第3章 关系数据库标准语言SQL

SQL(Structured Query Language)是关系数据库的标准语言,也称结构化查询语言。它是介于关系代数和元组演算之间的一种语言。SQL是一种综合性的数据库语言,实现对数据的定义、操纵和控制等功能。本章将对SQL的语法规则进行详细的介绍。

3.1 SQL概述

3.1.1 SQL的发展

自从1970年美国IBM研究中心的E. F. Codd提出关系模型,并连续发表多篇论文以后,人们对关系数据库的研究日益深入。1972年,IBM公司开始研制实验型关系数据库管理系统SYSTEM R,并且为其配置了SQUARE(Specifying Queries As Relational Expression)查询语言。1974年,Boyce和Chamberlin在此基础上对其进行改进,将SQUARE语言改为SEQUEL(Structured English Query Language),后来SEQUEL简称为SQL,并首先在IBM公司研制的关系数据库系统System R上实现。

由于它具有功能丰富、使用方便灵活、语言简洁、易学等突出优点,因此深受计算机工业界和计算机用户的欢迎。各厂商纷纷开发基于SQL的商业应用产品,并将SQL作为关系数据库产品事实上的标准,如Oracle、DB2、Sybase等。1986年10月,经美国国家标准局(ANSI)的数据库委员会X3H2批准,将SQL作为关系数据库语言的美国标准,同年公布了标准SQL。1987年6月,国际标准化组织(International Organization for Standardization, ISO)将其采纳为国际标准。这两个标准现在称为"SQL 86"。ANSI在1989年10月颁布了增强完整性特征的SQL 89标准,1992年又公布了SQL 92标准,1999年发布了SQL 99,以后每隔几年会推出一个新版本。

本章的论述主要遵循SQL 92标准,由于各数据库厂商的SQL产品在支持标准SQL 92语法的同时,在功能上都做了相应的扩充,在实现上略有不同,因此,在使用具体的DBMS时,请查阅系统提供的参考手册。

3.1.2 SQL的特点

SQL有许多特点,主要体现在以下4点。

1. 高度非过程化

"过程化"是指用户不但要知道"做什么",还应该知道"怎样做"。对于SQL,用户只需要提出"做什么",无须具体指明"怎么做"。例如,存取路径选择、具体处理操作过程等均由

系统自动完成。这种特点使得用户更能集中精力考虑要“做什么”和所要得到的结果，大大提高了开发效率。

2. 功能完备并且一体化

数据库的主要功能就是通过数据库支持的数据语言来实现的。SQL 不但具有数据定义、数据查询、数据操作、数据控制功能，而且这些功能被集成到一个语言系统中，只要用 SQL 就可以实现数据库生命周期中的全部活动。可见，SQL 功能是完备的。

3. 统一的语法结构

SQL 可用于所有用户的模型，包括系统管理员、数据库管理员、应用程序员及终端用户，这些用户可以通过自含式语言和嵌入式语言两种方式对数据库进行访问，这两种方式使用统一的语法结构。

4. 语言简洁，易学易用

尽管 SQL 的功能很强，但语言十分简洁，SQL 完成核心功能只用了以下 8 个动词。

数据定义：CREATE(创建)、DROP(撤销)。

数据查询：SELECT(查询)。

数据操作：INSERT(插入)、UPDATE(修改)、DELETE(删除)。

数据控制：GRANT(授权)、REVOKE(收权)。

3.1.3 SQL 体系结构

SQL 支持关系数据库体系结构，即外模式、模式和内模式。利用 SQL 可以实现对三级模式的定义、修改和数据的操作功能，在此基础上形成了 SQL 体系结构，如图 3-1 所示。

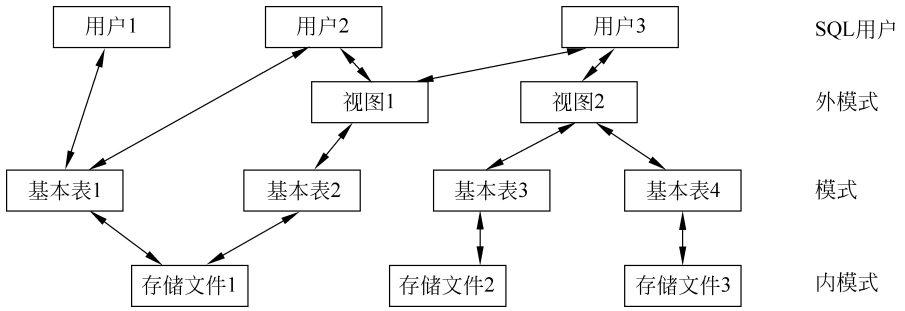

图 3-1 SQL 体系结构

图 3-1 中对应的几个基本概念如下。

(1) SQL 用户。可以是应用程序，也可以是终端用户。SQL 语句可嵌入在宿主语言的程序中使用，也可以作为独立的用户接口，供交互环境下的终端用户使用。

(2) 基本表，简称基表。它是数据库中实际存在的表，在 SQL 中一个关系对应于一个基本表。

(3) 视图。SQL 用视图概念支持非标准的外模式概念。视图是从一个或几个基本表导出的表，虽然它也是关系形式，但它本身不实际存储在数据库中，只存放对视图的定义信息(没有对应的数据)。因此，视图是一个虚表(Virtual Table)或虚关系，而基本表是一种实关系(Practical Relation)。

(4) 存储文件。每个基本表对应一个存储文件,每个存储文件都与外部存储器上一个物理文件对应。一个基本表还可以带一个或几个索引,存储文件和索引一起构成了关系数据库的内模式。

由此可以看出,一个基本表可以存放在多个存储文件中;一个视图可以来自多个基本表,一个基本表可以构造多个视图;一个用户可以查询多个视图,一个视图也可以被多个用户访问。

3.2 SQL 的定义功能

3.2.1 基本表的定义

1. 表结构的定义

建立数据库最重要的一步就是定义基本表的结构。SQL 用于创建基本表的语法结构为:

```
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>]);
```

说明:

(1) 表名是所要定义的基本表的名字,表可以由一个或多个属性(列)组成。

(2) 定义表的各个列时需要指明其数据类型及长度。表 3-1 列出了主要数据类型。

表 3-1 SQL 92 提供的主要数据类型

类　型	数据类型举例及缩写	说　明
Binary	BinaryLargeOBject(BLOB)	这种数据类型以十六进制格式存储二进制字符串的值
BitString	BIT(n) BIT VARYING(n)	这两种数据类型可以存储二进制和十六进制数据,BIT 数据类型长度固定,而 BIT VARYING 数据类型具有可变长度
Boolean	BOOLEAN	这种数据类型存储真、假值——true、false 或 unknown
Character	CHAR(n) VARCHAR(n)	这两种数据类型可以存储适宜的字符集中的任意字符组合。VARCHAR 数据类型允许字符长度变化,而 CHAR 数据类型只能有固定的字符长度。VARCHAR 数据类型自动删除后继的空格,而 CHAR 数据类型则添加空格达到指定长度
Numeric	INTEGER SMALLINT DECIMAL(i,j) FLOAT(p,s) REAL DOUBLE PRECISION	这些数据类型存储数据的准确值(整数或小数)或近似值(浮点数)

续表

类　　型	数据类型举例及缩写	说　　明
Temporal	DATE TIME TIMESTAMP INTERVAL	这些数据类型处理时间的值。DATE 和 TIME 分别处理日期和时间。TIMESTAMP 类型存储按机器当前运行时间计算出来的值。INTERVAL 指定一个时间间隔，它是一个相对值，用于增加或减少一个日期、时间或时间戳类型数据的绝对值

(3) 完整性约束条件。关系完整性约束包括实体完整性、参照完整性和用户定义完整性。这三种完整性约束条件都可以在表的定义中给出。其中，实体完整性定义表的主关键字(Primary Key)参照完整性定义外关键字(Foreign Key)，用户定义完整性根据具体应用对关系模式提出要求，主要包括对数据类型、数据格式、取值范围、空值约束等的定义。

完整性约束又可分为列完整性、元组完整性和表级完整性约束三个级别。在关系模式的定义中，最常定义的是列完整性约束和表级完整性约束。用户定义的完整性规则属于列级完整性约束，而实体完整性和参照完整性都属于表级完整性约束。

由于完整性约束条件也是关系模式定义的一部分，因此下面给出部分完整性约束条件的定义方法。这些完整性约束条件被存入系统的数据字典中，当用户操作表中数据时由 DBMS 自动检查该操作是否违背这些完整性约束条件。

【例 3-1】 建立一个学生表，它由学号、姓名、性别、出生年份、籍贯和所在学院 6 个列组成，其中，学号属性不能为空，并且其值是唯一的。

```
CREATE TABLE 学生
   (学号    CHAR(8)   NOT  NULL   UNIQUE,
    姓名    CHAR(8),
    性别    CHAR(2),
    出生年份    SMALLINT,
    籍贯    CHAR(8),
    学院    CHAR(15));
```

上述 SQL 语句执行后，将建立一个新的空学生表。其中，NOT NULL 和 UNIQUE 分别说明学号不能取空值和重复的值，该约束等同于主码的约束。

2. 主关键字的定义

一个关系可能有多个候选关键字，但在定义基本表时只能定义一个主关键字。一个关系的主关键字由一个或几个属性构成，在 CREATE TABLE 中声明主关键字的方法如下。

(1) 在列出关系模式的属性时，在属性及其类型后加上保留字 PRIMARY KEY，表示该属性是主关键字。

(2) 在列出关系模式的所有属性后，再附加一个声明：

```
PRIMARY  KEY (<属性 1>[,<属性 2>,…])
```

说明：如果关键字由多个属性构成，则必须使用第二种方法。

【例 3-2】 建立一个学生表，它由学号、姓名、性别、出生年份、籍贯和所在学院 6 个列组成，其中，学号为主关键字。

方法一：

```
CREATE    TABLE    学生
    (学号       CHAR(8)      PRIMARY  KEY,
     姓名       CHAR(8),
     性别       CHAR(2),
     出生年份   SMALLINT,
     籍贯       CHAR(8),
     学院       CHAR(15));
```

方法二：

```
CREATE    TABLE    学生
     (学号         CHAR(8),
      姓名         CHAR(8),
      性别         CHAR(2),
      出生年份     SMALLINT,
      籍贯         CHAR(8),
      学院         CHAR(15),
      PRIMARY      KEY(学号));
```

