

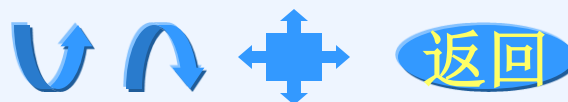


# 第3章 SQL语言

# 本章概要



- SQL是结构化查询语言（Structured Query Language）的缩写，其功能包括数据查询、数据操纵、数据定义和数据控制四个部分。
- SQL 语言简洁、方便实用、功能齐全，已成为目前应用最广的关系数据库语言。



## 3.1 SQL语言的基本概念与特点

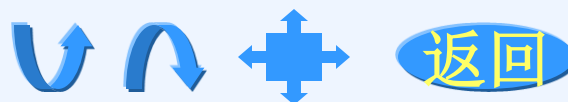


### 3.1.1 SQL语言的发展

#### ➤ 3.1.1.1 SQL语言发展史

SQL语言是当前最为成功、应用最为广泛的关系数据库语言，其发展主要经历了以下几个阶段：

1. 1974 年由 CHAMBERLIN 和 BOYEE 提出，当时称为 SEQUEL(STUCTURED ENGLISH QUERY LANGUAGE)；
2. IBM公司对其进行了修改，并用于其SYSTEM R关系数据库系统中；
3. 1981年 IBM推出其商用关系关系数据库SQL/DS，并将其名字改为SQL，由于SQL语言功能强大，简洁易用，因此得到了广泛的使用；
4. 今天广泛应用于各种大型数据库，如SYBASE、INFORMIX、ORACLE、DB2、INGRES等，也用于各种小型数据库，如FOXPRO、ACCESS。



# SQL概述及特点



字面看SQL只是一个查询语言，而实际上SQL作为一种标准数据库语言，从对数据库的随机查询到数据库的管理和程序设计，SQL几乎无所不能，功能十分丰富。

- SQL语言是一种关系数据库语言，提供数据的定义、查询、更新和控制等功能。
- SQL语言不是一个应用程序开发语言，只提供对数据库的操作能力，不能完成屏幕控制、菜单管理、报表生成等功能，可成为应用开发语言的一部分。
- SQL语言不是一个DBMS，它属于DBMS语言处理程序。
- 大部分DBMS产品都支持SQL，成为操作数据库的标准语言

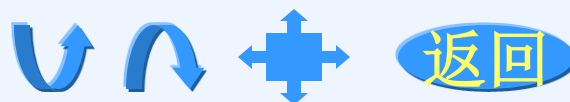
# SQL的特点



## ➤ SQL具有自含式与嵌入式两种形式

- ❖ 交互式SQL：一般DBMS都提供联机交互工具，用户可直接键入SQL命令对数据库进行操作由DBMS来进行解释
- ❖ 嵌入式SQL：能将SQL语句嵌入到高级语言（宿主语言），使应用程序充分利用SQL访问数据库的能力、宿主语言的过程处理能力,一般需要预编译，将嵌入的SQL语句转化为宿主语言编译器能处理的语句
- ❖ SQL的语法结构基本一致

## ➤ SQL具有语言简洁、易学易用的特点



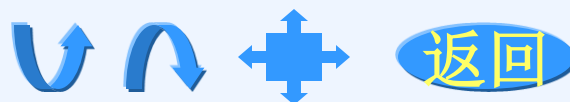


# SQL的特点



## 3、SQL支持三级模式结构

- 一个SQL数据库的总体逻辑结构是基本表（Table）的集合，对应于概念模式
  - SQL数据库的底层存储结构采用文件，一个或几个表对应一个存储文件,以及索引文件。对应内模式
  - 用户所见的数据结构是视图（View），用户可直接操作的表，可为视图或部分基本表。对应外模式
- 
- 注：支持sql语言的数据库称为sql数据库







➤ 例如：学生数据库中有学生基本情况表

**STUDENT(SNO,SNAME,SSEX,SAGE,SDEPT)**，此表为基本表，对应一个存储文件。可以在其基础上定义一个男生基本情况表

**STUDENT\_MALE(SNO,SNAME,SAGE,SDEPT)**，

- 它是从STUDENT中选择SSEX='男'的各个行，然后在SNO,SNAME,SAGE,SDEPT上投影得到的。
- 在数据库中只存有STUDENT\_MALE的定义，而STUDENT\_MALE的记录不重复存储。
- 在用户看来，视图是通过不同路径去看一个实际表，就象一个窗口一样，透过视图可以看到数据库中自己感兴趣的内容。





### 3.1.2 SQL语言的基本概念

- 首先介绍两个基本概念：基本表和视图。
- 基本表（**BASE TABLE**）：是独立存在的表，不是由其它的表导出的表。一个关系对应一个基本表，一个或多个基本表对应一个存储文件。
- 视图（**VIEW**）：是一个虚拟的表，是从一个或几个基本表导出的表。它本身不独立存在于数据库中，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。当基本表中的数据发生变化时，从视图中查询出来的数据也随之改变。



➤ SQL语言支持数据库的三级模式结构，如图3.1所示。其中**外模式**对应于视图和部分基本表，**模式**对应于基本表，**内模式**对应于存储文件。

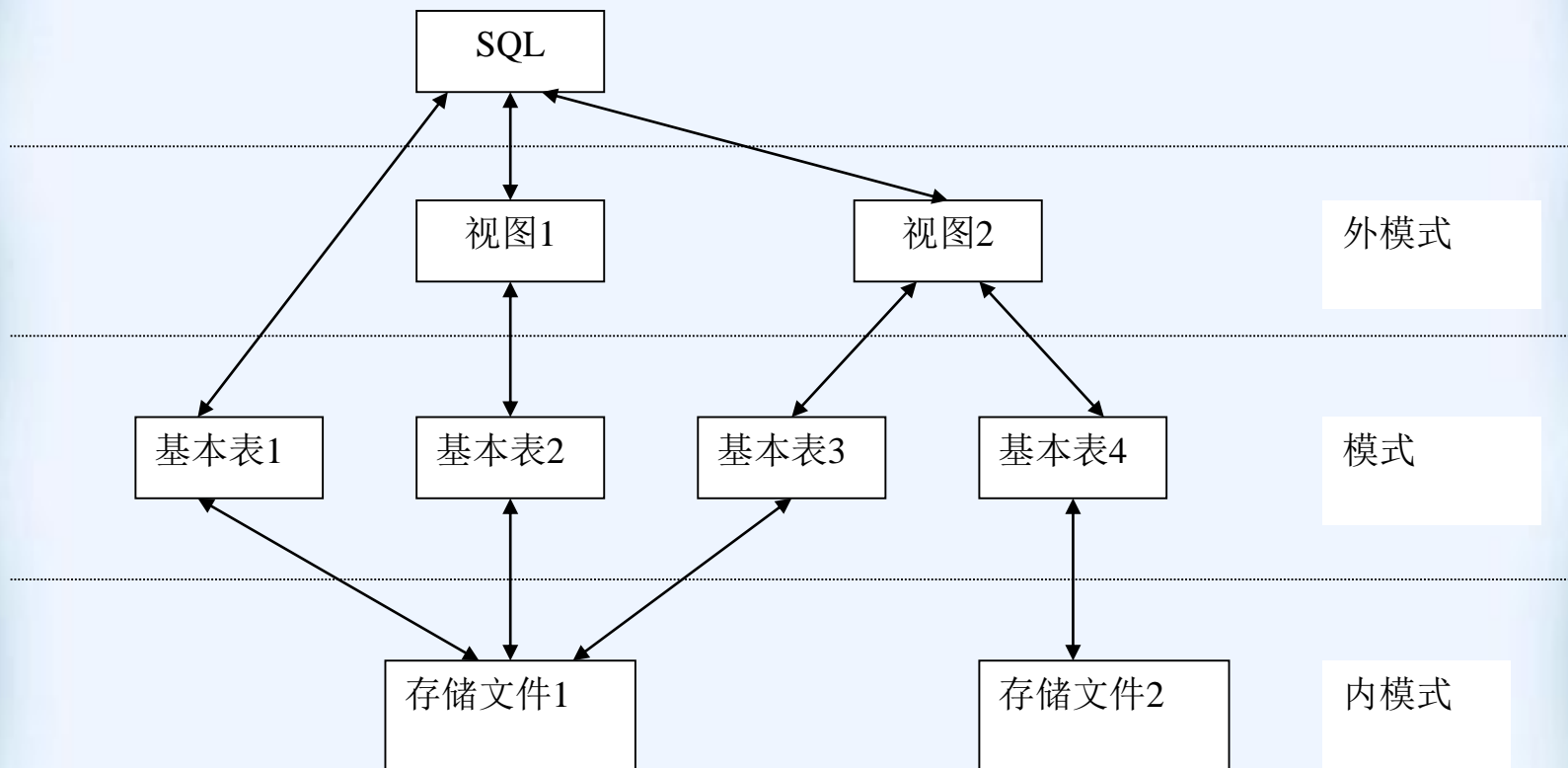
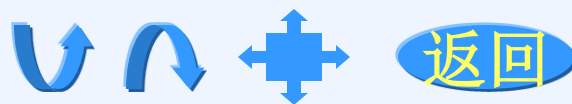
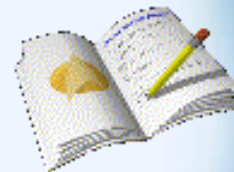


图3.1 SQL语言支持的关系数据库的三级逻辑结构



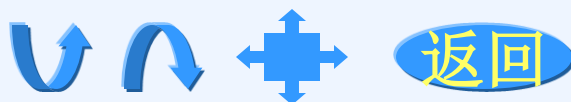
# SQL的特点



SQL语言具有:

- 数据定义 (DEFINITION)
- 数据查询 (QUERY)
- 数据操纵 (MANIPULATION)
- 数据控制 (CONTROL)

下面以SQL SERVER 为例分别介绍其各个功能。各例题中所用的基本表如图1.12所示。

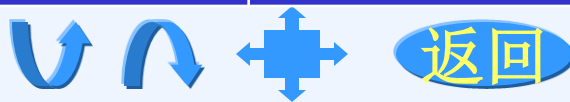


## 3.2 SQL数据定义



- SQL 语言使用 **数据定义语言**（**DATA DEFINITION LANGUAGE**，简称**DDL**）实现其数据定义功能。

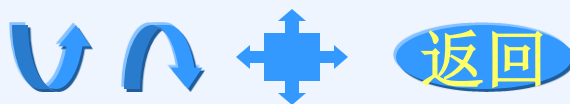
操作对象	操作对象		
	创建	删除	修改
表	<i>Create table</i>	<i>Drop table</i>	<i>Alter table</i>
视图	<i>Create view</i>	<i>Drop view</i>	
索引	<i>Create index</i>	<i>Drop index</i>	
数据库	<i>Create database</i>	<i>Drop database</i>	<i>Alter database</i>



# SQL语句格式的约定符号



- 语句格式中,<> 中的内容是必须的, 是用户自定义语义;
- []为任选项
- {}或分隔符|表示必选项,即必选其中之一项
- [...N]表示前面得项可以重复多次





