

# 数据库系统原理

---

## 第 3 章 关系数据库标准语言 SQL

刘济全

浙江大学生物医学工程与仪器科学学院

E-mail: liujq@zju.edu.cn

个人主页 : <http://person.zju.edu.cn/liujiquan>

# 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
- SQL:2003、SQL/XML:2006

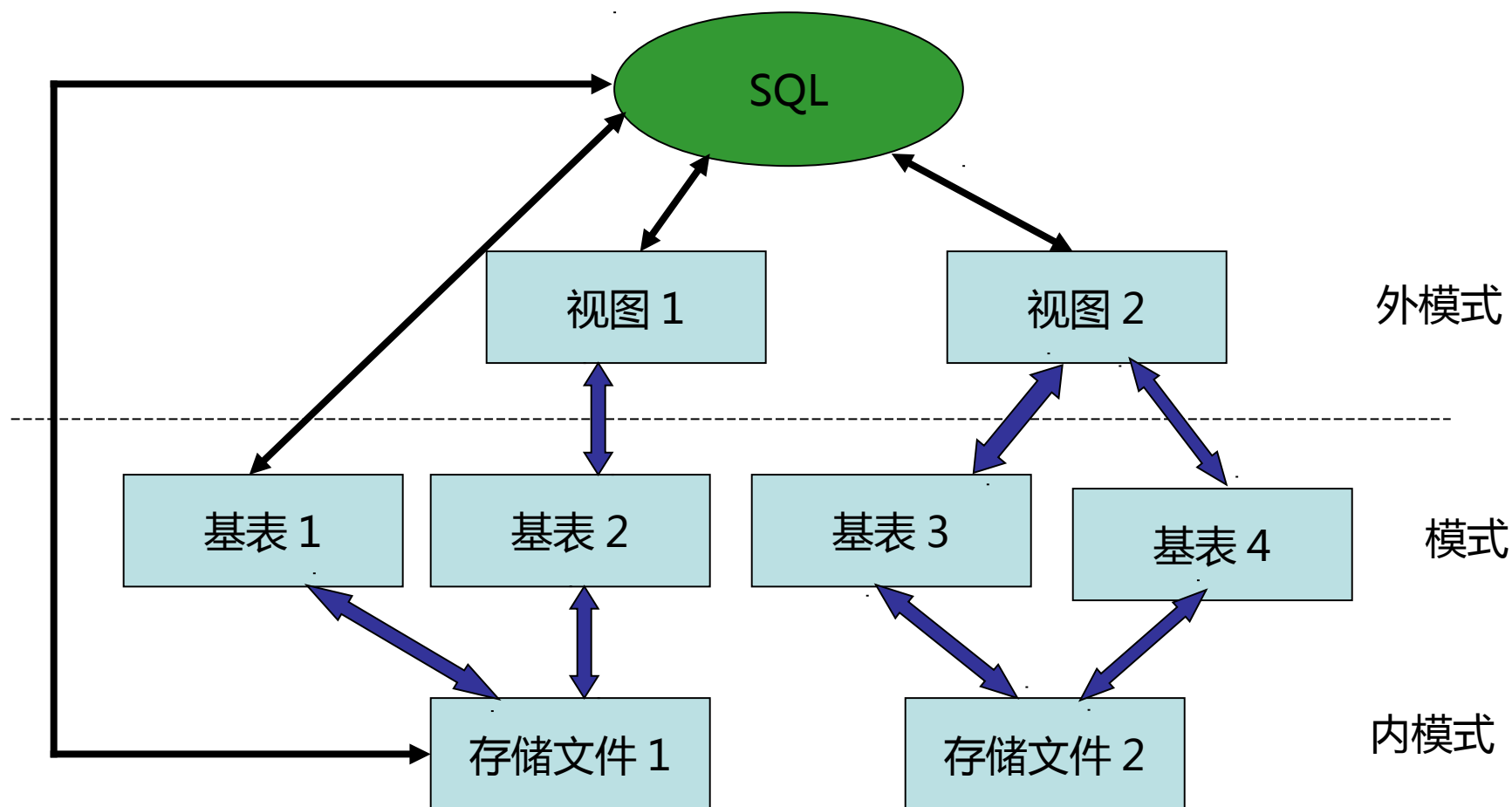
## 3.1.2 SQL 的特点

---

### SQL 的特点

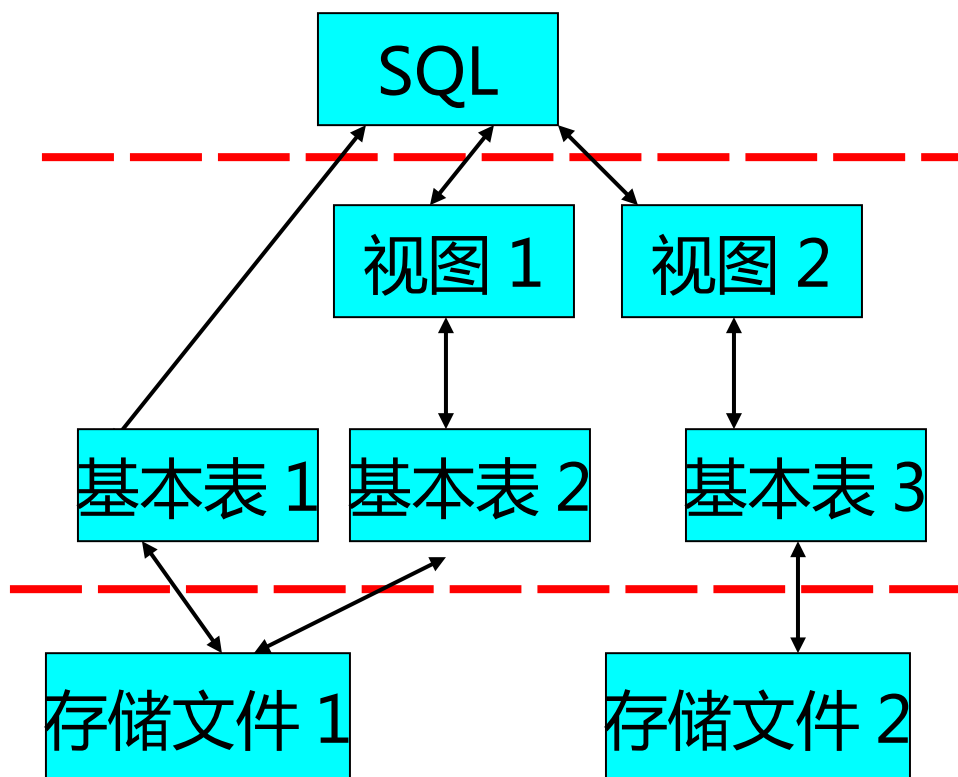
1. 综合统一
2. 高度非过程化
3. 面向集合的操作方式
4. 以同一种语法结构提供两种使用方法
5. 语言简洁，易学易用

## 3.1.2 综合统一（操纵三级模式）



## 3.1.2 综合统一（操纵三级模式）

### SQL 数据库层次结构



### 术语对照：

- | 一般关系模型      | SQL           |
|-------------|---------------|
| ■ 外模式 ----- | 视图 ( VIEW )   |
| ■ 模式 -----  | 基本表 ( TABLE ) |
| ■ 内模式 ----- | 存储文件 ( 索引 )   |
| ■ 元组 -----  | 行 ( ROW )     |
| ■ 属性 -----  | 列 ( COLUMN )  |

