHERE Sno IN
      (SELECT Sno                    --② 然后在SC关系中找出选
       FROM  SC                          --修了3号课程的学生学号
       WHERE  Cno IN
            (SELECT Cno              --① 首先在Course关系中找出“信
             FROM Course                 --息系统”的课程号，结果为3号
             WHERE Cname= '信息系统'));
```

带有IN谓词的子查询（续）

结果：

Sno	Sname
200215121	李勇
200215122	刘晨

带有IN谓词的子查询（续）

- 连接查询

```
SELECT student.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

- 使用范围
  - 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39]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）

- 子查询一定要跟在比较符之后

  错误的例子：(SQL Server 可以使用)

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

带有比较运算符的子查询（续）

[例41]查询每个学生超过他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)
```

带有比较运算符的子查询（续）

- 上例中的 x 是SC的别名，称为元组变量，可以用来表示SC的一个元组。内层查询是求一个学生所有选修课程平均成绩的。具体是哪个学生与父查询相关，这类查询就是相关子查询。
- 分析该查询的执行过程（P106）

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44]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= ’1’，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
    WHERE Sno = Student.Sno AND
                Cno = '1');
```

求解过程

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= '1';
```

带有EXISTS谓词的子查询（续）

[例45]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');
```

此例用连接运算难于实现

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39]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）

- 4. 相关子查询的效率
  - 由于带EXISTS量词的相关子查询只关心内层查询是否有返回值，并不需要查具体值，因此其效率并不一定低于其他形式的查询。

例39：不相关子查询的效率高于相关子查询的效率

带有EXISTS谓词的子查询（续）

相关子查询的效率可能高于连接查询

例：查询选修了课程的学生姓名

法一：

```
SELECT Sname
FROM Student
WHERE EXISTS
   (SELECT *
    FROM SC
    WHERE Sno=Student.Sno);
```

带有EXISTS谓词的子查询（续）

法二：

```
SELECT Sname
FROM Student, SC
WHERE Student.Sno=SC.Sno;
```

带有EXISTS谓词的子查询（续）

法三：

```
SELECT Sname
FROM Student
WHERE sno in
   (SELECT distinct sno
    FROM SC);
```

带有EXISTS谓词的子查询（续）

- 5.用EXISTS/NOT EXISTS实现全称量词(难点)
  - 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46］查询选修了全部课程的学生姓名。

```
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）

6. 用EXISTS/NOT EXISTS实现逻辑蕴函(难点)
   – 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47］ 查询至少选修了学生200215122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200215122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200215122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 DISTINCT Sno
   FROM SC SCX
   WHERE NOT EXISTS
        (SELECT *
         FROM SC SCY
         WHERE SCY.Sno = '200215122'  AND
              NOT EXISTS
                   (SELECT *
                    FROM SC SCZ
                    WHERE SCZ.Sno=SCX.Sno AND
                                SCZ.Cno=SCY.Cno));
```

带有EXISTS谓词的子查询(续）

- 用NOT EXISTS谓词表示（最后一个条件，使用SCX）：

```
SELECT DISTINCT Sno
   FROM SC SCX
   WHERE NOT EXISTS
        (SELECT *
         FROM SC SCY
         WHERE SCY.Sno = '200215122'  AND
              NOT EXISTS
                   (SELECT *
                    FROM SC SCZ
                    WHERE SCZ.Sno=SCX.Sno AND
                                SCZ.Cno=SCX.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 基于派生表的查询
- 3.4.6 SELECT 语句的一般格式

3.4.4 集合查询

- 标准SQL直接支持的集合操作种类
  - 并操作(UNION)

- 一般商用数据库支持的集合操作种类
  - 并操作(UNION)
  - 交操作(INTERSECT)
  - 差操作(MINUS)　　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48] 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49] 查询选修了课程1或者选修了课程2的学生。

方法一：

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

并操作（续）

方法二：

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

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50] 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
```

2005

2. 交操作

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51] 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');
```

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。
- T-SQL2005中，差操作使用EXCEPT实现。

3. 差操作

[例52] 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;
```

3. 差操作

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

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（可以使用属性名）

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;
```

对集合操作结果的排序（续）

[例54]

