第3章 关系数据库标准语言

SQL(structured query language,结构化查询语言)是关系数据库的标准语言,它以关系代数运算与谓词演算为基础,主要用于对关系数据库进行各种操作。虽然用关系代数表示的查询很简洁,但是,以关系代数为基础设计的数据库语言,用户不但要说明需要什么数据,还要说明获得这些数据的过程。因此,这种语言实际上是一种过程性语言。

在数据库的应用中,需要一种对用户更加友好的查询语言,使用这种语言时,用户只要说明需要的数据,如何获得这些数据则不必由用户说明,而是由系统来实现。SQL 就是这样一种语言,也是一种非过程性语言。SQL 的功能非常强大,除了能够提供数据库的查询功能外,还可以提供数据定义功能、数据操纵功能以及数据控制功能等。

数据库语言本身不是计算完备的语言,不能用来独立编制应用程序。目前常用方法是将数据库语言嵌入到一种高级程序设计语言中(如 C 语言)。这种高级程序设计语言称为数据库语言的宿主语言。数据库语言主要用于访问数据库,而宿主语言主要用来处理数据。两种语言是独立发展起来的,由此带来许多不匹配的问题。例如,宿主语言一般是过程性语言,而数据库语言往往是非过程语言;两者所支持的数据类型一般也不一致;数据库语言常常是面向集合的语言,即执行一条语句可以一次性地提供所需的所有元组,而这些元组,作为程序设计语言的输入,只能逐个地对程序设计语言有关变量赋值而分别地处理。为了解决这些问题,SQL 提供了另一种使用方式,即将 SQL 嵌入到高级语言中使用。这种方式下使用的 SQL 称为嵌入式 SQL。

本章首先介绍 SQL 的发展过程,然后详细讨论 SQL 的基本功能,最后讨论嵌入式 SQL。

3.1 SQL 的发展过程

SQL 有许多版本,最早的版本是由当时 IBM 的 San Jose 研究室提出的。该语言最初叫 Sequel,是 20 世纪 70 年代作为 System 项目的一部分实现的。发展到现在,该语言的名字已经变为 SQL,并明确地确定了其作为标准关系数据库语言的地位。

1986 年 10 月,美国国家标准局(ANSI)的数据库委员会颁布了第一个标准 SQL。

1987 年 6 月,国际标准化组织(ISO)将其采纳为关系数据库语言的国际标准,即 SQL86。

1989 年 4 月,ISO 在 SQL86 的基础上增强了完整性特征,形成了 SQL89。

1992 年,ISO 又推出了新标准 SQL92,即 SQL2。

1999 年,ANSI 在 SQL2 的基础上扩充了面向对象的功能,并制定通过了 SQL99 标准,即 SQL3。

目前,许多关系数据库产品,如 SQL Server、Oracle、DB2、Sybase 等都使用 SQL 作为数据的查询语言。

3.2 SQL 的组成和基本结构

SQL 按其功能主要由以下几个部分组成。

① DDL(data definition language,数据定义语言),用于定义、撤销和修改关系模式,

如表、视图、索引。

② QL(query language,查询语言),用于查询数据库中的数据。

③ DML(data manipulation language,数据操纵语言),用于在关系模式中增加、删除以及修改元组数据。

④ DCL(data control language,数据控制语言),用于数据访问权限的控制。

SQL 的基本结构主要由 **SELECT**、**FROM** 和 **WHERE** 三个子句组成。一个典型的 SQL 查询语句具有以下形式:

```
SELECT  A1,A2,…,An
FROM    r1,r2,…,rm
WHERE   P;
```

其中:

SELECT 子句对应关系代数中的投影运算,用来列出查询结果的内容。每个 A_i 表示一个目标列表达式,该表达式可以是属性列、聚集函数和常量的任意算数运算组成的运算公式。

FROM 子句对应关系代数中的笛卡儿积运算,用来列出查询中需扫描的关系。每个 r_i 表示一个关系。

WHERE 子句对应关系代数中的选择谓词,用来列出查询时必须满足的条件。P 是谓词条件。

上面这个查询语句等价于下面的关系代数表达式:

$$\Pi_{A_1,A_2,\cdots,A_n}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m))$$

即,SQL 先构造 **FROM** 子句中关系的笛卡儿积,根据 **WHERE** 子句中的谓词 P 进行关系代数的选择运算,然后将结果投影到 **SELECT** 子句中的属性列上。

下面将详细讨论 SQL 语言的基本功能。

3.3 数据定义语言

关系是关系数据库的基本组成单位,也称为表。在 SQL 中,表分为以下两种。

(1) 基表

基表是一个实表,其数据显式地存储在数据库的表中。

(2) 视图

视图是一个虚表。数据库中仅存放视图的定义,而不存放视图对应的数据,这些数据仍存放在原来的基表中。基表中数据发生变化时,从视图中查询出的数据也随之发生变化。视图又分为普通视图和临时视图。

普通视图仅有逻辑定义,可根据其定义由其他基表导出,但不作为一个表显式地存储在数据库中。视图可像基表一样,参与各种数据库操作。

对于较复杂的查询,可将查询中相对独立部分作为查询的中间结果,定义为临时视图。临时视图在功能上与普通视图一样,但仅用于附在临时视图定义后的查询语句中。

该查询语句结束后,临时视图随之自行消失。

本书主要介绍普通视图。

3.3.1 基表模式的定义

SQL 中的数据定义功能包括定义表和定义索引。

基表模式的定义主要使用 SQL 中 **CREATE TABLE** 语句,其一般形式为

```
CREATE TABLE <表名>
    (<属性列名> <数据类型>[ <属性级完整性约束条件> ]
    [,<属性列名> <数据类型>[ <属性级完整性约束条件>] ] …
    [,<表级完整性约束条件> ]);
```

其中:

——<属性级完整性约束条件>:包括该属性列是否为空,默认值。

——<表级完整性约束条件>:包括引用完整性检查以及唯一性检查。

注意:*方括号内为任选项。*

一般 SQL 都支持的数据类型如表 3.1 所示。

表 3.1 SQL 支持的数据类型

数据类型	说明符	备 注
整数	INT	字长 32 位
短整数	SMALLINT	字长 16 位
十进制数	DEC(m,n)	m 为总十进制位数(不包括小数点),n 为小数点后的十进制位数
浮点数	FLOAT	一般指双精度浮点数,字长 64 位
定长字符串	CHAR(n)	按固定长度 n 存储字符串。如果实际字符串长小于 n,后面填空格符;如果实际字符串长大于 n,则报错
变长字符串	VARCHAR(n)	按实际字符串长度存储,但字符串长不得超过 n,否则报错
日期	DATE	格式为"yyyymmdd",yyyy 表示年份,范围为 0001～9999;mm 表示月份,范围为 1～12;dd 表示日,范围为 1～31
时间	TIME	格式为"hhmmss",hh 表示时,范围为 0～24;mm 表示分,ss 表示秒,范围都是 0～59
时标	TIMESTAMP	格式为"yyyymmddhhmmssnnnnnn",nnnnnn 表示微秒,范围为 0～999999,其他符号的意义同上

常用的完整性约束有如下。

① 主键约束:**PRIMARY KEY**(定义该属性列为主键)。

② 唯一性约束:**UNIQUE**(不能取相同值但允许多个空值)。

③ 非空值约束:**NOT NULL**(该属性列的值不能为空)。

④ 引用(参照)完整性约束。

```
FOREIGN KEY <属性列> REFERENCES <表名>(<属性列>)
```

例 3.3.1 创建一个学生表 STUDENT，它由学号 SNO、姓名 SNAME、性别 SEX、出生日期 BDATE、所在系 DEPT 五个属性构成。其中，学号 SNO 不能为空，值是唯一的。

利用 SQL 中 **CREATE TABLE** 语句定义如下：

```
CREATE TABLE STUDENT
  (SNO      CHAR(8),
   SNAME    VARCHAR(10),
   SEX      CHAR(2),
   BDATE    DATE,
   DEPT     CHAR(10),
   PRIMARY  KEY(SNO));
```

其中：

PRIMARY KEY(SNO)表示属性 SNO 为学生关系的主键。主键要求非空且值是唯一的。

SNO CHAR(8)表示属性 SNO 的数据类型为固定长度字符串，即关系中每个元组在该属性上的值都是具有 8 个字符长度的字符串。

SNAME VARCHAR(10)表示属性 SNAME 的数据类型为可变长度字符串，即关系中每个元组在该属性上的值是最多可以有 10 个字符长度的字符串。

BDATE DATE 表示属性 BDATE 的数据类型为日期型，即关系中每个元组在该属性上的值均用日期型数据表示。

DEPT CHAR(10)表示属性 DEPT 的数据类型为固定长字符串，即关系中每个元组在该属性上的值都是具有 10 个字符长度的字符串。

例 3.3.2 创建一个课程表 COURSE，它由课程号 CNO、课程名 CNAME、学时数 LHOUR、学分数 CREDIT、开设学期 SEMESTER 五个属性组成。

利用 SQL 中 CREATE TABLE 语句定义如下：

```
CREATE TABLE COURSE
  (CNO       CHAR(6),
   CNAME     VARCHAR(20),
   LHOUR     SMALLINT,
   CREDIT    INT,
   SEMESTER  CHAR(2),
   PRIMARY KEY (CNO));
```

其中：

PRIMARY KEY(CNO)表示属性 CNO 为该课程关系的主键。主键要求非空且值是唯一的。

例 3.3.3 创建一个选课表 SC，它由学号 SNO、课程号 CNO、成绩 GRADE 三个属性组成。

利用 SQL 中 **CREATE TABLE** 语句定义如下：

```
CREATE TABLE SC
 (SNO CHAR (8) NOT NULL,
 CNO CHAR (6)  NOT NULL,
 GRADE DEC (4,1)  DEFAULT NULL,
 PRIMARY KEY (SNO, CNO),
 FOREIGN KEY (SNO)
    REFERENCES STUDENT,
    ON DELETE CASCADE,
FOREIGN KEY (CNO)
    REFERENCES COURSE,
    ON DELETE RESTRICT);
```

其中：

PRIMARY KEY(SNO,CNO)表示属性(SNO,CNO)为选课关系的主键。主键要求非空且值是唯一的。

FOREIGN KEY(SNO)**REFERENCES** STUDENT 表示 SNO 也是外键,引用的是学生关系中的主键 SNO。

ON DELETE 是引用完整性的任选项,**ON DELETE CASCADE** 表示当主表中删除了某一主键时,基表中引用此主键的行也随之被删除。

FOREIGN KEY(CNO)**REFERENCES** COURSE 表示 CNO 也是外键,引用的是课程关系中的主键 CNO。

ON DELETE RESTRICT 表示凡是被基表所引用的主键,不得被删除。

3.3.2 基表模式的修改

如果需要对已定义的基表模式进行修改,可以利用 SQL 提供的修改基表模式的命令。SQL 提供 7 种修改基表模式的命令。

1. 增加列

```
ALTER TABLE <表名>
     ADD <属性列><类型>;
```

例 3.3.4 向学生 STUDENT 表中增加“入学时间”属性列 SCOME,其数据类型为日期型。

```
ALTER TABLE STUDENT ADD SCOME DATETIME DEFAULT GETDATE();
```

