第 3 章　关系数据库标准语言 SQL

3.1　SQL 概述

3.1.1　SQL 的产生与发展

结构化查询语言(Structured Query Language,SQL)是一种介于关系代数与关系演算之间的语言。1974 年,SQL 由 Ray Boyce 和 Don Chamberlain 提出,1975—1979 年,IBM San Jose Research Lab 的关系数据库管理系统原型 System R 实施了这种语言。SQL-86 是第一个 SQL 标准(ANSI/ISO),之后还有 SQL-89、SQL-92(SQL2)、SQL-99(SQL3)、SQL2003(SQL4)。目前,大部分 RDBMS 产品都支持 SQL,它已成为操作关系数据库的标准语言。

SQL 之所以能够为用户和业界所接受,成为国际标准,是因为它是一个综合的、通用的、功能极强同时又简洁易学的语言。SQL 集数据查询、数据操纵、数据定义和数据控制功能于一身,充分体现了关系数据语言的特点与优点。

3.1.2　SQL 的基本概念及组成

支持 SQL 的 RDBMS 同样支持关系数据库 3 级模式结构,如图 3-1 所示。其中,外模式对应于视图(View)和部分基本表(Base Table),模式对应于基本表,内模式对应于存储文件。

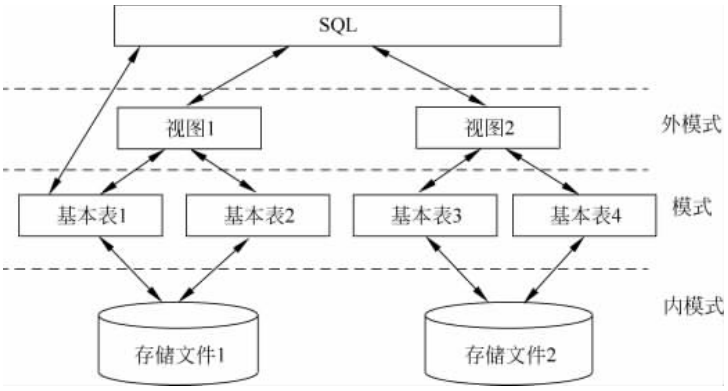

图 3-1　SQL 对 RDBS 模式的支持

用户可以用 SQL 对基本表和视图进行查询或其他操作。在用户观点上理解,基本表和视图一样,都是关系,表中的列对应关系属性,表中的行对应关系的元组。

基本表是本身独立存在的表,在 SQL 中一个关系就对应一个基本表,一个或多个基本

表对应一个存储文件，一个表可以带若干索引，索引也存放在存储文件中。

存储文件的逻辑结构组成了关系数据库的内模式。存储文件的物理结构是任意的，对用户是透明的。

视图是从一个或多个基本表中导出的表，它本身不独立存储在数据库中，即数据库中只存储视图的定义，而不存储视图对应的数据。因此，可以将其理解为一个虚表。

下面逐一介绍主要 SQL 语句的功能和使用格式。为了突出基本概念和基本功能，略去了许多语法细节。每个 RDBMS 产品在实现标准 SQL 时各有差异，与 SQL 标准的符合程度也不同。因此，具体使用某个 RDBMS 产品时，还应参阅系统提供的有关手册或联机文档。

本章用学生成绩数据库作为一个例子来讲解 SQL 的数据定义、数据查询、数据操纵和数据控制语句的具体应用。

学生成绩数据库包括以下 4 个表。

- 学生表：Student(学号，姓名，性别，出生日期，入学成绩，党员否，班级编号，简历，照片)
- 班级表：Class(班级编号，班级名称，所属专业，人数)
- 课程表：Course(课程编号，课程名称，先修课号，学时，学分)
- 选课表：Score(学号，课程编号，成绩，学期)

关系的主码用下画线表示，外码用波浪线表示。各个表中的数据示例如图 3-2 所示。

Student

学　号	姓　名	性别	出生日期	入学成绩	党员否	班级编号	简历	照片
2006091001	张楚	男	1986-01-15	545	1	200601	NULL	NULL
2006091002	欧阳佳慧	女	1987-10-12	516	0	200601	NULL	NULL
2006091003	孔灵柱	男	1986-05-21	526	1	200601	NULL	NULL
2006091004	门静涛	男	1987-04-28	530	0	200601	NULL	NULL
2006091005	王广慧	女	1986-06-26	550	1	200601	NULL	NULL
2006091006	孙晓楠	女	1987-08-16	517	1	200602	NULL	NULL
2006091007	张志平	男	1987-03-15	500	0	200602	NULL	NULL
2006091008	刘晓晓	女	1985-09-28	555	1	200602	NULL	NULL
2006091009	王大伟	男	1987-12-12	515	0	200603	NULL	NULL
2006091010	谢辉	男	1986-10-10	544	0	200603	NULL	NULL

Class

班级编号	班级名称	所属专业	人数
200601	工商管理 061	工商管理	30
200602	财务管理 062	财务管理	35
200603	信息管理 063	信息管理	31

Course

课程编号	课程名称	先修课号	学时	学分
04010101	管理学	04010103	64	4
04010102	数据库系统	NULL	48	3
04010103	统计学	04010102	50	3
04010104	技术经济学	04010101	45	2.5

Score

学　号	课程编号	成绩	学　期
2006091001	04010101	92	200620071
2006091001	04010102	84	200620072
2006091001	04010103	54	200620072
2006091001	04010104	NULL	200620072
2006091002	04010101	86	200620072
2006091002	04010102	90	200620072
2006091002	04010103	67	200620072
2006091003	04010101	74	200620071
2006091003	04010102	45	200620072
2006091004	04010101	72	200620071
2006091005	04010101	56	200620071

图 3-2　学生成绩数据库的数据示例

3.1.3 SQL 的特点

1. 综合统一

SQL 集数据定义语言(DDL)、数据操纵语言(DML)、数据控制语言(DCL)的功能于一体,语言风格统一,可以独立完成数据库生命周期中的全部活动,包括定义关系模式、建立数据库、查询、更新、维护、数据库重构、数据库安全控制等一系列操作要求,这就为数据库应用系统开发提供了良好的环境。

2. 高度非过程化

非关系数据模型的数据操纵语言是面向过程的语言,用其完成某项请求,必须指定存取路径。而用 SQL 进行数据操作,用户只需提出“做什么”,而不必指明“怎么做”,因此用户无须了解存取路径,存取路径的选择以及 SQL 语句的操作过程由 DBMS 自动完成。这不但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

非关系数据模型采用的是面向记录的操作方式,操作对象是一条记录。例如,查询所有平均成绩在 80 分以上的学生姓名,用户必须一条一条地把满足条件的学生记录找出来(通常要说明具体处理过程及存取路径)。而 SQL 采用集合操作的方式,不仅操作对象、查询结果可以是元组的集合,而且一次插入、删除、更新操作的对象也可以是元组的集合。

4. 以同一种语法结构提供两种使用方式

SQL 既是自含式语言,又是嵌入式语言。

作为自含式语言,SQL 能够独立地用于联机交互的使用方式,用户可以在终端键盘上直接键入 SQL 命令对数据库进行操作;作为嵌入式语言,SQL 语句能够嵌入到高级语言(如 C、C++、Java)程序中,供程序员设计程序时使用。而在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做法,提供了极大的灵活性与方便性。

5. 语言简洁、易学易用

SQL 功能极强,但由于设计巧妙,语言十分简洁,完成数据定义、数据查询、数据操纵、数据控制的核心功能只用了 9 个动词,见表 3-1。而且 SQL 语法简单,接近英语口语,因此容易学习、容易使用。

表 3-1 SQL 动词

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

3.2 数据定义

关系数据库系统支持 3 级模式结构,其模式、外模式和内模式中的基本对象有基本表、视图和索引。因此,SQL 的数据定义功能包括基本表的定义、视图的定义和索引的定义。SQL 数据定义语句见表 3-2。

表 3-2 SQL 数据定义语句

操作对象	操作方式		
	创　建	删　除	修　改
基本表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

本节只介绍如何定义基本表和索引，视图的概念及其定义方法将在第 3.5 节专门介绍。

3.2.1 基本表

1. 定义基本表

SQL 使用 CREATE TABLE 语句定义基本表，其基本格式如下：

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

其中，<表名>是所要定义的基本表的名字；<列名>是组成该表的各个属性（列）；<数据类型>用来实现域的概念，限制列的取值范围及运算；<列级完整性约束条件>是指涉及相应属性列的完整性约束条件；<表级完整性约束条件>是指涉及一个或多个属性列的完整性约束条件。

列级完整性约束主要包括：

- 主码约束：PRIMARY KEY
- 唯一性约束：UNIQUE
- 非空值约束：NOT NULL
- 参照完整性约束：FOREIGN KEY REFERENCES <被参照表名> (<主码>)
- 域完整性约束：CHECK (<条件>)

表级完整性约束包括：

- 主码约束：PRIMARY KEY(<列组>)
- 唯一性约束：UNIQUE(<列组>)
- 参照完整性约束：FOREIGN KEY (<外码>) REFERENCES <被参照表名> (<主码>)
- 域完整性约束：CHECK (<条件>)

上述各种约束在定义时均可选择在前面加上 CONTRAINT <约束名>子句来指定约束名。约束名用来标识一个特定的约束，在一个特定模式（数据库）中的约束名必须是唯一的。

如果完整性约束条件涉及该表的多个属性列，则必须定义在表级上，否则既可以定义在列级上，也可以定义在表级上。在一个基本表中只能定义一个 PRIMARY KEY 约束，但可定义多个 UNIQUE 约束；对于指定为 PRIMARY KEY 的一个列或多个列的组合，其中任何一个列都不能出现空值，而对于 UNIQUE 约束的唯一键，则允许为空。不能为同一个列或一组列既定义 UNIQUE 约束，又定义 PRIMARY KEY 约束。

