第3章 关系数据库标准语言SQL

SQL是关系数据库的标准语言，利用它不但可以查询数据库中数据，还可以定义数据库，以及编辑数据库中的数据。SQL功能强大、简洁易学，而且被绝大多数的关系数据库管理系统所支持，这为它的广泛使用打下了坚实基础。本章主要讨论SQL的基本语法，以及使用技巧。

内容提要：

☑ SQL的基本知识
☑ 使用SQL创建、修改和删除数据库对象
☑ 使用SQL查询数据库中的数据
☑ 使用SQL更新数据库中的数据

3.1 SQL概述

3.1.1 SQL简介

SQL标准一经推出就受到各个数据库厂商的欢迎，它们纷纷推出实现SQL标准的各种数据库产品。目前SQL已经成为主流数据库的标准查询语言，用户只要掌握了SQL语言就相当于拥有了打开数据库的金钥匙。

1. SQL的起源与发展

SQL最初是由IBM的San Jose实验室开发出来的，并在1974年的ACM会议上发表。最初的名字并不叫SQL，而是叫SEQUEL(Structured English Query Language，结构化英语查询语言)，随后简写为SQL(Structured Query Language，结构化查询语言)。由于SQL具有功能强大、使用灵活、简单易学等优点，所以受到广大用户和计算机工业界的认可。

1986年10月美国国家标准局(American National Standard Institute，ANSI)的数据库委员会批准了SQL作为关系数据库语言的美国标准。同年公布了SQL标准文本，称为SQL 86。1987年6月，国际标准化组织(International Organization for Standardization，简称ISO)采纳其为国际标准。

在1989年10月，ANSI又颁布了增强完整性特征的SQL 89标准。随后在1992年ISO对标准又进行了修订，增加了许多新功能。该版本被称为SQL 92，但人们更习惯称之

为 SQL2。

在 1999 年，ISO 又颁布了 SQL 99，人们习惯称之为 SQL3，主要特性是支持对象关系模型，是至今使用最多的比较完整的 SQL 标准。在 SQL 92 发布之后，SQL 标准又经历了三次修订，分别为 SQL 2003、SQL 2006 和 SQL 2008，这些版本主要扩充了对对象模型和 XML 模型的支持。

2. SQL 组成

核心 SQL 主要由数据定义语言、数据操纵语言、数据控制语言和嵌入式 SQL 语言所组成。

1) 数据定义语言

数据定义语言(DDL)完成对模式、基本表、视图、索引、域等的创建、修改和删除操作。

2) 数据操纵语言

数据操纵语言(DML)用于查询和更新数据库中的数据。其中，查询完成对数据库中的数据进行检索、统计、分组、排序等操作；更新完成对数据库中的数据的插入、修改和删除操作。

3) 数据控制语言

数据控制语言(DCL)用于对基本表和视图的授权、完整性规则的描述、事务控制等。

4) 嵌入式 SQL 语言

嵌入式 SQL 规定了 SQL 语句在高级程序设计语言中使用的规范与标准。

3. SQL 的特点

SQL 之所以成为数据库的主流语言，是因为它是一个综合的、简单易学、功能强大的语言。SQL 除了具有一般关系数据库语言的特点以外，还具有以下特点。

1) 综合统一

SQL 集数据定义语言、数据操纵语言和数据控制语言于一身，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。

另外，在关系模型中，实体和联系都是用关系表示的。这种数据结构的单一性使得对数据的查询和更新操作都使用一种操作符，从而克服了非关系系统由于信息表示的多样性带来的操作复杂性。

2) 高度非过程化

SQL 是一种第四代语言(4GL)，用户只需要提出“做什么”，而无须指明“怎么做”，因此不需要了解存取路径。存取路径的选择以及 SQL 的操作过程由系统自动完成，从而减轻了用户负担，而且有利于提高数据独立性。

3) 面向集合的操作方式

SQL 采用集合操作模式，其操作对象和查找结果都是元组的集合，而且插入、删除和修改的对象与结果也是元组的集合。即 SQL 语句将集合作为输入，返回结果也是集合，具有“一次一集合”的特征。

4) 语言简洁、易学易用

SQL 功能非常强大，但语言非常简洁，完成所有核心功能只需要 9 个动词，如表 3-1 所

示。而且 SQL 语句与英语非常接近，对于熟悉英语的学习者非常容易学习。

表 3-1　SQL 的动词

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

5）一种语法结构两种使用方式

SQL 既可以作为独立语言使用，又可以作为嵌入式语言使用。无论哪种使用方式，SQL 的语法结构基本上是一致的。

作为独立的语言，SQL 可以独立用于联机交互中。用户在终端上输入 SQL 命令对数据库进行操作。

作为嵌入式语言，SQL 语句可以嵌入到高级程序设计语言编写的程序中，既能够对数据库进行操作，又能够与程序进行交互。

6）支持三级模式结构

SQL 支持关系数据库的三级模式结构，如图 3-1 所示。其中，外模式对应于视图和部分基本表，模式对应于基本表，内模式对应于存储文件。

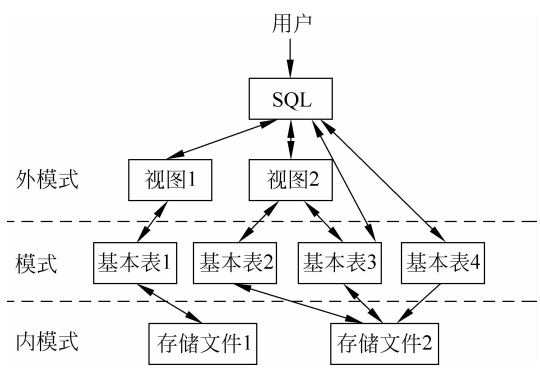


图 3-1　SQL 支持的三级模式结构

由图 3-1 可以看出，全体基本表组成了数据库的模式；由视图和部分基本表组成了数据库的外模式。用户使用 SQL 对基本表和视图进行查询或其他操作；存储文件的逻辑结构组成了关系数据库的内模式。一个或多个基本表对应一个存储文件，一个基本表可以有多个索引，索引也存储在文件中。

4. 示例数据库

为了帮助读者更好地理解 SQL，在本章中使用了大量示例。这些示例均基于 library 数据库。在 library 数据库中，包含三张基本表，它们分别为：

读者表：reader(rno，rname，rgender，rage，rspecialty)。其中，rno 表示读者编号，是读者表的主码；rname 表示读者姓名；rgender 表示性别；rage 表示年龄；rspecialty 表示所属

专业，其中，CP 表示计算机专业，FR 表示外语专业，IF 表示信息专业，MA 表示数学专业。读者表的数据如表 3-2 所示。

表 3-2 reader 表

rno	rname	rgender	rage	rspecialty
14001	王建立	男	18	CP
14002	张玲	女	19	FR
14003	王军	男	20	IF
14004	李建国	男	19	CP
14005	赵文娟	女	18	MA

图书表：book(bisbn，bname，bauthor，bpublisher，bprice，bcategory)。其中，bisbn 表示图书 ISBN 号，是图书表的主码；bname 表示图书名称；bauthor 表示作者；bpublisher 表示出版社；bprice 表示价格；bcategory 表示图书类别。图书表的数据如表 3-3 所示。

表 3-3 book 表

bisbn	bname	bauthor	bpublisher	bprice	bcategory
04-012955-7	积分变换	张元林	高等教育出版社	8.00	理
04-023909-6	线性代数	贾兰香	高等教育出版社	16.00	理
302-00860-4	C 程序设计	谭浩强	清华大学出版社	20.00	工
302-02368-5	数据结构_C 语言版	严蔚敏	清华大学出版社	22.00	工
302-03314-5	数据结构题集	严蔚敏	清华大学出版社	16.00	工
5613-3644-6	读者文摘精粹版	周宏	陕西师范大学出版社	38.00	文
81080-159-1	大学英语	李荫华	上海外语教育出版社	35.00	文

借阅表：borrow(rno，bisbn，startdate，enddate，fine)。rno 表示读者编号；bisbn 表示图书 ISBN 号；startdate 表示借书日期；enddate 表示还书日期；fine 表示罚款金额。rno、bisbn 与 startdate 一起作为借阅表的主码。借阅表的数据如表 3-4 所示。

表 3-4 borrow 表

rno	bisbn	startdate	enddate	fine
14001	302-00860-4	2013/8/6		0.00
14001	302-02368-5	2013/3/10	2013/6/15	2.00
14001	302-03314-5	2013/3/10	2013/7/12	3.00
14003	04-012955-7	2013/10/7	2013/12/20	3.50
14004	302-00860-4	2013/1/9	2013/3/9	0.00
14004	302-03314-5	2013/5/9		0.00
14005	302-00860-4	2013/6/18	2013/7/10	0.00

3.1.2 SQL 基本知识

1. 基本概念

在 SQL 中，关系被称为基本表，元组被称为行或者记录，属性被称为列或者字段。一个

基本表由行的集合构成，一行是列的序列，每列对应一个数据项。

表有三种基本类型：

- 基本表——是实际存储在数据库中的表，在其中存储实际的数据，是“实表”。
- 视图——是由一个或多个基本表或其他视图导出的表，在数据库中只存储视图的定义，不存储实际数据，实际数据仍存储在导出视图的基本表中，它是“虚表”。
- 查询表——是执行查询后，由查询结果形成的表，是“临时表”。

2. 数据类型

基本表的数据都是有类型的，用于决定数据的存储方式、数据的长度、能参与的运算，以及是字符、数值，还是日期时间数据。这些在 SQL 中用数据类型来表示。

SQL 提供的主要数据类型一般包括以下几种。

1) 数值型

数值型用于表示整数与小数，主要包括：

- INT(INTEGER)——长整数，其长度取决于具体实现，有可能是 16 位、32 位、64 位或者 128 位。
- SMALLINT——短整数，通常只有标准整型的一半大小。
- REAL——单精度浮点数。
- DOUBLE PRECISION——双精度浮点数。
- FLOAT(n)——能够设置精度的浮点数，其精度至少为 n 位数字。
- NUMERIC(p,d)(或 DECIMAL(p,d))——定点数，包含整数部分和小数部分。由 p 位数字(不包括正负号和小数点)组成，小数点后有 d 位数字。

例如，可以将读者基本表的年龄属性列定义为 INT 型。

2) 字符型

字符型用于表示字符或者字符串，主要包括：

- CHAR(n)——长度为 n 的定长字符串。即使存储少于 n 个字符，系统仍然会分配 n 个字符的空间，而空余空间用空格填充。
- VARCHAR(n)——最大长度为 n 的变长字符串。存储多少个字符，系统就分配多少个字符的空间，直至最大长度 n。因此，它比 CHAR(n)节省空间。

例如，可以将读者基本表的读者姓名属性列定义为 CHAR(30)。

3) 日期型

日期型用于表示日期和时间，主要包括：

- DATE——日期。包含年、月、日，一般格式为“YYYY-MM-DD”。
- TIME——时间。包括时、分、秒，一般格式为“HH：MM：SS”。

例如，可以将借阅基本表的借阅日期属性列定义为 DATE 型。

4) 位串型

位串型用于表示二进制数据，主要包括：

- BINARY(n)——长度为 n 的二进制位串。
- VARBINARY (n)——最大长度为 n 的变长二进制位串。

除了以上这些绝大部分数据库管理系统都支持的数据类型以外，每一个数据库管理系

统都有自己特有的数据类型，这些数据类型不能用于其他数据库管理系统中。比如，有的数据库管理系统会提供货币型、文本型、图像型、二进制大对象等数据类型。

温馨提示：

SQL 一般不区分大小写，但为了表达清晰，本书将 SQL 关键字以大写形式表示，将表名、属性列名等以小写形式表示。

3. 常量

常量也称为标量值，用于表示一个特定数据值。常量的格式取决于它所表示的值的数据类型。

1）数值型常量

例如，2、－3.14、0.5E－2（科学记数法，表示 0.5×10^{-2}）。

2）字符型常量

字符型常量由括在单引号内的数字、字符和特殊字符组成。例如，'money'、'1234'、'计算机'。

温馨提示：

也可以用双引号代替单引号，但一般建议使用单引号。

3）日期型常量

日期型常量表示特定的日期或者时间，并用单引号将日期或者时间括起来。例如，'2013-11-20'、'11/20/2013'、'2013/11/20'、'15：28：25'。

温馨提示：

不同的数据库管理系统所提供的日期或者时间的格式不尽相同，在使用时要仔细阅读相应的手册。

4. 运算符与表达式

在 SQL 中，运算符主要用于确定条件和进行数学运算。主要包括算术运算符、比较运算符、逻辑运算符和一些特殊运算符。

表达式由常量、属性列、运算符和函数组成。在 SQL 语句中，表达式通常用于表示条件。例如，在查询读者编号为 14001 的读者信息时，使用 SQL 语句

```
SELECT *
        FROM reader WHERE rno = '14001';
```

其中，“rno＝'14001'”即为表示条件的表达式，含义为 rno 属性列的取值等于 14001。

1）比较运算符

比较运算符用于比较大小的操作。比较运算符包括：＝、＜、＜＝、＞、＞＝、＜＞（或者！＝），表示等于、小于、小于等于、大于、大于等于和不等于。例如，rno＝'14001'。

2）逻辑运算符

在 SQL 中，逻辑运算符主要用于连接多个子表达式。逻辑运算符包括：AND、OR 和 NOT，表示与、或和非。

如果 AND 连接的多个子表达式均成立，则表达式成立。例如，查询年龄（rage 属性列）

小于 19 岁的男(rgender 属性列)读者的信息。其表达式为：

```
rage < 19 AND rgender = '男'
```

如果 OR 连接的多个子表达式中至少有一个成立，则表达式成立。例如，查询年龄为 18 岁或者 19 岁的读者。其表达式为：

```
rage = 18 OR rage = 19
```

