第3章

关系数据库标准语言SQL

SQL是关系数据库的标准语言，本章将详细介绍SQL的数据定义、数据查询、数据更新等功能。通过本章的学习，了解SQL的发展和特点，理解SQL数据库的体系结构，熟练掌握基本表、索引、视图的定义与删除，数据查询语句的各种用法，数据插入、删除和修改操作。

3.1 结构化查询语言概述

SQL(Structured Query Language，结构化查询语言)，并不是只支持查询操作，它包含数据定义、数据更新和数据控制等与数据库有关的全部功能，是一个通用的、功能极强的关系数据库标准语言。本节介绍SQL的产生与发展过程、特点以及主要功能。

微课视频

3.1.1 SQL的产生与发展

SQL最早是由IBM在20世纪70年代开发，并作为System R项目的一部分实现，最初叫作Sequel(Structured English Query Language)。随着SQL的颁布，几乎所有的数据库产品都支持SQL，并对SQL进行了扩充与修改，SQL成为当前应用最为广泛的关系数据语言。

随着关系数据库系统和SQL的广泛应用，SQL的标准化工作已经历了十多年的发展，制定了多个SQL标准。

1982年，美国标准化局(American National Standard Institute，ANSI)开始制定SQL标准。

1986年10月，由美国国家标准化协会公布了SQL的第一个标准SQL-86。

1987年，国际标准化组织(ISO)通过了SQL标准SQL-86。

1989 年 4 月，国际标准化组织(ISO)对 SQL-86 进行了补充，推出了具备完整性特征的 SQL，称为 SQL-89。

1992 年 11 月，ISO 又公布了新的 SQL 标准，称为 SQL-92，也称为 SQL2。增加了新数据类型，丰富了数据操作和完整性支持等。

1999 年推出 SQL-99，也称为 SQL3。支持面向对象的一些特征、抽象数据类型、行对象、列对象等，对递归和触发器等复杂操作予以规范化定义，废弃了 SQL2 的分级，但定义了 Core-SQL 及扩展的 SQL。

后来推出的版本有 SQL 2003、SQL 2006、SQL 2008，SQL Server 2019 是目前通行的数据库语言 SQL 的较新版本。

现在 SQL 已经成为一种标准，常用的 Oracle、SQL Server、MySQL、DB2、Ingres 以及中型数据库 FoxPro、Access 等数据库产品对 SQL 的支持大部分相似，但它们之间存在着一定的差异，与 SQL 标准的符合程度也不相同，一般在 85%以上。因此，在使用具体的数据库管理系统产品时，要参阅相应手册。


微课视频

3.1.2　SQL 的特点

SQL 之所以能够被用户和业界所接受与广泛应用，是因为它是一个综合的、功能强大的且简洁易学的语言。SQL 集数据查询、数据操纵、数据定义和数据控制功能于一体，其主要特点如下。

1. 一体化

SQL 集数据定义语言(DDL)、数据操纵语言(DML)、数据控制语言(DCL)功能于一体，语言风格统一，能够独立地完成数据库应用系统的全部需要。

2. 高度非过程化

SQL 是一种高度的非过程化语言，用户在使用 SQL 操作数据时，只要提出“做什么”，无须了解具体操作过程以及存取路径的选择，只要指明所需数据，由数据库管理系统自动完成全部工作。

3. 面向集合的操作

SQL 是一种面向集合的语言，每个命令的操作对象是一个或多个关系，结果仍是一个关系，而且一次插入、删除和更新操作的对象也可以是元组的集合。

4. 提供多种使用方式

SQL 既是独立语言，又是嵌入式语言。独立地应用于联机交互，适用于终端用户、应用程序员和 DBA；嵌入式语言使其嵌入到高级语言(如 C、C++、Java)中使用，供应用程序员开发应用程序。

5. 功能强大，简洁易用

SQL 功能极强，只用 9 个动词就可以完成核心功能，如表 3.1 所示。SQL 的语法比较简单，类似于英语的自然语言，不区分大小写，容易学习和掌握。

表 3.1 SQL 语句的动词

SQL 功能	动 词	作 用
数据定义	Create	模式的定义，包括创建数据库、数据表、视图、索引等
	Alter	修改数据表、索引
	Drop	删除数据库、数据表、视图、索引等
数据查询	Select	数据检索
数据更新	Insert	插入数据
	Delete	删除数据
	Update	更新数据
数据控制	Grant	授权
	Revoke	撤销授权

3.1.3 SQL 与数据库三级模式

SQL 支持关系数据库的三级模式结构。如图 3.1 所示，外模式包括若干视图和部分基本表，模式包括若干基本表，内模式包括若干存储文件。

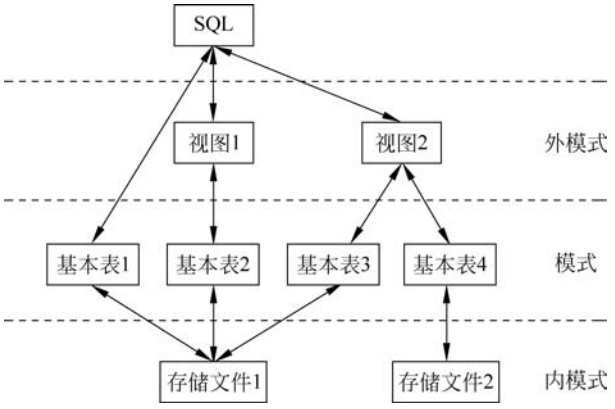

图 3.1 SQL 支持关系数据库的三级模式结构

1. 基本表

基本表是独立存在的表。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。一个基本表上可以建立多个索引，索引也存放在存储文件中。

2. 视图

视图是一个虚表，是从一个或几个基本表导出的表。在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。当基本表中数据发生变化时，从视图中查询出的数据也随之变化。通过视图用户可以抽取自己感兴趣的数据，用户还可以在视图上再定义视图。

3. 存储文件

存储文件的逻辑结构组成了关系数据库的内模式。存储文件的物理结构对最终用户是

隐蔽的。用户可以用 SQL 语句对视图和基本表进行查询等操作。

4. SQL 用户

SQL 可以作为独立的用户接口，供终端用户在交互环境下使用。SQL 语句也可以嵌入在程序设计语言中使用，如 C、C++、Java 等。这两种使用方式下，SQL 的语法结构基本保持一致。

3.2　数据定义

SQL 的数据定义功能包括对数据库模式、基本表、视图、索引的创建和删除。本章仍以学生-课程-成绩数据库为例讲解 SQL 的数据定义、数据查询、数据更新等功能。

学生-课程-成绩数据库包括学生信息表 S、课程信息表 C 和选课信息表 SC 三个表，各表中的数据示例如图 2.2 所示，这里不再赘述。

SQL 提供的一些基本数据类型如表 3.2 所示。

表 3.2　SQL 的主要数据类型

数据类型		含　义
数值型	INT,INTEGER	长整数,4Byte
	SMALLINT	短整数,2Byte
	TINYINT	微整数,1Byte
	BIGINT	大整数,8Byte
	REAL	单精度浮点数,取决于机器精度
	DOUBLE PRECISION	双精度浮点数,取决于机器精度
	FLOAT(n)	可选精度的浮点数,精度至少为 n 位数字
	NUMRIC(p, d)	定点数,由 p 位数字组成(不包括符号和小数点),小数点后面有 d 位数字
字符串型	CHAR(n)	长度为 n 的定长字符串,尾端空白字符保留
	VARCHAR(n)	长度为 n 的变长字符串,尾端空白字符删除
布尔型	BOOLEAN	逻辑布尔量
日期时间型	DATE	日期,包含年、月、日,格式为 YYYY-MM-DD
	TIME	时间,一日的时、分、秒,格式为 HH-MM-SS
	DATETIME	日期时间,包含年、月、日、时、分、秒 格式为 YYYY-MM-DD-HH-MM-SS


微课视频

3.2.1　模式的定义和删除

1. 模式的创建

创建一个 SQL 模式就是定义了一个命名的存储空间，并且包含模式中基本表、视图、索引等数据库对象的定义。一个关系数据库管理系统的实例中可以建立多个数据库，一个数据库中可以建立多个模式，一个模式下可以包括多个基本表、视图、索引等数据库对象。