在 SQL 中，域的概念用数据类型来实现。定义表的各个属性列时需要指明其数据类型

及长度(或精度)。SQL 提供的主要数据类型见表 3-3。注意,不同的 RDBMS 支持的数据类型不尽相同。

表 3-3 SQL 提供的主要数据类型

数 据 类 型	含 义
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
INT	长整数(也可以写作 INTEGER)
SMALLINT	短整数
NUMERIC(p,d)	定点数,由 p 位数字(不包括符号、小数点)组成,小数后面有 d 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
FLOAT(n)	浮点数,精度至少为 n 位数字
BOOLEAN	逻辑布尔量(真 True/假 False)
DATE	日期,包含年、月、日,格式为 YYYY-MM-DD
TIME	时间,包含一日的时、分、秒,格式为 HH:MM:SS
CLOB	CHARACTER LARGEOBJECT 用来存放大文本值(如文档)
BLOB	BINARY LARGE OBJECT 用来存放大二进制值(如图像)

【例 3-1】 建立一个学生表 Student。其中学号为主码,班级编号为外码,并且要求姓名取值唯一,性别取值只能是“男”或“女”。

```
CREATE TABLE Student(
学号          CHAR(10) PRIMARY KEY,
姓名          CHAR(10) CONSTRAINT S1 UNIQUE,
性别          CHAR(2) CHECK(性别 in(男,女)),
出生日期      DATE,          -- 在 SQL Server 环境下应为 DATETIME 类型
入学成绩      INT,
党员否        BOOLEAN,       -- 在 SQL Server 环境下应为 BIT 类型
班级编号      CHAR(6) FOREIGN KEY REFERENCES Class(Sno),
简历          CLOB,          -- 在 SQL Server 环境下应为 TEXT 类型
照片          BLOB);         -- 在 SQL Server 环境下应为 IMAGE 类型
```

【例 3-2】 建立一个选课表 Score,其中(学号,课程编号)为主码,学号和课程编号分别为外码。

```
CREATE TABLE Score(
    学号        CHAR(10),
    课程编号    CHAR(8),
    成绩        SMALLINT,
    学期        CHAR(9),
    PRIMARY KEY (学号,课程编号),
    FOREIGN KEY(学号) REFERENCES Student(学号),
    FOREIGN KEY(课程编号) REFERENCES Coure(课程编号),
    CHECK(成绩>=0 and 成绩<=100));
```

2. 修改基本表

随着应用环境和应用需求的变化,有时需要修改已建立好的基本表,SQL 用 ALTER

TABLE 语句修改基本表，其一般格式为：

```
ALTER TABLE <表名>
    [ ADD COLUMN <新列名> <数据类型> [ 完整性约束 ] ]
    [ DROP COLUMN <列名> ]
    [ DROP CONSTRAINT <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 COLUMN 子句用于增加新列和新的完整性约束条件，DROP COLUMN 子句用于删除指定的列，DROP CONSTRAINT 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，包括列名和数据类型。

【例 3-3】 向 Student 表增加“入学时间”列，其数据类型为日期型。

```
ALTER TABLE Student ADD 入学时间 DATE;
```

注意：不论基本表中原来是否已有数据，新增加的列一律被赋予一个空值(NULL)。

【例 3-4】 删除 Student 表中的“入学时间”列。

```
ALTER TABLE Student DROP COLUMN 入学时间;
```

【例 3-5】 删除 Student 表中姓名必须取唯一值的约束。

```
ALTER TABLE Student DROP CONSTRAINT S1;
```

【例 3-6】 将 Student 表中“入学成绩”的数据类型改为半字长整数。

```
ALTER TABLE Student ALTER COLUMN 入学成绩 SMALLINT;
```

注意：修改原有的列定义有可能会破坏已有的数据。

3. 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，其一般格式如下：

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

其中，若选择 RESTRICT 选项，表示有条件删除，即欲删除的基本表不能被其他表的约束所引用(如 CHECK、FOREIGN KEY 等约束)，不能有基于此表的视图、触发器、存储过程或函数等。如果存在着这些依赖表的对象，则此表不能被删除。若选择 CASCADE，则该表的删除没有限制条件。在删除基本表的同时，相关的依赖对象(如视图等)都将被一起删除。默认是 RESTRICT 选项。

【例 3-7】 删除 Student 表。

```
DROP TABLE Student CASCADE;
```

3.2.2 索引

索引是加快查询速度的有效手段，用户可以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，加快查询速度。

一般来说，建立与删除索引由数据库管理员(DBA)或表的属主(OWNER)，即建立表的人负责完成。系统在存取数据时会自动选择合适的索引作为存取路径，用户不必显式地选

择索引。

1. 建立索引

建立索引的语句格式如下：

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

其中，<表名>是要建索引的基本表的名字。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。每个<列名>之后还可以用<次序>指定索引值的排列次序，可选 ASC(升序)或 DESC(降序)，默认值为 ASC。

UNIQUE 选项表示要建立**唯一索引**，此索引的每个索引值只对应唯一的数据记录。

CLUSTER 选项表示要建立的索引是**聚簇索引**。聚簇索引是指所引项的顺序和表中记录的物理顺序一致的索引组织。

缺省 UNIQUE 和 CLUSTER 选项时，表示要建立**非唯一索引**，即普通索引。

【例 3-8】 为学生成绩数据库中的 Student、Course、Score 3 个表建立索引。其中，Student 表按姓名升序建普通索引，Course 表按课程名升序建唯一索引，Score 表按学号升序和课程编号降序建聚簇索引。

```
CREATE INDEX St_Id_name ON Student(姓名);
CREATE UNIQUE INDEX Co_Id_name ON Course(课程名);
CREATE CLUSTER INDEX SC_Id_no ON Score(学号,课程编号 DESC);
```

注意：

- 对于已含重复值的属性列，不能建 UNIQUE 索引。
- 对某个列建立 UNIQUE 索引后，插入新记录时，DBMS 会自动检查新记录在该列上是否取了重复值。这相当于增加了一个 UNIQUE 约束。
- 在一个基本表上最多只能建立一个聚簇索引。可以在最经常查询的列上建立聚簇索引，以提高查询效率。而对于经常更新的列，则不宜建立聚簇索引。

2. 删除索引

索引一经建立，就由系统使用和维护它，不须用户干预。建立索引是为了减少查询操作的时间。如果数据增、删、改频繁，系统就会花费许多时间来维护索引，从而降低了查询效率。因此，有时需要删除一些不必要的索引，以提高系统效率。删除索引时，系统会从数据字典中删去有关该索引的描述。

删除索引使用 DROP INDEX 语句，其一般格式如下：

```
DROP INDEX <索引名>;
```

【例 3-9】 删除 Student 表的 St_Id_name 索引。

```
DROP INDEX St_Id_name;
```

3.3 数据更新

数据更新操作有 3 种：向表中添加若干行数据、修改表中的数据和删除表中的若干行数据。SQL 提供了相应的插入(INSERT)、修改(UPDATE)和删除(DELETE)3 类语句。

3.3.1 插入数据

插入语句 INSERT 通常有两种形式：一种是插入单个元组；另一种是插入子查询结果。后者可以一次插入多个元组。

1. 插入单个元组

语句格式为：

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

其功能是将新元组插入指定表中。其中，新元组的<属性列 1>的值为<常量 1>，<属性列 2>的值为<常量 2>，以此类推。INTO 子句中没有出现的属性列，新元组在这些列上将取空值（NULL）。但必须注意的是，在表定义时指定了 NOT NULL 约束的属性列不能取空值，否则会出错。如果 INTO 子句中没有指定任何属性列名，则要求 VALUES 子句提供的常量值的顺序、个数、数据类型应该与待插入数据表的属性列的顺序、个数、数据类型完全一致。

【例 3-10】 将一个新课程记录（课程编号：04010105；课程名称：运筹学；学时：64；学分：4）插入到 Course 表中。

```
INSERT INTO Course(课程编号,课程名称,学时,学分)
VALUES ('04010105','运筹学',64,4);
```

或

```
INSERT INTO Course
VALUES ('04010105','运筹学',NULL,64,4); /* 先修课号为 NULL */
```

【例 3-11】 插入一条选课记录（'04010104','2006091002'）。

```
INSERT INTO Score(课程编号,学号)
VALUES ('04010104','2006091002');
```

或

```
INSERT INTO Score
VALUES ('2006091002','04010104',NULL,NULL);
```

注意：属性列的顺序可与表定义中的顺序不一致，此时一定要在 INTO 子句中指定属性列名。

2. 插入子查询结果

语句格式为：

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

其功能是将子查询的结果插入到指定表中。同样要求子查询结果列与 INTO 子句的属性列名匹配。

【例 3-12】 对每个班,求学生的平均入学成绩,并把结果存入数据库中。

首先在数据库中建立一个新表(Avg_score),其中一列存放班级编号,另一列存放平均入学成绩。

```
CREATE TABLE Avg_score(
    班级编号 CHAR(6),
    平均入学成绩 INT);
```

然后对 Student 表按班分组求平均入学成绩,再把平均入学成绩插入新表中。

```
INSERT INTO Avg_score (班级编号,平均入学成绩)
SELECT 班级编号,AVG(入学成绩)
FROM Student
GROUP BY 班级编号;
```

3.3.2 修改数据

修改又称为更新,其语句格式如下:

```
    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 子句给出<表达式>的值用于取代相应的属性列值。缺省 WHERE 子句表示要修改表中的所有元组。

1. 修改一个元组的值

【例 3-13】 将学生 2006091010 的出生日期改为 1987 年 10 月 10 日。

```
UPDATE Student
SET 出生日期 = '1987 - 10 - 10'
WHERE 学号 = '2006091010';
```

2. 修改多个元组的值

【例 3-14】 将所有学生党员的入学成绩增加 10 分。

```
UPDATE Student
SET 入学成绩 = 入学成绩 + 10
WHERE 党员否 = 1;
```