3) 谓词

• BETWEEN… AND …和 NOT BETWEEN … AND …

谓词 BETWEEN … AND …用于判断属性列值是否在指定范围内。例如，在读者基本表中，查找年龄(属性列为 rage)在 18～20 岁之间的读者。其表达式为：

```
rage BETWEEN 18 AND 20
```

也可以表示为：

```
rage >= 18 AND rage <= 20
```

谓词 NOT BETWEEN … AND …用于判断属性列值是否不在指定范围内。例如，在读者基本表中，查找年龄(属性列为 rage)不在 18～20 岁之间的读者。其表达式为：

```
rage NOT BETWEEN 18 AND 20
```

也可以表示为：

```
rage <= 18 OR rage >= 20
```

温馨提示：

在 BETWEEN … AND …所表示的范围中，范围的两个边界值也包括在内。

• IN 和 NOT IN

谓词 IN 用于判断属性列值是否在指定集合内。例如，在读者基本表中，查找年龄为 15 岁、17 岁或 18 岁的读者。其表达式为：

```
rage IN(15,17,18)
```

也可以表示为：

```
rage = 15 OR rage = 17 OR rage = 18
```

谓词 NOT IN 用于判断属性列值是否不在指定集合内。例如，在读者基本表中，查找年龄不为 15 岁、17 岁或 18 岁的读者。其表达式为：

```
rage NOT IN(15,17,18)
```

也可以表示为：

```
rage <> 15 AND rage <> 17 AND rage <> 18
```

• LIKE 和 NOT LIKE

谓词 LIKE 用于进行字符串匹配。其语法格式为：

[NOT] LIKE 匹配串[ESCAPE 转义字符]

通常用于判断基本表的属性列值是否与匹配串相匹配。匹配串可以是由普通字符组成的字符串,也可以是包含通配符的字符串。一般情况下,通配符有两个,即%和_。其含义为:

%(百分号)——代表任意长度(长度可以为0)的字符串。

_(下划线)——代表任意单个字符。

例如,查找姓李的读者。其表达式为:

```
rname LIKE '李%'
```

含义为:判断属性列 rname 的值是否以“李”开头,并且后跟若干个字符,即姓李的读者。

例如,查找名字倒数第二个字符为“建”的读者。其表达式为:

```
rname LIKE '%建__'
```

含义为:判断属性列 rname 的值倒数第二个字符是否为“建”。

温馨提示:

一个汉字占两个字符的位置,所以匹配串“%建”后跟两个“_”。

如果匹配串中包含%和_,但它们不作为通配符,而是作为普通字符,这时就要使用 ESCAPE 对通配符进行转义了。

例如,查找“small_join”。其表达式为:

```
rname LIKE 'smal\_join' ESCAPE '\'
```

则匹配串中的“_”就不作为通配符了,而是作为普通字符“_”。

- IS NULL 和 IS NOT NULL

谓词 IS NULL 用于判断是否为空。

例如,在读者基本表中,查找专业(属性列为 rspecialty)为空的读者。其表达式为:

```
rspecialty IS NULL
```

谓词 IS NOT NULL 用于判断是否不为空。

例如,在读者基本表中,查找专业(属性列为 rspecialty)不为空的读者。其表达式为:

```
rspecialty IS NOT NULL
```

4) 算术运算符

算术运算符用于执行数学运算。算术运算符包括+、-、* 和/,表示加、减、乘和除。例如,将读者年龄增加一岁。其表达式为:

```
rage = rage + 1
```

5) 运算符的优先级

在表达式中,运算符是有优先级的,必须先对较高级别的运算符进行运算。如果优先级相同,则按照从左到右的顺序进行运算。运算符的优先级如表 3-5 所示。

表 3-5　运算符优先级

级　别	运　算　符
1	*(乘)、/(除)
2	+(正)、-(负)、+(加)、-(减)
3	=、>、<、>=、<=、<>
4	NOT
5	AND
6	OR、BETWEEN、IN、LIKE、ALL、ANY
7	=(赋值)

在表达式中,可以使用括号来改变运算符的优先级。应首先对括号中的内容进行运算。如果表达式有嵌套的括号,那么首先对嵌套最深的表达式求值。

3.2　数据定义

SQL 支持关系数据库的三级模式结构,其模式、外模式和内模式中包含有基本表、视图和索引。因此,SQL 的数据定义功能包括模式定义、基本表定义、视图定义和索引定义。

3.2.1　模式定义

对于数据库设计者而言,模式代表的是一个完整数据库整体的逻辑设计。对于 SQL 而言,模式只不过是基本表、视图、索引等数据库对象的一个容器。定义模式相当于定义了一个命名空间,在该空间中可以进一步定义各种数据库对象。

1. 创建模式

在 SQL 中,创建模式的语句为 CREATE SCHEMA,其格式为:

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

【例 3-1】 创建一个图书借阅模式 reader_borrow。

解:

```
CREATE SCHEMA  reader_borrow;
```

默认情况下,模式属于创建它的用户,这些用户一般具有 DBA 权限。模式的拥有者是能够修改该模式的唯一用户。如果要将模式分配给其他拥有者而不是创建者本身,可以在创建模式时使用 AUTHORIZATION 子句。

【例 3-2】 为用户 LI 创建一个图书借阅模式 reader_borrow。

解:

```
CREATE SCHEMA  reader_borrow  AUTHORIZATION  LI;
```

或者

```
CREATE SCHEMA  AUTHORIZATION  LI;
```

该语句没有指定模式名，则其用户名即为隐含的模式名，因此其模式名为 LI。

在模式中，创建数据库对象时，有多种方式可供选择。

1）在创建模式时，直接创建数据库对象

```
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
```

【例 3-3】 为用户 LI 创建一个图书借阅模式 reader_borrow，并且在该模式下定义读者数据表 reader。

解：

```
CREATE SCHEMA reader_borrow AUTHORIZATION LI
    CREATE TABLE reader(
      rno CHAR(5) PRIMARY KEY,
      rname CHAR(20),
      rgender CHAR(2),
      rage INT,
      rspecialty CHAR(20)
    );
```

2）在数据库对象前限定模式名

【例 3-4】 在图书借阅模式 reader_borrow 下定义读者数据表 reader。

解：

```
CREATE TABLE reader_borrow.reader(
    rno CHAR(5) PRIMARY KEY,
    rname CHAR(20),
    rgender CHAR(2),
    rage INT,
    rspecialty CHAR(20)
);
```

其中，reader_borrow.reader 中的“reader_borrow”即为限定模式名。

3）使用 SET SCHEMA 语句指定当前模式

语句格式为：

```
SET SCHEMA 模式名
```

其中，模式名即为当前模式。例如，

```
SET SCHEMA reader_borrow;
```

指定当前模式后，就可以直接使用

```
CREATE TABLE reader(
…
);
```

在模式 reader_borrow 下创建数据表 reader。

温馨提示：

本书中所有 SQL 语句均以“；”结尾，但这不是 SQL 的标准语法，只不过大多数数据库

管理系统都使用它，以便将一条 SQL 语句写在多行上。

2. 删除模式

在 SQL 中，删除模式的语句为 DROP SCHEMA，其格式为：

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

其中，CASCADE 和 RESTRICT 两者必选其一。CASCADE（级联）表示在删除模式的同时把该模式中所有的数据库对象一并删除；RESTRICT（限制）表示在删除模式时，如果该模式中已经定义了数据库对象，则拒绝删除该模式。如果在该模式中没有任何数据库对象，则可以删除该模式。

【例 3-5】 删除图书借阅模式 reader_borrow。

解：

```
DROP SCHEMA  reader_borrow CASCADE;
```

该语句删除了 reader_borrow 模式，同时也将该模式中定义的基本表 reader 也一并删除了。

温馨提示：

很多数据库管理系统把“模式”称为“数据库”，因此“创建模式”就变成了“创建数据库”。同时，SQL 语句也由“CREATE SCHEMA”改为“CREATE DATABASE”。

3.2.2　基本表定义

定义基本表就是创建基本表的结构。SQL 使用 CREATE TABLE 语句创建基本表，其格式为：

```
CREATE TABLE 基本表名(属性列名 数据类型 [列级完整性约束]
                    [,属性列名 数据类型 [列级完整性约束]
                    …
                    [,表级完整性约束]
                    [,完整性约束命名子句]]);
```

【例 3-6】 创建读者表 reader。

解：

```
CREATE TABLE reader
(
    rno CHAR(5),                    /* 读者编号 */
    rname CHAR(20),                 /* 姓名 */
    rgender CHAR(2),                /* 性别 */
    rage INT,                       /* 年龄 */
    rspecialty CHAR(20)             /* 专业 */
);
```

该语句在数据库中创建一个空的读者表 reader，并定义该表有 5 个属性列，以及指定每一个属性列的数据类型。

【例 3-7】 创建图书表 book。

解：

```
CREATE TABLE book
(
    bisbn CHAR(11),                    /* 图书 ISBN 号 */
    bname CHAR(50),                    /* 图书名称 */
    bauthor CHAR(40),                  /* 作者 */
    bpublisher CHAR(30),               /* 出版社 */
    bprice NUMERIC(7,2),               /* 价格 */
    bcategory CHAR(2)                  /* 图书类别 */
);
```

该语句在数据库中创建一个空的图书表 book，并定义该表的 6 个属性列，以及每个属性列的数据类型。

【例 3-8】 创建借阅表 borrow。

解：

```
CREATE TABLE borrow
(
    rno CHAR(5),                       /* 读者编号 */
    bisbn CHAR(11),                    /* 图书 ISBN 号 */
    startdate DATE,                    /* 借书日期 */
    enddate DATE,                      /* 还书日期 */
    fine NUMERIC(7,2)                  /* 罚款金额 */
);
```

该语句在数据库中创建一个空的借阅表 borrow，并定义该表的 5 个属性列，以及每个属性列的数据类型。

3.2.3 实现完整性约束

在创建基本表的同时，通常还要定义与该表有关的完整性约束。

1. 实现完整性约束的方法

通常有三种实现完整性约束的方法。

1）列级完整性约束

列级完整性约束针对基本表中的单个属性列设置限定条件，只能应用在一个属性列上。在定义时，将约束直接写在属性列名及其数据类型之后。

【例 3-9】 创建读者表 reader，且在列级实现完整性约束。

解：

```
CREATE TABLE reader
(
    rno CHAR(5) PRIMARY KEY,
    rname CHAR(20),
    rgender CHAR(2),
```

```
    rage INT,
    rspecialty CHAR(20)
);
```

在创建基本表 reader 时，指定其属性列 rno 为主码。

2）表级完整性约束

表级完整性约束针对基本表中涉及的多个属性列设置限定条件，可以应用在多个属性列上。在定义时，将约束写在所有属性列之后。

当需要在一个基本表中的一个属性列上建立约束条件时，可以使用列级完整性约束或者表级完整性约束；当一个约束条件涉及基本表的多个属性列时，只能使用表级完整性约束。

【例 3-10】 创建借阅表 borrow，且在表级实现完整性约束。

解：

```
CREATE TABLE borrow
(
    rno CHAR(5),
    bisbn CHAR(11),
    startdate DATE,
    enddate DATE,
    fine NUMERIC(7,2),
    PRIMARY KEY(rno,bisbn,startdate)
);
```

在创建基本表 borrow 时，指定其属性列 rno、bisbn 和 startdate 为主码。

3）完整性约束命名子句

完整性约束命名子句使用 CONSTRAINT 定义约束，并对完整性约束条件进行命名。其格式为：

```
CONSTRAINT 完整性约束条件名[PRIMARY KEY 短语| FOREIGN KEY 短语| CHECK 短语]
```

【例 3-11】 创建图书表 book，且用完整性约束命名子句实现完整性约束。

解：

```
CREATE TABLE book
(
    bisbn CHAR(11),
    bname CHAR(50),
    bauthor CHAR(40),
    bpublisher CHAR(30),
    bprice NUMERIC(7,2),
    bcategory CHAR(2),
    CONSTRAINT bookkey PRIMARY KEY(bisbn)
);
```

在创建基本表 book 时，使用完整性约束命名子句指定其属性列 bisbn 为主码。

温馨提示：

完整性约束命名子句既可以在列级使用，也可以在表级使用。例 3-11 是在表级使用的例子。

2. 实现实体完整性约束

实体完整性约束是指基本表的任何主属性均不能取空值，即基本表中的每一个元组在主码上的每一个分量都必须有值，且主码取值必须唯一。

在 SQL 中，使用 PRIMARY KEY 来实现实体完整性。其格式为：

```
PRIMARY KEY(属性列名)
```

【例 3-12】 创建读者表 reader，且定义属性列 rno 为主码。

解：

```
CREATE TABLE reader
(
    rno CHAR(5) PRIMARY KEY,
    rname CHAR(20),
    rgender CHAR(2),
    rage INT,
    rspecialty CHAR(20)
);
```

在创建基本表 reader 时，使用列级完整性约束定义属性列 rno 为主码。

【例 3-13】 创建图书表 book，且定义属性列 bisbn 为主码。

解：

```
CREATE TABLE book
(
    bisbn CHAR(11),
    bname CHAR(50),
    bauthor CHAR(40),
    bpublisher CHAR(30),
    bprice NUMERIC(7,2),
    bcategory CHAR(2),
    CONSTRAINT bookkey PRIMARY KEY(bisbn)
);
```

在创建基本表 book 时，使用完整性约束命名子句定义其属性列 bisbn 为主码。

【例 3-14】 创建借阅表 borrow，且定义属性列 rno、bisbn 和 startdate 为主码。

解：

```
CREATE TABLE borrow
(
    rno CHAR(5),
    bisbn CHAR(11),
    startdate DATE,
    enddate DATE,
    fine NUMERIC(7,2) ,
```

```
    PRIMARY KEY(rno,bisbn,startdate)
);
```