在 SQL 中，一个模式由模式名和模式拥有者的用户名来确定，创建模式的语句如下。

```
CREATE SCHEMA <模式名> AUTHORIZATION <用户名>
```

【例 3-1】 为用户张明创建一个教学数据库模式 Study。

```
CREATE SCHEMA Study AUTHORIZATION 张明;
```

大多数 DBMS 不采用“模式”这个名词，而是采用“数据库”（DATABASE），在 DBMS 中创建模式不是用“CREATE SCHEMA…”，而是用“CREATE DATABASE…”，如现在最流行的 Oracle、SQL Server、MySQL 等数据库产品都是用“CREATE DATABASE …”语句创建数据库。

2. 删除模式

当一个模式及其所属的表、视图等对象都不需要时，可以用 DROP 语句删除。DROP 语句如下。

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

其中，CASCADE|RESTRICT 两者必选其一，CASCADE（级联）表示删除模式的同时把该模式中所有的数据库对象全部删除；RESTRICT（限制）表示如果该模式中定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行。当该模式中没有任何下属的对象时才能执行。

【例 3-2】 删除模式 Study，同时删除该模式中定义的所有对象。

```
DROP SCHEMA Study CASCADE;
```

3.2.2 基本表的定义、删除和修改

创建一个模式就是建立了一个数据库的命名空间，一个框架。接着要在这个空间中定义其所包含的对象，如表、视图、索引等。

1. 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> ]);
```

其中，<表名>是合法标识符，尖括号表示必选项，方括号表示可选项，一个表至少包含一列。如果完整性约束条件涉及表的多个属性列时，则必须定义在表级上，否则既可以定义在列级也可以定义在表级。完整性约束有以下几种。

NOT NULL：限制列取值非空。

DEFAULT：指定列的默认值。

UNIQUE：限制列取值不能重复。

CHECK：限制列的取值范围。

PRIMARY KEY：指定本列为主码。

FOREIGN KEY：定义本列为引用其他表的外码。使用形式为：

```
FOREIGN KEY(<外码列名>) REFERENCES <主码所在表名> (<主码列名>)
```

【例 3-3】 创建学生表 *S*。

```
CREATE TABLE S
(Sno CHAR(8) PRIMARY KEY,          /* 列级完整性约束条件,主码约束 */
Sname CHAR(8) NOT NULL,
Ssex  CHAR(2),
Sbirth  DATE,
Sdept  VARCHAR (20)
);
```

【例 3-4】 创建课程表 *C*。

```
CREATE TABLE  C
(Cno CHAR(3) PRIMARY KEY,          /* 列级完整性约束条件,主码约束 */
Cname VARCHAR(20) NOT NULL,        /*  Cname 不能取空值 */
Cpno  CHAR(3),
Ccredit TINYINT,
FOREIGN KEY (Cpno) REFERENCES C(Cno)
/* Cpno 是外码,取值要参照 C 表中的 Cno 列/
);
```

【例 3-5】 创建学生选课表 SC。

```
CREATE TABLE  SC
(Sno  CHAR(8),
Cno  CHAR(3),
Grade  SMALLINT,
PRIMARY KEY (Sno,Cno),
/* 主码由两个属性组成,必须在表级上定义完整性约束 */
FOREIGN KEY (Sno) REFERENCES S (Sno),
/* Sno 是外码,取值要参照 S 表中的 Sno 列 */
FOREIGN KEY (Cno) REFERENCES C (Cno)
/*  Cno 是外码,取值要参照 C 表中的 Cno 列 */
);
```

2. 修改基本表

```
ALTER TABLE <表名>
[ ALTER COLUMN <列名> <新数据类型>]
| [ ADD [COLUMN] <列名> <数据类型>
| [ DROP COLUMN <列名>  ]
| [ADD PRIMARY KEY(列名 [,… n ] )]
| [ADD FOREIGN KEY(列名)REFERNECES 表名(列名)];
```

【例 3-6】 在 SC 表中增加“考试时间”列，其数据类型为日期型。

```
ALTER  TABLE  SC  ADD  exam-time  DATE;
```

注意：新增加的列一律为空值。

【例 3-7】 将学生的学号由 8 个字符改为 11 个字符。

```
ALTER  TABLE  S;
ALTER  COLUMN  Sno  CHAR(11);
```

【例 3-8】 增加学生姓名必须取唯一值的约束条件。

```
ALTER  TABLE  S  ADD  UNIQUE(Sname);
```

注意：在实际的 DBMS 中往往有以下限制。

(1) 不能改变列名。

(2) 不能将含有空值的列修改为 NOT NULL 约束。

(3) 若列中已有数据，则不允许减小该列的宽度，也不能修改其数据类型。

(4) 除 NULL/NOT NULL 约束之外，其类型的约束在修改之前必须先将其删除。

3. 删除基本表

```
DROP TABLE <表名>[RESTRICT| CASCADE];
```

RESTRICT：表示限制。要删除的基本表不能被其他表的约束所引用，如果存在依赖该表的对象，则此表不能被删除。

CASCADE：表示删除该表没有限制。在删除基本表的同时，依赖该表的所有对象一起删除。

【例 3-9】 删除学生表 S。

```
DROP TABLE  S  CASCADE;
```

基本表定义被删除，表中数据被删除，表上建立的索引、视图、触发器等对象也将被删除。如果在学生表 S 中创建了软件工程系学生的视图 SE_Student，若选择 CASCADE，则 S 表被删除的同时视图 SE_Student 也被自动删除。

微课视频

3.2.3 索引的创建与删除

查询操作是数据库中最常用的操作，为了提高查询速度，SQL 标准和现在的 DBMS 中都使用索引技术。

1. 索引的创建

创建索引的语句格式为：

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

说明：

(1) 可选项[UNIQUE]表示建立唯一索引。该索引的每一个索引值只能对应唯一的元组，因此唯一索引常用于 PRIMARY KEY 的列上，以区别每一行。

(2) 可选项[CLUSTER]表示建立聚簇索引。按照索引值排列元组，并将排好顺序的元组存储在表中。因此，一个表最多只能建立一个聚簇索引。当表中有一个被设置为 UNIQUE 的列时，系统会自动建立一个非聚簇的唯一索引。非聚簇索引的排列结果不会存储在表中，而是另外存储，所以一个表可以有多个非聚簇索引。而聚簇索引比非聚簇索引查询速度快。DBMS 一般会自动地在 PRIMARY KEY 的列上建立聚簇索引。

(3) 必选项<索引名>是为创建的索引命名一个合法的标识符。

(4) 必选项<表名>是指定要建立索引的基本表名字。

(5) <列名>表示要建立索引的列。索引可以建立在一列或多列上，各列之间用逗号分隔。每个<列名>后面还可以用<次序>指定索引值的排列次序。次序有升序和降序两种，ASC 表示升序，DESC 表示降序，默认值为 ASC。

(6) 只有 DBA 或表的主人(即建立表的人)才能建立索引。索引并不是建的越多越好，系统维护索引顺序要花费代价，一般原则是在经常查询的列上建立索引，经常更新的列不宜建立聚簇索引。

(7) 在查询时 DBMS 自动选择是否使用索引以及使用哪些索引。

(8) RDBMS 中索引一般采用 B+树、HASH 索引来实现，B+树索引具有动态平衡的优点，而 HASH 索引具有查找速度快的特点，是采用 B+树，还是采用 HASH 索引，这由具体的 RDBMS 来决定。索引是关系数据库的内部实现技术，属于内模式的范畴。

【例 3-10】 在 *S* 表的 Sname(姓名)列上建立一个聚簇索引。

```
CREATE CLUSTER INDEX Stusname  ON  S(Sname);
```

【例 3-11】 为学生-课程-成绩数据库中的 *S*，*C*，SC 三个表建立索引。

```
CREATE UNIQUE INDEX  Stusno ON S (Sno);
CREATE UNIQUE INDEX  Coucno ON C (Cno);
CREATE UNIQUE INDEX  SCno ON SC(Sno ASC,Cno DESC);
```

表示在 *S* 表上按学号升序建立唯一索引，在 *C* 表上按课程号升序建立唯一索引，在 SC 表上按学号升序和课程号降序建立唯一索引。

2. 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，DBMS 要花费许多时间来维护索引。这就需要删除不必要的索引。

删除索引的语句格式为：