3. 带子查询的修改语句

子查询也可以嵌套在更新语句的 WHERE 子句中,用以构造修改的条件。

【例 3-15】 将 200601 班全体学生的选修课程成绩置零。

```
UPDATE Score
SET 成绩 = 0
WHERE '200601' = (SELECT 班级编号
                  FROM Student
                  WHERE Student.学号 =  Score.学号);
```

3.3.3 删除数据

删除语句的一般格式为：

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

该语句的功能是删除指定表中满足 WHERE 子句条件的所有元组。如果省略 WHERE 子句，表示要删除表中的所有元组，但表的定义仍在数据字典中，即 DELETE 语句删除的是表中的数据，不是关于表的定义。

1. 删除一个元组的值

【例 3-16】 删除课程编号为 04010105 的课程记录。

```
DELETE
FROM Course
WHERE 课程编号 = '04010105';
```

2. 删除多个元组的值

【例 3-17】 删除所有学生的选课记录。

```
DELETE
FROM Score;
```

3. 带子查询的删除语句

子查询同样可以嵌套在删除语句的 WHERE 子句中，用以构造执行删除操作的条件。

【例 3-18】 删除 200601 班所有学生的选课记录。

```
DELETE
FROM Score
WHERE '200601' = (SELECT 班级编号
      FROM Student
      WHERE Student.学号 = Score.学号);
```

对基本表中的数据进行的插入、修改和删除操作有可能会破坏在表上已定义的完整性规则，第 6.2 节将详细介绍如何进行完整性检查与控制。

3.4 数据查询

SQL 提供了 SELECT 语句进行数据的查询，该语句具有灵活的使用方式和丰富的功能。其一般格式为：

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

其中：

- SELECT 子句：指定要显示的属性列，实现关系代数中的投影操作。
- FROM 子句：指定查询对象（基本表或视图），当指定多个查询对象时，实现连接操作。
- WHERE 子句：指定查询条件，实现关系代数的选择操作。
- GROUP BY 子句：对查询结果按指定列的值分组，该属性列值相等的元组为一个组。通常会在每组中作用集函数。
- HAVING 短语：筛选出只满足指定条件的组。
- ORDER BY 子句：对查询结果表按指定列值升序（ASC）或降序（DESC）排序。
- SELECT … FROM …是最基本的查询语句（必选）。

整个语句的含义是根据 WHERE 子句的条件表达式，从 FROM 子句指定的基本表或视图中找出满足条件的元组，再按 SELECT 子句中的目标列表达式选出元组中的属性值形成结果表。如果有 GROUP BY 子句，则将结果按<列名 1>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。如果 GROUP BY 子句带有 HAVING 短语，则只有满足指定条件的组才予以输出。如果有 ORDER BY 子句，则结果表还要按<列2>的值升序或降序排序。

SELECT 语句既可以完成简单的单表查询，也可以完成复杂的连接查询和嵌套查询。下面以学生成绩数据库为例，说明该语句的各种用法。

3.4.1 单表查询

单表查询是指仅涉及一个表的查询，是一种最简单的查询操作。

1. 选择表中的若干列

1）查询指定列

在很多情况下，用户只对表中的一部分属性列感兴趣，这时可以在 SELECT 子句的<目标列表达式>中指定要查询的属性列。

【例 3-19】 查询全体学生的学号与姓名。

```
SELECT 学号,姓名
FROM Student;
```

【例 3-20】 查询全体学生的姓名、学号和班级编号。

```
SELECT 姓名,学号,班级编号
FROM Student;
```

由<目标列表达式>指定的查询结果列的排列顺序可以与表中的顺序不一致。用户可以根据应用需要改变列的显示顺序。【例 3-20】中先列出姓名，再列出学号和班级编号。

2）查询全部列

将表中的所有属性列都选出来，有两种方法：一种是在 SELECT 关键字后面列出所有列名；另一种是用星号（*）表示查询表的所有列（列的显示顺序与其在基表中的顺序一致）。

【例 3-21】 查询所有课程的详细记录。

```
SELECT *
FROM Course;
```

等价于：

```
SELECT 课程编号,课程名称,先修课号,学时,学分
FROM Course;
```

3）查询经过计算的值

SELECT 子句中的<目标列表达式>不仅可以是表中的属性名，也可以是任何合法的表达式（常量、函数、算术表达式等），即可以将查询出的属性列经过一定计算后再列出结果。

【例 3-22】 查询全体学生的姓名、年龄和入学成绩。

```
SELECT 姓名,2012 - year(出生日期),入学成绩
FROM Student;
```

查询结果中的第二列不是列名，而是一个表达式，是用当时的年份（假设为 2012 年）减去学生出身的年份，计算出学生的年龄。其中函数 year( )返回年份。其输出结果为：

姓名	2012 - year(出生日期)	入学成绩
张楚	26	545
欧阳佳慧	25	516
孔灵柱	26	526
门静涛	25	530
王广慧	26	550
孙晓楠	25	517
张志平	25	500
刘晓晓	27	555
王大伟	25	515
谢辉	26	544

在 SQL 中，对于查询结果表中出现的任何属性列，均可通过指定别名的方式对其重命名，来改变查询结果的列标题。具体格式为：

```
<目标列表达式> AS <别名>
```

如【例 3-22】可以改写为：

```
SELECT 姓名,2012 - year(出生日期) AS 年龄,入学成绩
FROM Student;
```

其输出结果为：

姓名	年龄	入学成绩
张楚	26	545
欧阳佳慧	25	516
孔灵柱	26	526
⋮	⋮	⋮

2. 选择表中的若干元组

1）消除取值重复的行

两个本来并不完全相同的元组，投影到指定的某些列上以后，就可能变成相同的行了。

在SELECT子句中选择使用DISTINCT短语去掉结果表中的重复行，若不指定DISTINCT短语，则保留结果表中的重复行(默认为ALL短语)。

【例3-23】 查询选修了课程的学生学号。

```
SELECT DISTINCT 学号
FROM Score;
```

输出结果为：

```
学号
----------
2006091001
2006091002
2006091003
2006091004
2006091005
```

而命令：

```
SELECT 学号 FROM Score;
```

等价于：

```
SELECT ALL 学号 FROM Score;
```

2) 查询满足条件的元组

查询满足条件的元组，即选择操作，可以通过WHERE子句实现。常用的查询条件见表3-4。

表3-4 常用的查询条件

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

(1) 比较大小。在WHERE子句的<条件表达式>中使用表3-4中的比较运算符：

=(等于)，>(大于)，<(小于)，>=(大于等于)，<=(小于等于)，!=或<>(不等于)，!>(不大于)，!<(不小于)。

【例3-24】 查询考试成绩不及格的学生的学号。

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

或

```
SELECT DISTINCT 学号
FROM Score
WHERE NOT 成绩>=60;
```

(2) 确定范围。使用谓词 BETWEEN … AND …和 NOT BETWEEN … AND …可以查找属性值在(或不在)指定范围内的元组,其中 BETWEEN 后是范围的下限(即最低值),AND 后是范围的上限(即最高值)。

【例 3-25】 查询入学成绩为 500～530(包括 500 和 530)的学生信息。

```
SELECT *
FROM Student
WHERE 入学成绩 BETWEEN 500 AND 530;
```

【例 3-26】 查询出生日期不在 1985 年 1 月 1 日至 1986 年 1 月 1 日之间的学生姓名、性别和出生日期。

```
SELECT 姓名,性别,出生日期
FROM Student
WHERE 出生日期 NOT BETWEEN'1985-01-01' AND '1986-01-01';
```

(3) 确定集合。使用谓词 IN(<值表>)和 NOT IN(<值表>)查找属性值属于(或不属于)指定集合的元组。其中,<值表>是用逗号分隔的一组离散值。

【例 3-27】 查询 200601、200602 和 200603 班学生的姓名和性别。

```
SELECT 姓名,性别
FROM Student
WHERE 班级编号 IN ( '200601','200602','200603' );
```

【例 3-28】 查询既不是 200601 班,也不是 200602 班学生的姓名和性别。

```
SELECT 姓名,性别
FROM Student
WHERE 班级编号 NOT IN ( '200601','200602' );
```

(4) 字符串匹配。使用谓词[NOT] LIKE '<匹配串>'[ESCAPE '<换码字符>']可以实现模糊查询。其中,<匹配串>指固定字符串或含通配符的字符串,当<匹配串>为固定字符串时,可以用＝运算符取代 LIKE 谓词;用!＝或<>运算符取代 NOT LIKE 谓词。

通配符有两种。

- %(百分号):代表任意长度(长度可以为 0)的字符串。例如,'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。
- _(下画线):代表任意单个字符。例如,'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。

当用户要查询的字符串本身就含通配符%或_时,要使用 ESCAPE '<换码字符>'短语对通配符进行转义。例如:LIKE'DB_Design'ESCAPE'\'中,通配符_被换码字符\转义为普通字符,满足条件的字符串为'DB_Design'。

【例 3-29】 查询所有姓刘的学生姓名、学号和性别。

```
SELECT 姓名,学号,性别
```

```
FROM Student
WHERE 姓名 LIKE '刘%';
```

【例 3-30】 查询姓“欧阳”且全名为 3 个汉字的学生的姓名。

```
SELECT 姓名
FROM Student
WHERE 姓名 LIKE '欧阳_ _';
```

【例 3-31】 查询名字中第 2 个字为“阳”字的学生的姓名和学号。

```
SELECT 姓名,学号
FROM Student
WHERE 姓名 LIKE '_ _阳%';
```

【例 3-32】 查询所有不姓刘的学生姓名。

```
SELECT 姓名
FROM Student
WHERE 姓名 NOT LIKE '刘%';
```

【例 3-33】 查询名为 DB_Design 课程的课程编号和学分。

```
SELECT 课程编号,学分
FROM Course
WHERE 课程名 LIKE 'DB\_Design' ESCAPE '\'
```