## 3.1.2 综合统一（操纵三级模式）

---

- 基本表是独立存在的表，在 SQL 中一个关系就对应一个表。一些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。
- 存储文件的逻辑结构组成了关系数据库的内模式。存储文件的物理文件结构是任意的。
- 视图是从基本表或其他视图中导出的表，本身不独立存储在数据库中，数据库中只存放视图的定义而不存放视图对应的数据，这些数据仍存放在导出视图的基本表中，因此视图是一个虚表。用 SQL 语言对视图和基本表进行查询。在用户眼中，视图和基本表都是关系，而存储文件对用户是透明的



## 3.1.2 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 , <del>DELETE</del>
数 据 控 制	GRANT , REVOKE

## 3.1.3 SQL 环境

---

### ( 1 ) SQL 模式与目录

SQL **模式**：基本表的集合。

好处：允许在不同的 SQL 模式中出现同名的基表名或视图名。

**目录**：SQL 环境中所有模式的集合。

**定位基表的方式**： < 目录名 > . < 模式名 > . < 表名 >

### ( 2 ) SQL 环境

- 设置默认的目录和模式
- 设置用户身份

## 3.1.3 SQL 环境

---

### ( 3 ) 存储过程

**存储过程**是存储在 SQL 服务器上的预编译好的一组为了完成特定功能的 SQL 语句集。

通过指定存储过程的名字并给出参数来执行它。可分为两类：

- ◆ 系统存储过程：由系统自动创建，完成的功能主要是从系统表中获取信息。
- ◆ 用户定义的存储过程：由用户为完成某一特定功能而编写的存储过程。

使用存储过程的**好处**：

- ◆ 可减少网络流量。
- ◆ 增强代码的重用性和共享性。
- ◆ 加快系统运行速度。
- ◆ 保证数据安全。

# 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

## 3.2 学生 - 课程数据库

---

### 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)

## 3.2 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	BME
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	C 语言	6	4

**Course**

(b)

## 3.2 学生 - 课程数据库

---

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

(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  
长度为 5  
不能为空值

↑

字符型  
长度为 8

↑

字符型  
长度为 2

↑

整数

↑

字符型  
长度为 10

# 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

## 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));
```

# 例题（续）

注意

- 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，否则既可定义在列级，也可以定义在表级。

例：以上三个表结构，并指定相应的数据完整性约束条件。

分析

Student 表：

主码：sno      姓名：非空      性别：男、女两值

Course 表

主码：cno      课程名：非空      外码：pcno

SC 表：

主码：(sno,cno)      成绩：0—100

外码：sno,cno

## 例题（续）

CREATE TABLE Student

( sno CHAR (5),

sname CHAR (8) NOT NULL,  列级完整性约束条件

sex CHAR (2),

age SMALLINT,

dept CHAR (20),

PRIMARY KEY(sno),  实体完整性约束条件

CHECK sex IN ( '男' , '女' )

);

 用户自定义完整性约束条件



# 例题（续）

---

```
CREATE TABLE Course  
( cno CHAR (4),  
  cname CHAR (10) NOT NULL,  
  pcno CHAR (4),  
  credit SMALLINT,  
  PRIMARY KEY (cno),  
  FOREIGN KEY (pcno) REFERENCES Course(cno)  
);
```



参照完整性约束条件

## 例题（续）

---

```
CREATE TABLE SC
( sno CHAR (5),
  cno CHAR (4) ,
  grade SMALLINT,
  PRIMARY KEY (sno,cno),
  FOREIGN KEY (sno)REFERENCES Student(sno),
  FOREIGN KEY (cno)REFERENCES Course(cno),
  CHECK ((grade IS NULL) OR
         (grade BETWEEN 0 AND 100))
);
```

## 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

## 二、修改基本表

---

### 补充定义主码

**格式：** ALTER TABLE < 表名 >  
ADD PRIMARY KEY (< 列名表 > )

### 删除主码

**格式：** ALTER TABLE < 表名 >  
DROP PRIMARY KEY

# 例 题

---

[例] 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 ] 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

# 例题

---

## ■ 修改基本表实例

例 向 Student 表增加 “入学时间” 列，其数据类型为日期型

```
ALTER TABLE Student ADD Scome DATE;
```

例 将年龄的数据类型改为半字长整数

```
ALTER TABLE Student MODIFY Sage SMALLINT;
```

例 删除关于学号必须取唯一值的约束

```
ALTER TABLE Student DROP UNIQUE(Sno);
```

# 语句格式（续）

---

- 删除属性列

[例] 将入学日期列删除掉。

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



# 语句格式（续）

---

- 修改属性列

[ 例 ] 删除关于学号必须取唯一值的约束

```
ALTER TABLE Student DROP UNIQUE(Sno);
```

### 三、删除基本表

---

**DROP TABLE < 表名 >; 基本表删除**

基本表定义一旦删除，表中的数据、在此表上建立的视图、索引、触发器、断言都将自动被删除掉。RESTRICT 确保只有不具有相关对象的表才能被撤销。

# 例题

---

[例] 删除 SC 表