在创建基本表 borrow 时，使用表级完整性约束定义其属性列 rno、bisbn 和 startdate 为主码。

温馨提示：

一个基本表只能有一个 PRIMARY KEY。

3. 实现参照完整性约束

参照完整性是指一个基本表的外码要么取值为空，要么取值为被参照表的某一个主码值。

在 SQL 中，使用 FOREIGN KEY 来实现参照完整性。其格式为：

```
FOREIGN KEY(属性列名)
    REFERENCES 被参照表名(属性列名)
```

其中，FOREIGN KEY 后的“属性列名”表示外码；REFERENCES 后的“属性列名”表示被参照表的主码。

【例 3-15】 创建借阅表 borrow，且定义属性列 rno 和 bisbn 为外码。

解：

```
CREATE TABLE borrow
(
    rno CHAR(5),
    bisbn CHAR(11),
    startdate DATE,
    enddate DATE,
    fine NUMERIC(7,2),
    PRIMARY KEY(rno,bisbn),
    FOREIGN KEY(rno) REFERENCES reader(rno),
    FOREIGN KEY(bisbn) REFERENCES book(bisbn)
);
```

在创建基本表 borrow 时，使用表级完整性约束定义其属性列 rno 和 bisbn 为外码。

4. 实现用户定义的完整性约束

用户定义的完整性是用户针对特定要求而规定的一些特殊约束条件。

在 SQL 中，使用 NOT NULL、UNIQUE、CHECK 和 DEFAULT 来实现用户定义的完整性约束。另外，还可以使用触发器来实现用户定义的完整性约束。有关触发器的介绍请参见第 9 章。

1) NOT NULL 与 DEFAULT 约束

NOT NULL 和 NULL 约束是非空和空值约束。当某个属性取值不能为空时，使用 NOT NULL 约束。

DEFAULT 约束是默认值约束。当某个属性可以取默认值时，使用该约束。

【例 3-16】 创建读者表 reader，且定义属性列 rname 取值不能为空。

解：

```
CREATE TABLE reader
(
    rno CHAR(5) PRIMARY KEY,
    rname CHAR(20) NOT NULL,
    rgender CHAR(2),
    rage INT,
    rspecialty CHAR(20)
);
```

在创建基本表 reader 时，使用列级完整性约束定义属性列 rname 取值不能为空。

【例 3-17】 创建图书表 book，且定义属性列 bname、bauthor 和 bisbn 取值不能为空。

解：

```
CREATE TABLE book
(
    bisbn CHAR(11),
    bname CHAR(50) NOT NULL,
    bauthor CHAR(40) NOT NULL,
    bpublisher CHAR(30) NOT NULL,
    bprice NUMERIC(7,2),
    bcategory CHAR(2),
    CONSTRAINT bookkey PRIMARY KEY(bisbn)
);
```

在创建基本表 book 时，使用列级完整性约束定义属性列 bname、bauthor 和 bisbn 取值不能为空。

【例 3-18】 创建借阅表 borrow，且定义属性列 fine 的默认值为 0。

解：

```
CREATE TABLE borrow
(
    rno CHAR(5),
    bisbn CHAR(11),
    startdate DATE,
    enddate DATE,
    fine NUMERIC(7,2) DEFAULT 0,
    PRIMARY KEY(rno,bisbn,startdate)
);
```

在创建基本表 borrow 时，使用列级完整性约束定义属性列 fine 的默认值为 0。即不为 fine 属性列赋值时其取值为 0，当为 fine 属性列赋值时其取值为新赋的值。

2) UNIQUE 约束

UNIQUE 约束也称为唯一性约束，它确保在一个非主码列上值的唯一性。

UNIQUE 与 PRIMARY KEY 的区别是：UNIQUE 约束主要用在非主码的一列或多列上要求值唯一的情况，而 PRIMARY KEY 主要用在确保主码值唯一的情况；在一个基本

表上可以定义多个 UNIQUE 约束，但只能定义一个主码。

【例 3-19】 创建专业表 specialty，且定义属性列 sname 取值必须唯一。

解：

```
CREATE TABLE specialty
(
    sno INT PRIMARY KEY,                    /* 专业编号 */
    sname CHAR(20) NOT NULL UNIQUE          /* 专业名称 */
);
```

在专业表 specialty 中，专业名称 sname 取值必须唯一，不能出现重名现象。例如，取值“计算机”、“信息”、“数学”等。

3）CHECK 约束

CHECK 约束也称为检查约束，它对属性列取值进行约束。其格式为：

```
CHECK(条件表达式)
```

在条件表达式中，可以出现属性列名、常量、运算符、函数等。

【例 3-20】 创建读者表 reader，要求性别只能取值“男”或“女”，年龄取值在 0～100 之间。

解：

```
CREATE TABLE reader
(
    rno CHAR(5) PRIMARY KEY,
    rname CHAR(20) NOT NULL,
    rgender CHAR(2) CHECK(rgender IN('男','女')),
    rage INT CHECK(rage BETWEEN 0 AND 100),
    rspecialty CHAR(20)
);
```

【例 3-21】 创建图书表 book，要求图书类别只能是“理”、“工”或“文”。

解：

```
CREATE TABLE book
(
    bisbn CHAR(11) PRIMARY KEY,
    bname CHAR(50) NOT NULL,
    bauthor CHAR(40) NOT NULL,
    bpublisher CHAR(30) NOT NULL,
    bprice NUMERIC(7,2),
    bcategory CHAR(2) CHECK(bcategory IN('理','工','文')),
);
```

【例 3-22】 创建借阅表 borrow，要求每本书的借阅时间最多为 2 个月。

解：

```
CREATE TABLE borrow
(
    rno CHAR(5),
```

```
    bisbn CHAR(11),
    startdate DATE NOT NULL,
    enddate DATE,
    fine NUMERIC(7,2) ,
    PRIMARY KEY(rno,bisbn),
    FOREIGN KEY(rno) REFERENCES reader(rno),
    FOREIGN KEY(bisbn) REFERENCES book(bisbn),
    CHECK(enddate IS NULL OR MONTH(enddate) - MONTH(startdate)<=2)
);
```

温馨提示：

在例 3-22 中，使用了函数 MONTH，用于获取日期的月份值。

3.2.4 更新基本表

创建基本表以后，可以使用 SQL 对基本表的结构进行修改，也可以删除基本表。

1. 修改基本表

修改基本表包括增加属性列、修改属性列、删除属性列、增加完整性约束和删除完整性约束。

1）增加属性列

增加属性列的格式为：

```
ALTER TABLE 基本表名 ADD 新属性列名 数据类型 [完整性约束]
```

【例 3-23】 为读者表 reader 添加“联系电话”属性。

解：

```
ALTER TABLE reader ADD rphone CHAR(13);
```

温馨提示：

在增加属性列时，不能使用 NOT NULL 约束，以确保原有元组在该列上取空值。

2）增加完整性约束

增加完整性约束的格式为：

```
ALTER TABLE 基本表名 ADD 完整性约束
```

【例 3-24】 已知有一个学院表 colledge，其主码为学院编号 cno。要求为读者表 reader 添加一个属性列 cno（表示读者的所属学院），然后将 cno 设置为外码。

解：

```
ALTER TABLE reader ADD cno INT;
ALTER TABLE reader ADD FOREIGN KEY(cno) REFERENCES colledge(cno);
```

3）删除属性列

删除属性列的格式为：

```
ALTER TABLE 基本表名 DROP COLUMN 属性列名
```

【例 3-25】 删除读者表 reader 的“联系电话”属性。

解：

```
ALTER TABLE reader DROP COLUMN rphone;
```

温馨提示：

删除属性列时，所有引用该列的视图和约束也一起被删除。有的系统规定：当没有视图和约束引用该列时，才允许删除该列。

4）删除完整性约束

删除完整性约束的格式为：

```
ALTER TABLE 基本表名 DROP CONSTRAINT 完整性约束名
```

【例 3-26】 删除借阅表 borrow 的外码。

解：

```
ALTER TABLE borrow DROP CONSTRAINT borrowReader;
ALTER TABLE borrow DROP CONSTRAINT borrowBook;
```

其中，borrowReader 和 borrowBook 为约束名。

5）修改属性列

修改属性列的格式为：

```
ALTER TABLE 基本表名 ALTER COLUMN 属性列名 数据类型
```

【例 3-27】 将读者表 reader 的“联系电话”列的数据类型修改为 CHAR(20)。

解：

```
ALTER TABLE reader ALTER COLUMN rphone CHAR(20);
```

2. 删除基本表

当不需要某个基本表时，可以使用 DROP TABLE 语句删除它。其格式为：

```
DROP TABLE 基本表名 [RESTRICT | CASCADE]
```

其中，RESTRICT 表示删除该表是受到限制的，即被删除的基本表不能被其他表的约束所引用，否则不能被删除。在删除基本表时，默认使用 RESTRICT。

CASCADE 表示级联删除，即删除该表时，其相关的依赖对象（如视图、存储过程等）也一并被删除。

【例 3-28】 删除借阅表 borrow。

解：

```
DROP TABLE borrow CASCADE;
```

温馨提示：

基本表一旦被删除，表中的数据以及在该表上建立的索引、视图等都将被删除，并且无法恢复，因此要小心使用该语句。

3.3 数据查询

数据查询是数据库的核心操作，它能够按照用户的要求从数据库中检索出用户所需要的数据。在 SQL 中，使用 SELECT 语句实现数据查询。SELECT 语句是 SQL 中使用最广泛、方式最灵活、功能最丰富的语句。

SELECT 语句的一般格式为：

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

SELECT 语句包含 SELECT 子句、FROM 子句、WHERE 子句、GROUP BY 子句和 ORDER BY 子句。其中，SELECT 子句和 FROM 子句是必需的，其他子句是可选的。用户需要根据具体需求来选择不同的子句。

各子句的作用为：

- SELECT 子句用于指定查询结果中包含哪些属性列；
- FROM 子句用于指定要查询的数据来自哪些基本表或视图；
- WHERE 子句给出选择基本表或视图中元组的条件；
- GROUP BY 子句将结果按照“列名 1”的值进行分组，即将该列值相等的元组分为一组；
- ORDER BY 子句用于对结果进行排序。

SELECT 语句的执行过程为：根据 WHERE 子句的条件，从 FROM 子句指定的基本表或视图中找出满足条件的元组。再按照 SELECT 子句的目标列表达式，选出元组中的属性列值形成结果集。如果还有 GROUP BY 子句，则将结果集按照“列名 1”的值进行分组，将该属性列值相同的元组分为一组。如果 GROUP BY 子句后带 HAVING 短语，则只有满足指定条件的组才予输出。如果还有 ORDER BY 子句，则结果集还要按照“列名 2”的值进行升序或者降序排列。

3.3.1 简单查询

1. 查询特定列

通常情况下，用户只需要获取基本表中的一部分属性列。这时，可以在 SELECT 子句的“目标列表达式”中指定需要获取的属性列名。

SELECT 子句的格式为：

```
SELECT [ALL | DISTINCT]目标列表达式[,目标列表达式, …]
```

【例 3-29】 查询所有读者的姓名。

解：

```
SELECT rname
```

```
    FROM reader;
```

查询结果为：

rname
王建立
张玲
王军
李建国
赵文娟

【例 3-30】 查询图书馆中所有藏书的 ISBN 号。

解：

```
SELECT bisbn
    FROM book;
```

查询结果为：

bisbn
04-012955-7
04-023909-6
302-00860-4
302-02368-5
302-03314-5
5613-3644-6
81080-159-1

2. 查询多列

如果需要获取多列，则需要在“目标列表达式”中列出这些列的名字，并用“,”号分隔它们。

【例 3-31】 查询所有读者的姓名、编号、专业。

解：

```
SELECT rname, rno, rspecialty
    FROM reader;
```

查询结果为：

rname	rno	rspecialty
王建立	14001	CP
张玲	14002	FR
王军	14003	IF
李建国	14004	CP
赵文娟	14005	MA

【例 3-32】 查询图书馆中所有藏书的图书名称、ISBN 号、出版社。

解：

```
SELECT bname, bisbn, bpublisher
    FROM book;
```

查询结果为：

bname	bisbn	bpublisher
积分变换	04-012955-7	高等教育出版社
线性代数	04-023909-6	高等教育出版社
C 程序设计	302-00860-4	清华大学出版社
数据结构_C 语言版	302-02368-5	清华大学出版社
数据结构题集	302-03314-5	清华大学出版社
读者文摘精粹版	5613-3644-6	陕西师范大学出版社
大学英语	81080-159-1	上海外语教育出版社

温馨提示：

在输出结果中，属性列的输出顺序与“目标列表达式”中属性列出现的顺序完全一致。

3. 查询所有列

如果需要查询所有列，则只需要用“*”来替换“目标列表达式”，当然也可以使用查询多列的方法，列出所有列的名字。

【例 3-33】 查询所有读者的详细信息。

解：

```
SELECT *
    FROM reader;
```

查询结果为：

rno	rname	rgender	rage	rspecialty
14001	王建立	男	18	CP
14002	张玲	女	19	FR
14003	王军	男	20	IF
14004	李建国	男	19	CP
14005	赵文娟	女	18	MA

【例 3-34】 查询图书馆中所有藏书的详细信息。

解：

```
SELECT *
    FROM book;
```

查询结果为：

bisbn	bname	bauthor	bpublisher	bprice	bcategory
04-012955-7	积分变换	张元林	高等教育出版社	8.00	理
04-023909-6	线性代数	贾兰香	高等教育出版社	16.00	理
302-00860-4	C 程序设计	谭浩强	清华大学出版社	20.00	工
302-02368-5	数据结构_C 语言版	严蔚敏	清华大学出版社	22.00	工
302-03314-5	数据结构题集	严蔚敏	清华大学出版社	16.00	工
5613-3644-6	读者文摘精粹版	周宏	陕西师范大学出版社	38.00	文
81080-159-1	大学英语	李荫华	上海外语教育出版社	35.00	文