```
DROP  INDEX <表名.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定义。

【例 3-12】 删除 S 表的 Stusname 索引。

```
DROP INDEX Stusname;
```

3.3 数据查询

数据查询是数据库中最常见的操作，通过 SQL 提供的 SELECT 语句进行操作可得到所需的信息。


微课视频

3.3.1 SELECT 语句格式

SQL 的查询语句一般格式为：

```
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 ] ];
```

说明：

（1）可选项[ALL|DISTINCT]：ALL 表示输出全部查询结果，DISTINCT 表示取掉查询结果中的重复行，默认为 ALL。

（2）目标列表达式：表示要查询的数据，可以是列名、含有列名的函数或表达式，还可以是常量。

（3）<表名或视图名>：表示数据来源，可以是一个或两个以上的基本表或视图。

（4）可选项 WHERE <条件表达式>子句：表示选取满足条件的元组。

（5）可选项 GROUP BY <列名 1 > [HAVING <条件表达式>] 子句：表示将查询结果按照<列名 1 >分组，值相同的元组被分为一组。HAVING <条件表达式>是对组的条件限制，只输出满足条件的组，它不能单独使用，只能配合 GROUP BY 子句。

（6）可选项 ORDER BY <列名 2 >子句：表示将查询的结果按照<列名 2 >的值排序输出，ASC 表示升序，DESC 表示降序，默认是升序。也可以按多个列排序，之间用逗号分隔。

（7）SELECT 语句执行的结果仍是一个表。SELECT 子句相当于关系代数中的投影，WHERE 子句相当于选择操作，当查询涉及多个表时，WHERE 子句要同时给出连接条件。SELECT 语句的执行顺序是 FROM→WHERE→GROUP BY→SELECT→ORDER。

下面以学生-课程-成绩数据库为例说明 SELECT 语句的各种用法。


微课视频

3.3.2 单表查询

单表查询就是数据源只涉及一张表。

1. 选择表中若干列

1）查询指定列

在实际应用中，用户可能只需要表中部分属性列，这时使用 SELECT 子句的<目标列表达式>指定。

【例 3-13】 查询全体学生的学号、姓名和出生日期。

```
SELECT Sno,Sname,Sbirth
FROM S;
```

查询结果如下。

Sno	Sname	Sbirth
20418001	于文轩	2001-11-18
20418002	吴广斌	2002-02-08
20418003	王美	2001-09-17
20418004	刘方超	2002-05-09
20418005	孙涛	2002-11-29
20418006	沈蒙	2002-01-22
20418007	田玲	2001-12-17

【例 3-14】 查询全体学生的姓名、学号和所在系。

```
SELECT Sname, Sno,Sdept
FROM S
```

查询结果如下。

Sname	Sno	Sdept
于文轩	20418001	计算机科学系
吴广斌	20418002	计算机科学系
王美	20418003	软件工程系
刘方超	20418004	软件工程系
孙涛	20418005	网络工程系
沈蒙	20418006	网络工程系
田玲	20418007	信息科学系

用户可以根据应用的需要改变<目标列表达式>中各个列的先后顺序。

2）查询全部列

查询全部列有两种方法：一种是在<目标列表达式>中全部列出；另一种是若列的显示顺序与其在表中的顺序相同，则可以简单地将<目标列表达式>指定为 * 。

【例 3-15】 查询全体学生的详细信息。

```
SELECT Sno,Sname, Ssex,Sbirth,Sdept
FROM S
```

等价于语句：

```
SELECT * FROM  S;
```

3）查询经过计算的列

【例 3-16】 查询全体学生的姓名以及年龄。

```
SELECT Sname, Year(CURDATE()) - Year(Sbirth)
FROM S
```

这里通过函数获取当前日期的年份减去出生日期的年份计算得到学生年龄，以MySQL 为例，使用函数 YEAR()取日期型数据的年份，CURDATE()取当前日期进行计算，查询结果如下。

Sname	Year(CURDATE())-Year(Sbirth)
于文轩	20
吴广斌	19
王美	20
刘方超	21
孙涛	21
沈蒙	21
田玲	20

SELECT 子句的<目标列表达式>可以是算术表达式、字符串常量、函数等，如果<目标列表达式>是字符串常量，则原样输出。用户还可以指定别名来改变查询结果的列标题，这对<目标列表达式>非常有用。

指定列别名的语法格式为：

```
<目标列表达式> [as] 别名
```

为例 3-16 的查询结果指定列名：

```
SELECT Sname as 姓名,Year(CURDATE()) - Year(Sbirth ) as 年龄
FROM S
```

查询结果如下。

姓名	年龄
于文轩	20
吴广斌	19
王美	20
刘方超	21
孙涛	21
沈蒙	21
田玲	20

4）消除取值重复的行

在对表进行列的选择后，查询结果中可能会出现取值完全相同的行，可以用DISTINCT消除它们。

【例 3-17】 查询选修了课程的学生学号。

```
SELECT Sno
FROM SC;
```

查询结果如下。

Sno
20418001
20418001
20418001
20418001
20418001
20418001
20418001
20418001
20418002
20418002
20418002
20418003
20418003
20418003
20418004
20418005

上述查询结果中出现许多重复行，要取消掉这些重复行，必须在SELECT后面加上DISTINCT。

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

执行结果为：

Sno
20418001
20418002
20418003
20418004
20418005

2. 选择表中若干元组

在查询中用户除了选择所需列之外，还可以对行数据进行选择，通过WHERE子句实

现。WHERE 子句常用的查询条件如表 3.3 所示。

表 3.3 WHERE 子句常用的查询条件

查询条件	谓 词
比较	=,>,<,>=,<=,!=,<>,!>,!<；NOT+上述比较运算符
确定范围	BETWEEN AND,NOT BETWEEN AND
确定集合	IN,NOT IN
字符匹配	LIKE,NOT LIKE
空值	IS NULL,IS NOT NULL
多重条件(逻辑运算)	AND,OR,NOT

1）比较大小

【例 3-18】 查询软件工程系的学生学号及姓名。

```
SELECT Sno,Sname
FROM S
WHERE Sdept = '软件工程系';
```

【例 3-19】 查询 2002 年 1 月 1 日以后出生的学生姓名及出生日期。

```
SELECT Sname 姓名,Sbirth 出生日期
FROM S
WHERE Sbirth>'2002 - 01 - 01';
```

【例 3-20】 查询考试成绩不及格的学生学号。

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

2）确定范围

确定范围使用谓词 BETWEEN AND 和 NOT BETWEEN AND,查找属性值在(或不在)指定范围内的元组。其中,BETWEEN 后面是下限(低值),AND 后面是上限(高值)。其语法格式为：

```
列名|<表达式> [NOT] BETWEEN <下限> AND <上限>
```

【例 3-21】 查询考试成绩为 80～90 的学生学号和课程号。

```
SELECT Sno,Cno
FROM SC
WHERE Grade BETWEEN 80 AND 90;
```

等价于：

```
SELECT Sno,Cno
FROM SC
WHERE Grade >= 80 AND Grade <= 90;
```

【例 3-22】 查询考试成绩不为 80～90 的学生学号和课程号。

```
SELECT Sno,Cno
FROM SC
WHERE Grade NOT BETWEEN 80 AND 90;
```

等价于：

```
SELECT Sno,Cno
FROM SC
WHERE Grade < 80 OR Grade > 90;
```

3）确定集合

确定某个属性的值是否在指定的集合内使用 IN 谓词。其语法格式为：

```
列名 [NOT] IN(常量 1, 常量 2, …, 常量 n)
```

【例 3-23】 查询计算机科学系、软件工程系和网络工程系的学生学号和姓名。

```
SELECT Sno,Sname
FROM S
WHERE Sdept IN('计算机科学系','软件工程系','网络工程系');
```

等价于：

```
SELECT Sno,Sname
FROM S
WHERE Sdept = '计算机科学系' OR Sdept = '软件工程系' OR Sdept = '网络工程系';
```

【例 3-24】 查询不是计算机科学系、软件工程系和网络工程系的学生学号和姓名。

```
SELECT Sno,Sname
FROM S
WHERE Sdept NOT IN('计算机科学系','软件工程系','网络工程系');
```