【例 3-34】 查询以 DB_开头,且倒数第 3 个字符为 i 的课程的详细情况。

```
SELECT *
FROM Course
WHERE 课程名 LIKE 'DB\_%i_ _' ESCAPE '\';
```

(5) 涉及空值的查询。SQL 允许使用 NULL 值表示关于某属性值的信息缺失。使用谓词 IS NULL 或 IS NOT NULL 来判断属性值为空或非空。注意,IS NULL 不能用 = NULL 代替。

如果算术运算的输入有一个是空值,则该算术表达式(例如,包括+、-、*、/)的结果是空;如果有空值参与比较运算,SQL 将比较运算的结果看成是 unknown(既不是 **IS NULL**,也不是 **IS NOT NULL**)。unknown 是 SQL: 1999 中引入的新的布尔(Boolean)类型的数据。有 unknown 值参与的逻辑运算结果见表 3-5。

表 3-5 有 unknown 值参与的逻辑运算结果

结果值 / 表达式	A=True	A=False	A=unknown
unknown AND A	unknown	False	unknown
unknown OR A	True	unknown	unknown
NOT A	False	True	unknown

因此,WHERE 子句中的<条件表达式>可以使用 AND、OR、NOT 等逻辑运算符处理 unknown 值,如果某元组使<条件表达式>的值为 False 或 unknown,那么该元组就不会添

加到查询结果中去。

【例 3-35】 某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩（为NULL）。查询缺少成绩的学生的学号和相应的课程编号。

```
SELECT 学号,课程编号
FROM Score
WHERE 成绩 IS NULL;
```

【例 3-36】 查询2011—2012第二学期(201120122)所有选修成绩不及格的学生学号、课程编号及成绩。

```
SELECT 学号,课程编号,成绩
FROM Score
WHERE 成绩< 60 and 学期 = '201120122';
```

注意：当某个学生的成绩为NULL时，则表达式“成绩< 60 and 学期＝'201120122'”的运算结果为unknown，所以该学生未被列入查询结果中。

(6) 多重条件查询。用逻辑运算符AND和OR联结多个查询条件（AND的优先级高于OR，可以用括号改变优先级），可实现多种其他谓词查询功能（如[NOT] IN，[NOT] BETWEEN … AND …）。

【例 3-37】 查询200601班学生党员的名单。

```
SELECT 姓名
FROM Student
WHERE 班级编号 = '200601' AND 党员否 = 1;
```

改写【例3-37】，查询入学成绩为500～530（包括500和530）的学生信息。

```
SELECT *
FROM Student
WHERE 入学成绩> = 500 AND 入学成绩< = 530
```

3. 对查询结果排序

可以使用ORDER BY子句对查询结果按一个或多个属性列升序（ASC）或降序（DESC）排序，默认值为升序。

当有多个排序列时，则先按第一列排序；当第一列值相同时，再按第二列排序，以此类推。

【例 3-38】 查询2006091001号学生选修课程的课程编号及其成绩，查询结果按成绩降序排列。

```
SELECT 课程编号,成绩
FROM Score
WHERE 学号 = '2006091001'
ORDER BY 成绩 DESC;
```

【例 3-39】 查询全体学生情况，查询结果按所在班级编号升序排列，同班中的学生按姓名降序排列。

```
SELECT *
```

```
FROM Student
ORDER BY 班级编号,姓名 DESC;
```

4. 使用聚集函数

为了便于数据统计,增强查询功能,SQL 提供了 6 类聚集函数,其格式与功能见表 3-6。

表 3-6 聚集函数的格式与功能

聚集函数	功 能
COUNT(*)	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(该列必须是数值型)
AVG([DISTINCT|ALL] <列名>)	计算一列值的平均值(该列必须是数值型)
MAX([DISTINCT|ALL] <列名>)	求一列值中的最大值(该列可为数值型、日期型、字符型)
MIN([DISTINCT|ALL] <列名>)	求一列值中的最小值(该列可为数值型、日期型、字符型)

这些聚集函数可以用在 SELECT 子句或 HAVING 子句中。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 ALL 短语,则表示不取消重复值,ALL 为默认值。

另外,聚集函数根据以下原则处理空值:除 COUNT(*)外,所有的聚集函数都忽略输入集合中的空值。空值被忽略,有可能造成参加函数运算的输入集合为空集。规定空集的 COUNT 运算值为 0,其他所有聚集函数在输入为空集的情况下返回一个空值。

【例 3-40】 查询学生总人数。

```
SELECT COUNT ( * ) as 总人数
FROM Student;
```

【例 3-41】 查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT 学号) as 选课人数
FROM Score;
```

注:用 DISTINCT 以避免重复计算学生人数。

【例 3-42】 计算 04010101 号课程的学生平均分、最高分及最低分。

```
SELECT AVG(成绩) as 平均分,MAX(成绩) as 最高分,MIN(成绩) as 最低分
FROM Score
WHERE 课程编号 = '04010101';
```

5. 对查询结果分组

有时我们不仅希望将聚集函数作用在单个元组集上,而且也希望将其作用在一组元组集上。在 SQL 中,可用 GROUP BY 子句实现这个愿望。GROUP BY 子句中的一个或多个属性是用来构造分组的,在 GROUP BY 子句中的所有属性上具有相同值的元组将被分到一个组中。如果未对查询结果分组,聚集函数将作用于整个查询结果(单个元组);如果使用 GROUP BY 子句对查询结果分组后,聚集函数将分别作用于每个组。

【例 3-43】 求各个课程编号及相应的选课人数。

```
SELECT 课程编号,COUNT(学号) as 选课人数
FROM Score
GROUP BY 课程编号;
```

查询结果如下。

```
课程编号      选课人数
-------------------
04010101      5
04010102      3
04010103      2
04010104      1
```

注意：使用 GROUP BY 子句后，SELECT 子句的属性名列表中只能出现分组属性和聚集函数。

有时对分组限定条件比对元组限定条件更有用。使用 HAVING 子句对分组进行筛选，只有满足 HAVING 子句指定条件的分组才会输出。

【例 3-44】 查询选修了 2 门以上课程的学生学号。

```
SELECT 学号
FROM Score
GROUP BY 学号 HAVING COUNT( * ) > 2;
```

【例 3-45】 查询有 2 门以上课程是 80 分以上的学生的学号及课程门数。

```
SELECT 学号, count( * ) AS 课程门数
FROM Score
WHERE 成绩>= 80
GROUP BY 学号 HAVING COUNT( * )>= 2;
```

查询结果如下。

```
学号          课程门数
--------------------
2006091002 2
```

值得说明的是，如果在同一个查询语句中同时存在 WHERE 子句和 HAVING 子句，那么 SQL 首先应有 WHERE 子句中的条件，满足条件的元组通过 GRPOUP BY 子句形成分组。若 HAVING 子句存在，将作用于每个分组，不符合条件的分组将被抛弃，剩余的组被 SELECT 子句用来产生查询结果元组。

3.4.2 连接查询

若一个查询同时涉及两个以上的表，则称之为连接查询。用来连接两个表的条件称为连接条件或连接谓词。通过连接操作可查询出存放在多个表中的不同实体的信息。连接操作给用户带来很大的灵活性。

连接可以在 SELECT 语句的 FROM 子句或 WHERE 子句中建立，而在 FROM 子句中指出连接时有助于将连接操作与 WHERE 子句中的搜索条件区分开来。

SQL-92 标准定义的 FROM 子句的连接语法格式为：

```
FROM join_table join_type join_table [ON (join_condition)]
```

其中，join_table 指出参与连接操作的表名，可以用 AS 指定表别名。连接可以对同一个表操作，也可以对多表操作，对同一个表操作的连接又称为自身连接。

join_type 指出连接类型，可分为 3 种：内连接、外连接和交叉连接。

内连接(INNER JOIN 或 JOIN)使用比较运算符进行表间某(些)列数据的比较操作，并列出这些表中与连接条件相匹配的数据行。根据所使用的比较方式不同，内连接又分为等值连接、自然连接和不等连接 3 种。

外连接分为左外连接(LEFT OUTER JOIN 或 LEFT JOIN)、右外连接(RIGHT OUTER JOIN 或 RIGHT JOIN)和全外连接(FULL OUTER JOIN 或 FULL JOIN)3 种。与内连接不同的是，外连接不是列出与连接条件相匹配的行，而是列出左表(左外连接时)、右表(右外连接时)或两个表(全外连接时)中所有符合搜索条件的数据行。

交叉连接(CROSS JOIN)等价于没有连接条件的内连接，它返回连接表中所有数据行的笛卡儿积。

连接操作中的 ON (join_condition)子句指出连接条件，它由被连接表中的列和比较运算符、逻辑运算符等构成。连接条件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，但不必是相同的。

1. 内连接

内连接查询操作列出与连接条件匹配的元组，它使用比较运算符比较连接字段的值。内连接分 3 种。

(1) 等值连接：在连接条件中使用等号(=)运算符比较连接字段的值，其查询结果中列出被连接表中的所有列，包括其中的重复列。

【例 3-46】 查询每个学生及其选修课程的情况。

学生情况存放在 Student 表中，学生选课情况存放在 Score 表中，所以该查询涉及两个表，这两个表之间的联系是通过公共属性“学号”实现的。

```
SELECT s.* , sc.*
FROM Student AS s JOIN Score AS sc ON s.学号 = sc.学号;
```

或

```
SELECT s.* , sc.*
FROM Student AS s , Score AS sc
WHERE s.学号 = sc.学号;
```

注意：任何子句中引用两个表中的同名属性时，都必须加表名前缀，这是为了避免混淆。引用唯一属性名时可以加，也可以省略表名前缀。

(2) 自然连接：在连接条件中使用等号(=)运算符比较连接字段的值，但它使用选择列表指出查询结果集合中包括的列，并删除连接表中的重复列，即在等值连接中把目标列中的重复属性列去掉。

【例 3-47】 对【例 3-46】用自然连接完成。