温馨提示：

在输出结果中，属性列的输出顺序与定义基本表时属性列出现的顺序完全一致。如果需要改变列的输出顺序，只能采用多列方式，将列的名字依序列出。

4. 表达式列

在 SELECT 子句中，“目标列表达式”不仅可以是属性列的序列，还可以是表达式。在表达式中，可以出现常量、属性列、运算符、函数等。

【例 3-35】 查询所有读者的姓名及其出生年份。

解：

```
SELECT rname, 2014 - rage
    FROM reader;
```

查询结果为：

rname	2014-rage
王建立	1996
张玲	1995
王军	1994
李建国	1995
赵文娟	1996

【例 3-36】 查询所有被借图书的借阅年份和罚款金额，要求罚款金额四舍五入为整数。

解：

```
SELECT YEAR(startdate), 'FINES', ROUND(fine, 0)
    FROM borrow;
```

其中，YEAR 和 ROUND 为函数。由于各个数据库管理系统中的函数不尽相同，因此在使用时要查询相应的手册。

查询结果为：

YEAR(startdate)	FINES	ROUND(fine, 0)
2013	FINES	0.00
2013	FINES	2.00
2013	FINES	3.00
2013	FINES	4.00
2013	FINES	0.00
2013	FINES	0.00
2013	FINES	0.00

5. 为列指定别名

通过为列指定别名，可以改变查询结果的列标题，使其显示一个有意义的名字。特别是包含常量、函数、算术表达式等的目标列表达式尤其有意义。其格式为：

目标列表达式　别名

【例 3-37】 查询所有读者的姓名及其出生年份。

解：

```
SELECT rname 姓名, 2014 - rage 出生年份
    FROM reader;
```

查询结果为：

姓名	出生年份
王建立	1996
张玲	1995
王军	1994
李建国	1995
赵文娟	1996

6. 去掉重复值

由于主码的唯一性确保基本表中没有重复行，但去掉基本表中的某些属性列后，就有可能出现重复行。如果在查询时不希望出现重复行，可以使用 DISTINCT。

如果没有指定 DISTINCT，则默认使用 ALL，即保留重复行。

【例 3-38】 查询所有借过书的读者编号。

解：

```
SELECT rno
    FROM borrow
```

查询结果为：

rno
14001
14001
14001
14003
14004
14004
14005

【例 3-39】 查询所有借过书的读者编号，并且查询结果中不包含重复值。

解：

```
SELECT DISTINCT rno
    FROM borrow
```

查询结果为：

rno
14001
14003
14004
14005

3.3.2　条件查询

如果想要查询基本表中的指定元组，就必须使用 WHERE 子句指定选择行的条件。WHERE 子句的格式为：

```
WHERE 条件表达式
```

1. 用关系表达式表示的条件

在 WHERE 子句中，可以使用关系表达式表示选择元组的条件。在关系表达式中，可以出现常量、属性列、关系运算符和函数。

【例 3-40】 显示计算机专业全体读者的名单。

解：

```
SELECT rname
    FROM reader
    WHERE rspecialty = 'CP';
```

查询结果为：

rname
王建立
李建国

【例 3-41】 显示所有年龄在 19 岁以下的读者名单。

解：

```
SELECT rname
    FROM reader
    WHERE rage < 19;
```

查询结果为：

rname
王建立
赵文娟

【例 3-42】 查询有过罚款记录的读者编号。

解：

```
SELECT DISTINCT rno
    FROM borrow
    WHERE fine <> 0;
```

查询结果为：

rno
14001
14003

2. 用逻辑表达式表示的条件

在 WHERE 子句中，可以使用逻辑表达式表示选择元组的条件。通常情况下，使用 AND 和 OR 运算符连接多个查询条件。

【例 3-43】 查询计算机专业所有男读者的姓名。

解：

```
SELECT rname
    FROM reader
    WHERE rspecialty = 'CP' AND rgender = '男';
```

查询结果为：

rname
王建立
李建国

【例 3-44】 查询价格在 20 元以下，且由严蔚敏编著的图书。

解：

```
SELECT bname
    FROM book
    WHERE bauthor = '严蔚敏' AND bprice < 20;
```

查询结果为：

bname
数据结构题集

【例 3-45】 查询由严蔚敏或者谭浩强编著的图书。

解：

```
SELECT bname, bauthor
    FROM book
    WHERE bauthor = '严蔚敏' OR bauthor = '谭浩强';
```

查询结果为：

bname	bauthor
C 程序设计	谭浩强
数据结构_C 语言版	严蔚敏
数据结构题集	严蔚敏

【例 3-46】 查询计算机专业和信息专业所有男读者的编号和姓名。

解：

```
SELECT rno, rname
    FROM reader
    WHERE rspecialty = 'CP' OR rspecialty = 'IF' AND rgender = '男';
```

查询结果为：

rno	rname
14001	王建立
14003	王军
14004	李建国

3. 在指定范围内匹配值

在 WHERE 子句中，可以使用 BETWEEN AND 谓词查找属性值在指定范围内的元组，可以使用 NOT BETWEEN AND 谓词查找属性值不在指定范围内的元组。

【例 3-47】 查询年龄在 17～19 岁之间的读者姓名和专业。

解：

```
SELECT rname, rspecialty
```

```
    FROM reader
    WHERE rage BETWEEN 17 AND 19;
```

或者

```
SELECT rname,rspecialty
    FROM reader
    WHERE rage>=17 AND rage<=19;
```

查询结果为：

rname	rspecialty
王建立	CP
张玲	FR
李建国	CP
赵文娟	MA

【例 3-48】 查询价格不在 16～30 元之间的图书名称和出版社。

解：

```
SELECT bname, bpublisher
    FROM book
    WHERE bprice NOT BETWEEN 16 AND 30;
```

查询结果为：

bname	bpublisher
积分变换	高等教育出版社
读者文摘精粹版	陕西师范大学出版社
大学英语	上海外语教育出版社

【例 3-49】 查询借阅起始日期在 2013 年 5 月至 2013 年 10 月之间的读者编号和图书的 ISBN 号。

解：

```
SELECT rno , bisbn
    FROM borrow
    WHERE startdate BETWEEN '2013/5/1' AND '2013/10/31';
```

查询结果为：

rno	bisbn
14001	302-00860-4
14003	04-012955-7
14004	302-03314-5
14005	302-00860-4

4. 在指定集合内匹配值

在 WHERE 子句中，可以使用 IN 谓词查找属性值在指定集合内的元组，可以使用

NOT IN 谓词查找属性值不在指定集合内的元组。

【例 3-50】 查询计算机专业、信息专业和数学专业的读者姓名和年龄。

解：

```
SELECT rname, rage
    FROM reader
    WHERE rspecialty IN('CP', 'IF', 'MA');
```

或者

```
SELECT rname, rage
    FROM reader
    WHERE rspecialty = 'CP' OR rspecialty = 'IF' OR rspecialty = 'MA';
```

查询结果为：

rname	rage
王建立	18
王军	20
李建国	19
赵文娟	18

【例 3-51】 查询图书价格为 8 元、16 元或 35 元的图书名称。

解：

```
SELECT bname
    FROM book
    WHERE bprice IN(8 , 16 , 35);
```

查询结果为：

bname
积分变换
线性代数
数据结构题集
大学英语

【例 3-52】 查询既不是计算机专业、信息专业，也不是数学专业的读者姓名和年龄

解：

```
SELECT rname, rage
    FROM reader
    WHERE rspecialty NOT IN('CP', 'IF', 'MA');
```

查询结果为：

rname	rage
张玲	19

3.3.3 聚集函数

聚集函数是一类能够完成统计任务的函数。常用的聚集函数如表 3-6 所示。

表 3-6 聚集函数

函 数	功 能
COUNT([DISTINCT | ALL] *)	统计元组个数
COUNT([DISTINCT | ALL]列名)	计算一列中值的个数
SUM([DISTINCT | ALL]列名)	计算一列值的总和
AVG([DISTINCT | ALL]列名)	计算一列值的平均值
MAX([DISTINCT | ALL]列名)	求一列值中的最大值
MIN([DISTINCT | ALL]列名)	求一列值中的最小值

温馨提示：

- 如果指定 DISTINCT，则表示在计算时去掉指定列中的重复值。如果不指定 DISTINCT 或者指定 ALL，则表示在计算时不去掉指定列中的重复值。
- SUM 函数和 AVG 函数只对数值型的列进行计算，其他类型的列不能使用这两个函数。
- 在 SQL 中，不允许对聚集函数进行复合运算。例如，

```
SELECT MAX(COUNT(DISTINCT rno))
    FROM borrow;
```

是错误的，不能以“MAX(COUNT( ))”的形式使用聚集函数。

- 聚集函数只能用于 SELECT 子句的目标列表达式中，或者 HAVING 短语（参见 3.3.6 节）中，不能用在 WHERE 子句中。

【例 3-53】 查询图书馆的藏书数量。

解：

```
SELECT COUNT( * ) quantity
    FROM book;
```

查询结果为：

quantity
7

【例 3-54】 查询有多少读者借阅了图书。

解：

```
SELECT COUNT(DISTINCT rno) people
    FROM borrow;
```

查询结果为：

people
7

【例 3-55】 查询编号为 14004 的读者借过多少本书。

解：

```
SELECT COUNT( * ) amount
    FROM borrow
    WHERE rno = '14004';
```

查询结果为：

amount
2

【例 3-56】 查询编号为 14001 的读者被罚款的总金额。

解：

```
SELECT SUM(fine) amount
    FROM borrow
    WHERE rno = '14001';
```

查询结果为：

amount
5.00

【例 3-57】 查询加收 20%罚款的罚款金额。

解：

```
SELECT SUM(fine) * 1.2 amount
    FROM borrow
```

查询结果为：

amount
10.20

该语句也可以写成如下形式：

```
SELECT SUM(fine * 1.2) amount
    FROM borrow
```

【例 3-58】 查询图书的平均价格。

解：

```
SELECT AVG(bprice) price
```

```
    FROM book
```

查询结果为：

price
22.14

【例 3-59】 查询读者的平均年龄。

解：

```
SELECT AVG(rage) age
    FROM reader
```

查询结果为：

age
18

【例 3-60】 查询图书的最高价。

解：

```
SELECT MAX(bprice) price
    FROM book
```

查询结果为：

price
38.00

【例 3-61】 查询“理”类图书的最低价。

解：

```
SELECT MIN(bprice) price
    FROM book
    WHERE bcategory = '理';
```

查询结果为：

price
8.00

3.3.4 模糊查询

在查询时，有时查询条件并不是很精确，比如只知道要查询读者的姓氏，而不知道具体名字，这时就可以使用模糊查询，对姓氏进行匹配，而不查找全称。

在 SELECT 语句中，通过在 WHERE 子句中使用 LIKE 谓词，可以实现模糊查询。其

格式为：

```
[NOT]LIKE 匹配串[ESCAPE 转义字符]
```

其中，匹配串中可以包含通配符“%”和“_”，分别代表任意长度的字符串和任意单个字符。

【例 3-62】 查询 14001 读者的姓名。

解：

```
SELECT rname
    FROM reader
    WHERE rno LIKE '14001';
```

查询结果为：

rname
王建立

该查询中的条件“rno LIKE '14001'”，相当于“rno＝'14001'”，因此，从严格意义上讲为精确查询。

【例 3-63】 查询所有姓“王”的读者的姓名。

解：

```
SELECT rname
    FROM reader
    WHERE rname LIKE '王%';
```

查询结果为：

rname
王建立
王军

【例 3-64】 查询姓“王”且全名为两个汉字的读者姓名。

解：

```
SELECT rname
    FROM reader
    WHERE rname LIKE '王__';
```

查询结果为：

rname
王军

【例 3-65】 查询有关“数据结构”的图书。

解：

```
SELECT bname
```

```
    FROM book
    WHERE bname LIKE '%数据结构%';
```

查询结果为：

bname
数据结构_C 语言版
数据结构题集

【例 3-66】 查询所有不姓“赵”的读者姓名。

解：

```
SELECT rname
    FROM reader
    WHERE rname NOT LIKE '赵%';
```

查询结果为：

rname
王建立
张玲
王军
李建国

【例 3-67】 查询图书名中包含“_”的图书名称。

解：

```
SELECT bname
    FROM book
    WHERE bname LIKE '%\_%' ESCAPE '\';
```

查询结果为：

bname
数据结构_C 语言版

【例 3-68】 查询以“数据结构”开头，且倒数第二个汉字为“题”的图书名称。

解：

```
SELECT bname
    FROM book
    WHERE bname LIKE '数据结构%题__';
```

查询结果为：

bname
数据结构题集

3.3.5　对查询结果排序

如果想要对查询结果进行排序，使之按照升序或者降序顺序排列，可以使用 ORDER BY 子句。在 ORDER BY 子句中，可以指定按照一个属性列或者多个属性列进行升序或者降序排序。使用 ASC 表示升序，使用 DESC 表示降序，默认为升序。

ORDER BY 子句的格式为：

```
ORDER BY 列名[ASC | DESC]
```

【例 3-69】 查询“清华大学出版社”出版的图书名称和价格，查询结果按价格的降序排列。

解：

```
SELECT bname, bprice
    FROM book
    WHERE bpublisher = '清华大学出版社'
    ORDER BY bprice DESC;
```

查询结果为：

bname	bprice
数据结构_C 语言版	22.00
C 程序设计	20.00
数据结构题集	16.00

【例 3-70】 查询全部读者的姓名、年龄和专业，查询结果按专业编码升序排列，同一专业的读者按年龄降序排列。

解：

```
SELECT rname, rage,rspecialty
    FROM reader
    ORDER BY rspecialty, rage DESC;
```

查询结果为：

rname	rage	rspecialty
李建国	19	CP
王建立	18	CP
张玲	19	FR
王军	20	IF
赵文娟	18	MA

【例 3-71】 查询全部读者的姓名、年龄和专业，查询结果按读者年龄升序排列，同一年龄的读者按专业编码降序排列。

解：

```
SELECT rname, rage,rspecialty
```

```
FROM reader
ORDER BY rage, rspecialty DESC;
```