等价于：

```
SELECT Sno,Sname
FROM S
WHERE Sdept!= '计算机科学系' AND Sdept!= '软件工程系' AND Sdept!= '网络工程系';
```

4）字符匹配

在实际查询中用户可能不能给出精确字符条件，这时就使用 LIKE 谓词和通配符来实现模糊查询。其语法格式如下。

```
列名 [NOT]LIKE '<匹配串>'  [ESCAPE<'换码字符'>]
```

其含义是查找指定的属性列值与<匹配串>相匹配的元组。匹配串中可以包含字符常量，也可包含下列通配符。

%：匹配 0 个或多个任意字符。

_：匹配任意一个字符。

【例 3-25】 查询姓“刘”的学生信息。

```
SELECT *
FROM S
WHERE Sname LIKE '刘%';
```

【例 3-26】 查询不姓“刘”的学生信息。

```
SELECT *
FROM S
WHERE Sname NOT LIKE '刘%';
```

【例 3-27】 查询姓名中最后一个字是“玲”的学生信息。

```
SELECT *
FROM S
WHERE Sname LIKE '%玲';
```

【例 3-28】 查询姓名中第二个字是“文”的学生信息。

```
SELECT *
FROM S
WHERE Sname LIKE  '_文%';
```

注意：数据库字符集为 GBK 时一个汉字只需要一个“_”；当字符集为 ASCII 时一个汉字需要两个“_”。

5）涉及空值的查询

空值(NULL)在数据库中表示不确定或不知道的值。某个属性的值是否为 NULL，不能使用比较运算符=或!=，只能使用 IS NULL 或 IS NOT NULL 谓词，其语法格式如下。

```
列名 IS [NOT] NULL;
```

【例 3-29】 查询没有考试成绩的学生学号和相应的课程号。

```
SELECT Sno,Cno
FROM SC
WHERE Grade IS NULL;
```

【例 3-30】 查询有考试成绩的学生学号和相应的课程号。

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

6）多重条件查询

在 WHERE 子句中可以用 AND 和 OR 逻辑运算符将多个条件连接起来查询。

【例 3-31】 查询软件工程系 2002 年 1 月 1 日以后出生的学生姓名及出生日期。

```
SELECT Sname,Sbirth
FROM S
WHERE Sdept = '软件工程系' AND Sbirth > '2002 - 01 - 01';
```

【例 3-32】 查询软件工程系和计算机科学系 2002 年 1 月 1 日以后出生的学生姓名及出生日期。

```
SELECT Sname,Sbirth
FROM S
WHERE (Sdept = '软件工程系' OR Sdept = '计算机科学系') AND Sbirth > '2002 - 01 - 01';
等价于:
SELECT Sname,Sbirth
FROM S
WHERE Sdept IN('软件工程系','计算机科学系') AND Sbirth > '2002 - 01 - 01';
```

3. 对查询结果排序

用户可以使用 ORDER BY 子句将查询的结果按照一列或多列排序输出。语句格式如下。

```
ORDER BY <列名 1 >[ASC|DESC] [,<列名 2 >[ASC|DESC]]
```

ASC 表示升序，DESC 表示降序，默认是升序，多列排序时之间用逗号分隔。

【例 3-33】 查询选修了课程号为“002”的学生学号及成绩，查询结果按成绩的降序排列。

```
SELECT Sno,Grade
FROM SC
WHERE Cno = '002'
ORDER BY Grade DESC;
```

【例 3-34】 查询全体学生信息，查询结果按系名的升序排列，同一系的学生按出生日期的降序排列。

```
SELECT *
FROM S
ORDER BY Sdept,Sbirth DESC;
```

查询结果如下。

Sno	Sname	Ssex	Sbirth	Sdept
20418002	吴广斌	男	2002-02-08	计算机科学系
20418001	于文轩	男	2001-11-18	计算机科学系
20418004	刘方超	男	2002-05-09	软件工程系

续表

Sno	Sname	Ssex	Sbirth	Sdept
20418003	王美	女	2001-09-17	软件工程系
20418005	孙涛	男	2002-11-29	网络工程系
20418006	沈蒙	男	2002-01-22	网络工程系
20418007	田玲	女	2001-12-17	信息科学系

4. 使用聚集函数统计汇总查询

SQL 提供了许多库函数，用于统计和汇总查询，增加了基本检索能力。这些函数是对给定值进行计算并返回一个单值，因此称之为聚集函数或集合函数。常用的聚集函数如表 3.4 所示。

表 3.4 常用的聚集函数及其功能

函数名称	功　能
COUNT	统计元组个数
COUNT(DISTINT|ALL <列名>)	统计一列中值的个数
SUM(DISTINT|ALL <列名>)	计算一列值的总和，必须是数值型
AVG(DISTINT|ALL <列名>)	计算一列值的平均值，必须是数值型
MAX(DISTINT|ALL <列名>)	求一列中的最大值
MIN(DISTINT|ALL <列名>)	求一列中的最小值

说明：除 COUNT(*)外，其他函数在计算过程中均忽略 NULL 值；聚集函数不能用于 WHERE 子句，只用于对整个表进行计算和 GROUP BY 的 HAVING 子句中。

【例 3-35】 统计学生总人数。

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

【例 3-36】 统计选课学生的人数。

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

由于一个学生可选多门课程，所以加 DISTINCT 取掉相同值，避免重复计算学生人数。

【例 3-37】 求学号为 20418001 学生的总分、平均分、最高分和最低分。

```
SELECT SUM(Grade),AVG(Grade),MAX(Grade),MIN(Grade)
FROM SC
WHERE Sno = '20418001';
```

5. 分组查询

GROUP BY 子句可以将查询结果按某一列或多列的值分组，值相同的元组为一组，此时聚集函数作用于每一组，即每一组都有一个函数值，否则聚集函数作用于整个查询结果。

GROUP BY 子句的语法格式为：

```
GROUP BY <分组列名 1>  [,<分组列名 2>…]  [HAVING <组的筛选条件>]
```

HAVING 子句不能单独使用，只能配合 GROUP BY 子句，但 GROUP BY 子句可以没有 HAVING 子句；HAVING 子句是对组的筛选条件，而 WHERE 子句是对表或视图中元组的筛选条件。当在一个 SQL 查询中同时使用 WHERE 子句和 GROUP BY 子句时，其顺序是 WHERE，GROUP BY，HAVING，所以 GROUP BY 子句一般跟在 WHERE 子句后面。

【例 3-38】 查询每门课的选课人数。

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

查询结果如下。

Cno	人数
001	2
002	3
003	3
004	2
005	2
006	1
007	1
008	1

【例 3-39】 查询选修两门及以上课程的学生学号和选课门数。

```
SELECT Sno 学号,COUNT( * ) 选课门数
FROM SC
GROUP BY Sno HAVING (COUNT( * )>= 2);
```

查询结果如下。

学号	选课门数
20418001	8
20418002	3
20418003	3

该语句的处理过程为：先执行 GROUP BY 子句对 SC 表数据按 Sno 进行分组，然后用 COUNT 分别对每一组进行统计，最后按条件筛选出统计结果大于或等于 2 的组并输出。

3.3.3　多表连接查询


微课视频

前面介绍了单表查询，在实际中用户往往需要从多个表中获取所需信息。如果一个查询同时涉及两个以上的表，则称为连接查询。连接查询实际上是通过比较各个表之间共同

属性列的值是否满足条件而进行的。连接方法有以下两种。

（1）表之间满足一定条件的行进行连接时，在 FROM 子句中指明进行连接的表名，在 WHERE 子句中指明连接的列名及连接条件。

（2）利用关键字 JOIN 进行连接。又分为以下几种。

INNER JOIN(内连接)：显示符合条件的元组，系统默认内连接。

LEFT(OUTER)JOIN：左外连接，用于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，相应的右边数据行以 NULL 来显示。

RIGHT(OUTER)JOIN：右外连接，用于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，相应的左边数据行以 NULL 来显示。

FULL(OUTER)JOIN：完全连接，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。此时相应的数据行都以 NULL 来显示。