```
SELECT s.学号,姓名,性别,出生日期,课程编号,成绩,学期
FROM Student AS s JOIN Score AS sc ON s.学号 = sc.学号;
```

查询结果如下：

学号	姓名	性别	出生日期	课程编号	成绩	学期
2006091001	张楚	男	1986-01-15	04010101	92	200620071
2006091001	张楚	男	1986-01-15	04010102	84	200620072
2006091001	张楚	男	1986-01-15	04010103	54	200620072
2006091001	张楚	男	1986-01-15	04010104	NULL	200620072
2006091002	欧阳佳慧	女	1987-10-12	04010101	86	200620072
2006091002	欧阳佳慧	女	1987-10-12	04010102	90	200620072
2006091002	欧阳佳慧	女	1987-10-12	04010103	67	200620072
2006091003	孔灵柱	男	1986-05-21	04010101	74	200620071
2006091003	孔灵柱	男	1986-05-21	04010102	45	200620072
2006091004	门静涛	男	1987-04-28	04010101	72	200620071
2006091005	王广慧	女	1986-06-26	04010101	56	200620071

连接操作不仅可以在两个表之间进行，也可以是一个表与自己连接，称为表的自身连接。需要给表起别名以示区别，由于所有属性名都是同名属性，因此必须使用别名前缀。

【例 3-48】 查询每门课的间接先修课（即先修课的先修课）。

在 Course 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编号，查找它的先修课，即可得到间接先修课。为此，要将 Course 表与其自身连接，并为其取两个别名，一个为 FIRST，另一个为 SECOND。可以将 FIRST 和 SECOND 看作 Course 表的两个不同的副本。进行连接查询的 SQL 语句为：

```
SELECT FIRST.课程编号,SECOND.先修课号
FROM Course AS FIRST JOIN Course AS SECOND
    ON FIRST.先修课号 = SECOND.课程编号;
```

查询结果如下：

```
课程编号      先修课号
------------------
04010101 04010102
04010103 NULL
04010104 04010103
```

（3）不等连接：在连接条件中使用除等号（=）运算符以外的其他比较运算符比较连接字段的值。这些运算符包括>、>=、<=、<、!>、!<和<>。

2. 外连接

内连接时，返回查询结果集合中的仅是符合查询条件（WHERE 搜索条件或 HAVING 条件）和连接条件的元组。而采用外连接时，它返回到查询结果集合中的不仅包括符合连接条件的行，而且还包括左表（左外连接时）、右表（右外连接时）或两个连接表（全外连接）中的所有元组。

【例 3-49】 查询所有学生的选修课程的情况（包括未选修课程的学生信息）。

本例既可以用左外连接实现，也可以用右外连接实现，关键是看主体表（Student）放在关键字 JOIN 的哪一边。

• 使用左外连接实现。

```
SELECT s.学号,姓名,性别,课程编号,成绩
FROM Student AS s LEFT JOIN Score AS sc ON s.学号 = sc.学号;
```

查询结果如下：

学号	姓名	性别	课程编号	成绩
2006091001	张楚	男	04010101	92
2006091001	张楚	男	04010102	84
2006091001	张楚	男	04010103	54
2006091001	张楚	男	04010104	NULL
2006091002	欧阳佳慧	女	04010101	86
2006091002	欧阳佳慧	女	04010102	90
2006091002	欧阳佳慧	女	04010103	67
2006091003	孔灵柱	男	04010101	74
2006091003	孔灵柱	男	04010102	45
2006091004	门静涛	男	04010101	72
2006091005	王广慧	女	04010101	56
2006091006	孙晓楠	女	NULL	NULL
2006091007	张志平	男	NULL	NULL
2006091008	刘晓晓	女	NULL	NULL
2006091009	王大伟	男	NULL	NULL
2006091010	谢辉	男	NULL	NULL

• 使用右外连接实现。

```
SELECT s.学号,姓名,性别,课程编号,成绩
FROM Score AS sc RIGHT JOIN Student AS s ON s.学号 = sc.学号;
```

查询结果与左外连接相同。由此可见，左外连接列出左边关系(本例为 Student)中所有的元组；右外连接则是列出右边关系(本例为 Student)中所有的元组。

3. 交叉连接

交叉连接是不带连接谓词的连接，它返回被连接的两个表的广义笛卡儿积，很少使用。

例如，Student 表中有 10 行，而 Score 表中有 11 行，则下列交叉连接检索到的记录数将等于 10×11=110 行。

```
SELECT Student.* ,Score.*
FROM Student, Score
```

或

```
SELECT Student.* ,Score.*
FROM Student CROSS JOIN Score
```

4. 多表连接

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 3-50】 查询每个学生的学号、姓名、选修的课程名称及成绩。

```
SELECT Student.学号,姓名,课程名称,成绩
FROM Student JOIN Score ON Student.学号 = Score.学号
          JOIN Course ON Score.课程编号 = Course.课程编号;
```

【例 3-51】 查询选修“数据库系统及应用”课程且成绩在 80 分以上的学生的学号、姓名及成绩。

```
SELECT Student.学号,姓名,课程名称,成绩
FROM Student JOIN Score ON Student.学号 = Score.学号
          JOIN Course ON Score.课程编号 = Course.课程编号
WHERE 课程名称 = '数据库系统及应用' AND 成绩> 80;
```

3.4.3 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 子句的条件中的查询称为嵌套查询。例如：

```
SELECT 姓名                          -- 外层查询或父查询
FROM Student
WHERE 学号 IN(
            SELECT 学号              -- 内层查询或子查询
            FROM Score
            WHERE 课程编号 = '04010101');
```

本例中，在谓词 IN 后边的查询块(SELECT 学号 FROM Score WHERE 课程编号＝'04010101')称为子查询(内层查询)，而外层查询块称为父查询(外层查询)。

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。需要特别指出的是，子查询中不能使用 ORDER BY 子句，ORDER BY 子句只能对最终结果排序。

嵌套查询使我们可以用多个简单查询构成复杂的查询，从而增强 SQL 的查询能力。以层层嵌套的方式来构造程序正是 SQL“结构化”的含义所在。

1. 带有 IN 谓词的子查询

【例 3-52】 查询与“张楚”在同一班学习的学生。

先分步来完成此查询，然后再构造嵌套查询。

(1) 确定“张楚”所在班级编号。

```
SELECT 班级编号
FROM Student
WHERE 姓名 = '张楚';
```

结果为：

```
班级编号
---------
200601
```

(2) 查找所有在 200601 班学习的学生。

```
SELECT 学号,姓名,班级编号
FROM Student
```

```
WHERE 班级编号 = '200601'
```

结果为：

```
学号          姓名        班级编号
-----------------------------
2006091001    张楚        200601
2006091002    欧阳佳慧    200601
2006091003    孔灵柱      200601
2006091004    门静涛      200601
2006091005    王广慧      200601
```

(3) 构造嵌套查询。

将第一步查询嵌入到第二步查询的条件中，构造嵌套查询如下：

```
SELECT 学号,姓名,班级编号
FROM Student
WHERE 班级编号 IN(
                  SELECT 班级编号
                  FROM Student
                  WHERE 姓名 = '张楚')
```

本例中，子查询的查询条件不依赖于父查询，称为不相关子查询。DBMS 的一种求解方法是由里向外处理，即先执行子查询，子查询的结果用于建立父查询的查询条件，即得到如下语句：

```
SELECT 学号,姓名,班级编号
FROM Student
WHERE 班级编号 IN('200601')
```

该查询也可用自身连接来完成：

```
SELECT S1.学号,S1.姓名,S1.班级编号
FROM Student AS S1 JOIN Student AS S2 ON S1.班级编号 = S2.班级编号
WHERE S2.姓名 = '张楚';
```

可见，实现同一个查询要求可以有多种方法，当然，不同的方法其执行效率可能会有差别，甚至会影响应用程序的实用性。这就是数据库编程人员应该掌握的查询优化技术，有兴趣的读者可以参考有关文献，包括具体 DBMS 的查询优化方法。

【例 3-53】 查询选修了课程名称为“管理学”的学生学号和姓名。

```
SELECT 学号,姓名
FROM Student
WHERE 学号 IN
          (SELECT 学号
          FROM Score
          WHERE 课程编号 IN
              (SELECT 课程编号
              FROM Course
              WHERE 课程名称 = '管理学'));
```

本查询的步骤如下：

（1）在 Course 关系中找出“管理学”的课程编号，结果为{04010101}。

（2）在 Score 关系中找出选修了 04010101 号课程的学生学号集合 X={2006091001，2006091002，2006091003，2006091004，2006091005}。

（3）在 Student 关系中选出学号在集合 X 中的学生的学号和姓名。结果为：

```
学号          姓名
--------------------
2006091001   张楚
2006091002   欧阳佳慧
2006091003   孔灵柱
2006091004   门静涛
2006091005   王广慧
```

2. 带有比较运算符的子查询

当能确切知道内层查询返回单个值（标量值）时，可用比较运算符（>，<，=，>=，<=，!=或< >）连接父查询与子查询。

例如，由于一个学生只能在一个班学习，并且必须属于某一个班，则在【例 3-52】中子查询的结果肯定是一个值，所以可以用=代替 IN。

```
SELECT 学号,姓名,班级编号
FROM Student
WHERE 班级编号 = (SELECT 班级编号
                  FROM Student
                  WHERE 姓名 = '张楚')
```

注意：子查询一定要跟在比较符之后，下列写法是错误的。

```
SELECT 学号,姓名,班级编号
FROM Student
WHERE (SELECT 班级编号 FROM Student WHERE 姓名 = '张楚') = 班级编号
```

【例 3-54】 找出每个学生超出他选修课程平均成绩的课程编号。