【例 3-3】 建立一个课程表，它由课程号、课程名、学时、开课学期、课程性质 5 个属性组成，其中，课程号为主关键字。

```
CREATE    TABLE    课程
     (课程号         CHAR(8)  NOT  NULL  UNIQUE,
      课程名         CHAR(15),
      学时           SMALLINT,
      开课学期       CHAR(4),
      课程性质       CHAR(15),
      PRIMARY        KEY(课程号));
```

从例 3-3 可以看出，虽然非空(NOT NULL)约束和唯一(UNIQUE)约束结合在一起的作用等同于主键(PRIMARY KEY)约束，但是二者是可以重复定义的。同时，虽然主键的声明是可选的，但为每个关系指定一个主键会更好些。

【例 3-4】 建立一个学习表，它由学号、课程号、成绩 3 个属性组成，其中，学号和课程号为主关键字。

```
CREATE  TABLE  学习
   (学号  CHAR(8),
    课程号 CHAR(8),
    成绩   SMALLINT,
    PRIMARY  KEY(学号, 课程号));
```

该例中，由于组成主关键字的属性有两个：学号和课程号，因此只能在属性列表的最后定义该主关键字。

3. 外部关键字的定义

外部关键字的定义是建立参照完整性的约束，它是关系模式的另一种重要约束。根据参照完整性的概念，在 SQL 中，有两种方法用于说明一个外部关键字。

(1) 如果外部关键字只有一个属性，可以在它的属性名和类型后面直接用 REFERENCES 说明它参照了某个表的某些属性(必须是主关键字)。其语法格式是：

```
REFERENCES  <表名> (<属性>)
```

(2) 在 CREATE TABLE 语句的属性列表后面增加一个或几个外部关键字说明，其格

式为：

```
FOREIGN  KEY (<属性 1 >) REFERENCES <表名> (<属性 2 >)
```

其中，“属性 1”是外部关键字，“属性 2”是被参照的属性。

【例 3-5】 建立一个学习表，它由学号、课程号、成绩 3 个属性组成，其中，学号和课程号的集合为主关键字，同时，学号、课程号也分别是外关键字，分别参照了学生表中的学号和课程表中的课程号。

```
CREATE  TABLE  学习
   (学号  CHAR(8),
    课程号 CHAR(8),
    成绩   SMALLINT,
    PRIMARY KEY(学号, 课程号),
    FOREIGN KEY(学号)  REFERENCES  学生(学号)
    FOREIGN KEY(课程号)  REFERENCES  课程(课程号));
```

该例中定义了两个外关键字：学习表中的学号和课程号。根据参照完整性规则，学习表中的学号要么取空值，要么取学生表中的学号值。但是，由于学习表中的学号又是该关系主关键字中的属性，根据实体完整性约束条件，主属性不能取空值。因此，学习表中的外关键字学号只能取学生表中学号的值，不能取空值。另一个外关键字课程号的取值亦然。

4. 默认值的定义

可以在定义属性时增加保留字 DEFAULT 为表中某列的取值定义一个默认值。

例如：

```
性别   CHAR(2)    DEFAULT  '男';
年龄   SMALLINT   DEFAULT  19;
```

3.2.2 基本表的修改和删除

1. 修改基本表

随着应用环境和应用需求的变化，有时需要修改已经建立好的基本表，如增加列、增加新的完整性约束条件、修改原有的列定义或删除已有的完整性约束条件等，此处仅列出其中的部分语法。

语法格式：

```
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>]
```

其中，表名是要修改的基本表，ADD 子句用于增加新列和新的完整性约束条件，DROP 子句用于删除指定的完整性约束条件，ALTER COLUMN 子句用于修改原有的列定义，包括修改列名和数据类型。

【例 3-6】 在学生表中增加“年龄”属性，类型为 SMALLINT。

```
ALTER  TABLE  学生 ADD  COLUMN 年龄 SMALLINT;
```

【例 3-7】 在学生表中删除“年龄”属性。

```
ALTER  TABLE  学生  DROP COLUMN 年龄;
```

【例 3-8】 课程表中修改“课程名称”属性的长度为 20 位。

```
ALTER  TABLE  课程  ALTER  COLUMN 课程名  CHAR(20);
```

注意:

(1) 可以增加或减少某一列的长度,但是修改后的长度不能小于该列原有数据的长度。

(2) 对于 NULL 值约束进行修改的问题。该问题产生于将某列的约束从 NULL 改变为 NOT NULL 时,要求指定的字段中不能有 NULL 值。如果包含值为 NULL 的字段,则必须先删掉所发现的任何 NULL 值,然后使用 ALTER TABLE 命令进行修改。

(3) MySQL 中,ALTER COLUMN 子句中的 ALTER 用 MODIFY 替代。

2. 基本表的删除

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

其中,RESTRICT 表示如果有视图或约束条件涉及要删除的表时,就禁止 DBMS 执行该命令;而 CASCADE 选项则将该表与其涉及的对象一起删除。

【例 3-9】 假设已经存在一个表,表名为“临时表”,现将其删除,并将与该表有关的其他数据库对象一起删除。

```
DROP  TABLE 临时表 CASCADE;
```

3.2.3 索引的建立与删除

可以用两种方法从数据库中获得数据。第一种方法常被称为顺序访问方式,它需要 SQL 检查每一个记录以找到与之相匹配的数据项。这种查找的方法效率很低,但它是使记录准确定位的唯一方法。

第二种获取数据的方法就是使用索引,数据库中的索引可以帮用户快速地获得想要的数据。索引实际上是根据关系(表)中某些字段的值建立一个树状结构的文件。索引文件中存储的是按照某些字段的值排列的一组记录号,每个记录号指向一个待处理的记录。所以,索引实际上可以理解为根据某些字段的值进行逻辑排序的一组指针。

目前,很多 DBMS 直接使用主键的概念建立主索引,方法是建立基本表时直接定义主键,即建立了主索引。一个表只能有一个主索引,同时用户还可以建立其他索引,不同的 DBMS 略有区别,如 MySQL 常用的索引有聚集索引、普通索引、唯一索引等;Access 中有主索引、重复索引和非重复索引;SQL Server 中则是聚簇索引、非聚簇索引和唯一索引。

SQL 支持用户根据应用环境的需要,在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。

1. 建立索引

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

其中,表名是指要建立索引的基本表的名字,索引名是用户自己为建立的索引起的名字。索引可以建立在该表的一列或多列上,各列名之间用逗号分隔,这种由两列或多列属性组成的索引,称为复合索引(Composite Index)。

UNIQUE 表明此索引的每个索引值只对应唯一的数据记录。CLUSTER 表示要建立的索引是聚簇索引。聚簇索引是指索引项的顺序与表中记录的物理顺序一致的索引组织。

【例 3-10】 为学生、课程和学习表建立索引。

```
CREATE UNIQUE  INDEX  STU_IDX_SNO  ON  学生 (学号);
CREATE UNIQUE  INDEX   COU_IDX_CNO  ON  课程 (课程号);
CREATE UNIQUE  INDEX  SC_IDX_SNO_CNO ON 学习 (学号 ASC,课程号 DESC);
```

其中,ASC 表示按照升序排列,DESC 表示按照降序排列。

2. 删除索引

```
DROP   INDEX  <索引名>;
```

DROP INDEX 命令可以删除当前数据库内的一个或几个索引。当一个索引被删除后,该索引先前占用的存储空间就会被收回。但是,DROP INDEX 不会影响 PRIMARY KEY 和 UNIQUE 约束条件,这些约束条件必须用 ALTER TABLE…DROP 命令来完成。

【例 3-11】 删除学生表上在学号上建立的索引。

```
DROP  INDEX  STU_IDX_SNO;
```

虽然采用了索引技术可以提高数据查询的速度,但增加了数据插入、删除和修改的复杂性,以及维护索引的时间开销。因此,是否使用索引、对哪些属性建立索引,数据库设计人员必须全面考虑,权衡折中。下面给出几个使用索引的技巧。

(1) 对于小表来说,使用索引性能不会有任何提高。

(2) 索引列中有较多不同的数据时,索引会使性能有极大的提高。

(3) 当查询要返回的数据很少时,索引可以优化查询(比较好的情况是少于全部数据的 25%),如果要返回的数据很多,则索引会加大系统开销。

(4) 索引可以提高数据的返回速度,但是它使得数据的更新操作变慢。如果要进行大量的更新操作,则在执行更新操作时先删除索引,当执行完更新操作后只需要简单地恢复索引即可。

(5) 索引会占用数据库的空间。如果数据库管理系统允许管理数据库的磁盘空间,那么在设计数据库的可用空间时要考虑索引所占用的空间。

(6) 不要对经常需要更新或修改的字段创建索引,更新索引的开销会降低所期望获得的性能。

(7) 不要将索引与表存储在同一个驱动器上。分开存储会去掉访问的冲突,从而使结果返回得更快。

3.3 数 据 查 询

建立数据库的目的就是对数据库进行操作,以便能够从中提取有用的信息。数据库查询是数据库的核心操作,SQL 提供了 SELECT 语句进行数据库的查询。

SQL 92 标准中 SELECT 语句的语法格式如下:

```
SELECT [ALL | DISTINCT] <属性列表>
FROM  <表名或视图名> [,<表名或视图名>]…
[WHERE <条件表达式>]
[GROUP  BY  <列名> ]
[HAVING  <条件表达式>]
[ORDER  BY  <列名>  [ASC | DESC] ];
```

SELECT 语句的含义是,根据 WHERE 子句中的条件表达式,从 FROM 子句指定的基本表中找出满足条件的元组,并按 SELECT 子句中指出的属性,选出元组中的分量形成结果表。

并不是所有的查询都会用到所有的子句,最小限度情况下,查询只需要一个 SELECT 和一个 FROM 子句。实际上,SELECT 子句所完成的功能类似于关系代数中的投影运算,而 WHERE 子句的功能类似于关系代数中的选择运算。

如果有 GROUP 子句选项,则将结果按列名的值进行分组,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,通常会在每组中使用集函数。如果 GROUP 子句带有 HAVING 短语,则结果只有满足指定条件的组才被选出来。