CROSSJOIN 为交叉连接，将一个表的每一个元组和另一表的每个元组匹配成新的元组。

连接查询包括等值连接查询、自然连接查询、自身连接查询、外连接查询和复合条件连接查询等。

1. 等值连接与非等值连接

在连接查询中用来连接两个表的条件一般格式为：

```
[<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＝，＞，＜，＞＝，＜＝，!＝等。

当为“＝”时，称为等值连接。使用其他运算符时为非等值连接。

连接条件中的列名称为连接字段，其必须有可比性，名字不一定相同。当两个表的列名相同时，必须用表名前缀来指明所属的表。如果列名是唯一的，则不必加前缀。

【例 3-40】 查询所有选课学生及其选课信息。

```
第一种写法：
SELECT S. * , SC. *
FROM S, SC
WHERE S. Sno = SC. Sno;
第二种写法：
SELECT S. * , SC. *
FROM S JOIN SC ON S. Sno = SC. Sno;
```

执行结果一致，如下。

Sno	Sname	Ssex	Sbirth	Sdept	Sno	Cno	Grade
20418001	于文轩	男	2001-11-18	计算机科学系	20418001	001	80
20418001	于文轩	男	2001-11-18	计算机科学系	20418001	002	91
20418001	于文轩	男	2001-11-18	计算机科学系	20418001	003	88
20418001	于文轩	男	2001-11-18	计算机科学系	20418001	004	85
20418001	于文轩	男	2001-11-18	计算机科学系	20418001	005	77
20418001	于文轩	男	2001-11-18	计算机科学系	20418001	006	73
20418001	于文轩	男	2001-11-18	计算机科学系	20418001	007	68

续表

Sno	Sname	Ssex	Sbirth	Sdept	Sno	Cno	Grade
20418001	于文轩	男	2001-11-18	计算机科学系	20418001	008	80
20418002	吴广斌	男	2002-02-08	计算机科学系	20418002	001	79
20418002	吴广斌	男	2002-02-08	计算机科学系	20418002	002	73
20418002	吴广斌	男	2002-02-08	计算机科学系	20418002	003	84
20418003	王美	女	2001-09-17	软件工程系	20418003	002	92
20418003	王美	女	2001-09-17	软件工程系	20418003	003	81
20418003	王美	女	2001-09-17	软件工程系	20418003	005	82
20418004	刘方超	男	2002-05-09	软件工程系	20418004	004	75
20418005	孙涛	男	2002-11-29	网络工程系	20418005	007	85

DBMS 执行连接操作的基本过程是：先取 *S* 表中的第 1 个元组，在 SC 表中从头开始扫描，逐一检查每个元组是否满足连接条件，若是，则连接将其放到结果表，直到 SC 表全部查找完毕，再取 *S* 表中第 2 个元组。重复以上过程，直到 *S* 表中的元组全部处理完毕。由此可见，连接操作比较费时，如果在 SC 表的 Sno 列上建立排序或索引，这样只需对 SC 表扫描一遍，能够提高查询效率，这就是为什么要在经常查询的列或出现在连接条件的列上建立索引的原因。

例 3-40 属于等值连接，在查询结果中没有取消掉重复的列 Sno，要想取消掉重复列，则在 SELECT 后的<目标列表达式>中指明。如果连接表中属性列名唯一，可以直接写出列名，否则要在列名前加上所属的表名。

【例 3-41】 取消掉例 3-40 中的重复列。

```
SELECT S.Sno,Sname,Sbirth,Sdept,Cno,Grade
FROM S,SC
WHERE S.Sno = SC.Sno;
```

【例 3-42】 查询所有学生的学号、姓名、选修的课程名及成绩。

```
第一种写法:
SELECT S.Sno,Sname,SC.Cno,C.Cname,SC.Grade
FROM S,SC,C
WHERE S.Sno = SC.Sno AND C.Cno = SC.Cno;
第二种写法:
SELECT S.Sno,Sname,SC.Cno,C.Cname,SC.Grade
FROM S
JOIN SC ON S.Sno = SC.Sno
JOIN C ON C.Cno = SC.Cno;
```

连接查询不仅可以是两个表，也可以是两个以上的表进行连接。在例 3-42 中首先是 *S* 表和 SC 表进行连接，其结果再和 *C* 表连接。

2. 自身连接

连接查询不仅是不同的表进行连接，还可以是一个表与其自己进行连接，这就是自身连

接。为了实现自身连接，要给同一个表起两个别名，以此区分。

【例 3-43】 查询每门课程的间接先修课。

```
SELECT CX.Cno 课程号 ,CY.Cpno 间接先修课号
FROM C Cx,C CY
WHERE CX.Cpno = CY.Cno
```

查询结果如下。

课程号	间接先修课号
001	
004	002
006	007
007	
008	002
009	007

【例 3-44】 查询与“数据库原理及应用”课程学分相同的课程名。

```
SELECT Cy.Cno 课程号 ,CY.Cname 课程名
FROM C Cx,C CY
WHERE CX.Credits = CY.Credits AND CX.Cname = '数据库原理及应用';
```

查询结果如下。

课程号	课程名
004	数据库原理及应用
005	高等数学

3. 外连接

在以上的连接查询中，不满足连接条件的元组不会在结果关系中出现。但有时用户希望这些元组出现在结果中，例如，例 3-40 的查询结果只包括选课的学生信息，而不会有两个没有选课的学生，但是还想让这些元组仍然出现在结果关系中，而在选课 SC 表的属性上填空值 NULL，这时就需要使用外连接。

外连接中最常用的是左外连接和右外连接两种。外连接是利用关键字 JOIN 进行连接的。

【例 3-45】 查询所有学生的学号、姓名、选修的课程号及成绩。

```
SELECT S.Sno,Sname,Cno,Grade
FROM S
LEFT OUTER JOIN SC
ON S.Sno = SC.Sno;
```

查询结果如下。

Sno	Sname	Cno	Grade
20418001	于文轩	001	80
20418001	于文轩	002	91
20418001	于文轩	003	88
20418001	于文轩	004	85
20418001	于文轩	005	77
20418001	于文轩	006	73
20418001	于文轩	007	68
20418001	于文轩	008	80
20418002	吴广斌	001	79
20418002	吴广斌	002	73
20418002	吴广斌	003	84
20418003	王美	002	92
20418003	王美	003	81
20418003	王美	005	82
20418004	刘方超	004	75
20418005	孙涛	007	85
20418006	沈蒙	NULL	NULL
20418007	田玲	NULL	NULL

在查询结果中包括所有学生，没有选课的两个学生沈蒙和田玲的选课信息用空值填充。

3.3.4　嵌套查询

SQL 允许多层嵌套。将一个 SELECT-FROM-WHERE 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。在处理嵌套查询时，先求出内层查询（又称为子查询）的结果，再求外层查询（又称为父查询）的结果。当子查询中查询条件依赖于外层查询中的某个值，而且子查询要反复求值用于外层查询使用时，这种查询称为相关子查询，否则，称为不相关子查询。子查询的 SELECT 不能使用 ORDER BY 子句，因为 ORDER BY 子句只能对最终查询结果进行排序。

当查询涉及多个表时使用嵌套查询逐次求解层次分明，反映了 SQL 具有结构化程序设计的特点，并且嵌套查询的执行效率比连接查询的笛卡儿积效率高。

在嵌套查询中，当用户能够确切地知道内层查询返回单值时，可以使用比较运算符（=，>，<，>=，<=，!=或<>等）。如果子查询的返回值不止一个，而是一个集合，一般使用 IN 谓词，也可以在比较运算符和子查询之间插入 ANY 或 SOME（它们都表示任意一个，=ANY 等价于 IN）以及 ALL（表示全部）。SQL 允许<SOME，<=SOME，>=SOME，=SOME，和<>SOME 的比较，也允许<ALL，<=ALL，>= ALL，=ALL，和<>ALL 的比较。下面举例说明其用法。

1. 子查询返回一个值

【例 3-46】 查询与“刘方超”在同一系学习的学生学号、姓名和系名。