## 3.2.2 创建、修改和删除数据表

### 3.2.2.1 创建数据表

➤数据表是关系数据库的基本组成单位，它物理地存储于数据库的存储文件中。

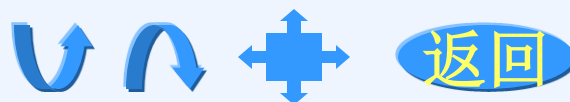
CREATE TABLE [<库名.>]<表名>

(<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]]

[, ...n]

[,<表级完整性约束条件>][, ...n] )

- (1) **<表名>**是合法标识符，最多可有128个字符，如S,SC,C，不允许重名。
- (2) **列名**(字母开头，可含字母、数字、#、\$、\_ <=128字符)。同一表中不许有重名列；
- (2) **数据类型**：见表3.2；
- (3) **字段的长度、精度和小数位**数；



## 3.2.1 字段数据类型



- 当用SQL语句定义表时，需要为表中的每一个字段设置一个数据类型，用来指定字段所存放的数据是整数、字符串、货币或是其它类型的数据。
- SQL SERVER 的数据类型有很多种，分为以下9类：
  1. 整数数据类型：依整数数值的范围大小，有BIT, INT, SMALLINT, TINYINT四种。
  2. 精确数值类型：用来定义可带小数部分的数字，有NUMERIC和DECIMAL两种。十进制数，共P位，其中小数点后S位。 $0 \leq S \leq P$ ， $S=0$ 时可省略。如：123.0、8000.56



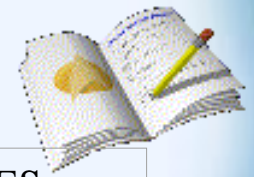


3. **近似浮点数值数据类型**：当数值的位数太多时，可用此数据类型来取其近似值，用**FLOAT**和**REAL**两种。  
如：1.23E+10
4. **日期时间数据类型**：用来表示日期与时间，依时间范围与精确程度可分为 **DATETIME** 与 **SMALLDATETIME**两种。如：1998-06-08 15:30:00
5. **字符串数据类型**：用来表示字符串的字段。包括：**CHAR, VARCHAR, TEXT**三种，如：“数据库”
6. **标记数据类型**：有 **UNIQUEIDENTIFIER**，**TIMESTAMP**两种，此数据类型通常系统自动产生，而不是用户输入的，**TIMESTAMP**记录数据更新的时间戳印，而**UNIQUEIDENTIFIER**用来识别每一笔数据的唯一性。



➤ 各种数据类型的有关规定如下表:

数据 类型	数据内容与范围	占用的字节
BIT	0, 1, NULL	实际使用1BIT，但会占用1BYTE，若一个数据中有数个BIT字段，则可共占1个BYTE
INT	$-2^{31}$ 到 $2^{31}-1$	4BYTES
SMALLINT	$-2^{15}$ 至 $2^{15}-1$	2BYTES
TINYINT	0至255	1BYTES



NUMERIC	$-10^{38-1}$ 至 $10^{38-1}$	1-9位数使用5BYTES 10-19位数使用9BYTES 20-28位数使用13BYTES 29-38位数使用17BYTES
DECIMAL	$-10^{38-1}$ 至 $10^{38-1}$	5-17BYTES因长度而异， 与NUMERIC相同
FLOAT	-1.79E+306 至 1.79E+308，最多可表 示53位数	8BYTES
REAL	-3.40E+38 到 3.40E+38， 最多可表示24位数	4BYTES



DATETIME	1753/1/1至9999/12/31	8BYTES
SMALLDATETIME	1900/1/1至2079/6/6	4BYTES
CHAR	1-8000个字符	1个字符占1B，尾端空白字符保留
VARCHAR	1-8000个字符	1个字符占1B，尾端空白字符删除。
TEXT	2 <sup>31</sup> -1个字符	1个字符占2B，最大可存储2GB

# 字段的长度、精度和小数位数

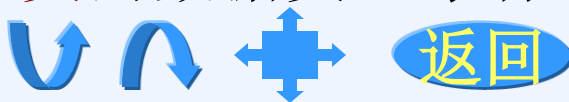


①字段的长度：指字段所能容纳的最大数据量，但对不同的数据类型来说，长度对字段的意义可能有些不同。

- 对字符串数据类型而言，长度代表字段所能容纳的字符的数目，因此它会限制用户所能输入的文本长度。
- 对数值类的数据类型而言，长度则代表字段使用多少个字节来存放数字。。

## ②精度和小数位数

- 精度是指数字的位数，包括小数点左侧的整数部分和小数点右侧的小数部分；
- 小数位数则是指数字小数点右侧的位数。
- 例如：数字12345.678，其精度为8，小数位数为3；
- 所以只有数值类的数据类型才有必要指定精度和小数位数。





- 经常以如下所示的格式来表示数据类型以及它所采用的长度、精度和小数位数，其中的N代表长度，P代表精度，S表示小数位数。
  - BINARY(N) ----- BINARY(10)
  - CHAR(N) ----- CHAR(20)
  - NUMERIC(P,[S]) ----- NUMERIC(8,3)
- 但有的数据类型的精度与小数位数是固定的，对采用此类数据类型的字段而言，不需设置精度与小数位数，
  - 如：如果某字段采用INT数据类型，其长度固定是4，精度固定是10，小数位数则固定是0，这表示字段将能存放10位数没有小数点的整数。存储大小则是4个字节。



➤ 例3.4 建立一学生表



```
USE STUDENT  
CREATE TABLE S  
(SNO CHAR(8) ,  
SN VARCHAR(20),  
AGE INT,  
SEX CHAR(2),  
DEPT VARCHAR(20));
```

- 执行该语句后，便产生了学生基本表的表框架，此表为一个空表。





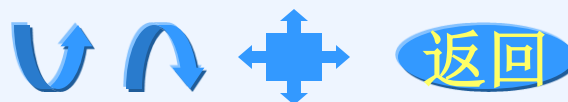
### 3. 定义完整性约束

- 上列为创建基本表的最简单形式，还可以对表进一步定义，如**主键**、**空值**的设定，使数据库用户能够根据应用的需要对基本表的定义做出更为精确和详尽的规定。
- 在SQL SERVER中，对于基本表的约束分为**列约束**和**表约束**。
  - 列约束是对某一个特定列的约束，包含在列定义中，直接跟在该列的其他定义之后，用**空格**分隔，不必指定列名； 
  - 表约束与列定义相互独立，不包括在列定义中，通常用于对多个列一起进行约束，定义表约束时必须指出要约束的那些列的名称。完整性约束的基本语法格式为：  
[ **CONSTRAINT** <约束名> ] <约束类型> 
- **约束名**：约束不指定名称时，系统会给定一个名称。



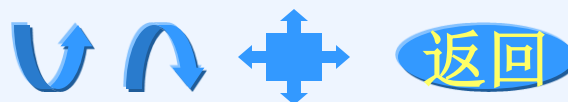
例建立一个S表，定义SN+SEX为唯一。

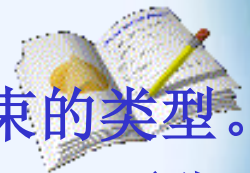
```
USE STUDENT  
CREATE TABLE S  
( SNO CHAR(5),  
  SN CHAR(8),  
  SEX CHAR(2),  
  CONSTRAINT S_UNIQ UNIQUE(SN,SEX));
```





```
USE STUDENT  
CREATE TABLE S  
(SNO CHAR(10) NOT NULL ,  
SN VARCHAR(20),  
AGE INT,  
SEX CHAR(2) DEFAULT '男',  
DEPT VARCHAR(20));
```





- **约束类型：**在定义完整性约束时必须指定完整性约束的类型。
- 在SQL SERVER中可以定义五种类型的完整性约束，下面分别加以介绍：

### (1) NULL/NOT NULL

- 是否允许该字段的值为NULL。
- NULL值不是0也不是空白，更不是填入字符串“NULL”，而是表示“不知道”、“不确定”或“没有数据”的意思。
- 当某一字段的值一定要输入才有意义的时候，则可以设置为NOT NULL。
- 如主键列就不允许出现空值，否则就失去了唯一标识一条记录的作用
- 只能用于定义列约束，
- 其语法格式如下：

**[CONSTRAINT <约束名> ][NULL|NOT NULL]**



### 例3.5 建立一个S表，对SNO字段进行NOT NULL约束。

```
USE STUDENT  
CREATE TABLE S  
(SNO CHAR(10) CONSTRAINT S_CONS NOT NULL,  
SN VARCHAR(20),  
AGE INT,  
SEX CHAR(2) DEFAULT '男' ,  
DEPT VARCHAR(20));
```

- 当SNO为空上时，系统给出错误信息，无NOT NULL约束时，系统缺省为NULL。
- 其中S\_CONS为指定的约束名称，当约束名称省略时，系统自动产生一个名字。如下列功能同上，只是省略约束名称。

## (2) UNIQUE约束

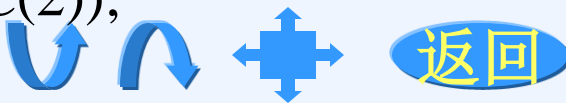


- UNIQUE约束用于指明基本表在某一系列或多个列的组合上的取值必须唯一。
- 定义了UNIQUE约束的那些列称为**唯一键**，系统自动为唯一键建立唯一索引，从而保证了唯一键的唯一性。
- 唯一键允许为空，但系统为保证其唯一性，最多只可以出现一个NULL值。
- UNIQUE既可用于列约束，也可用于表约束。
- UNIQUE用于定义列约束时，其语法格式如下：

[CONSTRAINT <约束名>] UNIQUE

- 例3.6 建立一个S表，定义SN为唯一键。

```
USE STUDENT
CREATE TABLE S
(SNO CHAR(6),
SN CHAR(8) CONSTRAINT SN_UNIQ UNIQUE,
SEX CHAR(2),
AGE NUMERIC(2));
```



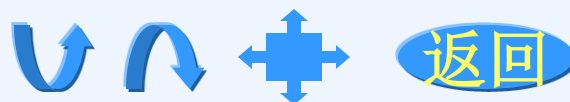


- 其中SN\_UNIQ为指定的约束名称，约束名称可以省略，
- 如下例：

```
USE STUDENT  
CREATE TABLE S  
(SNO CHAR(6),  
SN CHAR(8) UNIQUE,  
SEX CHAR(2),  
AGE NUMERIC(2));
```

- UNIQUE用于定义表约束时，其语法格式如下：

[CONSTRAINT <约束名>] UNIQUE (<列名>[{,<列名>}])







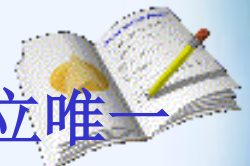
例3.7 建立一个S表，定义SN+SEX为唯一键。

```
USE STUDENT
CREATE TABLE S
( SNO CHAR(5),
  SN CHAR(8),
  SEX CHAR(2),
  CONSTRAINT S_UNIQ UNIQUE(SN,SEX));
```

