第3章 关系型数据库标准语言SQL

关系型数据库的标准语言是结构化查询语言，简称为 SQL（Structured Query Language）语言。它是一种通用的、功能强大的、集数据库的定义、操纵和控制于一体的关系型数据库语言。目前几乎所有的关系型数据库管理系统都支持 SQL 语言，即它作为国际化的标准语言广泛应用于关系型数据库管理系统之中。本章将详细介绍 SQL 语言的查询、插入、删除、修改和控制等语句的语法及其使用，重点是查询语句中查询条件的组织和表示问题。

3.1 SQL 概述

SQL 语言是用于访问数据库的标准语言，也是关系型数据库系统的管理与使用、数据库设计与编程等至关重要的数据库语言。为了学习好 SQL 语言，需要了解 SQL 语言的标准、组成及其特点。

3.1.1 SQL 简介

SQL 语言是于 1974 年由 Boyce 和 Chamberlin 提出的，IBM 公司在 1975—1979 年间研制出著名的关系型数据库管理系统原型 System R，并在该系统上实现了这种语言。1986 年 10 月，美国国家标准局 ANSI 批准了 SQL 作为关系型数据库语言的美国标准，同年发布了 SQL 标准文本（简称为 SQL-86 标准）。1987 年此标准也获得了国际标准化组织（ISO）的认可，成为国际标准语言。此后 ANSI 不断地修改和完善 SQL 标准，并于 1989 年发布了 SQL-89 标准，1992 年又发布了 SQL-92 标准（也称为 SQL2），1999 年发布了 SQL-99 标准（SQL3），2003 年发布了 SQL2003 标准，2006 年发布了 SQL2006 标准，2008 年发布了 SQL2008 标准。从 SQL-99 到 SQL2008，可以看到标准修订的周期越来越短，反映了技术的需求变化非常快。

SQL 语言成为国际标准语言以后，随着数据库技术的发展不断发展。各个数据库厂商纷纷推出了自己的数据库系统软件或与 SQL 相关的接口软件。这使大多数数据库均用 SQL 作为共同的数据存取语言和标准接口，使不同数据库系统之间的相互操作有了共同的基础。SQL 语言已经成为数据库领域中的主流核心语言。

3.1.2 SQL 数据库结构

支持 SQL 的关系型数据库管理系统同样支持关系型数据库三级模式结构,如图 3.1 所示。

1) 视图和部分基本表构成了关系型数据库的外模式

图 3.1 中外模式对应于视图和部分基本表。视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中,即数据库中只存放视图的定义而不直接存放视图对应的数据。这些数据仍存放在与视图相关的基本表中。因此,视图可以称为虚表。

用户可用 SQL 对基本表和视图进行查询或其他操作,基本表和视图一样,都是关系。在数据查询时,SQL 对基本表和视图等同对待。

2) 全体基本表构成了关系型数据库的模式

基本表是本身独立存在的表,SQL 中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,一个表可以带有若干索引,索引也存放在存储文件中。

3) 数据库的存储文件构成了关系型数据库的内模式

基本表对应的存储文件及索引文件等的逻辑结构组成了关系型数据库的内模式。

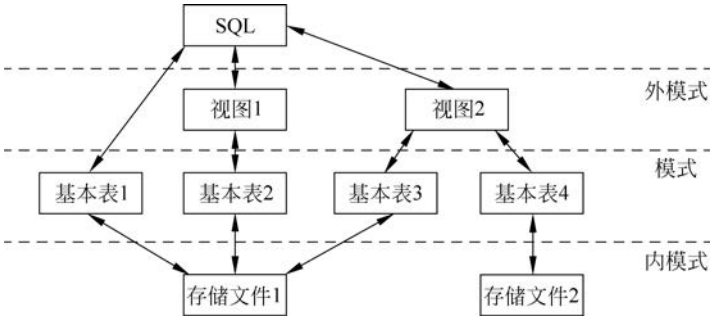

图 3.1 SQL 对关系型数据库模式的支持

3.1.3 SQL 的组成及特点

1. SQL 的组成

SQL 语言是一个通用的、功能极强的关系型数据库语言。SQL 的功能可以分为以下 3 类。

1) 数据定义

SQL 的数据定义功能是通过数据定义语言(DDL)来实现的,用来定义关系型数据库的模式、外模式和内模式,以实现对基本表、视图以及索引文件的定义、修改和删除等操作。

2) 数据操纵

SQL 的数据操纵功能是通过数据操作语言(DML)来实现的。DML 包括数据查询和数据更新两种数据操作语句。其中,数据查询语句是对数据库中的数据查询、统计、分组、排序、检索等操作,数据更新语句是数据的插入、删除和修改等操作。

3）数据控制

数据库的数据控制是指数据的安全性和完整性控制。SQL 的数据控制功能是通过数据控制语言(DCL)来实现的。SQL 通过对数据库用户的授权和回收命令来实现数据的存取控制，以保证数据库的安全性。当然 SQL 还提供了数据完整性约束条件的定义和检查机制来保障数据库的完整性。

2. SQL 的特点

SQL 之所以能够被用户和数据库行业所广泛接受，并成为国际标准，是因为它是一个综合的、功能极强的、简洁易学的数据库语言。SQL 语言具有以下特点。

1）综合统一

SQL 语言将数据定义语言 DDL、数据操纵语言 DML、数据控制语言 DCL 的功能集成于一体，语言风格统一，可独立完成数据库生命周期中的全部活动，主要包括以下 4 点。

(1) 定义关系模式，插入数据，建立数据库。

(2) 对数据库中的数据进行查询和更新。

(3) 数据库重构和维护。

(4) 数据库安全性、完整性控制，等等。

这些给数据库应用系统的开发提供了良好的数据环境。尤其是用户在数据库系统投入运行后，可以根据需求的变更而修改模式，但并不影响数据库的运行，使系统具有良好的可扩展性。

在关系模型中实体和实体之间的联系都使用关系来表示，这种数据结构的单一性带来了数据操作符号的统一性，数据的查找、插入、删除、更新等操作都只需要一种操作符号，从而简化了系统中的复杂多样化的数据信息的统一表示方式。

2）高度非过程化

SQL 语言进行数据操作，只要提出“做什么”的功能，而无须指明“怎么做”，也就是不必了解数据的存取路径。数据存取路径的选择以及 SQL 的操作过程由系统自动完成。这不仅大大减轻了用户负担，而且有利于提高数据独立性。

3）面向集合的操作方式

关系模型中实体和实体之间的联系都用关系表示，而对关系的操作特点是集合操作方式，即操作的对象和结果都是集合。关系型数据库语言 SQL 也是采用集合操作方式，不仅操作对象是元组的集合，而且查询、插入、删除、修改等操作的结果也都是元组的集合。

4）统一的语法结构，多种使用方式

SQL 既是独立的自含式语言，又是嵌入式语言。独立自含式 SQL 能够独立进行联机交互，用户只需在终端键盘上直接输入 SQL 命令就可对数据库进行操作；而作为嵌入式 SQL 语言，能够嵌入高级语言(如 C/C++、Java、C#)程序中来实现对数据库的数据存取操作。在这两种不同的使用方式下，SQL 的语法结构基本上是一致的。这种以统一的语法结构提供多种不同使用方式的特点，为使用 SQL 的程序员与用户提供了极大的灵活性与方便性。

5）语言简洁，易学易用

尽管 SQL 语言功能极强，而且又有两种使用方式，但由于设计巧妙，语言十分简洁，完

成核心功能的语句只用了 9 个动词。SQL 的命令动词及其功能如表 3.1 所示。

表 3.1 SQL 的命令动词

SQL 功 能	命 令 动 词
数据定义(数据模式定义、删除、修改)	CREATE,DROP,ALTER
数据操纵(数据查询和维护)	SELECT,INSERT,UPDATE,DELETE
数据控制(数据存取控制授权和回收)	GRANT,REVOKE

3.2 SQL 的数据定义

SQL 的数据定义包括模式定义、表定义、索引定义、视图定义和定义数据库,如表 3.2 所示。

表 3.2 SQL 的数据定义语句

操作对象	创建语句	删除语句	修改语句
模式	CREATE SCHEMA	DROP SCHEMA	—
基本表	CREATE TABLE	DROP TABLE	ALTER TABLE
索引	CREATE INDEX	DROP INDEX	—
视图	CREATE VIEW	DROP VIEW	—
数据库	CREATE DATABASE	DROP DATABASE	ALTER DATABASE

在 SQL 语句格式中,语法规定和约定符号说明如下。

1) 语句格式约定符号

SQL 语句语法中,尖括号"< >"中的内容为实际语义;中括号"[ ]"中的内容为任选项;花括号"{}"或分隔符"|"中的内容为必选项,即必选其一;[,…] 和[,…n]表示前面的项可重复多次。

2) 一般语法规定

SQL 中以英文半角逗号",”作为数据项(包括表、视图和字段或属性列)的分隔符号,其字符串常量的分界符使用英文半角单引号"'"表示。其他的标点符号也是在英文半角下表示的。

3) SQL 特殊语法规定

SQL 的关键字一般使用大写字母表示;SQL 语句的结束符为英文半角分号";"。注意,在 MS SQL Server 中可以省略分号。

另外,为了讲解 SQL 语句语法,本章中使用一个简单的学生课程数据库作为样例数据库。学生课程数据库中的 3 个关系模式分别为学生、课程、选课成绩。

学生:Student(Sno,Sname,Ssex,Sbirthday,Sdept) 其属性分别表示为:学号,姓名,性别,出生日期,系名;

课程:Course(Cno,Cname,Cpno,Ccredit) 其属性分别表示为:课程号,课程名,先行课,学分;

选课成绩:SC(Sno,Cno,Grade) 其属性分别表示为:学号,课程号,成绩。

其中,关系的主关键字加下画线表示,外关键字加波浪线表示。Cpno 是 Course 表的外关键字,Sno、Cno 一起构成 SC 表的主关键字。

其对应有 3 张表，该 3 张表的定义详见 3.2.3 节中的例 3.4。为了对该数据库的结构充分理解，且帮助后面的 SQL 语法的学习与理解，该数据库的表结构图如图 3.2 所示。

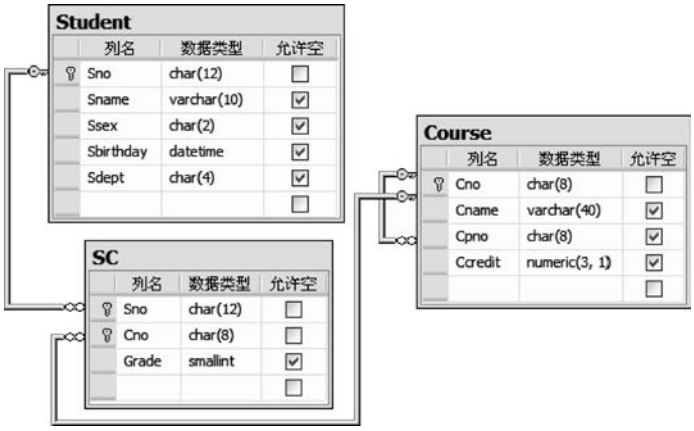

图 3.2　学生课程数据库的表结构图

各个表中的数据示例分别如表 3.3～表 3.5 所示。

表 3.3　学生表 Student 数据示例

Sno	Sname	Ssex	Sbirthday	Sdept
070107011101	卜玉	女	1989-8-1	CS
070107011102	陈博	男	1990-5-11	CS
070107011103	陈亮	男	1989-1-7	CS
080171011301	王萧	男	1988-2-2	CS
080171011304	程杏红	女	1989-9-12	CS
090301011101	蔡华兵	男	1989-8-17	EA
090301011102	陈晓骏	女	1988-2-3	EA
090301011116	罗莹莹	女	1990-5-20	EA
090171011304	陈敏	女	1989-11-10	CS
090171021317	吴伟平	男	1990-6-2	CS