```
SELECT Sno,Sname,Sdept
FROM S
WHERE Sdept =
   (SELECT Sdept
   FROM S
   WHERE Sname = '刘方超');
```

查询结果如下。

Sno	Sname	Sdept
20418003	王美	软件工程系
20418004	刘方超	软件工程系

该语句的执行过程是先处理子查询，检索出“刘方超”所在系“软件工程系”作为外层查询的条件，然后在外层查询中找出在“软件工程系”学习的学生信息。因为一个学生只能在一个系学习，子查询只能返回一个值，所以外层查询条件中可以用＝。

例 3-46 还可以使用自身连接完成。

```
SELECT  SX.Sno,SX.Sname,SX.Sdept
FROM S SX,S SY
WHERE  SX.Sdept = SY.Sdept  AND  Sy.Sname = '刘方超';
```

2. 子查询返回一组值

【例 3-47】 查询选修了课程号为“004”的学生姓名。

```
SELECT Sname
FROM S
WHERE Sno IN
(SELECT Sno
   FROM SC
   WHERE Cno = '004');
```

该语句的执行过程是先处理子查询，检索出选修了课程号为“004”的学生号作为外层查询的条件，然后再找出学号在此集合中的学生姓名。因为选修“004”课程的学生不止一个，子查询可能返回一组值，所以，在外层查询条件中可以用 IN 或＝ANY。

该查询还可以用连接查询实现：

```
SELECT Sname
FROM S,SC
WHERE  S.Sno = SC.Sno  AND  Cno = '004';
```

【例 3-48】 查询其他系中比计算机科学系某一学生出生日期大的学生姓名及年龄。

```
SELECT Sname,Sbirth
FROM S
```

```
WHERE Sbirth > ANY
   (SELECT Sbirth
   FROM S
   WHERE Sdept = '计算机科学系')
   AND Sdept!= '计算机科学系';
```

先执行子查询，找到计算机科学系中所有学生的出生日期集合('2001-11-18'，'2002-02-08')；再执行父查询，查询所有不是计算机科学系且出生日期大于'2001-11-18'或'2002-02-08'的学生姓名和出生日期，在此使用> ANY 谓词。

查询结果如下。

Sname	Sbirth
刘方超	2002-05-09
孙涛	2002-11-29
沈蒙	2002-01-22
田玲	2001-12-17

此查询也可以写成：

```
SELECT Sname,Sbirth
FROM S
WHERE Sbirth >
   (SELECT MIN(Sbirth)
   FROM S
   WHERE Sdept = '计算机科学系')
   AND Sdept!= '计算机科学系';
```

先执行子查询，利用聚集函数 MIN 找到计算机科学系中所有学生的最小出生日期'2001-11-18'；再执行父查询，查询所有不是计算机科学系且出生日期大于'2001-11-18'的学生姓名和出生日期。

【例 3-49】 查询其他系中比计算机科学系所有学生出生日期都大的学生姓名及年龄。

```
SELECT Sname,Sbirth
FROM S
WHERE Sbirth > ALL
   (SELECT Sbirth
   FROM S
   WHERE Sdept = '计算机科学系')
   AND Sdept!= '计算机科学系'
```

查询结果如下。

Sname	Sbirth
刘方超	2002-05-09
孙涛	2002-11-29

此查询也可以写成：

```
SELECT Sname,Sbirth
FROM S
WHERE Sbirth>
   (SELECT MAX(Sbirth)
   FROM S
   WHERE Sdept = '计算机科学系')
   AND Sdept!= '计算机科学系';
```

先执行子查询，找到计算机科学系中所有学生的出生日期集合('2001-11-18', '2002-02-08')；再执行父查询，查询所有不是计算机科学系且出生日期大于'2001-11-18'和'2002-02-08'的学生姓名和出生日期，在此使用> ALL 谓词。也可以利用聚集函数 MAX 找到计算机科学系中所有学生的最大出生日期 '2002-02-08'；再执行父查询，查询所有不是计算机科学系且出生日期大于'2002-02-08'的学生姓名和出生日期。

【例 3-50】 查询没有选修课程的学生姓名。

```
SELECT Sname
FROM S
WHERE Sno!= ALL
    (SELECT Sno
     FROM SC);
```

!=ALL 的含义是不等于子查询结果中的任何一个值，也可使用 NOT IN 代替!=ALL。以上例题中子查询的查询条件不依赖于外层查询，它们都属于不相关子查询。

3. 使用 EXISTS 谓词

SQL 中用 EXISTS 表示存在量词∃。

(1) EXISTS 谓词表示：若内层查询结果非空，则外层的 WHERE 子句返回真值；若内层查询结果为空，则外层的 WHERE 子句返回假值。

(2) NOT EXISTS 表示：若内层查询结果非空，则外层的 WHERE 子句返回假值；若内层查询结果为空，则外层的 WHERE 子句返回真值

由于带有 EXISTS 谓词或 NOT 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”，所以由 EXISTS 引出的子查询，其<目标列表达式>通常都用 * 。在嵌套查询中，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。

【例 3-51】 查询选修了课程号为“003”的学生姓名。

```
SELECT Sname
FROM S
WHERE EXISTS
      (SELECT *
```

```
        FROM SC
         WHERE SC.Sno = S.Sno AND CNO = '003');
```

带 EXISTS 或 NOT EXISTS 谓词的子查询是相关子查询，例 3-51 的执行过程是：首先取外层查询 S 表中的第一个元组的学号，作为内层查询 SC 表的查询条件，若内层查询结果非空，则外层 WHERE 条件为真，取此元组放入结果表中；然后再检查外层查询 S 表中的第二个元组，重复这一过程，直到 S 表全部检查完毕为止。所以相关子查询的处理不止一次，要反复求值，供外层查询使用。

【例 3-52】 查询没有选修课程号为“003”的学生姓名。

```
SELECT Sname
FROM S
WHERE NOT EXISTS
      (SELECT *
       FROM SC
        WHERE SC.Sno = S.Sno AND CNO = '003');
```

【例 3-53】 查询选修了全部课程的学生姓名。

在 S 表中查询学生姓名，要求这个学生选修了全部课程。尽管 SQL 不支持全称量词，但可以通过存在量词实现。本题可理解为：在 S 表中找出这样的学生姓名，不存在对 C 表中每一门课程不选修的情况。

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

【例 3-54】 查询至少选修了 20418002 学生选修的全部课程的学生学号。

本题可理解为：在 SC 表中找出这样的学生学号，不存在 20418002 学生选修的课程他不选的情况。在这里给 SC 表起三个别名即元组变量。

```
SELECT DISTINCT Sno
FROM SC X
WHERE NOT EXISTS
     (SELECT *
      FROM SC Y
      WHERE Y.Sno = '20418003' AND NOT EXISTS
          (SELECT *
           FROM SC Z
           WHERE Z.Sno = X.Sno AND Z.Cno = Y.Cno));
```

3.3.5 集合查询

SELECT 语句执行的结果是元组集合，当两个查询语句的结果结构完全相同即列数相同，相应的数据类型相同时，可以对这两个查询结果进行并（UNION）、交（INTERSECT）和差（EXCEPT）操作。

集合操作的一般格式为：

```
(SELECT 查询语句 1)
UNION [ALL] | INTERSECT [ALL] | EXCEPT [ALL]
(SELECT 查询语句 2)
```

用户根据需要选择三种集合操作之一。省略可选项 ALL 时，则表示返回结果中取掉重复元组；若使用了 ALL，则表示返回结果中没有取掉重复元组。

【例 3-55】 查询计算机科学系或者软件工程系的学生信息。

```
SELECT *
FROM S
WHERE Sdept = '计算机科学系'
UNION
SELECT *
FROM S
WHERE Sdept = '软件工程系';
```

等价于：

```
SELECT *
FROM S
WHERE Sdept = '计算机科学系' OR Sdept = '软件工程系';
```

【例 3-56】 查询同时选修了课程号“004”和课程号“009”的学生学号。

```
SELECT Sno
FROM SC
WHERE Cno = '004'
INTERSECT
SELECT Sno
FROM SC
WHERE Cno = '009';
```

等价于：

```
SELECT Sno
FROM SC
WHERE Cno = '004'
AND Sno IN
(SELECT Sno
FROM SC
WHERE Cno = '009');
```

【例 3-57】　查询没有选课的学生学号。