查询结果为：

rname	rage	rspecialty
赵文娟	18	MA
王建立	18	CP
张玲	19	FR
李建国	19	CP
王军	20	IF

3.3.6 分组查询

如果想要对元组进行分组，然后针对每一组中的元组进行运算，可以使用 GROUP BY 子句。在 GROUP BY 子句中，可以指定按照一列值或者多列值进行分组，列值相同的分为一组。

如果分组后只想对满足条件的分组进行运算，可以使用 HAVING 短语指定筛选分组的条件。

一般情况下，分组和聚集函数一起使用，将聚集函数作用于每个分组，而不是作用于整个查询结果。

GROUP BY 子句的格式为：

```
GROUP BY 列名[HAVING 条件表达式]
```

【例 3-72】 查询图书馆藏书中各出版社的图书数量。

解：

```
SELECT bpublisher, COUNT(bpublisher) quantity
    FROM book
    GROUP BY bpublisher;
```

查询结果为：

bpublisher	quantity
高等教育出版社	2
清华大学出版社	3
陕西师范大学出版社	1
上海外语教育出版社	1

本查询按照出版社进行分组，将同一出版社的图书分为一组，然后针对每一组使用聚集函数 COUNT 统计该组的图书数量。

【例 3-73】 统计每个读者被罚款的金额。

解：

```
SELECT rno, SUM(fine) fee
```

```
    FROM borrow
    GROUP BY rno;
```

查询结果为：

rno	fee
14001	5.00
14003	3.50
14004	0.00
14005	0.00

【例 3-74】 查询出版两本及以上图书的出版社名称及图书数量。

解：

```
SELECT bpublisher, COUNT(bpublisher) quantity
    FROM book
    GROUP BY bpublisher HAVING COUNT( * )>= 2;
```

查询结果为：

bpublisher	quantity
高等教育出版社	2
清华大学出版社	3

本查询按照出版社进行分组，将同一出版社的图书分为一组，然后使用 HAVING 短语筛选符合条件的组，即选出该组中图书数量大于等于 2 的分组。最后针对筛选出来的组使用聚集函数 COUNT 统计该组的图书数量。

【例 3-75】 查询罚款金额在 3 元以上的读者编号及罚款金额。

解：

```
SELECT rno, SUM(fine) fee
    FROM borrow
    GROUP BY rno HAVING SUM(fine)> 3;
```

查询结果为：

rno	fee
14001	5.00
14003	3.50

【例 3-76】 统计年龄在 20 岁以下各年龄段读者人数。

解：

```
SELECT rage, COUNT( * ) people
    FROM reader
    GROUP BY rage HAVING rage < 20;
```

或者

```
SELECT rage, COUNT( * ) people
    FROM reader
    WHERE rage < 20
    GROUP BY rage;
```

查询结果为：

rage	people
18	2
19	2

温馨提示：

- 虽然 WHERE 子句与 HAVING 短语都可以完成筛选的任务，但它们是有区别的。WHERE 子句用于从基本表或视图中选择满足条件的元组，而 HVING 短语用于从所在的分组中选择满足条件的组。
- 在 SELECT 子句的目标列表达式中，只能出现 GROUP BY 子句中的分组依据列或者聚集函数。例如，以下查询是错误的：

```
SELECT rname
    FROM reader
    GROUP BY rage
```

- 在 HAVING 短语中，可以使用聚集函数，但在 GROUP BY 中不能使用聚集函数。例如，以下查询是错误的：

```
SELECT rno
    FROM borrow
    GROUP BY SUM(fine)> 3
```

3.3.7 涉及空值的查询

空值就是“不知道”或“不存在”的值。比如某本书被借出了，但还未还回，则该本书的还书日期就为空。

在查询中，可以使用 IS NULL 判断是否为空，使用 IS NOT NULL 判断是否非空。

【例 3-77】 查询已借出但还没有还回的图书的编号及借阅起始日期。

解：

```
SELECT bisbn, startdate
    FROM borrow
    WHERE enddate IS NULL;
```

查询结果为：

bisbn	startdate
302-00860-4	2013-08-06
302-03314-5	2013-05-09

【例 3-78】　查询曾经借出过但已被还回的图书的编号及借阅起始日期。

解：

```
SELECT bisbn, startdate
    FROM borrow
    WHERE enddate IS NOT NULL;
```

查询结果为：

bisbn	startdate
302-02368-5	2013-03-10
302-03314-5	2013-03-10
04-012955-7	2013-10-07
302-00860-4	2013-01-09
302-00860-4	2013-06-18

聚集函数在遇到空值时，除了 COUNT(*)以外，都跳过空值而只处理非空值。

【例 3-79】　统计 14001 读者已借出但还没有还回的图书数量。

解：

```
SELECT COUNT( * ) - COUNT(enddate) quantity
    FROM borrow
    WHERE rno = '14001';
```

查询结果为：

quantity
1

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

【例 3-80】　查询已借图书的编号及还书日期，查询结果按还书日期升序排序。

解：

```
SELECT bisbn , enddate
    FROM borrow
    ORDER BY enddate;
```

查询结果为：

bisbn	enddate
302-00860-4	
302-03314-5	
302-00860-4	2013-03-09
302-02368-5	2013-06-15
302-00860-4	2013-07-10
302-03314-5	2013-07-12
04-012955-7	2013-12-20

温馨提示：

涉及＋、－、＊、/的算术表达式中有一个值是空值，则表达式的值也是空值；涉及空值的比较操作的结果，认为是 false。

3.3.8 连接查询

前面的查询均是针对一个基本表进行的，而更一般的查询会涉及多个基本表。这时，为了查询多个基本表中的数据，就要使用连接操作，将多个基本表连接起来，然后再进行查询，这种查询就是连接查询。

1. 连接的两种表示方法

1）使用 WHERE 子句表示连接

在 WHERE 子句中，添加用来连接两个表的条件。这个条件称为连接条件或连接谓词。其表示方法为：

```
WHERE [表名 1.]列名 1 比较运算符[表名 2.]列名 2
```

其中，列名称为连接字段，一般要求连接条件中的连接字段的类型必须是可比的，但名字不必相同；比较运算符主要包括＝、＞、＜、＞＝、＜＝和＜＞。

【例 3-81】 查询每个读者及其所借图书的情况。

解：

```
SELECT reader.*, borrow.*
    FROM reader, borrow
    WHERE reader.rno = borrow.rno;
```

查询结果为：

reader.rno	rname	rgender	rage	rspecialty	borrow.rno	bisbn	startdate	enddate	fine
14001	王建立	男	18	CP	14001	302-00860-4	2013-08-06		0.00
14001	王建立	男	18	CP	14001	302-02368-5	2013-03-10	2013-06-15	2.00
14001	王建立	男	18	CP	14001	302-03314-5	2013-03-10	2013-07-12	3.00
14003	王军	男	20	IF	14003	04-012955-7	2013-10-07	2013-12-20	3.50
14004	李建国	男	19	CP	14004	302-00860-4	2013-01-09	2013-03-09	0.00
14004	李建国	男	19	CP	14004	302-03314-5	2013-05-09		0.00
14005	赵文娟	女	18	MA	14005	302-00860-4	2013-06-18	2013-07-10	0.00

当连接运算符为“＝”时，称为等值连接。使用其他运算符时，称为非等值连接。若在等值连接中去掉目标列中重复的属性列，则为自然连接。

【例 3-82】 使用自然连接查询每个读者及其所借图书的情况。

解：

```
SELECT reader.rno,rname,rgender,rage,rspecialty,bisbn,startdate,enddate,fine
    FROM reader, borrow
    WHERE reader.rno = borrow.rno;
```

查询结果为：

rno	rname	rgender	rage	rspecialty	bisbn	startdate	enddate	fine
14001	王建立	男	18	CP	302-00860-4	2013-08-06		0.00
14001	王建立	男	18	CP	302-02368-5	2013-03-10	2013-06-15	2.00
14001	王建立	男	18	CP	302-03314-5	2013-03-10	2013-07-12	3.00
14003	王军	男	20	IF	04-012955-7	2013-10-07	2013-12-20	3.50
14004	李建国	男	19	CP	302-00860-4	2013-01-09	2013-03-09	0.00
14004	李建国	男	19	CP	302-03314-5	2013-05-09		0.00
14005	赵文娟	女	18	MA	302-00860-4	2013-06-18	2013-07-10	0.00

2）使用 FROM 子句表示连接

在 FROM 子句中，添加用来连接两个表的条件。这种表示方法更为简单和灵活。其表示方法为：

```
FROM 表名 1 JOIN 表名 2 ON 表名 1.列名 1 比较运算符 表名 2.列名 2
```

【例 3-83】 查询每个读者及其所借图书的情况。

解：

```
SELECT reader.rno,rname,rgender,rage,rspecialty,bisbn,startdate,enddate,fine
    FROM reader JOIN borrow ON reader.rno = borrow.rno;
```

查询结果为：

rno	rname	rgender	rage	rspecialty	bisbn	startdate	enddate	fine
14001	王建立	男	18	CP	302-00860-4	2013-08-06		0.00
14001	王建立	男	18	CP	302-02368-5	2013-03-10	2013-06-15	2.00
14001	王建立	男	18	CP	302-03314-5	2013-03-10	2013-07-12	3.00
14003	王军	男	20	IF	04-012955-7	2013-10-07	2013-12-20	3.50
14004	李建国	男	19	CP	302-00860-4	2013-01-09	2013-03-09	0.00
14004	李建国	男	19	CP	302-03314-5	2013-05-09		0.00
14005	赵文娟	女	18	MA	302-00860-4	2013-06-18	2013-07-10	0.00

温馨提示：

如果在两个表的所有相同名字列上进行连接时，可以省略 ON 子句。例如上例可以写成如下形式：

```
SELECT reader.rno,rname,rgender,rage,rspecialty,bisbn,startdate,enddate,fine
    FROM reader JOIN borrow
```

2. 复合条件连接

所谓复合条件连接，就是在 WHERE 子句中包含多个连接条件。

【例 3-84】 查询王建立所借图书的图书 ISBN 号。

解：

```
SELECT reader.rname, borrow.bisbn
    FROM reader, borrow
    WHERE reader.rno = borrow.rno AND reader.rname = '王建立';
```

查询结果为：

rname	bisbn
王建立	302-00860-4
王建立	302-02368-5
王建立	302-03314-5

3. 多表连接

连接操作可以针对一个表(自身连接)、两个表，以及两个以上表，后者就称为多表连接。

【例 3-85】 查询每个读者的读者编号、姓名及其所借图书的名称。

解：

```
SELECT r.rno, r.rname, bk.bname
    FROM reader r, borrow b, book bk
    WHERE r.rno = b.rno AND b.bisbn = bk.bisbn;
```

其中，“r.rno＝b.rno”为 reader 表和 borrow 表的连接条件，“b.bisbn＝bk.bisbn”为 book 表和 borrow 表的连接条件。查询结果为：

rno	rname	bname
14001	王建立	C 程序设计
14001	王建立	数据结构_C 语言版
14001	王建立	数据结构题集
14003	王军	积分变换
14004	李建国	C 程序设计
14004	李建国	数据结构题集
14005	赵文娟	C 程序设计

温馨提示：

为了简化 SQL 语句的书写，可以为表取别名。在例 3-85 中，为 reader 表取别名为 r，为 borrow 表取别名为 b，为 book 表取别名为 bk。取别名后，就可以在 SQL 语句的各个子句中使用别名来代替表名。

4. 自身连接

当一个表与其自身连接时，这种连接操作称为自身连接。

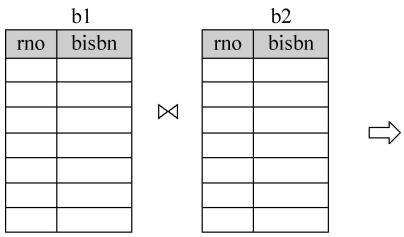
【例 3-86】 查询借阅 ISBN 号为“302-02368-5”和“302-03314-5”两本书的读者。

解：

```
SELECT b1.rno, b1.bisbn ,b2.bisbn
```

```
FROM borrow b1, borrow b2
WHERE b1.rno = b2.rno
    AND b1.bisbn = '302 - 02368 - 5' AND b2.bisbn = '302 - 03314 - 5';
```

可以将 b1 和 b2 看成两个表，然后将两个表连接起来，连接过程如图 3-2 所示。



b1.mo	b1.bisbn	b2.rno	b2.bisbn
14001	302-00860-4	14001	302-00860-4
14001	302-00860-4	14001	302-02368-5
14001	302-00860-4	14001	302-03314-5
14001	302-02368-5	14001	302-02368-5
14001	302-02368-5	14001	302-03314-5
14001	302-03314-5	14001	302-03314-5
14003	04-012955-7	14003	04-012955-7
14004	302-00860-4	14004	302-00860-4
14004	302-00860-4	14004	302-03314-5
14004	302-03314-5	14004	302-03314-5
14005	302-00860-4	14005	302-00860-4

图 3-2　自身连接过程

查询结果为：

b1. rno	b1. bisbn	b2. bisbn
14001	302-02368-5	302-03314-5

如果写成如下形式：

```
SELECT rno, bisbn
    FROM borrow
    WHERE bisbn = '302 - 02368 - 5' AND bisbn = '302 - 03314 - 5';
```

则是错误的。因为每个元组的 bisbn 分量值只能有一个，不可能既是“302-02368-5”，又是“302-03314-5”。

5. 外连接

在前面的例子中，查询结果集中都是符合连接条件的元组，这种连接称为内连接。与之相对应的是在查询结果集中保留那些不满足连接条件的元组，这种连接称为外连接。

外连接分为左外连接和右外连接，分别用 LEFT OUTER 和 RIGHT OUTER 表示。左外连接的查询结果集中保留连接条件左边基本表中的所有元组，而不管这些元组是否满足连接条件。对于那些不满足连接条件的元组，其右表的属性上填空值(NULL)。右外连接与之相反，查询结果集中保留右边基本表中的所有元组。