```
DROP TABLE SC;
```

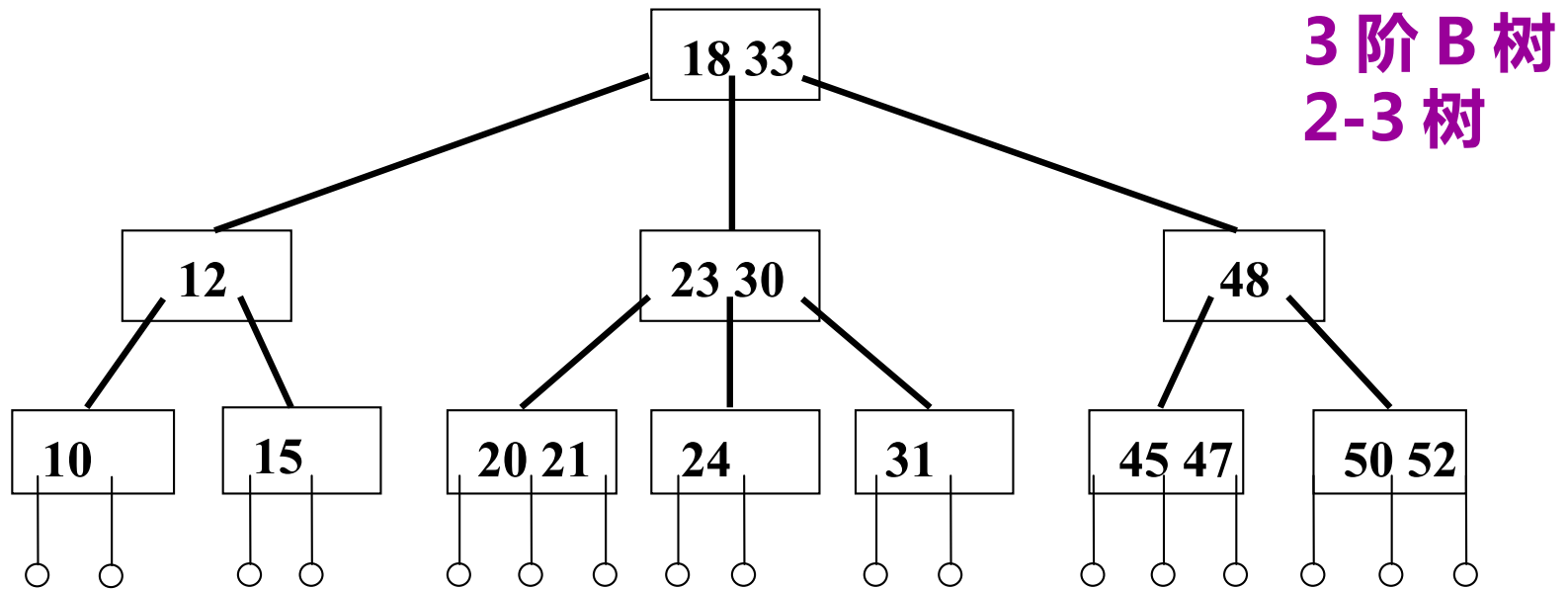
## 3.3 数据定义

---

- 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

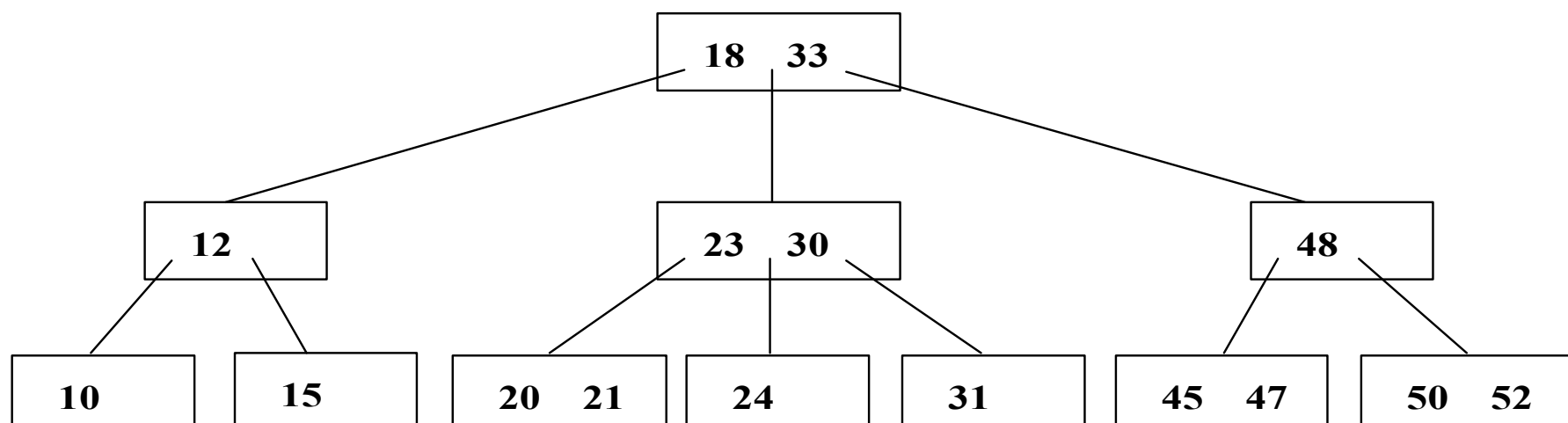
# 索引 --- B 树

- 一种平衡的多分树 (Balanced Tree)



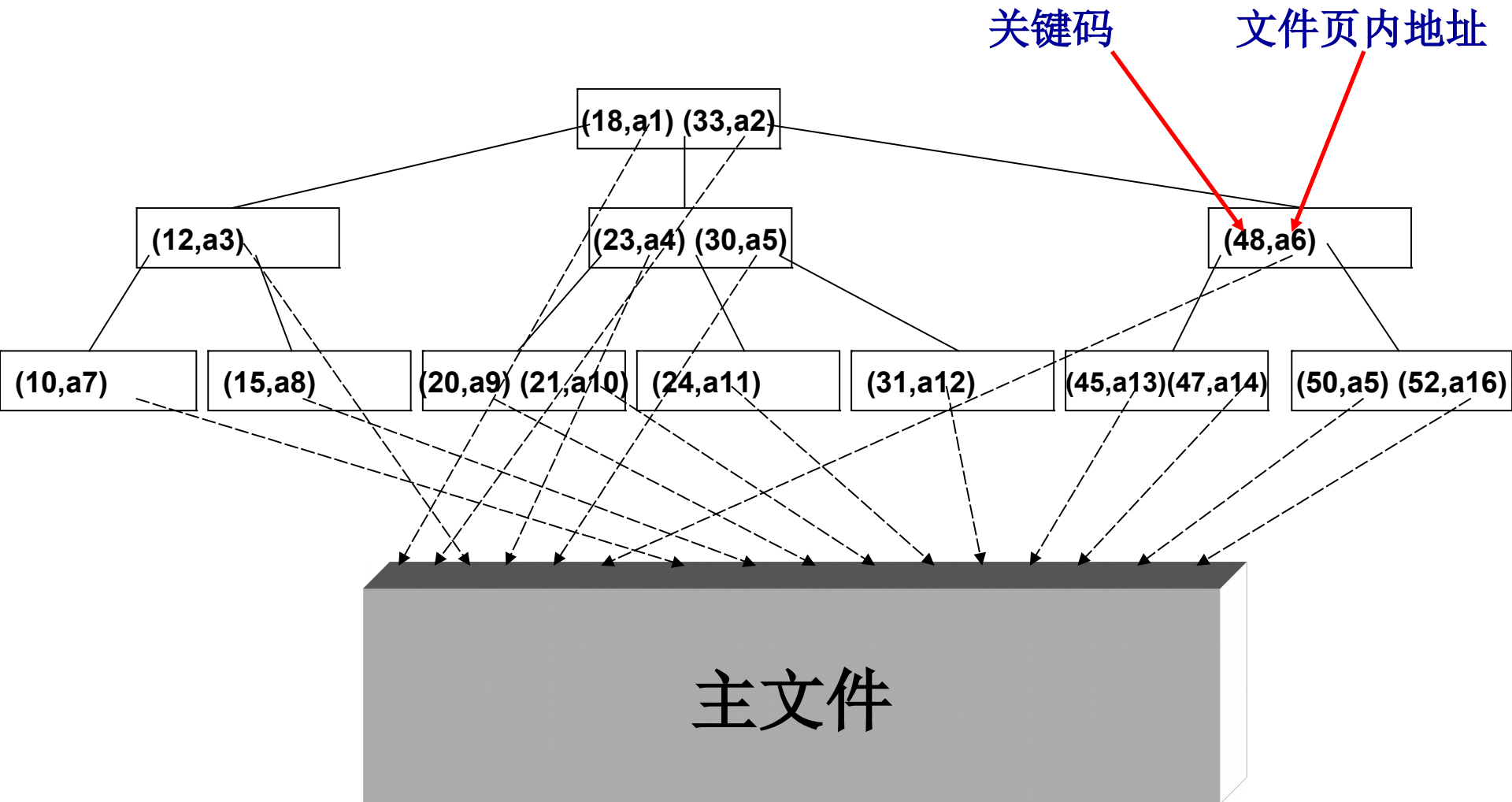
# 索引 --- B 树

---



B 树隐含指针

# 索引 --- B 树



# 索引 --- B 树

---

## m 阶 B 树的结构定义

- (1) 每个结点至多有  $m$  个子结点;
- (2) 除根结点和叶结点外, 其它每个结点至少有  $\lceil m/2 \rceil$  个子结点;
- (3) 根结点至少有两个子结点
  - 唯一例外的是根结点就是叶结点时没有子结点
  - 此时 B 树只包含一个结点
- (4) 所有的叶结点在同一层;
- (5) 有  $k$  个子结点的非根结点恰好包含  $k-1$  个关键码。



# 索引 --- B 树

---

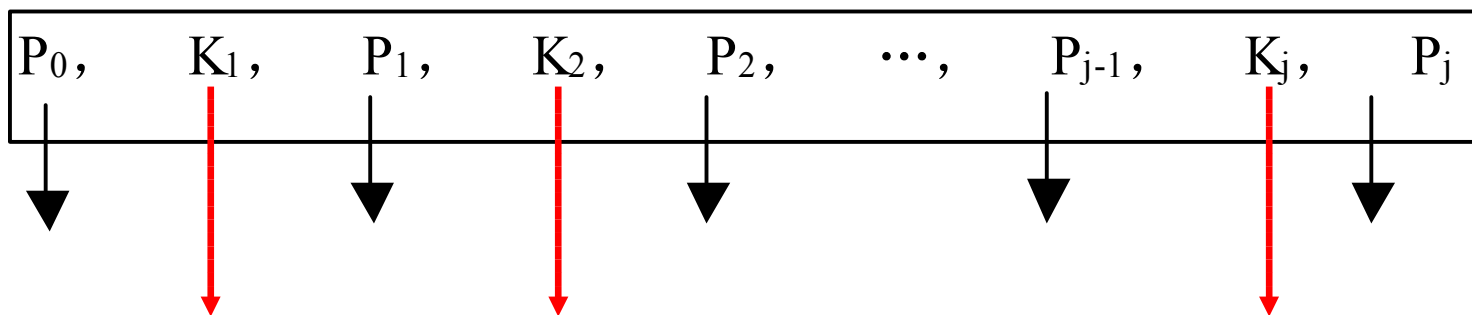
## B 树的性质

- (1) 树高平衡，所有叶结点都在同一层；
- (2) 关键码没有重复，父结点中的关键码是其子结点的分界；
- (3) B 树把（值接近）相关记录放在同一个磁盘页中，从而利用了访问局部性原理；
- (4) B 树保证树中至少有一定比例的结点是满的
  - 这样能够改进空间的利用率
  - 减少检索和更新操作的磁盘读取数目

# 索引 --- B 树

## B 树的结点结构

B 树的一个包含  $j$  个关键码,  $j+1$  个指针的结点的一般形式为:

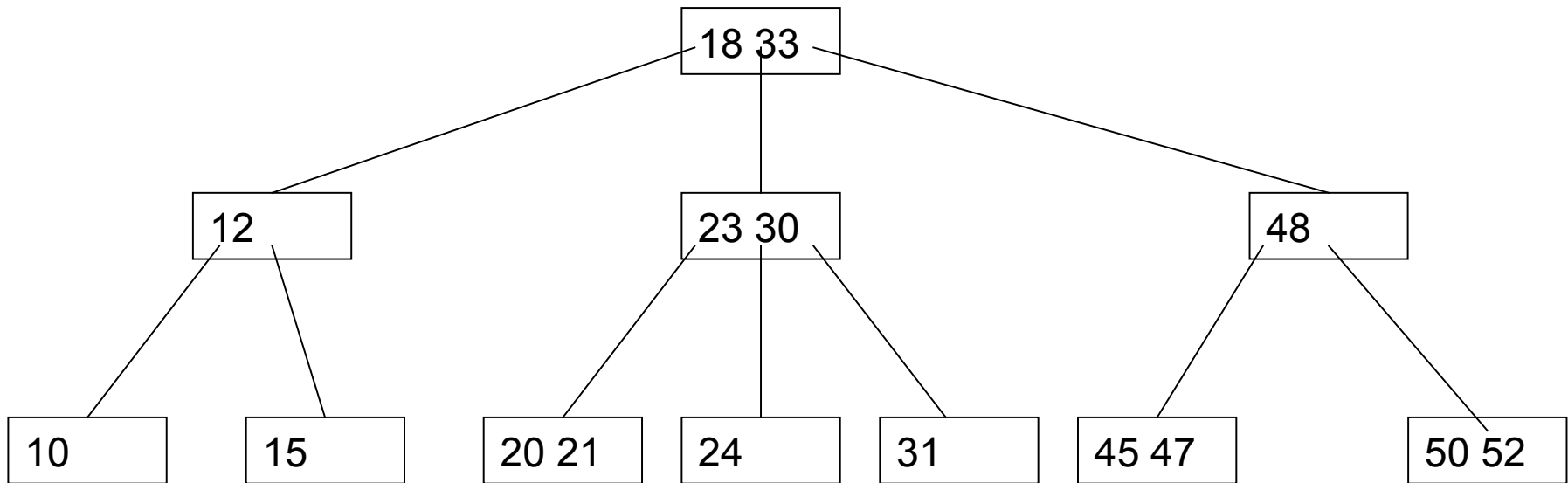


- 其中  $K_i$  是关键码值,  $K_1 < K_2 < \dots < K_j$ ,
- $P_i$  是指向包括  $K_i$  到  $K_{i+1}$  之间的关键码的子树的指针。

# 索引 --- B 树

---

2-3 树 = 3 阶 B 树



# 索引 --- B 树

---

## B 树的查找

- 交替的两步过程
  - 1. 把根结点读出来，在根结点所包含的关键码  $K_1, \dots, K_j$  中查找给定的关键码值
    - 找到则检索成功
  - 2. 否则，确定要查的关键码值是在某个  $K_i$  和  $K_{i+1}$  之间，于是取  $p_i$  所指向的结点继续查找
- 如果  $p_i$  指向外部空结点，表示检索失败

# 索引 --- B 树

---

## B 树检索长度

- 设 B 树的高度为  $h$ 
  - 独根树高为 1
- 在自顶向下检索到叶结点的过程中可能需要进行  $h$  次读盘

## 3.3.3 索引的建立与删除

---

**建立索引的目的**：基本表上建立一个或多个索引，以提供多种存取路径，加快查找速度。

**命令格式**：

CREATE [UNIQUE] [CLUSTER] INDEX < 索引名 >

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

次序：

升序 ( ASC , 缺省 )

降序 ( DESC )

UNIQUE: 每一个索引值只对应惟一的数据记录。

CLUSTER: 建立聚簇索引，即索引项的顺序与表中记录的物理顺序一致。

## 3.3.3 索引的建立与删除

---

建索引有几项参考原则：

- 值得建索引：记录有一定规模，而查询只局限于少数记录。
  - 索引用得上：索引列在 where 子句中应频繁使用。
- 先装数据，后建索引：对于大多数基本表，总是有一批初始数据需要装入。该原则是说，建立关系后，先将这些初始数据装入基表，然后再建索引，这样可加快初始数据的录入。如果建表时就建索引，那么在输入初始数据时，每插入一个记录都要维护一次索引。当然，索引早建晚建都是允许的。

## 3.3.3 索引的建立与删除

**注意：**在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。

- **所建索引放何处？**

- **例：**① 为Student表按学号升序建惟一聚簇索引。  
② 为SC表按学号升序和课程号降序建惟一索引。

① CREATE UNIQUE CLUSTER INDEX **Stno**  
ON Student(Sno) ;

② CREATE UNIQUE INDEX **Scno**  
ON SC(Sno , Cno DESC) ;

- **删除索引一般格式为：**

DROP INDEX < 索引名 > ;

- **例：** DROP INDEX **Stno** ;

- 删除索引时，系统会同时从**数据字典**中  
删去有关该索引的描述。



# 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
- 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，哪种索引效率高？

答：case2

问题 2: 按 C2 搜索时，哪种索引效率高？

答：case2, 并且，case1 的索引无效

问题 3：按 C1,C2,C3 搜索哪种效率高？

答：case1

问题 4：按 C2,C3,C1 搜索时哪种效率高？