```
SELECT s1.学号,s1.课程编号
FROM Score AS s1
WHERE 成绩>(SELECT AVG(成绩)                  --求一个学生所有选修课程的平均成绩,
                  FROM Score AS s2            --至于哪个学生要看参数 s1.学号的值,而
                  WHERE s2.学号 = s1.学号)    --该值是与父查询相关的
```

本例中，子查询的查询条件依赖于父查询，这类子查询称为相关子查询，整个查询语句称为相关嵌套查询语句。求解相关嵌套查询语句的一种可能执行过程为：

（1）首先取父查询表中的第一个元组，将其学号值（2006091001）传送给内层查询，构成子查询。

```
SELECT AVG(成绩)
FROM Score AS s2
WHERE s2.学号 = '2006091001'
```

（2）执行子查询，得到一个值 76，用该值代替子查询，构成父查询：

```
SELECT s1.学号,s1.课程编号
FROM Score AS s1
WHERE 成绩> 76
```

(3) 执行父查询,把得到的元组集合{(2006091001,04010101),(2006091001,04010102)}放入结果表中。

然后再取父查询表的下一个元组重复上述步骤(1)～(3)的处理,直到父查询表的所有元组全部处理完毕。查询结果为:

```
学号                课程编号
------------------
2006091001          04010101
2006091001          04010102
2006091002          04010101
2006091002          04010102
2006091003          04010101
```

3. 带有 ANY(SOME)或 ALL 谓词的子查询

子查询返回单值时可以用比较运算符,但返回多值时要用 ANY(有的系统用 SOME)或 ALL 谓词修饰符。而使用 ANY 或 ALL 谓词时必须同时使用比较运算符。其语义见表 3-7。

表 3-7 ANY、ALL 谓词与比较运算符结合的语义

谓　词	语　义
>ANY	大于子查询结果中的某个值,即大于最小值
>ALL	大于子查询结果中的所有值,即大于最大值
<ANY	小于子查询结果中的某个值,即小于最大值
<ALL	小于子查询结果中的所有值,即小于最小值
>=ANY	大于等于子查询结果中的某个值
>=ALL	大于等于子查询结果中的所有值
<=ANY	小于等于子查询结果中的某个值
<=ALL	小于等于子查询结果中的所有值
=ANY	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(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

【例 3-55】 查询其他班中比“财务管理 062”班任意一个(其中某一个)学生年龄小的学生姓名和年龄。

```
SELECT 姓名,2012 - year(出生日期) AS 年龄          -- 父查询
FROM Student
WHERE 2012 - year(出生日期) < ANY (
      SELECT 2012 - year(出生日期)                 -- 子查询
      FROM Student
      WHERE 班级编号 = (
          SELECT 班级编号                          -- 最内层子查询
```

```
        FROM Class
        WHERE 班级名称 = '财务管理 062'))
    AND 班级编号<>(                                    -- 注意,这是父查询块中的条件
        SELECT 班级编号
        FROM Class
        WHERE 班级名称 = '财务管理 062');
```

查询结果如下：

```
姓名              年龄
------------------
张楚              26
欧阳佳慧          25
孔灵柱            26
门静涛            25
王广慧            26
王大伟            25
谢辉              26
```

RDMS 执行此查询时，首先处理最内层子查询，查出“财务管理 062”班的班级编号值为 200602；然后用该值代替最内层子查询，处理上一层子查询，找出 200602 班中所有学生的年龄，构成一个集合(25,25,27)；最后处理父查询，找出所有不是“财务管理 062”班且年龄小于 27 或 25 学生的姓名与年龄。

本查询也可以用聚集函数实现。首先用子查询找出“财务管理 062”班中的最大年龄(27)，然后通过父查询查出所有非“财务管理 062”班且年龄小于 27 的学生。SQL 语句如下：

```
SELECT 姓名,2012 - year(出生日期) AS 年龄
FROM Student
WHERE 2012 - year(出生日期) < (
            SELECT MAX(2012 - year(出生日期))
            FROM Student
            WHERE 班级编号 = (
                  SELECT 班级编号
                  FROM Class
                  WHERE 班级名称 = '财务管理 062'))
            AND 班级编号<>(
                  SELECT 班级编号
                  FROM Class
                  WHERE 班级名称 = '财务管理 062')
```

【例 3-56】 查询其他班中比“财务管理 062”班所有学生年龄都小的学生姓名和年龄。

方法一：用 ALL 谓词实现。

```
SELECT 姓名,2012 - year(出生日期) AS 年龄
FROM Student
WHERE 2012 - year(出生日期) < ALL(
            SELECT 2012 - year(出生日期)
            FROM Student
            WHERE 班级编号 = (
```

```
        SELECT 班级编号
        FROM Class
        WHERE 班级名称 = '财务管理 062'))
    AND 班级编号<>(
        SELECT 班级编号
        FROM Class
        WHERE 班级名称 = '财务管理 062')
```

方法二：用聚集函数实现。

```
SELECT 姓名,2012 - year(出生日期) AS 年龄
FROM Student
WHERE 2012 - year(出生日期) < (
        SELECT MIN(2012 - year(出生日期))
        FROM Student
        WHERE 班级编号 = (
            SELECT 班级编号
            FROM Class
            WHERE 班级名称 = '财务管理 062'))
        AND 班级编号<>(
            SELECT 班级编号
            FROM Class
            WHERE 班级名称 = '财务管理 062')
```

实际上，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，因为前者通常能够减少比较次数。ANY、ALL 谓词与聚集函数、IN 谓词的等价转换关系见表 3-8。

表 3-8 ANY、ALL 谓词与聚集函数、IN 谓词的等价转换关系

比较符 / 谓词	=	<>或!=	<	<=	>	>=
ANY	IN	无意义	<MAX	<=MAX	>MIN	>=MIN
ALL	无意义	NOT IN	<MIN	<=MIN	>MAX	>=MAX

4. 带有 EXISTS、NOT EXISTS 谓词的子查询

带有 EXISTS 或 NOT EXISTS 谓词的子查询不返回任何数据，只产生逻辑真值(True)或假值(False)，所以子查询的目标列表达式通常都用 * ，因为即使给出列名，也无实际意义。

使用 EXISTS 谓词时，若子查询结果非空，则父查询的 WHERE 子句返回真值；若子查询结果为空，则返回假值。

使用 NOT EXISTS 谓词时，若子查询结果为空，则父查询的 WHERE 子句返回真值；否则返回假值。

【例 3-57】 查询所有选修了 04010101 号课程的学生姓名。

• 思路分析。

(1) 本查询涉及 Student 和 Score 关系。

(2) 在 Student 中依次取每个元组的学号值(Student.学号)，用此值去检查 Score

关系。

（3）若 Score 中存在这样的元组，其学号值（Score.学号）等于 Student.学号的值，并且其课程编号＝'04010101'，则取此 Student.姓名送入结果关系。

• 用嵌套查询。

```
SELECT 姓名
  FROM Student
    WHERE EXISTS
      (SELECT *
      FROM Score /* 相关子查询 */
      WHERE Score.学号 = Student.学号 AND 课程编号 = '04010101');
```

• 用连接查询。

```
SELECT 姓名
FROM Student JOIN Score ON Score.学号 = Student.学号
WHERE 课程编号 = '04010101';
```

【例 3-58】 查询没有选修 04010101 号课程的学生姓名。

```
SELECT 姓名
FROM Student
WHERE NOT EXISTS
    (SELECT *
    FROM Score      /* 相关子查询 */
    WHERE Score.学号 = Student.学号 AND 课程编号 = '04010101');
```

此例用连接查询难以实现。

一些带 EXISTS 或 NOT EXISTS 谓词的子查询不能被其他形式的子查询等价替换。但所有带 IN 谓词、比较运算符、ANY 和 ALL 谓词的子查询都能用带 EXISTS 谓词的子查询等价替换。例如，带有 IN 谓词的【例 3-52】可以用如下带 EXISTS 谓词的子查询替换。

```
SELECT 学号,姓名,班级编号
FROM Student AS S1
WHERE EXISTS (
            SELECT *
            FROM Student AS S2
            WHERE S2.班级编号 = S1.班级编号 AND S2.姓名 = '张楚');
```

【例 3-59】 查询选修了全部课程的学生姓名。

可将此题目的意思转换为：查找这样的学生，没有一门课是他不选修的。其 SQL 语句为：

```
SELECT 姓名
FROM Student
WHERE NOT EXISTS
      (SELECT *
      FROM Course
      WHERE NOT EXISTS
            (SELECT *
```

```
FROM Score
WHERE 学号 = Student.学号 AND 课程编号 = Course.课程编号));
```

3.4.4 集合查询

SELECT 语句的查询结果是元组的集合，所以多个 SELECT 语句的结果可进行集合操作。集合操作主要包括并操作(UNION)、交操作(INTERSECT)和差操作(EXCEPT)。需要注意的是，参加集合操作的各查询结果表的列数必须相同；并且对应列的数据类型也必须相同。

【例 3-60】 查询选修了课程 04010101 或者选修了课程 04010102 的学生。

```
SELECT 学号
FROM Score
WHERE 课程编号 = '04010101'
UNION
SELECT 学号
FROM Score
WHERE 课程编号 = '04010102';
```

本查询实际上是求选修课程 04010101 的学生集合与选修课程 04010102 的学生集合的并集。使用 UNION 将多个查询结果合并起来时，系统会自动去掉重复元组。如果要保留重复元组，则需用 UNION ALL 操作符。

【例 3-61】 查询既选修了课程 04010101，又选修了课程 04010102 的学生。

本例实际上是查询选修了课程 04010101 的学生集合与选修了课程 04010102 的学生集合的交集。

```
SELECT 学号
FROM Score
WHERE 课程编号 = '04010101'
INTERSECT
SELECT 学号
FROM Score
WHERE 课程编号 = '04010102';
```

【例 3-62】 查询未被学生选修的课程编号。

本例实际上是查询课程编号的集合与已被选修的课程编号的差集。