【例 3-87】 查询每个读者及其所借图书的情况，即使没有借书的读者也要显示其信息。

解：

```
SELECT reader.rno, rname, rage , bisbn, startdate
    FROM reader LEFT OUTER JOIN borrow ON(reader.rno = borrow.rno);
```

查询结果为：

rno	rname	rage	bisbn	startdate
14001	王建立	18	302-00860-4	2013-08-06
14001	王建立	18	302-00860-4	2013-03-10
14001	王建立	18	302-00860-4	2013-03-10
14002	张玲	19		
14003	王军	20	04-012955-7	2013-10-07
14004	李建国	19	302-00860-4	2013-01-09
14004	李建国	19	302-03314-5	2013-05-09
14005	赵文娟	18	302-00860-4	2013-06-18

本查询的结果集中包含不满足连接条件的元组“(14002,张玲,19)”,其 bisbn 和 startdate 分量均以 NULL 值填充。

3.3.9 嵌套查询

在 SQL 语言中,一个 SELECT-FROM-WHERE 语句称为一个查询块。当一个查询块嵌套在另一个查询块中作为选择条件的一部分时,该查询称为嵌套查询。通过使用多个简单查询的嵌套,可以构造出复杂查询,从而增强查询能力。这种层层嵌套的方式,体现了 SQL(Structured Query Language,数据查询语言)中“结构化”的含义。

【例 3-88】 查询借阅“302-02368-5”图书的读者姓名。

下面先分步完成该查询,然后再构造嵌套查询。

(1) 确定借阅“302-02368-5”图书的读者编号。

```
SELECT rno
    FROM borrow
    WHERE bisbn = '302 - 02368 - 5';
```

查询结果为：

rno
14001

(2) 根据读者编号,查找读者姓名。

```
SELECT rname
    FROM reader
    WHERE rno IN('14001');
```

查询结果为：

rname
王建立

将第(1)步作为查询块嵌套到第(2)步的查询块中，形成嵌套查询如下：

```
SELECT rname
    FROM reader
    WHERE rno IN
        (SELECT rno
            FROM borrow
            WHERE bisbn = '302 - 02368 - 5');
```

内部查询块“SELECT rno FROM borrow WHERE bisbn＝'302-02368-5'”嵌套在外部查询块“SELECT rname FROM reader WHERE rno IN”的 WHERE 子句中。其中，外部的查询块称为父查询或外层查询，内部的查询块称为子查询或内层查询。

温馨提示：

在嵌套查询中，子查询不能使用 ORDER BY 子句。ORDER BY 子句只能对最终查询结果排序，因此 ORDER BY 子句只能出现在父查询中。

1. 带有 IN 谓词的子查询

在嵌套查询中，子查询的结果往往是集合，因此常使用 IN 谓词。

【例 3-89】 查询已借书但还未还书的读者姓名。

解：

```
SELECT rname
    FROM reader
    WHERE rno IN
        (SELECT rno
            FROM borrow
            WHERE enddate IS NULL);
```

查询结果为：

rname
王建立
李建国

【例 3-90】 查询与“王建立”同一专业的读者姓名。

解：

```
SELECT rname
    FROM reader
    WHERE rname <>'王建立'
        AND rspecialty IN
            (SELECT rspecialty
                FROM reader
                WHERE rname = '王建立');
```

查询结果为：

rname
李建国

2. 带有比较运算符的子查询

当用比较运算符连接父查询和子查询时，要看子查询的返回结果。如果查询返回单个值，则可以直接使用比较运算符连接父查询与子查询。

常用的比较运算符包括>、<、>=、<=、=和<>。

【例 3-91】 查询罚款金额超过平均罚款金额的读者编号。

解：

```
SELECT DISTINCT rno
    FROM borrow
    WHERE fine >
        (SELECT AVG(fine)
            FROM borrow
            WHERE fine <> 0);
```

查询结果为：

rno
14001
14003

【例 3-92】 查询比“张玲”年龄小的读者姓名。

解：

```
SELECT rname
    FROM reader
    WHERE rage <
        (SELECT rage
            FROM reader
            WHERE rname = '张玲');
```

查询结果为：

rname
王建立
赵文娟

3. 带有 ANY(SOME)或 ALL 谓词的子查询

如果子查询返回多个值，则需要 ANY(SOME)或 ALL 谓词与比较运算符一起使用。ANY(SOME)表示子查询中某个值，ALL 表示子查询中的所有值。当使用 ANY(SOME 与 ANY 等价)谓词时，子查询返回的多个值之间是或的关系；当使用 ALL 谓词时，子查询

返回的多个值之间是与的关系。

【例 3-93】 查询至少有一本书的罚款金额超过 14001 读者一本书的罚款金额的读者编号。

解：

```
SELECT DISTINCT rno
    FROM borrow
    WHERE fine > ANY
        (SELECT fine
            FROM borrow
            WHERE rno = '14001');
```

查询结果为：

rno
14001
14003

本查询的子查询结果为{2.00,3.00,0.00},则该查询相当于

```
SELECT DISTINCT rno
    FROM borrow
    WHERE fine > 2.00 OR fine > 3.00 OR fine > 0.00;
```

本查询也可以使用聚集函数来实现,具体的 SQL 语句为：

```
SELECT DISTINCT rno
    FROM borrow
    WHERE fine >
        (SELECT MIN(fine)
            FROM borrow
            WHERE rno = '14001');
```

【例 3-94】 查询其他专业中比计算机专业某一读者年龄大的读者姓名和年龄。

解：

```
SELECT rname, rage
    FROM reader
    WHERE rspecialty <>'CP'
        AND rage > ANY
            (SELECT rage
                FROM reader
                WHERE rspecialty = 'CP');
```

查询结果为：

rname	rage
张玲	19
王军	20

本查询也可以使用聚集函数来实现，具体的 SQL 语句为：

```
SELECT rname, rage
    FROM reader
    WHERE rspecialty<>'CP'
        AND rage>
            (SELECT MIN(rage)
                FROM reader
                WHERE rspecialty = 'CP');
```

【例 3-95】 查询其他专业中比计算机专业所有读者年龄都大的读者姓名和年龄。

解：

```
SELECT rname, rage
    FROM reader
    WHERE rspecialty<>'CP'
        AND rage> ALL
            (SELECT rage
                FROM reader
                WHERE rspecialty = 'CP');
```

查询结果为：

rname	rage
王军	20

本查询的子查询结果为{18,19}，则该查询相当于

```
SELECT rname, rage
        FROM reader
        WHERE rspecialty<>'CP' AND rage> 18 AND rage> 19;
```

本查询也可以使用聚集函数来实现，具体的 SQL 语句为：

```
SELECT rname, rage
    FROM reader
    WHERE rspecialty<>'CP'
        AND rage>
            (SELECT MAX(rage)
                FROM reader
                WHERE rspecialty = 'CP');
```

【例 3-96】 查询没被借出过的图书名称。

解：

```
SELECT bname
    FROM book
    WHERE bisbn<> ALL
        (SELECT DISTINCT bisbn
            FROM borrow);
```

查询结果为：

bname
线性代数
读者文摘精粹版
大学英语

本查询也可以使用以下 SQL 语句实现：

```
SELECT bname
    FROM book
    WHERE bisbn NOT IN
        (SELECT DISTINCT bisbn
            FROM borrow);
```

综上所述，ANY(SOME)、ALL 可以用聚集函数替换，它们与聚集函数的关系如表 3-7 所示。

表 3-7　ANY(SOME)、ALL 与聚集函数的关系

谓词 \ 比较运算符	<	<=	>	>=	=	<>
ANY(SOME)	< MAX	<= MAX	> MIN	>= MIN	IN	
ALL	< MIN	<= MIN	> MAX	>= MAX		NOT IN

4. 相关子查询

在前面的嵌套查询中，子查询的查询条件不依赖于父查询，这类子查询称为不相关子查询。与之相对应的是子查询的查询条件依赖于父查询，这类查询称为相关子查询。

不相关子查询的查询过程是：先执行子查询，并用子查询的结果构造父查询的条件，然后再执行父查询。

相关子查询的查询过程比较复杂，由于父查询和子查询相互关联，需要反复求值，而不像非相关子查询那样，一次将子查询求解出来，然后求解父查询。其查询过程是：先将父查询的一个元组的相关属性值传递给子查询，并执行子查询，用得到的子查询的结果值构造父查询的条件，并执行父查询。然后再取出父查询的下一个元组，重复上述过程，直至父查询的所有元组全部处理完毕为止。

【例 3-97】 查询所有借过“302-00860-4”图书的读者编号和读者姓名。

解：

```
SELECT rno, rname
    FROM reader
    WHERE '302 - 00860 - 4' IN
        (SELECT bisbn
            FROM borrow
            WHERE rno = reader.rno);
```

查询结果为：

rno	rname
14001	王建立
14004	李建国
14005	赵文娟

【例 3-98】 查询每位读者罚款金额最高的图书编号和罚款金额。

解：

```
SELECT rno, bisbn, fine
    FROM borrow b1
    WHERE fine =
        (SELECT MAX(fine)
            FROM borrow b2
            WHERE b1.rno = b2.rnoAND b2.fine <> 0);
```

查询结果为：

rno	bisbn	fine
14001	302-03314-5	3.00
14003	04-012955-7	3.50

温馨提示：

子查询一定要跟在比较运算符和谓词之后，下列写法是错误的：

```
SELECT rname
    FROM reader
    WHERE
        (SELECT rno
            FROM borrow
            WHERE enddate IS NULL) = '302 - 00860 - 4';
```

5. 带有 EXISTS 谓词的子查询

EXISTS 代表存在量词，用于判断子查询的结果是否存在。带有 EXISTS 谓词的子查询不返回任何数据，只返回逻辑真值(true)或逻辑假值(false)。当子查询的结果集不为空时，返回逻辑真值；当子查询的结果集为空时，返回逻辑假值。

与之相对应的是 NOT EXISTS。当子查询的结果集不为空时，返回逻辑假值；当子查询的结果集为空时，返回逻辑真值。

【例 3-99】 查询所有借过“302-00860-4”图书的读者编号和姓名。

解：

```
SELECT rno, rname
    FROM reader
    WHERE EXISTS
        (SELECT *
```

```
        FROM borrow
        WHERE rno = reader.rno AND bisbn = '302 - 00860 - 4');
```

查询结果为：

rno	rname
14001	王建立
14004	李建国
14005	赵文娟

本查询中，首先从父查询的 reader 表中取出第一个元组(14001，王建立)，将该元组的分量 rno 的值“14001”传递给子查询，执行子查询，获取 borrow 表中的一个元组(14001，302-00860-4)。由于子查询的结果集非空，则父查询的 WHERE 子句返回真值，因此父查询将元组(14001，王建立)放入查询的结果集中。然后父查询再取下一个元组(14002，张玲)，重复这一过程，直至 reader 表中的所有元组都处理一遍为止。

本查询的另一种解法为：

```
SELECT rno, rname
    FROM reader
    WHERE rno IN
        (SELECT rno
            FROM borrow
            WHERE bisbn = '302 - 00860 - 4');
```

或者

```
SELECT reader.rno, rname
    FROM reader , borrow
    WHERE reader.rno = borrow.rno AND bisbn = '302 - 00860 - 4';
```

【例 3-100】 查询没有借过“302-00860-4”图书的读者编号和姓名。

解：

```
SELECT rno, rname
    FROM reader
    WHERE NOT EXISTS
        (SELECT *
            FROM borrow
            WHERE rno = reader.rnoAND bisbn = '302 - 00860 - 4');
```

或者

```
SELECT rno, rname
    FROM reader
    WHERE rno NOT IN
        (SELECT rno
            FROM borrow
            WHERE bisbn = '302 - 00860 - 4');
```

查询结果为：

rno	rname
14002	张玲
14003	王军

【例 3-101】 查询没被借出过的图书名称。

解：

```
SELECT bname
    FROM book
    WHERE NOT EXISTS
        (SELECT *
            FROM borrow
            WHERE bisbn = book.bisbn);
```

查询结果为：

bname
线性代数
读者文摘精粹版
大学英语

【例 3-102】 查询借过所有“工”类图书的读者编号和姓名。

实际上这是一个带有全称量词的查询，然而 SQL 中没有全称量词。为此可以将带有全称量词的谓词转换为等价的带有存在量词的谓词：

$$(\forall x)P \equiv \neg(\exists x(\neg P))$$

这样就可以用双嵌套的 NOT EXISTS 来实现带全称量词的查询。

通过以上分析，可以这样表述该查询：查询这样的读者，没有一本“工”类图书他没借过。因此，SQL 语句为：

```
SELECT rno, rname
    FROM reader
    WHERE NOT EXISTS
        (SELECT *
            FROM book
            WHERE bcategory = '工' AND NOT EXISTS
                (SELECT *
                    FROM borrow
                    WHERE rno = reader.rno AND bisbn = book.bisbn));
```

查询结果为：

rno	rname
14001	王建立

【例 3-103】 查询至少借阅了 14004 号读者借过的全部书籍的读者编号。

该查询也可以这样表述：不存在这样的读者，14004 号读者借过的图书他没借过。因

此，SQL 语句为：

```
SELECT DISTINCT rno
    FROM borrow b1
    WHERE NOT EXISTS
        (SELECT *
            FROM borrow b2
            WHERE b2.rno = '14004' AND NOT EXISTS
                (SELECT *
                    FROM borrow b3
                    WHERE b3.rno = b1.rno  AND b3.bisbn = b2.bisbn));
```