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 只选用能加快数据查询速度的索引。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首要条件。

# 例题

---

[ 例 ] 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 ] 删除 Student 表的 Stusname 索引。**

**DROP INDEX Student.Stusname ;**

# 参考试题：索引

---

索引的用途是什么？什么时候使用索引，什么时候不能使用索引

索引最大的好处是提高查询速度，缺点是更新数据时效率低，因为要同时更新索引 对数据进行频繁查询进建立索引，如果要频繁更改数据不建议使用索引。

# 参考试题：索引

---

什么是主码？什么是聚簇索引？什么是非聚簇索引？

表中经常有一个列或列的组合，其值能唯一地标识表中的每一行。这样的一列或多列称为表的主码。

聚簇索引确定表中数据的物理顺序。一个表中只能包含一个聚簇索引。

非聚簇索引与课本中的索引类似。数据存储在—个地方，索引存储在另—个地方，索引带有指针指向数据的存储位置

# 参考试题：索引

---

问：索引是什么？有什么作用以及优缺点？

问：聚簇索引和非聚簇索引区别？

# 参考试题：索引

---

问：索引是什么？有什么作用以及优缺点？

索引是对数据库表中一或多个列的值进行排序的结构，是帮助数据库系统高效获取数据的数据结构

问：聚簇索引和非聚簇索引区别？

聚簇索引和非聚簇索引的根本区别是表记录的排列顺序和与索引的排列顺序是否一致。

# 参考试题：索引

---

问：下列关于“数据库三级模式结构”的叙述中，不正确的是\_\_\_\_\_

- A)视图是外模式
- B)模式是数据库中全体数据的逻辑结构和特征的描述
- C)一个数据库可以有多个模式
- D)一个数据库只有一个内模式

问：在关系数据库中，索引 ( index) 是三级模式结构中的\_\_\_\_\_

- A) 概念模式
- B) 内模式
- C) 模式
- D) 外模式

# 参考试题：索引

---

问：“学生 - 选课 - 课程”数据库中的三个关系是

S(S#, SNAME, SEX, AGE),

SC (S#, C#, GRADE),

C (C#, CNAME, TEACHER)

为了提高查询学生成绩的查询速度，对关系 SC 创建唯一索引，应该创建在哪一个（组）属性上 \_\_\_\_\_

A)S#

B)C#

C)GRADE

D)S#, C#

# 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

---

- 其中 : [...] : 表示其中的成分为任选项。

<...> : 表示其中的成分由用户具体给定。

| : 表示其中并列的成分只能择一。

- 查询目标 :

ALL : 表示保留满足条件的所有元组 ( 缺省 ) 。

DISTINCT : 表示去掉重复元组。

目标列 : 可以为属性名、表达式、通配符 ' \* ' ( 表示所有属性列 ) 。




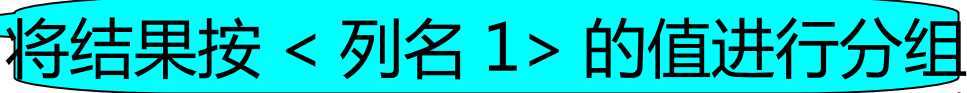


## 3.4 查 询

---

- **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

### ■ 一般形式：

- SELECT <A> 
- FROM <R> 
- WHERE <F> 
- GROUP BY <列名 1>   
HAVING <条件表达式> 
- ORDER BY <列名 2> [ASC|DESC]; 

### ■ 基本语句含义：

- 根据 WHERE 中的 **F**，从**基表或视图 R** 中找出满足条件的元组，再从中选出目标属性值形成结果表。

## 3.4 查 询

---

- 如果有 GROUP 子句，则将结果按 < 列名 1> 的值进行分组，该属性列值相等的元组为一个组，每个组产生结果表中的一条记录。通常在成组后再使用集函数。
- 如果 GROUP 子句带 HAVING 短语，则只有满足指定条件的组才予输出。
- 如果有 ORDER 子句，则结果表还要按 < 列名 2> 的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

### ■ 学生 - 课程数据库：

Student(sno , sname , sex , age , dept)

Course(cno , cname , pcno , credit)

SC(sno , cno , grade)

### 3.3.1 单表查询

#### 一、选择表中的列

- **例**：查询所有学生的姓名、学号、所在系。

SELECT **sname** , **sno** , **dept**

FROM Student

( 次序不同 )；

目标 A
来源 R
条件 F

<b>sname</b>	<b>sno</b>	<b>dept</b>
王三	200101	BME
.....		



## 3.4.1 单表查询

---

- 查询经过计算的值
- SELECT 子句的 < 目标列表达式 > 不仅可以是表中的属性列，也可以是有关系表达式，即可以将查询出来的属性列经过一定的计算后列出结果。
- 例 查全体学生的姓名及其出生年份  
SELECT Sname, 2020-Sage FROM Student;
- 例 查全体学生的姓名、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:', 2020-Sage,  
LOWER(Sdept) FROM Student;

## 3.4.1 单表查询

- **例**：查全体学生的姓名及其出生年份。