- 系统为SN+SEX建立唯一索引，确保同一性别的学生没有重名。

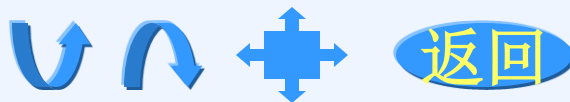
### (3) PRIMARY KEY约束

- PRIMARY KEY约束用于定义基本表的主键，起唯一标识作用，其值不能为NULL，也不能重复，以此来保证实体的完整性。



- **PRIMARY KEY**与**UNIQUE**约束类似，通过建立唯一索引来保证基本表在主键列取值的唯一性，但它们之间存在着很大的区别：
  - ①在一个基本表中只能定义一个**PRIMARY KEY**约束，但可定义多个**UNIQUE**约束；
  - ②对于指定为**PRIMARY KEY**的一个列或多个列的组合，其中任何一个列都不能出现空值，而对于**UNIQUE**所约束的唯一键，则允许为空。
- 注意：不能为同一个列或一组列既定义**UNIQUE**约束，又定义**PRIMARY KEY**约束。
- **PRIMARY KEY**既可用于列约束，也可用于表约束。
- **PRIMARY KEY**用于定义列约束时，其语法格式如下：

**CONSTRAINT <约束名> PRIMARY KEY**





➤ 例3.8 建立一个S表，定义SNO为S的主键

USE STUDENT

CREATE TABLE S

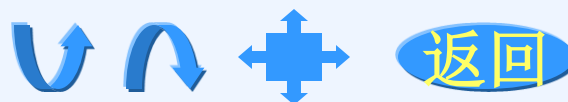
(SNO CHAR(5) NOT NULL CONSTRAINT S\_PRIM  
PRIMARY KEY,

SN CHAR(8),

AGE NUMERIC(2));

➤ PRIMARY KEY用于定义表约束时，即将某些列的组合定义为主键，其语法格式如下：

[CONSTRAINT <约束名>] PRIMARY KEY (<列名>[{<列名>}])





➤ 例3.9 建立一个SC表，定义SNO+CNO为SC的主键

USE STUDENT

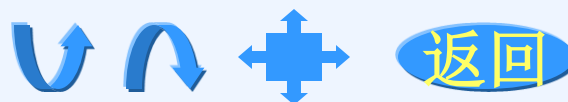
CREATE TABLE SC

(SNO CHAR(5) NOT NULL,

CNO CHAR(5) NOT NULL,

SCORE NUMERIC(3),

CONSTRAINT SC\_PRIM PRIMARY KEY(SNO,CNO));





#### (4) FOREIGN KEY约束

- FOREIGN KEY约束指定某一个列或一组列作为外码，其中，包含外码的表称为**从表**，包含外部键所引用的主键或唯一键的表称**主表**。
- 系统保证从表在外码上的取值要么是主表中某一个主码值，要么取空值。以此保证两个表之间的连接，确保了实体的参照完整性。
- FOREIGN KEY既可用于列约束，也可用于表约束，
- 其语法格式为：

[CONSTRAINT <约束名>] FOREIGN KEY  
REFERENCES <主表名> (<列名>[{<列名>}])



➤ 例3.10 建立一个SC表，定义SNO,CNO为SC的外码。

USE STUDENT

CREATE TABLE SC

(SNO CHAR(5) NOT NULL CONSTRAINT S\_FORE  
FOREIGN KEY REFERENCES S(SNO),

CNO CHAR(5) NOT NULL CONSTRAINT C\_FORE  
FOREIGN KEY REFERENCES C(CNO),

SCORE NUMERIC(3),

CONSTRAINT S\_C\_PRIM PRIMARY KEY  
(SNO,CNO));



## (5) CHECK约束

- CHECK约束用来检查字段值所允许的范围，如，一个字段只能输入整数，而且限定在0-100的整数，以此来保证域的完整性。
- CHECK既可用于列约束，也可用于表约束，
- 其语法格式为：

[CONSTRAINT <约束名>] CHECK (<条件>)

- 例3.10 建立一个SC表，定义SCORE 的取值范围为0到100之间。

```
USE STUDENT
```

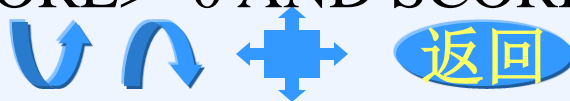
```
CREATE TABLE SC
```

```
(SNO CHAR(5),
```

```
CNO CHAR(5),
```

```
SCORE NUMERIC(5,1) CONSTRAINT SCORE_CHK
```

```
CHECK(SCORE>=0 AND SCORE <=100));
```







### 例3.11 建立包含完整性定义的学生表

```
USE STUDENT
CREATE TABLE S
(SNO CHAR(6) CONSTRAINT S_PRIM PRIMARY
  KEY,
SN CHAR(8) CONSTRAINT SN_CONS NOT NULL,
AGE NUMERIC(2) CONSTRAINT AGE_CONS NOT
  NULL
CONSTRAINT AGE_CHK CHECK (AGE BETWEEN
  15 AND 50),
SEX CHAR(2) DEFAULT '男',
DEPT CHAR(10) CONSTRAINT DEPT_CONS NOT
  NULL);
```



### 3.2.3.2 修改基本表

- 由于应用环境和应用需求的变化，经常需要修改基本表的结构，比如，增加新列和完整性约束、修改原有的列定义和完整性约束等。
- SQL语言使用ALTER TABLE命令来完成这一功能，有如下三种修改方式：

#### 1. ADD方式

- 用于增加新列和完整性约束，定义方式同CREATE TABLE语句中的定义方式相同，其语法格式为：

**ALTER TABLE <表名> ADD <列定义> | <完整性约束定义>**

- 例3.12 在S表中增加一个班号列和住址列。

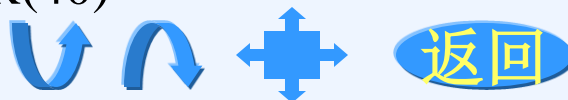
USE STUDENT

ALTER TABLE S

ADD

CLASS\_NO CHAR(6),

ADDRESS CHAR(40)





- 注意：使用此方式增加的新列自动填充NULL值，所以不能为增加的新列指定NOT NULL约束。
- 例3.13 在SC表中增加完整性约束定义，使SCORE在0-100之间。

```
USE STUDENT
ALTER TABLE SC
ADD
CONSTRAINT SCORE_CHK CHECK(SCORE
    BETWEEN 0 AND 100)
```



## 2. ALTER 方式

- 用于修改某些列，其语法格式为：

**ALTER TABLE<表名>**

**ALTER COLUMN <列名><数据类型>[NULL|NOT NULL]**

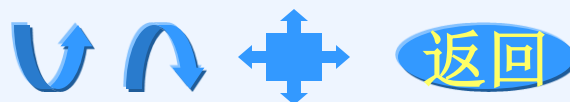
- 例3.14 把S表中的SNO列加宽到8位字符宽度

USE STUDENT

ALTER TABLE S

ALTER COLUMN

SNO CHAR(8)





注意：使用此方式有如下一些限制：

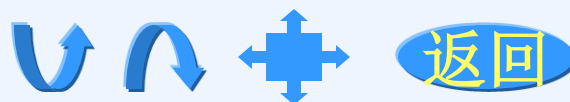
- ①不能改变列名；
- ②不能将含有空值的列的定义修改为NOT NULL约束；
- ③若列中已有数据，则不能减少该列的宽度，也不能改变其数据类型；
- ④只能修改NULL|NOT NULL约束，其它类型的约束在修改之前必须先删除，然后再重新添加修改过的约束定义。

### 3.DROP方式

➤ 删除完整性约束定义，其语法格式为：

**ALTER TABLE<表名>**

**DROP CONSTRAINT <约束名>**





### 例3.15 删除S表中的AGE\_CHK约束

```
USE STUDENT  
ALTER TABLE S  
DROP  
CONSTRAINT AGE_CHK
```

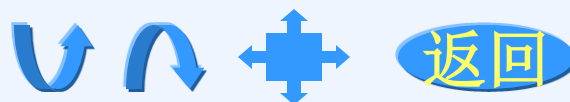
### 3.2.3.3 改变基本表的名字

- 使用**RENAME**命令，可以改变基本表的名字，其语法格式为：

**RENAME** <旧表名> **TO** <新表名>

- 例3.16 将S表的名字更改为STUDENT

```
USE STUDENT  
RENAME S TO STUDENT
```





### 3.2.3.4 删除基本表

- 当某个基本表无用时，可将其删除。
- 删除后，该表中的数据和在此表上所建的索引都被删除，而建立在该表上的视图不会随之删除，系统将继续保留其定义，但已无法使用。
- 如果重新恢复该表，这些视图可重新使用。
- 删除表的语法格式：

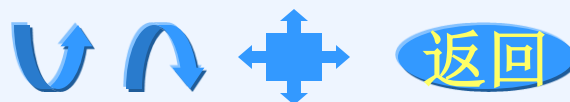
**DROP TABLE <表名>**

- **例3.17 删除表STUDENT**

USE STUDENT

DROP TABLE STUDENT

- 注意：只能删除自己建立的表，不能删除其他用户所建的表。

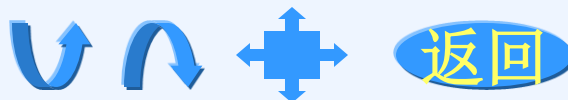




## 3.2.5 设计、创建和维护索引

### 3.2.5.1 索引的作用

- 在日常生活中我们会经常遇到索引，例如图书目录、词典索引等。
- 借助索引，人们会很快地找到需要的东西。
- 索引是数据库随机检索的常用手段，它实际上就是记录的关键字与其相应地址的对应表。
- 例如，当我们要在本书中查找有关“SQL查询”的内容时，应该先通过目录找到“SQL查询”所对应的页码，然后从该页码中找出所要的信息。这种方法比直接翻阅书的内容要快。
- 如果把数据库表比作一本书，则表的索引就如书的目录一样，通过索引可大大提高查询速度。
- 此外，在SQL SERVER中，行的唯一性也是通过建立唯一索引来维护的。
- 索引的作用可归纳为：
  1. 加快查询速度；
  2. 保证行的唯一性。







### 3.2.5.2 索引的分类

#### 1. 按照索引记录的存放位置可分为聚集索引与非聚集索引

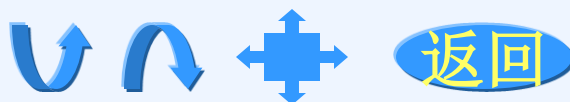
- **聚集索引**：按照索引的字段排列记录，并且依照排好的顺序将记录存储在表中。
- **非聚集索引**：按照索引的字段排列记录，但是排列的结果并不会存储在表中，而是另外存储。

#### 2. 唯一索引的概念

- **唯一索引**表示表中每一个索引值只对应唯一的数据记录，
- 这与表的PRIMARY KEY的特性类似，因此唯一性索引常用于PRIMARY KEY的字段上，以区别每一笔记录。
- 当表中有被设置为**UNIQUE**的字段时，SQL SERVER会自动建立一个**非聚集的唯一性索引**。
- 而当表中有**PRIMARY KEY**的字段时，SQL SERVER会在PRIMARY KEY字段建立一个**聚集索引**。