注意：不论基表中原来是否已有数据,新增加的属性列一律为当前日期。

2. 删除基表

```
DROP TABLE <表名>;
```

例 3.3.5 删除学生 STUDENT 表。

```
DROP TABLE STUDENT;
```

删除基表后，表里的数据、表上的索引都会被删除，表上的视图往往仍然保留，但无法引用。

删除基表时，系统会从数据目录中删去有关该基表及其索引的描述。

注意：SQL 未提供删除列的命令。若要删除列，只有另定义一个新表，并将原来表中要保留的属性列的内容复制到新表中，然后删除原表。最后还要用重命名命令把新表改为原表名。

3. 补充定义主键

如果原表以前未定义主键，需要时可利用此命令补充定义主键。

```
ALTER TABLE <表名>
     ADD PRIMARY KEY (<属性列表>);
```

4. 撤销主键定义

一般情况下，一个基表如果已定义了主键，则系统会在主键上自动建立索引。当插入新元组时，系统会进行主键唯一性检查，这样，当进行大量的插入操作时，必影响系统效率。为了提高效率，可利用撤销主键命令暂时撤销主键，完成插入操作后再补充定义主键。

```
ALTER TABLE <表名>
     DROP PRIMARY KEY;
```

5. 补充定义外键

如果原表以前未定义外键，需要时可利用此命令补充定义外键。

```
ALTER TABLE <表名 - 1 >
       ADD AFOREIGN KEY [<外键名>](<属性列表>)
          REFERENCES <表名 - 2 >
          [ON DELETE { RESTRICT|CASCADE|SET NULL}];
```

花括号表示任选三项之一，其中：

RESTRICT，凡被基表引用的主键，不能删除；该选项通常为默认项。

CASCADE，如主表中删除了某一主键，则基表中引用此主键的元组也被删除。

SET NULL，该列应无 NOT NULL 说明。

6. 撤销外键定义

由于定义外键后，须作引用完整性检查，这会影响系统性能，因此，SQL 提供了撤销外键的命令，必要时可暂时撤销。

```
ALTER TABLE <表名>
```

```
DROP <外键名>;
```

7. 定义和撤销别名

需要时,可以给基表或视图定义别名,定义别名的命令为

```
CREATE SYNONYM <标识符>
        FOR <表名>|<视图名>;
```

当然,需要时也可以撤销基表或视图别名,撤销别名的命令为

```
DROP SYNONYM <标识符>;
```

3.3.3 索引的建立与撤销

在基表上建立索引是加快表的查询速度的有效手段。用户可以根据应用环境的需要,在基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

建立索引使用 **CREATE INDENX** 命令,其一般形式为

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

其中:

——<索引名>为要建立的索引文件的名字。

——<表名>为要建索引的基表名字。

——索引可以建立在该表的一个属性列或多个属性列上,各属性列之间用逗号分隔。

——<次序>表示索引值的排列次序,**ASC** 表示升序,**DESC** 表示降序,默认值为 **ASC**。

——**UNIQUE** 表示此索引的每一个索引值只对应唯一的数据记录。

——**CLUSTER** 表示要建立的索引是簇集索引(关于簇集的概念在数据库物理设计一章再详细介绍)。

例 3.3.6 对选课文件 SC 按学号 SNO 属性降序、按课程号 CNO 属性升序建立索引文件,索引文件名为 SC_INDEX。

```
CREATE UNIQUE SC_INDEX ON SC (SNO DESC,CNO ASC);
```

2. 撤销索引

撤销索引可使用 **DROP INDENX** 命令,其一般形式为

```
DROP INDENX <索引名>
```

命令执行后,系统会从数据目录中删去有关该索引的描述。

注意：SQL 标准中没有定义对索引的修改功能，可以采用删除后重新定义索引的方式实现索引的修改功能。

例 3.3.7 撤销索引文件 SC_INDEX。

```
DROP INDENX SC_INDEX;
```

注意：若在 SQL Server 中使用 DROP INDEX，必须以“表名.索引名”的形式同时给出表名和索引名，即

```
DROP INDEX SC.SC_INDEX;
```

3.4 查询语言

数据库的查询是数据库操作的核心，所以，SQL 中的查询语句也是使用最多的。SQL 查询语句又分为基本查询语句和复杂查询语句，利用基本查询语句可以编写一些简单的查询，而利用复杂查询语句可以编写出一些较为复杂的查询。本节将详细讨论如何编写查询语句。

3.4.1 基本 SQL 查询语句

基本查询语句主要包括三个子句：**SELECT**、**FROM** 和 **WHERE**，其一般形式为

```
SELECT [ALL|DISTINCT] <目标列表达式>[别名][,<目标列表达式>[别名]]…
FROM <表名或视图名>[,<表名或视图名> ] …
[ WHERE <条件表达式> ]
[ GROUP BY <属性列 1> [ HAVING <条件表达式> ] ]
[ ORDER BY <属性列 2> [ ASC|DESC ] ];
```

其中：

——**SELECT** 后面的目标列表达式可以是属性列、聚集函数和常量的任意算术运算组成的运算公式。别名是为目标列表达式起的别名。**ALL** 表示查询结果中不消除重复元组，**DISTINCT** 表示查询结果中要消除重复元组。

——**FROM** 后面给出查询内容所涉及的基表或视图。

——**WHERE** 后面给出查询的条件，是任选项。

——**GROUP BY** 表示将查询结果根据属性列 1 的值进行分组，**HAVING** 是分组必须满足的附加条件，是任选项。

——**ORDER BY** 表示将查询结果根据属性列 2 进行排序，**ASC** 表示升序排序，**DESC** 表示降序排序，是任选项。**ORDER BY** 子句的次序必须位于 **GROUP BY** 子句的后面。

整个句子的含义是：根据 **WHERE** 子句的条件表达式，从基表（或视图）中找出满足条件的元组，按 **SELECT** 子句中的查询内容，从元组中选出相应的属性值形成查询结果。

如果有 **GROUP BY** 子句,则查询结果按指定的属性列进行分组,有 **HAVING** 时,满足条件的组才给予输出。如果有 **ORDER BY** 子句,则查询结果要根据指定的属性列按升序或降序排序。

假设学生 STUDENT 表、课程 COURSE 表以及选课 SC 表中的数据如表 3.2～表 3.4 所示。

表 3.2　STUDENT 表

SNO	SNAME	SEX	BDATE	DEPT
09920201	刘　芳	女	1974-3-12	计算机系
09920202	张晓晨	男	1974-1-24	计算机系
09920203	王文选	男	1973-11-15	计算机系
09920301	张　玲	女	1974-8-19	计算机系
09920302	李莉平	女	1975-9-24	计算机系

表 3.3　COURSE 表

CNO	CNAME	LHOUR	CREDIT	SEMESTER
CS-110	计算机概论	32	2	秋
CS-201	数据结构	80	5	秋
CS-221	数字电路	64	4	春
EE-122	继电保护	48	3	秋
EE-201	电力系统分析	80	5	春

表 3.4　SC 表

SNO	CNO	GRADE
09920201	CS-110	95
09920201	CS-201	90
09920202	CS-110	85
09920202	EE-201	80
09920203	CS-110	82
09920203	CS-201	75
09920203	EE-122	

例 3.4.1　查询所有学生详细情况。

分析:这是一个简单查询,查询的内容是学生的详细情况。查询内容所涉及的基表只有 STUDENT 一张表。因为是查询所有学生情况,所以不需要给出查询条件。该查询的 SQL 语句如下:

```
SELECT *
FROM STUDENT;
```

其中,"*"号表示查询基表中的所有内容(即所有属性列的值)。

查询结果为

SNO	SNAME	SEX	BDATE	DEPT
09920201	刘　芳	女	1974-3-12	计算机系
09920202	张晓晨	男	1974-1-24	计算机系
09920203	王文选	男	1973-11-15	计算机系
09920301	张　玲	女	1974-8-19	计算机系
09920302	李莉平	女	1975-9-24	计算机系

例 3.4.2　查询学生姓名和出生年月。

分析：本题查询是一个简单查询，查询内容是所有学生的姓名(SNAME)和出生年月(BDATE)。查询内容所涉及的基表也只有 STUDENT 一张表。该查询的 SQL 语句如下：

```
SELECT SNAME,BDATE
FROM STUDENT;
```

查询结果为

SNAME	BDATE
刘　芳	1974-3-12
张晓晨	1974-1-24
王文选	1973-11-15
张　玲	1974-8-19
李莉平	1973-9-24

例 3.4.3　查询选修了课程的学生学号。

分析：本题查询是个简单查询，查询的内容是选修了课程的学生学号(SNO)。由于一个学生可能选修多门课程，所以，查询结果中可能包含重复元组。在写 SQL 语句时要注意消除重复元组。该查询的 SQL 语句如下：

```
SELECT DISTINCT SNO
FROM SC;
```

其中，**DISTINCT** 表示消除查询结果中的重复元组。

注意：在关系代数投影运算(Π)的定义中直接去掉了结果中的重复元组，在 SQL 中必须在 **SELECT** 子句中用 **DISTINCT** 明确指定才能去掉重复元组。

查询结果为

SNO
09920201
09920202
09920203

例 3.4.4　查询所有女生的学号和姓名。

分析：本题查询是个简单查询，查询内容是所有女生的学号(SNO)和姓名

(SNAME),查询内容涉及的基表只有学生 STUDENT 表,查询条件是从学生表中选择出性别为“女”的那些元组。所以,该查询的 SQL 语句如下:

```
SELECT SNO,SNAME
FROM STUDENT
WHERE SEX = '女';
```

查询结果为

SNO	SNAME
09920201	刘　芳
09920301	张　玲
09920302	李莉平

注意:WHERE 子句常用的查询条件如表 3.5 所示。

表 3.5　WHERE 子句中常用的查询条件

查询条件	谓　词
比较	=,<,>,<=,>=,<>,!=,!>,!<; NOT+上述比较符
确定范围	BETWEEN…AND…,NOT　BETWEEN…AND…
确定集合	IN,NOT IN
字符匹配	LIKE,NOT LIKE
空值	IS NULL,IS NOT NULL
多重条件	AND,OR