注：表中 CS 代表“计算机科学系”，EA 代表“电气工程与自动化系”。

表 3.4　课程表 Course 数据示例

Cno	Cname	Cpno	Ccredit
0101001	数据结构	0107002	4
0101002	数据库系统原理	0101001	3.5
0101003	计算机网络		4
0101011	操作系统	0101001	4
0101014	软件工程	0107002	3
0101060	专业英语		2
0101066	离散数学		4
0107002	C 语言程序设计		4.5
0601001	大学英语		20
0702001	高等数学		10

表 3.5 选课成绩表 SC 示例数据

Sno	Cno	Grade
070107011101	0101002	82
070107011102	0101002	65
070107011103	0101002	70
080171011301	0101014	61
080171011304	0101014	82
090171021317	0101001	86

3.2.1 模式的创建与删除

1. 创建模式

创建模式的 SQL 语句语法：

```
CREATE SCHEMA [<模式名>] AUTHORIZATION <用户名>
      [<表定义子句> | <视图定义子句> |<授权定义子句>]
```

若没有指定<模式名>,则模式名隐含为<用户名>。调用该命令的用户必须拥有 DBA 权限,或者被授予了 CREATE SCHEMA 的权限。

【例 3.1】 给用户 Steven 定义一个学生-课程模式 StudentCourse。

```
语句 1: CREATE SCHEMA StudentCourse AUTHORIZATION Steven;
语句 2: CREATE SCHEMA AUTHORIZATION Steven;
```

解答说明：语句 2 使用的是隐含模式名形式定义的。

创建模式实际上定义了一个命名空间,在此空间中可以进一步定义该模式所包含的数据库对象,如基本表、视图、索引等。

若使用完整语法格式,即表示 CREATE SCHEMA 语句可以接受基本表定义、视图定义和授权子句,表示创建模式的同时在此模式中创建基本表、视图和定义授权。

【例 3.2】 CREATE SCHEMA StudentCourse AUTIONRIZATION Steven。

```
CREATE TABLE Student
(   Sno          CHAR(12),
    Sname        VARCHAR(10),
    Ssex         CHAR(2),
    Sbirthday    DATETIME,
    Sdept        CHAR(4)
);
```

解答说明：该语句给用户 Steven 创建一个模式 StudentCourse,并在此模式中定义了一张表 Student。

2. 删除模式

删除模式的 SQL 语句语法：

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

其中 CASCADE 和 RESTRICT 两者必选其一。级联 CASCADE 表示删除模式时将该模式

中所有的数据库对象同时删除。限制 RESTRICT 表示若该模式中存在数据库对象，则拒绝该模式的删除。

【例 3.3】 DROP SCHEMA StudentCourse CASCADE。

该语句删除模式 StudentCourse，即同时删除该模式中所有的数据库对象。

3.2.2　SQL 的数据类型

关系模型中的域是表示属性的特性或取值范围。在 SQL 中域的概念用数据类型来表示。定义基本表的各个属性列时需要指明其数据类型及长度。SQL 中提供了一些主要的数据类型，但不同的数据库系统支持的数据类型不完全相同。表 3.6 列出了 SQL 的主要数据类型。

表 3.6　SQL 的主要数据类型

类型表示		类型说明
数值型	SMALLINT	短整型
	INT 或 INTEGER	长整型
	NUMERIC(p,d)	定点数，由 p 位数字(不包括符号和小数点)组成，小数后面有 d 位数字
	FLOAT(n)	浮点数，精度至少为 n 位数字
	REAL	取决于机器精度的浮点数
	Double Precision	取决于机器精度的双精度浮点数
字符型	CHAR(n)	长度为 n 的定长字符串
	VARCHAR(n)	最大长度为 n 的变长字符串
日期时间型	DATE	日期型，格式为 YYYY-MM-DD，年月日
	TIME	时间型，格式为 HH:MM:SS，时分秒

关于属性列的数据类型选取需要根据实际情况来决定，一般要考虑属性的取值范围及参与什么运算。例如，对于学生的年龄属性，可使用字符型 CHAR(3)，但考虑年龄要参与算术运算，所以最好还是数值型的，因为字符型不能进行算术运算。又因为一个人的年龄在百岁左右，所以选用短整型或微整型作为年龄的数据类型。当然对于年龄而言，实际应用中通常使用日期型表示人的出生日期，用当前日期减去出生日期即可表示年龄。

3.2.3　基本表的创建、删除与修改

1. 定义基本表

创建了一个模式，也就是建立了一个数据库的命名空间，或称为表空间。在此空间中首先需要定义的数据库对象是该模式所包含的基本表。

SQL 语言使用 CREATE TABLE 语句定义基本表，其一般语法格式为：

```
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] [,…] );
```

创建基本表的同时通常定义与该表有关的完整性约束条件,这些约束条件存储在数据库的系统数据字典中,当用户操作表中的数据时由数据库系统自动检查该操作是否违背了完整性约束条件。若完整性约束条件涉及该表的多个属性列,则必须定义成表级约束,否则既可定义为列级也可定义为表级。关于完整性约束条件的几点说明如下。

1) 列级完整性约束条件

列级完整性约束是针对属性列赋值的限制条件。SQL 的列级完整性约束有以下 5 种。

(1) NOT NULL 或 NULL 约束。NOT NULL 约束不允许字段值为空,即非空,而 NULL 约束允许字段值为空。字段值为 NULL 的含义是该属性值"未知""不详"或"无意义"。关系的主属性必须限定为"NOT NULL",以满足实体完整性要求。

(2) PRIMARY KEY 约束。

(3) UNIQUE 约束。唯一性约束,即不允许属性列中出现重复的取值。

(4) DEFAULT 约束。默认值约束,即属性列的默认取值。

(5) CHECK 约束。检查约束,通过约束条件表达式设置属性列应满足的条件。

2) 表级完整性约束条件

表级完整性约束条件是指涉及基本表中多个字段列的限制条件。有以下 3 种表级约束。

(1) UNIQUE 约束。

(2) PRIMARY KEY 约束。

(3) FOREIGN KEY 约束。

【例 3.4】 创建学生-课程模式中的学生表 Student、课程表 Course、选课成绩表 SC。

```
CREATE TABLE Student
(   Sno         CHAR(12)  PRIMARY KEY,          /* 列级完整性约束条件,Sno 为主键 */
    Sname       VARCHAR(10)  UNIQUE,            /* 学生姓名 Sname 取唯一值 */
    Ssex        CHAR(2),                        /* 性别 */
    Sbirthday   DATETIME,                       /* 出生日期 */
    Sdept       CHAR(4)                         /* 所在系别 */
);
CREATE TABLE Course
(   Cno         CHAR(8)  PRIMARY KEY,           /* 列级完整性约束条件,Cno 为主键 */
    Cname       VARCHAR(40),                    /* 课程名称 */
    Cpno        CHAR(8),                        /* 先修课程 */
    Ccredit     NUMERIC(3,1),                   /* 学分 */
    FOREIGN KEY ( Cpno ) REFERENCES Course( Cno )
    /* 表级完整性约束条件,Cpno 为外键,被参照表是 Course,被参照列是 Cno */
);   /* 注: 此表的外键定义表示了同表之间的联系 */

CREATE TABLE SC
(   Sno     CHAR(12),                           /* 学生编号 */
    Cno     CHAR(8),                            /* 课程编号 */
    Grade   SMALLINT,                           /* 成绩 */
    PRIMARY KEY ( Sno,Cno ),
      /* 表级完整性约束条件,主键由两个属性列构成 */
    FOREIGN KEY ( Sno ) REFERENCES Student( Sno ),
      /* 表级完整性约束条件,Sno 为外键,被参照表是 Student,被参照列是 Sno */
    FOREIGN KEY ( Cno ) REFERENCES Course( Cno ),
      /* 表级完整性约束条件,Cno 为外键,被参照表是 Course,被参照列是 Cno */
);
```

2. 删除基本表

当不再需要某个基本表时，可以使用DROP TABLE语句删除它。其一般语法格式为：

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

其中，RESTRICT和CASCADE两者可选，一般默认为RESTRICT，但并不是所有的DBMS都支持。限制RESTRICT表示删除是有条件的，若该表被其他表的约束所引用（如CHECK，FOREIGN KEY等约束），或者存在视图、触发器、存储过程或函数等使用了该表，则拒绝删除该表。级联CASCADE表示删除表没有限制条件，删除表的同时删除相关的依赖对象，如视图。

【例3.5】 删除学生表Student。

```
DROP  TABLE  Student ;
```

注意：Microsoft SQL Server没有RESTRICT和CASCADE选项。Oracle没有RESTRICT选项，却有CASCADE CONSTRAINTS（级联约束）选项，表示级联删除所有参照完整性约束。即若没有使用级联约束选项，则表示默认限制删除的。

3. 修改基本表

有时由于需求的变更导致数据库的表结构的变化，则需要使用SQL语句ALTER TABLE来修改基本表。其一般语法格式为：

```
ALTER  TABLE <表名>
  [ ADD <新列名> <数据类型> [<完整性约束条件>] ]
  [ DROP <完整性约束条件> ]
  [ ALTER COLUMN <列名> <数据类型> ];
```

其中，ADD子句用于添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】 向课程表Course中增加“学时”字段列，其数据类型为短整型。

```
ALTER TABLE Course ADD Chour SMALLINT;
```

对于新添加的列，基本表中无论有无数据都一律为空值。

【例3.7】 修改选课成绩表SC，将“成绩”字段类型改为带1位的小数类型。

```
ALTER TABLE SC ALTER COLUMN Grade NUMERIC(3,1);
```

【例3.8】 修改课程表Course，添加课程名称必须取唯一值的约束条件。

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

注：如果要删除约束条件，则需要建立约束时使用命名约束。如下示例代码所示：

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

删除时则使用：

```
ALTER TABLE Course DROP UQ__Course;
```

3.2.4　索引的创建与删除

索引是基本表的目录。一个基本表可以根据应用环境的需要建立一个或多个索引，以

提供多种存取路径,加快查找速度。基本表的存储文件和索引的存储文件一起构成了数据库系统的内模式。

1. 索引的作用

1) 使用索引加快数据查询的速度

如果基本表中的数据量非常大,则其数据文件会非常大。在查询数据时,如果不使用索引,则需要将整个数据文件分块,逐个读到内存中,进行查找比较操作。而使用索引后,先将索引文件读入内存,根据索引项找到元组数据的地址,然后再根据该地址将元组数据直接读入计算机。索引文件中只含有索引项和元组地址,一般可以一次性读入内存。而且索引项是经过排序的,所以很快找到索引项及元组地址。使用索引大大减少了磁盘的 I/O 操作,从而加快查询速度。

2) 使用索引保证数据的唯一性

定义索引时可以包括定义数据唯一性的要求。这样在对相关数据进行输入或更改时,系统将进行检查来确保数据的唯一性。

3) 使用索引加快连接速度

在两个基本表进行连接操作时,系统需要在连接关系中对每一个被连接字段进行查询操作。显然,如果在连接文件的连接字段上建立索引,则可以大大提高连接操作速度。

2. 创建索引

SQL 语言中,创建索引使用 CREATE INDEX 语句,其一般语法格式为:

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

其中:

- <表名>是要创建索引的基本表的名称。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。
- 每个<列名>后面还可以用<次序>来指定索引值的排列次序,次序可选 ASC(升序)或 DESC(降序),默认值为 ASC。
- UNIQUE 表示此索引的每个索引值只对应唯一的数据记录。
- CLUSTER 表示要创建的索引是聚簇索引。所谓聚簇索引是指使索引项的排列顺序与基本表中数据的物理顺序一致的索引组织。

【例 3.9】 CREATE CLUSTER INDEX IX_Stusname ON Student(Sname)。

该语句将会在学生 Student 表的姓名 Sname 列上创建一个聚簇索引(索引名为 IX_Stusname),而且 Student 表中的数据将按照 Sname 值的升序存放。

