



# 第一篇 基础篇

## 第三章 关系数据库标准语言SQL(1)

福州大学数计学院

程 烨

chengye@fzu.edu.cn

2013年3月25日



## 第三章 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

### 3.7 小结

## 3.1 SQL概述

- ❖ SQL (Structured Query Language) 结构化查询语言，是关系数据库的标准语言
- ❖ SEQUEL (Structured English Query Language )
- ❖ S-Q-L或sequel
- ❖ SQL是一个通用的、功能极强的关系数据库语言

## SQL概述（续）

- ❖ **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，在IBM公司的System R上实现。
- ❖ 1986年被美国国家标准局（ANSI）批准为关系数据库语言的美国标准。
- ❖ 1987年国际标准化组织（ISO）通过这一标准

# SQL标准的进展过程

标准	大致页数	发布日期
■ SQL/86		1986.10
■ SQL/89(FIPS 127-1)	120页	1989年
■ SQL/92	622页	1992年
■ SQL99	1700页	1999年
■ SQL2003	360页	2003年
■ SQL2006		
■ SQL2008		

## 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

### 3.1.2 SQL的特点

- ❖ 每个数据库管理系统(DBMS)都基于某种数据模型，都为用户提供一个数据库语言，以便对数据库中的数据进行定义、操纵与控制。
- ❖ 数据库语言依据数据模型不同而不同。



## 3.1.2 SQL的特点

### 1.综合统一

❖ 非关系模型的数据语言一般分为：

- 模式数据定义语言（模式DDL）
- 外模式数据定义语言（外模式DDL，子模式DDL）
- 数据存储有关的描述语言（DSDL）
- 数据操纵语言（DML）

当用户数据库投入运行后，如果要修改模式，必须停止现有数据运行，转储数据，修改模式并编译后再重装数据库

# 1.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）功能于一体。
- 可以独立完成数据库生命周期中的全部活动：
  - 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
- 用户数据库投入运行后，可根据需要随时逐步修改模式，不影响数据的运行。
- 数据操作符统一，查找、插入、删除、更新等只需一种操作符

## 2.高度非过程化

- ❖ 非关系数据模型的数据操纵语言“**面向过程**”，  
必须制定存取路径
- ❖ **SQL**只要提出“做什么”，无须了解存取路径，  
没有必要一步步地告诉计算机“如何”去做。  
存取路径的选择以及**SQL**的操作过程由系统自动  
完成。

### 3.面向集合的操作方式

- ❖ 非关系数据模型采用面向记录的操作方式，操作对象是一条记录
- ❖ SQL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.以同一种语法结构提供多种使用方式