查询结果为：

rno
14001
14004

温馨提示：

由 EXISTS 引出的子查询，其目标列表达式均使用“*”，因为带 EXISTS 的子查询只返回真或假值，给出列名没有实际意义。

所有带 IN 谓词、比较运算符、ANY(SOME)和 ALL 谓词的子查询都能用带 EXISTS 谓词的子查询等价替换，但反之不成立。

3.3.10　集合查询

SQL 采用集合操作模式，其操作对象和查找结果都是元组的集合，因此每个 SELECT 语句的结果都是一个元组的集合。有时，需要对多个 SELECT 语句的结果进行并(UNION)、交(INTERSECT)和差(EXCEPT)集合操作。

1. 并操作

【例 3-104】 查询借阅 ISBN 号为“302-02368-5”或“302-03314-5”图书的读者。

解：

```
SELECT DISTINCT rno
    FROM borrow
    WHERE bisbn = '302 - 02368 - 5'
UNION
SELECT DISTINCT rno
    FROM borrow
    WHERE bisbn = '302 - 03314 - 5';
```

查询结果为：

rno
14001
14004

2. 交操作

【例 3-105】 查询借阅 ISBN 号为“302-02368-5”和“302-03314-5”图书的读者。

解：

```
SELECT rno
    FROM borrow
    WHERE bisbn = '302 - 02368 - 5'
INTERSECT
SELECT rno
    FROM borrow
    WHERE bisbn = '302 - 03314 - 5';
```

查询结果为：

rno
14001

3. 差操作

【例 3-106】 查询没借过 ISBN 号为“302-03314-5”图书的读者。

解：

```
SELECT rno
    FROM reader
EXCEPT
SELECT rno
    FROM borrow
    WHERE bisbn = '302 - 03314 - 5';
```

查询结果为：

rno
14002
14003
14005

温馨提示：

参与集合操作的各个查询结果的列数必须相同，对应项的数据类型也必须相同。

3.4 数据更新

建立数据库以后，就可以管理数据库中的数据，包括插入数据、修改数据和删除数据。

3.4.1 插入数据

一个基本表只有插入数据以后，才能供用户使用。在 SQL 中，插入数据使用 INSERT

语句。使用该语句可以插入新数据，也可以插入已有的数据。

1. 插入新数据

使用 INSERT 语句，可以将新数据插入到指定的基本表中，并且每次只能向基本表中插入一个元组。其一般格式为：

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

其中，“属性列名 1 [,属性列名 2 , …]”指明在新插入元组的哪些属性列上赋予新值。如果不指定任何属性列，则表示在元组的所有属性列上插入新值；如果仅指定元组的部分属性列，则仅在这些属性列上插入新值，没有指定的属性列则插入空值。

“常量 1 [,常量 2 , …]”指定新插入元组的属性列 1 的值为常量 1，属性列 2 的值为常量 2，……。

【例 3-107】 在 reader 表中，增加一个读者。读者信息如表 3-8 所示。

表 3-8　读者信息

读者编号	姓　名	性　别	年　龄	专　业
14006	孙国庆	男	17	信息

解：

```
INSERT INTO reader(rno, rname, rgender, rage, rspecialty)
    VALUES('14006', '孙国庆', '男', 17, 'IF');
```

或

```
INSERT INTO reader
    VALUES('14006', '孙国庆', '男', 17, 'IF');
```

【例 3-108】 图书馆新进了一本书，书的信息如表 3-9 所示。

表 3-9　图书信息

读者 ISBN 号	图书名称	作　者	出版社	价　格	图书类别
302-02480-4	计算机英语	刘兆毓	清华大学出版社	26	理

解：

```
INSERT INTO book
    VALUES('302 - 02480 - 4', '计算机英语', '刘兆毓', '清华大学出版社', 26, '理');
```

【例 3-109】 读者孙国庆在 2003 年 12 月 20 日借阅了图书“积分变换”，请用 SQL 语句完成此操作。

解：

```
INSERT INTO borrow
    VALUES('14006', '04 - 012955 - 7', '2013/12/20',  NULL, 0);
```

2. 复制现有数据

如果希望将已有数据插入到指定的基本表中，可以在 INSERT 语句中使用子查询。其一般格式为：

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

该语句通过子查询将查询出的数据插入到指定的基本表中，通过这种方式一次可以插入多个元组。

【例 3-110】 汇总每个读者借书总数，并将结果保存在数据库中。

为了保存借书总数，先建立一个新表：

```
CREATE TABLE borrow_count
(
    rno CHAR(5) PRIMARY KEY,                /* 读者编号 */
    amount INT                              /* 借书总数 */
);
```

然后向该表中插入数据：

```
INSERT INTO borrow_count(rno, amount)
    SELECT rno, COUNT( * )
        FROM borrow
        GROUP BY rno;
```

3.4.2 修改数据

通过使用 UPDATE 语句，可以修改基本表中的数据。该操作也称为更新操作。其一般格式为：

```
UPDATE 表名
    SET 属性列名 1 = 表达式 1 [,属性列名 2 = 表达式 2, …]
    [WHERE 条件]
```

其中，“属性列名 1＝表达式 1 [,属性列名 2＝表达式 2，…]”表示用表达式 1 的值取代属性列 1 的值，用表达式 2 的值取代属性列 2 的值，……。

WHERE 子句指定需要修改基本表中的哪些元组值。它与 SELECT 语句中的 WHERE 子句用法相同。

1. 修改全部数据

如果在 UPDATE 语句中不使用 WHERE 子句，则该语句修改基本表中的全部数据。

【例 3-111】 将图书价格增加 10%。

解：

```
UPDATE book
    SET bprice = bprice * 1.1;
```

2. 修改指定数据

如果在 UPDATE 语句中使用 WHERE 子句，则该语句修改基本表中满足条件的数据。

【例 3-112】 将数学专业的读者转入信息专业。

解：

```
UPDATE reader
    SET rspecialty = 'IF'
    WHERE rspecialty = 'MA';
```

3. 带有子查询的修改

可以在 UPDATE 语句中嵌套子查询，用于构造修改的条件。

【例 3-113】 将计算机专业的读者的罚款金额置为 0。

解：

```
UPDATE borrow
    SET fine = 0
    WHERE 'CP' =
        (SELECT rspecialty
            FROM reader
            WHERE reader.rno = borrow.rno);
```

3.4.3　删除数据

通过使用 DELETE 语句，可以删除基本表中的数据。其一般格式为：

```
DELETE FROM 表名
    [WHERE 条件]
```

1. 删除全部数据

如果在 DELETE 语句中不使用 WHERE 子句，则该语句删除基本表中的全部数据，相当于清空基本表。

【例 3-114】 删除所有借阅记录。

解：

```
DELETE FROM borrow;
```

2. 删除指定数据

如果在 DELETE 语句中使用 WHERE 子句，则该语句删除基本表中的满足条件的数据。

【例 3-115】 删除 14005 读者的借阅记录。

解：

```
DELETE FROM borrow
    WHERE rno = '14005';
```

3. 带子查询的删除

可以在 DELETE 语句中嵌套子查询，用于构造删除的条件。

【例 3-116】 删除计算机专业所有读者的借阅记录。

解：

```
DELETE FROM borrow
    WHERE 'CP' =
        (SELECT rspecialty
            FROM reader
            WHERE reader.rno = borrow.rno);
```

3.5 索引

平常在对数据库的操作中，使用最多的是查询。一般情况下，查询数据时需要扫描整个基本表来搜索数据。当基本表中的数据较多时，搜索数据需要很长时间。而使用索引以后，由于索引中的属性列比较少，并且索引项是有序排列的，因此可以加快查找速度。如果在基本表中某些属性列上建立了索引，查询数据时先在索引中搜索，再根据索引定位到基本表中的数据行，就可以大大加快数据查询的速度。图 3-3 显示了一个索引与其基本表之间的关系。

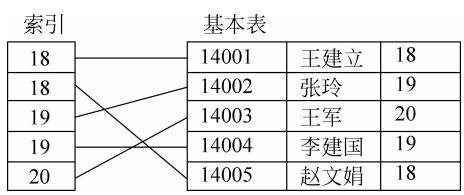


图 3-3 索引与基本表之间的关系

常见的索引包括：

- 唯一索引——唯一索引的每个索引值只对应唯一的一个元组；
- 聚簇索引——聚簇索引的索引项的顺序与基本表中元组的物理顺序完全一致。

由于聚簇索引的索引值的排列顺序与基本表中元组的排列顺序完全一样，因此每个基本表只能有一个聚簇索引，而唯一索引可以有多个。

3.5.1 创建索引

通过使用 CREATE INDEX 语句，可以创建索引，包括唯一索引、聚簇索引和非唯一索引。其一般格式为：

```
CREATE [UNIQUE | CLUSTER] INDEX 索引名
    ON 表名(属性列名 1[次序 1] [,属性列名 2 [次序 2], …]);
```

其中,“表名”是要建立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,各属性列名之间用“,”分隔。每个属性列名之后还可以使用“次序”指定索引值的排列次序。排列次序可以是 ASC(升序)或 DESC(降序),其中 ASC 为默认值。

UNIQUE 表示创建唯一索引。

CLUSTER 表示创建聚簇索引。

【例 3-117】 为读者表的读者编号建一个聚簇索引。

解:

```
CREATE CLUSTER INDEX reader_rno
    ON reader(rno);
```

创建聚簇索引后,更新该索引列上的数据时,往往导致基本表中元组的物理顺序发生变化,代价较大,因此对于经常更新的列不宜创建聚簇索引。

【例 3-118】 为图书表的 ISBN 号建一个唯一索引。

解:

```
CREATE UNIQUE INDEX book_bisbn
    ON book(bisbn);
```

【例 3-119】 为借阅表的读者编号和图书 ISBN 建一个索引,且读者编号为升序,图书 ISBN 为降序。

解:

```
CREATE INDEX borrow_rno
    ON borrow(rno ASC, bisbn DESC);
```

3.5.2　删除索引

通过使用 DROP INDEX 语句,可以删除索引。其一般格式为:

```
DROP INDEX 索引名;
```

【例 3-120】 删除读者表的 reader_rno 索引。

解:

```
DROP INDEX reader_rno;
```

3.6　视图

视图是从若干个基本表和其他视图导出的表。在创建视图时,只把视图的定义存储在数据库中,而不存储视图对应的数据,这些数据仍然存放在原来的基本表中。在使用视图时,才根据视图定义从基本表中导出数据。因此,视图被称为“虚表”。

使用视图能够简化用户的操作、使用户能以多种角度看待同一数据、能够对机密数据提供安全保护，以及能够为数据库重构提供一定的逻辑独立性。

3.6.1 创建视图

通过使用 CREATE VIEW 语句，可以创建视图，其一般格式为：

```
CREATE VIEW 视图名 [(属性列名 1[, 属性列名 2, …])]
    AS 子查询
    [WITH CHECK OPTION];
```

其中，“属性列名 1[，属性列名 2，…]”指定组成视图的属性列。如果省略了视图的各个属性列名，则隐含该视图的属性列由子查询的 SELECT 子句指定。但在以下三种情况下不能省略视图的列名：

(1) SELECT 子句的目标列表达式中包含聚集函数或者表达式；

(2) 多表连接时选出几个同名列作为视图的字段；

(3) 需要在视图中为某个列指定更合适的名字。

子查询可以是任意 SELECT 语句，但不能包含 ORDER BY 子句和 DISTINCT 短语。

“WITH CHECK OPTION”表示对视图进行 INSERT、UPDATE 或 DELETE 操作时，要保证插入、更新或删除的行满足在视图定义时子查询中的条件表达式。

【例 3-121】 建立一个视图，使其包含计算机专业的读者信息。

解：

```
CREATE VIEW reader_cp
    AS
        SELECT rno, rname, rgender, rage
            FROM reader
            WHERE rspecialty = 'CP'
    WITH CHECK OPTION;
```

在本例中，省略了视图 reader_cp 的列名，隐含该视图由 SELECT 子句的 4 个列组成。本例中的视图是行列子集视图。所谓行列子集视图，是指视图是从单个基本表导出的，并且只是去掉了基本表的某些行和某些列，但保留了主码。

【例 3-122】 建立一个视图，使其包含已借书但还未还书的读者信息，这些信息包括读者姓名、图书的 ISBN 号和借书日期。

解：

```
CREATE VIEW borrow_reader
    AS
        SELECT rname, bisbn, startdate
            FROM reader, borrow
            WHERE reader.rno = borrow.rno AND enddate IS NULL;
```

【例 3-123】 建立一个视图，使其包含读者出生年份。

解：

```
CREATE VIEW reader_rage(rno, rname, rbirthday)
```

```
    AS
        SELECT rno, rname, 2014 - rage
            FROM reader
```

在本例中，子查询使用了表达式列，因此必须指定视图的列名。

【例 3-124】 建立一个视图，使其包含每个借书读者的编号和所借图书数量。

解：

```
CREATE VIEW borrow_count(rno, amount)
    AS
        SELECT rno, COUNT( * )
            FROM borrow
            GROUP BY rno;
```

在本例中，子查询使用了聚集函数，因此必须指定视图的列名。

【例 3-125】 建立一个视图，使其包含计算机专业男读者的信息。

解：

```
CREATE VIEW reader_man
    AS
        SELECT rno, rname, rage
            FROM reader_cp
            WHERE rgender = '男';
```

在本例中，视图 reader_man 是从视图 reader_cp 导出的。

【例 3-126】 建立一个视图，使其包含每个借书读者的姓名和所借图书数量。

解：

```
CREATE VIEW borrow_name
    AS
        SELECT rname, amount
            FROM borrow_count, reader
            WHERE borrow_count.rno = reader.rno;
```

3.6.2　删除视图

通过使用 DROP VIEW 语句，可以删除视图，其一般格式为：

```
DROP VIEW 视图名
[CASCADE]
```

在视图删除后，视图的定义被从数据库中删除。如果该视图上还导出了其他视图，则使用 CASCADE 级联删除语句，把该视图和它导出的所有视图一并删除。也可以通过删除基本表的方式撤销视图。当基本表被删除后，由该基本表导出的所有视图没有被删除，但已经无法使用了。