ORDER BY 子句是将查询的结果进行排序显示,ASC 表示升序,DESC 表示降序,默认为升序。

可选项[ALL|DISTINCT]的含义是,如果没有指定 DISTINCT 短语,则默认为 ALL,即保留结果中取值重复的行。

下面将查询分为单表查询、连接查询和嵌套查询进行举例说明,通过这些例子可以看出查询语句的丰富功能和灵活的使用方式。

说明:本节中所有操作均作用于学生成绩管理数据库,该数据库中包括学生表、课程表和学习表,关系模式见表 3-2～表 3-4。

表 3-2 学生表

学　号	姓　名	性　别	籍　贯	出生年份	学　院
091501	王　英	女	河北	1997	计算机
091502	王小梅	女	江苏	2000	信电
091503	张小飞	男	江西	1996	计算机
091504	孙志鹏	男	海南	1998	计算机
091505	徐　颖	女	江苏	1997	外文
091506	钱易蒙	男	河北	2000	信电
…	…	…	…	…	…

表 3-3 课程表

课程号	课程名	学时	先修课程号	课程性质
180101	C++程序设计	56		必修
180102	数据结构	48	180101	必修
180103	操作系统	48	180102	必修
180104	数据库原理	48	180103	必修
180105	DB_Design	32	180104	选修
…	…	…	…	…

表 3-4 学习表

学号	课程号	成绩
091501	180101	78
091501	180102	80
091501	180103	77
091503	180101	89
091503	180102	78
091503	180103	70
091503	180104	90
091504	180101	59
091504	180102	50
091504	180103	
…	…	…

3.3.1 单表查询

最简单的 SQL 查询只涉及一个关系(基本表),可归纳为以下 5 种操作。

(1) 选择表中的若干列(关系代数中的投影运算);

(2) 选择表中的若干元组(关系代数中的选择运算);

(3) 对查询进行分组;

(4) 使用集函数;

(5) 对查询结果排序。

1. SQL 中的投影

关系代数中的投影运算,对应于 SQL 中利用 SELECT 子句指定属性的功能。

【例 3-12】 在学生表中找出所有学生的姓名和籍贯。

```
SELECT  姓名,籍贯
FROM  学生;
```

输出的结果见表 3-5。

与关系代数不同,SQL 语句的投影运算默认获得投影列上的所有元组(包括所有的重复元组),若希望在重复列上仅取一次数值,则添加关键字 DISTINCT,请看例 3-13。

【例 3-13】 在学生表中找出所有学生的籍贯。

```
SELECT  DISTINCT  籍贯
FROM  学生;
```

输出的结果如表 3-6 所示。

表 3-5　例 3-12 结果

姓　名	籍　贯
王　英	河北
王小梅	江苏
张小飞	江西
孙志鹏	海南
徐　颖	江苏
钱易蒙	河北
…	…

表 3-6　例 3-13 结果

籍　贯
河北
江苏
江西
海南
…

【例 3-14】 在学生表中找出学生的所有信息。

```
SELECT  学号,姓名,性别,籍贯,出生年份,院系
FROM  学生;
```

该查询可以在 SELECT 语句中将所有的属性列出,同时也可以使用通配符“*”简化输入,如下所示。

```
SELECT  *
FROM  学生;
```

输出的结果如表 3-7 所示。

表 3-7　例 3-14 结果

学　号	姓　名	性　别	籍　贯	出生年份	学　院
091501	王　英	女	河北	1997	计算机
091502	王小梅	女	江苏	2000	信电
091503	张小飞	男	江西	1996	计算机

续表

学　号	姓　名	性　别	籍　贯	出生年份	学　院
091504	孙志鹏	男	海南	1998	计算机
091505	徐　颖	女	江苏	1997	外文
091506	钱易蒙	男	河北	2000	信电
…	…	…	…	…	…

使用 SELECT 语句不仅可以选择出表中存在的列值,而且可以通过对列值进行算术运算得到表中不存在的信息,请看例 3-15。

【例 3-15】 查询学生的姓名和年龄。

由于表 3-2 中没有年龄属性,因此不能直接列出年龄,但是 SELECT 子句中可以出现计算表达式,从而可以查询经过计算的值。

```
SELECT  学号, year(now()) - 出生年份
FROM  学生;
```

说明:year()和 now()是 MySQL 中处理日期和时间的函数,now()返回为获取当前日期时间,year()为返回指定日期中的年,因此 year(now())获得当前年份。MySQL 还支持其他日期时间处理函数,如 curdate()、curtime()、datediff()……其他数据库管理系统也提供类似的函数,只是名称略有区别,如 SQLite 中获取当前年份的表达式为 strftime('%Y','now'),需要时可以查阅相关的技术手册。

输出的结果如表 3-8 所示。

表 3-8　例 3-15 结果

学　号	Expr1001	学　号	Expr1001
091501	21	091505	21
091502	18	091506	18
091503	22	…	…
091504	20		

【例 3-16】 给列定义别名。

从例 3-15 中可以发现,通过运算得到的列,系统都会自动地赋给它一个列名(Expr1001),这样的列名晦涩难懂,不易理解。在这种情况下,可以添加列的别名,以替换在结果中列出的默认列标题。使用的格式为:

```
COLUMN  AS  <别名>  或者  COLUMN  <别名>
```

例 3-15 运用别名可表示为:

```
SELECT  学号, year(now()) - 出生年份 AS 年龄
FROM  学生;
```

输出的结果如表 3-9 所示。

表 3-9　例 3-16 结果

学　号	年　龄	学　号	年　龄
091501	21	091505	21
091502	18	091506	18

续表

学　号	年　龄	学　号	年　龄
091503	22	…	…
091504	20		

2. SQL 中的选择运算

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表 3-10 所示。

表 3-10　常用的查询条件

查 询 条 件	谓　词
比较	=、<>、>、<、>=、<=
算术运算	+、-、*、/
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

下面分别针对以上列出的查询条件，给出查询的实例。

1）比较运算

【例 3-17】 查询有不及格课程的学生的学号、课程号及成绩。

```
SELECT  学号, 课程号, 成绩
FROM  学习
WHERE  成绩 < 60;
```

输出的结果如表 3-11 所示。

如果只想查看有不及格课程的学生，且重复的学号只输出一次，也是可以的。

【例 3-18】 查询有不及格课程的学生的学号。

```
SELECT  DISTINCT  学号
FROM  学习
WHERE  成绩 < 60;
```

输出的结果如表 3-12 所示。

表 3-11　例 3-17 结果

学　号	课　程　号	成　绩
091504	180101	59
091504	180102	50
…	…	…

表 3-12　例 3-18 结果

学　号
091504
…

2）多重条件和算术运算

【例 3-19】 在学生表中找出信电学院 2000 年后出生的学生的记录。

```
SELECT *
FROM  学生
WHERE  学院 = '信电'
AND  出生年份>=2000;
```

输出的结果如表 3-13 所示。

表 3-13　例 3-19 结果

学　号	姓　名	性　别	籍　贯	出生年份	学　院
091502	王小梅	女	江苏	2000	信电
091506	钱易蒙	男	河北	2000	信电
…	…	…	…	…	…

3) 确定范围

【例 3-20】 查询出生年份在 1996—1998 年(包括 1996 年和 1998 年)的学生的姓名、性别、学院和出生年份。

```
SELECT  姓名, 性别, 学院, 出生年份
FROM  学生
WHERE  出生年份 BETWEEN  1996  AND  1998;
```

输出的结果如表 3-14 所示。

表 3-14　例 3-20 结果

姓　名	性　别	学　院	出生年份
王　英	女	计算机	1997
张小飞	男	计算机	1996
孙志鹏	男	计算机	1998
徐　颖	女	外文	1997
…	…	…	…

该查询等价于：

```
SELECT  姓名, 性别, 学院, 出生年份
FROM  学生
WHERE  出生年份 >= 1996
AND  出生年份 <= 1998;
```

4) 确定集合

【例 3-21】 查询信电学院、理学院和计算机学院的学生的学号、姓名和学院。

```
SELECT  学号, 姓名, 学院
FROM  学生
WHERE  学院 IN ('信电', '理学院', '计算机');
```

输出的结果如表 3-15 所示。

表 3-15　例 3-21 结果

学　号	姓　名	学　院
091501	王　英	计算机
091502	王小梅	信电
091503	张小飞	计算机
091504	孙志鹏	计算机
091506	钱易蒙	信电
091511	李　雷	理学院
…	…	…

与 IN 相对的谓词是 NOT IN，用于查找属性值不属于指定集合的元组。如查找不在信电学院、理学院和计算机学院的学生的学号、姓名和学院的语句如下。

```
SELECT  学号, 姓名, 学院
FROM  学生
WHERE  学院 NOT  IN ('信电', '理学院', '计算机');
```

5）字符匹配

谓词 LIKE 可以用来进行字符串的匹配。其一般语法格式如下：

```
[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。匹配串可以是一个完整的字符串，也可以是含有通配符“%”和“_”。%（百分号）代表任意长度（长度可以为 0）的字符串，_（下画线）代表任意单个字符。

【例 3-22】 查询所有姓王的学生的姓名、学号和性别。

```
SELECT  姓名, 学号, 性别
FROM  学生
WHERE  姓名 LIKE  '王%';
```

输出的结果如表 3-16 所示。

【例 3-23】 查找名字中第二个字为“小”字的学生的姓名和学号。

```
SELECT  姓名, 学号
FROM  学生
WHERE  姓名 LIKE "_小%"
```

输出的结果如表 3-17 所示。

表 3-16　例 3-22 结果

姓　名	学　号	性　别
王　英	091501	女
王小梅	091502	女
…	…	…

表 3-17　例 3-23 结果

姓　名	学　号
王小梅	091502
张小飞	091503
…	…

如果用户要查询的匹配字符串本身就含有“%”或“_”，例如要查课程名为 DB_Design 的课程的学分，应如何实现呢？这时就要使用 ESCAPE'\'对通配符进行转义。

【例 3-24】 查找课程名是 DB_Design 课程的课程号、课程性质。

```
SELECT  课程号, 课程性质
FROM  课程
WHERE  课程名 LIKE 'DB\_Design'  ESCAPE '\'
```

ESCAPE '\'短语表示“\”为转义字符，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。

输出的结果如表 3-18 所示。

6）空值

【例 3-25】 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，下面查询缺少成绩的学生的学号和相应的课程号。