### ❖ SQL是独立的语言

能够独立地用于联机交互的使用方式

### ❖ SQL又是嵌入式语言

SQL能够嵌入到高级语言（例如C，C++，Java）

程序中，供程序员设计程序时使用

## 5.语言简洁，易学易用

- ❖ SQL功能极强，完成核心功能只用了9个动词。

表 3.1 SQL 语言的动词

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

## 3.1 SQL概述

❖ 3.1.1 SQL 的产生与发展

❖ 3.1.2 SQL的特点

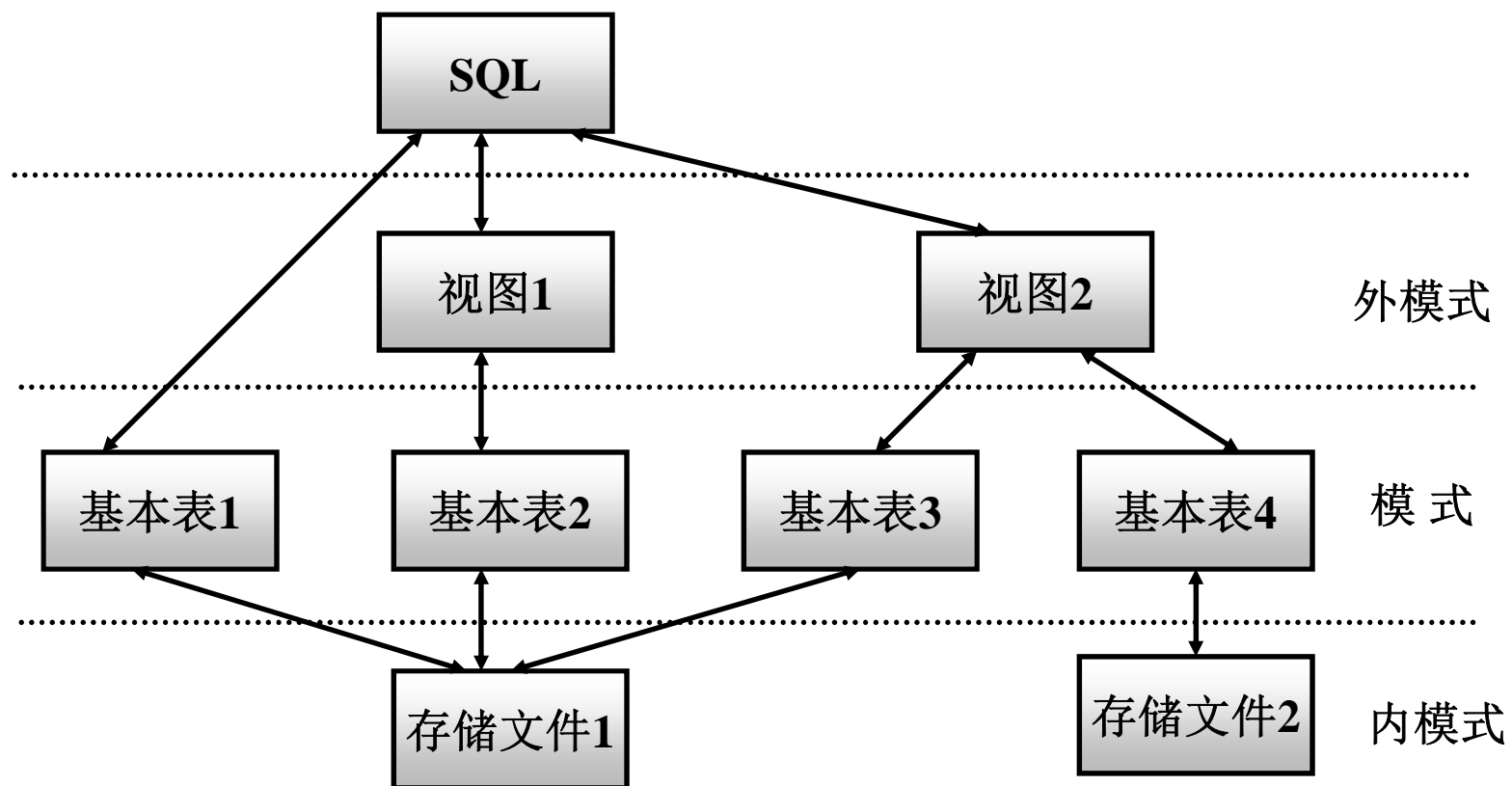
❖ 3.1.3 SQL的基本概念

### 3.1.3 SQL的基本概念

- ❖ SQL支持关系数据库三级模式结构
- ❖ 外模式对应视图（**View**）和部分基本表（**Base Table**）
- ❖ 模式对应基本表
- ❖ 内模式对应存储文件（**Stored File**）



## SQL的基本概念（续）



# SQL的基本概念（续）

## ❖ 基本表

- 本身独立存在的表
- **SQL**中一个关系就对应一个基本表
- 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

## ❖ 存储文件

- 逻辑结构组成了关系数据库的内模式
- 物理结构是任意的，对用户透明

## ❖ 视图

- 从一个或几个基本表导出的表
- 数据库中只存放视图的定义而不存放视图对应的数据
- 视图是一个虚表
- 用户可以在视图上再定义视图



## 第三章 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

### 3.7 小结

## 3.2 学生-课程数据库

❖ 首先定义学生-课程模式S-T,

学生-课程数据库包括:

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)

## Student表

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所 在 系 Sdept
200215121	李勇	男	20	CS
200215122	刘晨	女	19	CS
200215123	王敏	女	18	MA
200515125	张立	男	19	IS

学生-课程数据库(a)

## Course表

课程号 <b>Cno</b>	课程名 <b>Cname</b>	先行课 <b>Cpno</b>	学分 <b>Ccredit</b>
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	<b>PASCAL</b> 语言	6	4

学生-课程数据库(b)

## SC表

学 号 Sno	课程号 Cno	成绩 Grade
200215121	1	92
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80

学生-课程数据库(c)



## 第三章 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

### 3.7 小结



## 3.3 数据定义

❖ SQL的数据定义功能包括模式定义、表定义、视图和索引的定义

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
模式	<b>CREATE SCHEMA</b>	<b>DROP SCHEMA</b>	
表	<b>CREATE TABLE</b>	<b>DROP TABLE</b>	<b>ALTER TABLE</b>
视 图	<b>CREATE VIEW</b>	<b>DROP VIEW</b>	
索 引	<b>CREATE INDEX</b>	<b>DROP INDEX</b>	

❖ SQL通常不提供修改模式定义、修改视图定义和修改索引定义的操作  
用户如果想修改这些对象，只能先将它们删除掉，然后再重建

## 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>

■ 如果没有指定<模式名>，那么<模式名>隐含为<用户名>

■ 用户必须拥有DBA权限，或者获得了DBA授予的

CREATE SCHEMA的权限

## 定义模式（续）

[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

[例2]CREATE SCHEMA AUTHORIZATION WANG;

<模式名>隐含为用户名WANG

## 定义模式（续）

- ❖ 定义模式实际上定义了一个命名空间
- ❖ 在这个空间中可以定义该模式包含的数据库对象，例如基本表、视图、索引等。
- ❖ 在CREATE SCHEMA中可以接受CREATE TABLE，CREATE VIEW和GRANT字句。

CREATE SCHEMA <模式名> AUTHORIZATION <用户名>[<表定义字句>|<视图定义字句>|<授权定义字句>]

## 定义模式（续）

[例3]

```
CREATE SCHEMA TEST AUTHORIZATION ZHANG  
CREATE TABLE TAB1 (COL1 SMALLINT,  
                     COL2 INT,  
                     COL3 CHAR (20)  
                     COL4 NUMERIC (10, 3)  
                     COL5 DECIMAL (5, 2)  
                     ) ;
```

为用户**ZHANG**创建了一个模式**TEST**，并在其中定义了一个表**TAB1**。

## 二、删除模式

### ❖ 删除模式语句

- DROP SCHEMA <模式名> <CASCADE|RESTRICT>

- CASCADE(级联)