```
SELECT Sno
FROM S
EXCEPT
SELECT Sno
FROM SC;
```

等价于：

```
SELECT Sno
FROM S
WHERE Sno NOT IN
  (SELECT Sno
   FROM SC);
```

3.3.6　存储查询结果到表中

可以使用 SELECT…INTO 语句将查询结果存储到一个新建的数据表或临时表中。

【例 3-58】　查询每位学生的学号及总分，将结果存储到一个新建的数据表 Sno_Sum。

```
SELECT Sno 学号,SUM(Grade)总分
INTO S_Sum
FROM SC
GROUP BY Sno;
```

如果将 INTO S_Sum 改为 INTO ＃S_Sum，则查询的结果被存放到一个临时表中，临时表只存储在内存中，并不存储在当前数据库中，所以其存在的时间非常短。

3.3.7　基于派生表的查询

子查询不仅可以出现在 WHERE 子句中，还可以出现在 FROM 子句中，这时将子查询生成的临时派生表作为数据源。

【例 3-59】　查询每位学生超过他自己选修课程平均分的课程号。

```
SELECT SC.Sno,SC.Cno
FROM SC, (SELECT Sno, AVG(Grade) FROM SC GROUP BY Sno )
AS S_AVG(Sno,Grade_AVG)
WHERE SC.Sno = S_AVG.Sno AND SC.Grade > S_AVG.Grade_AVG;
```

在本例中子查询(SELECT Sno, AVG(Grade) FROM SC GROUP BY Sno)生成一个派生表 S_AVG(Sno,Grade_AVG)，存储每位学生的学号和平均成绩。主查询将 SC 表与 S_AVG 进行连接查询，筛选出成绩大于其平均成绩的课程号。

3.4 数据更新

SQL 的数据更新包括插入数据、删除数据和修改数据等操作。

3.4.1 插入数据


微课视频

插入数据就是把新的元组插入到一个数据表中。插入数据使用 INSERT INTO 语句，可分为以下两种情况。

1. 插入一个元组

将一行新元组插入指定表的语句格式为：

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

其中，<列名>是可选项，省略时表示 VALUES 后的元组值列的顺序与基本表中的列名顺序一致，并且对应列上的数据类型保持一致。若指定列名，则表示 VALUES 后的元组值只提供给这些列名的值。

【例 3-60】 在学生表 S 中插入一个学生信息（学号：20418008，姓名：王小力，性别：男，出生日期：2002-05-01，系别：软件工程系）。

```
INSERT INTO S
VALUES('20418008','王小力','男','2002-05-01','软件工程系');
```

表名 S 后面没有指定列名，表示新插入的元组在每个列上均有值，且 VALUES 子句中值的排列与表中各列顺序一致，对应列上数据类型一致。

注意：VALUES 子句中和各个数据之间必须用逗号分开，字符型数据和日期型数据要用单引号括起来。

【例 3-61】 在选课表 SC 中插入一个选课信息（学号：20418008，课程号：004）。

```
INSERT INTO SC(Sno,Cno)
VALUES('20418008','004');
```

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，对于没有出现的列，将用 NULL 填充。在本例中 Grade 上取 NULL。但在表定义时有 NOT NULL 约束的列上则不能取 NULL。例 3-61 还可以写成：

```
INSERT INTO SC
VALUES('20418008','004',NULL)
```

2. 插入多个元组

将一个子查询的结果插入到指定表中。语句格式为：

```
INSERT
INTO <表名> [(<列名 1>[,<列名 2>…)]
子查询
```

【例 3-62】　查询每位学生的平均成绩总分，将结果存储到一个新建的数据表 AVGgrade。

先建立新表 AVGgrade，用来存放每个学生的平均成绩。

```
CREATE TABLE AVGgrade(Sno CHAR(8),AVGGR SMALLINT)
```

然后利用子查询求出每个学生的平均成绩，把结果存入新表 AVGgrade 中。

```
INSERT INTO AVGgrade
SELECT Sno,AVG(Grade)总分
FROM SC
GROUP BY Sno;
```

3.4.2　删除数据

使用 DELETE 语句删除指定表中的一行或多行元组。语句格式为：

```
DELETE
FROM <表名>
[WHERE <条件>]
```

可选项 WHERE 子句表示要删除满足条件的元组。省略时，则表示删除表中所有元组，但表的定义仍在数据字典中。

1. 删除单个元组

【例 3-63】　删除学生王小力的信息。

```
DELETE FROM S
WHERE Sname = '王小力';
```

2. 删除多个元组

【例 3-64】　删除所有学生的选课信息。

```
DELETE FROM SC;
```

执行此语句后，SC 成为一个空表，但其定义仍存在数据字典中。

3. 利用子查询选择要删除的元组

【例 3-65】　删除“数据库原理及应用”这门课的选课信息。

```
DELETE FROM SC
WHERE Cno IN
   (SELECT Cno
     FROM C
     WHERE Cname = '数据库原理及应用');
```

3.4.3 修改数据

当用户需要修改关系中元组的某些值时，可以使用 UPDATE 实现。其语句格式为：

```
UPDATE <表名>
SET  <列名> = <表达式>[,<列名> = <表达式>]…
[WHERE <条件>]
```

该语句的功能是修改指定表中满足条件的元组中的指定属性值，其中，SET 子句用于指定修改方法，即将<表达式>的值赋给相应的属性列。若省略 WHERE 子句，则表示要修改表中所有元组。

1. 修改单个元组

【例 3-66】 把学生田玲转到网络工程系。

```
UPDATE S
SET Sdept = '网络工程系'
WHERE Sname = '田玲';
```

2. 修改多个元组

【例 3-67】 把选修了课程号为 009 的学生成绩提高 5%。

```
UPDATE SC
SET Grade = Grade * (1 + 0.05)
WHERE Cno = '009';
```

3. 修改子查询选择要删除的元组

【例 3-68】 把网络工程系的学生成绩提高 10%。

```
UPDATE SC
SET Grade = Grade * 1.1
WHERE '网络工程系' =
      (SELECT Sdept
       FROM S
       WHERE S.Sno = SC.Sno);
```

子查询的作用是得到网络工程系的学生，这是一个相关子查询。

3.5　视图

视图是一个虚表，它是从一个或几个基本表（或视图）导出的表，数据库中只存放视图的定义，而不存放视图对应的数据，当基本表中的数据发生变化，从视图中查询出的数据也随之改变。


微课视频

3.5.1　定义和删除视图

1. 定义视图

定义视图语句格式为：

```
CREATE VIEW <视图名> [(<列名>  [,<列名>]…)]
AS <子查询>
[WITH CHECK OPTION]
```

说明：

（1）<列名>为可选项，省略时，组成视图的属性列名由子查询的结果决定。

（2）当视图由多个表连接得到，不同表中可能含有相同列以及视图列名为表达式或聚集函数的计算时，必须指定列名。

（3）子查询不允许含有 ORDER BY 子句和 DISTINCT 短语。

（4）带可选项 WITH CHECK OPTION 表示对视图进行更新操作时系统自动加上视图定义中的谓词条件。

【例 3-69】 定义网络工程系学生视图。

```
CREATE VIEW NE_S
AS
SELECT *
FROM S
WHERE Sdept = '网络工程系'
WITH CHECK OPTION;
```

带了可选项 WITH CHECK OPTION，表示对视图 NE_S 进行更新操作时要满足视图定义中学生所在系名是“网络工程系”的约束。

【例 3-70】 定义全体女生视图。

```
CREATE VIEW S_Gril
AS
SELECT Sno,Sname
FROM S
WHERE Ssex = '女';
```

【例 3-71】 定义软件工程系学生的平均成绩视图。

```
CREATE VIEW SE_AVG(学号,平均成绩)
AS
SELECT SC.Sno,AVG(Grade)
FROM S,SC
WHERE S.Sno = SC.Sno AND Sdept = '软件工程系'
GROUP BY SC.Sno;
```

视图 SE_AVG 是由两个表连接查询而成，并且含有聚集函数的计算列，所以在此指定了视图的列名学号和平均成绩。

2. 删除视图

删除视图语句格式为：

```
DROP VIEW <视图名>
```

将视图定义从数据字典中删除。

【例 3-72】 删除视图 S_Gril。

```
DROP VIEW S_Gril
```

3.5.2 查询视图

视图定义后，用户可以像基本表一样对视图进行查询(SELECT)操作。

【例 3-73】 查询网络工程系 2001 年以后出生的学生学号和姓名。