例 3.4.5　检索学习课程号为 CS-221 的课程的学生学号与姓名。

分析:本题的查询内容是学生的学号(SNO)和姓名(SNAME),由于学生的选课信息在 SC 表中,而学生姓名在 STUDENT 表中,所以该查询内容涉及两张基表。查询条件是查询结果中的这些学生必须选修了 CS-221 课程。该查询的 SQL 语句如下:

```
SELECT SC.SNO,SNAME
FROM STUDENT,SC
WHERE STUDENT.SNO = SC.SNO
  AND CNO = 'CS-201';
```

该语句执行时,先对 STUDENT 和 SC 做笛卡儿积操作,然后在两个基表的公共属性上选择出属性值相等且课程号为 CS-201 的那些元组。由于 SNO 在 STUDENT 和 SC 中都出现,因此引用 SNO 时需注明基表名,例如,STUDENT.SNO 表示引用的是 STUDENT 基表中的 SNO。

查询结果为

SNO	SNAME
09920201	刘　芳
09920203	王文选

例 3.4.6 查询至少选修课程号为 CS-110 和 CS-201 的课程的学生学号与姓名。

分析：本题的查询内容是学生学号(SNO)和姓名(SNAME)，查询内容涉及选课表 SC 和学生表 STUDENT。查询条件是学生必须选修了课程号为 CS-110 和 CS-201 的课程。由于 CS-110 和 CS-201 出现在同一个基表 SC 的不同元组中，为了区别，可以引入别名 X、Y，也可看成定义了两个元组变量 X、Y。在语句中应该用别名加以限定，保留字 AS 可省。该查询的 SQL 语句如下：

```
SELECT SNO, SNAME
FROM STUDENT
WHERE SNO IN
    (SELECT X.SNO
     FROM SC AS X, SC AS Y
     WHERE X.SNO = Y.SNO
       AND X.CNO = 'CS - 110'
       AND Y.CNO = 'CS - 201');
```

查询结果为

SNO	SNAME
09920201	刘　芳
09920203	王文选

例 3.4.7 查询 1973 年出生的学生名及其秋季所修课程的课程号及成绩。

分析：本题的查询内容是学生名(SNAME)、课程号(CNO)及成绩(GRADE)，由于学生名在学生表中，课程号和成绩在选课表中，课程开设时间为秋季还是春季的详细信息在课程表中，所以，查询内容涉及的基表有学生表 STUDENT、课程表 COURSE 和选课表 SC。查询条件是 1973 年出生的学生且所选课程必须是秋季开设的课程。显然，这是具有多个条件的查询，可用逻辑运算符 **AND** 和 **OR** 来联结多个查询条件。该查询的 SQL 语句如下：

```
SELECT SNAME,COURSE.CNO,GRADE
FROM STUDENT,COURSE,SC
WHERE STUDENT.SNO = SC.SNO
  AND SC.CNO = COURSE.CNO
  AND YEAR(BDATE) = 1973
  AND SEMESTER = '秋';
```

查询结果为

SNAME	CNO	GRADE
王文选	CS-110	82
王文选	CS-201	75
王文选	EE-122	

下面,根据学生表 STUDENT、课程表 COURSE 和选课表 SC 中的数据来验证查询结果。

从 STUDENT 表中可知,1973 年出生的学生如下:

SNO	SNAME	SEX	BDATE	DEPT
09920203	王文选	男	1973-11-15	计算机系

从 SC 表中可知,09920203 所选的课程如下:

SNO	CNO	GRADE
09920203	CS-110	82
09920203	CS-201	75
09920203	EE-122	

从 COURSE 表中可知,秋季开设的课程如下:

CNO	CNAME	LHOUR	CREDIT	SEMESTER
CS-110	计算机概论	32	2	秋
CS-201	数据结构	80	5	秋
EE-122	继电保护	48	3	秋

对它们进行自然连接操作,然后在 SNAME、CNO 和 GRADE 属性上投影,最后就可得到上面的查询结果。

例 3.4.8 将课程号为 CS-110 的课程根据学生的成绩进行降序排列。

分析:本题的查询内容主要是查询学生的成绩,查询结果要求降序排列。由于每个成绩应该对应学生的学号和姓名,所以,该查询内容涉及学生表 STUDENT 和选课表 SC 两张基表。查询条件是只查询 CS-110 课程的成绩。该查询的 SQL 语句如下:

```
SELECT STUDENT.SNO,SNAME,GRADE
FROM STUDENT,SC
WHERE STUDENT.SNO = SC.SNO
  AND CNO = 'CS - 110';
```

查询结果为

SNO	SNAME	GRADE
09920201	刘　芳	95
09920202	张晓晨	85
09920203	王文选	82

例 3.4.9 查询缺成绩的学生学号和课程号。

分析:本题查询涉及对空值的查询。查询内容只涉及选课表 SC,查询条件是元组在 GRADE 属性上没有值。该查询的 SQL 语句如下:

```
SELECT SNO,CNO,GRADE
FROM SC
WHERE GRADE IS NULL;
```

查询结果为

SNO	CNO	GRADE
09920203	EE-122	NULL

为了使查询结果更加清晰，可以根据情况，在 SELECT 子句后面适当增加属性列。

注意：空值是状态而不是值，所以不能写成 GRADE=**NULL** 或 **NOT** GRADE=**NULL**。

3.4.2 较复杂的 SQL 查询语句

较复杂的查询是指查询条件比较复杂的查询语句。例如，带谓词 **IN**、**BETWEEN…AND…**、**LIKE** 的查询，嵌套查询、带聚集函数的查询等。

1. 带谓词 IN 的查询

带谓词 **IN** 的查询语句格式一般为

```
(集合 1) IN (集合 2)
(集合 1) NOT IN (集合 2)
```

集合 1 与集合 2 可以是一个 SELECT 子查询，或值的集合，但它们的结构相同。

IN 操作表示：如果集合 1 中每个元素都在集合 2 内，那么其逻辑值为 true，否则为 false；

NOT IN 操作表示：如果集合 1 中某个元素不在集合 2 内，那么其逻辑值为 true，否则为 false。

例 3.4.10 查询秋季学期有一门以上课程获 90 分以上成绩的学生。

分析：本题的查询内容是学生，查询条件是这些学生必须选修了秋季的课程且成绩在 90 分以上。由于学生姓名在 STUDENT 表中，课程开设的学期信息在 COURSE 表中，而成绩信息在 SC 表中，因此，该查询涉及学生表 STUDENT、课程表 COURSE 和选课表 SC 三张基表。这种查询可有多种写法，现在用谓词 **IN** 和嵌套查询来写查询语句。所谓嵌套查询是指一个 **SELECT FROM WHERE** 查询块可以嵌套在另一个 **SELECT FROM WHERE** 查询块中。其中的每一个 **SELECT FROM WHERE** 查询块也称为子查询。SQL 中允许多层嵌套。该查询的 SQL 语句如下：

```
SELECT SNAME
FROM STUDENT
WHERE SNO IN
    (SELECT DISTINCT SNO
     FROM SC
     WHERE GRADE>=90.0
       AND CNO IN
```

```
        (SELECT CNO
         FROM COURSE
         WHERE SEMESTER = '秋'));
```

查询结果为

SNAME
刘　芳

注意：嵌套查询是由里向外处理的，即每个子查询在上一级处理之前求解。这样外层查询可以利用内层查询的结果。

例如，本题最内层子查询：

```
SELECT CNO
FROM COURSE
WHERE SEMESTER = '秋';
```

查询结果为

CNO
CS-110
CS-201
EE-122

中间层子查询：

```
SELECT DISTINCT SNO
FROM SC
WHERE GRADE >= 90.0
  AND CNO IN
      (集合 2);
```

此时集合 2 中的值为(CS-110,CS-201,EE-122)，所以中间层子查询的查询结果为

SNO
09920201

最外层子查询：

```
SELECT SNAME
FROM STUDENT
WHERE SNO IN
      (集合 2);
```

此时集合 2 中的值为(09920201)，所以最外层子查询的查询结果为

SNAME
刘　芳

例 3.4.11 查询只有一个人选修的课程号。

分析：本题查询内容是课程号，查询条件是这些课程只能有一个学生选修。该查询只涉及 SC 表，为了区别不同层次上对同一个表的查询，外层上的表取了别名 SCX。该查询的 SQL 语句如下：

```
SELECT CNO
FROM SC SCX
WHERE CNO NOT IN
    (SELECT CNO
     FROM SC
     WHERE SNO <> SCX.SNO);
```

查询结果为

CNO
EE-201
EE-122

2. 带谓词 EXISTS 的查询

EXISTS 是存在量词。带有 **EXISTS** 的子查询不返回任何数据，只返回逻辑真值和逻辑假值。若内层查询结果非空，则外层查询的 **WHERE** 后面的条件为真值，否则为假值。

例 3.4.12 查询没有选修 EE-201 课程的学生姓名。

分析：本题查询内容是学生姓名，查询条件是没有选修 EE-201 课程的那些学生。可以利用 **EXISTS** 存在量词，该查询的 SQL 语句如下：

```
SELECT SNAME
FROM STUDENT
WHERE NOT EXISTS
     (SELECT *
      FROM SC
      WHERE SNO = STUDENT.SNO AND CNO = 'EE - 201');
```

查询结果为

SNAME
刘芳
王文选
张玲
李莉平

本例的执行过程为：从头到尾扫描 STUDENT 表中的各个元组，对于每一个元组，执行子查询，如果在 SC 表中能找到 SNO＝STUDENT.SNO 的元组，则 **EXISTS** 子句返回真值，而外层 **WHERE** 条件则为假值，查询结果中不输出该元组；如果在 SC 表中找不到 SNO＝STUDENT.SNO 的元组，则 **EXISTS** 子句返回假值，而外层 **WHERE** 条件则为真

值,查询结果中输出该元组。

注意:当查询涉及多个关系时,用嵌套查询逐次求解层次分明,具有结构程序设计特点,并且嵌套查询的执行效率也比连接查询的效率高。

3. 使用 BETWEEN…AND…的查询

用 **BETWEEN…AND…**可进行范围查询,其一般形式为

E **[NOT] BETWEEN** E_1 **AND** E_2

相当于:

[NOT](E >= E_1 **AND** E <= E_2)

例 3.4.13 查询 1973 年到 1974 年出生的学生姓名。

分析:本题查询内容是学生姓名,查询条件是 1973 年到 1974 年出生。查询只涉及 STUDENT 表,因为是范围查询,可以利用 **BETWEEN…AND…**。该查询的 SQL 语句如下:

```
SELECT SNAME
FROM STUDENT
WHERE YEAR(BDATE)BETWEEN 1973 AND 1974;
```