删除模式的同时把该模式中所有的数据库对象全部删除

- RESTRICT(限制)

- 如果该模式中定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行。
- 当该模式中没有任何下属的对象是才能执行。

## 删除模式（续）

[例4] DROP SCHEMA ZHANG CASCADE;

删除模式ZHANG

同时该模式中定义的表TAB1也被删除



## 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
- 如果完整性约束条件涉及到该表的多个属性列，则必须定义在表级上，否则既可以定义在列级也可以定义在表级。

# 学生表Student

[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。其中学号是主码，并且姓名取值也唯一。

主码

```
CREATE TABLE Student
```

```
(Sno CHAR(9) PRIMARY KEY, /* 列级完整性约束条件*/  
  Sname CHAR(20) UNIQUE, /* Sname取唯一值*/  
  Ssex CHAR(2),  
  Sage SMALLINT,  
  Sdept CHAR(20)  
);
```

## 课程表Course

[例6] 建立一个“课程”表Course

```
CREATE TABLE Course
```

```
( Cno    CHAR(4) PRIMARY KEY,
```

```
  Cname  CHAR(40),
```

```
  Cpno   CHAR(4),
```

```
  Ccredit SMALLINT,
```

```
FOREIGN KEY Cpno REFERENCES Course(Cno)
```

```
);
```

先修课

Cpno是外码  
被参照表是Course  
被参照列是Cno

## 学生选课表SC

[例7] 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*/
```

```
);
```

## 二、数据类型

- ❖ SQL中域的概念用数据类型来实现
- ❖ 定义表的各个属性时需要指明其数据类型及长度
- ❖ 选用哪种数据类型
  - 取值范围
  - 要做哪些运算

## 数据类型（续）

数据类型	含义
<b>CHAR(n)</b>	长度为 <b>n</b> 的定长字符串
<b>VARCHAR(n)</b>	最大长度为 <b>n</b> 的变长字符串
<b>INT</b>	长整数（也可以写作 <b>INTEGER</b> ）
<b>SMALLINT</b>	短整数
<b>NUMERIC(p, d)</b>	定点数，由 <b>p</b> 位数字（不包括符号、小数点）组成，小数后面有 <b>d</b> 位数字
<b>REAL</b>	取决于机器精度的浮点数
<b>Double Precision</b>	取决于机器精度的双精度浮点数
<b>FLOAT(n)</b>	浮点数，精度至少为 <b>n</b> 位数字
<b>DATE</b>	日期，包含年、月、日，格式为 <b>YYYY-MM-DD</b>
<b>TIME</b>	时间，包含一日的时、分、秒，格式为 <b>HH:MM:SS</b>

### 三、模式与表

- ❖ 每一个基本表都属于某一个模式
- ❖ 一个模式包含多个基本表
- ❖ 创建基本表（其他数据库对象也一样）时，若没有指定模式，系统根据搜索路径来确定该对象所属的模式
- ❖ **RDBMS**会使用模式列表中第一个存在的模式作为数据库对象的模式名



## 四、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> ];

- <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 (MODIFY)子句：用于修改列名和数据类型

## 修改基本表（续）

[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。

[例9]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;
```

[例10]增加课程名称必须取唯一值的约束条件。

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

## 五、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**CHECK**, **FOREIGN KEY**等约束），不能有视图，不能有触发器，不能有存储过程或函数等。
- 如果存在这些依赖该表的对象，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)

### [例11] 删除Student表

```
DROP TABLE Student CASCADE ;
```

- 基本表定义被删除，数据和表定义被删除
- 表上建立的索引、视图、触发器等根据具体的数据库管理系统的规定来处理

## 删除基本表（续）

[例12] 若表上建有视图，选择**RESTRICT**时表不能删除；**CASCADE**时可以删除表，视图也自动被删除

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

```
DROP TABLE Student RESTRICT;  
--ERROR: cannot drop table Student because other  
   objects depend on it
```

## 删除基本表（续）

[例12]（续）

```
DROP TABLE Student CASCADE;
```

--**NOTICE**: drop cascades to view IS\_Student

```
SELECT * FROM IS_Student;
```

--**ERROR**: relation " IS\_Student " does not exist

# 删除基本表（续）

DROP TABLE时，SQL99与3个RDBMS的处理策略比较

序号	标准及主流数据库的处理方式 依赖基本表的对象	SQL99		Kingbase ES		ORACLE 9i		MS SQL SERVER 2000
		R	C	R	C		C	
1.	索引	无规定		√	√	√	√	√
2.	视图	×	√	×	√	√ 保留	√ 保留	√ 保留
3.	DEFAULT, PRIMARY KEY, CHECK（只含该表的列）NOT NULL 等约束	√	√	√	√	√	√	√
4.	Foreign Key	×	√	×	√	×	√	×
5.	TRIGGER	×	√	×	√	√	√	√
6.	函数或存储过程	×	√	√ 保留	√ 保留	√ 保留	√ 保留	√ 保留

R表示RESTRICT, C表示CASCADE

'×'表示不能删除基本表, '√'表示能删除基本表, '保留'表示删除基本表后, 还保留依赖对象

## 删除基本表（续）

### ❖ 比较结果：

- 索引：删除基本表后，这3个RDBMS都自动删除该基本表上已经建立的所有索引
- 视图：
  - ORACLE 9i与SQL SERVER 2000是删除基本表后，还保留此基本表上的视图定义，但是已经失效。
  - KingbaseES分两种情况，若删除基本表时带RESTRICT选项，则不可以删除基本表；若删除基本表时带CASCADE选项，可以删除基本表，同时也删除视图；KingbaseES的这种策略符合SQL99标准