```
SELECT  学号, 课程号, 成绩
```

```
FROM  学习
WHERE 成绩 IS  NULL;
```

注意：这里的 IS 不能用等号(=)代替。

输出的结果如表 3-19 所示。

表 3-18　例 3-24 结果

课　程　号	课 程 性 质
180105	选修
…	…

表 3-19　例 3-25 结果

学　　号	课　程　号	成　　绩
091504	180103	

3. 对查询结果进行分组

GROUP BY 子句可以将查询结果的各行按一列或多列取值相等的原则进行分组。对查询结果分组的目的是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,即每组都有一个函数值。

【例 3-26】 查询各个课程号相应的选课人数。

```
SELECT  课程号, COUNT(学号)  AS 选课人数
FROM  学习
GROUP BY 课程号;
```

输出的结果如表 3-20 所示。

该 SELECT 语句对学习表按课程号的取值进行分组,所有具有相同课程号值的元组为一组,然后对每一组用集函数 COUNT()以求得该组的学生人数。

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 3-27】 查询学号在 091501～091506 至少选修了三门课程的学生的学号和选修课程的课程数。

```
SELECT  学号,COUNT(课程号)  AS 选课数
FROM  学习
WHERE  学号 BETWEEN  '091501'  AND  '091506'
GROUP BY 学号
HAVING  COUNT(课程号)>=3;
```

输出的结果如表 3-21 所示。

表 3-20　例 3-26 结果

课　程　号	选 课 人 数
180101	3
180102	3
180103	3
180104	1
…	…

表 3-21　例 3-27 结果

学　　号	选　课　数
091501	3
091503	4
091504	3
…	…

查询学号在 091501～091506 至少选修三门课程的学生,首先需要通过 WHERE 子句中的条件,从学习表中筛选出学号在 091501～091506 的选课记录；然后根据 GROUP BY 子句中指定的属性进行分组,即学号相同的选课记录为一组；再用集函数 COUNT()对每

一组计数，如果某一组的元组数目大于或等于3，则表示此学生选修的课至少有三门，应将他的学生号选出来。HAVING短语指定选择组的条件，只有满足条件（即元组个数≥3）的组才会被选出来。

说明：WHERE子句与HAVING短语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短语作用于组，从中选择满足条件的组。

4. 使用集函数

为了进一步方便用户，增强检索功能，SQL提供了许多集函数，主要如下。

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>)：计算一列值的总和（此列必须是数值型）。

AVG([DISTINCT|ALL] <列名>)：计算一列值的平均值（此列必须是数值型）。

MAX([DISTINCT|ALL] <列名>)：求一列值中的最大值。

MIN([DISTINCT|ALL] <列名>)：求一列值中的最小值。

如果指定DISTINCT短语，则表示在计算时要取消指定列中的重复值。如果不指定DISTINCT短语或指定ALL短语（ALL为默认值），则表示不取消重复值。

【例3-28】 查询学生总人数。

```
SELECT COUNT( * )  AS 总人数
FROM  学生;
```

输出的结果如表3-22所示。

【例3-29】 查询计算机学院学生的平均年龄。

```
SELECT AVG(year(now()) - 出生年份)  AS 平均年龄
FROM  学生
WHERE  学院 = '计算机';
```

输出的结果如表3-23所示。

【例3-30】 查询学习180101号课程的学生最高分。

```
SELECT MAX(成绩)  AS  最高分
FROM  学习
WHERE  课程号 = '180101';
```

输出的结果如表3-24所示。

表3-22 例3-28结果

总人数
30

表3-23 例3-29结果

平均年龄
21

表3-24 例3-30结果

最高分
95

5. 对查询结果排序

如果没有指定查询结果的显示顺序，DBMS将按其最方便的顺序（通常是元组在表中的先后顺序）输出查询结果。用户也可以用ORDER BY子句指定按照一个或多个属性列的升序（ASC）或降序（DESC）重新排列查询结果，其中，升序ASC为默认值。

【例 3-31】 查询选修了 180102 号课程的学生学号和成绩,查询结果按成绩从高到低排列。

```
SELECT  学号,成绩
FROM  学习
WHERE  课程号 = '180102'
ORDER BY  成绩  DESC;
```

输出的结果如表 3-25 所示。

表 3-25 例 3-31 结果

学 号	成 绩
091501	80
091503	78
091504	50
…	…

【例 3-32】 查询全体学生情况,查询结果按所在学院的名称升序排列,对同一学院中的学生按年龄降序排列。

```
SELECT *
FROM  学生
ORDER BY  学院  ASC, year(now()) - 出生年份 DESC;
```

输出的结果如表 3-26 所示。

表 3-26 例 3-32 结果

学 号	姓 名	性 别	籍 贯	出生年份	学 院
091508	黎 明	男	北京	1998	采矿
091507	郭小娜	女	江苏	1997	机电
091503	张小飞	男	江西	1996	计算机
091501	王 英	女	河北	1997	计算机
091504	孙志鹏	男	海南	1998	计算机
091509	徐 明	男	河北	2000	体育
091505	徐 颖	女	江苏	1997	外文
091502	王小梅	女	江苏	2000	信电
091506	钱易蒙	男	河北	2000	信电
…	…	…	…	…	…

3.3.2 连接查询

一个数据库中的多个表之间一般都存在某种内在联系,它们共同提供有用的信息。前面的查询都是针对一个表进行的。若一个查询同时涉及两个以上的表,则称为连接查询。连接查询根据连接的对象和方法不同,可以包含以下 5 方面的内容。

- 等值连接和非等值连接查询;
- 自身连接查询;
- 外连接查询;
- 复合条件连接查询;
- 集合运算查询。

在 SQL 92 中,连接方法可以分为 ANSI 方式和 theta 方式两类。

(1) ANSI 方式。

该方式通过“表 1 [INNER] JOIN 表 2 ON 连接条件”的语法实现两个表的连接。

【例 3-33】 查询每个学生及其选修课程的情况。

学生情况存放在学生表中,学生选课情况存放在学习表中,所以本查询实际上同时涉及

学生与学习两个表中的数据。这两个表之间的联系是通过两个表都具有的属性“学号”实现的。要查询学生及其选修课程的情况，就必须将这两个表中学号相同的元组连接起来。这是一个等值连接。完成本查询的 SQL 语句为：

```
SELECT  学生.*, 学习.*
FROM  学生  INNER JOIN  学习  ON  学生.学号 = 学习.学号;
```

(2) theta 方式。

该方式通过 WHERE 子句指定条件来进行连接。

例 3-33 的查询语句也可以写成：

```
SELECT  学生.*, 学习.*
FROM  学生, 学习
WHERE  学生.学号 = 学习.学号;
```

输出的结果如表 3-27 所示。

表 3-27　例 3-33 结果

学生.学号	姓名	性别	籍贯	出生年份	学院	学习.学号	课程号	成绩
091501	王　英	女	河北	1997	计算机	091501	180101	78
091501	王　英	女	河北	1997	计算机	091501	180102	80
091501	王　英	女	河北	1997	计算机	091501	180103	77
091503	张小飞	男	江西	1996	计算机	091503	180101	89
091503	张小飞	男	江西	1996	计算机	091503	180102	78
091503	张小飞	男	江西	1996	计算机	091503	180103	70
091503	张小飞	男	江西	1996	计算机	091503	180104	90
…	…	…	…	…	…	…	…	…

1. 等值与非等值连接查询

用来连接两个表的条件称为连接条件或连接谓词，其一般格式为：

```
[<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＝、＞、＜、＞＝、＜＝、＜＞。

此外，连接谓词还可以使用下面的形式。

```
[<表名 1>.]<列名 1> BETWEEN [<表名 2>.]<列名 2> AND [<表名 2>.]<列名 3>
```

当连接运算符为＝时，称为等值连接，使用其他运算符时称为非等值连接。

说明：

(1) 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，但不必相同。例如，可以都是字符型，或都是日期型；也可以一个是整型，另一个是实型，整型和实型都是数值型，因此是可比的。但若一个是字符型另一个是整数型则不允许，因为它们是不可比的类型。

(2) 从概念上讲，DBMS 执行连接操作的过程是，首先在表 1 中找到第一个元组，然后从头开始顺序扫描或按索引扫描表 2，查找满足连接条件的元组，每找到一个元组，就将表 1 中的第一个元组与该元组拼接起来，形成结果表中的一个元组。表 2 全部扫描完毕后，再到表 1 中找第二个元组，然后从头开始顺序扫描或按索引扫描表 2，查找满足连接条件的元组，每找到一个元组，就将表 1 中的第二个元组与该元组拼接起来，形成结果表中的一个元

组。重复上述操作,直到表 1 全部元组都处理完毕为止。

自然连接是等值连接运算中的一种特殊情况,即按照两个表中的相同属性进行等值连接,且目标列中去掉了重复的属性列,但保留了所有不重复的属性列。

【例 3-34】 自然连接学生表和学习表。

```
SELECT 学生.学号, 姓名, 性别, 出生年份, 学院, 课程号, 成绩
FROM 学生, 学习
WHERE 学生.学号 = 学习.学号;
```

输出的结果如表 3-28 所示。

表 3-28 例 3-34 结果

学 号	姓 名	性 别	籍 贯	出生年份	学 院	课程号	成 绩
091501	王 英	女	河北	1997	计算机	180101	78
091501	王 英	女	河北	1997	计算机	180102	80
091501	王 英	女	河北	1997	计算机	180103	77
091503	张小飞	男	江西	1996	计算机	180101	89
091503	张小飞	男	江西	1996	计算机	180102	78
091503	张小飞	男	江西	1996	计算机	180103	70
091503	张小飞	男	江西	1996	计算机	180104	90
…	…	…	…	…	…	…	…

在本查询中,由于姓名、性别、出生年份、学院、课程号和成绩属性列在学生与学习表中是唯一的,因此引用时可以去掉表名前缀。而学号在两个表中都出现了,因此引用时必须加上表名前缀。该查询的执行结果不再出现“学习.学号”列。

2. 自身连接查询

【例 3-35】 求每一门课程的间接先修课(先修课的先修课)。

分析:题目要求查询每一门课程的先修课的先修课,在课程表中,只有每门课的直接先修课信息,而没有先修课的先修课,要得到这个信息,必须先对一门课找到其先修课,再按此先修课的课程号,查找它的先修课,这相当于将课程表与其自身连接后,取第一个副本的先修课程号与第二个副本的课程号进行等值条件连接。具体写 SQL 语句时,为清楚起见,可以为课程表取两个别名,一个是 FIRST,另一个是 SECOND,也可以在考虑问题时就把课程表认为是两个完全一样的表,一个是 FIRST 表,另一个是 SECOND 表,如下所示。

```
SELECT  FIRST.课程号 AS 课程号, FIRST.课程名 AS 课程名, SECOND.先修课程号 AS  间接先修课程号
FROM 课程 AS FIRST, 课程 AS SECOND
WHERE  FIRST.先修课程号 = SECOND.课程号;
```