```
SELECT 课程编号
FROM Course
EXCEPT
SELECT DISTINCT 课程编号
FROM Score
```

3.5 视图

视图是从一个或几个基本表(或视图)导出的表。数据库中只存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中，所以，视图是虚表。若基本表中的

数据发生变化，则从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中自己感兴趣的数据及其变化。

视图一经定义，就可以和基本表一样被查询、被删除，也可以在一个视图上再定义一个新视图，但对视图的更新操作有一定的限制。

3.5.1 定义视图

1. 创建视图

创建视图的语句格式如下。

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

注意：组成视图的属性列名可以全部省略或全部指定，如果省略，则由子查询中SELECT子句中目标列中的诸字段组成；当子查询的某个目标列是*、集函数、列表达式或需要在视图中为某个列重新命名时，则要全部指定视图的列名。WITH CHECK OPTION选项表示通过视图进行增、删、改操作时，不得破坏视图定义中的谓词条件（即子查询中的条件表达式）。另外，子查询中不允许含有ORDER BY子句和DISTINCT短语。

1）行列子集视图

从单个基本表导出，只是去掉了基本表的某些行和某些列，而保留了码的视图称为行列子集视图。

【例 3-63】 建立工商管理061班学生的视图。

```
CREATE VIEW GS_Student
AS
  SELECT 学号,姓名,性别,入学成绩
  FROM Student
  WHERE 班级编号 = (SELECT 班级编号
                    FROM Class
                    WHERE 班级名称 = '工商管理 061')
```

【例 3-64】 建立学生党员的视图，并要求通过该视图进行的更新操作只涉及学生党员。

```
CREATE VIEW DY_Student
AS
  SELECT 学号,姓名,性别
  FROM Student
  WHERE 党员否 = 1
WITH CHECK OPTION;
```

2）基于多个基表的视图

视图不仅可以建立在单个基本表上，也可以建立在多个基本表上。也就是说，视图的属性列可来自多个基本表。

【例 3-65】 建立200601班选修了04010101号课程的学生视图。

```
CREATE VIEW GS_S1(学号,姓名,成绩)
```

```
AS
  SELECT Student.学号,姓名,成绩
  FROM Student JOIN Score ON Student.学号 = Score.学号
  WHERE 课程编号 = '04010101';
```

3）基于视图的视图

视图不仅可以建立在一个或多个基本表上，也可以建立在一个或多个已建立好的视图上，或建立在基本表与视图上。

【例 3-66】 建立 200601 班选修了 04010101 号课程且成绩在 60 分以上的学生的视图。

```
CREATE VIEW GS_S2
AS
  SELECT 学号,姓名,成绩
  FROM GS_S1
  WHERE 成绩>=60;
```

4）带表达式的视图

由于视图中的数据并不实际存储，所以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，所以也称为虚拟列。带虚拟列的视图也称为带表达式的视图。

【例 3-67】 建立一个反映学生年龄的视图。

```
CREATE VIEW AGE_Student(学号,姓名,年龄)
AS
  SELECT 学号,姓名,year(getdate())-year(出生日期)
  FROM Student
```

5）分组视图

还可以用带有聚集函数和 GROUP BY 子句的查询来定义视图，这种视图称为分组视图。

【例 3-68】 将学生的学号及他的平均成绩定义为一个视图。这类视图必须明确定义组成视图的各个属性列名。

```
CREATE VIEW S_AVG(学号,平均成绩)
AS
  SELECT 学号,AVG(成绩)
  FROM Score
  GROUP BY 学号;
```

2. 删除视图

删除视图的语句格式为：

```
DROP VIEW <视图名>;
```

该语句的功能是从数据字典中删除指定的视图定义。而由该视图导出的其他视图的定义却仍存在数据字典中，但这些视图已失效。为了防止用户使用时出错，要用 DROP VIEW 语句把那些失效的视图一一删除。同样，删除基表后，由该基表导出的所有视图定义都必须显式地使用 DROP VIEW 语句删除。

【例 3-69】 删除视图 GS_Student。

```
DROP VIEW GS_Student;
```

3.5.2 查询视图

从用户角度而言，查询视图与查询基本表相同。DBMS 实现视图查询的方法有两种。

(1) 实体化视图(View Materialization)。

- 有效性检查：检查所查询的视图是否存在。
- 执行视图定义，将视图临时实体化，生成临时表。
- 查询视图转换为查询临时表。
- 查询完毕后，删除被实体化的视图(临时表)。

(2) 视图消解法(View Resolution)。

- 进行有效性检查，检查查询的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义。
- 把视图定义中的子查询与用户的查询结合起来，转换成等价的对基本表的查询。
- 执行修正后的查询。

【例 3-70】 在工商管理 061 班学生的视图中找出入学成绩大于 500 分的学生姓名。

```
SELECT 姓名
FROM GS_Student
WHERE 入学成绩> 500;
```

实体化视图法是将 GS_Student 实体化成临时表后再查询。而视图消解法是将该查询转换成如下的查询语句对基表查询。

```
SELECT 姓名
FROM Student
WHERE 班级编号 = (SELECT 班级编号
                  FROM Class
                  WHERE 班级名称 = '工商管理 061')
        AND 入学成绩> 500
```

【例 3-71】 查询工商管理 061 班选修了 04010101 号课程的学生姓名。

```
SELECT 姓名
FROM GS_Student JOIN Score ON GS_Student.学号 =  Score.学号
WHERE Score.课程编号 = '04010101';
```

注意：有些情况下，视图消解法不能生成正确的查询。采用视图消解法的 DBMS 会限制这类查询。

【例 3-72】 在 S_AVG 视图中查询平均成绩在 85 分以上的学生学号和平均成绩。

```
SELECT *
FROM S_AVG
WHERE 平均成绩> = 85;
```

将本例中的查询语句与 S_AVG 视图中定义的查询结合，形成如下的查询转换语句：

```
SELECT 学号,AVG(成绩)
FROM Score
WHERE AVG(成绩)>=85
GROUP BY 学号
```

由于 WHERE 子句中不能用聚集函数作为条件表达式,所以该查询不能被执行。正确的查询转换语句为:

```
SELECT 学号,AVG(成绩)
FROM Score
GROUP BY 学号 HAVING AVG(成绩)>=85
```

目前多数关系数据库系统对行列子集视图的查询均能进行正确的转换,但对非行列子集视图的查询(如【例 3-72】)就不一定能正确转换了,因此这类查询应该直接对基本表进行转换。

3.5.3 修改视图

修改(更新)视图是指通过视图来插入、删除和修改数据。由于视图是不存储数据的虚表,因此对视图的更新,最终要转换为对基本表的更新。

为防止用户更新视图时有意无意地对不属于视图范围内的基本表数据进行操作,可在定义视图时指定 WITH CHECK OPTION 子句,这样,DBMS 在更新视图时会检查视图定义中的条件,若不满足条件,则拒绝执行更新操作。

【例 3-73】 将工商管理 061 班学生视图 GS_Student 中学号为 2006091001 的学生入学成绩改为 550。

```
UPDATE GS_Student
SET 入学成绩=550
WHERE 学号='2006091001';
```

【例 3-74】 向工商管理 061 班学生视图 GS_Student 中插入一个新的学生记录:(2006091021,赵新,男,500)。

```
INSERT
INTO GS_Student
VALUES('2006091021','赵新','男',500);
```

注意:导出视图的基表 Student 中,除指定了具体值的学号、姓名、性别和入学成绩属性列之外,其他未明确给定值的属性应该允许插入 NULL 值,否则,插入操作将不会执行。显然,由于新插入学生记录的班级编号属性列为 NULL,即该学生不属于任何班级,因此,新插入的学生记录不会出现在虚表 GS_Student 中,而只是在基表 Student 中。

如果在定义视图 GS_Student 的 CREATE VIEW 语句中指定 WITH CHECK OPTION 选项,则本例的插入操作将不会被执行,因为新插入的元组不符合视图定义时的条件,即未明确指定班级编号属性。

【例 3-75】 删除视图 GS_Student 中学号为 2006091002 的记录。

```
DELETE
FROM GS_Student
WHERE 学号='2006091002';
```

在关系数据库中，并不是所有的视图都是可更新的，因为有些视图的更新不能唯一地有意义地转换成对基本表的更新。

例如，在【例 3-68】中定义的视图 S_AVG 是不可更新的。对于如下更新语句：

```
UPDATE S_AVG
SET 平均成绩 = 90
WHERE 学号 = '2006091001';
```

系统无法将其转换成对基本表 Score 的更新，因为系统无法修改 Score 表的各门课成绩，以使平均成绩为 90。所以，S_AVG 视图是不可更新的。

总之，可以得出如下结论：

（1）如果视图属性中包含基本表的主码（也可能是其他一些候选码），那么由单一基本表导出的视图，即行列子集视图是可以更新的，这是因为每个视图（虚）元组都可以映射到一个基本表的元组中。

（2）在多个表上使用连接操作定义的视图一般都是不可更新的。

（3）使用分组和聚集函数定义的视图是不可更新的。

一般来说，实际 DBMS 都允许对行列子集视图进行更新；而且各个系统对视图的更新还有更进一步的规定。例如，SQL Server 对视图更新的规定如下：

（1）若视图的字段来自聚集函数，则此视图不允许更新。

（2）若视图定义中含有 GROUP BY 子句，则此视图不允许更新。

（3）若视图定义中含有 DISTINCT 短语，则此视图不允许更新。

（4）在一个不允许更新的视图上定义的视图也不允许更新。

（5）由于向视图插入数据的实质是向其所引用的基本表中插入数据，所以必须确认那些未包括在视图中的列但属于基表的列允许 NULL 值或有默认值。对多表视图，若要执行 INSERT 语句，则一个插入语句只能对属于同一个表的列执行操作，即一个插入操作需要用多个 INSERT 语句来实现。

（6）通过视图对数据进行修改与删除需要注意两个问题：执行 UPDATE 或 DELETE 时所删除与修改的数据，必须包含在视图结果集中；视图引用多个表时，无法用 DELETE 命令删除数据，若使用 UPDATE，则应与 INSERT 操作一样，被修改的列必须属于同一个表。

3.5.4 视图的作用

视图是定义在基本表上的，对视图的一切操作最终也要转换为对基本表的操作，并且对视图的更新操作还会受到种种限制。既然如此，为什么还要使用视图呢？这是因为合理使用视图能够带来许多好处。

1. 视图能够简化用户的操作

视图机制使用户可以将注意力集中在所关心的数据上。如果这些数据不是直接来自基本表，则可以通过定义视图，使数据库看起来结构简单、清晰，并且可以简化用户的数据查询操作。例如，基于多张表连接形成的视图，就将表与表之间的连接操作对用户隐藏起来了。换句话说，用户所做的只是对一个虚表的简单查询，而这个虚表是怎样得来的，用户无须了解。

2. 视图使用户能以多种角度看待同一数据

视图机制能使不同用户以不同方式看待同一数据，但许多不同种类的用户共享同一个

数据库时，这种灵活性是非常重要的。

3. 视图对重构数据库提供了一定程度的逻辑独立性

在关系数据库中，数据库的重构往往是不可避免的。重构数据库最常见的是将一个基本表“垂直”地分成多个基本表。例如，将学生基本表：