## 删除基本表（续）

- 存储过程和函数：删除基本表后，这3个数据库产品都不自动删除建立在此基本表上的存储过程和函数，但是已经失效
- 触发器或者被其他基本表的约束所引用（**CHECK**，**FOREIGN KEY**等），可以从比较表中得到这3个系统的处理策略

## 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

## 索引

- ❖ RDBMS中索引一般采用B+树、HASH索引来实现
  - B+树索引具有动态平衡的优点
  - HASH索引具有查找速度快的特点
- ❖ 索引是关系数据库的内部实现技术，属于内模式的范畴
- ❖ CREATE INDEX语句定义索引时，可以定义索引是唯一索引、非唯一索引或聚簇索引
- ❖ 采用B+树，还是HASH索引则由具体的RDBMS来决定

# 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>] ]...);

- <表名>: 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
- <次序>: 指定索引值的排列次序, 升序: ASC, 降序: DESC。缺省值: ASC
- UNIQUE: 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, 索引项的顺序与表中记录的物理顺序一致的索引组织

## 建立索引（续）

[例13] 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。
- ❖ 在最经常查询的列上建立聚簇索引以提高查询效率
- ❖ 一个基本表上最多只能建立一个聚簇索引
- ❖ 经常更新的列不宜建立聚簇索引

## 建立索引（续）

[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

❖ **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15] 删除Student表的Stusname索引

```
DROP INDEX Stusname;
```





## 第三章 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

### 3.7 小结

# 数据查询

## ❖ 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 ] ];

# 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 数据查询

- ❖ 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**语句的一般形式

## 3.4.1 单表查询

❖ 查询仅涉及一个表：

- 一、 选择表中的若干列
- 二、 选择表中的若干元组
- 三、 ORDER BY子句
- 四、 聚集函数
- 五、 GROUP BY子句

## 一、选择表中的若干列

### ❖ 1. 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;
```

- <目标列表表达式>中各个列的先后顺序可以与表中的顺序不一致

## 2. 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 \*

[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>可以为:

- 算术表达式
- 字符串常量
- 函数
- 列别名



## 查询经过计算的值（续）

[例4] 查全体学生的姓名及其出生年份。

```
SELECT Sname, 2000-Sage  
FROM Student;
```

输出结果：

Sname	2000-Sage
-------	-----------

李勇	1984
刘晨	1985
王敏	1986
张立	1985

## 查询经过计算的值（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: ', 2000-Sage, ISLOWER(Sdept)
FROM Student;
```

输出结果：

```
Sname 'Year of Birth:' 2000-Sage ISLOWER(Sdept)
```

---

李勇	Year of Birth:	1984	cs
刘晨	Year of Birth:	1985	is
王敏	Year of Birth:	1986	ma
张立	Year of Birth:	1985	is

## 查询经过计算的值（续）

- ❖ 使用列别名改变查询结果的列标题:

```
SELECT Sname NAME, 'Year of Birth: ' BIRTH,  
       2000-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	is
王敏	Year of Birth:	1986	ma
张立	Year of Birth:	1985	is

## 3.4.1 单表查询

❖ 查询仅涉及一个表：

- 一、 选择表中的若干列
- 二、 选择表中的若干元组
- 三、 ORDER BY子句
- 四、 聚集函数
- 五、 GROUP BY子句

## 二、选择表中的若干元组

### ❖ 1. 消除取值重复的行

如果没有指定**DISTINCT**关键词，则缺省为**ALL**

[例6] 查询选修了课程的学生学号。

```
SELECT Sno FROM SC;
```

等价于：

```
SELECT ALL Sno FROM SC;
```

执行上面的**SELECT**语句后，结果为：

Sno

---

200215121

200215121

200215121

200215122

200215122

## 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：

Sno
200215121
200215122

## 消除取值重复的行（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

### 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（逻辑运算）	AND, OR, NOT



## (1) 比较大小

❖ 比较运算符:

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

[例7] 查询计算机科学系全体学生的名单。

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

## 比较大小（续）

- ❖ [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;
```

- ❖ [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

❖ 谓词: BETWEEN ... AND ...

NOT BETWEEN ... AND ...

- 用来查找属性值在（或不在）指定范围内的元组
- BETWEEN后是范围的下限（即低值）
- AND后是范围的上限（即高值）

## 确定范围（续）

[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

❖ 谓词：**IN <值表>**, **NOT IN <值表>**

[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) 字符匹配

❖ 谓词： [NOT] LIKE ‘<匹配串>’ [ESCAPE ‘<换码字符>’]

■ 含义： 查找指定的属性列值与<匹配串>相匹配的元组

■ 匹配串： 固定字符串或含通配符的字符串

■ 当匹配串为固定字符串时，

➤ 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。



## 字符匹配（续）

### 1) 匹配串为固定字符串

[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';
```

## 字符匹配（续）

### 2) 匹配串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16】 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__'; （两个下划线_或一个下划线_）
```

## 字符匹配（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%';
```

[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%';
```