输出的结果如表 3-29 所示。

表 3-29 例 3-35 结果

课 程 号	课 程 名	间接先修课程号
180102	数据结构	
180103	操作系统	180101
180104	数据库原理	180102
180105	DB_Design	180103
…	…	…

结合表 3-3 可以看到,“C++程序设计”没有先修课,因此更没有间接先修课的信息;“数据结构”的先修课是“C++程序设计”,由于“C++程序设计”没有先修课,因此“数据结构”的间接先修课为空;“操作系统”的先修课是“数据结构”,而“数据结构”的先修课是“C++程序设计”,因此“操作系统”的间接先修课是“C++程序设计”(180101);以此类推其他课程。

2.2.3 节中的例 2-19 也可以通过该方法进行查询,读者可以自行练习。

3. 外连接查询

在通常的连接操作中,只有满足连接条件的元组才能作为结果输出,如在例 3-34 的结果表中没有关于 091511 和 091512 两个学生的信息,原因在于他们没有选课,在选课表中没有相应的元组。但是有时想以学生表为主体列出每个学生的基本情况及其选课情况,若某个学生没有选课,则只输出其基本情况信息,其选课信息为空值即可,这时就需要使用外连接(Outer Join)。

在 SQL 92 中,外连接分为 ANSI 连接和 theta 连接两类,但是目前主要使用 ANSI 连接方式。下面分别以左外连接(Left Outer Join)和右外连接(Right Outer Join)为例进行说明。

1) 左外连接

规定所有记录都应该从连接语句左侧的表中返回。当右侧表中并没有匹配的记录时,左表中该记录依然会返回,而对应的右侧表中的列值将自动填充 NULL 值。

【例 3-36】 查询所有学生的姓名以及他们选修课程的课程号和成绩。

```
SELECT 姓名, 课程号, 成绩
FROM 学生
LEFT OUTER JOIN 学习 ON 学生.学号 = 学习.学号;
```

输出的结果如表 3-30 所示。

表 3-30 例 3-36 结果

姓　名	课 程 号	成　绩
王　英	180101	78
王　英	180102	80
…	…	…
张小飞	180101	89
张小飞	180102	78
…	…	…
孙志鹏	180101	59
孙志鹏	180102	50
…	…	…
于　诚		
…	…	…

2) 右外连接

规定所有记录都应该从连接语句右侧的表中返回。当左侧表中并没有匹配的记录时,

右侧表中的值依然返回,而对应的左侧表中的列值将自动填充 NULL 值。

【例 3-37】 查询所有的课程信息及选修该课程的学生的学号及成绩。

```
SELECT 课程名, 学号, 成绩
FROM 学习
RIGHT OUTER JOIN 课程 ON 学习.课程号 = 课程.课程号;
```

输出的结果如表 3-31 所示。

表 3-31 例 3-37 结果

课 程 名	学 号	成 绩
C++程序设计	091501	78
C++程序设计	091503	89
…	…	…
数据结构	091501	80
数据结构	091503	78
…	…	…
操作系统	091501	77
操作系统	091503	70
…	…	…
算法设计		
…	…	…

全外连接一般没有什么意义,MySQL 并不能直接支持全外连接,但可以通过左、右外连接的并集来模拟实现。

4. 复合条件连接查询

上面各个连接查询中,WHERE 子句中只有一个条件,WHERE 子句中有多个条件的连接操作,称为复合条件连接。

【例 3-38】 查询选修 180101 号课程且成绩在 90 分以上的学生学号、姓名及成绩。

```
SELECT 学生.学号, 姓名,成绩
FROM 学生, 学习
WHERE 学生.学号 = 学习.学号
AND  学习.课程号 = '180101'
AND  学习.成绩> 90
```

以上是复合条件连接的 theta 连接方式,若表示为 ANSI 连接方式,则为以下形式。

```
SELECT 学生.学号, 姓名,成绩
FROM 学生
JOIN 学习 ON 学生.学号 = 学习.学号
AND  学习.课程号 = '180101'
AND  学习.成绩> 90
```

输出的结果如表 3-32 所示。

表 3-32　例 3-38 结果

学　　号	姓　　名	成　　绩
091517	王高飞	91
…	…	…

连接操作除了可以是两表连接、一个表与其自身连接外，还可以是两个以上的表进行连接，后者通常称为多表连接。

【例 3-39】 查询每个学生及其选修的课程名和成绩。

```
SELECT 学生.学号, 姓名,课程名, 学习.成绩
FROM   学生, 学习, 课程
WHERE 学生.学号 = 学习.学号
AND   学习.课程号 = 课程.课程号;
```

以上是多个表的 theta 连接方式，若表示为 ANSI 连接方式，则为以下形式。

```
SELECT 学生.学号, 姓名,课程名, 学习.成绩
FROM 学生
  JOIN  学习 ON 学生.学号 = 学习.学号
  JOIN  课程 ON 学习.课程号 = 课程.课程号
```

输出的结果如表 3-33 所示。

表 3-33　例 3-39 结果

学　　号	姓　　名	课　程　名	成　　绩
091501	王　英	C++程序设计	78
091501	王　英	数据结构	80
091501	王　英	操作系统	77
091503	张小飞	C++程序设计	89
091503	张小飞	数据结构	78
091503	张小飞	操作系统	70
…	…	…	…

5. 集合运算连接查询

有时候，用户希望在 SQL 查询中利用关系代数中的集合运算（并、交、差）来组合关系，SQL 为此提供了相应的运算符：UNION、INTERSECT、EXCEPT，分别对应于集合运算的∪、∩、－。它们用于两个查询之间，对每个查询都要用圆括号括起来。对于不同的 DBMS，支持的集合运算有所不同，如 MySQL 只支持并运算。

【例 3-40】 查询选修了 180101 号或 180102 号课程或二者都选修了的学生学号、课程号和成绩。

```
(SELECT  学号, 课程号, 成绩
FROM   学习
WHERE   课程号 = '180101')
   UNION
(SELECT  学号, 课程号, 成绩
FROM  学习
WHERE    课程号 = '180102')
```

输出的结果如表 3-34 所示。

与 SELECT 子句不同，UNION 运算自动去除重复。因此在本例中，若只输出学生的

学号,则相同的学号只出现一次。如果想保留所有的重复,则必须用 UNION ALL 代替 UNION,且查询结果中出现的重复元组数等于两个集合中出现的重复元组数的和。

表 3-34 例 3-40 结果

学　号	课 程 号	成　绩
091501	180101	78
091501	180102	80
091503	180101	89
091503	180102	78
…	…	…

【例 3-41】 查询同时选修了 180101 和 180102 号课程的学生学号。

```
(SELECT  学号
FROM   学习
WHERE 课程号 = '180101')
INTERSECT
(SELECT 学号
FROM 学习
WHERE 课程号 = '180102')
```

INTERSECT 运算自动去除重复,如果想保留所有的重复,必须用 INTERSECT ALL 代替 INTERSECT,结果中出现的重复元组数等于两集合出现的重复元组数里较少的那个。

【例 3-42】 查询选修了 180101 号课程的学生中没有选修 180102 号课程的学生学号。

```
(SELECT  学号
FROM   学习
WHERE  课程号 = '180101')
EXCEPT
(SELECT 学号
FROM 学习
WHERE   课程号 = '180102')
```

EXCEPT 运算自动去除重复,如果想保留所有的重复,必须用 EXCEPT ALL 代替 EXCEPT,结果中出现的重复元组数等于两集合出现的重复元组数之差(前提是差是正值)。

在不支持 INTERSECT 和 EXCEPT 运算的 DBMS 中,必须使用其他方法实现。其中,嵌套查询是十分有效的一种方法。

3.3.3 嵌套查询

在 SQL 中,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 条件中的查询称为嵌套查询。请看下面的例子。

【例 3-43】 查询选修了 180101 号课程的学生姓名。

可以使用如下的嵌套形式实现该查询。

```
SELECT  姓名
FROM  学生
WHERE  学号 IN
(SELECT  学号
  FROM  学习
  WHERE  课程号 = '180101');
```

说明：在这个例子中，下层查询块“SELECT 学号 FROM 学习 WHERE 课程号＝'180101'”是嵌套在上层查询块“SELECT 姓名 FROM 学生 WHERE 学号 IN”的WHERE条件中的。上层的查询块又称为外层查询、父查询或主查询，下层查询块又称为内层查询或子查询。SQL允许多层嵌套查询，即一个子查询中还可以嵌套其他子查询。

嵌套查询的求解方法是由里向外一层层处理。根据子查询是否独立于父查询，可以将嵌套查询分为两大类：①不相关子查询，该类中的子查询独立于上层的父查询，每个子查询在其上一级查询处理之前可以完成求解，子查询的结果用于建立其父查询的查找条件；②相关子查询，该类中的子查询依赖于父查询，最内层子查询的执行需要用到上层父查询中的某些属性值，因此最内层的子查询不能独立于父查询先完成。

嵌套查询使得可以用一系列简单查询构成复杂的查询，从而明显地增强了SQL的查询能力。以层层嵌套的方式来构造程序正是SQL中“结构化”的含义所在。

嵌套查询主要内容如下。

1. 带有IN谓词的子查询

带有IN谓词的子查询是指父查询与子查询之间用IN进行连接，判断某个属性列值是否在子查询的结果中。由于在嵌套查询中，子查询的结果往往是一个集合，因此谓词IN是嵌套查询中最经常使用的谓词。

【例3-44】 查询与“王颖”在同一个学院学习的学生学号和姓名。

查询与“王颖”在同一个学院学习的学生，可以首先确定“王颖”所在学院名，再查找所有在该学院学习的学生。所以可以分步来完成此查询。

（1）确定“王颖”所在学院名。

```
SELECT  学院
FROM  学生
WHERE  姓名 = '王颖';
```

假设结果为Q。

（2）查找所有在Q学院学习的学生学号和姓名。

```
SELECT  学号, 姓名
FROM  学生
WHERE  学院 IN 'Q';
```

若用嵌套查询的形式，则可以表示为：

```
SELECT  学号, 姓名
FROM  学生
WHERE  学院  IN
(SELECT  学院
FROM  学生
WHERE  姓名 = '王颖');
```