```
SELECT sname , 2020-age AS Birthday
FROM Student ;
```

别名

### 二、选择表中的行

#### 1 消除取值重复的行

- **例**：查考试成绩不及格的学号。

```
SELECT DISTINCT sno
FROM SC
WHERE grade<60 ;
```

sname	<b>Birthday</b>
-------	-----------------

王三	1997
----	------

.....

## 3.4.1 单表查询

---

- 2. 查询满足条件的元组
- 查询满足指定条件的元组可以通过 WHERE 子句实现。WHERE 子句常用的查询条件如下所示。

### 常用的查询条件：

查询条件	谓 词
■ 比较	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

## 3.4.1 单表查询

---

- (1) 比较 ( =, <, >... )

**例** 查 BME 全体学生的姓名

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

**例** 查所有年龄在 20 岁以下的学生姓名及其年龄

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

## 3.4.1 单表查询

(2) 确定范围 (**BETWEEN ... AND ...** )

- **例：**查询选 002 号课程且成绩在 80--90 的学生号。

```
SELECT sno
```

```
FROM SC
```

```
WHERE sno= '002' AND grade BETWEEN 80 AND 90 ;
```

关系运算？

(3) 确定集合 (**IN**)

- **例：**查询 BME、IS 何计算机学院学生的姓名和性别。

```
SELECT sname , sex
```

```
FROM Student
```

```
WHERE dept IN ( 'BME' , ' IS' , ' 计算机学院 ' )
```

- **？** 查询不是这三个系的学生们的姓名和性别。

## 3.4.1 单表查询

### (4) 字符匹配 --- 近似查询，模糊查询

**格式：** [NOT] LIKE ' < 匹配串 > '

**含义：** 是查找指定的属性列值与 < 匹配串 > 相匹配的元组。

- 其中匹配串可含：

%：代表任意长度（可为 0）的字符串。

\_：代表任意单个字符。

**例：** 查所有姓刘或姓王的学生姓名、学号和性别。

```
SELECT sname , sno , sex
```

```
FROM Student
```

NOT

```
WHERE sname LIKE '刘%' OR sname LIKE '王%' ;
```

NOT

？ 查询所有非姓刘或非姓王的学生姓名、学号和性别。

## 3.4.1 单表查询

---

- 例 查姓“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 __';
```

注意，由于一个汉字占两个字符的位置，所以匹配串欧阳后面需要跟 2 个 \_。

- 例 查名字中第二字为“阳”字的学生的姓名和学号
- ```
SELECT Sname, Sno  
FROM Student  
WHERE Sname LIKE '__ 阳 %';
```

- **注意：**一个汉字要占两个字符的位置

## 3.4.1 单表查询

**格式 2** : LIKE ' < 匹配串 >' ESCAPE ' < 换码字符 >'

- 若要查的串本身就含有%或\_，则用 ESCAPE ' < 换码字符 >' 对通配符进行转义。ESCAPE '\ ' 短语表示\为换码字符，这样匹配串中紧跟在\后面的字符 '\_' 不再具有通配符的含义，而是取其本身含义，被转义为普通的 '\_' 字符。

- **例**：查 "DB\_" 开头且倒数第 2 个字符为 i 的课程情况。

```
SELECT *
```

```
FROM Course
```

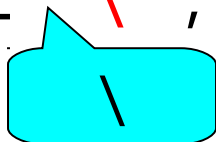
```
WHERE cname LIKE 'DB\ _ % i _' ESCAPE '\ ' ;
```