## 字符匹配（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' ESCAPE '\'
```

[例20] 查询以"DB\_"开头，且倒数第3个字符为i的课程的具体情况。

```
SELECT *  
FROM Course  
WHERE Cname LIKE 'DB\__%i\__' ESCAPE '\\';
```

➤ **ESCAPE '\'** 表示 “ \ ” 为换码字符

## (5) 涉及空值的查询

- 谓词：IS NULL 或 IS NOT NULL
- “IS” 不能用 “=” 代替

[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

❖ 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 ...

## 多重条件查询（续）

[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;
```

## 多重条件查询（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 ';
```



## 3.4.1 单表查询

❖ 查询仅涉及一个表：

- 一、 选择表中的若干列
- 二、 选择表中的若干元组
- 三、 **ORDER BY**子句
- 四、 聚集函数
- 五、 **GROUP BY**子句

### 三、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

#### ❖ 当排序列含空值时

- ASC：排序列为空值的元组最后显示
- DESC：排序列为空值的元组最先显示

## ORDER BY子句（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;
```

[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;
```

## 3.4.1 单表查询

❖ 查询仅涉及一个表：

- 一、 选择表中的若干列
- 二、 选择表中的若干元组
- 三、 ORDER BY子句
- 四、 聚集函数
- 五、 GROUP BY子句

## 四、聚集函数

### ❖ 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>)

- 最大最小值

MAX ([DISTINCT|ALL] <列名>)

MIN ([DISTINCT|ALL] <列名>)

## 聚集函数（续）

- ❖ **DISTINCT**: 取消指定列中的重复值
- ❖ **ALL**: 不取消重复值（缺省值）
- ❖ 在聚集函数遇到空值时，除**COUNT（\*）**外，都跳过空值而只处理非空值

## 聚集函数（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;
```

[例28] 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）

[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 ';
```

[例30] 查询学生200215012选修课程的总学分数。

```
SELECT SUM(Ccredit)
FROM SC, Course
WHERE Sno='200215012' AND SC.Cno=Course.Cno;
```



## 3.4.1 单表查询

❖ 查询仅涉及一个表：

- 一、 选择表中的若干列
- 二、 选择表中的若干元组
- 三、 **ORDER BY**子句
- 四、 聚集函数
- 五、 **GROUP BY**子句

## 五、GROUP BY子句

### ❖ 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
- 作用对象是查询的中间结果表
- 按指定的一列或多列值分组，值相等的为一组

## GROUP BY子句（续）

[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：

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

## GROUP BY子句（续）

[例32] 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）

### ❖ 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 数据查询

- ❖ 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**语句的一般形式

## 3.4.2 连接查询

❖ 连接查询：同时涉及多个表的查询

❖ 连接条件或连接谓词：用来连接两个表的条件

一般格式：

■ [

■ 比较运算符：=、>、<、>=、<=、!=

■ [

❖ 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。
- 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



# 连接操作的执行过程

## ❖ 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

## 连接操作的执行过程

- 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中的第一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

# 连接操作的执行过程

## ❖ 索引连接(INDEX-JOIN)

- 对表2按连接字段建立索引
- 对表1中的每个元组，依次根据其连接字段值查询表2的索引，从中找到满足条件的元组，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

## 连接查询（续）

- 一、等值与非等值连接查询
- 二、自身连接
- 三、外连接
- 四、复合条件连接

## 一、等值与非等值连接查询

- ❖ 等值连接：当连接运算符为=
- ❖ 非等值连接：使用其他运算符
- ❖ 自然连接：在等值连接中把目标列中重复的属性列去掉

[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）

查询结果：

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	CS	200215122	2	90
200215122	刘晨	女	19	CS	200215122	3	80

## 等值与非等值连接查询（续）

❖ 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;
```

## 连接查询（续）

- 一、等值与非等值连接查询
- 二、自身连接
- 三、外连接
- 四、复合条件连接



## 二、自身连接

- ❖ 自身连接：一个表与其自己进行连接
- ❖ 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;
```

## 自身连接（续）

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	Pcno
1	7
3	5
5	6

# 连接查询

- 一、等值与非等值连接查询
- 二、自身连接
- 三、外连接
- 四、复合条件连接

### 三、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

## 外连接（续）

[例 36] 改写[例33]

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

/\* 也可以使用**USING** 来去掉结果中的重复值:

```
FROM Student LEFT OUT JOIN SC USING (Sno); */
```



## 外连接（续）

执行结果：

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	CS	2	90
200215122	刘晨	女	19	CS	3	80
200215123	王敏	女	18	MA	NULL	NULL
200215125	张立	男	19	IS	NULL	NULL

## 外连接（续）

### ❖ 左外连接

- 列出左边关系（如本例**Student**）中所有的元组

### ❖ 右外连接

- 列出右边关系中所有的元组

### ❖ 全外连接

- 包含左外连接和右外连接

## 连接查询（续）

- 一、等值与非等值连接查询
- 二、自身连接
- 三、外连接
- 四、复合条件连接

## 四、复合条件连接

❖ 复合条件连接：WHERE子句中含多个连接条件

[例37]查询选修2号课程且成绩在90分以上的所有学生

```
SELECT Student.Sno, Sname  
FROM   Student, SC  
WHERE  Student.Sno = SC.Sno AND  
        /* 连接谓词*/  
        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;
```

## 3.4 数据查询

- ❖ 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**语句的一般形式

## 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)

```
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

## 嵌套查询求解方法

- ❖ 不相关子查询：子查询的查询条件不依赖于父查询
  - 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：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