#### 3. 复合索引的概念

- **复合索引**是将两个字段或多个字段组合起来建立的索引，而单独的字段允许有重复的值。



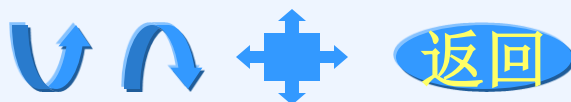


### 3.2.5.3 建立索引

- 建立索引的语句是**CREATE INDEX**，其语法格式为：  
**CREATE [UNIQUE] [CLUSTER] INDEX <索引名> ON <表名> (<列名> [次序] [{,<列名>}] [次序]...)**
- **UNIQUE**表明建立唯一索引。
- **CLUSTER**表示建立聚集索引。
- 次序用来指定索引值的排列顺序，可为**ASC**（升序）或**DESC**（降序），缺省值为**ASC**。
- **例3.18** 为表**SC**在**SNO**和**CNO**上建立唯一索引。

USE STUDENT

CREATE UNIQUE INDEX SCI ON SC(SNO,CNO)





- 执行此命令后，为SC表建立一个索引名为SCI的唯一索引，
- 此索引为SNO和CNO两列的复合索引，即对SC表中的行先按SNO的递增顺序索引，对于相同的SNO，又按CNO的递增顺序索引。
- 由于有UNIQUE的限制，所以该索引在(SNO,CNO)组合列的排序上具有唯一性，不存在重复值。
- 例3.19 为教师表T在TN上建立聚集索引。  
**CREATE CLUSTER INDEX TI ON T(TN)**
- 执行此命令后，为T表建立一个索引名为TI的聚集索引，T表中的记录将按照TN值的升序存放。



➤ **注意：**

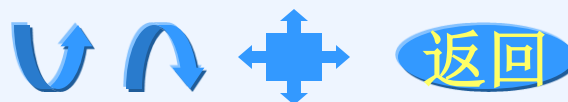
1. 改变表中的数据（如增加或删除记录）时，索引将自动更新。索引建立后，在查询使用该列时，系统将自动使用索引进行查询。
2. 索引数目无限制，但索引越多，更新数据的速度越慢。对于仅用于查询的表可多建索引，对于数据更新频繁的表则应少建索引。

#### 3.2.5.4 删除索引

- 建立索引是为了提高查询速度，但随着索引的增多，数据更新时，系统会花费许多时间来维护索引。这时，应删除不必要的索引。
- 删除索引的语句是**DROP INDEX**，其语法格式为：  
**DROP INDEX 数据表名.索引名**

- **例3.20 删除表SC的索引SCI。**

**DROP INDEX SC.SCI**



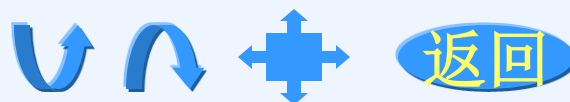
## 3.3 SQL数据查询



### 3.3.1 SELECT命令的格式与基本使用

- 数据查询是数据库中最常见的操作。
- SQL语言提供SELECT语句，通过查询操作可得到所需的信息。
- SELECT语句的一般格式为：

```
SELECT <列名> [{, <列名> }]  
FROM <表名或视图名> [{, <表名或视图名> }]  
[WHERE <检索条件> ]  
[GROUP BY <列名1>[HAVING <条件表达式>]]  
[ORDER BY <列名2>[ASC|DESC]];
```





## ➤ SELECT语句的格式:

SELECT [ALL|DISTINCT][TOP N [PERCENT][WITH  
TIES]]

列名1 [AS 别名1]

[, 列名2 [AS 别名2]...]

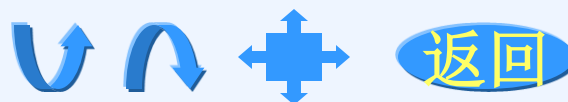
[INTO 新表名]

FROM 表名 1[[AS] 表1别名]

[INNER|RIGHT|FULL|OUTER][OUTER]JOIN

表名2 [[AS] 表2别名]

ON 条件





- 查询的结果是仍是一个表。
- **SELECT**语句的执行过程是：
  - 根据**WHERE**子句的检索条件，从**FROM**子句指定的基本表或视图选取满足条件的元组，再按照**SELECT**子句中指定的列，投影得到结果表。
  - 如果有**GROUP**子句，则将查询结果按照<列名1>相同的值进行分组。
  - 如果**GROUP**子句后有**HAVING**短语，则只输出满足**HAVING**条件的元组。
  - 如果有**ORDER**子句，查询结果还要按照<列名2>的值进行排序。





例3.21 查询全体学生的学号、姓名和年龄。

**SELECT SNO, SN, AGE FROM S**

例3.22 查询学生的全部信息。

**SELECT \* FROM S**

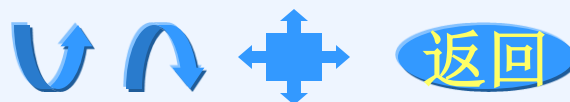
➤ 用 ‘\*’ 表示S表的全部列名，而不必逐一列出。

例3.23 查询选修了课程的学生号。

**SELECT DISTINCT SNO FROM SC**

➤ 查询结果中的重复行被去掉

➤ 上述查询均为不使用WHERE子句的无条件查询，也称作**投影查询**。







- 另外，利用投影查询可控制列名的顺序，并可通过指定**别名**改变查询结果的列标题的名字。

**例3.24** 查询全体学生的姓名、学号和年龄。

**SELECT SNAME NAME, SNO, AGE FROM S**

- 其中，NAME为SNAME的别名



### 3.3.2 条件查询

- 当要在表中找出满足某些条件的行时，则需使用 **WHERE**子句指定查询条件。
- **WHERE**子句中，条件通常通过三部分来描述：
  1. 列名；
  2. 比较运算符；
  3. 列名、常数。

运算符	含义
=, >, <, >=, <=, !=	比较大小
多重条件	AND, OR
BETWEEN AND	确定范围
IN	确定集合
LIKE	字符匹配
IS NULL	空值

表3.8 常用的比较运算符



### 3.3.2.1 比较大小

**例3.25** 查询选修课程号为 ‘C1’的学生的学号和成绩。

**SELECT SNO,SCORE FROM SC WHERE CNO='C1'**

**例3.26** 查询成绩高于85分的学生的学号、课程号和成绩。

**SELECT SNO,CNO,SCORE FROM SC WHERE  
SCORE>85**

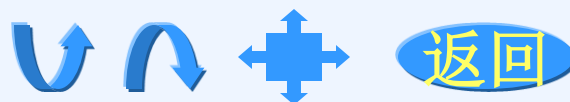


### 3.3.2.2 多重条件查询

- 当WHERE子句需要指定一个以上的查询条件时，则需要使用逻辑运算符AND、OR和NOT将其连结成复合的逻辑表达式。
- 其优先级由高到低为：NOT、AND、OR，用户可以使用括号改变优先级。

例3.27 查询选修C1或C2且分数大于等于85分学生的学号、课程号和成绩。

```
SELECT SNO, CNO, SCORE  
FROM SC  
WHERE (CNO='C1' OR CNO='C2') AND  
SCORE>=85
```





### 3.3.2.3 确定范围

**例3.28** 查询工资在1000至1500之间的教师的教师号、姓名及职称。

```
SELECT TNO,TN,PROF  
FROM T  
WHERE SAL BETWEEN 1000 AND 1500
```

➤ 等价于

```
SELECT TNO,TN,PROF  
FROM T  
WHERE SAL>=1000 AND SAL<=1500
```



**例3.29** 查询工资不在1000至1500之间的教师的教师号、姓名及职称。

```
SELECT TNO,TN,PROF  
FROM T
```

```
WHERE SAL NOT BETWEEN 1000 AND 1500
```

### 3.2.2.4 确定集合

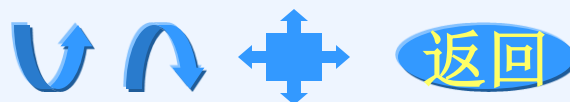
➤ 利用 “**IN**”操作可以查询属性值属于指定集合的元组。

**例3.30** 查询选修C1或C2的学生的学号、课程号和成绩。

```
SELECT SNO, CNO, SCORE  
FROM SC
```

```
WHERE CNO IN('C1', 'C2')
```

➤ 此语句也可以使用逻辑运算符 “**OR**”实现。





```
SELECT SNO, CNO, SCORE  
FROM SC  
WHERE CNO='C1' OR CNO= 'C2'
```

➤ 利用 “**NOT IN**”可以查询指定集合外的元组。

例3.31 查询没有选修C1，也没有选修C2的学生的学号、课程号和成绩。

```
SELECT SNO, CNO, SCORE  
FROM SC  
WHERE CNO NOT IN('C1', 'C2')
```

➤ 等价于：

```
SELECT SNO, CNO, SCORE  
FROM SC  
WHERE CNO!='C1' AND CNO!='C2'
```

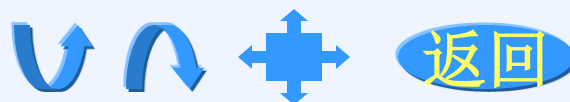


### 3.3.2.5 部分匹配查询

- 上例均属于完全匹配查询，当不知道完全精确的值时，用户还可以使用**LIKE**或**NOT LIKE**进行部分匹配查询（也称模糊查询）。
- **LIKE**定义的一般格式为：  
    <属性名> **LIKE** <字符串常量>
  - 属性名必须为字符型，字符串常量的字符可以包含如下两个特殊符号：
  - %：表示任意知长度的字符串；
  - \_：表示任意单个字符。

例3.32 查询所有姓张的教师的教师号和姓名。

```
SELECT TNO, TN  
FROM T  
WHERE TN LIKE '张%'
```







例3.33 查询姓名中第二个汉字是“力”的教师号和姓名。

```
SELECT TNO, TN
```

```
FROM T
```

```
WHERE TN LIKE ‘__力%’
```

➤ 注：一个汉字占两个字符。

### 3.3.2.6 空值查询

- 某个字段没有值称之为具有空值（NULL）。
- 通常没有为一个列输入值时，该列的值就是空值。
- 空值不同于零和空格，它不占任何存储空间。
- 例如，某些学生选课后没有参加考试，有选课记录，但没有考试成绩，考试成绩为空值，这与参加考试，成绩为零分的不同。



例3.34 查询没有考试成绩的学生的学号和相应的课程号。

```
SELECT SNO, CNO
```

```
FROM SC
```

```
WHERE SCORE IS NULL
```

➤ 注意：这里的空值条件为 IS NULL，不能写成 SCORE=NULL。



### 3.2.2 常用库函数及统计汇总查询

- SQL提供了许多库函数，增强了基本检索能力。
- 常用的库函数，如表3.2所示

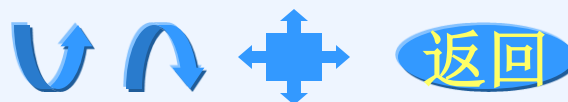
函数名称	功能
AVG	按列计算平均值
SUM	按列计算值的总和
MAX	求一列中的最大值
MIN	求一列中的最小值
COUNT	按列值计个数



例3.35 求学号为S1学生的总分和平均分。

```
SELECT SUM(SCORE) AS TotalScore, AVG(SCORE)
      AS AveScore
FROM SC
WHERE (SNO = 'S1')
```