## 3.4.1 单表查询

■ 查以“ DB\_” 开头，且倒数第三个字符为 i 的课程的具体情况

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



注意这里的匹配字符串 ' DB\\_%i\\_ '。第一个 \_ 前面有换码字符 \，所以它被转义为普通的 \_ 字符。而 %、第二个 \_ 和第三个 \_ 前面均没有换码字符 \，所以它们仍作为通配符。

执行结果为：

Cno	Cname	Ccredit
8	DB_Design	4
10	DB_Programming	2
13	DB_DBMS Design	4

## 3.4.1 单表查询

---

- (5) 涉及空值的查询
- 例 某些学生选修某门课程后没有参加考试，所以有选课记录，但没有考试成绩，下面我们来查一下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;
```

注意这里的 'IS' 不能用等号 ( ' = ' ) 代替。

- 例 查所有有成绩的记录的学生学号和课程号

```
SELECT Sno, Cno  
FROM SC  
WHERE Grade IS NOT NULL;
```

## 3.4.1 单表查询

### 三、对查询结果排序

- **例：** 查询全体男学生的学号、系、年龄

Student 表

条件 F

目标 A

来源

R

结果按所在的系升序排列，  
同一系中的学生按年龄降序排列。

排序一

排序二

```
SELECT sno, dept, age
```

```
FROM Student
```

```
WHERE sex= '男'
```

```
ORDER BY dept , age DESC ;
```

Orderby 的排序键应该是  
查询清单中的列名

## 3.4.1 单表查询

无重复值

缺省值

### 四、使用集函数

COUNT([DISTINCT | ALL] \* )      统计元组个数

COUNT([DISTINCT | ALL]<列名>)      统计一列中值的个数

SUM([DISTINCT | ALL]<列名>)      计算一数值型列值的总和

AVG([DISTINCT | ALL]<列名>)      计算一数值型列值的平均值

MAX([DISTINCT | ALL]<列名>)      求一列值的最大值

MIN([DISTINCT | ALL]<列名>)      求一列值的最小值

## 3.4.1 单表查询

---

例： 查询女学生的总人数和平均年龄。

```
SELECT COUNT(sno),AVG(age)
```

```
FROM Student
```

```
WHERE sex= '女'
```

例： 查询选修 001 号课程并及格的学生的最高分数、最低分及总分。

```
SELECT MAX(grade),MIN(grade),SUM(grade)
```

```
FROM SC
```

```
WHERE cno= '001' and grade>=60
```

## 3.4.1 单表查询

---

### 五、对查询结果分组： GROUP BY 子句

- 将查询结果表按某一（多）列值分组，值相等的为一组。
- 目的：细化集函数的作用对象。如果未对查询结果分组，集函数将作用于整个查询结果，即整个查询结果只有一个函数值。否则，集函数将作用于每一个组，即每一组都有一个函数值。

**例** 查询各个课程号与相应的选课人数

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

## 3.4.1 单表查询

---

- 该 SELECT 语句对 SC 表按 Cno 的取值进行分组，所有具有相同 Cno 值的元组为一组，然后对每一组作用集函数 COUNT 以求得该组的学生人数。

查询结果为：

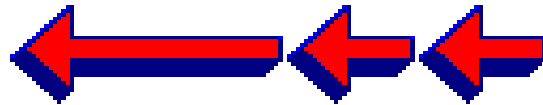
Cno	COUNT(Sno)
-----	-----
1	22
2	34
3	44
4	33
5	48

## 3.4.1 单表查询

- 如果分组后还要求按一定的条件对这些组进行筛选，最终只输出满足指定条件的组，则可以使用 HAVING 短语指定筛选条件。

例：求每个学生（号）的平均成绩，并将其超过 88 分的输出。

```
SELECT  sno , AVG( grade)
FROM  SC
GROUP BY  sno ;
HAVING  AVG( grade) > 88 ;
```



WHERE ?

- HAVING 短语：在**各组**中选择满足条件的**小组**。
- WHERE 子句：在**表**中选择满足条件的**元组**。



## 3.4.1 单表查询

---

HAVING 子句的条件运算数之一是一个集函数。

集函数在成组之前不计算，因此不能用于 WHERE 子句，一般将简单条件写入 WHERE。

若 HAVING 子句无前导 GROUPBY，选择清单中不能含有非集函数列。

## 3.4.1 单表查询

---

**例：**找出选课学生超过 30 人的课程的平均成绩及选课人数。

```
SELECT cno,AVG(grade),COUNT(*) AS st_number  
FROM SC  
GROUP BY cno  
HAVING st_number >= 30
```

## 3.4.1 单表查询

---

**例：**求学生关系中女生的每一年龄组（超过 20 人）有多少，要求查询结果按人数升序排列，人数相同时按年龄降序排列。

```
SELECT age,COUNT(sno) AS number
FROM Student
WHERE sex= '女'
GROUP BY age
HAVING number>20
ORDER BY number,age DESC
```

## 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：

- [

比较运算符：=、>、<、>=、<=、!=

- [

# 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 ] 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
  - [ $\langle$  表名 1  $\rangle$ .] $\langle$  列名 1  $\rangle =$  [ $\langle$  表名 2  $\rangle$ .] $\langle$  列名 2  $\rangle$
  - 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] 对上例用自然连接完成。

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

# 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 ] 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 ;
```

# 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 ] 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 ] 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 ;
```



# 自然连接与外连接

## 1. 自然连接 (NATURAL JOIN)

**例：**查系别为 BME、课程成绩在 90 分以上的学生及其成绩情况。

SELECT \*

FROM Student NATURAL JOIN SC

WHERE dept= 'BME' AND grade >=90 ;

自然连接的  
定义？

FROM SC,  
Student?

## 2. 外连接

- 左外连接。 **LEFT [OUTER] JOIN** , 保留左关系的所有元组。
- 右外连接。 **RIGHT [OUTER] JOIN** , 保留右关系的所有元组。
- 全外连接。 **FULL [OUTER] JOIN** , 保留左右两关系的所有元组。

# 自然连接与外连接

---

教师表：Teacher( 教师号，姓名，所属大学，职称)

任职表：Post( 编号，姓名，职务)

例：SELECT \*

FROM Teacher FULL OUTER JOIN Post ;

例：SELECT \*

FROM Teacher LEFT OUTER JOIN Post ;

例：SELECT \*

FROM Teacher RIGHT OUTER JOIN Post ;

# 自然连接与外连接

```
mysql> use RUNOOB;
```

```
Database changed
```

```
mysql> SELECT * FROM tcount_tbl;
```

```
+-----+-----+
| runoob_author | runoob_count |
+-----+-----+
| 菜鸟教程     | 10           |
| RUNOOB.COM   | 20           |
| Google       | 22           |
+-----+-----+
```

```
3 rows in set (0.01 sec)
```

```
mysql> SELECT * from runoob_tbl;
```

```
+-----+-----+-----+-----+
| runoob_id | runoob_title | runoob_author | submission_date |
+-----+-----+-----+-----+
| 1         | 学习 PHP    | 菜鸟教程     | 2017-04-12      |
| 2         | 学习 MySQL  | 菜鸟教程     | 2017-04-12      |
| 3         | 学习 Java   | RUNOOB.COM   | 2015-05-01      |
| 4         | 学习 Python | RUNOOB.COM   | 2016-03-06      |
| 5         | 学习 C      | FK           | 2017-04-05      |
+-----+-----+-----+-----+
```

```
5 rows in set (0.01 sec)
```

# 自然连接与外连接

## INNER JOIN

```
mysql> SELECT a.runoob_id, a.runoob_author, b.runoob_count FROM runoob_tbl a INNER JOIN tcount_tbl b ON a.runoob_author = b.runoob_author;
```

a.runoob_id	a.runoob_author	b.runoob_count
1	菜鸟教程	10
2	菜鸟教程	10
3	RUNOOB.COM	20
4	RUNOOB.COM	20