### 3.4.3 嵌套查询

- 一、带有IN谓词的子查询
- 二、带有比较运算符的子查询
- 三、带有**ANY (SOME)** 或**ALL**谓词的子查询
- 四、带有**EXISTS**谓词的子查询

## 一、带有IN谓词的子查询

[例39] 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: 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）

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[例39]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）

[例40]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'
            )
      );
```

号

③ 最后在Student关系中  
取出Sno和Sname

② 然后在SC关系中找到选  
修了3号课程的学生学号

① 首先在Course关系中找到  
“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[例40]

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

### 3.4.3 嵌套查询

- 一、带有IN谓词的子查询
- 二、带有比较运算符的子查询
- 三、带有ANY（**SOME**）或**ALL**谓词的子查询
- 四、带有**EXISTS**谓词的子查询

## 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：

```
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] 找出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);
```

相关子查询

## 带有比较运算符的子查询（续）

### ❖ 可能的执行过程：

1. 从外层查询中取出SC的一个元组x，将元组x的Sno值（200215121）传送给内层查询。

```
SELECT AVG(Grade)
FROM SC y
WHERE y.Sno='200215121';
```

2. 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;
```



## 带有比较运算符的子查询（续）

3. 执行这个查询，得到

(200215121, 1)

(200215121, 3)

4. 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:

(200215121, 1)

(200215121, 3)

(200215122, 2)

### 3.4.3 嵌套查询

- 一、带有**IN**谓词的子查询
- 二、带有比较运算符的子查询
- 三、带有**ANY (SOME)** 或**ALL**谓词的子查询
- 四、带有**EXISTS**谓词的子查询

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

### 需要配合使用比较运算符

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

## 带有ANY (SOME) 或ALL谓词的子查询 (续)

[例42] 查询其他系中比计算机科学系某一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 */
```

## 带有ANY (SOME) 或ALL谓词的子查询 (续)

结果:

Sname	Sage
王敏	18
张立	19

执行过程:

1. RDBMS执行此查询时, 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[例42]

```
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)

[例43] 查询其他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）

表3.5 **ANY**（或**SOME**），**ALL**谓词与聚集函数、**IN**谓词的等价转换关系

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

## 带有ANY (SOME) 或ALL谓词的子查询 (续)

- $=ANY$ 等价于IN谓词,  $<ANY$  等价于  $<MAX$ ,  $<>ALL$ 等价于NOT IN谓词,  $<ALL$ 等价于 $<MIN$
- 用聚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

### 3.4.3 嵌套查询

- 一、带有**IN**谓词的子查询
- 二、带有比较运算符的子查询
- 三、带有**ANY** (**SOME**) 或**ALL**谓词的子查询
- 四、带有**EXISTS**谓词的子查询

# 带有EXISTS谓词的子查询(续)

## ❖ 1. 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

## ❖ 2. 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)

[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 ' );

## 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');
```

## 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(难点)

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:

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

- ### ❖ 表示: 对所有的x, 谓词P都成立; 等价于: 不存在一个x使P不成立

## 带有EXISTS谓词的子查询(续)

例：[例39]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)

[例46] 查询选修了全部课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS ——不存在一门课程
    (SELECT *
     FROM Course
     WHERE NOT EXISTS ——该生没有选修
        (SELECT *
         FROM SC
         WHERE Sno= Student.Sno
           AND Cno= Course.Cno) ) ;
```

## 带有EXISTS谓词的子查询(续)

用EXISTS/NOT EXISTS实现逻辑蕴涵(难点)

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

$$p \rightarrow q \equiv \neg p \vee q \quad \text{蕴含等值式}$$

## 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：

- 用逻辑蕴涵表达：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200215122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:

$$\begin{aligned}(\forall y)p \rightarrow q &\equiv \neg (\exists y (\neg(p \rightarrow q))) \\ &\equiv \neg (\exists y (\neg(\neg p \vee q))) \\ &\equiv \neg \exists y(p \wedge \neg q)\end{aligned}$$

- 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 ---不存在学生200215122选修的课程
    (SELECT *
     FROM SC SCY
     WHERE SCY.Sno = '200215122' AND
           NOT EXISTS ---x学生没有选修同一课程
               (SELECT *
                FROM SC SCZ
                WHERE SCZ.Sno=SCX.Sno AND
                      SCZ.Cno=SCY.Cno));
```



## 3.4 数据查询

- ❖ 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**语句的一般形式

## 3.4.4 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

❖ 某些DBMS不支持INTERSECT、EXCEPT

❖ 某些DBMS用MINUS取代了EXCEPT

## 集合查询（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;
```

- 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）

方法二：

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

## 集合查询（续）

[例49] 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）

[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
```

## 集合查询（续）

❖ [例50] 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）

[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

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



## 集合查询（续）

[例51]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 ');
```