```
SELECT Sname, Sage, Sdept
FROM Student
UNION  ALL
SELECT H_Sname, H_Sage, H_Sdept
FROM History_Student
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 基于派生表的查询
- 3.4.6 SELECT 语句的一般格式

3.4.5 基于派生表的查询

- 子查询不仅可以出现在Where子句中，还可以出现在From子句中，这时子查询生成的临时派生表成为主查询的查询对象。（该方法也称为内嵌视图查询）
- 【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,(select sno,avg(grade) from sc group by sno)
                as avg_sc(avg_sno,avg_grade)
where sc.sno = avg_sc.avg_sno
      and sc.grade>=avg_sc.avg_grade
```

3.4.5 基于派生表的查询

- 【例】查询所有选修了1号课程的学生姓名。

```
select sname
from student,(select sno  from sc where cno='1')
                        as sc1
where student.sno = sc1.sno
```

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 基于派生表的查询
- 3.4.6 SELECT 语句的一般格式

3.4.6 SELECT语句的一般格式

- 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]] …];

小结（续）

- 目标列表达式
  - 目标列表达式格式

  (1) [<表名>.]*

  (2) [<表名>.]<属性列名表达式>[，[<表名>.]<属性列名表达式>] …

  <属性列名表达式>：由属性列、作用于属性列的集函数和常量的任意算术运算（+，-，*，/）组成的运算公式。

小结（续）

– 集函数格式

COUNT
SUM
AVG
MAX
MIN

} ([DISTINCT|ALL] <列名>)

COUNT ([DISTINCT|ALL] *)

小结（续）

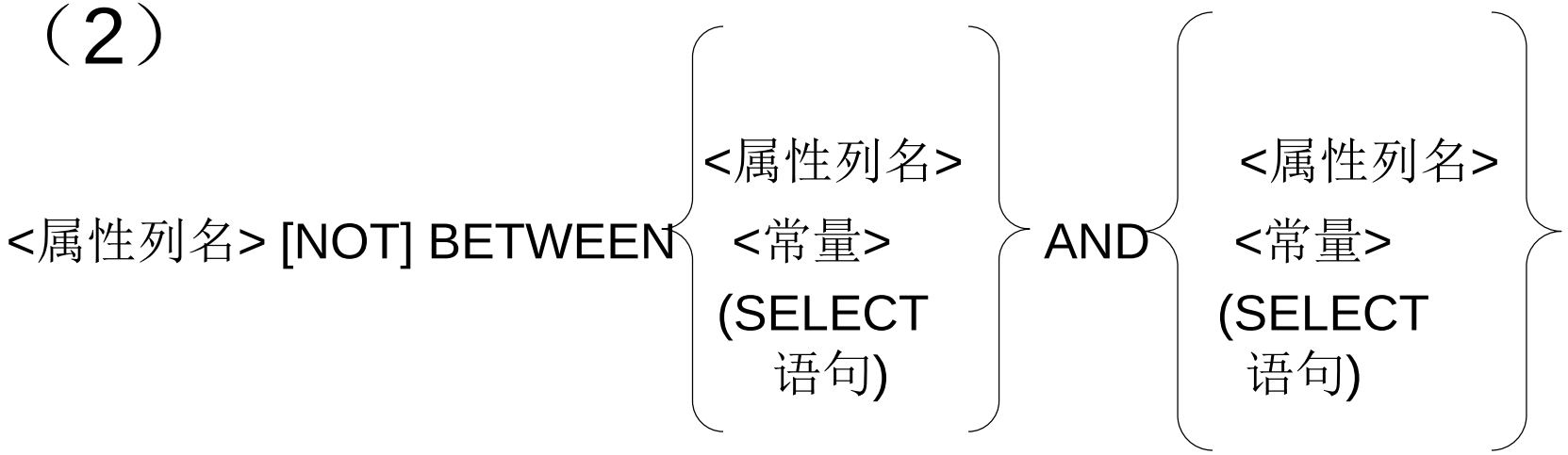
- 条件表达式格式

（1）

<属性列名> θ { <属性列名> | <常量> | [ANY|ALL] (SELECT语句) }

小结（续）

（2）



小结（续）

（3）

<属性列名> [NOT] IN { (<值1>[，<值2>] …) | (SELECT语句) }

小结（续）

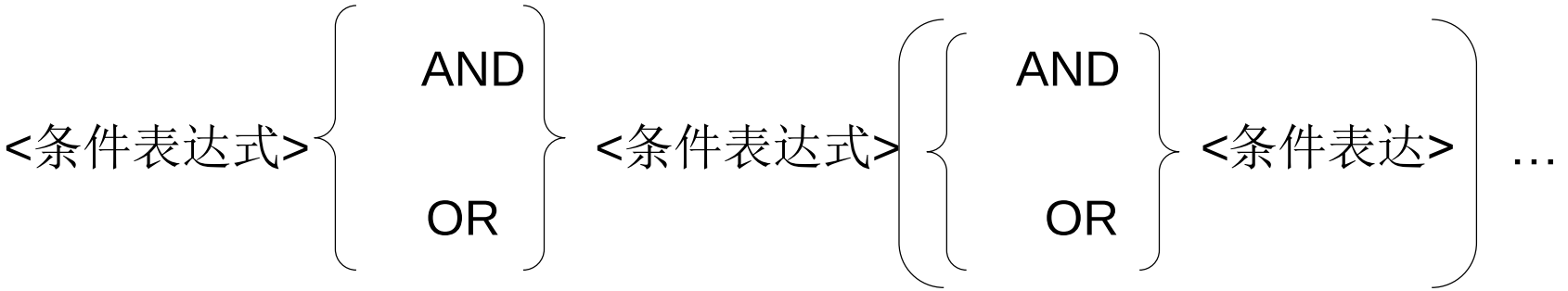
(4) <属性列名> [NOT] LIKE <匹配串>

(5) <属性列名> IS [NOT] NULL

(6) [NOT] EXISTS (SELECT语句)

小结（续）

（7）



小结（续）

- 教材中的4个重要的查询示例：
  - 查询每个学生超过其自己平均成绩的课程（p106）
  - 查询没有选修‘1’号课程的人（p110）
  - 查询选修了全部课程的人（p110）
  - 查询至少选修了“某某”所选全部课程的人（p111）

第三章 关系数据库标准语言SQL

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 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）

- 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

插入单个元组（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

插入单个元组（续）

[例1] 将一个新学生记录（学号：200215128；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 ('200215128', '陈冬', '男', 'IS', 18);
```