【例 3-127】 删除视图 borrow_name。

解：

```
DROP VIEW borrow_name;
```

【例 3-128】 删除视图 reader_cp。

解：

```
DROP VIEW reader_cp CASCADE;
```

3.6.3 查询视图

定义好视图以后，就可以像对基本表一样对视图进行查询了。

对视图的查询与对基本表的查询过程不同。系统对视图查询时，首先进行有效性检查。检查查询中涉及的基本表、视图等是否存在。如果存在，则从数据库中取出视图定义，把定义中的子查询和用户的查询结合起来，转换成等价的对基本表的查询，然后再执行修正了的查询。这一转换过程称为视图消解。

【例 3-129】 查询计算机专业年龄小于 19 岁的读者信息。

解：

```
SELECT *
    FROM reader_cp
    WHERE rage < 19;
```

本例通过视图消解转换后的查询语句为：

```
SELECT *
    FROM reader
    WHERE rspecialty = 'CP' AND rage < 19;
```

【例 3-130】 查询所借图书数量小于 3 本的读者编号。

解：

```
SELECT rno
    FROM borrow_count
    WHERE amount < 3;
```

本例通过视图消解转换后的查询语句为：

```
SELECT rno
    FROM borrow
    WHERE COUNT( * )< 3
    GROUP BY rno;
```

由于 WHERE 子句中不能出现聚集函数，因此执行修正后的查询将会出现语法错误。正确的查询语句为：

```
SELECT rno
    FROM borrow
    GROUP BY rno HAVING COUNT( * )< 3;
```

目前大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转换。但对非行列子集视图的查询不能保证转换正确，因此这类查询应该直接对基本表进行。

【例 3-131】 查询已借书但还未还书的读者姓名和图书名称。

解：

```
SELECT rname, bname
    FROM borrow_reader, book
    WHERE borrow_reader.bisbn = book.bisbn;
```

本例通过视图消解转换后的查询语句为：

```
SELECT rname, bname
    FROM reader, borrow, book
    WHERE reader.rno = borrow.rno
        AND enddate IS NULL AND borrow.bisbn = book.bisbn;
```

3.6.4　更新视图

更新视图就是对视图执行插入、删除和修改操作。

由于视图不实际存储数据，因此对视图的更新，最终要转换为对基本表的更新。这种转换也是通过视图消解的方式完成的。

为了防止用户更新不属于视图范围内的数据，可以在定义视图时加上 WITH CHECK OPTION 子句。加上该子句后，用户更新视图时，系统会检查视图定义中的条件，若不满足该条件，则拒绝执行该操作。

【例 3-132】 向 reader_cp 视图插入一条新记录，其读者编号为 14007，姓名为郑丽丽，性别为女，年龄为 19 岁。

解：

```
INSERT INTO reader_cp
    VALUES('14007', '郑丽丽', '女', 19);
```

本例通过视图消解转换后的更新语句为：

```
INSERT INTO reader
    VALUES(, '郑丽丽', '女', 19, 'CP');
```

【例 3-133】 将计算机专业中读者编号为 14004 的读者年龄修改为 20 岁。

解：

```
UPDATE reader_cp
    SET rage = 20
    WHERE rno = '14004';
```

本例通过视图消解转换后的更新语句为：

```
UPDATE reader
    SET rage = 20
    WHERE rno = '14004' AND rspecialty = 'CP';
```

【例 3-134】 删除 reader_cp 视图中读者编号为 14007 的记录。

解：

```
DELETE FROM reader_cp
```

```
    WHERE rno = '14007';
```

本例通过视图消解转换后的更新语句为：

```
DELETE FROM reader_cp
    WHERE rno = '14007' AND rspecialty = 'CP';
```

【例 3-135】 将 borrow_count 视图中读者编号为 14003 的读者所借图书数量修改为 2。

解：

```
UPDATE borrow_count
    SET amount = 2
    WHERE rno = '14003';
```

本例通过视图消解转换后，无法转换成对基本表的更新操作，因此该视图是不可更新的。

一个视图想要成为可更新视图，必须满足以下两个条件：

- 视图的每一行必须对应基本表的唯一一行；
- 视图的每一列必须对应基本表的唯一一列。

通常情况下，行列子集视图是可更新视图，它满足以上两个条件。

针对什么样的视图可以更新，不同的系统有不同的规定，因此在对视图进行更新之前，应该仔细研究系统所附带的手册。

3.7 习题

1. 选择题

(1) SQL 语言集数据查询、数据操纵、数据定义和数据控制功能于一体，语句 CREATE TABLE 实现_____功能。

A. 数据查询　　B. 数据操纵　　C. 数据定义　　D. 数据控制

(2) 下列有关 SQL 语言的描述中，不正确的是_____。

A. SQL 语言是高度非过程化的语言

B. SQL 语言是面向集合的语言

C. SQL 既可以交互使用，又可以嵌入到高级语言中使用

D. 使用 SQL 可以定义索引，但不能定义视图

(3) 在 SQL 中，能够直接进行查询的是_____。

A. 基本表　　B. 视图　　C. 基本表和视图　　D. 基本表和索引

(4) 在 SQL 中，使用_____可以实现实体完整性。

A. PRIMARY KEY　　B. FOREIGN KEY

C. NOT NULL　　D. UNIQUE

(5) 在 SQL 中，属于 DML 的是_____。

A. CREATE　　B. ALTER　　C. INSERT　　D. DROP

(6) 在 SELECT 语句中，SELECT 子句与关系代数_____等价。

A. σ　　B. $\prod$　　C. $\bowtie$　　D. $\div$

(7) 已知学生数据表中，属性列 age 为 SMALLINT 类型，则能够表示“16＜age＜25”的是_____。

A. age＞16 AND age＜25　　B. age IN(16,25)

C. age BETWEEN 16 AND 25　　D. age BETWEEN 17 AND 24

(8) 已知客户 Customer(CNO,CName,City)、产品 Product(PNO,PName)和订单 Order(CNO,PNO,Quantity)三张数据表，则要查找订购海尔洗衣机产品的客户姓名，所涉及的基本表是_____。

A. Customer, Product　　B. Customer, Order

C. Product, Order　　D. Customer, Product, Order

(9) 针对第 8 题中的 Customer、Product 和 Order 三张数据表，希望查询客户号为 1001 客户订购的产品总量，应该使用_____聚集函数。

A. SUM　　B. MAX　　C. AVG　　D. COUNT

(10) 已知 SELECT 语句的 WHERE 子句中包含“%李___”，则_____不包含在查询结果中。

A. 陈李广　　B. 李晓　　C. 张田李　　D. 赵李杰

(11) 在 SQL 中，用于测试列值非空的是_____。

A. NOT IS NULL　　B. IS NOT NULL

C. NOT NULL　　D. NOT EMPTY

(12) 在 SQL 中，与“＜ MAX”等价的是_____。

A. ＞ ANY　　B. ＞ ALL　　C. ＜ ANY　　D. ＜ ALL

(13) 已知学生 S(SNO,SName,CNO)和班级 C(CNO,CName)两张基本表，有 SQL 语句：

```
SELECT Sname
    FROM S WHERE CNO NOT IN
        (SELECT CNO
            FROM C
            WHERE CName = '计算机 032 班')
```

则与其等价的关系代数表达式是_____。

A. $\prod_{SName}(\sigma_{CName='计算机032班'}(S \bowtie C))$

B. $\prod_{SName}(\sigma_{CName='计算机032班'}(S \times C))$

C. $\prod_{SName}(S) - \prod_{SName}(\sigma_{CName='计算机032班'}(S \bowtie C))$

D. $\prod_{SName}(S) - \prod_{SName}(\sigma_{CName='计算机032班'}(S \times C))$

(14) 在定义索引时，使每一个索引值只对应唯一的记录值，则应使用_____。

A. UNIQUE　　B. CLUSTER　　C. DISTINCT　　D. RESTRICT

(15) 以下有关视图的说法中，不正确的是_____。

A. 视图能简化用户操作　　B. 查询视图与查询基本表类似

C. 视图是虚表　　D. 无论什么样的视图都是可更新的

2. 填空题

(1) 在 SQL 中,表有三种,即_____、_____和_____。

(2) 视图是一个虚表。在数据库中,只存放视图的_____,而不存放视图对应的_____。

(3) 在 SQL 中,使用_____实现实体完整性,使用_____实现参照完整性。

(4) 在 CREATE TABLE 中,使用_____可以限定列值非空,使用_____可以检查列值是否满足一个布尔表达式。

(5) 在 ALTER TABLE 中,使用_____子句可以添加新的完整性约束条件。

(6) 如果想查询年龄为 18 岁、20 岁或者 21 岁的学生,应该使用_____谓词。

(7) 如果想查询年龄为 18～21 岁的学生,应该使用_____谓词。

(8) 要求将查询结果有序输出,应该在 SELECT 语句中使用_____子句。

(9) 要想统计某个班级中的学生人数,应该在 SELECT 语句中使用_____函数。

(10) 已知雇员 EMP(雇员号 ENO,雇员名 EName,年龄 EAge),则与关系代数表达式 $\sigma_{EAge>35}(EMP)$ 相对应的 SQL 语句为_____。

(11) 向第 10 题的雇员数据表插入一个元组("1031","张三",30)的 SQL 语句为_____。

(12) 有一定义视图的 SQL 语句:CREATE VIEW EMPOrder AS SELECT * FROM EMP ORDER BY EAge,请判断该语句是否正确_____。

(13) 如果想判断元组值大于集合中的每一个元组值,应该使用_____谓词,也可以在子查询中使用_____函数。

(14) SQL 语言有两种使用方式,分别是_____和_____。

3. 应用题

(1) 设有一个学生选课数据库,包含学生关系 *S*、课程关系 *C* 和选修关系 SC,如图 3-4 所示(注:在 *S* 关系中,CS 表示计算机系,IS 表示信息系)。

S

学号 S#	姓名 sname	性别 gender	年龄 age	所属系 dept
1001	王强	男	19	CS
1002	张敏	女	18	IS
1003	王军	男	20	CS
1004	刘冬梅	女	19	IS

C

课程号 C#	课程名 cname	先修课 pno	学分 credit
301	数据结构	302	4
401	数据库	301	4
201	离散数学		6
302	C 语言	201	3

SC

学号 S#	课程号 C#	成绩 grade
1001	201	88
1001	301	85
1001	302	89
1001	401	92
1002	201	46
1002	301	78
1003	201	92

图 3-4　(1)题图

试用 SQL 语句建立这 3 张基本表。

(2) 针对第(1)题建立的 3 张基本表,试用 SQL 语句完成以下查询:

① 查询计算机系所有学生的信息;

② 查询计算机系年龄小于 20 岁的学生信息;

③ 查询信息系男学生的学号和姓名;

④ 查询选修 301 课程或 401 课程的学生学号;

⑤ 查询选修 301 课程和 401 课程的学生学号;

⑥ 查询选修 C 语言课程的学生学号;

⑦ 查询选修 C 语言课程的学生学号和姓名;

⑧ 查询至少选修一门其直接先行课为 301 号课程的学生姓名;

⑨ 查询没有选修 302 号课程的学生学号;

⑩ 查询选修了全部课程的学生学号和姓名。

(3) 针对第 1 题建立的三张基本表,试用 SQL 语句完成以下任务:

① 查询全体学生的姓名及其出生年份;

② 查询选修了课程的学生学号;

③ 查询考试成绩有不及格的学生的学号;

④ 查询年龄在 17～19 岁(包含 17 岁和 19 岁)之间的学生姓名和年龄;

⑤ 查询既不是计算机系,也不是信息系的学生学号和姓名;

⑥ 查询所有姓王的学生学号和姓名;

⑦ 查询课程名称以“数据”开头,且为 4 个汉字的课程名称和学分;

⑧ 查询所有有成绩的学生学号和成绩;

⑨ 查询选修了 301 号课程的学生学号和成绩,要求查询结果按分数降序排列;

⑩ 查询每个学生选课门数、最高成绩、最低成绩和平均成绩;

⑪ 查询选修 2 门以上课程的学生学号;

⑫ 查询选修 301 号课程且成绩在 80 分以上的学生姓名和成绩;

⑬ 查询每个学生其成绩超过他选修课程平均成绩的课程号;

⑭ 查询年龄大于所有女同学年龄的男同学的姓名和年龄;

⑮ 查询张敏不学课程的课程名;

⑯ 查询选修了全部课程的学生姓名。

(4) 针对第(1)题建立的 3 张基本表,试用 SQL 语句完成以下任务:

① 向学生数据表中添加一条记录(1005,“张军”,“男”,17,“CS”);

② 将学号为 1004 的学生年龄修改为 17 岁;

③ 将所有成绩均提高 10%;

④ 删除刘同学的学生记录;

⑤ 删除王军同学的所有选课记录;

⑥ 将每一门课程成绩均在 80～95 分之间的学生的学号和姓名插入到基本表 Score(S#, SName)中;

⑦ 将离散数学课不及格的成绩修改为及格;

⑧ 为学生基本表添加 1 列,名称为 Phone,类型为 CHAR(15);

⑨ 建立计算机系学生的视图；

⑩ 建立计算机系选修了离散数学的学生的视图。

(5) 已知两个基本表 $R(X, Y, Z)$ 和 $S(X, P)$，以及关系代数表达式：

① $R \times S$

② $\prod_{X,P}(\sigma_{P>2}(R \bowtie S))$

③ $\prod_{X,Y}(R) \bowtie \prod_{X,P}(S)$

④ $\prod_{X,Y}(R) - \prod_{X,Y}(\sigma_{P<5}(R \bowtie S))$

试用 SQL 语句表示这些关系代数表达式。