## 集合查询（续）

[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。

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

## 集合查询（续）

[例52]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;
```

## 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**子句中也可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 数据查询

- ❖ 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**语句的一般形式

### 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>

[HAVING <条件表达式>]]

[ORDER BY <列名2> [ASC|DESC]



# 1.目标列表表达式

## ❖ 目标列表表达式格式

(1) \*

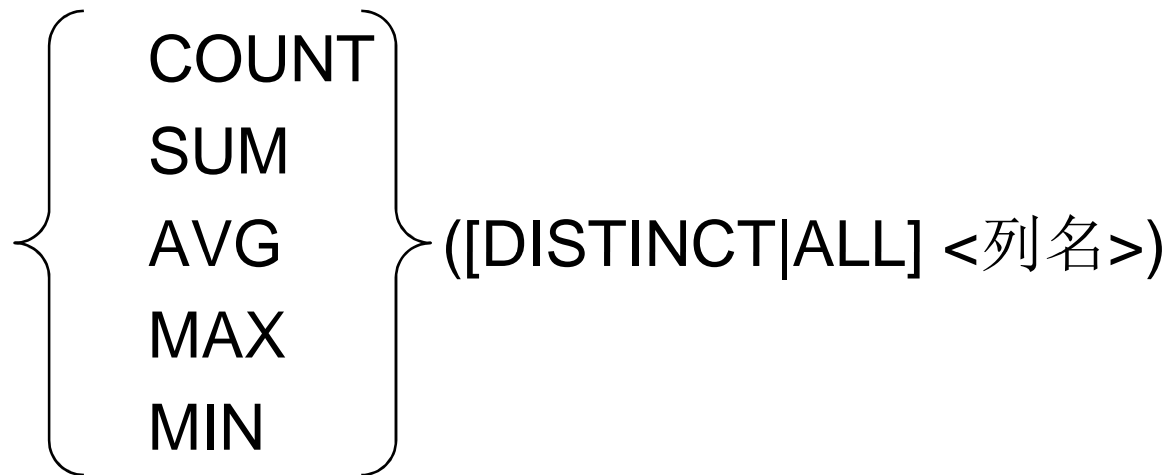
(2) <表名>.\*

(3) COUNT( [DISTINCT|ALL] \* )

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

其中<属性列名表达式>可以是由属性列、作用于属性列的聚集函数和常量的任意算术运算 (+, -, \*, /) 组成的运算公式

## 2. 聚集函数格式

  
COUNT  
SUM  
AVG  
MAX  
MIN  
([DISTINCT|ALL] <列名>)

### 3.WHERE子句的条件表达式格式

(1)

$$\langle \text{属性列名} \rangle \theta \left\{ \begin{array}{l} \langle \text{属性列名} \rangle \\ \langle \text{常量} \rangle \\ [\text{ANY}|\text{ALL}] (\text{SELECT语句}) \end{array} \right\}$$

## WHERE子句的条件表达式格式（续）

(2)

<属性列名> [NOT] BETWEEN {  
    <属性列名>  
    <常量>  
    (SELECT语句)} AND {  
    <属性列名>  
    <常量>  
    (SELECT语句)}

## WHERE子句的条件表达式格式（续）

(3)  
<属性列名> [NOT] IN {  
    (<值1>[, <值2> ] ...)  
    (SELECT语句)

## WHERE子句的条件表达式格式（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)

## WHERE子句的条件表达式格式（续）

(7)

$$\langle \text{条件表达式} \rangle \left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\} \langle \text{条件表达式} \rangle \left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\} \langle \text{条件表达} \rangle \dots$$



## 第三章 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

### 3.7 小结



## 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

1. 插入元组

2. 插入子查询结果

➤ 可以一次插入多个元组

# 一、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

### ❖ 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）

[例1] 将一个新学生元组（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);

## 插入元组（续）

[例2] 将学生张成民的信息插入到Student表中。

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

## 插入元组（续）

[例3] 插入一条选课记录( '200215128', '1 ' )。

```
INSERT
```

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

```
VALUES ( ' 200215128 ', ' 1 ' );
```

RDBMS将在新插入记录的**Grade**列上自动地赋空值。

或者：

```
INSERT
```

```
INTO SC
```

```
VALUES ( ' 200215128 ', ' 1 ', NULL);
```

## 二、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）

[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;
```

## 插入子查询结果（续）

RDBMS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

## 3.4.2 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

1. 修改某一个元组的值
2. 修改多个元组的值
3. 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的成绩置零。

```
UPDATE SC
SET Grade=0
WHERE 'CS'=
    (SELETE 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）

RDBMS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

### ❖ 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全部元组，表的定义仍在字典中

## 删除数据（续）

### ❖ 三种删除方式

1. 删除某一个元组的值
2. 删除多个元组的值
3. 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'=  
      (SELETE Sdept  
       FROM Student  
       WHERE Student.Sno=SC.Sno);
```



## 第三章 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

### 3.7 小结

## 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

## 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

## 3.6.1 定义视图

❖ 建立视图

❖ 删除视图

## 一、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）

- ❖ RDBMS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。

## 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：

- (1) 某个目标列是聚集函数或列表表达式
- (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

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）

### ❖ 子查询不允许含有ORDER BY子句和DISTINCT短语

## 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';
```

## 建立视图（续）

[例2]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）

对IS\_Student视图的更新操作：

- ❖ 修改操作：RDBMS自动加上Sdept= 'IS'的条件
- ❖ 删除操作：RDBMS自动加上Sdept= 'IS'的条件
- ❖ 插入操作：RDBMS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'

## 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
- 保留了主码

## 建立视图（续）

### ❖ 基于多个基表的视图

[例3] 建立信息系选修了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';
```



## 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;
```

## 建立视图（续）

### ❖ 虚拟列：

- 派生属性列
- 在基本表中并不实际存在

### ❖ 带表达式的视图：带虚拟列的视图

### ❖ 分组视图：用带有聚集函数和**GROUP BY**子句的查询来定义视图

## 建立视图（续）

### ❖ 带表达式的视图

[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, 2000-Sage
FROM Student;
```