➤ 注意：函数SUM和AVG只能对数值型字段进行计算。





### 例3.36 求选修C1号课程的最高分、最低分及之间相差的分数

```
SELECT MAX(SCORE) AS MaxScore, MIN(SCORE)
      AS MinScore, MAX(SCORE) - MIN(SCORE)
      AS Diff
FROM SC
WHERE (CNO = 'C1')
```

### 例3.37 求计算机系学生的总数

```
SELECT COUNT(SNO) FROM S
WHERE DEPT='计算机'
```



### 例3.38 求学校中共有多少个系

```
SELECT COUNT(DISTINCT DEPT) AS DeptNum  
FROM S
```

- 注意：加入关键字**DISTINCT**后表示消去重复行，可计算字段“**DEPT**”不同值的数目。
- **COUNT**函数对**空值**不计算，但对**零**进行计算。

### 例3.39 统计有成绩同学的人数

```
SELECT COUNT (SCORE)  
FROM SC
```

- 上例中成绩为零的同学计算在内，没有成绩（即为空值）的不计算。



### 例3.40 利用特殊函数COUNT(\*)求计算机系学生的总数

```
SELECT COUNT(*) FROM S  
WHERE DEPT='计算机'
```

- COUNT (\*) 用来统计元组的个数
- 不消除重复行，不允许使用DISTINCT关键字。



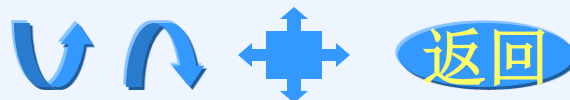
### 3.3.3 分组查询

- **GROUP BY**子句可以将查询结果按属性列或属性列组合在行的方向上进行分组，每组在属性列或属性列组合上具有相同的值。

例3.42 查询各位教师的教师号及其任课的门数。

```
SELECT TNO,COUNT(*) AS C_NUM  
FROM TC  
GROUP BY TNO
```

- **GROUP BY**子句按TNO的值分组，所有具有相同TNO的元组为一组，对每一组使用函数COUNT进行计算，统计出各位教师任课的门数。





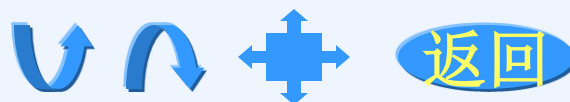


- 若在分组后还要按照一定的条件进行筛选，则需使用 **HAVING** 子句。

### 例3.43 查询选修两门以上课程的学生学号和选课门数

```
SELECT SNO,COUNT(*) AS SC_NUM  
FROM SC  
GROUP BY SNO  
HAVING COUNT(*)>=2
```

- **GROUP BY**子句按SNO的值分组，所有具有相同SNO的元组为一组，对每一组使用函数**COUNT**进行计算，统计出每位学生选课的门数。
- **HAVING**子句去掉不满足**COUNT (\*) >=2**的组。





- 当在一个SQL查询中同时使用**WHERE**子句，**GROUP BY**子句和**HAVING**子句时，其顺序是**WHERE — GROUP BY — HAVING**。
- **WHERE**与**HAVING**子句的根本区别在于作用对象不同。
  - **WHERE**子句作用于基本表或视图，从中选择满足条件的元组；
  - **HAVING**子句作用于组，选择满足条件的组，必须用于**GROUP BY**子句之后，但**GROUP BY**子句可没有**HAVING**子句。

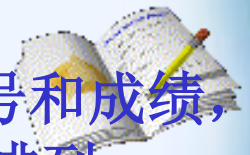


### 3.3.5 查询的排序

- 当需要对查询结果排序时，应该使用**ORDER BY**子句
- **ORDER BY**子句必须出现在其他子句之后
- 排序方式可以指定，**DESC**为降序，**ASC**为升序，缺省时为升序

例3.44 查询选修C1 的学生学号和成绩，并按成绩降序排列。

```
SELECT SNO, SCORE  
FROM SC  
WHERE CNO='C1'  
ORDER BY SCORE DESC
```

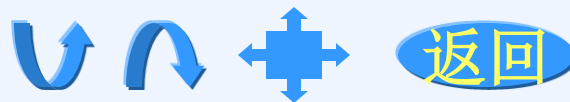


**例3.45** 查询选修C2、C3、C4或C5课程的学号、课程号和成绩，查询结果按学号升序排列，学号相同再按成绩降序排列。

```
SELECT SNO,CNO, SCORE
FROM SC
WHERE CNO IN ('C2' ,'C3', 'C4','C5')
ORDER BY SNO,SCORE DESC
```

**例3.46** 求选课在三门以上且各门课程均及格的学生的学号及其总成绩，查询结果按总成绩降序列出。

```
SELECT SNO,SUM(SCORE) AS TotalScore FROM SC
WHERE SCORE>=60
GROUP BY SNO
HAVING COUNT(*)>=3
ORDER BY SUM(SCORE) DESC
```



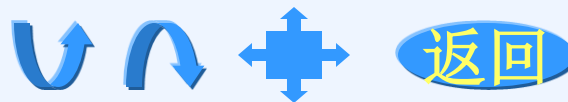


➤ 此语句为**分组排序**，执行过程如下：

1. (FROM) 取出整个SC
2. (WHERE) 筛选SCORE $\geq$ 60的元组
3. (GROUP BY) 将选出的元组按SNO分组
4. (HAVING) 筛选选课三门以上的分组
5. (SELECT) 以剩下的组中提取学号和总成绩
6. (ORDER BY) 将选取结果排序

➤ ORDER BY SUM(SCORE) DESC 可以改写成  
**ORDER BY 2 DESC**

**2** 代表查询结果的第二列。





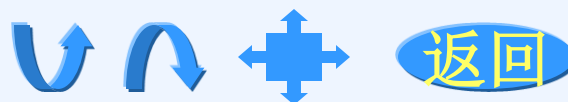
### 3.3.6 数据表连接及连接查询

- 数据表之间的联系是通过表的**字段值**来体现的，这种**字段**称为**连接字段**。
- 连接操作的**目的**就是通过加在连接字段的条件将多个表连接起来，以便从多个表中查询数据。
- 前面的查询都是针对一个表进行的，当查询同时涉及两个以上的表时，称为**连接查询**。
- 表的连接方法有两种：
  - **方法1**：表之间满足一定的条件的行进行连接，此时**FROM**子句中指明进行连接的表名，**WHERE**子句指明连接的列名及其连接条件。
  - **方法2**：利用关键字**JOIN**进行连接。



具体分为以下几种：

- **INNER JOIN**：显示符合条件的记录，此为默认值；
- **LEFT (OUTER) JOIN**：显示符合条件的数据行以及左边表中不符合条件的数据行，此时右边数据行会以NULL来显示，此称为左连接；
- **RIGHT (OUTER) JOIN**：显示符合条件的数据行以及右边表中不符合条件的数据行，此时左边数据行会以NULL来显示，此称为右连接；
- **FULL (OUTER) JOIN**：显示符合条件的数据行以及左边表和右边表中不符合条件的数据行，此时缺乏数据的数据行会以NULL来显示；
- **CROSS JOIN**：会将一个表的每一笔数据和另一表的每笔数据匹配成新的数据行。
- 当将JOIN 关键词放于FROM子句中时，应有关键词ON与之相对应，以表明连接的条件。





### 3.3.6.1 等值连接与非等值连接

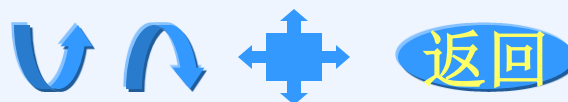
- 例3.47 查询刘伟老师所讲授的课程。
- 方法1:

```
SELECT T.TNO ,TN,CNO  
FROM T,TC  
WHERE (T.TNO = TC. TNO) AND (TN='刘伟' )
```

- 这里，TN='刘伟' 为查询条件，而T.TNO = TC.TNO 为连接条件，TNO为连接字段。连接条件的一般格式为：

[<表名1>.] <列名1> <比较运算符> [<表名2>.] <列名2>

- 其中,比较运算符主要有：=、>、<、>=、<=、!=。
- 当比较运算符为“=”时，称为等值连接，其他情况为非等值连接。



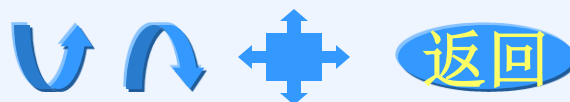




- 引用列名TNO时要加上**表名前缀**，是因为两个表中的列名相同，必须用表名前缀来确切说明所指列属于哪个表，以避免二义性。如果列名是唯一的，比如TN，就不必须加前缀。
- 上面的操作是将T表中的TNO 和TC表中的TNO相等的行**连接**，同时**选取**TN为“刘伟“的行，然后再在TN，CNO列上**投影**，这是**连接**、**选取**和**投影**的操作组合。

➤ **方法2:**

```
SELECT T.TNO,TN,CNO  
FROM T INNER JOIN TC  
ON T.TNO=TC.TNO AND T.TN='刘伟'
```





**例3.48** 查询所有选课学生的学号、姓名、选课名称及成绩。

```
SELECT S.SNO,SN,CN,SCORE  
FROM S,C,SC  
WHERE S.SNO=SC.SNO AND SC.CNO=C.CNO
```

- 本例涉及三个表，**WHERE**子句中有两个连接条件。当有两个以上的表进行连接时，称为**多表连接**。



### 3.3.6.2 自身连接

- 当一个表与其自己进行连接操作时，称为表的**自身连接**。

**例3.49** 查询所有比刘伟工资高的教师姓名、性别、工资和刘伟的工资。

- 要查询的内容均在同一表T中，可以将表T分别取两个**别名**，一个是X，一个是Y。将X, Y 中满足比刘伟工资高的行连接起来。这实际上是同一表T的自身连接。

- **方法1:**

```
SELECT X.TN,X.SAL AS SAL_a,Y.SAL AS SAL_b  
FROM T AS X ,T AS Y  
WHERE X.SAL>Y.SAL AND Y.TN='刘伟'
```



➤ 方法2:

```
SELECT X.TN, X.SAL, Y.SAL  
FROM T AS X INNER JOIN T AS Y  
ON X.SAL > Y.SAL AND Y.TN = '刘伟'
```

➤ 方法3:

```
SELECT R1.TN, R1.SAL, R2.SAL FROM  
(SELECT TN, SAL FROM T) AS R1  
INNER JOIN  
(SELECT SAL FROM T  
WHERE TN = '刘伟') AS R2  
ON R1.SAL > R2.SAL
```



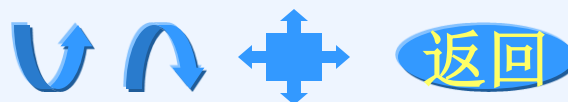
### 例3.50 检索所有学生姓名，年龄和选课名称。

#### ➤ 方法1:

```
SELECT SN,AGE,CN
FROM S,C,SC
WHERE S.SNO=SC.SNO AND SC.CNO=C.CNO
```

#### ➤ 方法2:

```
SELECT R3.SNO,R3.SN,R3.AGE,R4.CN
FROM
(SELECT SNO,SN,AGE FROM S) AS R3
INNER JOIN
(SELECT R2.SNO,R1.CN
FROM
(SELECT CNO,CN FROM C) AS R1
INNER JOIN
(SELECT SNO,CNO FROM SC) AS R2
ON R1.CNO=R2.CNO) AS R4
ON R3.SNO=R4.SNO
```