查询结果为

SNAME
刘　芳
张晓晨
王文选
张　玲

注意:**BETWEEN** 后面是低值,**AND** 后面是高值。

4. 使用谓词 LIKE 的查询

用谓词 **LIKE** 可以进行全部或部分字符串的匹配,其一般形式为

属性列 **LIKE** 字符串常数

如果使用部分字符串匹配,则用通配符:“%”,匹配 0 个或多个字符;或“_”,匹配 1 个字符。如:a%b 表示以 a 开头,以 b 结尾的任意长度的字符串,如 acb,addgb,ab 等都满足该匹配串。a_b 表示以 a 开头,以 b 结尾的长度为 3 的任意字符串,如 acb,afb 等都满足该匹配串。

对数据库进行查询时,使用 **LIKE** 和通配符可以实现模糊查询。

例 3.4.14 查询计算机系所开课程的详细情况。

分析:计算机系开设的课程可以从课程号的头两个字母识别出来,即以 CS 开头的课程号是计算机系开设的课程。而以 EE 开头的课程号是电机系开设的课程。本题查询

内容是课程的详细情况，查询条件是计算机系开设的课程，查询只涉及 COURSE 表。可以利用 **LIKE** 谓词进行模糊匹配，因此，该查询的 SQL 语句如下：

```
SELECT *
FROM COURSE
WHERE CNO LIKE'CS%';
```

查询结果为

CNO	CNAME	LHOUR	CREDIT	SEMESTER
CS-110	计算机概论	32	2	秋
CS-201	数据结构	80	5	秋
CS-221	数字电路	64	4	春

5. 使有 ANY 或 ALL 谓词的子查询

谓词 **ANY** 的语义是指某些值；谓词 **ALL** 的语义是指所有值。它们通常需要配合比较运算符使用，例如：

> **ANY**　大于子查询结果中的某些值。
> **ALL**　大于子查询结果中的所有值。
< **ANY**　小于子查询结果中的某些值。
< **ALL**　小于子查询结果中的所有值。
>=**ANY**　大于或等于子查询结果中的某些值。
>=**ALL**　大于或等于子查询结果中的所有值。
<=**ANY**　小于或等于子查询结果中的某些值。
<=**ALL**　小于或等于子查询结果中的所有值。
=**ANY**　等于子查询结果中的某些值。
=**ALL**　等于子查询结果中的所有值(通常没有实际意义)。
!=(或<>) **ANY**　不等于子查询结果中的某个值。
!=(或<>) **ALL**　不等于子查询结果中的任何一个值。

注意：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，因为聚集函数通常能够减少比较次数。

6. 使用聚集函数的查询

SQL 提供了许多聚集函数，可以进行简单的统计和计算功能。常用的聚集函数有五类。

(1) 计数

```
COUNT([DISTINCT|ALL] * )                统计关系中元组的个数
COUNT([DISTINCT|ALL] <属性列名>)        统计某属性中值的个数
```

(2) 计算总和

SUM([DISTINCT|ALL] <属性列名>) 计算某属性值(该属性必须为数值型)的总和

(3) 计算平均值

AVG([DISTINCT|ALL] <属性列名>) 计算某属性值的平均值

(4) 求最大值

MAX([DISTINCT|ALL] <属性列名>) 求某属性值的最大值

(5) 求最小值

MIN([DISTINCT|ALL] <属性列名>) 求某属性值的最小值

例 3.4.15 统计学生总人数。

分析：本题主要统计 STUDENT 表中所有元组的个数，即学生总人数，所以可以使用聚集函数 **COUNT**。该查询的 SQL 语句如下：

```
SELECT COUNT( * ) AS NUM
FROM STUDENT;
```

查询结果为

NUM
5

例 3.4.16 查询选修了课程的学生人数。

分析：本题主要对 SC 表统计选修了课程的学生人数，由于一个学生可能选修了多门课程，但统计时不能重复计算学生人数，否则出错，所以要利用 **DISTINCT** 选项。该查询的 SQL 语句如下：

```
SELECT COUNT (DISTINCT SNO) AS NUM
FROM SC;
```

查询结果为

NUM
3

注意：用 **DISTINCT** 是为了避免重复计算学生人数。

例 3.4.17 查询每门课程的选课人数。

分析：本题主要对 SC 表统计每门课程的选课人数，可以使用 **GROUP BY** 子句按课程号进行分组，然后用 **COUNT** 函数统计各组的人数。该查询的 SQL 语句如下：

```
SELECT CNO,COUNT(SNO) AS NUM
FROM SC
```

```
GROUP BY CNO;
```

查询结果为

CNO	NUM
CS-110	3
CS-201	2
EE-122	1
EE-201	1

注意：**GROUP BY** 子句是按某一属性列或多个属性列的值进行分组，值相等的分为一组。另外，分组的属性列名(本题为 CNO)必须在本层的 **SELECT** 语句中出现。

例 3.4.18 查询选修了 3 门以上课程的学生的学号。

分析：本题查询内容是学生的学号，查询条件是这些学生必须选修了 3 门以上的课程。该查询只涉及 SC 表。可以利用 **GROUP BY** 子句根据 SNO 进行分组，然后统计每组的课程数，凡是选修了 3 门以上课程的学生的学号可出现在查询结果中。该查询的 SQL 语句如下：

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

查询结果为

SNO
09920203

注意：**HAVING** 短语指定选择组的条件，只有满足条件(元组数大于 3，表示此学生选修的课程数超过 3 门)的组才会在查询结果中。

WHERE 子句与 **HAVING** 短语是有区别的。**WHERE** 子句作用于基表或视图，**HAVING** 短语作用于组。另外，**HAVING** 短语必须紧跟在 **GROUP BY** 后面。

对于前面的例 3.4.11，也可以使用聚集函数写 SQL 语句：

```
SELECT SNO
  FROM SC
  GROUP BY SNO
  HAVING COUNT( * )> = 1;
```

这说明同一个查询可用不同的 SQL 语句写，即查询语句不是唯一的。

例 3.4.19 查询计算机系所开课程的最高成绩、最低成绩和平均成绩。如果某门课程的成绩不全(即 GRADE 中有 NULL 出现)，则该课程不予统计，结果按 CNO 升序排列。

分析：本题查询内容为每门课程的最高成绩、最低成绩和平均成绩，查询条件是这些

课程必须为计算机系开设的课程。由于课程由哪个系开设的信息在 CNO 中可以看出，而成绩信息也在 SC 表中，所以本题查询只涉及 SC 表。该查询的 SQL 语句如下：

```
SELECT CNO,MAX (GRADE) AS MAXGRADE,MIN (GRADE) AS MINGRADE,AVG(GRADE) AS AVGGRADE
FROM  SC
WHERE  CNO LIKE 'CS%'              -- 选择计算机系所开的课程
GROUP BY CNO                       -- 按 CNO 分组
HAVING CNO NOT IN
              (SELECT CNO          -- 删除成绩不全的组
              FROM SC
              WHERE GRADE IS NULL)
ORDER BY CNO;   -- 结果按 CNO 升序排序
```

查询结果为

CNO	MAXGRADE	MINGRADE	AVGGRADE
CS-110	95	82	87.3
CS-201	90	75	82.5
EE-201	80	80	80

注意：对 SQL 语句进行注释的方法有如下两种。

(1) 单行语句注释，使用注释符号(--)。

(2) 多行注释方法，使用注释符号(/*…*/)。

3.4.3 集合查询

关系代数中可以进行两个关系的并、交和差的集合运算，在 SQL 中也可实现集合运算。SQL 提供了保留字 **UNION**、**INTERSECT** 和 **EXCEPT** 以实现关系的集合运算，前提是参加集合运算的两个关系必须具有相等的目，且对应的属性域相同。

1. 查询的并集

例 3.4.20　查询 1973 年出生的学生和选修电机系所开课程(EE 标志)的学生的学号。

分析：本题查询内容是学生的学号。查询条件有两个，一个条件是出生时间为 1973 年的学生，此信息在 STUDENT 表中；另一个条件是选修了电机系所开课程的学生，此信息在 SC 表中。可以先分别查询 1973 年出生的学生和选修电机系所开课程的学生的学号，然后利用关系的并操作，将两个查询结果进行合并。查询并集用保留字 **UNION**。该查询的 SQL 语句如下：

```
(SELECT SNO
 FROM STUDENT
 WHERE YEAR(BDATE) = 1973)
  UNION
```

```
(SELECT SNO
 FROM SC
 WHERE CNO LIKE 'EE%');
```

查询结果为

SNO
09920203

2. 查询的交集

例 3.4.21 查询计算机系 1973 年出生的学生详细信息。

分析：本题可以用查询的交集来得到结果。查询交集保留字用 AND。该查询的 SQL 语句如下：

```
SELECT *
FROM STUDENT
WHERE DEPT = '计算机系'
  AND YEAR(BDATE) = 1973;
```

查询结果为

SNO	SNAME	SEX	BDATE	DEPT
09920203	王文选	男	1973-11-15	计算机系

3. 查询的差集

例 3.4.22 查询选修了 CS-110 课程但没有选修 CS-201 课程的学生学号。

分析：本题可以先查询选修了 CS-110 课程的学生学号集合 X，然后查询选修了 CS-201 课程的学生学号集合 Y，最后通过差运算从 X 中减去 Y 得到选修了 CS-110 课程但没有选修 CS-201 课程的学生学号。该查询的 SQL 语句如下：

```
(SELECT CNO
FROM SC SCX)
EXCEPT
 (SELECT CNO
  FROM SC);
```

查询结果为

SNO	SNAME
09920301	张　玲
09920302	李莉平

注意：由于在不同层次上对同一个表查询，为区别起见，外层上的表取了别名 SCX。

3.5 数据操纵语言

SQL 中的数据操纵语言主要用于在关系模式中插入、删除以及修改元组数据。

3.5.1 插入数据

SQL 中提供了 **INSERT** 语句将元组数据插入关系模式中去。

1. 插入一个元组

插入元组的语句格式为

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

其中：

——**INSERT** 表示对表进行插入元组操作。

——**INTO** 子句：

- 指定要插入元组的表名及属性列。
- 属性列的顺序可与表定义中的顺序不一致。
- 如果没有指定属性列则表示要插入的是一条完整的元组，且属性列与表定义中的顺序一致。
- 如果指定部分属性列则插入的元组在其余属性列上取空值。

——**VALUES** 子句：

- 提供给属性列插入的值。
- 提供的值的个数和值的类型必须与 INTO 子句匹配。

例 3.5.1 将一个新学生记录（学号，09920303；姓名，韩晓红；性别，女；出生年月，1976-4-7；所在系，计算机系）插入 STUDENT 表中。

分析：该题是向 STUDENT 表中插入一个新的元组。SQL 插入语句为

```
INSERT INTO STUDENT
VALUES ('09920303', '韩晓红', '女', 1976 - 4 - 7, '计算机系');
```