```
Student(Sno,Sname,Ssex,Sage,Sdept)
```

“垂直”地分成两个基本表。

```
SX(Sno,Sname,Sage)
SY(Sno,Ssex,Sdept)
```

通过建立一个视图 Student：

```
CREATE VIEW Student(Sno,Sname,Ssex,Sage,Sdept)
    AS
    SELECT SX.Sno,SX.Sname,SY.Ssex,SX.Sage,SY.Sdept
    FROM SX,SY
    WHERE SX.Sno = SY.Sno;
```

使用户的外模式保持不变，用户的应用程序通过视图仍然可以查询数据。

当然，视图只能在一定程度上提供数据的逻辑独立性，例如，由于对视图的更新是有条件的，所以应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。

4. 视图能够对机密数据提供安全保护

视图可以作为一种安全机制。通过视图，用户只能查看和修改他们所能看到的数据。在涉及数据库应用系统时，对不同用户定义不同视图，使机密数据不出现在不应看到这些数据的用户视图上，这样视图机制就自动提供了对机密数据的安全保护功能。

3.6 SQL 的数据控制

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。

SQL 提供了数据控制功能，能够在一定程度上保证数据库中数据的安全性、完整性，并提供了一定的并发控制及恢复能力。

安全性指保护数据库，防止不合法的使用造成的数据泄露和破坏。保证数据安全性的主要措施是存取控制机制，控制用户只能存取他有权存取的数据，同时令所有未被授权的用户无法接近数据。

目前，大型数据库管理系统几乎都支持自主存取控制（即用户对不同的数据库对象有不同的存取权限，不同的用户对同一对象也有不同的权限，而且用户还可以将其拥有的存取权限转授给其他用户），在 SQL 标准中，主要是通过 GRANT 和 REVOKE 语句来实现的。

用户权限由两个要素组成：数据库对象和操作类型。定义用户的存取权限就是定义这个用户可以在哪些数据库对象上进行哪些类型的操作。这一操作称为授权。

关系数据库系统中存取控制的对象不仅有数据本身（基本表中的数据、属性列上的数据），还有数据库模式（数据库、基本表、视图和索引等）。表 3-9 列出了关系数据库系统中的存取权限。

表 3-9 关系数据库系统中的存取权限

对 象	对象类型	操作类型
属性列	数据	SELECT, INSERT, UPDATE, REFERENCE, ALL PRIVILEGES
基本表与视图	数据	SELECT, INSERT, UPDATE, DELETE, REFERENCE, ALL PRIVILEGES
索引	模式	CREATE INDEX
视图	模式	CREATE VIEW
基本表	数据库	CREATE TABLE, ALTER TABLE

3.6.1 授权

授权语句的一般格式为：

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

其功能为：将对指定操作对象的指定操作权限授予指定的用户。发出该授权语句的可以是 DBA，也可以是该数据库对象的创建者（即属主 OWNER），也可以是已经拥有该权限的用户。接受权限的用户可以是一个或多个具体用户，也可以是 PUBLIC（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，不能传播该权限。

【例 3-76】 把查询 Student 表的权限授予用户 U1。

```
GRANT SELECT
ON TABLE Student
TO U1;
```

【例 3-77】 把对 Student 表的全部权限授予用户 U2 和 U3。

```
GRANT ALL PRIVILEGES
ON TABLE Student
TO U2, U3;
```

【例 3-78】 把对 Score 表的查询权限授予所有用户。

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

【例 3-79】 把查询 Student 表和修改学生学号的权限授予用户 U4。

```
GRANT UPDATE(学号),SELECT
    ON TABLE Student
TO U4;
```

注意：对属性列的授权必须明确指出相应的属性列名称。

【例 3-80】 把对 Score 表的 INSERT 权限授予用户 U5，并允许他再将此权限授予其

他用户。

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

执行此 SQL 语句后，用户 U5 不仅拥有了对表 Score 的 INSERT 权限，还可以传播此权限。例如，用户 U5 可以将此权限授予用户 U6。

```
GRANT INSERT
ON TABLE Score
TO U6
WITH GRANT OPTION;
```

同样，用户 U6 还可以将此权限授予用户 U7。

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

但用户 U7 不能再传播此权限。因为用户 U6 未给用户 U7 传播的权限。也不允许循环授权，即被授权者不能把权限再授回给授权者或其祖先。例如，用户 U6 不能再把权限授回给用户 U5。

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>[,<对象类型> <对象名>]…
FROM <用户>[,<用户>]…[CASCADE|RESTRICT];
```

该语句的功能为从指定用户那里收回对指定对象的指定权限。

【例 3-81】 把用户 U4 修改学生学号的权限收回。

```
REVOKE UPDATE(学号)
ON TABLE Student
FROM U4;
```

【例 3-82】 收回所有用户对 Score 表的查询权限。

```
REVOKE SELECT
ON TABLE Score
FROM PUBLIC;
```

【例 3-83】 把用户 U5 对 Score 表的 INSERT 权限收回。

```
REVOKE INSERT
ON TABLE Score
FROM U5 CASCADE;
```

将用户 U5 的 INSERT 权限收回时必须级联(CASCADE)收回，不然系统将拒绝(RESTRICT)执行该命令。因为用户 U5 将对 Score 的 INSERT 权限授予用户 U6，而用户 U6 又将其授予用户 U7。

注意，这里的默认值为 RESTRICT，有的 DBMS 默认值为 CASCADE，会自动执行级联操作，而不必明确加 CASCADE 选项。如果用户 U6 或用户 U7 还从其他用户处获得了对表 Score 的 INSERT 权限，则他们仍具有此权限，系统只是收回直接或间接从用户 U5 处获得的权限。

SQL 提供了非常灵活的授权机制。DBA 拥有对数据库中所有对象的所有权限，并可以根据实际情况将不同的权限授予不同的用户。

用户对自己建立的基本表和视图拥有全部的操作权限，并且可以用 GRANT 语句把其中某些权限授予其他用户。被授权的用户如果有“继续授权”的许可，还可以把获得的权限再授予其他用户。所有授予出去的权利在必要时又都可以用 REVOKE 语句收回。

习题

1. 简述 SQL 的特点。

2. 什么是基本表，什么是视图，两者的区别和联系是什么？

3. 简述视图的优点。所有视图是否都可以更新？为什么？

4. 针对第 2 章中习题 5 的题意要求，试用 SQL 语句表示其查询。

5. 设有下列 4 个关系模式：

S(SNO,SNAME,CITY)

P(PNO,PNAME,COLOR,WEIGHT)

J(JNO,JNAME,CITY)

SPJ(SNO,PNO,JNO,QTY)

其中，供应商 S 由供应商号(SNO)、供应商姓名(SNAME)、供应商所在城市(CITY)组成，记录各个供应商的情况。零件表 P 由零件号(PNO)、零件名称(PNAME)、零件颜色(COLOR)、零件重量(WEIGHT)组成，记录各种零件的情况。工程项目表 J 由项目号(JNO)、项目名(JNAME)、项目所在城市(CITY)组成，记录各个项目的情况。供应情况表 SPJ 由供应商号(SNO)、零件号(PNO)、项目号(JNO)、供应数量(QTY)组成，记录各供应商供应各种零件给各工程项目的数量。试使用 SQL 命令完成下列查询操作。

(1) 求供应工程 J1 零件的供应商号(SNO)。

(2) 求供应工程 J1 零件 P1 的供应商号(SNO)。

(3) 求供应工程 J1 红色零件的供应商号(SNO)。

(4) 求没有使用天津供应商生产的红色零件的项目号(JNO)。

(5) 求至少用了 S1 供应商所供应的全部零件的项目号(JNO)。

6. 设要建立学生选课数据库，库中包括学生、课程和选课 3 个基本表，其表结构为：

学生(学号，姓名，性别，年龄，所在系)

课程(课程号，课程名，先行课)

选课(学号，课程号，成绩)

试用 Transact-SQL 完成下列操作。

(1) 建立学生选课数据库。

(2) 建立学生、课程和选课表。

(3) 创建成绩单视图,包括学号、姓名、课程号、课程名及成绩5列。

(4) 建立在对选课表输入或更改数据时,必须服从参照完整性约束的INSERT和UPDATE触发器。

(5) 建立在删除学生记录时,同时也要删除相应选课记录的DELETE触发器。

(6) 建立统计某学生所修课程平均成绩的存储过程。

(7) 查询各系及学生数,最后求出共有多少系和多少学生。

(8) 将学生表与选课表进行内连接、左外连接和右外连接。

(9) 查询学生学号、姓名及学习情况。学习情况用好、较好、一般或较差表示。当其所修课程的平均成绩大于85分时,学习情况为好;当平均成绩为70～85分时,学习情况为较好;当平均成绩为60～69分时,学习情况为一般;当平均成绩在60分以下时,学习情况为较差。