插入单个元组（续）

[例2] 将学生 张成民 的信息插入表中。

```
INSERT
INTO  Student
VALUES ('200215126' , '张成民' , '男' , 18 , 'CS');
```

插入单个元组（续）

[例3] 插入一条选课记录('200215128','1 ')。

```
INSERT
INTO SC(Sno,Cno)
VALUES ('200215128','1');
```

新插入的记录在Grade列上取空值

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）

- 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

插入子查询结果（续）

[例4] 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*/
```

插入子查询结果（续）

第二步：插入数据

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

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）

- 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
  - 一些DBMS规定主码不允许修改
  - 参照完整性
  - 用户定义的完整性
    - NOT NULL约束
    - UNIQUE约束
    - 值域约束

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5] 将学生200215121的年龄改为22岁。

```
UPDATE  Student
SET Sage=22
WHERE  Sno='200215121';
```

2. 修改多个元组的值

[例6] 将所有学生的年龄增加1岁。

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

3. 带子查询的修改语句

[例7] 将计算机科学系全体学生成绩置零。

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

带子查询的修改语句（续）

[例7] 解法2

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

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

- 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
    - 缺省表示要删除表中的所有元组

删除数据(续)

- DBMS在执行删除语句时会检查所删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

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200215128的学生记录。

```
DELETE
FROM Student
WHERE Sno='200215128';
```

2. 删除多个元组的值

[例9] 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10] 删除计算机科学系所有学生的选课记录。