插入后的 STUDENT 表中的最后一行增加了一个新的元组。

SNO	SNAME	SEX	BDATE	DEPT
09920201	刘　芳	女	1974-3-12	计算机系
09920202	张晓晨	男	1974-1-24	计算机系
09920203	王文选	男	1973-11-15	计算机系
09920301	张　玲	女	1974-8-19	计算机系
09920302	李莉平	女	1973-9-24	计算机系
09920303	韩晓红	女	1976-4-7	计算机系

2. 插入子查询的结果

插入子查询结果的语句格式为

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

例 3.5.2 生成一个女学生成绩临时表 FGRADE，表中包括 SNAME、CNO、GRADE 三个属性。

分析：本题是将子查询的结果插入到指定的表中。可以首先定义一个临时表 FGRADE，即

```
CREATE TABLE FGRADE
    (SNAME VCHAR(8)  NOT NULL,
    CNO  CHAR(6) NOT NULL,
    GRADE DEC(4,1)  DEFAULT NULL);
```

然后，利用子查询的结果将其插入临时表 FGRADE 中。

```
INSERT INTO FGRADE
    SELECT SNAME,CNO,GRADE
    FROM STUDENT,SC
    WHERE STUDENT.SNO = SC.SNO;
```

插入后的临时表 FGRADE 具有如下结果：

SNAME	CNO	GRADE
刘　芳	CS-110	95
刘　芳	CS-201	90
张晓晨	CS-110	85
张晓晨	EE-201	80
王文选	CS-110	82
王文选	CS-201	75
王文选	EE-122	

注意：DBMS 在执行插入语句时会检查欲插入的元组是否会破坏表上已定义的完整性规则。如果破坏，系统会给出提示，且该记录不会被插入数据库中。

3.5.2 修改数据

SQL 中提供了 UPDATE 语句对指定关系模式中的数据进行修改操作。修改语句的格式为

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

其中：

——**UPDATE** 表示对指定的表中满足条件的元组进行修改操作。表达式中可以出现常数、属性列名、聚集函数以及运算符。

——**SET** 子句指定要修改的属性和修改后取值。

——**WHERE** 子句：

- 指定要修改的元组必须满足的条件。
- 默认时表示要修改表中的所有元组。

例 3.5.3 将 SC 表中所有学生的成绩置 0，缺成绩的除外。

分析：本题是多记录修改。除了缺成绩的以外，其他所有学生的成绩都要修改为 0。SQL 修改语句为

```
UPDATE SC
SET GRADE = 0
WHERE GRADE IS NOT NULL;
```

修改后的 SC 表为

SNO	CNO	GRADE
09920201	CS-110	0
09920201	CS-201	0
09920202	CS-110	0
09920202	EE-201	0
09920203	CS-110	0
09920203	CS-201	0
09920203	EE-122	

注意：DBMS 在执行修改语句时会检查欲修改的元组是否会破坏表上已定义的完整性规则。如果破坏，系统会给出提示，且该记录不会被修改。

例 3.5.4 把学生“李莉平”的姓名改为“李莉萍”。

分析：本题是单记录修改。SQL 修改语句为

```
UPDATE STUDENT
SET SNAME = '李莉萍'
WHERE SNAME = '李莉平';
```

修改后的 STUDENT 表为

SNO	SNAME	SEX	BDATE	DEPT
09920201	刘　芳	女	1974-3-12	计算机系
09920202	张晓晨	男	1974-1-24	计算机系
09920203	王文选	男	1973-11-15	计算机系
09920301	张　玲	女	1974-8-19	计算机系
09920302	李莉萍	女	1975-9-24	计算机系

3.5.3 删除数据

SQL 提供了 DELETE 语句对指定关系模式中的元组数据进行删除。删除语句的格式为

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

其中：

——**DELETE** 表示对表中的元组进行删除操作。

——**FROM** 指定在哪张表中删除。

——**WHERE** 子句：

- 指定要删除的元组必须满足的条件。
- 默认时表示要修改表中的所有元组。

例 3.5.5 删除姓名为“李莉萍”的学生记录。

```
DELETE
FROM STUDENT
WHERE SNAME = '李莉萍';
```

删除后的 STUDENT 表为

SNO	SNAME	SEX	BDATE	DEPT
09920201	刘　芳	女	1974-3-12	计算机系
09920202	张晓晨	男	1974-1-24	计算机系
09920203	王文选	男	1973-11-15	计算机系
09920301	张　玲	女	1974-8-19	计算机系

注意：DBMS 在执行删除语句时会检查删除所选的元组后是否会破坏表上已定义的完整性规则。如果破坏，系统会给出提示，且该记录不会被删除。

前面介绍过 **DROP** 语句，现在又介绍了 **DELETE** 语句，这两个语句之间的区别如下。

- **DROP TABLE** <表名>　表示删除指定的基表。DBMS 执行后，该表不再存在。
- **DELETE　FROM** <表名>　表示删除指定表中的元组。DBMS 执行后，该表仍然存在，但所有元组被删除，已为空表。

3.6 视图

视图是从一个或几个基表(或视图)导出的表，它与基表不同，是一个虚表，本身不保存数据，数据仍保存在基表中。DBMS 执行 **CREATE VIEW** 语句时只是把视图的定义存入数据目录，并不执行其中的 **SELECT** 语句。在对视图查询时，按视图的定义从基表中将数据

查出。

视图一经定义,就可以像基表一样被查询和删除,并且可以在视图之上再定义新的视图。如果基表中的数据发生变化,从视图中查询出的数据也随之改变。不过,视图的更新(增加、删除、修改)操作会受到一定的限制。

3.6.1 定义视图

SQL 中视图的定义主要使用 **CREATE VIEW** 语句,其一般形式为

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

注意:

——**CREATE VIEW** <视图名>表示视图的名字,该视图内容由若干个属性列表组成。

——<子查询>是视图的定义,子查询中的属性列不允许定义别名,一般不允许含有 **ORDER BY** 和 **DISTINCT** 短语。

——**WITH CHECK OPTION** 表示对视图进行 **UPDATE**、**INSERT**、**DELETE** 操作时要保证所操作的元组必须满足视图定义的谓词条件。

例 3.6.1 定义一个视图 ENROL-SPRING,作为学生春季选课一览表,其中含有 SNO、SNAME、CNO、CREDIT 属性。

分析:本题要求定义一个学生春季选课视图,由于学生姓名在 STUDENT 表中,选课信息在 SC 表中,而课程开设学期在 COURSE 表中,所以该视图根据这三个基表进行定义。SQL 语句如下:

```
CREATE VIEW ENROL - SPRING
AS SELECT SNO,SNAME,CNO,CREDIT
   FROM STUDENT,COURSE,SC
   WHERE STUDENT.SNO = SC.SNO
     AND COURSE.CNO = SC.CNO
     AND SEMESTER = '春';
```

该视图是利用 SELECT 查询的结果进行定义的。

例 3.6.2 定义一个视图 GRADE-AVG,表示学生的平均成绩,其中包括 SNO 和 AVGGRADE(平均成绩)两个属性。

分析:定义一个表示学生平均成绩的视图,该视图可以根据 STUDENT 和 SC 进行定义。SQL 语句如下:

```
CREATE VIEW GRADE - AVG(SNO,AVGGRADE)
AS SELECT SNO,AVG(GRADE)
   FROM SC
   GROUP BY SNO;
```

该视图也是利用 **SELECT** 查询的结果进行定义的。可见,视图实际上是一个 **SELECT** 语句。

3.6.2 查询视图

一旦定义了视图,就可以像查询基表一样查询视图。事实上,对视图的查询最终将转变为对基表的查询。

例 3.6.3 查询平均成绩大于 90 分的学生学号。

分析:本题查询可以利用例 3.6.2 定义好的视图 GRADE-AVG。SQL 语句如下:

```
SELECT *
FROM GRADE - AVG
WHERE AVGGRADE > = 90;
```

DBMS 在数据目录中找到该视图的定义,然后将其转换为对基表的查询,相当于执行了以下 SQL 语句:

```
SELECT SNO, AVG(GRADE)
FROM SC
GROUP BY SNO
HAVING AVG(GRADE)> = 90;
```

需要注意的是,对视图的两次查询结果可能不同,原因是虽然没有改动视图 GRADE-AVG 的定义,但是基表 SC 在此期间可能发生了变化。

3.6.3 删除视图

当视图不再需要时,可以将视图删除。SQL 中视图的删除主要使用 **DROP VIEW** 语句,其一般形式为

```
DROP VIEW <视图名>;
```

注意:

——该语句从数据目录中删除指定的视图定义。

——由该视图导出的其他视图定义仍在数据目录中,但已不能使用,必须显式删除。

——删除基表时,由该基表导出的所有视图定义都必须显式删除。

例 3.6.4 删除视图 ENROL-SPRING。

分析:本题要求从数据目录中将视图 ENROL-SPRING 的定义删除。SQL 语句如下:

```
DROP VIEW ENROL - SPRING;
```

3.6.4 更新视图

更新视图是一个较复杂的问题,通常都会加以限制。例如,由一个基表定义的视图,如果只含有基表的主键或候补键,并且视图中没有用表达式或函数定义的属性,才允许更新;如果是由多表连接所定义的视图,则视图不允许更新;如果视图定义中用到 **GROUP BY** 子句或聚集函数,则视图不允许更新。如果定义视图时带有 **WITH CHECK OPTION** 子句,则 DBMS 在更新视图时会进行检查,防止用户通过视图对不属于视图范围内的基表数据进行更新。

3.6.5 视图的作用

视图是关系数据库系统提供给用户以多种角度观察数据库中数据的重要机制。视图的作用主要体现在以下几点。

(1) 提供了逻辑数据独立性

在关系数据库中,数据的整体逻辑结构或存储结构都有可能发生改变,如果这些改变与用户无关,那么原有的应用程序就不必修改;当这些改变与用户有关时,也只要修改视图,应用程序仍可不改动或只需做少量改动。

(2) 简化了用户观点

数据库的全部结构是复杂的,并有多种联系。一般用户只用到数据库中一部分数据,而视图机制正好适应了用户的需要。视图是由一个 SELECT 语句定义的,用户只需关心视图的内容,而不必关心构成视图的若干关系的连接、投影操作。

(3) 提供数据的安全保护功能

在数据库中,有些数据是保密的,不能让用户随便使用。此时,可针对不同的用户定义不同的视图,在视图中只出现用户需要的数据。系统提供视图让用户使用,而不是关系。这样,就可达到数据的安全保护功能。

例如,全校学生的成绩在一张表中,如果要限制各系的教务员只能查询本系学生的成绩,就可以为每个系教务员定义一个只包含本系学生成绩的视图。这样,各系教务员查成绩时就只能查自己系的学生成绩。