【例3-45】 查询选修了“数据库原理”课程的学生学号和姓名。

```
SELECT  学号, 姓名
FROM  学生
WHERE  学号  IN
(SELECT  学号
FROM  学习
WHERE  课程号  IN
```

```
    (SELECT   课程号
     FROM     课程
       WHERE  课程名 = '数据库原理'));
```

输出的结果如表 3-35 所示。

本查询也可以用非嵌套的形式表示。

```
SELECT  学生.学号,姓名
FROM    学生,学习,课程
WHERE   课程.课程名称 = '数据库原理'
AND     学生.学号 = 学习.学号
AND   学习.课程号 = 课程.课程号;
```

表 3-35　例 3-45 结果

学　　号	姓　　名
091503	张小飞
…	…

2. 带有比较运算符的子查询

例 3-45 中“数据库原理”课程的课程号是唯一的,但选修该课程的学生并不只有一个,所以也可以用“=”运算符和 IN 谓词共同完成。

```
SELECT  学号, 姓名
FROM  学生
WHERE  学号 IN
  (SELECT  学号
  FROM  学习
  WHERE  课程号 =
      (SELECT  课程号
        FROM  课程
      WHERE  课程名 = '数据库原理'));
```

3. 带有 ANY 或 ALL 谓词的子查询

子查询返回单值时可以用比较运算符。当返回的结果有可能不只一个时,则不能使用比较运算符,此时,可以使用 ANY 或 ALL 谓词来实现比较操作,而使用 ANY 或 ALL 谓词时则必须同时使用比较运算符,其语义如表 3-36 所示。

表 3-36　比较运算符

运　算　符	ANY	ALL
>	大于子查询结果中的某个值	大于子查询结果中的所有值
<	小于子查询结果中的某个值	小于子查询结果中的所有值
>=	大于或等于子查询结果中的某个值	大于或等于子查询结果中的所有值
<=	小于或等于子查询结果中的某个值	小于或等于子查询结果中的所有值
=	等于子查询结果中的某个值	通常没有实际意义
<>	不等于子查询结果中的某个值	不等于子查询结果中的任何一个值

【例 3-46】 查询其他学院中比计算机学院某个学生年龄小的学生名单。

```
SELECT  姓名
FROM  学生
WHERE  year(now()) - 出生年份< ANY
   (SELECT  year(now()) - 出生年份
    FROM   学生
    WHERE  学院 = '计算机' )
    AND  学院 <> '计算机'
ORDER BY  year(now()) - 出生年份 DESC;
```

该查询的含义是，其他学院中的学生只要比计算机学院中的某个学生的年龄小，就可以选择满足条件，换句话说，若其他院系中的某个学生的年龄比计算机学院中的年龄最大的学生小，则该学生一定满足选择的条件，所以本查询实际上也可以用集函数 MAX()实现。语句如下。

```
SELECT  姓名, year(now()) - 出生年份
FROM  学生
WHERE  year(now()) - 出生年份<
(SELECT  MAX(year(now()) - 出生年份)
FROM 学生
WHERE  学院 = '计算机')
AND  学院 <> '计算机'
ORDER  BY year(now()) - 出生年份 DESC;
```

事实上，用集函数实现子查询通常比直接用 ANY 或 ALL 查询效率要高。ANY、ALL 谓词与集函数及 IN 谓词的对应关系如表 3-37 所示。

表 3-37　ANY、ALL 谓词与集函数及 IN 谓词的对应关系

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

上述嵌套查询均属于不相关子查询，其执行过程都是按照由内到外的顺序，先独立完成最内层的子查询，然后将查询的结果作为上层父查询的条件，由上层父查询再继续逐层执行，最终完成整个查询过程。

4. 带有 EXISTS 谓词的子查询

EXISTS 代表存在量词∃。带有 EXISTS 谓词的子查询不返回任何数据，只产生逻辑真值 true 或者逻辑假值 false。

【例 3-47】 查询选修了 180102 号课程的学生学号和姓名。

使用 EXISTS 的嵌套查询语句表示如下。

```
SELECT 学号,姓名
FROM  学生
WHERE  EXISTS
    (SELECT  *
     FROM  学习
    WHERE  学生.学号 = 学习.学号
    AND    学习.课程号 = '180102');
```

输出的结果如表 3-38 所示。

表 3-38　例 3-47 结果

姓　名	学　号
王　英	091501
张小飞	091503
…	…

该语句的执行顺序是，从学生表中依次取出每个元组的学号，用此值去检查学习表。若学习表中存在这样的元组，其学号值与此学生.学号值相等，并且该学生学习的课程号为 180102，则取此学生的学号和姓名送入结果表中。

由 EXSITS 引出的子查询，其目标列表达式通常都用“*”，因为带 EXSITS 的子查询只返回真值或假值，给出列名无实际意义。

与 EXISTS 谓词相对应的是 NOT EXISTS 谓词，若内层结果为空，则 NOT EXISTS

结果为真,外层的 WHERE 子句返回真值,否则返回假值。

【例 3-48】 查询没有选修 180102 号课程的学生学号和姓名。

使用 NOT EXISTS 的嵌套查询语句表示如下。

```
SELECT 学号,姓名
FROM   学生
WHERE  NOT  EXISTS
      (SELECT   *
       FROM  学习
       WHERE  学生.学号 = 学习.学号
       AND   学习.课程号 = '180102');
```

该语句的执行顺序是,从学生表中依次取出每个元组的学号,用此值去检查学习表。若学习表中某元组不存在这样的情况,其学号值与此学生.学号值相等,并且该学生学习的课程号为 180102,则取此学生的学号和姓名送入结果表中,从而表示了没有选修 180102 号课程的学生信息,实现了差运算相同的效果。

可以看出,由 EXISTS 和 NOT EXISTS 引导的子查询属于相关子查询,因为子查询在执行过程中需要用到父查询的表中相应的属性值。

注意:一些带 EXISTS 或 NOT EXISTS 谓词的子查询不能被其他形式的子查询所代替,但是所有带 IN 谓词、比较运算符、ANY 和 ALL 谓词的子查询都能够用 EXISTS 谓词的子查询等价替换。

例如,例 3-45 可以用带 EXISTS 的子查询替换为:

```
SELECT   学号, 姓名
FROM     学生
WHERE   EXISTS
   (SELECT *
      FROM  学习,课程
     WHERE 学习.课程号 = 课程.课程号
      AND   学生.学号 = 学习.学号
     AND   课程名 = '数据库原理')
```

【例 3-49】 查询至少选修了 091501 号学生选修的全部课程的学生学号。

本查询是一个典型的除法问题,其关系代数可以表示为:

$$\prod_{学号,课程号}(学习) \div \prod_{课程号}(\sigma_{学号='091501'}(学习))$$

通过第 2 章的学习可知,能够使用除法的查询,其查询条件中必含有“一个集合包含另一个集合”的概念,因此在转换为 SQL 语句时,首先要找出这两个集合,其次需要通过一定的运算来确定一个集合包含另一个集合。

设有两个集合 A 和 B,证明 A 包含 B,即 A⊇B。如果正向证明,则只能通过穷举的方法,即检查 B 集合中的每个元素是否在 A 集合中。可以看出,该方法不具有通用性和自动化性。因此考虑用逆向证明,B－A 得到的是“在 B 集合中有的,而在 A 集合中没有的元素”,如果该结果集为空,则说明 B 集合中的元素都存在于 A 集合中,即 A⊇B。

因此可以得出 A⊇B ≡ ¬∃(B－A),其中,¬∃表示“不存在”。

在本题中,设 A 集合表示某个学生选修课程的课程号集合,B 集合表示 090101 号学生选修的课程号集合,根据题意,如果 A 集合大于或等于 B 集合,即 A⊇B,则当前这个学生是要查找的学生,需要将其学号送入结果表。对应的 SQL 语句如下。

(1) 明确两个集合。

A 集合表示为：

```
SELECT  课程号
FROM  学习
WHERE 学习.学号 = 'xxx'(xxx 表示某个学生的学号);
```

B 集合表示为：

```
SELECT  课程号
FROM  学习
WHERE 学习.学号 = '091501';
```

(2) 表示 A 包含 B，即 ¬ ∃(B－A)。

```
NOT EXISTS
(  (SELECT  课程号
    FROM  学习 AS  First
    WHERE  First.学号 = '091501')
   EXCEPT
      (SELECT  课程号
       FROM  学习 AS  Second
       WHERE  Second.学号 = 'xxx'));
```

其中，由于用到学习表两次，为了方便引用，分别对其取了别名。

(3) 得到查询的信息。

本题最终要查找的是学生的学号，而查找过程中需要对每个学生进行筛选，因此选用学生表：

```
SELECT  学号
FROM  学生
WHERE  NOT EXISTS
  (  (SELECT  课程号
      FROM  学习 AS  First
      WHERE  First.学号 = '091501')
    EXCEPT
      (SELECT 课程号
       FROM  学习 AS  Second
       WHERE  Second.学号 = 学生.学号));
```

此时，Second.学号不再是“未知值”，而是“学生.学号”，因为要对学生表中的每个学生检查一遍，故从学生表中依次取出每个学生的学号，放到内层查询中进行检验，符合条件的学生将其学号送入结果表中。

由于“差运算”也可以由 NOT EXISTS 来表示，因此可以进一步表示成如下形式。

```
SELECT  学号
FROM 学生
WHERE  NOT EXISTS
  ( SELECT *
   FROM  学习 AS  First
   WHERE  First.学号 = '091501' AND  NOT EXISTS
     ( SELECT *
      FROM  学习 AS  Second
```

```
        WHERE   Second.学号 = 学生.学号
         AND     Second.课程号 =  First.课程号));
```

说明：由于 NOT EXISTS 不关心返回的内容，只关心是否有值返回，因此内层查询的 SELECT 语句中可以用“*”替代。

【例 3-50】 查询选修了全部课程的学生的姓名。

本查询也是一个典型的除法问题，其关系代数可以表示为：

$$\prod_{姓名,课程号}(学习 \infty 学生) \div \prod_{课程号}(课程)$$

设 A 代表某个学生的选课集合，B 代表全部课程的集合，根据题目要求“选修了全部课程”可知，A 集合中的课程号应该包含 B 集合中的所有课程号(即 A⊇B)，符合该条件的学生的姓名就是最终查询的结果。因此，可以按如下三步进行除法的求解。