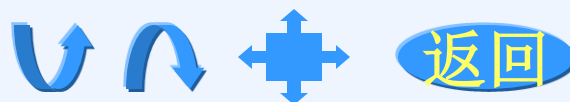
### 3.3.6.3 外连接

- 在上面的连接操作中，不满足连接条件的元组不能作为查询结果输出。
- 如例3.48的查询结果只包括有选课记录的学生，而不会有吴丽同学的信息。若将例3.48改成：

**例3.51** 查询所有学生的学号、姓名、选课名称及成绩。（没有选课的同学的选课信息显示为空）则应写成如下的SQL语句。

```
SELECT S.SNO,SN,CN,SCORE  
FROM S  
LEFT OUTER JOIN SC  
ON S.SNO=SC.SNO  
LEFT OUTER JOIN C  
ON C.CNO=SC.CNO
```

- 则查询结果只包括所有的学生，没有选课的吴丽同学的选课信息显示为空。





### 3.3.7 子查询

- 在WHERE子句中包含一个形如SELECT-FROM-WHERE的查询块，此查询块称为子查询或嵌套查询，包含子查询的语句称为父查询或外部查询。
- 嵌套查询可以将一系列简单查询构成复杂查询，增强查询能力。
- 子查询的嵌套层次最多可达到255层，以层层嵌套的方式构造查询充分体现了SQL“结构化”的特点。
- 嵌套查询在执行时由里向外处理，每个子查询是在上一级外部查询处理之前完成，父查询要用到子查询的结果。



### 3.3.7.1 返回一个值的子查询

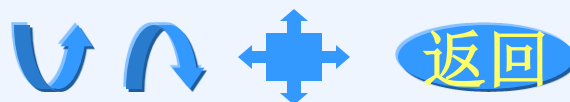
- 当子查询的返回值只有一个时，可以使用比较运算符（=，>，<，>=，<=，!=）将父查询和子查询连接起来。

**例3.52** 查询与刘伟教师职称相同的教师号、姓名。

```
SELECT TNO,TN
FROM T
WHERE PROF=(SELECT PROF
FROM T
WHERE TN='刘伟')
```

- 此查询相当于分成两个查询块来执行。先执行子查询：

```
SELECT PROF
FROM T
WHERE TN='刘伟'
```

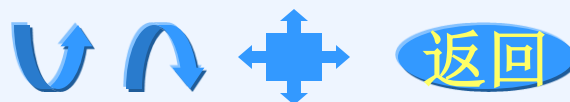






- 子查询向主查询只返回一个值，即刘伟教师的职称“讲师”，然后以此作为父查询的条件，相当于再执行父查询，查询所有职称为“讲师”的教师号、姓名。

```
SELECT TNO, TN  
FROM T  
WHERE PROF='讲师'
```





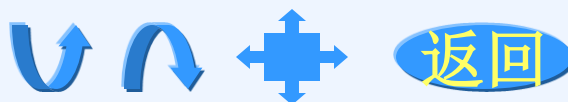
### 3.3.7.2 返回一组值的子查询

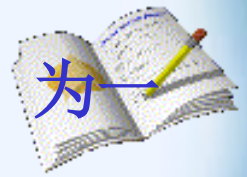
- 如果子查询的返回值不止一个，而是一个集合时，则不能直接使用比较运算符，可以在比较运算符和子查询之间插入ANY或ALL。其具体含义详见以下各例。

#### 1. 使用ANY

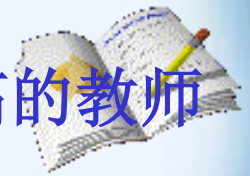
**例3.53** 查询讲授课程号为C5的教师姓名。

```
SELECT TN  
FROM T  
WHERE TNO=ANY  
      (SELECT TNO  
       FROM TC  
       WHERE CNO='C5')
```





- 先执行子查询，找到讲授课程号为C5的教师号，为一组值构成的集合 (T2, T3, T5)；
- 再执行父查询，其中ANY的含义为任意一个，查询教师号为T2、T3、T5的教师的姓名。
- 该例也可以使用前面所讲的连接操作来实现：  
SELECT TN  
FROM T,TC  
WHERE T.TNO=TC.TNO  
AND TC.CNO='C5'
- 可见，对于同一查询可使用子查询和连接两种方法来解决，可根据习惯任意选用。



**例3.54** 查询其他系中比计算机系某一教师工资高的教师的姓名和工资。

```
SELECT TN,SAL
FROM T
WHERE SAL>ANY
      (SELECT SAL
       FROM T
        WHERE DEPT='计算机')
AND DEPT!= '计算机'
```

/\*注意：此行是父查询中的条件\*/

- 先执行子查询，找到计算机系中所有教师的工资集合(1500, 900)；
- 再执行父查询，查询所有不是计算机系且工资高于1500或900的教师姓名和工资。



➤ 此查询也可以写成:

```
SELECT TN,SAL  
FROM T  
WHERE SAL>  
      (SELECT MIN(SAL )  
FROM T  
WHERE DEPT='计算机')  
AND DEPT!= '计算机'
```

- 先执行子查询，利用库函数MIN找到计算机系中所有教师的最低工资——900；
- 再执行父查询，查询所有不是计算机系且工资高于900的教师。



## 2. 使用IN

- 可以使用IN代替“=ANY”。
- 例3.55（题目同3.53）

```
SELECT TN  
FROM T  
WHERE TNO IN  
      (SELECT TNO  
       FROM TC  
       WHERE CNO='C5')
```



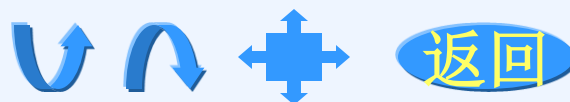
### 3. 使用ALL

- ALL的含义为**全部**。

**例3.56** 查询其他系中比计算机系所有教师工资都高的教师的姓名和工资。

```
SELECT TN,SAL
FROM T
WHERE SAL>ALL
      (SELECT SAL
       FROM T
        WHERE DEPT='计算机')
AND DEPT!= '计算机'
```

- 子查询找到计算机系中所有教师的工资集合(1500, 900)；
- 父查询找到所有不是计算机系且工资高于1500的教师姓名和工资。





➤ 此查询也可以写成:

```
SELECT TN,SAL
FROM T
WHERE SAL>
      (SELECT MAX(SAL )
FROM T
      WHERE DEPT='计算机')
AND DEPT!= '计算机'
```

➤ 库函数MAX的作用是找到计算机系中所有教师的最高工资1500。





例3.57 查询不讲授课程号为C5的教师姓名。

```
SELECT DISTINCT TN  
FROM T  
WHERE 'C5' !=ALL  
      (SELECT CNO  
FROM TC  
WHERE TNO=T.TNO)
```

- !=ALL的含义为不等于子查询结果中的任何一个值，也可使用NOT IN代替!=ALL。
- 子查询包含普通子查询和相关子查询。
- 前面所讲的子查询均为普通子查询，而本例中子查询的查询条件引用了父查询表中的属性值（T表的TNO值），我们把这类查询称为相关子查询。



## ➤ 二者的执行方式不同：

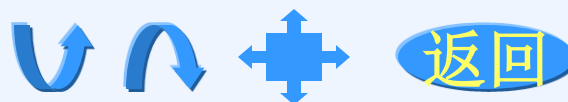
### ➤ 普通子查询的执行顺序是：

- 首先执行子查询，然后把子查询的结果作为父查询的查询条件的值。
- 普通子查询只执行一次，而父查询所涉及的所有记录行都与其查询结果进行比较以确定查询结果集合。

### ➤ 相关子查询的执行顺序是：

- 首先选取父查询表中的第一行记录，内部的子查询利用此行中相关的属性值进行查询，
- 然后父查询根据子查询返回的结果判断此行是否满足查询条件。如果满足条件，则把该行放入父查询的查询结果集合中。重复执行这一过程，直到处理完父查询表中的每一行数据。

## ➤ 由此可以看出，相关子查询的执行次数是由父查询表的行数决定的。





- 如上例表T中每的一行即每个教师记录都要执行一次子查询以确定该教师是否讲授C5这门课，当 C5不是教师的任一门课时，则该教师被选取。
- 以下几例均为相关子查询。

#### 4. 使用EXISTS

- EXISTS表示存在量词，带有EXISTS的子查询不返回任何实际数据，它只得到逻辑值“真”或“假”。
- 当子查询的的查询结果集合为非空时，外层的WHERE子句返回真值，否则返回假值。NOT EXISTS与此相反。
- 含有IN的查询通常可用EXISTS表示，但反过来不一定。



例3. 58 (题目同3. 53) 略

```
SELECT TN
FROM T
WHERE EXISTS
  (SELECT *
   FROM TC
    WHERE TNO=T.TNO
   AND CNO='C5')
```

- ▶ 当子查询TC表存在一行记录满足其WHERE子句中的条件时，则父查询便得到一个TN值，重复执行以上过程，直到得出最后结果。



### 例3. 59 查询选修所有课程的学生姓名

```
SELECT SN FROM S
WHERE NOT EXISTS
(SELECT * FROM C
    WHERE NOT EXISTS
    (SELECT * FROM SC
    WHERE SNO = S.SNO
    AND CNO=C.CNO))
```

- 选出这样一些学生名单，在SC表中不存在他们没有选修课程的记录。

## 3.4 SQL数据更新



- SQL语言的数据更新语句DML主要包括插入数据、修改数据和删除数据三种语句。

### 3.4.1 插入数据记录

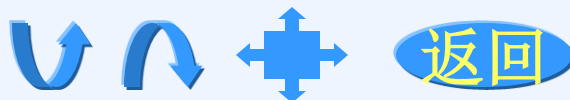
- 插入数据是把新的记录插入到一个存在的表中。插入数据使用语句INSERT INTO，可分为以下几种情况。

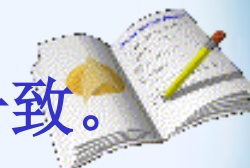
#### 3.4.1.1 插入一行新记录

- 语法格式为：

INSERT INTO <表名>[(<列名1>[, <列名2>...])] VALUES(<值>)

- 其中，<表名>是指要插入新记录的表  
<列名>是可选项，指定待添加数据的列  
VALUES子句指定待添加数据的具体值。





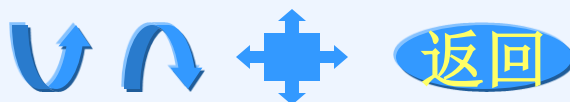
- 列名的排列顺序不一定要和表定义时的顺序一致。
- 但当指定列名表时VALUES子句值的排列顺序必须和列名表中的列名排列顺序一致，个数相等，数据类型一一对应。
- **例3.60** 在S表中插入一条学生记录（学号：S7；姓名：郑冬；性别：女；年龄：21；系别：计算机）。

INSERT INTO S

VALUES ('s7','郑冬','女',21,'计算机')