```
SELECT Sno,Sname
FROM NE_S
WHERE YEAR(Sbirth)> 2001;
```

此查询的执行过程是：首先系统从数据字典中找到 NE_S 定义，然后把该定义与用户的查询要求结合起来，转换成等价的对基本表 S 的查询，这一转换过程称为视图消解(View Resolution)，转换后的查询语句为：

```
SELECT Sno,Sname
FROM S
WHERE Sdept = '网络工程系' AND YEAR(Sbirth)> 2001
```

【例 3-74】 查询平均成绩大于 90 分的软件工程系学生。

```
SELECT *
FROM SE_AVG
WHERE 平均成绩> 90
```

由于视图中的平均成绩列是通过对基本表中的数据计算而来，所以不能直接转换查询，

需要修正。修改后的查询语句为：

```
SELECT SC.Sno 学号,AVG(Grade)  平均成绩
FROM S,SC
WHERE S.Sno = SC.Sno AND Sdept = '软件工程系'
GROUP BY SC.Sno HAVING(Grade)> 90
```

从以上例题可以看出，当对一个基本表进行复杂查询时，可以先定义一个视图，然后只需对该视图进行查询，这样将一个复杂的查询转换成一个简单的查询，从而简化用户的查询操作。

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要转换成对基本表的更新。视图的更新操作有插入、修改和删除操作，其语法格式与基本表的更新操作相同。

【例 3-75】 在学生表 S 中插入一个学生信息（学号：20418010，姓名：李小丽，性别：女，出生日期：2002-09-10，系别：网络工程系）。

```
INSERT INTO NE_S
VALUES('20418010','李小丽','女','2002 - 09 - 10');
```

此语句的执行过程是：首先系统从数据字典中找到 NE_S 定义，然后把该定义与插入操作结合起来，转换成等价的对基本表 S 的插入。转换后插入操作为：

```
INSERT INTO S
VALUES('20418010','李小丽','女','2002 - 09 - 10','网络工程系')
```

【例 3-76】 将网络工程系视图 NE_S 中学生李小丽转到软件工程系。

```
UPDATE NE_S
SET Sdept = '软件工程系'
WHERE Sname = '李小丽'
```

此语句最终要转换成对基本表 S 的修改，转换后修改操作为：

```
UPDATE S
SET Sdept = '软件工程系'
WHERE Sname = '李小丽'  AND Sdept = '网络工程系'
```

并不是所有视图中的数据都可以修改，例如，将软件工程系视图 SE_AVG 中学号为 20418003 的学生平均成绩改为 90，这个操作不能实现。因为 SE_AVG 中的平均成绩列是通过基本表计算得出的，在基本表没有对应的列，所以无法转换。

3.5.4 视图的作用

由于视图中不存放相应的数据，所以对视图的所有操作最终要转换成对基本表的操作，而且对视图的更新操作还要受到限制，这样看起来使操作更加复杂，那为什么还要使用视图？


微课视频

使用视图有以下几个优点。

（1）视图能够简化用户操作。为复杂的查询定义一个视图，用户不必输入复杂的查询语句，只需对该视图进行简单查询即可。

（2）视图能够提供数据保密。对不同用户定义不同的视图，使机密数据不出现在不应看到这些数据的用户视图中。例如，在学生表上定义了软件工程系视图，本系学生只能使用此视图，而无法访问其他系的学生数据。

（3）视图能够保证数据的逻辑独立性。数据的逻辑独立性是指当数据库的逻辑结构发生改变而保证应用程序不受影响。由于应用程序基于视图，当定义视图的基本表结构发生改变时，只需修改视图定义中的子查询部分，从而保证了视图不变，这样使用户的应用程序不受影响。

（4）视图使用户能以多种角度看待同一数据。数据库的特点之一是数据共享，当许多不同类型的用户共享同一数据库时，视图使不同的用户以不同的方式看待同一数据。

小结

SQL 是关系数据库的标准语言，被广泛应用于商用的数据库管理系统。本章首先介绍了 SQL 数据库的三级模式结构，然后介绍了 SQL 的数据定义、数据查询和数据更新功能，并通过实例详细地讲解了 SQL 模式、基本表、索引、视图的创建与删除，数据查询语句的各种用法，数据插入、删除和修改操作。SQL 的数据查询是本章要求掌握的重点内容。现在的数据库管理系统都能支持标准的 SQL，但在使用中还存在一些差异。读者在学习理论知识的同时进行上机练习，在实验的过程中加深理解本章内容。

习题

1. 概念题

（1）名词解释：SQL 模式、基本表、视图、索引、相关子查询、连接查询、嵌套查询。

（2）简述 SQL 的特点。

（3）简述 SQL 支持的关系数据库的三级逻辑结构。

（4）相关子查询和不相关子查询的区别是什么？

（5）简述 SQL 的关系代数特点和元组演算特点。

（6）什么是视图？它有什么作用？

（7）视图与查询表（又称导出表）有什么区别？

（8）在 SQL 中，对于查询结果是否允许存在重复元组是如何实现的？

2. 操作题

(1) 用 SQL 语句建立教学管理数据库的五个基本表,其结构描述如表 3.5～表 3.9 所示。

表 3.5　S 表结构

列　名	数据类型	约束控制	说　明
Sno	Char(6)	主键	学号
Sname	varchar(10)	Not Null	姓名
Ssex	Char(2)	'男'或'女'	性别
Sbirthday	date		出生日期

表 3.6　C 表结构

列　名	数据类型	约束控制	说　明
Cno	Char(5)	主键	课程号
Cname	Char(20)	非空	课程名
Credits	smallint		学分

表 3.7　SC 表结构

列　名	数据类型	约　束	说　明
Sno	Char(6)	外键,引用 S 表的主键值	学号
Cno	Char(5)	外键,引用 C 表的主键值	课程号
Grade	smallint	取值 0～100	成绩

表 3.8　T 表结构

列　名	数据类型	约束控制	说　明
Tno	Char(6)	主键	教师工号
Tname	varchar(10)	Not Null	教师姓名
Tsex	Char(2)	'男'或'女'	性别
Ttitle	Char(10)		职称

表 3.9　TC 表结构

列　名	数据类型	约束控制	说　明
Cno	Char(5)	外键,引用 C 表的主键值	课程号
Tno	Char(6)	外键,引用 T 表的主键值	教师工号

(2) 针对上题中教学管理数据库中的五个基本表输入数据后,用 SQL 语句完成以下操作。

① 查询年龄在 18～22 岁(含 18 和 22)的女学生的学号和姓名。

② 查询教授所授课程的课程号和课程名。

③ 查询至少选修两门课程的学生号。

④ 查询全部学生都选修的课程号和课程名。

⑤ 查询所有缺少成绩的学生。

⑥ 查询没有选课的学生名单。

⑦ 查询张老师所授课程的每门课程的学生平均成绩。

⑧ 统计每位老师的授课门数。

⑨ 统计每门课程的学生选修人数(超过 30 人的课程才统计)。要求输出课程号和选修人数,查询结果按人数的降序排列,人数相同时,按课程号升序排列。

⑩ 求年龄大于所有女同学平均年龄的男同学的姓名和年龄。

⑪ 在基本表 T 中插入一位老师,工号：001198,姓名：张立志,性别：男,职称：副教授。

⑫ 把选修了 Java 课不及格的成绩全部置为空值。

⑬ 删除李小力同学的所有选课信息。

⑭ 修改数据结构课程的学生成绩,若成绩小于 70 分,则提高 10%。

(3) 在上述教学管理数据库上创建一个视图,包含每位同学的学号、姓名、成绩总分、平均分、最高分和最低分。针对该视图完成以下查询。

① 查询田玲同学的成绩总分、平均分,最高分和最低分。

② 查询成绩总分最高的学生姓名及其总分。

(4) 设有两个关系 $R(a,b)$和 $S(a,c)$,用 SQL 查询语句表示下列域表达式。

$\{a|(\exists b(R(ab)\wedge b='18')\}$

$\{abc|\quad R(ab)\wedge S(ac)\quad\}$