用户可以在最经常查询的字段列上创建聚簇索引以提高查询效率。但建立聚簇索引后,在更新该索引列上的数据时,往往会导致表中记录数据的物理顺序的变更,因而代价比较大。显然一个基本表上最多只能创建一个聚簇索引,对于经常更新的列不宜创建聚簇索引。

【例 3.10】 给学生-课程数据库中的学生表 Student、课程表 Course、选课成绩表 SC 创建索引。其中 Student 表按学号升序建立唯一索引; Course 表按课程号升序建立唯一索

引；SC 表按学号升序和课程号降序建立唯一索引。

```
CREATE UNIQUE INDEX IX_Stusno ON Student ( Sno );
CREATE UNIQUE INDEX IX_Coucno ON Course ( Cno );
CREATE UNIQUE INDEX IX_SCno ON SC ( Sno ASC,Cno DESC );
```

解答说明：本例中所建立的索引，其命名使用了“IX_前缀(Index 的简写)＋表名＋索引字段名”格式，只是一种良好的命名习惯而已。另外，索引可以建立在多个字段上，并可根据需要决定是升序还是降序排列，如本例中的唯一索引 IX_SCno。

3. 删除索引

索引建立后，由系统使用和维护它，并不需要用户干预。创建索引是为了减少查询操作的时间，但如果数据的增删改非常频繁，则系统将会花费大量时间来维护索引，这样反而会降低查询效率。因此，有时需要删除一些不必要的索引。在 SQL 语言中使用 DROP INDEX 语句删除索引，其一般语法格式为：

```
DROP INDEX <索引名>;
```

【例 3.11】 删除学生 Student 表的聚簇索引 IX_Stusname。

```
DROP INDEX IX_Stusname;
```

4. 创建索引的原则

创建索引是加快数据查询的有效手段，在创建索引时用户应该遵循以下原则。

1) 索引的创建和维护由数据库管理员 DBA 和 DBMS 完成

索引由 DBA 和 DBO(表的属主，即建表人)负责创建和删除。索引由系统自动选择和维护，也就是不需要用户显式使用索引，这些工作都由 DBMS 自动完成。

2) 建议大表创建索引，小表则不必创建索引

如果表的记录数据很多，记录很长，则非常有必要创建索引。相反，对于记录比较少的基本表，创建索引的意义并不大。

3) 对于一个基本表，不要创建过多的索引

索引文件需要占用文件目录和存储空间，索引过多会造成系统负担加重。DBMS 需要维护索引，当基本表的数据增加、删除或修改时，索引文件也随之变化，以保持与基本表一致。显然，索引过多会影响数据增加、删除和修改的速度。

4) 根据查询要求建立索引

索引要根据数据查询或处理的要求来建立。对那些查询频度高、实时性要求高的数据一定要建立索引，而对于其他的数据则不应建立索引。

3.3　SQL 的数据查询

数据查询是数据库的核心操作，是根据用户的需要以一种可读的方式从数据库中提取所需数据的功能。SQL 语言提供了 SELECT 语句进行数据查询，它是 SQL 语言中功能强大的语句，也是最常见的数据操纵语句。

3.3.1 SELECT 语句的结构

SELECT 语句具有数据查询、统计、分组和排序的功能,其语句表达能力非常强大。SELECT 语句的一般语法格式为:

```
SELECT  [ALL | DISTINCT]  <目标列表达式> [,<目标列表达式>]…
FROM    <数据源(或称: 表名或视图名)> [,<表名或视图名>]…
[WHERE  <条件表达式> ]
[GROUP BY <分组列名 1>[,<分组列名 2>]… [HAVING <组选择条件表达式>] ]
[ORDER BY <排序列名 1> [ASC | DESC] [,<排序列名 2> [ASC | DESC] ]… ];
```

查询语句的功能是根据 WHERE 子句的条件表达式,从 FROM 子句所指定的数据源(基本表或视图)中找出满足条件的元组,再按照 SELECT 子句中的目标列表达式,选出元组中的属性值形成结果集。

查询语句共有 5 种子句,其中 SELECT 和 FROM 语句为必选子句,而 WHERE、GROUP BY 和 ORDER BY 子句为可选子句。

1) SELECT 子句

SELECT 子句指明查询结果集的目标列。目标列可以是数据源中的字段及相关表达式、常量或数据统计的函数表达式。若目标列中使用了两个基本表(或视图)中的相同列名,则需要在列名前加上表名或视图名限定("<表名或视图名>.<列名>")。

2) FROM 子句

FROM 子句用于指明查询的数据源,通常是基本表或视图,若存在多个的话,则用逗号分隔。有时若有一表多用的,则需要给表加上表别名以示区别。

3) WHERE 子句

WHERE 子句通过该子句中的条件表达式来描述关系中元组的选择条件,即选择满足该子句中的条件表达式的元组数据。

4) GROUP BY 子句

该子句的作用是按分组列的值对结果集分组。当 SELECT 子句的目标列表达式中有统计函数时,若也存在 GROUP BY 子句,则应进行分组统计,否则应对整个结果集进行统计。GROUP BY 子句可以带有 HAVING 短语,此时表示进一步对分组后的数据进行筛选,只有满足了组选择条件表达式的组才予以输出。

5) ORDER BY 子句

该子句的作用是对结果集进行排序。查询结果集可以按多个排序列进行排序,每个排序列后可指定是升序还是降序排序。多个排序列之间用逗号分隔。

3.3.2 单表查询

SELECT 语句可以用于简单的单表查询,也可以用于复杂的多表关联查询和嵌套查询。单表查询是指仅仅涉及一个表的查询。它是最简单最基本的一种查询语句。

1. 选择表中若干列

选择表中的全部列或部分列,或者经过计算的值,这也就是关系代数中的投影运算。

【例 3.12】 查询全体学生的详细信息。

```
SELECT  *  FROM  Student;
```

等价于：

```
SELECT  Sno, Sname, Ssex, Sbirthday, Sdept
FROM   Student;
```

查询全部列可以在 SELECT 子句后列出所有列名，如果列的显示顺序与其在表或视图中的顺序相同，则可简单写成 SELECT * 的形式。

【例 3.13】 查询全体学生的姓名、学号、所在系别。

```
SELECT  Sname, Sno, Sdept
FROM   Student;
```

<目标表达式>可以是算术表达式、字符串常量、函数等。

【例 3.14】 查询全体学生的姓名、出生日期及年龄。

```
SELECT  Sname, Sbirthday, DATEPART(year,getdate()) - DATEPART(year,Sbirthday)
FROM    Student;
```

解答说明：DATEPART 是 SQL Server 中取日期时间中的部分（如年、月、日等）；getdate()是获取当前时间的函数。本例是比较常用的一种根据出生日期来计算年龄的方法。一般系统中记录人员的信息中多数记录了出生日期这个固有特性，因为随着时间的推移，年龄会逐渐增加，这就必须每年定期修改关于年龄的数据信息，所以只记录人员的年龄是不现实的。

还可以通过指定别名来改变查询结果的列标题，尤其在含有计算表达式、常量、函数的目标列表达式中更有用。

【例 3.15】 查询全体学生的姓名、出生年份和所在系别，且用小写字母表示所有系名。

```
SELECT   Sname, '出生年份:' Birth, DATEPART(year, Sbirthday) BirthYear, LOWER
(Sdept) Department
FROM   Student;
```

解答说明：本例中使用了 LOWER 函数将字符串转换为小写，即将系名使用小写字母表示。

2. 选择表中若干元组

【例 3.16】 查询选修了课程的学生学号。

```
SELECT  Sno  FROM  SC;
```

解答说明：本例的语句执行结果可能包含有许多重复的行。若想去掉结果中的重复行，则必须使用 DISTINCT 关键词：

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

通常没有指定 DISTINCT 关键词时，则默认为 ALL，即保留结果集中的重复行。

```
SELECT  Sno  FROM  SC;
```

等价于：

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

实际应用中,多数情况下查询表中满足条件的若干元组,此时可以使用 WHERE 子句实现。

1) 比较大小

用于比较大小的运算符有:等于(=),大于(>),小于(<),大于或等于(>=),小于或等于(<=),不等于(!= 或 <>),不大于(!>),不小于(!<)。

【例 3.17】 查询计算机科学系全体学生的名单。

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

【例 3.18】 查询考试成绩有不及格的学生的学号。

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

2) 确定范围

可用来确定范围的关键词有:BETWEEN…AND…和 NOT BETWEEN…AND…。前者表示查找属性值在指定范围内的元组,后者表示查找属性值不在指定范围内的元组,其中,BETWEEN 后是范围的下限值,AND 后是范围的上限值。

【例 3.19】 查询考试成绩为 80～100 分(包括 80 和 100 分)的学生的学号。

```
SELECT  Sno
FROM    SC
WHERE  Grade  BETWEEN  80  AND  100;
```

【例 3.20】 查询考试成绩不为 80～100 分(包括 80 和 100 分)的学生的学号。

```
SELECT  Sno
FROM    SC
WHERE  Grade  NOT  BETWEEN  80  AND  100;
```

3) 确定集合

关键词 IN 可用来查找属性值属于指定集合的元组。相对立的关键词是 NOT…IN,查找属性值不属于指定集合的元组。

【例 3.21】 查询属于计算机科学系 CS、信息系 IS 的学生姓名和性别。

```
SELECT  Sname,  Ssex
FROM    Student
WHERE  Sdept  IN  ('CS', 'IS') ;
```

4) 涉及空值 NULL 的查询

当需要判别是否存在空值 NULL 的情况下,可使用 IS NULL 或 IS NOT NULL 来进行。

【例 3.22】 查询缺少成绩的学生的学号和相应的课程号。

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

【例 3.23】 查询所有有成绩的学生的学号和课程号。

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

5）模糊查询

模糊查询又称为字符匹配查询。一般语法格式为：

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

表示查找指定的属性列值与<匹配串>相匹配的元组。其中<匹配串>可以是一个完整字符串，也可含有通配符%和_(下画线)。其中：

(1) %(百分号)代表任意长度的字符串，包括长度为0。

例如，'DB%C'表示以DB开头，以C结尾的任意长度的字符串，如DBC、DBAEC、DBMSC等都属于满足匹配的字符串。

(2) _(下画线)代表任意单个字符。

例如，'DB_C'表示以DB开头，以C结尾的长度为4的任意字符串，如DBMC、DBAC等。

【例3.24】 查询学号以0701开头的学生的详细信息。

```
SELECT  *   FROM  Student
WHERE  Sno  LIKE  '0701%';
```

如果LIKE后的匹配串不含通配符，则可用 = (等于)运算符取代LIKE关键词，用!=或 <> (不等于)运算符取代NOT LIKE关键词。

【例3.25】 查询学号为“07010701110”的学生的详细信息。

```
SELECT  *   FROM  Student
WHERE  Sno  LIKE  '070107011101';
```

等价于：

```
SELECT  *   FROM  Student
WHERE  Sno = '070107011101';   /* 因为查询确定的学号,不需要使用通配符,所以直接使用=(等于) */
```

【例3.26】 查询所有姓赵的学生的姓名、学号等信息。

```
SELECT  Sname,  Sno   FROM  Student
WHERE  Sname  LIKE  '赵%';
```

【例3.27】 查询所有姓“欧阳”且全名为4个汉字的学生姓名及学号。

```
SELECT  Sname,  Sno   FROM  Student
WHERE  Sname  LIKE  '欧阳____';
```

解答说明：一个汉字占两个字符的位置，所以匹配字符串“欧阳”后面需要跟4个下画线。当然，具体在某个DBMS中，还需要看是否已经对汉字的存储处理，如果已经处理了，则使用一个下画线代表占一个汉字。

有时查询的字符串本身就含有通配符或下画线，此时就需要使用ESCAPE '<转义字符>'短语对通配符进行转义处理了。

【例3.28】 查询课程名为“DB_”开头，且倒数第3个字符为i的课程的详细信息。

```
SELECT  *   FROM  Course
```

```
WHERE  Cname  LIKE  'DB\_ % i_ _'  ESCAPE '\';
```

解答说明：本例的匹配串为'DB_%i_ _'。第一个下画线前面有转义字符\,所以此下画线被转义为普通的下画线,而不再表示单个字符的占位。而 i 后面的两个下画线的前面都没有转义字符\,所以它们仍作为通配符。思考：如果倒数的 3 个字符串为“i_ _”的话那么该如何表达?