注意：

- 必须用**逗号**将各个数据分开，字符型数据要用**单引号**括起来。
- INTO子句中没有指定列名，则新插入的记录必须在每个属性列上均有值，且VALUES子句中值的排列顺序要和表中各属性列的排列顺序一致。







### 3.4.1.2 插入一行的部分数据值

**例3.61** 在SC表中插入一条选课记录（'S7', 'C1'）。

```
INSERT INTO SC (SNO,CNO)  
VALUES ('s7','c1')
```

- 将VALUES子句中的值按照INTO子句中指定列名的顺序插入到表中
- 对于INTO子句中没有出现的列，则新插入的记录在这些列上将取空值，如上例的SCORE即赋空值。
- 但在表定义时有NOT NULL约束的属性列不能取空值。





### 3.4.1.3 插入多行记录

➤ 用于表间的拷贝，将一个表中的数据抽取数行插入另一表中，可以通过子查询来实现。

➤ 插入数据的命令语法格式为：

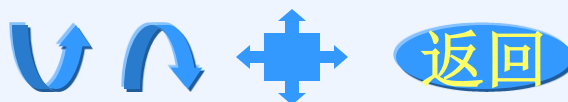
```
INSERT INTO <表名> [( <列名1>[, <列名2>...])]
```

子查询

➤ **例3.62** 求出各系教师的平均工资，把结果存放在新表AVGSAL中。

➤ 首先建立新表AVGSAL，用来存放系名和各系的平均工资

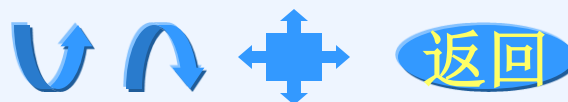
```
CREATE TABLE AVGSAL  
(DEPARTMENT VARCHAR(20),  
AVGSAL SMALLINT)
```





- 然后利用子查询求出T表中各系的平均工资，把结果存放在新表AVGSAL中。

```
INSERT INTO AVGSAL  
SELECT DEPT,AVG(SAL)  
FROM T  
GROUP BY DEPT
```





## 2.4.2 修改数据记录

- SQL语言可以使用UPDATE语句对表中的一行或多行记录的某些列值进行修改，其语法格式为：

UPDATE <表名>

SET <列名>=<表达式> [, <列名>=<表达式>]...

[WHERE <条件>]

其中：

- <表名>是指要修改的表
- SET子句给出要修改的列及其修改后的值
- WHERE子句指定待修改的记录应当满足的条件，WHERE子句省略时，则修改表中的所有记录。



### 3.4.2.1 修改一行

**例3.63** 把刘伟教师转到信息系。

```
UPDATE T  
SET DEPT='信息'  
WHERE TN='刘伟'
```

### 3.4.2.2 修改多行

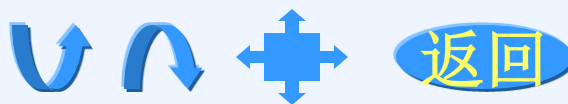
**例3.64** 将所有学生年龄增加1岁

```
UPDATE S  
SET AGE=AGE+1
```



**例3.65** 把教师表中工资小于等于1000元的讲师的工资提高20%。

```
UPDATE T  
SET SAL=1.2*SAL  
WHERE PROF='讲师'  
AND SAL <=1000
```



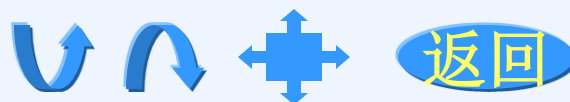


### 3.4.2.3 用子查询选择要修改的行

**例3.66** 把讲授C5课程的教师的岗位津贴增加100元。

```
UPDATE T
SET COMN=COMN+100
WHERE TNO IN
(SELECT T.TNO
FROM T,TC
WHERE T.TNO=TC.TNO
AND TC.CNO='C5')
```

➤ 子查询的作用是得到讲授C5课程的教师号。





### 3.4.2.4 用子查询提供要修改的值

例3.67 把所有教师的工资提高到平均工资的1.2倍

```
UPDATE T  
SET SAL =  
(SELECT 1.2*AVG(SAL) FROM T)
```

➤ 子查询的作用是得到所有教师的平均工资。



### 3.4.3 删除数据记录

- 使用DELETE语句可以删除表中的一行或多行记录，其语法格式为：

DELETE

FROM<表名>

[WHERE <条件>]

其中，

- <表名>是指要删除数据的表。
- WHERE子句指定待删除的记录应当满足的条件，WHERE子句省略时，则删除表中的所有记录。





### 3.4.3.1 删除一行记录

**例3.68** 删除刘伟教师的记录。

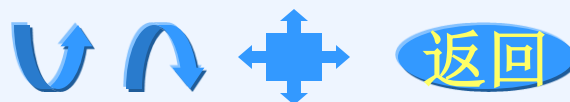
```
DELETE  
FROM T  
WHERE TN='刘伟'
```

### 3.4.3.2 删除多行记录

**例3.69** 删除所有教师的授课记录

```
DELETE  
FROM TC
```

- 执行此语句后，TC表即为一个空表，但其定义仍存在数据字典中。

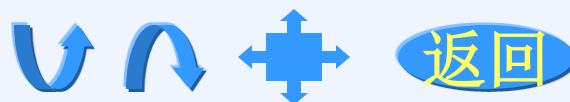




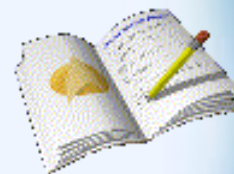
### 3.4.3.3 利用子查询选择要删除的行

例3.70 删除刘伟教师授课的记录。

```
DELETE  
FROM TC  
WHERE TNO=  
(SELECT TNO  
FROM T  
WHERE TN=' 刘伟' )
```



## 3.5 视图



- 视图是**虚表**，其数据不存储，其记录来自基本表，只在数据库中**存储其定义**。
- 视图在概念上与基本表等同，用户可以在视图上再定义视图，可以对视图进行查询、删除、更新等操作。

### 3.5.1 定义和删除视图

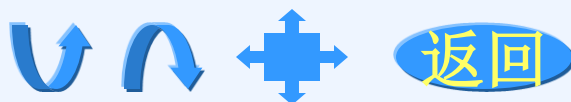
#### 3.5.1.1 定义视图

- 定义视图使用语句CREATE VIEW，其语法格式为：

CREATE VIEW <视图名>[(<视图列表>)]

AS <子查询>

- 其中，<视图列表>为可选项，省略时，视图的列名由子查询的结果决定。



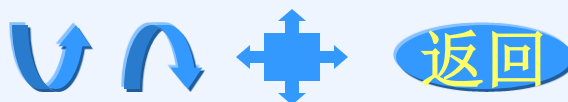


以下两种情况下，视图列名不可省略：

1. 视图由多个表连接得到，在不同的表中存在同名列，则需指定列名；
  2. 当视图的列名为表达式或库函数的计算结果时，而不是单纯的属性名时，则需指明列名。
- 在子查询中不许使用ORDER BY 子句和DISTINCT短语，如果需要排序，则可在视图定义后，对视图查询时再进行排序。

例3.71 创建一个计算机系教师情况的视图SUB\_T。

```
CREATE VIEW SUB_T  
AS SELECT TNO,TN,PROF  
FROM T WHERE DEPT='计算机'
```





➤ 其中：

- 视图名字为SUB\_T，省略了视图列表。
  - 视图由子查询中的三列TNO, TN, PROF组成。
  - 视图创建后，对视图SUB\_T的数据的访问只限制在计算机系内，且只能访问TNO, TN, PROF三列的内容，从而达到了数据保密的目的。
- 视图创建后，只在数据字典中存放视图的定义，而其中的子查询SELECT语句并不执行。
- 只有当用户对视图进行操作时，才按照视图的定义将数据从基本表中取出。



**例3.72** 创建一学生情况视图S\_SC\_C（包括学号、姓名、课程名及成绩）。

```
CREATE VIEW S_SC_C(SNO, SN, CN, SCORE)
AS SELECT S.SNO, SN, CN, SCORE
FROM S, C, SC
WHERE S.SNO = SC.SNO AND
SC.CNO = C.CNO
```

- 此视图由三个表连接得到，在S表和SC表中均存在SNO列，则需指定视图列名。



### 例3.73 创建一学生平均成绩视图S\_AVG

```
CREATE VIEW S_AVG (SNO,AVG)  
AS SELECT SNO, AVG (SCORE)  
FROM SC GROUP BY SNO
```

- 此视图的列名之一AVG为库函数的计算结果，则在定义时需指明列名。



### 3.5.1.2 删除视图

- 视图定义后可随时删除，删除视图的语法格式为：

DROP VIEW <视图名>

**例3.74** 删除计算机系教师情况的视图SUB\_T。

DROP VIEW SUB\_T

- 视图删除后，只会删除该视图在数据字典中的定义，而与该视图有关的基本表中的数据不会受任何影响，由此视图导出的其他视图的定义不会删除，但已无任何意义。用户应该把这些视图删除。





### 3.5.2 查询视图

- 视图定义后，对视图的查询操作如同对基本表的查询操作一样。

**例3.75** 查找视图SUB\_T中职称为教授的教师号和姓名。

```
SELECT TNO, TN  
FROM SUB_T  
WHERE PROF='教授'
```



- 此查询的执行过程是系统首先从数据字典中找到SUB\_T的定义，然后把此定义和用户的查询结合起来，转换成等价的对基本表T的查询，这一转换过程称为视图消解（View Resolution），相当于执行以下查询：

```
SELECT TNO, TN  
FROM T  
WHERE DEPT ='计算机'    AND PROF='教授'
```

- 由上例可以看出，当对一个基本表进行复杂的查询时，可以先对基本表建立一个视图，然后只需对此视图进行查询，这样就不必再键入复杂的查询语句，而将一个复杂的查询转换成一个简单的查询，从而简化了查询操作。



### 3.5.3 更新视图

- 由于视图是一张虚表，所以对视图的更新，最终实际上是转换成对基本表的更新。
- 其更新操作包括**插入**、**修改**和**删除**数据，
- 其语法格式如同对基本表的更新操作一样。
- 有些更新在理论上是不可能的，有些实现起来比较困难，以下仅考虑可以更新的视图。



### 3.5.3.1 插入 (INSERT)

**例3.76** 向计算机系教师视图SUB\_T中插入一条记录（教师号：T6；姓名：李丹；职称：副教授）。

```
INSERT INTO SUB_T  
VALUES ('T6','李丹','副教授')
```

- 系统在执行此语句时，首先从数据字典中找到SUB\_T的定义，然后把此定义和插入操作结合起来，转换成等价的对基本表T的插入。相当于执行以下操作：

```
INSERT INTO T  
VALUES ('T6','李丹', '副教授', '计算机')
```



### 3.5.3.2 修改 (UPDATE)

**例3.77** 将计算机系教师视图SUB\_T中刘伟的职称改为“副教授”。

```
UPDATE SUB_T  
SET PROF = '副教授'  
WHERE TN = '刘伟'
```

➤ 转换成对基本表的修改操作:

```
UPDATE T  
SET PROF='副教授'  
WHERE TN='刘伟' AND DEPT='计算机'
```