```
DELETE
FROM SC
WHERE  'CS'=
    (SELECT Sdept
     FROM Student
     WHERE Student.Sno=SC.Sno);
```

第三章 关系数据库标准语言SQL

3.6 空值的处理

- 空值的产生
- 空值的判断
- 空值的约束条件
- 空值的运算

第三章 关系数据库标准语言SQL

3.7 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
  - 基于视图的操作
    - 定义视图(DDL)
      - 建立
      - 定义基于该视图的新视图
      - 删除
    - 查询(DML)
    - 受限更新(DML)

3.7 视　图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用

3.7 视　图

3.7.1 定义视图

1. 建立视图

2. 删除视图

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]；

建立视图（续）

- 组成视图的属性列名或全部省略或全部指定
  - 省略视图的各个属性列名，则隐含该视图由子查询中SELECT子句目标列中的诸字段组成。
  - 必须明确指定组成视图的所有列名的情形
    (1) 某个目标列不是单纯的属性名，而是集函数或列表达式
    (2) 目标列为 *
    (3) 多表连接时选出了几个同名列作为视图的字段
    (4) 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）

- 子查询
  - 不含ORDER BY子句和DISTINCT短语的SELECT语句
- 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）

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。只是在对视图查询时，才按视图的定义从基本表中将数据查出。

建立视图（续）

- 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

建立视图（续）

- 行列子集视图
  - 从单个基本表导出
  - 只是去掉了基本表的某些行和某些列，但保留了码

建立视图（续）

[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';
```

行列子集视图视图IS_Student由Sno，Sname，Sage三列组成。

建立视图（续）

- 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;
```

建立视图（续）

- 对IS_Student视图的更新操作
  - 修改操作：DBMS自动加上Sdept= 'IS'的条件
  - 删除操作：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
    - 如果没有提供Sdept属性值，则自动定义Sdept为'IS'
- 不同的DBMS系统对违背WITH CHECK OPTION的操作的处理方式不同，SQL Server的处理为拒绝

建立视图（续）

- 基于多个基表的视图

  [例3] 建立信息系统选修了1号课程的学生视图。

```
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';
```

  - 由于视图IS_S1的属性列中包含了Student表与SC表的同名列Sno，所以必须在视图名后面明确说明视图的各个属性列名。

建立视图（续）

- 基于视图的视图

  [例4] 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;
```

  – 视图IS_S2建立在视图IS_S1之上

建立视图（续）

- 带表达式的视图
  - 在设计数据库时，为了减少数据冗余，基本表中只存放基本数据，由基本数据经过各种计算派生出的数据一般是不存储的。
  - 视图中的数据并不实际存储，所以定义视图时可以根据应用的需要，设置一些派生属性列，以方便应用程序的编制。
  - 派生属性称为虚拟列。带虚拟列的视图称为带表达式的视图。
  - 带表达式的视图必须明确定义组成视图的各个属性列名。

建立视图（续）

[例5] 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,2017-Sage
        FROM  Student;
```

建立视图（续）

- 分组视图
  - 用带集函数和GROUP BY子句的查询来定义的视图称为分组视图
  - 分组视图必须明确定义组成视图的各个属性列名

建立视图（续）

[例6] 将学生的学号及他的平均成绩定义为一个视图。

```
CREATE  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7]将Student表中所有女生记录定义为一个视图

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

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
                    (F_sno,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。

2. 删除视图

- 语句格式

  DROP VIEW <视图名> [CASCADE] ;

  - 该语句从数据字典中删除指定的视图定义。
  - 由该视图导出的其他视图可以使用CASCADE级联删除。
  - 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8] 删除视图BT_S：DROP VIEW BT_S ;

删除视图IS_S1：DROP VIEW IS_S1 ;

由于在 IS_S1上还定义了IS_S2视图，该语句被拒绝执行，可以使用CASCADE进行级联删除。

DROP VIEW IS_S1 CASCADE;

3.7 视　图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用

3.7.2 查询视图

- 从用户角度而言，查询视图与查询基本表的方法相同

查询视图（续）

[例9] 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）

– 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10] 查询信息系选修了1号课程的学生

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

查询视图（续）

[例11] 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;
```

查询视图（续）

S_G视图定义：

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

查询视图（续）

正确转换：

```
SELECT  Sno,AVG(Grade)
FROM  SC
GROUP BY Sno
HAVING AVG(Grade)>85
```

3.7 视　图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用

3.7.3 更新视图

- 从用户角度而言，更新视图与更新基本表的方法相同
- 定义视图时指定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数据进行增加、删除、修改时，操作不属于视图范围内的基本表数据。

更新视图（续）

[例12] 将信息系学生视图IS_Student中学号为200215122的学生姓名改为“刘辰”。

```
UPDATE  IS_Student
SET  Sname= '刘辰'
WHERE  Sno= '200215122' ;
```

更新视图（续）

转换后的查询语句为：