```
4 rows in set (0.00 sec)
```

## WHERE 子句

```
mysql> SELECT a.runoob_id, a.runoob_author, b.runoob_count FROM runoob_tbl a, tcount_tbl b WHERE a.runoob_author = b.runoob_author;
```

a.runoob_id	a.runoob_author	b.runoob_count
1	菜鸟教程	10
2	菜鸟教程	10
3	RUNOOB.COM	20
4	RUNOOB.COM	20

```
4 rows in set (0.01 sec)
```

# 自然连接与外连接

## LEFT JOIN

```
mysql> SELECT a.runoob_id, a.runoob_author, b.runoob_count FROM runoob_tbl a LEFT JOIN tcount_tbl b ON a.runoob_author = b.runoob_author;
```

a.runoob_id	a.runoob_author	b.runoob_count
1	菜鸟教程	10
2	菜鸟教程	10
3	RUNOOB.COM	20
4	RUNOOB.COM	20
5	FK	NULL

```
5 rows in set (0.01 sec)
```

# 自然连接与外连接

## RIGHT JOIN

```
mysql> SELECT a.runoob_id, a.runoob_author, b.runoob_count FROM runoob_tbl a RIGHT JOIN tcount_tbl b ON a.runoob_author = b.runoob_author;
```

a.runoob_id	a.runoob_author	b.runoob_count
1	菜鸟教程	10
2	菜鸟教程	10
3	RUNOOB.COM	20
4	RUNOOB.COM	20
NULL	NULL	22

```
5 rows in set (0.01 sec)
```

## 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 ] 查询与 “王三” 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 “王三” 所在系名

```
SELECT Sdept  
FROM Student  
WHERE Sname= ' 王三 '
```

结果为：

Sdept
BME



## 带有 IN 谓词的子查询（续）

---

② 查找所有在 BME 系学习的学生。

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

结果为：

Sno	Sname	Sdept
17001	王三	BME
17004	李四	BME

# 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= ' 王三 ' ) ;
```

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 = '王三' ;
```

# 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= ' 王三 ' ) ;
```

## 带有 IN 谓词的子查询（续）

---

[ 例 ] 查询选修了课程名为 “数据库” 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  
             WHERE Cname= '数据库' ));
```

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 “数据库”  
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= '数据库' ;
```

## 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，则可以用 **=** 代替 **IN**：

```
SELECT Sno , Sname , Sdept
FROM Student
WHERE Sdept =
    (SELECT Sdept
     FROM Student
     WHERE Sname= ' 王三' ) ;
```



# 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] 查询其他系中比 BME 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= ' BME ')
AND Sdept <> ' BME ' ;
/* 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

<u>Sname</u>	<u>Sage</u>
王三	18

执行过程

1. DBMS 执行此查询时，首先处理子查询，找出 BME 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 BME 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

## 帶有 ANY 或 ALL 谓词的子查询（续）

---

[ 例 ] 查询其他系中比 BME 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= ' BME ')
AND Sdept <> ' BEM' ;
```



## 帶有 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= ' BME ')
AND Sdept <> 'BME';
```

## 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 实现逻辑蕴涵

# 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 ]     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 ] 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：查询与“王三”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= ' 王三
' ) ;
```

## 带有 EXISTS 谓词的子查询（续）

---

例：查询与“王三”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 = ' 王三' ) ;
```

# 全称量词和蕴涵逻辑

---

- SQL 语言中没有全称量词  $\forall$  ( For all )。因此对于求所有的操作，必须利用谓词演算将一个带有全称量词的谓词转换为等价的带有存在量词的谓词。
- SQL 语言中也没有蕴涵 ( Implication ) 逻辑运算。因此也必须利用谓词演算将一个逻辑蕴涵的谓词转换为等价的带有存在量词的谓词。

# 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.cjno=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〕 查询 BME 的学生或年龄不大于 19 岁的学生。

方法一：

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

# 并操作（续）

---

方法二：

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

# 并操作（续）

---

[例] 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 ] 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 ] 查询 BME 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 BME 中年龄不大于 19 岁的学生

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

# 交操作（续）

---

[例] 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] 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 ] 查询 BME 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 BME 中年龄大于 19 岁的学生

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

# 差操作（续）

---

[例] 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] 错误写法

```
SELECT *  
FROM Student  
WHERE Sdept= 'BME'  
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= 'BME'  
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]

< 目标列表表达式 > [ 别名 ] [ , < 目标列表表达式 > [ 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 ] 将一个新学生记录

( 学号 : 95020 ; 姓名 : 陈冬 ; 性别 : 男 ; 所在系 :  
BME ; 年龄 : 18 岁 ) 插入到 Student 表中。

INSERT

INTO Student

VALUES ('95020', '陈冬', '男', 'BME', 18);



# 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）

---

[ 例 ] 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

## 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] 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 */  
   Average 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 , Average)
```

```
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 ] 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 ] 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 ] 将 BME 所有学生的年龄增加 1 岁。

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

### 3. 帶子查詢的修改語句

---

[ 例 ] 將 BME 全體學生的成績置零

```
UPDATE SC
```

```
    SET Grade=0
```

```
    WHERE 'BME' =
```

```
        (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 ] 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] 删除 2 号课程的所有选课记录。

DELETE

FROM SC ;

WHERE Cno='2' ;

[例] 删除所有的学生选课记录。

DELETE

FROM SC ;

### 3. 帶子查詢的刪除語句

---

[ 例 ] 刪除 BME 所有學生的選課記錄。

DELETE

FROM SC

WHERE 'BME' =

(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

# 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 ] 建立 BME 学生的视图

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

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 ] 建立 BME 学生的视图，并要求透过该视图进行的更新操作只涉及 BME 学生。

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

## 对 BME\_Student 视图的更新操作

---

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

# 基于多个基表的视图

---

[ 例 ] 建立选修了 1 号课程的学生视图 ( 包括学号、姓名、成绩 )。

```
CREATE VIEW BME_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] 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 BME_S2
```

```
AS
```

```
SELECT Sno , Sname , Grade
```

```
FROM BME_S1
```

```
WHERE Grade >= 90 ;
```



# 带表达式的视图

---

[例] 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 , 2019-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 ] 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 ] 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 BME\_S1

```
DROP VIEW BME_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] 在 BME 学生的视图中找出年龄小于 20 岁的学生。

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

BME\_Student 视图的定义：

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

## 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= 'BME' AND Sage<20 ;
```

## 查询视图（续）

---

[ 例 ] 查询 BME 选修了 1 号课程的学生

```
SELECT Sno , Sname  
FROM    BME_Student , SC  
WHERE   BME_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] 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 ] 将 BME 学生视图 BME\_Student 中学号 95002 的学生姓名改为 “刘辰” 。

```
UPDATE BME_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= 'BME' ;
```

# 更新视图（续）

---

[例] 向 BME 学生视图 IS\_S 中插入一个新的学生记录：95029，赵新，20 岁

```
INSERT
```

```
INTO BME_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 , 'BME') ;
```

## 更新视图（续）

---

[ 例 ] 删除视图 BME\_S 中学号为 95029 的记录

```
DELETE
```

```
FROM BME_Student
```

```
WHERE Sno= '95029' ;
```

转换为对基本表的更新：

```
DELETE
```

```
FROM Student
```

```
WHERE Sno= '95029' AND Sdept= 'BME' ;
```

# 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

### 3. 视图对重构数据库提供了一定程度的逻辑独立性

---

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)

### 3. 视图对重构数据库提供了一定程度的逻辑独立性

---

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

## 3.7 空值的处理

---

- 空值就是 “不知道” 或 “不存在” 或 “无意义” 的值。
- 一般有以下几种情况：
  - 该属性应该有一个值，但目前不知道它的具体值
  - 该属性不应该有值
  - 由于某种原因不便于填写



# 空值的产生

---

- 空值是一个很特殊的值，含有不确定性。对关系运算带来特殊的问题，需要做特殊的处理。

- 空值的产生

[例] 向 SC 表中插入一个元组，学生号是“ 201215126” ，课程号是“ 1” ，成绩为空。

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