又如,为了不让无关人员了解有关职工个人收入情况,可以定义一个不包含经济收入属性的视图,供查询一般情况用。

要注意,有些 DBMS 没有视图功能,但系统可以根据用户访问限制条件,自动地修改查询条件,使用户只能在给定访问范围内查询。例如,计算机系教务员要查学生情况,DBMS 可在其查询语句上自动加上 "DEPT = '计算机系'"的限制条件。因此,查询修改实际上起了视图的作用。而且处理起来比视图还简单,容易实现,效率也高。

3.7 数据控制语言

SQL 中的数据控制功能包括事务管理功能和数据保护功能，即数据库的恢复和并发控制，数据库的安全性和完整性控制。本节主要讨论数据库的安全性和完整性控制，而数据库的恢复和并发控制将在第 4 章讨论。

数据库的安全性措施之一是通过控制用户对数据库中数据的访问权限来保证数据的安全性。数据库的安全性控制可通过授权机制来实现，详见本节其余部分的讨论。不同类型的操作对象具有不同的操作权限，常见的操作权限如表 3.6 所示。

表 3.6 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 PRIVIEGES（四种权限总和）
视图	TABLE	SELECT，INSERT，UPDATE，DELETE，ALL PRIVIEGES（四种权限总和）
基表	TABLE	SELECT，INSERT，UPDATE，DELETE，ALTER，INDEX，ALL PRIVIEGES(六种权限总和）
数据库	DATABASE	CREATETAB 建立表的权限，可由 DBA 授予普通用户

访问控制在数据库之间是相互独立的，一个用户在一个数据库所获得的访问权限不能用于其他数据库。一个用户可能在一个数据库中享有 DBA 特权，而在另一个数据库中可能只是一般用户。

数据库用户按其访问权力的大小，一般可分为以下三类。

（1）一般数据库用户

在 SQL 中，这种用户称为“具有 CONNECT 特权的用户”。这种用户可以与数据库连接，并具有下列特权。

① 按照授权可以查询或更新数据库中的数据。

② 可以创建视图或定义数据的别名。

（2）具有支配部分数据库资源特权的数据库用户

在 SQL 中，这种用户称为“具有 RESOURCE 特权的用户”，除具有一般数据库用户所拥有的所有特权外，还具有下列特权。

① 可以创建表、索引和簇集。

② 可以授予或收回其他数据库用户对其所创建的数据对象所拥有的访问权。

③ 有权对其所创建的数据对象跟踪审查。

（3）具有 DBA 特权的数据库用户

DBA 拥有支配整个数据库资源的特权。这种用户除具有上述两种用户所拥有的一切特权外，还具有下列特权。

① 有权访问数据库中的任何数据。

② 不但可以授予或收回数据库用户对数据对象的访问权,还可以批准或收回数据库用户。

③ 可以为 PUBLIC 定义别名,PUBLIC 是所有数据库用户的总称。

④ 有权对数据库进行调整、重组或重构。

⑤ 有权控制整个数据库的跟踪审查。

具有 DBA 特权的用户对数据库拥有最大的特权,因而也对数据库负有特别的责任。

DBMS 按用户的访问权限来控制用户的数据访问,因此,需要解决两个问题。

• 用户的标识与鉴别

用户的标识是数据库用户注册时给出的,这是一个公开的标识。

• 授权

SQL 语言提供 GRANT 语句向用户授予对数据库的操作权限。

3.7.1 授权

授权就是给予用户一定的访问特权,这是对用户访问权限的规定和限制。在 SQL 中,有两种授权:一种是授予某类数据库用户的特权,只有得到这种授权,才能成为数据库用户,这只能由 DBA 授予;另一种是授予对某些数据对象进行某些操作的特权,这可以由 DBA 授予,也可由数据对象的创建者授予。

对于第一种授权,可用下面的 SQL 语句:

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

其中:

——**GRANT** 表示授权操作,后面<权限>是指定的操作权限。

——**ON** 后面是指定的对象类型和对象名。

——**TO** 后面是接受权限的用户,可以是一个或多个用户,也可是 PUBLIC 用户。

——**WITH GRANT OPTION** 表示用户可将权限授予其他用户。

例 3.7.1 把查询表 STUDENT 的权限授给用户 U1。

分析:本题是一个授权操作,即把查询表 STUDENT 的权限(**SELECT**)授予用户 U1。SQL 语句如下:

```
GRANT SELETE
ON TABLE STUDENT
TO U1;
```

例 3.7.2 把对表 STUDENT 和 COURSE 的全部操作权限授予用户 U2 和 U3。

分析:本题也是一个授权操作,即把查询表 STUDENT 和 COURSE 的全部操作权限(ALL PRIVILIGES)授予用户 U2 和 U3。SQL 语句如下:

```
GRANT ALL PRIVILIGES
```

```
ON TABLE STUDENT, COURSE
TO U2,U3;
```

例 3.7.3 把查询表 STUDENT 和修改学生学号的权限授予用户 U4。

分析：本题也是一个授权操作，即把查询表 STUDENT 和修改学生学号的权限（**SELECT** 和 **UPDATE**）授予用户 U4。SQL 语句如下：

```
GRANT UPDATE(SNO),SELETE
ON TABLE STUDENT
TO U4;
```

例 3.7.4 把对表 SC 的 INSERT 权限给用户 U5，并允许将此权限再授予其他用户。

分析：本题除了给用户 U5 授权外，还允许该用户将此权限再授予其他用户。SQL 语句如下：

```
GRANT INSERT
ON TABLE SC
TO U5 WITH GRANT OPTION;
```

如 U5 要将此权限授予 U6，并允许 U6 将此权限再授予其他用户，则 SQL 语句如下：

```
GRANT NSERT
ON TABLE SC
TO U6 WITH GRANT OPTION;
```

U6 还可以将此权限授予 U7，SQL 语句如下：

```
GRANT INSERT
ON TABLE SC
TO U7;
```

因为 U6 未给 U7 再授权的权限，因此 U7 不能再传播此权限。

例 3.7.5 DBA 把在数据库 SELECT-COURSE 中建立表的权限授予用户 U8。

分析：本题是授予用户 U8 在数据库 SELECT-COURS 中建立表的权限。授权 SQL 语句如下：

```
GRANT CREATE TABLE
ON DATABASE SELECT - COURSE
TO U8
```

3.7.2 收回权限

SQL 还提供了将用户权限收回的功能。收回权限的 SQL 语句如下：

```
REVOKE <权限>
```

```
ON <对象类型> <对象名>
FROM <用户>[,<用户>];
```

注意：收回权限时，连同转授予其他用户的权限一起收回。

例 3.7.6 设有用户 U1、U2、U3 和 U4，下面是一连串授权过程：

U1 授权给 U2：

```
GRANT SELECT ON TABLES TO U2 WITH GRANT OPTION;
```

U2 授权给 U3：

```
GRANT SELECT ON TABLES TO U3 WITH GRANT OPTION;
```

U3 授权给 U4：

```
GRANT SELECT ON TABLES TO U4 WITH GRANT OPTION;
```

U1 收回授给 U2 的权限：

```
REVOKE SELECT ON TABLES TO U2 WITH GRANT OPTION;
```

此语句不仅收回 U1 授予 U2 的权限而且还收回 U2 授予 U3、U3 授予 U4 的这种权限。

例 3.7.7 收回所有用户对 SC 表的查询权限。

分析：本题是收回所有用户(**PUBLIC**)对 SC 表的查询(**SELECT**)权限。收回权限的 SQL 语句如下：

```
REVOKE SELECT
ON SC
FROM PUBLIC;
```

3.7.3 完整性控制

SQL 标准使用了一系列的技术来表达完整性，包括实体完整性、引用完整性、域完整性。这些语义约束完整性条件的功能主要在 **CREATE TABLE** 语句中表达，由 DBMS 实现。

具体例子可见 3.3.1 节内容。

3.8 嵌入式 SQL

3.8.1 嵌入式 SQL 介绍

SQL 是一种非过程化的查询语言，大多数语句可以独立执行，不能根据不同的条件执行不同的任务，所以单纯用 SQL 语句很难完成实际的应用，往往需要将 SQL 与其他高级语言结合起来使用。

SQL 又可分为交互式 SQL 和嵌入式 SQL。所谓交互式 SQL 是指在 DBMS 环境中直接使用 SQL 对关系进行交互式操作，但是这种操作仅限于数据库操作，缺少数据处理能力。而嵌入式 SQL 是指把 SQL 嵌入程序设计语言中，即宿主语言中，嵌入式 SQL 利用了宿主语言的数据处理能力，SQL 语句负责操纵数据库，宿主语言语句负责控制程序流程。

由于 SQL 是基于关系数据模型的语言，而宿主语言是基于整数、实数、字符、记录、数组等数据类型的语言，因此两者之间存在很大差别。例如，SQL 语句不能直接使用指针、数组等数据结构，而宿主语言一般不能直接进行集合运算。为了能在宿主语言的程序中嵌入 SQL 语句，必须做某些规定。本节主要介绍嵌入式 SQL 的一些使用规定和使用技术。

为了能够将 SQL 语言同其他宿主语言结合起来使用。嵌入式 SQL 必须解决下列 4 个问题。

① 宿主语言编译器不可能识别和接受 SQL 语言，如何将嵌有 SQL 的宿主语言程序编译成可执行码，这是首先要解决的问题。

② 宿主语言和 DBMS 之间如何传递数据和信息。

③ 数据库的查询结果一般是元组的集合，这些元组须逐个地赋值给宿主语言程序中的变量，供其处理，其间存在一个转换问题，如何进行这种转换。

④ 两者的数据类型有时不完全对应或等价，须解决必要的数据类型转换问题。需要进行何种数据类型转换，与宿主语言和 DBMS 有关。

各个 DBMS 在实现嵌入式 SQL 时，对不同的宿主语言，所用的基本方法是一样的。但由于宿主语言的差异，在实现时须利用其各自的特点，也须解决各自的特殊问题。下面以 SQL 嵌入 C 语言为例，说明实现嵌入式 SQL 的一般方法。

3.8.2 嵌入式 SQL 的说明部分

为了在程序中能够区别 C 语句和 SQL 语句，凡是 SQL 语句，前面都以 EXEC SQL 开头，结尾加分号“；”(注意：不同宿主语言中 SQL 的结束标志会有区别)。C 和 SQL 之间的数据传送则通过宿主变量进行。所谓宿主变量是指 SQL 中可引用的 C 语言变量；凡是宿主变量，前面须用 EXEC SQL 开头的说明语句说明。在 SQL 语句中引用宿主变量时，为了有别于数据库本身的变量，例如属性名，在宿主变量前须加冒号“:”。因此，即使宿主变量与数据库中的变量同名也是允许的。在宿主语言语句中，宿主变量可与其他变量一样使用，不须加冒号。宿主变量按宿主语言的数据类型及格式定义，若与数据库中的数据类型不一致，则由数据库系统按实现时的约定进行必要的转换。在实现嵌入式 SQL 时，往往对宿主变量的数据类型加以适当的限制，例如对 C 语言，不允许用户定义宿主变量为数组或结构。