```
UPDATE Student
SET Sname= '刘辰'
WHERE Sno= ' 200215122 ' AND Sdept= 'IS'；
```

更新视图（续）

[例13] 向信息系学生视图IS_S中插入一个新的学生记录，其中学号为200215129，姓名为赵新，年龄为20岁。

```
INSERT
INTO IS_Student
VALUES('200215129','赵新',20) ;
```

更新视图（续）

转换为对基本表的更新：

```
INSERT
INTO  Student(Sno,Sname,Sage,Sdept)
VALUES('200215129','赵新',20,'IS' ) ;
```

更新视图（续）

[例14] 删除计算机系学生视图CS_S中学号为200215129的记录

```
DELETE
FROM IS_Student
WHERE Sno= '200215129' ;
```

更新视图（续）

转换为对基本表的更新：

```
DELETE
FROM Student
WHERE Sno= '200215129' AND Sdept= 'IS' ;
```

更新视图（续）

- DBMS对视图更新的限制
  - 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。

/*凡是带有聚类函数的视图都不可以直接更新*/

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）

S_G 中，“平均成绩” Gavg属性列为导出列

对于如下更新语句：

```
UPDATE S_G
SET Gavg=90
WHERE Sno= '200215121' ;
```

无法将其转换成对基本表SC的更新。

更新视图（续）

- 视图的可更新性
  - 行列子集视图是可更新的。
  - 除行列子集视图外，还有些视图理论上是可更新的，但它们的确切特征还是尚待研究的课题。
  - 还有些视图从理论上是不可更新的。

更新视图（续）

- 实际系统对视图更新的限制
  - 允许对行列子集视图进行更新
  - 对其他类型视图的更新不同系统有不同限制

  DB2对视图更新的限制：

  (1) 若视图是由两个以上基本表导出的，则此视图不允许更新。

更新视图（续）

(2) 若视图的字段来自字段表达式或常数，则不允许对此视图执行INSERT和UPDATE操作，但允许执行DELETE操作。

(3) 若视图的字段来自集函数，则此视图不允许更新。

(4) 若视图定义中含有GROUP BY子句，则此视图不允许更新。

(5) 若视图定义中含有DISTINCT短语，则此视图不允许更新。

更新视图（续）

(6) 若视图定义中有嵌套查询，并且内层查询的FROM子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。

例：视图GOOD_SC(修课成绩在平均成绩之上的元组)

```
CREATE VIEW GOOD_SC
    AS SELECT  Sno , Cno , Grade
       FROM SC
       WHERE Grade >
          (SELECT AVG(Grade)
           FROM  SC) ;
```

更新视图（续）

(7) 一个不允许更新的视图上定义的视图也不允许更新。

3.7 视　图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用

3.7.4 视图的作用

- 视图最终是定义在基本表之上的，对视图的一切操作最终也要转换为对基本表的操作。而且对于非行列子集视图进行查询或更新时还有可能出现问题。

视图的作用（续）

- 合理使用视图能够带来许多好处
  - 1. 视图能够简化用户的操作
  - 2. 视图使用户能以多种角度看待同一数据
  - 3. 视图对重构数据库提供了一定程度的逻辑独立性
  - 4. 视图能够对机密数据提供安全保护
  - 5. 适当的使用视图可以更清晰的表达查询

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

3. 视图对重构数据库提供了一定程度的逻辑独立性

- 物理独立性与逻辑独立性的概念

- 视图在一定程度上保证了数据的逻辑独立性

视图对重构数据库提供了一定程度的逻辑独立性（续）

例：数据库逻辑结构发生改变

将学生关系

Student(Sno , Sname , Ssex , Sage , Sdept)

“垂直”地分成两个基本表：

SX(Sno , Sname , Sage)

SY(Sno , Ssex , Sdept)

视图对重构数据库提供了一定程度的逻辑独立性（续）

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

视图对重构数据库提供了一定程度的逻辑独立性（续）

- 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。

4. 视图能够对机密数据提供安全保护

- 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

- 通过WITH CHECK OPTION对关键数据定义操作时间限制

5. 适当的使用视图可以更清晰的表达查询

- 对于经常需要执行的查询，可以先定义一个视图，由此视图再完成最终的查询。

第三章 关系数据库标准语言SQL

3.8 小结

- 数据定义
- 数据查询
- 数据更新
- 数据控制