```
VALUES('201215126 ','1',NULL); /* 该学生还没有考试成绩，取空值 */
```

或

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

```
VALUES(' 201215126 ','1'); /* 没有赋值的属性，其值为空值 */
```

# 空值的产生（续）

---

[例] 将 Student 表中学生号为“ 201715200” 的学生所属的系改为空值。

```
UPDATE Student  
SET Sdept = NULL  
WHERE Sno='201715200';
```

## 2. 空值的判断

---

- 判断一个属性的值是否为空值，用 IS NULL 或 IS NOT NULL 来表示。

[例] 从 Student 表中找出漏填了数据的学生信息

```
SELECT *
```

```
FROM Student
```

```
WHERE Sname IS NULL OR Ssex IS NULL OR Sage IS NULL OR  
Sdept IS NULL;
```

### 3. 空值的约束条件

---

- 属性定义（或者域定义）中
  - 有 NOT NULL 约束条件的不能取空值
  - 加了 UNIQUE 限制的属性不能取空值
  - 码属性不能取空值

## 4. 空值的算术运算、比较运算和逻辑运算

---

- 空值与另一个值（包括另一个空值）的算术运算的结果为空值
- 空值与另一个值（包括另一个空值）的比较运算的结果为 UNKNOWN。
- 有 UNKNOWN 后，传统二值（TRUE，FALSE）逻辑就扩展成了三值逻辑

# 空值的算术运算、比较运算和逻辑运算 ( 续 )

表 3.7 逻辑运算符真值表

<b>x</b>	<b>y</b>	<b>x AND y</b>	<b>x OR y</b>	<b>NOT x</b>
<b>T</b>	<b>T</b>	<b>T</b>	<b>T</b>	<b>F</b>
<b>T</b>	<b>U</b>	<b>U</b>	<b>T</b>	<b>F</b>
<b>T</b>	<b>F</b>	<b>F</b>	<b>T</b>	<b>F</b>
<b>U</b>	<b>T</b>	<b>U</b>	<b>T</b>	<b>U</b>
<b>U</b>	<b>U</b>	<b>U</b>	<b>U</b>	<b>U</b>
<b>U</b>	<b>F</b>	<b>F</b>	<b>U</b>	<b>U</b>
<b>F</b>	<b>T</b>	<b>F</b>	<b>T</b>	<b>T</b>
<b>F</b>	<b>U</b>	<b>F</b>	<b>U</b>	<b>T</b>
<b>F</b>	<b>F</b>	<b>F</b>	<b>F</b>	<b>T</b>

**T 表示 TRUE ， F 表示 FALSE ， U 表示 UNKNOWN**

# 空值的算术运算、比较运算和逻辑运算 ( 续 )

---

[ 例 ] 找出选修 1 号课程的不及格的学生。

```
SELECT Sno  
FROM SC  
WHERE Grade < 60 AND Cno='1';
```

查询结果不包括缺考的学生，因为他们的 Grade 值为 null。

# 空值的算术运算、比较运算和逻辑运算 ( 续 )

---

[ 例 ] 选出选修 1 号课程的不及格的学生以及缺考的学生。

```
SELECT Sno  
FROM SC  
WHERE Grade < 60 AND Cno='1'  
UNION  
SELECT Sno  
FROM SC  
WHERE Grade IS NULL AND Cno='1'
```

或者

```
SELECT Sno  
FROM SC  
WHERE Cno='1' AND (Grade<60 OR Grade IS NULL);
```



# 课内习题

---

- 若要求查找姓名中第一个字为 '王' 的学生号和姓名。下面列出的 SQL 语句中，哪个（些）是正确的？（ ）

I. SELECT S# , SNAME FROM S WHERE SNAME='王 %'  
II. SELECT S# , SNAME FROM S WHERE SNAME LIKE '王 %'  
III. SELECT S# , SNAME FROM S WHERE SNAME LIKE '王 \_'

- A) I  
B) II  
C) III  
D) 全部

# 课内习题

---

- 设有关系 SC ( sno, cname, grade ) , 各属性的含义分别为学号、课程名、成绩。若要将所有学生的“数据库原理”课程的成绩增加 5 分, 能正确完成该操作的 SQL 语句是\_\_\_\_\_。

# 课内习题

- 对于学生选课关系，其关系模式为：学生（学号，姓名，年龄，所在系）课程（课程号，课程名，先行课号）选课（学号，课程号，成绩）

用关系代数和 SQL 分别完成以下查询：求学过数据库的先行课的学生学号

答案

$\pi_{\text{学号}} (\pi_{\text{学号}, \text{课程号}} (\text{选课}) \bowtie \pi_{\text{先行课号}} (\sigma_{\text{课程名}=\text{"数据库"}} (\text{课程})))$

SELECT DISTINCT 学号

FROM 选课

WHERE 课程号 IN (SELECT 课程号

FROM 课程

WHERE 先行课号 IN (SELECT 课程号

FROM 课程

WHERE 课程名='数据库'))

# 参考习题

---

Student(S#, Sname, Sage, Ssex) 学生表	S# : 学号 ; Sname : 学生姓名 ; Sage : 学生年龄 ; Ssex : 学生性别
Course(C#,Cname,T#) 课程表	C#, 课程编号 ; Cname : 课程名字 ; T# : 教师编号
SC(S#,C#,score) 成绩表 绩	S# : 学号 ; C#, 课程编号 ; score : 成
Teacher(T#,Tname) 教师表	T# : 教师编号 ; Tname : 教师名字

# 参考习题

---

1、查询平均成绩大于 60 分的同学的学号和平均成绩；

```
select S#,avg(score)
```

```
from sc
```

```
group by S# having avg(score) >60;
```

# 参考习题

---

2、查询姓“黄”的老师的人数

```
select count(distinct(Tname))  
  
from Teacher  
  
where Tname like '黄 %';
```

# 参考习题

---

3、查询学过“张三”老师所教的所有课的同学的学号、姓名；

```
select S#,Sname
from Student
where S# in
(select S#
from SC ,Course ,Teacher
where SC.C#=Course.C# and Teacher.T#=Course.T# and
Teacher.Tname= '张三'
group by S#
having count(SC.C#)=
(select count(C#) from Course,Teacher
where Teacher.T#=Course.T# and Tname=' 张三' ));
```

# 参考习题

---

4、查询所有课程成绩小于 60 分的同学的学号、姓名

```
select S#, Sname  
from Student  
where S# not in  
    (select Student.S#  
     from Student,SC  
     where S.S#=SC.S# and score>60);
```



# 参考习题

---

5、查询和 “ 1002” 号的同学学习的课程完全相同的其他同学学号和姓名；

```
select S#  
  from SC  
 where C# in  
        (select C#  
          from SC  
          where S#=' 1002')  
 group by S#  
 having count(*)=(select count(*) from SC where S#=' 1002');
```

# 参考习题

---

## 6、查询各课程号、以及课程成绩最高和最低的分

```
SELECT L.C#, L.score AS 最高分 ,R.score AS 最低分
      FROM SC L , SC AS R
 WHERE L.C# = R.C# and
      L.score = (SELECT MAX(IL.score)
                FROM SC AS IL
                WHERE L.C# = IL.C#
                GROUP BY IL.C#)
 AND
      R.Score = (SELECT MIN(IR.score)
                FROM SC AS IR
                WHERE R.C# = IR.C#
                GROUP BY IR.C#
      );
```