6）多重条件查询

多重条件查询可使用逻辑运算符 AND 和 OR 来连接多个查询条件。AND 的优先级高于 OR,但可用括号改变优先级。

在例 3.19 中的 BETWEEN…AND…可用 AND 运算符和比较运算符改成多重条件查询来替换：

```
SELECT  Sno  FROM  SC
WHERE  Grade >= 80  AND  Grade <= 100;
```

在例 3.21 中的 IN 关键词实际上是多个 OR 运算符的缩写形式,因此可用 OR 运算符写成多重条件查询形式：

```
SELECT  Sname,  Ssex
FROM    Student
WHERE  Sdept = 'CS'  OR  Sdept = 'IS';
```

3. 使用 ORDER BY 子句的查询

通常查询时,需要按一定顺序显示查询结果,可以使用 ORDER BY 子句对查询结果按照一个或多个属性列的升序或降序排列显示,ASC 和 DESC 分别表示升序和降序,系统默认为升序。

【例 3.29】 查询选修了课程号为“0101003”的学生的学号及其成绩,查询结果按分数的降序排列显示。

```
SELECT  Sno, Grade  FROM  SC
WHERE  Cno = '0101003'
ORDER BY Grade DESC;
```

对应空值,若按升序排,则含空值的元组将最后显示；若按降序排,则含空值的元组将最先显示。

【例 3.30】 查询全体学生信息,查询结果按所在系别的系号升序排列,同一系中的学生按照出生日期降序排列。

```
SELECT  *  FROM  Student
ORDER BY Sdept ASC, Sbirthday DESC;
```

解答说明：因为 ASC 升序是默认排序方式,本例中的 Sdept ASC 中的 ASC 可省略不写。

4. 带聚集函数的查询

在实际应用中,常常需要对一个数据集进行统计、求和、求平均值等汇总统计操作,一般的 DBMS 都提供了聚集函数来实现这类功能。表 3.7 列出了 SQL 提供的聚集函数。

表 3.7　SQL 提供的聚集函数

聚 集 函 数	含　义
COUNT([DISTINCT|ALL] *)	计算总行数，或称为统计元组个数
COUNT([DISTINCT|ALL] <列名>)	计算一列中不同值的个数，若有 DISTINCT 则计算列的非空个数
SUM([DISTINCT|ALL] <列名>)	计算非空数字型列或表达式的总和；若有 DISTINCT，则计算不同值的总和，相同的值仅计算一次
AVG([DISTINCT|ALL] <列名>)	计算非空数字型列或表达式的平均值；若有 DISTINCT，则计算不同值的总和，相同的值仅计算一次
MAX([DISTINCT|ALL] <列名>)	计算一列值中的最大值
MIN([DISTINCT|ALL] <列名>)	计算一列值中的最小值

注：聚集函数遇到空值时，除 COUNT(*)外，都跳过空值而只处理非空值。另外，WHERE 子句中不能用聚集函数作为条件表达式的。

在使用聚集函数的查询语句中，通常需要分组进行统计。此时可使用 GROUP BY 子句和聚集函数，将数据分组后再使用聚集函数进行统计。GROUP BY 子句将查询结果按某一列或多列的值进行分组，值相等的当成一组。如果没有对查询结果分组，则聚集函数将作用于整个查询结果，如果使用了 GROUP BY 分组后聚集函数将作用于每个组，即每个组都有一个函数值。

当使用 GROUP BY 子句分组时，在 SELECT 子句的列表中，除了使用聚集函数的列外，其他各列都必须出现在 GROUP BY 子句的列表中，否则将会出错。

【例 3.31】 求各个课程号及相应的选课人数。

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

解答说明：本例语句对查询结果按 Cno 的值分组，所有具有相同 Cno 值的元组为一组，然后对每一组用聚集函数 COUNT 计算，求出该组的学生人数。

查询结果可能为：

Cno	COUNT(Sno)
0101001	22
0101002	46
0101003	38

当分组后还需要按一定条件对这些组进行筛选，最终只输出满足指定条件的组，则需要使用 HAVING 短语指定筛选条件。

【例 3.32】 查询选修了 3 门以上课程的学生学号。

```
SELECT  Sno   FROM   SC
GROUP  BY  Sno
HAVING  COUNT( * ) > 3;
```

解答说明：本例先使用 GROUP BY 子句按 Sno 进行分组，再用聚集函数 COUNT 对每一组计数。而 HAVING 短语给出了选择组的条件，只有满足条件的，即每组人数大于 3 的才会被选出来，表示学生选修了 3 门以上的课程。

HAVING 短语与 WHERE 子句的区别是：WHERE 子句作用于分组之前选择符合条

件的记录元组,而 HAVING 短语是作用于分组之后选择符合条件的分组结果,选择的是满足条件的组。

在 HAVING 短语后,使用 ORDER BY 子句,实现对各分组结果进行排序,但要求必须在 ORDER BY 子句中使用聚集函数或 GROUP BY 的分组列。一般而言,聚集函数可出现在 SELECT 子句、HAVING 短语、ORDER BY 子句中,但不能出现在 GROUP BY 子句中。

注意,如果同时使用了 WHERE 子句、GROUP BY 子句、HAVING 短语、ORDER BY 子句,则必须按先 WHERE 子句,其次是 GROUP BY 子句、HAVING 短语,最后是 ORDER BY 子句的顺序书写。HAVING 短语必须在含有 GROUP BY 子句的查询语句中使用,不能单独离开 GROUP BY 子句使用。

3.3.3 关联查询

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,则称为关联查询,也称为连接查询。在实际应用中,信息分布在不同的表或视图中,所以需要关联查询。关联查询的 WHERE 子句中用来连接两个表的条件称为连接条件,其一般格式为:

[<表名 1 或视图名 1>.]<列名 1> <比较运算符> [<表名 2 或视图名 2>.]<列名 2>

根据连接运算符的不同特点,连接运算可分为内连接和外连接。下面分别讨论。

1. 内连接

内连接是要求参与连接运算的基本表或视图满足给定的连接条件。根据连接条件的不同可分为等值连接、非等值连接、自然连接、自身连接。若连接运算符是相等(=)运算,并且参与比较运算的列的数据类型兼容,则称为等值连接;若比较运算是除了等号外的运算符,则称为非等值连接。

【例 3.33】 查询每个学生及其选修课程的情况。

```
SELECT  Student.*,  SC.*
FROM   Student,  SC
WHERE  Student.Sno = SC.Sno;
```

解答说明:学生情况存放在 Student 表中,学生选课信息存放在 SC 表中,所以本查询涉及 Student 与 SC 两个表。两个表之间通过公共属性学号 Sno 建立联系。本例中为了避免同名属性间的混淆,SELECT 子句与 WHERE 子句中的属性列名前都加上了表名前缀。若属性列名在参加连接的各表中是唯一的,则可以省略表名前缀,但通常还是习惯加上表名前缀以示区分。本例中存在 Student.Sno 与 SC.Sno 列重复的情况。

若在等值连接中把目标列中重复的属性列去掉则称为自然连接,它是一种特殊的等值连接。即要求查询结果中列不重复的等值连接。

【例 3.34】 对例 3.32 使用自然连接实现。

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

解答说明:由于两个表中只有 Sno 是相同的,所以 SELECT 子句中的属性列不同的可以省略掉表名前缀,而 Sno 前面必须加上表名前缀,本例 Sno 前使用 Student 作为前缀,也

可使用 SC 作为前缀。

SELECT 查询语句不仅支持不同表之间的连接，还支持同一表的自身的连接，称为自身连接，简称为自连接。注意自身连接与自然连接的区别，可把自连接理解为同一张表或视图的两个副本之间的连接，使用不同别名来区分副本。自身连接是等值连接和自然连接的特例。

【例 3.35】 查询每门课程的间接先修课(即先修课的先修课)。

```
SELECT  FIRST.Cno,  FIRST.Cname,  SECOND.Cpno
FROM  Course  FIRST,  Course  SECOND
WHERE  FIRST.Cpno = SECOND.Cno;
```

解答说明：在课程表 Course 中，只有每门课程的直接先修课信息，而没有先修课的先修课。要得到这个信息，必须先对一门课程找到其先修课，再按此先修课的课程号，查找它的先修课。这就是需要将 Course 表与其自身连接。所以，需要给 Course 表取两个别名表，即两个副本，一个命名为 FIRST，另一个命令为 SECOND。自身连接的条件是 FIRST 表的先修课与 SECOND 表的课程号等值连接，其结果中“SECOND 表的先修课号”为“FIRST 表的课程号”的间接先修课。

2. 外连接

在某些应用中，两张表的连接查询时，要求输出一张表的所有记录元组，而另外一张表只输出满足连接条件的记录，对没有满足条件的记录，则用空值(NULL)匹配输出，这种连接查询称为外连接。

【例 3.36】 查询所有学生的基本情况及其选课情况。

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

解答说明：本例是以 Student 表为主，列出所有学生的基本情况，加上学生的对应的选课情况信息，对于存在选课信息的则列出来，而对于不存在选课信息的则用空值(NULL)匹配显示。

注意，LEFT OUTER JOIN 表示左外连接，输出左边关系表中的所有元组，右边关系表中与左表匹配的则显示，否则使用空值(NULL)输出；RIGHT OUTER JOIN 表示右外连接，输出右边关系表的所有元组，左边关系表中与右表匹配的则显示，否则使用空值(NULL)输出。在 MS SQL Server 中还有一种全外连接，则是左外连接与右外连接所产生结果的并集。

3.3.4　嵌套查询

SQL 语言中，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中的 WHERE 子句或 HAVING 短语的条件中的查询称为嵌套查询。外层的查询称为主查询或父查询，内层的 SELECT 查询子句称为子查询。嵌套查询最多可以嵌套 255 层。按照父查询与子查询的关系，嵌套查询可分为不相关子查询和相关子查询。

在嵌套查询中，子查询只能在比较运算符的右边，而不能放在比较运算符的左边；与＝、<>、<、<＝、>、>＝等比较运算符相连的子查询必须返回非空的单值集合；如果子查询返回是空集或多值集合时，则子查询只能与 IN、NOT IN、ANY、ALL、EXISTS、

NOT EXISTS 等比较运算符相连。

1. 不相关子查询

不相关子查询是指子查询的查询条件不依赖于父查询。执行顺序是从嵌套层次最内层子查询开始执行,每个子查询在其直接外层查询处理之前执行,查询返回结果作为上层查询的查询条件,最后执行最外层的主查询。

【例 3.37】 查询选修课程号为"0101002"的学生姓名与学号。

```
SELECT  Sno,  Sname
FROM   Student
WHERE  Sno  IN  ( SELECT  Sno
                  FROM   SC
                  WHERE  Cno = '0101002');
```

解答说明:本例需要查询的学生姓名及学号在 Student 表中,而选修课程信息在 SC 表中,所以需要 Student 和 SC 这两张表参与查询。上面的 SQL 语句是带有 IN 谓词的嵌套子查询。本查询也可以使用自然连接来实现:

```
SELECT  Student.Sno,  Sname
FROM   Student,  SC
WHERE  Student.Sno = SC.Sno
   AND  SC.Cno = '0101002';
```

【例 3.38】 查询选修课程名为"数据库系统原理"的学生姓名与学号。

```
SELECT  Sname,  Sno
FROM   Student
WHERE  Sno  IN ( SELECT Sno
                 FROM  SC
                 WHERE Cno  IN ( SELECT  Cno
                                 FROM   Course
                                 WHERE  Cname = '数据库系统原理' )
                   );
```