(1) 明确两个集合。

A 集合表示为：

```
SELECT  课程号
FROM  学习
WHERE 学习.学号 = 'xxx'(xxx 表示某个学生的学号);
```

B 集合表示为：

```
SELECT  课程号
FROM  课程;
```

(2) 表示 A 包含 B，即 ¬ ∃(B－A)。

```
NOT EXISTS
(  (SELECT 课程号
    FROM  课程;)
   EXCEPT
    (SELECT 课程号
     FROM  学习
     WHERE  学习.学号 = 'xxx'));
```

(3) 得到查询的信息。

```
SELECT 姓名
FROM 学生
WHERE  NOT EXISTS
(  (SELECT *
   FROM 课程
   WHERE NOT EXISTS
    (SELCET *
     FROM  学习
     WHERE 学号 = 学生.学号
     AND     课程号 = 课程.课程号));
```

【例 3-51】 查询被所有学生都选修的课程名称。

```
SELECT  课程名
FROM  课程
WHERE NOT EXISTS
(SELECT *
    FROM 学生
    WHERE NOT EXISTS
```

```
(SELCET *
        FROM  学习
    WHERE  学号 = 学生.学号
    AND    课程号 = 课程.课程号));
```

通过上述例题可以看出,除法的求解主要分为三步:首先根据题目描述,找出两个集合A和B,并且明确两个集合之间的包含关系A⊇B;然后表示出A包含B的关系,也就是写成NOT EXISTS(B－A)的形式;最后根据题目所要查询的信息,写出最外层的查询,并完善最内层查询的条件。由于子查询的连接谓词是NOT EXISTS,因此要在最内层的查询条件中给出需要用到父查询中的属性。

3.4 数据更新

SQL对数据的更新包括数据插入、删除和修改三方面的功能。

3.4.1 插入数据

1. 插入单个元组

插入语句的一般格式为:

```
INSERT
INTO  <表名>  [(<属性列1>[,<属性列2>,…])
VALUES(<常量1>[,<常量2>]…);
```

如果某些属性列在INTO子句中没有出现,则新记录在这些列上将取空值。但必须注意的是,在表定义时说明了NOT NULL的属性列不能取空值,否则会出错。

如果INTO子句中没有指明任何列名,则新插入的记录必须在每个属性列上均有值且新插入记录的属性值与原表属性应一一对应。

【例3-52】 将一个新学生记录(学号:091530;姓名:夏雨;性别:男;籍贯:海南;出生年份:1999;学院:计算机)插入学生表中。

```
INSERT
INTO  学生
VALUES ('091530', '夏雨', '男','海南', '1999','计算机');
```

子查询也可以嵌套在INSERT语句中,用于生成要插入的数据。其功能是以批量插入,一次将子查询的结果全部插入指定表中。

2. 插入子查询结果

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

【例3-53】 设有关系模式DEPT_AGE(SDEPT ChAR(20),AVG_AGE SMALLINT),表示每个学院学生的平均年龄,请根据学生表中的数据,求得结果后存入数据库。

```
INSERT
INTO DEPT_AGE(SDEPT, AVG_AGE)
SELECT 学院, AVG(year(now()) - 出生年份)
```

```
FROM 学生
GROUP BY 学院
```


3.4.2 删除数据

删除语句的一般格式为：

```
DELETE
FROM  <表名>
[WHERE  <条件>];
```

删除可以分为如下几种。

(1) 删除某个(某些)元组的值：WHERE 子句中给出删除条件。

(2) 删除全部元组的值：省略 WHERE 子句。

(3) 带子查询的删除语句。

【例 3-54】 删除学号为 092010 的学生记录。

```
DELETE  FROM  学生
WHERE 学号 = '092010';
```

【例 3-55】 删除计算机学院所有学生的选课记录。

```
DELETE
FROM  学习
WHERE 学号 IN
  ( SELECT  学号
    FROM  学生
    WHERE  学院 = '计算机');
```

带有子查询的删除操作的执行过程类似于相关子查询。如例 3-55 中,首先考查父查询学习表的第一条记录,取其学号值放到子查询中执行；如果查询的结果为“计算机”,使得父查询中 WHERE 条件为真,则删除当前考查的这条记录；接着考查学习表中的第二条记录,以此类推。

3.4.3 修改数据

修改操作语句的一般格式为：

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

其功能是修改指定表中满足 WHERE 子句条件的元组。其中,SET 子句用于指定修改方法,即用表达式的值取代相应的属性列值。如果省略 WHERE 子句,则表示要修改表中的所有元组。

1. 更新表中的一个元组

【例 3-56】 将 091611 号学生的籍贯改为江苏。

```
UPDATE  学生
SET  籍贯 = '江苏'
WHERE  学号 = '091611';
```

2. 更新表中多个元组的数据

【例 3-57】 将 180101 号课程的成绩增加一分。

```
UPDATE  学习
SET  成绩 = 成绩 + 1
WHERE  课程号 = '180101';
```

3. 带子查询的修改

【例 3-58】 将计算机学院学生的成绩清零。

```
UPDATE  学习
SET  成绩 = 0
WHERE  学号 IN
  ( SELECT  学号
    FROM  学生
    WHERE  学院 = '计算机');
```

带有子查询的修改操作的执行过程与带子查询的删除操作类似。如例 3-58 中，首先考查父查询学习表的第一条记录，取其学号值放到子查询中执行；如果查询的结果为“计算机”，使得父查询中 WHERE 条件为真，则将当前考查的这条记录中的成绩修改为零；接着考查学习表中的第二条记录，以此类推。

3.5 视　　图

视图是从一个或几个基本表(或视图)导出的表，因此是一种非标准的子模式概念。一个用户可以定义若干视图，这样，用户的外模式就由若干基本表和若干视图组成。视图一旦被定义，就可以对它查询，在某些情况下甚至可以修改。

3.5.1 建立视图

建立视图的一般格式为：

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

DBMS 执行 CREATE VIEW 语句时只是把视图的定义存入数据字典，并不执行其中的子查询语句。所以数据库中只存放视图的定义，而不存放对应的数据，这也是视图被称为虚表的原因。

构成视图的属性列或者全部省略，或者全部给出，没有其他情况。如果全部省略，则构成视图的属性列由子查询中 SELECT 子句中的诸字段确定。但在下列三种情况下必须明确指定组成视图的所有列名。

(1) 某个目标列是集函数或列表达式。

(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。

WITH CHECK OPTION 表示对视图进行插入、删除和修改操作时要保证发生变动的行满足视图定义中的谓词条件(即子查询中的条件表达式)。

【例 3-59】 建立计算机学院学生的视图。

```
CREATE  VIEW CS_VIEW
AS  SELECT *
FROM  学生
WHERE  学院 = '计算机';
```

该语句生成一个名为 CS_VIEW 的视图,其中的属性列与学生表中的属性列一致,它相当于基本表的一个映像,只有在运行视图时,才会得到数据,不运行时,视图只是一个表的模式的定义。

需要注意的是,以 SELECT * 方式创建的视图可扩充性差,应尽可能避免。原因在于该类视图(CS_VIEW)中的属性列与定义视图时的基本表(学生表)中的属性列自动形成映射关系,当基本表的结构发生变化时,视图(CS_VIEW)与基本表(学生表)的映射关系被破坏,导致该类视图不能正确工作。为了避免出现这类问题,最好在修改基本表之后删除由其导出的视图,然后重建视图;或者将视图的建立语句改写为如下形式。

```
CREATE  VIEW CS_VIEW(学号, 姓名, 性别, 籍贯, 出生年份, 学院)
AS SELECT  学号, 姓名, 性别, 籍贯, 出生年份, 学院
FROM  学生
WHERE  学院 = '计算机';
```

(1) 视图可以建立在多个表上。

【例 3-60】 建立计算机学院选修了“数据库原理”这门课的学生的视图。

```
CREATE  VIEW DB_S1
AS
SELECT  学生.学号,姓名,性别,籍贯,学院,成绩
FROM  学生, 学习,课程
WHERE  课程名 = '数据库原理'
AND  学生.学号 = 学习.学号
AND  课程.课程号 = 学习.课程号
AND  学院 = '计算机';
```

(2) 视图可以建立在其他的视图上。

【例 3-61】 建立计算机学院选修“数据库原理”课程且成绩在 90 分以上的学生的视图。

```
CREATE  VIEW DB_S2
AS
SELECT *
FROM DB_S1
WHERE  成绩>= 90;
```

定义基本表时,为了减少数据库中的冗余数据,表中只存放基本数据,由基本数据经过各种计算派生出的数据一般是不存储的。由于视图中的数据并不实际存储,因此定义视图时可以根据应用的需要,设置一些派生属性列。这些派生属性由于在基本表中并不实际存在,因此有时也称它们为虚拟列。带虚拟列的视图称为带表达式的视图。

【例 3-62】 定义一个反映学生年龄的视图。

```
CREATE  VIEW BT_S(学号, 姓名, 年龄)
AS
SELECT  学号, 姓名, year(now()) - 出生年份 AS 年龄
FROM  学生;
```

3.5.2 删除视图

删除视图的一般格式为：

```
DROP  VIEW  <视图名>
```

一个视图被删除后，由此视图导出的其他视图也将失效，用户应该使用 DROP VIEW 语句将它们一一删除。

3.5.3 查询视图

视图定义后，用户就可以像对基本表进行查询一样对视图进行查询。DBMS 执行对视图的查询时，首先进行有效性检查，检查查询涉及的表、视图等是否在数据库中存在，如果存在，则从数据字典中取出查询涉及的视图的定义，把定义中的子查询和用户对视图的查询结合起来，转换为对基本表的查询，然后执行这个经过修正的查询。将对视图的查询转换为对基本表的查询的过程称为视图的消解(View Resolution)。

【例 3-63】 在计算机学院学生的视图中找出年龄小于 20 岁的学生。

```
SELECT *
FROM  CS_VIEW
WHERE  year(now()) - 出生年份< 20;
```

DBMS 执行此查询时，将其与 CS_VIEW 视图定义中的子查询

```
CREATE  VIEW CS_VIEW
AS SELECT *
FROM 学生
WHERE  学院 = '计算机'
```