在宿主变量中，有一个系统定义的特殊变量，叫 SQLCA(SQL communication area，SQL 通信区)。它是全局变量，供应用程序与 DBMS 通信之用。由于 SQLCA 已由系统

定义,只需在嵌入的可执行 SQL 语句开始前加 INCLUDE 语句就行了,而不必由用户说明,其格式为

```
EXEC SQL INCLUDE SQLCA
```

SQLCA 中有一个分量叫 SQLCODE,可表示为 SQLCA. SQLCODE。它是一个整数,供 DBMS 向应用程序报告 SQL 语句执行情况之用。每执行一条 SQL 语句后,都要返回一个 SQLCODE 代码,其具体含义随系统而异,一般规定:

- SQLCODE 为零,表示 SQL 语句执行成功,无异常情况。
- SQLCODE 为正数,表示 SQL 语句已执行,但有异常情况,例如 SQLCODE 为 100 时,表示无数据可取,可能是数据库中无满足条件的数据,也可能是查询的数据已被取完。
- SQLCODE 为负数,表示 SQL 语句因某些错误而未执行,负数的值表示错误的类别。

宿主变量不能直接接受空缺符 NULL。凡遇此情况,可在宿主变量后紧跟一指示变量。指示变量也是宿主变量,一般是一个短整数,用来指示前面的宿主变量是否为 NULL。如果指示变量为负,表示前面的宿主变量为 NULL,否则,不为 NULL。

所有 SQL 语句中用到的宿主变量,除系统定义之外,都必须说明,说明的开头行为

```
EXEC SQL BEGIN DECLARE SECTION;
```

结束行为

```
EXEC SQL END DECLARE SECTION;
```

例 3.8.1 说明宿主变量。

分析:本题意为在宿主语言中说明可在 SQL 中引用的 C 语言变量。说明语句如下:

```
EXEC SQL BEGINE DECLARE SECTION;
  CHAR SNO[7];
  CHAR GIVENSNO[7];
  CHAR CNO[6];
  CHAR GIVENCNO[6];
  FLOAT GRADE;
  SHORT GRADEI;
EXEC SQL END DECLARE SECTION;
```

本题中,SNO、CNO、GRADE 是作为宿主变量说明的,虽与表 SC 的属性列同名也无妨。GRADEI 是 GRADE 的指示变量,它只有与 GRADE 连用才有意义。必须注意,上述的宿主变量是按 C 语言的数据类型和格式说明的,与 SQL 有些区别。

3.8.3 嵌入式 SQL 的可执行语句

嵌入式 SQL 的说明部分不对数据库产生任何作用。下面介绍作用于数据库的嵌入

式 SQL 语句，即可执行语句。这包括嵌入的 DDL、QL、DML 及 DCL 语句。这些语句的格式与对应的交互式 SQL 语句基本一致，只不过因嵌入的需要增加了少许语法成分。此外，可执行语句还包括进入数据库系统的 CONNECT 语句以及控制事务结束的语句。例如，CONNECT 语句的格式为

```
EXEC SQL CONNECT : uid IDENTIFIED BY  : pwd;
```

其中，uid、pwd 为两个宿主变量，前者表示用户标识符，后者表示该用户的口令。这两个宿主变量应在执行 CONNECT 语句前由宿主语言程序赋值，执行本语句成功后才能执行事务中的其他可执行语句。执行成功与否可由 SQLCODE 判别。

嵌入式 SQL 的 DDL 和 DML 语句除了前面加 EXEC SQL 外，与交互式 SQL 没有什么区别。

例 3.8.2 将宿主变量 SNO，CNO，GRADE 中的值插入到表 SC 中。

分析：本题要求在宿主语言中实现往 SC 表中插入元组值的功能，插入的元组由三个宿主变量构成，宿主变量由宿主语言程序赋值。嵌入式 SQL 语句如下：

```
EXEC SQL INSERT INTO SC(SNO,CNO,GRADE)
VALUES(:SNO,:CNO,:GRADE);
```

例 3.8.3 在选课表中删除变量 SNAME 中指定的学生选课记录。

分析：本题要求在宿主语言中实现在 SC 表中删除元组的功能，被删除的元组由变量名 SNAME 指定。嵌入式 SQL 语句如下：

```
EXEC SQL DELETE
FROM SC
WHERE SNO IN
   (SELETE SNO
   FROM STUDENT
   WHERE NAME = :SNAME);
```

查询语句是用得最多的嵌入式 SQL 语句。如果查询的结果只有一个元组，则可将查询结果用 INTO 子句对有关的宿主变量直接赋值。

例 3.8.4 查询某个学生某门课程的成绩。

分析：本题要求在宿主语言中实现对 SC 表的查询功能，被查询学生的学号和课程由 GIVENSNO 和 GIVENCNO 指定。查询结果插入到宿主变量 GRADE 中。

```
EXEC SQL SELECT GRADE INTO :GRADE,:GRADEI
FROM SC
WHERE SNO = :GIVENSNO AND CNO = :GIVENCNO;
```

由于{SNO,CNO}是 SC 的主键，本句的查询结果不超过一个元组(单属性元组)，可以直接用 INTO 子句对有关的宿主变量赋值。如果不是用主键查询，则查询结果可能有多个元组；若仍直接对宿主变量赋值，则系统可能会报错。因为 GRADE 属性允许为 NULL，故在宿主变量 GRADE 后加了指示变量 GRADEI。

如查询结果超过一个元组,需在程序中开辟一个区域来存放查询的结果。该区域及其相应的数据结构称为游标。然后利用游标逐个地取出每个元组给宿主变量赋值。

使用游标是为了 SQL 的集合处理方式与宿主语言单记录处理方式的协调。由于 SQL 语句可以处理一组记录,而宿主语言语句一次只能处理一个记录,因此需要游标机制把集合操作转换成单记录方式。

使用游标需要下面 4 条语句。

(1) 说明游标语句

说明游标语句定义一个命名的游标,并将它与相应的查询语句相联系。说明游标语句格式为

```
EXEC SQL DECLARE <游标名> CURSOR FOR
    SELECT …
    FROM …
    WHERE …;
```

其中,<游标名>是为定义的游标取的名字,这个游标与对应的查询语句相联系。

(2) 打开游标语句

打开游标语句是将定义的一个游标打开,在打开游标时,执行与游标相联系的 SQL 查询语句,并将查询结果置于游标中。打开游标的语句格式为

```
EXEC SQL OPEN <游标名>;
```

注意:

- 游标打开时,位于第一个元组的前一位置。
- 游标中的查询结果对应于打开游标时的宿主变量的当前值。
- 打开游标后,即使宿主变量改变,游标中的查询结果也不随之改变,除非游标关闭后重新打开。

(3) 取数语句

取数语句是取游标所在位置的元组。取数语句格式为

```
EXEC SQL FETCH <游标名> INTO :hostvar1,:hostvar2,…;
```

注意:

- 每次执行取数语句时,首先把游标向前推进一个位置,然后按照游标的当前位置取一元组并对宿主变量 hostvar1,hostvar2,…赋值。
- 与单个元组的查询不一样,INTO 子句不是放在查询语句中,而是放在取数语句中。
- 要恢复游标的初始位置,必须关闭游标后重新打开。
- 在 SQL 中,有游标后退及跳跃功能,游标可以定位到任意位置。如果游标中的数已经取完,若再执行取数语句,SQLCODE 将返回代码 100。

(4) 关闭游标语句

因取完数或取数发生错误等原因而不再使用游标时,应关闭游标。关闭游标语句的

格式为

```
EXEC SQL CLOSE <游标名>;
```

游标关闭后，如果再对它取数，将返回出错信息，说明要从中取数的游标无效。

例 3.8.5 输入学号并存在变量 givensno 内，查询学号、课程号以及成绩。

分析：本题是在 C 语言中使用嵌入式 SQL 的一个例子。完成此功能的 C 语言程序如下：

```
#define NO - MORE - TUPLES
void sel()
{
    EXEC SQL BEGIN DECLARE SECTION  -- 对 SQL 语句中用到的宿主变量进行说明
    char snun[8],cnun[6],givensno [8];
    Int gr;
    char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION  -- 结束说明
    EXEC SQL DECLARE SCX CURSOR FOR
/ * 说明 SCX 为一个游标，并与查询语句对应 * /
       SELECT SNO,CNO,GRADE
       FROM SC
       WHERE SNO = : givensno;
    EXEC SQL OPEN SCX;  -- 打开游标 SCX
    while(1)
    {
    EXEC SQL FETCH SCX INTO :snun,:cnun,: gr;
/ * 将游标所在处的元组值取到宿主变量中 * /
    if(NO - MORE - TUPLES)break;
    printf(" % s, % s, % d",snun,cnun,gr);
    }
    EXEC SQL CLOSE SCX;  -- 关闭游标 SCX
}
```

注意：NO-MORE-TUPLES 是 C 语言中定义的宏，检测查询状态，当查找结束时值为.T.。

3.9 嵌入式 SQL 的实现

嵌入式 SQL 的实现有两种处理方式：第一种方式是通过扩充宿主语言的编译程序，使之能处理 SQL 语句；第二种方式是采用预处理方式，即先用预处理程序对源程序进行扫描，识别出 SQL 语句，并处理成宿主语言的过程调用语句；然后再用宿主语言的编译程序把源程序编译成目标程序。目前多数系统采用后一种方式。嵌入式 SQL 的处理过程如图 3.1 所示，其中最关键的一步是将嵌有 SQL 的宿主语言源码通过预编译器变成纯宿主语言源码。关系 DBMS 除了提供 SQL 语言接口外，一般还提供一批函数，称为 SQL 函数，供应用程序调用 DBMS 的各种功能。例如建立与 DBMS 的连接及其连接的

环境、传送 SQL 语句、执行 SQL 语句并建立游标、返回执行结果、返回执行状态及各种异常情况等。这些函数组成 SQL 函数库。SQL 函数库实际上是 DBMS 向应用程序提供的一种访问数据库的接口,称为调用级接口。预编译器将冠以 EXEC SQL 的语句编译成宿主语言对 SQL 函数的调用,从而把嵌有 SQL 的宿主语言源码变换成纯宿主语言源码,可以在编译连接后执行。