解答说明:本查询涉及的学号、姓名和课程名 3 个属性信息。而学号和姓名存放在 Student 表中,课程名存放在 Course 表中,但 Student 表与 Course 表没有直接联系,必须通过选课表 SC 建立桥梁联系。所以本例查询涉及了 3 个关系表的查询。上面 SQL 语句是使用了 IN 谓词的不相关嵌套子查询,同样也可以用连接查询实现:

```
SELECT  Sname,  Student.Sno
FROM   Student, SC, Course
WHERE  Student.Sno = SC.Sno
   AND  SC.Cno = Course.Cno
   AND  Course.Cname = '数据库系统原理';
```

从例 3.37 和例 3.38 可以看出,查询涉及多个关系表时,使用嵌套查询逐步求解,层次清楚,易于构造,具有结构化程序设计的优点。

2. 相关子查询

相关子查询是指依赖于主查询的子查询,即子查询的条件子句含有主查询中表的有关信息。当主查询语句处理每条记录时,根据它与内层查询相关列的值来处理内层查询,若子

查询的 WHERE 子句返回值为真，则取出主查询的记录放入结果表。在含有相关子查询的嵌套查询中，通常有 EXISTS 运算符与相关子查询相连。

【例 3.39】 找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT  Sno, Cno
FROM   SC  T1
WHERE  Grade >= ( SELECT  AVG( Grade )
                  FROM   SC  T2
                  WHERE  T1.Sno = T2.Sno );
```

解答说明：T1 是表 SC 的别名，又称为元组变量，可以用来表示 SC 的一个元组。内层查询是求一个学生所有选修课程平均成绩，至于是哪个学生的平均成绩要看参数 T1.Sno 的值，而该值与父查询相关。

3. 带有 ANY 或 ALL 谓词的子查询

子查询返回单值时可用比较运算符，但返回多值时要用 ANY 或 ALL 谓词修饰符。使用 ANY 或 ALL 谓词时必须同时使用比较运算符。

ANY 谓词是检查在子查询结果集中是否满足给定的条件。如果子查询的结果集中至少有一个值满足条件，则比较运算结果为真，否则为假。

ALL 谓词是检查在子查询结果集中所有值是否都满足给定的条件。只有当结果集的所有值均满足给定的条件，则比较运算结果为真，否则为假。

通常 ANY 和 ALL 需要与＝、!＝、＞、＜、＜＝或＞＝等比较运算符配合使用。其组合意义如表 3.8 所示。

表 3.8　与 ANY 和 ALL 相关的比较运算符

运　算　符	等 效 功 能	功 能 含 义
＞ANY	＞ 最小值(MIN)	大于子查询结果中的某个值
＞ALL	＞ 最大值(MAX)	大于子查询结果中的所有值
＜ANY	＜ 最大值(MAX)	小于子查询结果中的某个值
＜ALL	＜ 最小值(MIN)	小于子查询结果中的所有值
＞＝ANY	＞＝ 最小值(MIN)	大于或等于子查询结果中的某个值
＞＝ALL	＞＝ 最大值(MAX)	大于或等于子查询结果中的所有值
＜＝ANY	＜＝ 最大值(MAX)	小于或等于子查询结果中的某个值
＜＝ALL	＜＝ 最小值(MIN)	小于或等于子查询结果中的所有值
＝ANY	IN	等于子查询结果中的某个值
＝ALL		等于子查询结果中的所有值(无实际意义)
!＝ANY 或＜＞ANY		不等于子查询结果中的某个值
!＝ALL 或＜＞ALL	NOT IN	不等于子查询结果中的任何一个值

注意，表 3.8 中聚集函数 MAX 和 MIN 必须在子查询中使用。

【例 3.40】 查询其他系中比计算机系某一学生年龄小的学生姓名和年龄。

```
SELECT  Sname, Sage
FROM   Student
WHERE  Sage  < ANY  ( SELECT  Sage
                      FROM   Student
                      WHERE  Sdept = 'CS')
   AND  Sdept  <> 'CS';
```

解答说明：注意最后的 Sdept <> 'CS'是父查询中的条件，表示从其他系别中查找。本例也可以使用聚集函数来实现：

```
SELECT  Sname, Sage
FROM  Student
WHERE  Sage  <  ( SELECT  MAX(Sage)
                  FROM  Student
                  WHERE  Sdept = 'CS')
   AND  Sdept <> 'CS';
```

事实上，使用聚集函数实现子查询通常比直接用 ANY 或 ALL 查询效率要高些。

【例 3.41】 查询其他系中比计算机系所有学生年龄都小的学生姓名和年龄。

```
SELECT  Sname, Sage
FROM   Student
WHERE  Sage  <ALL  ( SELECT  Sage
                     FROM   Student
                     WHERE  Sdept = 'CS')
   AND  Sdept <> 'CS';
```

4. 带有 EXISTS 谓词的子查询

带有 EXISTS 谓词的子查询不返回任何数据，只产生逻辑真或逻辑假。

例 3.37 也可以使用带 EXISTS 谓词的子查询实现。

```
SELECT  Sno,  Sname
FROM   Student
WHERE  EXISTS  ( SELECT  *
                 FROM   SC
                 WHERE  SC.Sno = Student.Sno  AND  Cno = '0101002');
```

解答说明：由 EXISTS 引出的子查询，其目标列表达式通常都用 *，因为这种子查询只返回真值或假值，给出具体列名无意义。

【例 3.42】 查询选修了全部课程的学生姓名。

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

解答说明：本例涉及 3 张表 Student、Course 和 SC。SQL 中没有全称量词，但可以将其转换为等价的存在量词的形式，即查询这样的学生姓名，没有一门课程是他不选修的。

3.4 SQL 的数据更新

数据更新操作有 3 种，即数据的增加、删除、修改操作，对应的 SQL 数据更新语句为：INSERT 语句、DELETE 语句和 UPDATE 语句。更新操作是以查询操作为基础，所以

SQL 数据更新语句中基本上都带有查询子句。

3.4.1　数据的插入

SQL 的数据插入语句有两种使用形式：一种是使用常量，一次仅插入一个元组；另一种是插入子查询的结果，一次可以插入多个元组。

1. 插入单个元组

使用常量插入单个元组的 INSERT 语句的格式为：

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

该语句的功能是将新元组插入指定表中，新元组的<属性列 1>的值为<常量 1>，<属性列 2>的值为<常量 2>…。如果 INTO 子句中有属性列选项，则没有出现在 INTO 子句中的属性列将取空值。但需注意，在表定义时对已经指定为非空 NOT NULL 的属性列不能取空值，否则会出错。如果 INTO 子句没有指明任何属性列名，则新插入的单个元组必须在每个属性列上均有值。

【例 3.43】 向学生表 Student 中插入一个新学生记录，该学生信息为：学号为“070107011101”，姓名“卜玉”，性别“女”，出生日期为“1989 年 8 月 1 日”，所在系别“CS”为计算机科学系。

```
INSERT  INTO  Student (Sno,Sname,Ssex,Sbirthday,Sdept)
VALUES ('070107011101','卜玉','女','1989-8-1','CS');
```

本例中 INTO 子句指出了表名 Student，并指出了新增的记录在哪些属性上需要赋值，属性的顺序可以与该表 Student 的建立的顺序不一样。VALUES 子句对应各属性赋值，其中字符串需要使用英文单引号括起来。另外，对于日期型字段数据也需按照日期格式书写且使用英文单引号括起来。

由于学生表 Student 的字段顺序依次为：Sno、Sname、Ssex、Sbirthday、Sdept，所以上述语句还可以写成如下形式：

```
INSERT  INTO  Student
VALUES ('070107011101','卜玉','女','1989-8-1','CS');
```

【例 3.44】 插入一条选课记录(学号：'070107011101'，课程号：'C01'，成绩不详)。

```
INSERT  INTO  SC (Sno,Cno)
VALUES ('070107011101','C01');
```

或者

```
INSERT  INTO  SC
VALUES ('070107011101','C01',NULL);
```

2. 插入子查询的结果集

子查询不仅可以嵌套在 SELECT 语句中，以构造父查询的条件，也可以嵌套在 INSERT 语句中用以批量插入数据。当插入的数据需要查询才能得到时，可使用插入子查

询的结果集作为批量数据输入到基本表中。

插入子查询结果集的 INSERT 语句的格式为：

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

【例 3.45】 求每门课程的平均成绩，并把结果存入数据库中。

首先新建立一张表，列名包括课程编号和平均成绩。

```
CREATE TABLE SC_AVG
(  Cno          char(8)  PRIAMRY KEY,
   GradeAvg     numeric(3,1) );
```

然后对课程表 SC 按课程编号求平均成绩，并将课程编号和平均成绩插入该表中。

```
INSERT  INTO  SC_AVG
SELECT  Cno,  AVG(Grade)
FROM   SC
GROUP  BY  Cno;
```

3.4.2 数据的删除

数据删除语句的一般格式为：

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

DELETE 语句的功能是从指定表中删除满足 WHERE 条件的所有元组。如果省略了 WHERE 子句，则表示删除表中全部元组。DELETE 语句删除的是表中的数据，而不是表的定义，即使表中的数据全部被删除，表的定义仍在数据库中。

【例 3.46】 删除学号为“070107011102”的学生记录。

```
DELETE  FROM  Student
WHERE  Sno = '070107011102';
```

【例 3.47】 删除所有学生的选课记录。

```
DELETE  FROM  SC;
```

【例 3.48】 删除计算机系所有学生的选课记录。

```
DELETE  FROM  SC
WHERE  'CS' = ( SELECT  Sdept  FROM  Student
                WHERE  Student.Sno = SC.Sno );
```

解答说明：注意在编写代码时，此例中子查询必须放在比较运算符之后。

3.4.3 数据的修改

SQL 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。

【例 3.49】　将学生学号为“070107011102”的学生的选修课程号为“0101002”的成绩增加 5 分。

```
UPDATE  SC
SET  Grade = Grade + 5
WHERE  Sno = '070107011102'  AND  Cno = '0101002';
```

【例 3.50】　将计算机系全体学生的成绩置零。

```
UPDATE  SC
SET  Grade = 0
WHERE  'CS'  =  ( SELECT  Sdept  FROM  Student
                  WHERE  Student.Sno = SC.Sno );
```

3.5　视图

视图是从一个或几个基本表(或视图)中选定某些记录或列而导出的特殊类型的表。视图本身并不存储数据,数据仍存储在原来的基本表中,视图数据是虚拟的,视图只是提供了一种访问基本表中数据的方法。视图是一个虚表,数据库只存放视图的定义。当视图创建后,用户可以像基本表一样对视图进行数据查询,在某些特殊情况下,还可以对视图进行更新、删除和插入数据操作。

数据库设计时,使用视图的主要优点有:第一,使用视图增加了数据安全性,因为可以限制用户直接存取基本表的某些列或记录;第二,使用视图可以屏蔽数据的复杂性,因为通过视图可得到多个基本表经过计算后的数据。

3.5.1　视图的创建与删除

1. 创建视图

SQL 语言中使用 CREATE VIEW 命令来创建视图,其一般语法格式为:

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

其中,视图名的命名规则与基本表的命名规则相同。<SQL 子查询语句>可以是任意复杂的 SELECT 语句,但通常不允许含有 ORDER BY 子句和 DISTINCT 短语。

WITH CHECK OPTION 表示对视图进行 UPDATE、INSERT 和 DELETE 操作时要保证更新、插入或删除的行满足<SQL 子查询语句>中的条件表达式的条件。

视图的属性列名可包括多达 1024 个,可以全部省略或者全部指定。若没有指定属性列名,则该视图的列名隐含由<SQL 查询语句>中所指定的列名决定。通常下列 3 种情况下必须明确指定组成视图的所有列名。

(1) 某个列不是单纯的属性名,而是算术表达式或系统函数的计算结果,则必须给计算结果指定别名。

(2) 当多表连接时,选出了多个同名列作为视图的字段时,则需要加上表名或指定别名。

(3) 需要在视图中为某个列启用新的、更合适的名字。

视图不能为列指定数据类型和长度,而是默认为数据源(即基本表)的类型和长度。

【例 3.51】 建立计算机系的学生视图。