### 3.5.3.3 删除 (DELETE)

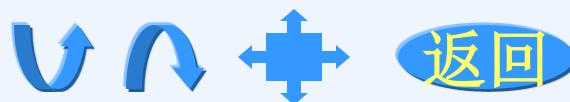
**例3.78** 删除计算机系教师视图SUB\_T中刘伟教师的记录。

```
DELETE  
FROM SUB_T  
WHERE TN='刘伟'
```

➤ 转换成对基本表的删除操作:

```
DELETE  
FROM T  
WHERE TN='刘伟' AND DEPT='计算机'
```

➤ 由于视图中的数据不是存放在视图中的，即视图没有相应的存储空间，对视图的一切操作最终都要转换成对基本表的操作，这样看来使操作更加复杂，那么为什么还要使用视图呢？





## ➤ 使用视图有如下几个优点：

1. **利于数据保密**，对不同的用户定义不同的视图，使用户只能看到与自己有关的数据。

例如，对教师表创建了计算机系视图，本系教师只能使用此视图，而无法访问其他系教师的数据。

2. **简化查询操作**，为复杂的查询建立一个视图，用户不必键入复杂的查询语句，只需针对此视图做简单的查询即可。如例3.75。

3. **保证数据的逻辑独立性**。对于视图的操作，比如查询，只依赖于视图的定义。当构成视图的基本表要修改时，只需修改视图定义中的子查询部分。而基于视图的查询不用改变。这就是第一章介绍过的外模式与模式之间的独立性，即**数据的逻辑独立性**。

## 3.6 SQL 数据控制



数据库中的数据由多个用户共享，为保证数据库的安全，SQL语言提供数据控制语句DCL(Data Control Language)对数据库进行统一的控制管理。

### 3.6.1 权限与角色

#### 3.6.1.1 权限

➤在SQL系统中，有两个安全机制：

- 一种是上一节介绍的视图机制，当用户通过视图访问数据库时，不能访问此视图外的数据，它提供了一定的安全性。
- 主要的安全机制是权限机制。
  - 权限机制的基本思想是给用户授予不同类型的权限，在必要时，可以收回授权。
  - 使用户能够进行的数据库操作以及所操作的数据限定在指定的范围内，禁止用户超越权限对数据库进行非法的操作，从而保证数据库的安全性。





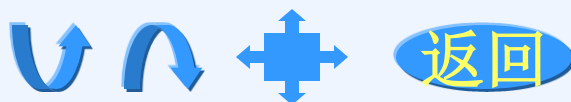
在SQL SERVER中，权限可分为系统权限和对象权限。



- **系统权限**由数据库管理员授予其他用户，是指数据库用户能够对数据库系统进行某种特定的操作的权力。
  - 如创建一个基本表（CREATE TABLE）
- **对象权限**由创建基本表、视图等数据库对象的用户授予其他用户，是指数据库用户在指定的数据库对象上进行某种特定的操作的权力。
  - 如查询（SELECT）、插入（INSERT）、修改（UPDATE）和删除（DELETE）等操作。

### 3.6.1.2 角色

- **角色**是多种权限的集合，可以把角色授予用户或其他角色。当要为某一用户同时授予或收回多项权限时，则可以把这些权限定义为一个角色，对此角色进行操作。这样就避免了许多重复性的工作，简化了管理数据库用户权限的工作。





## 3.6.2 系统权限与角色的授予与收回

### ➤ 3.6.2.1 系统权限与角色的授予

- SQL语言使用GRANT语句为用户授予系统权限，其语法格式为：

```
GRANT <系统权限>|<角色> [, <系统权限>|<角色>]...  
TO <用户名>|<角色>|PUBLIC[, <用户名>|<角色>]...  
[WITH ADMIN OPTION]
```

- 其语义为：将指定的系统权限授予指定的用户或角色。
- 其中：
  - PUBLIC代表数据库中的全部用户。
  - WITH ADMIN OPTION为可选项，指定后则允许被授权的用户将指定的系统特权或角色再授予其他用户或角色。

例3.79 为用户ZHANGSAN授予CREATE TABLE的系统权限。



```
GRANT CREATE TABLE  
TO ZHANGSAN
```

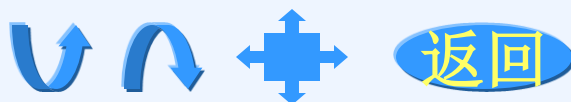
### 3.6.2.2 系统权限与角色的收回

- 数据库管理员可以使用REVOKE语句收回系统权限，其语法格式为：

```
REVOKE <系统权限>|<角色> [, <系统权限>|<角色>]...  
FROM <用户名>|<角色>|PUBLIC[, <用户名>|<角色>]...
```

- 例3.80 收回用户ZHANGSAN所拥有的CREATE TABLE的系统权限。

```
REVOKE CREATE TABLE  
FROM ZHANGSAN
```



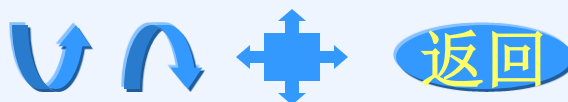


### 3.6.3 对象权限与角色的授予与收回

#### 3.6.3.1 对象权限与角色的授予

- 数据库管理员拥有系统权限，而作为数据库的普通用户，只对自己创建的基本表、视图等数据库对象拥有对象权限。
- 如果要共享其他的数据库对象，则必须授予他一定的对象权限。
- 同系统权限的授予类似，SQL语言使用GRANT语句为用户授予对象权限，其语法格式为：

```
GRANT ALL|<对象权限>[(列名[, 列名]...)][, <对象权限>]...ON <
对象名>
TO <用户名>|<角色>|PUBLIC[, <用户名>|<角色>]...
[WITH GRANT OPTION]
```





- 其语义为：将指定的操作对象的对象权限授予指定的用户或角色。
- 其中：
  - ALL代表所有的对象权限。
  - 列名用于指定要授权的数据库对象的一列或多列。如果不指定列名，被授权的用户将在数据库对象的所有列上均拥有指定的特权。
    - 实际上，只有当授予INSERT、UPDATE权限时才需指定列名。
  - ON子句用于指定要授予对象权限的数据库对象名，可以是基本表名、视图名等。
  - WITH ADMIN OPTION为可选项，指定后则允许被授权的用户将权限再授予其他用户或角色。

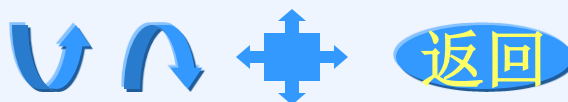


**例3.81** 将对S表和T表的所有对象权限授予USER1和USER2。

```
GRANT ALL  
ON S, T  
TO USER1, USER2
```

**例3.82** 将对C表的查询权限授予所有用户。

```
GRANT SELECT  
ON C  
TO PUBLIC
```





**例3.83** 将查询T表和修改教师职称的权限授予USER3, 并允许将此权限授予其他用户。

```
GRANT SELECT, UPDATE (PROF)
ON T
TO USER3
WITH ADMIN OPTION
```

➤ USER3具有此对象权限，并可使用GRANT命令给其他用户授权，如下例，USER3将此权限授予USER4：

```
GRANT SELECT, UPDATE (PROF)
ON T
TO USER4
```



### 3.6.3.2 对象权限与角色的收回

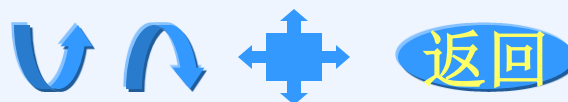
- 所有授予出去的权力在必要时都可以由数据库管理员和授权者收回，收回对象权限仍然使用REVOKE语句，其语法格式为：

```
REVOKE <对象权限>|<角色> [, <对象权限>|<角色>  
>]...
```

```
FROM <用户名>|<角色>|PUBLIC[, <用户名>|<角色>  
>]...
```

**例3.84** 收回用户USER1对C表的查询权限。

```
REVOKE SELECT  
ON C  
FROM USER1
```





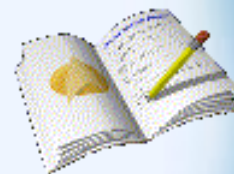


### 例3.85 收回用户USER3查询T表和修改教师职称的权限。

```
REVOKE SELECT, UPDATE (PROF)
ON T
FROM USER3
```

- 在例3.83中，USER3将对T表的权限授予了USER4，在收回USER3对T表的权限的同时，系统会自动收回USER4对T表的权限。

## 小 结

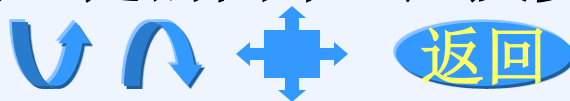


- 本章以SQL SERVER为例，详细介绍了SQL语言的使用方法。
- 在讲解SQL语言的同时，进一步介绍了关系数据库的有关概念，如索引和视图的概念及其作用。
- SQL语言具有数据定义、数据查询、数据更新、数据控制四大功能。其全部功能可以用表3.3的9个动词概括出来。

表3.3 SQL语言的动词

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

➤其中：其数据查询功能最为丰富和复杂，也非常重要，初学者掌握起来有一定的困难，应反复上机加强练习。





## ➤ 本章要求

- 了解 SQL语言的特点,
- 掌握SQL语言的四大功能及使用方法,
- 重点掌握其数据查询功能及其使用。

## 3.2.2 定义、修改和撤消数据库的用户

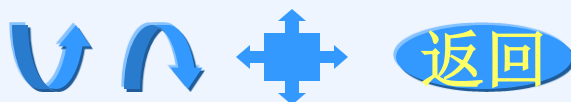


### 3.2.2.1 建立数据库用户

- 数据库用户是指能够登录到数据库，并能够对数据库进行存取操作的用户。
- 当SQL SERVER系统安装完毕后，数据库管理员就可以通过CREATE USER语句建立其他数据库用户了。
- 语法格式为：

**CREATE USER <用户名> IDENTIFIED BY <口令>**

- <用户名>指定数据库用户的帐号名字，即用户标识符，
- <口令>指用户登录到数据库系统时使用的口令，
- 这里的用户名和口令可以与用户登录到操作系统时所使用的用户名和口令不同。





- 例3.1 建立一个新用户，其名称为ZHANGSAN，登录口令为123。

**CREATE USER ZHANGSAN IDENTIFIED BY 123**

### 3.2.2.2 更改数据库用户的口令

- 数据库用户最初的口令是由数据库管理员指定的，数据库用户可以用ALTER USER命令来更改它，
- ALTER USER语句的基本语法格式为：

ALTER USER <用户名> IDENTIFIED BY <口令>

- 例3.2 将用户ZHANGSAN的口令改为456。

**ALTER USER ZHANGSAN IDENTIFIED BY 456**



### 3.2.2.3 删除用户

- 随着数据库应用的发展和变化，数据库的用户也会发生变化。
- 如果某些数据库用户不再需要使用数据库，数据库管理员就可以使用**DROP USER**把该用户删掉，
- **DROP USER** 语句的基本语法格式为：

**DROP USER <用户名>**

- 例3.3 删除用户ZHANGSAN  
**DROP USER ZHANGSAN**

- 注意：删除数据库用户之前应首先删除该用户建立的数据库对象，包括基本表、视图、索引等，否则系统将不允许删除这个用户。