## 建立视图（续）

### ❖ 分组视图

[例6] 将学生的学号及他的平均成绩定义为一个视图

假设SC表中“成绩”列Grade为数字型

```
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7]将Student表中所有女生记录定义为一个视图

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

缺点：

修改基表Student的结构后，Student表与F\_Student视图的映象关系被破坏，导致该视图不能正确工作。

## 二、删除视图

❖ 语句的格式:

**DROP VIEW** <视图名>;

- 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)

[例8] 删除视图BT\_S: `DROP VIEW BT_S;`

删除视图IS\_S1: `DROP VIEW IS_S1;`

- 由于IS\_S1视图上还导出了IS\_S2视图，此语句时被拒绝执行。
- 如果确要删除，则使用级联删除语句：

`DROP VIEW IS_S1 CASCADE;`

*/\*删除了视图IS\_S1和由它导出的所有视图\*/*





## 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

## 3.6.2 查询视图

- ❖ 从用户角度：查询视图与查询基本表相同
- ❖ RDBMS实现视图查询的方法
  - 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）

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）

[例10] 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';
```

## 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RDBMS会限制这类查询。

## 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;
```

S\_G视图的子查询定义:

```
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
- ❖ RDBMS实现视图更新的方法
  - 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12] 将信息系学生视图IS\_Student中学号200215122的学生姓名改为“刘辰”。

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

转换后的语句：

```
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：  
200215129，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('200215129 ', '赵新', 20, 'IS' );
```

## 更新视图（续）

[例14]删除信息系学生视图IS\_Student中学号为200215129的记录

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

转换为对基本表的更新:

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

## 更新视图（续）

- ❖ 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

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= '200215121';
```

这个对视图的更新无法转换成对基本表SC的更新

## 更新视图（续）

- ❖ 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。
- (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。



## 更新视图（续）

- (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。
- (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。
- (7) 一个不允许更新的视图上定义的视图也不允许更新



## 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**

### 3.6.4 视图的作用

- ❖ 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. 适当的利用视图可以更清晰的表达查询

## 视图的作用（续）

### 1. 视图能够简化用户的操作—关注所关心的数据

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

## 视图的作用（续）

### 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

## 视图的作用（续）

### 3. 视图对重构数据库提供了一定程度的逻辑独立性

❖ 数据库重构：

例：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)**

## 视图的作用（续）

3.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;
```

使外模式保持不变，从而对原**Student**表的查询程序不必修改

## 视图的作用（续）

### 3.视图对重构数据库提供了一定程度的逻辑独立性（续）

- ❖ 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。



## 视图的作用（续）

### 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

## 视图的作用（续）

- ❖ 例：建立1号课程的选课视图，并要求透过该视图进行的更新操作只涉及1号课程，同时对该视图的任何操作只能在工作时间进行。

```
CREATE VIEW IS_SC AS  
SELECT S#, C#, G  
FROM SC  
WHERE C#='1'  
AND TO_CHAR(SYSDATE,'HH24') BETWEEN 9 AND 17  
AND TO_CHAR(SYSDATE,'D') BETWEEN 1 AND 5  
WITH CHECK OPTION;
```

## 视图的作用（续）

### 5. 适当的利用视图可以更清晰的表达查询

- ❖ 经常需要执行这样的查询“对每个同学找出他获得最高成绩的课程号”。可以先定义一个视图，求出每个同学获得的最高成绩

```
CREATE VIEW VMGRADE
```

```
AS
```

```
SELECT Sno, MAX(Grade) Mgrade
```

```
FROM SC
```

```
GROUP BY Sno;
```

## 视图的作用（续）

然后用如下的查询语句完成查询：

```
SELECT SC.Sno, Cno  
FROM SC, VMGRADE  
WHERE SC.Sno=VMGRADE.Sno AND  
SC.Grade=VMGRADE .Mgrade;
```

## 小结

- ❖ **SQL**可以分为数据定义、数据查询、数据更新、数据控制四大部分。有时把数据更新称为数据操纵，或把数据查询与数据更新合称为数据操纵。
- ❖ **SQL**是关系数据库语言的工业标准。目前，大部分数据库管理系统产品都能支持**SQL92**，但是许多只支持**SQL99**的部分特征。
- ❖ **SQL**的数据查询功能最丰富，也是最复杂的。

## 小结

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
模式	<b>CREATE SCHEMA</b>	<b>DROP SCHEMA</b>	
表	<b>CREATE TABLE</b>	<b>DROP TABLE</b>	<b>ALTER TABLE</b>
视 图	<b>CREATE VIEW</b>	<b>DROP VIEW</b>	
索 引	<b>CREATE INDEX</b>	<b>DROP INDEX</b>	

## 小结

### ❖ 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 ] ];

## 小结

### ❖ 插入数据语句格式一:

**INSERT**

**INTO <表名> [( <属性列1> [, <属性列2 >... )]**

**VALUES ( <常量1> [, <常量2>] ... );**

### ❖ 插入数据语句格式二:

**INSERT**

**INTO <表名> [( <属性列1> [, <属性列2>... )]**

**子查询;**



## 小结

### ❖ 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>];**

### ❖ 删除数据语句格式:

**DELETE**

**FROM** <表名>

**[WHERE <条件>];**