```
CREATE VIEW V_Student_CS
AS
    SELECT Sno, Sname, Sbirthday
    FROM Student
    WHERE Sdept = 'CS';
```

解答说明:本例中省略了视图的列名,隐含列名由子查询中 SELECT 子句中的 Sno,Sname,Sbirthday 组成。

【例 3.52】 建立计算机系的学生视图,并要求进行更新、插入或删除操作时仍需保证该视图只有计算机系的学生。

```
CREATE VIEW V_Student_CS
AS
    SELECT Sno, Sname, Sbirthday
    FROM Student
    WHERE Sdept = 'CS'
    WITH CHECK OPTION;
```

解答说明:由于定义视图时使用了 WITH CHECK OPTION 子句,则以后对该视图进行插入、更新和删除操作时,RDBMS 会自动加上 Sdept = 'CS'的条件。

【例 3.53】 建立计算机系选修了数据库系统原理(编号为"0101002")课程的学生视图。

```
CREATE VIEW V_Student_CS1 ( Sno, Sname, Grade )
AS
    SELECT Student.Sno, Sname, Grade
    FROM Student, SC
    WHERE Sdept = 'CS'
      AND Student.Sno = SC.Sno
      AND SC.Cno = '0101002';
```

解答说明:本例中建立的视图是建立在多个基本表上。由于视图 V_Student_CS1 的属性列中包含了 Student 表和 SC 表的同名列 Sno,所以必须在视图名后明确指定视图的各属性列名。

【例 3.54】 建立计算机系选修了数据库系统原理(编号为"0101002")课程且成绩在 90 以上的学生视图。

```
CREATE VIEW V_Student_CS2
AS
    SELECT Sno, Sname, Grade
    FROM  V_Student_CS1
    WHERE Grade >= 90;
```

解答说明:本例中建立的视图是在已存在的视图 V_Student_CS1 上建立的。

【例 3.55】 使用视图来定义学生的学号及其平均成绩的信息。

```
CREATE VIEW V_Grade ( Sno, Gavg )
AS
```

```
SELECT Sno, AVG( Grade )
FROM   SC
GROUP BY Sno;
```

解答说明：本例建立的视图定义中使用了聚集函数 AVG 求平均值，并使用了 GROUP BY 子句来进行分组统计，这种视图称为分组视图。由于使用了 AVG 函数，所以定义视图时需要明确指定视图属性列名。

2. 删除视图

删除视图的语法格式为：

```
DROP  VIEW <视图名>[ CASCADE ];
```

删除视图仅仅是从系统中的数据字典中删除了视图的定义，并没有删除数据，不会影响基本表中的数据。只有视图的拥有者或有 DBA 权限的用户才能删除。若该视图被其他视图引用，则删除后引用视图将不能正常使用。此时可以使用 CASCADE 来级联删除本视图和导出引用的所有视图。

基本表删除后，由该基本表导出的所有视图没有被删除，但均无法使用了。故还需要删除不能使用的无意义的视图。

【例 3.56】 删除视图 V_Student_CS1。

```
DROP  VIEW  V_Student_CS1;
```

解答说明：由于 V_Student_CS1 视图上还导出了 V_Student_CS2 视图，所以执行此删除视图语句会被拒绝。如果确定要删除，则使用级联删除语句。

```
DROP  VIEW  V_Student_CS1 CASCADE;      /* 删除 V_Student_CS1 视图及导出的所有视图 */
```

3.5.2　视图的查询

定义视图后，使用视图查询时就可以与基本表一样使用了。

【例 3.57】 查询选修了“0101002”课程的计算机系学生的学号和姓名。

```
SELECT Sno, Sname
FROM  V_Student_CS1;
```

或者

```
SELECT V_Student_CS.Sno, Sname
FROM  V_Student_CS, SC
WHERE  V_Student_CS.Sno = SC.Sno
   AND  SC.Cno = '0101002';
```

解答说明：本例中前面使用了 V_Student_CS1 视图，该视图本来就是满足题目要求的视图。当然如果查询的是选修其他课程的学生信息，则需要使用第二种查询语句。

3.5.3　视图的更新

视图的更新是指通过视图来插入、删除、修改数据。由于视图是不存放数据的虚表，数据是来自其他基本表，因此，对视图的更新最终是转换为对基本表的更新。SQL 语言标准

规定：只能对直接定义在一个基本表上的视图进行插入、删除、修改等操作，对定义在多个基本表或其他视图上的视图，DBMS 不允许进行更新操作。

为了防止用户通过视图对数据进行增加、删除、修改时，无意地对不属于视图范围内的基本表数据进行操作，在定义视图时应尽量加上 WITH CHECK OPTION 子句。这样在视图上增删改数据时，RDBMS 会检查视图定义中的条件，若不满足条件，则拒绝执行该操作。

【例 3.58】 将计算机系的学生视图 V_Student_CS 中学号为“070107011101”的学生姓名改为“黄燕”。

```
UPDATE V_Student_CS
SET Sname = '黄燕'
WHERE Sno = '070107011101';
```

解答说明：本题使用视图来更新 Student 基本表中的数据，转换后的等价更新语句为：

```
UPDATE Student
SET Sname = '黄燕'
WHERE Sno = '070107011101'  AND  Sdept = 'CS';
```

【例 3.59】 向计算机系的学生视图 V_Student_CS 中插入一个新生的学生记录信息，该学生学号为“100107011120”，姓名改为“赵鑫”，出生日期为“1990-10-10”。

```
INSERT  INTO  V_Student_CS
VALUES ( '100107011120', '赵鑫', '1990-10-10' );
```

解答说明：本题使用视图来给 Student 表中增加新数据，转换后的等价插入语句为：

```
INSERT  INTO  Student
VALUES ('100107011120', '赵鑫', '1990-10-10', 'CS' ); 系统自动将系别名'CS'放入 VALUES 子句中。
```

另外，尽管视图数据只来源于一个基本表，但如果 SELECT 语句含有 GROUP BY、DISTINCT 或聚集函数等，除可以执行删除操作外，不能进行插入或修改操作。如果视图中包含由表达式计算的列，则也不允许进行修改操作。如果视图中没有包含基本表的所有非空列，则不能对该视图进行插入操作。如果视图定义中有嵌套查询，且内层查询的 FROM 子句中涉及的表也是导出该视图的基本表，则此视图也是不允许更新的。一个不允许更新的视图上定义的视图也是不允许更新的。

3.5.4 视图的作用

视图最终是定义在基本表上的，对视图的操作最终也就是要转换为对基本表的操作，所以使用视图作更新操作要受到很多的限制，而对查询操作没有限制。归结起来，视图的作用有以下 5 个方面。

(1) 视图能够简化用户的操作。

(2) 视图使用户能够以多种角度看待同一数据。

(3) 视图对重构数据库提供了一定程度的逻辑独立性。

(4) 视图能够对机密数据提供一定程度的安全保护。

(5) 适当地利用视图可以更清晰地表达查询。

3.6　嵌入式 SQL

SQL 语言有两种方式，分别是独立式 SQL 和嵌入式 SQL。独立式 SQL 作为独立语言以交互式方式使用，而嵌入式 SQL 是嵌入某种高级语言中混合使用。嵌入 SQL 的高级语言，如 C、C++、Java 等，称为宿主语言，或简称为主语言。嵌入式 SQL 和主语言相配合设计应用程序，充分利用了主语言的过程性结构和专业应用功能强的优点，并保留了 SQL 的强大的数据库管理功能。

3.6.1　嵌入式 SQL 的处理过程

关系型数据库管理系统 RDBMS 对嵌入式 SQL 的处理一般采用预编译方法，即由 RDBMS 的预处理程序对源程序进行扫描，识别出嵌入式 SQL 语句，把它们转换成主语言函数调用，以使主语言编译程序能够识别它们，然后由主语言的编译程序将纯的主语言程序编译成目标码。嵌入式 SQL 的基本处理过程如图 3.3 所示。

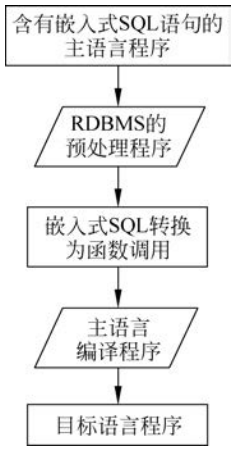

图 3.3　嵌入式 SQL 的基本处理过程

3.6.2　嵌入式 SQL 的使用规定

为了正确、合理地使用嵌入式 SQL，必须注意以下使用规定问题。

1. 区分 SQL 语句与主语言语句

在嵌入式 SQL 中，为了区分 SQL 语句与主语言语句，所有 SQL 语句都必须加上前缀 EXEC SQL，并用分号(;)结束当成一个程序片段“EXEC SQL < SQL 语句>;”。

嵌入式 SQL 的使用时一般使用分号结束，这个规定主要是针对 PL/1 和 C 语言，其他语言不尽相同。

2. 数据库的工作单元与程序工作单元之间的通信

在含有嵌入式 SQL 语句的应用程序中，SQL 语句负责操纵数据库，主语言语句负责控制程序流程和其他功能。因此，数据库的工作单元与程序工作单元存在如何通信的问题。

(1) 向主语言传递 SQL 语句的执行状态信息，即 SQL 语句的当前工作状态和运行环境数据需要反馈给应用程序。

SQL 将其执行信息送到 SQL 通信区(SQL Communication Area，SQLCA)中，应用程序从 SQLCA 中取出这些状态信息，并据此信息来控制该执行的语句。

SQLCA 是一个数据结构，在应用程序中使用 EXEC SQL INCLUDE SQLCA 来定义。SQLCA 中有一个变量 SQLCODE 存放每次执行 SQL 语句后返回的代码。应用程序每执行完一条 SQL 语句后都应该测试一下 SQLCODE 的值，以了解该 SQL 语句执行的情况并

做相应处理。如果 SQLCODE 等于预先定义的常量 SUCCESS,则表示 SQL 语句执行成功,否则 SQL 语句执行失败,在 SQLCODE 中存放着错误代码。

(2) 主语言使用主变量(Host Variable)向 SQL 语句提供参数。

主变量是主语言程序变量的简称,可分为输入主变量和输出主变量。输入主变量由应用程序提供值,SQL 语句引用;输出主变量由 SQL 语句提供值,返回给应用程序。

一个主变量可以附带一个任选的指示变量(Indicator Variable),用来指示主变量的值或条件。指示变量是一个整型变量,它可以指示输入主变量是否为空,可以检测输出主变量是否为空值,值是否被截断。

所有主变量和指示变量都必须在 SQL 说明语句"BEGIN DECLARE SECTION"与"END DECLARE SECTION"之间进行说明。通常为了与数据库对象名区别,SQL 语句中的主变量名和指示变量前要加冒号(:)作为标志。在 SQL 语句之外,主变量和指示变量均可以直接引用而不必加冒号。

(3) 使用游标解决 SQL 集合查询操作与主语言变量单记录操作的不一致的矛盾。

SQL 语言与主语言具有不同的数据处理方式。SQL 语言是面向集合的,一条 SQL 语句可以产生或处理多条记录。而主语言是面向记录的,一组主变量一次只能存放一条记录。所以仅使用主变量并不能满足 SQL 语句向应用程序输出数据的要求。为此,嵌入式 SQL 引入游标的概念,用游标来协调这两种不同的处理方式。游标是系统为用户提供的一个数据缓冲区,存放 SQL 语句的执行结果,每个游标区都有一个名字。用户可以使用游标获取逐条记录,并赋给主变量,交给主语言处理。

3.6.3 嵌入式 SQL 的使用技术

如何使用嵌入式 SQL 来编程呢? 下面主要介绍嵌入式 SQL 的使用技术。

1. 建立和关闭数据库连接

嵌入式 SQL 程序要访问数据库必须先连接数据库。关系型数据库管理系统 RDBMS 根据用户信息对连接请求进行合法性验证,只有通过合法身份验证才能建立一个可用的合法连接。

1) 建立数据库连接

建立连接的嵌入式 SQL 语句为:

```
EXEC SQL CONNECT TO <数据库服务器> [ AS <数据库连接名>] [ USER <用户名>];
```

其中:

<数据库服务器>是要连接的数据库服务器,使用服务器标识串来表示,如< dbname >@< hostname >:< port >。

<数据库连接名>是一个有效的标识符,主要用来识别一个程序内同时建立的多个连接,当整个程序内只有一个连接则可以省略不指定连接名。

2) 关闭数据库连接

当某个连接上的所有数据库操作完成后,应用程序应该主动释放所占用的连接资源。关闭数据库连接的嵌入式 SQL 语句是:

```
EXEC SQL DISCONNECT [<数据库连接名>];
```

2. 不用游标的 SQL 语句

不需要使用游标的语句包括：说明性语句、数据定义语句、数据控制语句、查询结果为单记录的查询语句、非 CURRENT 形式的增删改语句。

1）查询结果为单记录的查询语句

这种不用游标的单记录查询语句的一般格式为：

```
EXEC SQL SELECT [ ALL | DISTINCT ] <目标表达式> [, …]
        INTO  <主变量>  [ <指示变量> ] [, …]
        FROM  <表名或视图名> [, …]
        [WHERE  <条件表达式> ];
```

使用单记录的查询语句需要注意以下 3 点。

（1）INTO 子句、WHERE 子句和 HAVING 短语的条件表达式中均可以使用主变量。

（2）查询结果为空值的处理。若某列为空值，则对应的指示变量为负值，则认为对应的主变量为 NULL。指示变量只能用于 INTO 子句中。

（3）若查询结果实际上并不是单条记录，而是多条记录，则程序出错，RDBMS 会在 SQLCA 中返回错误信息；若查询结果为空集时，则 SQLCODE 值为 100。

【例 3.60】 查询学号为主变量 givensno 的值、课程号为主变量 givencno 的值的学生选修课程的信息。

```
EXEC SQL SELECT Sno,Cno,Grade
        INTO  : Hsno,: Hcno,: Hgrade: Gradeid        /* 指示变量 Gradeid */
        FROM  SC
        WHERE  Sno = : givensno  AND  Cno = : givencno;
```

解答说明：本题在 WHERE 子句中使用主变量 givensno 和 givencno，并在 INTO 子句中使用了 3 个输出主变量 Hsno、Hcno 和 Hgrade，其中 Hgrade 后加了一个指示变量 Gradeid。若 Gradeid 为负值，则不论 Hgrade 为何值，均认为该学生成绩为空值。

2）不用游标的数据删除语句

【例 3.61】 删除学号由主变量 givensno 决定的学生记录。

```
EXEC SQL DELECT  FROM  Student
        WHERE  Sno = : givensno;
```

3）不用游标的数据更新语句

在 UPDATE 语句中，SET 子句和 WHERE 子句中都可以使用主变量，SET 子句还可以使用指示变量。

【例 3.62】 修改 givensno 指定的学生选修课程编号为“0101001”的成绩，成绩由 newgrade 指定。

```
EXEC SQL UPDATE SC
        SET Grade = : newgrade
        WHERE  Sno = : givensno  AND  Cno = '0101001';
```

【例 3.63】 将计算机系全体学生的所有选修的课程成绩均提高主变量 addgrade 指定的分值。

```
EXEC SQL UPDATE SC
        SET Grade = Grade + : addgrade
        WHERE  Sno  IN  (
            SELECT  Sno  FROM  Student
            WHERE  Sdept = 'CS');
```

【例 3.64】 将计算机系全体学生的所有选修的课程成绩均清空。

```
gradeid = -1
EXEC SQL UPDATE SC
        SET Grade = Grade + : addgrade: gradeid
        WHERE  Sno  IN  (
        SELECT  Sno  FROM  Student
        WHERE  Sdept = 'CS');
```

解答说明：该题中变量 gradeid 为指示变量，在主语言中不需要冒号。由于其赋值为－1，故无论主变量 addgrade 为何值，该语句都会将计科系全体学生的选修课程的成绩置空。该语句此时可以用下面语句代替。

```
EXEC SQL UPDATE SC
        SET Grade = NULL
        WHERE  Sno  IN  (
            SELECT  Sno  FROM  Student
            WHERE  Sdept = 'CS');
```

4）不用游标的数据插入语句

INSERT 语句的 VALUES 子句可以使用主变量和指示变量，当需要插入空值时，可以把指示变量置为负值。

【例 3.65】 某学生新选修了某门课程，将有关记录插入 SC 表中。假设插入的学号、课程号分别存放在主变量 stdno、courseno 中。

```
gradeid = -1
EXEC SQL INSERT INTO
        SC ( Sno,Cno,Grade)
        VALUES (: stdno,: courseno,: gr: gradeid);
```

或

```
EXEC SQL INSERT INTO
        SC ( Sno,Cno,Grade)
        VALUES (: stdno,: courseno,NULL);
```

解答说明：由于该学生刚选修课程，成绩应当为空，所以赋值指示变量为负值，或者直接将成绩赋值为 NULL。

3. 使用游标的 SQL 语句

必须使用游标的 SQL 语句有：查询结果为多条记录的查询语句、CURRENT 形式的 UPDATE 和 DELETE 语句。

1）查询结果为多条记录的查询语句

通常多数情况下 SELECT 查询语句的结果是多条记录，因此需要使用游标，将多条记录一次一条地交给主程序来处理，从而把对集合的操作转换为对单个记录的处理。使用游

标的有 4 个步骤。

(1) 说明游标。使用 DECLARE 语句说明定义游标。

```
EXEC SQL DECLARE <游标名> CURSOR FOR < SELECT 语句>;
```

(2) 打开游标。使用 OPEN 语句打开已经定义的游标。

```
EXEC SQL OPEN <游标名>;
```

打开游标实际上是执行相应的 SELECT 语句,查询结果取到缓冲区中。此时游标处于活动状态,指针指向查询结果的首条记录。

(3) 推进游标并取出当前记录值。

```
EXEC SQL FETCH <游标名>
    INTO <主变量> [<指示变量>] [,<主变量> [<指示变量>] ]…;
```

其中,主变量必须与 SELECT 语句中的目标列表达式一一对应。

通过循环执行 FETCH 语句逐条取出结果集中的行分别存放到主变量中,然后由主程序进行处理。

(4) 关闭游标。用 CLOSE 语句关闭游标,释放结果集占用的缓冲区及其资源。

```
EXEC SQL CLOSE <游标名>;
```

关闭游标后,就不再和原来的查询结果集有联系了,但被关闭的游标还可以再次被打开。

2) CURRENT 形式的 UPDATE 和 DELETE 语句

当 UPDATE 语句和 DELETE 语句是集合操作时,若只想修改或删除其中单个记录,则需要使用带游标的 SELECT 语句查出所以满足条件的记录,从中进一步找出需要修改或删除的记录,然后再使用 CURRENT 形式的 UPDATE 和 DELETE 语句来修改或删除。也就是在 UPDATE 语句和 DELETE 语句中需要使用子句“WHERE　CURRENT OF <游标名>”来表示修改或删除的是最近一次取出的记录,即游标指针所指向的记录。

说明:当游标定义中的 SELECT 语句带有 UNION 或 ORDER BY 子句时,或者该 SELECT 语句相当于定义了一个不可更新的视图时,则不能使用 CURRENT 形式的 UPDATE 和 DELETE 语句。

3.7　动态 SQL 语句

嵌入式 SQL 语句中使用的主变量、查询字段、条件等都是固定不变的,属于静态 SQL 语句。但有时在某些应用程序中需要在执行时才能确定要提交执行的 SQL 语句和查询条件。此时需要使用动态 SQL 语句来解决。

动态 SQL 也就是在程序运行过程中动态生成 SQL 语句。动态 SQL 支持动态组装 SQL 语句和动态参数两种形式。

3.7.1　使用 SQL 语句主变量

程序主变量包含的内容是整个 SQL 语句的内容,这样的程序主变量称为 SQL 语句主

变量。SQL 语句主变量在程序执行期间可以设定不同的 SQL 语句,然后可立即执行。

【例 3.66】 创建基本表 TEMP。

```
EXEC SQL BEGIN DECLARE SECTION;
Const char * stmt = "CREATE TABLE temp(id  int); ";   /* SQL 语句主变量 */
EXEC SQL END DECLARE SECTION;
…
EXEC SQL EXECUTE IMMEDIATE :stmt;                     /* 执行动态 SQL 语句 */
```

3.7.2 使用动态参数

动态参数是 SQL 语句中的可变元素,使用参数符号问号(?)表示该位置上的数据在运行时设定。动态参数的输入不是编译时完成绑定的,而是通过准备 SQL 语句(Prepare)和执行时绑定数据或主变量来完成的。使用动态参数的步骤如下:

(1) 声明 SQL 语句主变量。

变量的 SQL 内容包含动态参数问号(?)。

(2) 准备 SQL 语句(PREPARE)。

PREPARE 将分析含主变量的 SQL 语句内容,建立语句中包含的动态参数的内部描述符,并用<语句名>标识它们的整体。

```
EXEC SQL PREPARE <语句名> FROM < SQL 语句主变量>;
```

(3) 执行准备好的语句(EXECUTE)。

EXECUTE 将 SQL 语句中分析出的动态参数和主变量或数据常量绑定成为语句的输入或输出变量。

```
EXEC SQL EXECUTE <语句名> [ INTO <主变量表>] [ USING <主变量或常量>];
```

【例 3.67】 向 TEMP 表中插入元组。

```
EXEC SQL BEGIN DECLARE SECTION;
Const char * stmt = "INSERT INTO temp VALUES ( ? ); "; /* 声明 SQL 主变量 */
EXEC SQL END DECLARE SECTION;
…
EXEC SQL PREPARE mystmt FROM :stmt;                   /* 准备语句 */
EXEC SQL EXECUTE mystmt USING 100;                    /* 执行语句 */
EXEC SQL EXECUTE mystmt USING 200;                    /* 执行语句 */
```

3.8 存储过程

SQL-99 标准中提出了 SQL-Invoked Routines 的概念。SQL-Invoked Routines 可以分为存储过程和函数两大类。下面介绍存储过程。PL/SQL(Procedural Language/SQL)是编写数据库存储过程的一种过程语言。它结合了过程化语言的流程控制能力和 SQL 的数据操作能力,是对 SQL 语言的过程化扩展。

3.8.1 存储过程的概念

存储过程是一种存储在数据库上的,执行某种功能的预编译 SQL 批处理语句。它是一

种封装重复任务操作的方法，支持用户提供的参数变量，具有强大的编程能力，可以被反复调用，运行速度较快。存储过程具有许多优点。

（1）加快程序执行速度，运行效率高。

存储过程不像解释执行的SQL语句那样在提出操作请求时才进行语句的语法分析和优化工作，由于存储过程在第一次被执行后，其执行规划就存储在高速缓存中，以后的操作只需从高速缓存中调用编译好的存储过程的二进制代码执行即可。因此，存储过程可以加快执行速度，提供运行效率与系统的性能。

（2）减少了客户端和服务器端之间的通信量。

这是使用存储过程的非常重要的原因之一。客户端的应用程序只要通过网络向服务器发出存储过程的名字和参数，即可让RDBMS执行许多条的SQL语句，并执行数据处理。只有最终处理结果才返回到客户端。这样极大地减轻了网络的负担，提供了系统的响应速度。

（3）允许程序模块化设计。

对应同一任务操作，只需创建一次存储过程并将其存储在数据库中，以后可以在不同程序中任意调用。相同的逻辑处理结果保证了数据修改的一致性。另外，模块化设计使存储过程独立于程序源代码，既利于集中控制，又能单独修改而无须修改其他程序代码，提高了程序的可用性。

3.8.2　存储过程的操作

存储过程的操作包括创建、重命名、执行和删除4种。

1. 创建存储过程

存储过程包括过程首部和过程体两部分。其创建语句的基本语法格式为：