结合起来，转换为对基本表学生表的查询，消解后的查询语句为：

```
SELECT *
FROM  学生
WHERE  学院 = '计算机' AND year(now()) - 出生年份< 20;
```

3.5.4 更新视图

更新视图包括插入(Insert)、删除(Delete)和修改(Update)三类操作。并非所有的视图都允许更新，只有某些非常简单的视图(称为可更新视图，Updatable View)，在把对视图的更新操作转换为对基本表的等价操作后，允许对这些视图进行更新。

到底什么样的视图是可更新的？若一个视图是从单个基本表导出的，并且只是去掉了某些行和列(不包括关键字)，如视图 CS_VIEW，称这类视图为行列子集视图。目前，关系系统只提供对行列子集视图的更新，并具有以下限制。

(1) 若视图的属性来自属性表达式或常数，则不允许对视图执行 INSERT 和 UPDATE 操作，但允许执行 DELETE 操作。

(2) 若视图的属性来自库函数，则不允许对此视图更新。

(3) 若视图定义中有 GROUP BY 子句，则不允许对此视图更新。

(4) 若视图定义中有 DISTINCT 选项，则不允许对此视图更新。

(5) 若视图定义中有嵌套查询，并且嵌套查询的 FROM 子句涉及导出该视图的基本表，则不允许对此视图更新。

(6) 若视图由两个以上的基本表导出,则不允许对此视图更新。

(7) 如果在一个不允许更新的视图上再定义一个视图,这种二次视图是不允许更新的。

【例 3-64】 以下视图的更新就是不允许的。

```
CREATE  VIEW GOOD_S_C_VIEW
AS
SELECT 学号, 课程号, 成绩
FROM 学习
WHERE 成绩>
    (SELECT  AVG(成绩)
      FROM 学习);
```

3.6 数 据 控 制

由 DBMS 提供统一的数据控制功能是数据库系统的特点之一。数据控制也称为数据保护,包括数据的安全性控制、完整性控制、并发控制和恢复。这里主要介绍 SQL 的安全控制中的存取控制的实现,其他详细内容见第 6 章数据库保护。

3.6.1 授权

SQL 用 GRANT 语句向用户授予操作权限,GRANT 语句的一般格式为:

```
GRANT <权限>[,<权限>]…
[ON <对象类型> <对象名>]
[TO <用户>[,<用户>]…
[WITH GRANT OPTION];
```

其语义为将对指定操作对象的指定操作权限授予指定的用户。

对不同类型的操作对象有不同的操作权限,常见的操作权限如表 3-39 所示。

表 3-39 不同对象类型允许的操作权限

对 象	对 象 类 型	操 作 权 限
属性列	TABLE	SELECT,INSERT,UPDATE,DELETE,ALL PRIVILEGES
视图	TABLE	SELECT,INSERT,UPDATE,DELETE,ALL PRIVILEGES
基本表	TABLE	SELECT,INSERT,UPDATE,DELETE ALTER, INDEX,ALL PRIVILEGES
数据库	DATABASE	CREATETAB

接受权限的用户可以是一个或多个具体用户,也可以是 PUBLIC 即全体用户。如果指定了 WITH GRANT OPTION 子句,则获得某种权限的用户还可以把这种权限再授予别的用户。如果没有指定 WITH GRANT OPTION 子句,则获得某种权限的用户只能使用该权限,但不能传播该权限。

【例 3-65】 把学生表的查询权限授予用户 User1。

```
GRANT SELECT ON TABLE 学生 TO User1;
```

【例 3-66】 把学生表和课程表的全部权限授予用户 User2 和 User3。

```
GRANT ALL PRIVILEGES ON TABLE 学生, 课程 TO User2, User3;
```

【例 3-67】 把学习表的查询权限授予全部用户。

```
GRANT SELECT ON TABLE 学习 TO PUBLIC;
```

【例 3-68】 把查询学习表和修改成绩的权限授予用户 User4。

```
GRANT UPDATE(成绩), SELECT ON TABLE 学习 TO User4;
```

【例 3-69】 把学生表的 INSERT 权限授予 User5 用户，并允许他再将此权限授予其他用户。

```
GRANT INSERT ON TABLE 学生 TO User5 WITH GRANT OPTION;
```

3.6.2 收回权限

授予的权限可以由 DBA 或其他授权者用 REVOKE 语句收回，REVOKE 语句的一般格式为：

```
REVOKE <权限>[,<权限>]…
[ON <对象类型> <对象名>]
[FROM <用户>[,<用户>]…
```

【例 3-70】 把用户 User4 修改成绩的权限收回。

```
REVOKE  UPDATE(成绩) ON TABLE 学习 FROM User4;
```

【例 3-71】 把用户 User5 对学生表的 INSERT 权限收回。

```
REVOKE INSERT ON TABLE 学生 FROM User5;
```

在例 3-69 中，用户 User5 获得了学生表的 INSERT 权限，并且可以将该权限授予其他用户。若 User5 将该权限授予了 User6，执行此 REVOKE 语句后，DBMS 在收回 User5 对学生表的 INSERT 权限的同时，还会自动收回 User6 对学生表的 INSERT 权限，即收回权限的操作会级联下去。但如果 User6 还从其他用户处获得对学生表的 INSERT 权限，则他仍具有此权限，系统只收回直接或间接从 User5 处获得的权限。

小　　结

SQL 是关系数据库的标准语言，已经广泛应用在商务系统中。SQL 主要由数据定义、数据操作、数据控制语句组成。

SQL 的数据定义部分包括对 SQL 基本表、视图、索引的创建和撤销。

SQL 的数据操作分成数据查询和数据更新两部分。

SQL 的数据查询用 SELECT 语句实现，兼有关系代数和元组演算的特点。查询的类型有单表查询、连接查询和嵌套查询，同时介绍了聚集函数、分组子句和排序子句的使用方法。

SQL 的数据更新包括数据的插入、删除和修改三种操作。

SQL 中的视图是一个虚表，只给出了结构的定义，并不存储数据。本章主要介绍了视图的定义、查询和更新操作，其中，只有行列子视图才允许进行更新。

习　题　3

3.1　试述 SQL 的特点。

3.2 解释下列术语。

SQL 模式 基本表 视图 单表查询 连接查询 嵌套查询

3.3 试述 SQL 的特点。

3.4 试述 SQL 体系结构和关系数据库模式之间的关系。

3.5 SQL 是如何实现实体完整性、参照完整性和用户定义完整性的?

3.6 讨论当对一个视图进行更新时可能会出现什么样的问题。

3.7 设有两个基本表 R(A,B,C) 和 S(A,B,C),试用 SQL 查询语句表达下列关系代数表达式。

(1) R∩S (2) R−S (3) R∪S (4) R×S

3.8 对于教学数据库的三个基本表:

S(学号,姓名,年龄,性别)
SC(学号,课程号,成绩)
C(课程号,课程名,任课教师姓名)

其中,下画线标记的是主码。试用 SQL 语句表达下列查询。

(1) 查询姓刘的老师所授课程的课程号和课程名。

(2) 查询年龄大于 23 岁的男同学的学号和姓名。

(3) 查询学号为 S3 的学生所学课程的课程号、课程名和任课教师姓名。

(4) 查询“张小飞”没有选修的课程号和课程名。

(5) 查询至少选修了三门课程的学生的学号和姓名。

(6) 查询全部学生都选修了的课程编号和课程名称。

(7) 在 SC 中删除尚无成绩的选课元组。

(8) 把“高等数学”课的所有不及格成绩都改为 60。

(9) 把低于总平均成绩的女同学的成绩提高 5%。

(10) 向 C 中插入元组('C8','VC++','王昆')。

3.9 设有下列 4 个关系模式:

PRODUCT(MAKER,MODEL,TYPE)
PC(MODEL,SPEED,RAM,HD,CD,PRICE)
LAPTOP(MODEL,SPEED,RAM,SCREEN,PRICE)
PRINTER(MODEL,COLOR,TYPE,PRICE)

注:PRODUCT 表中 TYPE 属性列的取值为 pc 或 laptop 或 printer;PRINTER 表中 color 属性列的取值为 true 或 false,代表彩色或单色。

试用 SQL 语句表达下列查询。

(1) 找出价格高于 15 000 元,并且运行速度低于同价位 PC 的平均运行速度的 LAPTOP。

(2) 找出生产价格最低的彩色打印机的厂家。

(3) 计算由厂家 HP 生产的 PC 和 LAPTOP 的平均价格。

(4) 计算各厂商所生产的 LAPTOP 的显示器的平均尺寸。

(5) 找出每一个生产厂商的 PC 的最高价格。

(6) 计算生产打印机的各个厂商所生产的 PC 的硬盘的平均容量。

3.10 试设计如图 3-2 中显示的数据库模式 Library,用来记录书籍、借书人和书籍借

出的情况，下画线标识每个表的主码，参照完整性在图 3-2 中用有向边来表示（备注：DateOut 记录书籍借出的日期，DueOut 记录书籍应还的日期，Is_return 记录书籍是否归还，日期格式为“年/月/日”）。

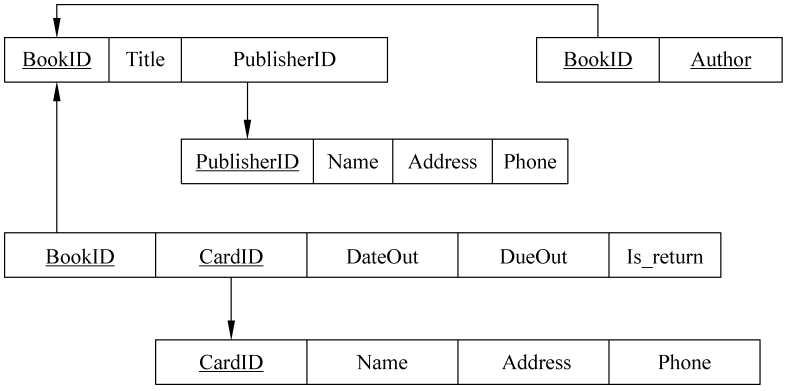

图 3-2　数据库模式 Library

3.11　针对 3.8 题建立的表，用 SQL 完成下列操作。

（1）把对所有表的 INSERT 权限授予“张丽”，并允许她将此权限授予其他用户。

（2）把查询和修改 BORROWER 的权限授予用户“王伟”。