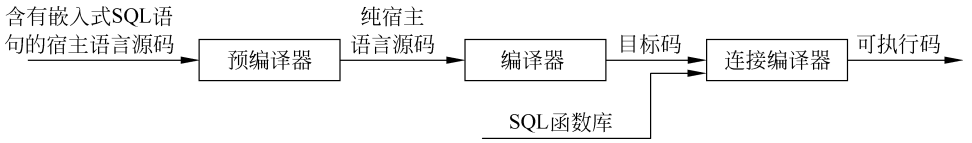

图 3.1　嵌入式 SQL 处理过程

3.10　动态 SQL

嵌入式 SQL 语句必须在源程序中完全确定,然后再由预处理程序预编译和宿主语言编译程序编译。在实际问题中,源程序往往还不能包括用户的所有操作。用户对数据库的操作有时在系统运行时才能提出来,例如,编译时下列信息不能确定:SQL 语句正文、宿主变量个数、宿主变量的数据类型、SQL 语句中引用的数据库对象(属性、索引、基表、视图等)。

例如,对 SC 表,任课老师想查询选修他所授课程的学生学号及其成绩;班主任想查询某个学生选修的课程号及相应成绩;学生想查询自己某门课程的成绩。如果用户对数据库的操作在系统运行时才能提出来,则要用到嵌入式 SQL 的动态技术才能实现。

动态 SQL 方法允许在程序运行过程中根据以下情况,临时"组装"SQL 语句:

① 语句可变——允许用户在程序运行时临时输入完整的 SQL 语句。

② 条件可变——WHERE 条件和 HAVING 短语。

③ 数据库对象、查询条件均可变——SELECT 中的列名,FROM 中的表名或视图名,WHERE 和 HAVING 中的条件都可能不确定。

如果使用动态 SQL,程序就能够在运行时以字符串的形式生成 SQL 查询语句,然后立即执行该查询或使其为后续使用做好准备。

例 3.10.1　删除某些学生记录,删除条件由用户在程序运行时给出。

分析:本题是一个直接执行的动态 SQL 例子,只用于非查询语句的执行。无须返回查询结果。程序如下:

```
        ...
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200]; /* 定义宿主变量 sqlstring */
EXEC SQL END DECLARE SECTION;
char cond[150];
/* 填入 SQL 语句的固定部分 */
strcpy(sqlstring,"DELETE FROM STUDENT WHERE")
```

```
/* 提示用户输入删除条件 */
printf("Enter delete condition:");
scanf("%s",cond); /* 临时输入删除条件 */
strcat(sqlstring, cond); /* 输入的条件加在 sqlstring 后面 */
/* 执行 sqlstring 中的 SQL 语句 */
EXEC SQL EXECUTE IMMEDIATE: sqlstring;
        …
```

例 3.10.2 删除某些学生记录。

分析：这是一个带动态参数的动态 SQL，也用于非查询语句的执行。这类 SQL 语句中，含有未定义的变量，仅起占位符的作用。执行前，程序提示用户输入相应的参数，以取代这些占位用的变量。也可有多个占位符，执行时根据语句中出现的顺序，依次用 USING 后的宿主变量取代。程序如下：

```
        …
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200]; /* 定义宿主变量 sqlstring */
In birth-year;
EXEC SQL END DECLARE SECTION;
strcpy(sqlstring,"DELETE FROM STUDENT WHERE YEAR(BDATE)<=:y;")
/* 提示用户输入参数 birth-year,用以指明删除何年以前出生的学生记录 */
printf("Enter birth-year for deleting:");
scanf("%d",&birth-year); /* 临时输入出生年份 */
/* 用 PREPARE 语句定义 sqlstring 中的 SQL 语句为命令 PURGE */
EXEC SQL PREPARE PURGE FROM: sqlstring;
/* 用参数 birth-year 取代 y 执行命令 PURGE */
EXEC SQL EXECUTE PURGE USING: birth-year;
        …
```

例 3.10.3 查询成绩，并将查询结果排序。

分析：本题是一个查询类动态 SQL，必须返回查询结果。由于查询结果可能是一个元组或一组元组，编译时不能确定，所以动态 SQL 须一律以游标取数。程序如下：

```
        …
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200];
char SNO[7];
float GRADE;
short GRADEI;
char GIVENCNO[6];
EXEC SQL END DECLARE SECTION;
char orderby[150];
strcpy(sqlstring,"SELECT SNO,GRADE FROM SC WHERE CNO=:c");
/* 提示用户输入 ORDER BY 子句 */
printf("Enter ORDER BY clause:");
scanf("%s", orderby);
strcat(sqlstring, orderby); /* 输入的子句语句加在 sqlstring 后面 */
/* 提示用户输入要查询成绩的课程号 */
```

```
printf("Enter the course number:");
scanf(" % s", GIVENCNO);
/ * 准备查询语句 * /
EXEC SQL PREPARE query FROM: sqlstring;
/ * 说明游标 * /
EXEC SQL DECLARE grade - cursor CURSOR FOR query;
/ * 打开游标 * /
EXEC SQL OPEN grade - cursor USING: GIVENCNO;
/ * 取数 * /
While(TURE)
{
    EXEC SQL FETCH  grade - cursor
      INTO :SNO, :GRADE, :GRADEI;
    if(SQLCA.SQLCODE == 100)
      break;
    if(SQLCA.SQLCODE < 0)
      break;
/ * 以下处理从游标所取的数据,从略 * /
          …
}
/ * 关闭游标 * /
EXEC SQL CLOSE grade - cursor;
            …
```

3.11 SQL 的存储过程

存储过程是指常用的访问数据库的程序。该程序作为一个过程,经过编译后,存储在数据库中,且在数据目录中登录,供用户调用。

创建存储过程的好处是可以方便用户,即存储过程只需用户提供参数,不必编写程序;可以改善性能,即存储过程编译后存于数据库中,不必再进行语法分析、查询处理和优化;可以扩充功能,即存储过程不仅可用 SQL 语句,还可用控制程序流程的语句。

例 3.11.1 定义学生退学的处理过程。

```
EXEC SQL
   CREATE PROCEDURE drop_student
       (IN student_no CHAR(7),
        OUT message CHAR(30))
   BEGIN ATOMIC
       DELETE FROM STUDENT
       WHERE SNO = student_no;
       DELETE FROM SC
       WHERE SNO = student_no;
       SET message = student_no || 'dropped';
   END;
 EXEC SQL
```

```
/* 以上是退学处理过程的定义,下面表示应用程序调用此过程 */
    …
CALL drop_student( … )
    …
```

存储过程的定义由 EXEC SQL…EXEC SQL 界定,DBMS 也是通过它们来识别存储过程;drop_student 是退学存储过程的过程名。该过程有一个输入(IN)参数 student_no,是退学人的学号;一个输出(OUT)参数 message,是返回给用户的信息;BEGIN…END 之间是过程体,完成三件事:①从 STUDENT 表中删除指定学生(由 student_no 决定)对应的元组;②从 SC 表中删除该学生所选的所有课程;③输出信息"××××××dropped",其中,××××××为退学人的学号。BEGIN 后面的 ATOMIC 表示过程体执行时要保持原子性,即要么全做,要么都不做。如果不加 ATOMIC,则允许过程体有些操作完成,有些操作失败。在本过程中,由于学生退学后,必须删除其学籍和所选的课程,所以 BEGIN 后面须加 ATOMIC。

思考题

1. 用关系代数表示的查询与用 SQL 表示的查询有什么区别与联系?
2. SQL 中的嵌套查询在什么情况下使用?如何使用?
3. 存储过程有什么作用?其意义是什么?

重点内容与典型题目

重点内容

基本 SQL 查询和较复杂的 SQL 查询。

典型题目

1. 假设有三张表:水手表(Sailors)、船表(Boats)和预订船表(Reserves),如图 3.2 所示。

Sailors

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Boats

bid	bname	color
101	tiger	red
103	lion	green
105	hero	blue

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

图 3.2　典型题目 1 图

(1) Find sailors who've reserved at least one boat.

(2) Find sid's of sailors who've reserved a red or a green boat.

(3) Find sid's of sailors who've reserved a red and a green boat.

(4) Find name and age of the oldest sailor(s).

2. 给定水手表(Sailors)的实例,如图 3.3 所示。

Sailors

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

图 3.3　典型题目 2 图

查询要求:

(1) Find age of the youngest sailor with age≥18, for each rating with at least 2 such sailors.

(2) Find age of the youngest sailor with age≥18, for each rating with at least 2 such sailors and with every sailor under 60.

习题

1. SQL 是一种什么语言? 包括哪些功能?

2. 数据库语言与宿主语言有什么区别?

3. 视图的作用是什么?

4. 简述 WHERE 子句与 HAVING 子句的区别。

5. 基表与视图的区别与联系是什么?

6. 所有视图是否都可以更新,为什么?

7. 使用 SQL 如何实现各种关系代数运算?

8. C 语言程序中,嵌入式 SQL 中是如何区分 SQL 语句和宿主语言语句的?

9. 嵌入式 SQL 中是如何解决宿主语言和 DBMS 之间数据通信的?

10. 游标的作用是什么?

11. 在嵌入式 SQL 中,如何协调 SQL 的集合处理方式与宿主语言单记录处理方式的关系?

12. 嵌入式 SQL 语句中,何时需要使用游标? 何时不需要使用游标?

13. 试分析空值产生的原因。为了处理空值，DBMS 要做哪些工作？

14. 试叙述 SQL 的关系代数特点和元组演算特点。

15. 根据关系代数公式写出 SQL 语句：

(1) $\Pi_{SNO}(\sigma_{CNO='CS\text{-}110'}(SC))$；

(2) $\sigma_{GRADE\leqslant 100 \wedge GRADE\geqslant 90}(SC)$；

(3) $\Pi_{NAME}(\sigma_{GRADE\geqslant 90}(SC)\infty(STUDENT))$。

16. 要求根据表 3.2～表 3.4，写出下列的 SQL 语句和查询结果：

(1) 查询计算机系秋季所开课程的课程号和学分数；

(2) 查询选修计算机系秋季所开课程的男生的姓名、课程号、学分数、成绩；

(3) 查询至少选修一门电机系课程的女生的姓名；

(4) 查询每位学生已选课程的门数和总平均成绩；

(5) 查询有一门以上(含一门)三学分以上课程的成绩低于 70 分的学生的姓名；

(6) 查询 1974—1976 年出生的学生的学号、总平均成绩及已修学分数；

(7) 查询每个学生选课门数、最高成绩、最低成绩和平均成绩；

(8) 查询秋季有两门以上课程成绩为 90 分以上的学生的姓名；

(9) 查询选课门数唯一的学生的学号；

(10) 查询所学每一门课程成绩均高于或等于该课程平均成绩的学生的姓名及相应课程号。

17. 为什么在嵌入式 SQL 中要使用游标？

18. 宿主语言与 SQL 的数据类型有时不完全对应或等价，如何解决数据类型转换问题？