```
CREATE PROCEDURE <过程名>([参数1,参数2,…])        /*存储过程首部*/
AS                                                /*存储过程体,描述该存储过程的操作*/
<PL/SQL语句块>;
```

过程名：是数据库中的合法标识符。

参数列表：用名字来标识调用时给出的参数值，必须指定值的参数类型。存储过程的参数可以定义输入参数（INPUT）、输出参数（OUTPUT）、输入/输出参数。默认为输入参数。

过程体：是一个<PL/SQL块>，它包含声明部分和可执行部分。

【例3.68】 利用存储过程计算某系学生选修了“数据库系统原理”（编号为“0101002”）课程的平均分和选修人数。

```
CREATE PROCEDURE pr_CourseAvg
    @Dept       VARCHAR(4),
    @GradeAvg   DECIMAL(4,1)   OUTPUT,
    @StudentNum   INT   OUTPUT
AS
BEGIN
    DECLARE   @TotalGrade    INT
    SELECT   @StudentNum = count(*)   FROM   SC
```

```
    WHERE  Sno  IN  ( SELECT  Sno  FROM  Student  WHERE  Sdept = @Dept )
      AND  Cno = '0101002'

    SELECT  @TotalGrade = SUM( isnull(Grade,0) )  FROM  SC
    WHERE  Sno  IN  ( SELECT  Sno  FROM  Student  WHERE  Sdept = @Dept )
      AND  Cno = '0101002'

    IF @StudentNum is null OR @StudentNum = 0 THEN
      SET @GradeAvg = NULL
    ELSE
      SELECT @GradeAvg = round( @TotalGrade / @StudentNum, 1)
    END IF
END
```

解答说明：注意存储过程中使用的变量在 MS SQL Server 中必须加上@前缀，而在 Oracle 数据库中不需要。本例的存储过程体中声明定义的变量@TotalGrade 的数据类型必须与选课表 SC 中的成绩字段 Grade 对应一致。另外输出参数@GradeAvg 平均值需要定义为带小数的数据类型。过程体中使用的 isnull 函数用于判断其值是否为 NULL，若是，则用 0 计算；round 函数用于四舍五入并保留指定小数位数。

2. 重命名存储过程

存储过程的重命名语句语法格式为：

```
ALTER PROCEDURE <旧过程名> RENAME TO <新过程名>;
```

3. 执行存储过程

存储过程的执行语句语法格式为：

```
CALL/PERFORM/EXECUTE PROCEDURE <过程名> ([参数 1,参数 2,…]);
```

通常习惯在 MS SQL Server 中使用 EXECUTE 或简写 EXEC 来执行存储过程。大多数 DBMS 中数据库服务器支持存储过程的嵌套调用。

【例 3.69】 调用存储过程计算机系学生选修了数据库系统原理(编号为“0101002”)课程的平均分和选修人数。

```
DECLARE  @num  INT,  @avg  DECIMAL(4,1)
EXEC  pr_CourseAvg  'CS',  @avg  OUTPUT,  @num  OUTPUT
PRINT  @avg, @num
```

4. 删除存储过程

删除存储过程的语法格式为：

```
DROP PROCEDURE <过程名>();
```

在 MS SQL Server 中，可以不需要带参数与括号。

【例 3.70】 删除存储过程 pr_CourseAvg。

```
DROP PROCEDURE pr_CourseAvg
```

小结

本章详细讲解了关系型数据库标准语言 SQL 的数据定义、数据查询、数据更新等语句的语法和使用，以及视图、存储过程、嵌入式 SQL 和动态 SQL 语句。其中，数据定义、数据查询、数据更新、视图等内容是本章重点，也是关系型数据库 SQL 语言编程重点，而数据查询语句的灵活运用更是学习关系型数据库标准语言 SQL 的难点。

(1) SQL 数据定义：SQL 的数据定义包括模式定义、表定义、索引定义、视图定义和创建数据库。本章中只对表、索引、视图的定义重点讲解，而模式定义与创建数据库的语法则针对不同数据库而不同。

(2) SQL 数据查询：SELECT 查询语句是 SQL 语言中功能强大的语句，也是最常见的数据操纵语句。其中，关联查询与嵌套查询较难掌握，也是非常灵活的数据查询，需要多练多操作。另外，对于关联查询中的内连接和外连接，学习时不仅要掌握内连接的使用，更需要弄清外连接的含义，因为在实际应用中存在外连接的情况。

(3) SQL 数据更新：SQL 的数据更新包括 INSERT 语句、DELETE 语句和 UPDATE 语句。使用时，插入语句 INSERT 需要注意表中的非空字段，删除语句 DELETE 与更新语句 UPDATE 需要注意是否有条件，也即删除或更新的数据范围。

(4) 视图：视图是从一个或几个基本表(或视图)导出的虚表。它本身不存储数据，数据仍存储在原来的基本表中。视图创建后，可以使用视图进行数据查询，在某些特殊情况下，还可以对视图进行更新、删除和插入数据操作。但通常利用视图进行数据查询而很少更改基本表中的数据。由于使用视图增加了数据安全性，并且可以屏蔽数据的复杂性，因此在数据库设计时对一些复杂的数据表达可以采用视图。

(5) 存储过程：存储过程是一种执行某种功能的预编译 SQL 批处理语句。它是一种封装重复任务操作的方法，支持用户提供的参数变量，具有强大的编程能力。它可被反复调用，运行速度较快。在熟练掌握灵活使用数据查询语句的基础上，可以较容易地学好存储过程。

(6) 嵌入式 SQL：嵌入式 SQL 是嵌入某种高级语言(如 C、C++、Java 等，俗称主语言)中混合使用。学会嵌入式 SQL 和主语言相配合设计应用程序，充分利用主语言的过程性结构和专业应用功能强的优点，而且保留了 SQL 的强大的数据库管理功能。在一些涉及底层编程的情况下，可能需要使用嵌入式 SQL 语言。

(7) 动态 SQL 语句：嵌入式 SQL 语句属于静态 SQL 语句，其中使用的主变量、查询字段、条件等都是固定不变的。动态 SQL 语句是在程序运行过程中动态生成的 SQL 语句。在某些应用程序中需要在执行时才能确定要提交执行的 SQL 语句和查询条件，此时需要使用动态 SQL 语句来解决。

习题 3

3.1　简答题

1. 试述 SQL 的组成及特点。

2. 什么是基本表？什么是视图？两者的区别和联系是什么？

3. 试述索引的功能作用及创建索引的原则。

4. 试述视图有哪些作用。

5. 哪类视图是可以更新的？哪类视图是不可更新的？各举一例说明。

6. 在嵌入式 SQL 语言中，如何区分 SQL 语句与主语言语句？

7. 在嵌入式 SQL 语言中，如何解决数据库工作单元与源程序工作单元之间的通信？

8. 什么是存储过程？存储过程有哪些优点？

9. 什么是游标？试述存储过程中使用游标的步骤。

3.2 设计编程题

1. 图书出版社管理数据库中有两个基本表：

图书(书号，书名，作者编号，出版社，出版日期)；

作者(作者编号，作者姓名，年龄，地址)；

试用 SQL 语句写出以下查询：检索年龄低于作者平均年龄的所有作者的姓名、书名和出版社。

2. 设某数据库中有 3 种关系：

员工表 EMP(Eno，Ename，age，sex，Ecity)，其属性分别表示员工编号、姓名、年龄、性别和籍贯；

公司表 COMP(Cno，Cname，city)，其属性分别表示公司编号、公司名称、公司所在城市；

工作表 WORKS(Eno，Cno，salary)，其属性分别表示员工编号、公司编号和工资；

试用 SQL 语句写出下列操作。

(1) 用“CREATE DATABASE”创建一个存放上述 3 个表的数据库，数据库名称为“WorkDB”。

(2) 用 SQL 语句定义上述 3 个表，并需要指出主关键字和外关键字。

(3) 检索超过 50 岁的男性职工的编号和姓名。

(4) 假设每个职工只能在一个公司工作，检索工资超过 2500 元的男性员工编号和姓名。

(5) 检索“美联公司”中低于本公司平均工资的员工编号和姓名。

(6) 在每个公司中为 50 岁以上的员工加薪 200 元。

(7) 删除年龄大于 60 岁的员工信息。

(8) 创建一个“美联公司”中关于女性员工信息的视图，属性包括(Eno，Ename，Cno，Cname，salary)。

3. 设职工_社团数据库中有 3 个基本表：

职工(职工号，姓名，性别，出生日期)；

社会团体(社团编号，社团名称，负责人，活动地点)；

参加社团(职工号，社团编号，参加日期)；

其中：

(1) 职工表的主关键字为职工号。

(2) 社会团体表的主关键字为社团编号；外关键字为负责人，被参照表是职工表，参照对象为职工号。

(3) 参加社团表的职工号和社团编号为主关键字；职工号为外关键字，其参照对象是职工表中的职工号；社团编号也是外关键字，其参照对象是社会团体表中的社团编号。

试用 SQL 语句表达下列操作。

① 定义职工表、社会团体表、参加社团表，并说明主关键字和参照关系。

② 建立两个视图：

社团负责人(社团编号，社团名称，负责人职工号，负责人姓名，负责人性别)；

参加人员情况(职工号，姓名，社团编号，社团名称，参加日期)；

③ 查找参加歌舞表演队或篮球队的职工号和姓名。

④ 查找没有参加任何社会团体的职工情况。

⑤ 查找参加了全部社会团体的职工情况。

⑥ 查找参加了职工号为“2012”的职工所参加的全部社会团体的职工号。

⑦ 统计每个社会团体的参加人数。

⑧ 查找参加人数超过 100 人的社会团体的名称和负责人。

⑨ 统计参加人数最多的社会团体的名称和参加人数。

⑩ 把对社会团体和参加社团两张表的数据查看、插入和删除数据的权力赋予用户李晨，并允许其再将此权力授予其他用户。

4. 假设工程_零件数据库中有 4 个基本表：

供应商(供应商代码，供应商名称，所在城市，联系电话)；

工程(工程代码，工程名，负责人，预算)；

零件(零件代码，零件名，规格，产地，颜色)；

供应零件(供应商代码，工程代码，零件代码，数量)；

试用 SQL 语句完成下列操作。

(1) 找出武汉市供应商的姓名和电话。

(2) 查找预算在 50000～100000 元的工程信息，并将结果按预算降序排列。

(3) 找出使用供应商 S1 所供零件的工程代码。

(4) 找出工程项目 J2 使用的各种零件名称及其数量。

(5) 找出上海厂商供应的所有零件代码。

(6) 找出使用了上海生产的零件的工程代码。

(7) 找出没有使用武汉生产的零件的工程代码。

(8) 把全部红色零件的颜色改成绿色。

(9) 将由供应商 S5 供给工程代码为 J3 的零件 P8 改为由 S3 供应，并做其他必要的修改。

(10) 从供应商关系中删除 S2 的记录，并从供应零件关系中删除相应的记录。

5. 请为三建工程项目建立一个供应情况的视图，包括供应商代码、零件代码、供应数量。针对该视图完成下列查询：

(1) 找出三建工程项目使用的各种零件代码及数量；

(2) 找出供应商 S1 的供应情况。

6. 利用 MS SQL Server 数据库编写一个获取数据库服务器当前日期时间的存储过程，该存储过程是为了方便在某系统或多个系统间共享使用，使应用系统时间统一同步。提示：MS SQL Server 中获取系统时间的函数为 getdate()。