第3章 结构化查询语言SQL

支持关系数据模型的数据库管理系统称为关系数据库管理系统(Relational DataBase Management System,RDBMS),简称关系数据库。第2章介绍了关系数据模型的概念与实例,从本章开始介绍关系数据库的管理与操作。

3.1 SQL概述

SQL(Structured Query Language,读作sequel)即结构化查询语言,是操作关系数据库的标准语言。SQL是一个通用的、功能极强的关系数据库语言,虽然称为"查询",但功能并不限于查询。SQL查询功能的数学基础是关系代数,参见2.2节和2.3节,除查询功能外,SQL中还包括修改数据库的语句、定义数据库模式的语句和对数据库运行进行控制的语句,因此SQL既是数据操作语言,又是数据定义语言,又是数据控制语言。但数据查询在数据库应用中是如此重要,因此有关查询的内容将在本章占有较大比例。

3.1.1 SQL的标准和特点

SQL在1974年由Boyce和Chamberlin提出,并在IBM公司研制的关系数据库管理系统System R上首先实现。由于SQL简单易学,功能丰富,深受数据库相关技术人员的欢迎,因此被各数据库厂商所采用,经各厂商的修改、扩充和完善,SQL得到业界普遍认可,并最终成为国际标准。

教学资源\数据库原理与实现-参考资料.doc:SQL之父——Don Chamberlin。

目前存在三个主要的SQL标准。

(1) ANSI SQL:1986年美国国家标准局(ANSI)的数据库委员会X3H2批准了SQL作为关系数据库语言的美国标准。1987年国际标准化组织(ISO)也通过了这一标准。

(2) SQL-92:也称为SQL2,是1992年采纳的对ANSI SQL进行修改的标准。

(3) SQL-99:以前也称为SQL3,是对SQL-92的扩充,引入了对象关系特征和许多其他的新功能。

还有一些对SQL-99的扩展,统称为SQL:2003。

另外,各大数据库厂商也都根据自己产品的特点提供不同版本的SQL。这些版本的SQL都包括最初的ANSI标准功能,还在很大程度上支持SQL-92。事实上,目前还没有一个数据库产品完全支持SQL-92,而是支持SQL-92的"子集的超集",即大部分的产品在SQL-92主要功能的基础之上各自做了修改和扩展,有的还部分地支持SQL-99和SQL:2003。本书对SQL语言的讨论主要以SQL-92为基础。

SQL 的数学基础是关系代数但不限于关系代数，其特点包括：

- 综合统一。SQL 语言集数据定义语言 DDL、数据操作语言 DML 和数据控制语言 DCL 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工作。
- 高度非过程化。每条 SQL 语句都完成独立的功能。
- 统一的语法结构。所有 SQL 语句都以一个动词开始，直接说明要“干什么”。
- 语言简洁。SQL 语言的核心动词只有 9 个，包括用于查询的 SELECT，用于数据定义的 CREATE、ALTER 和 DROP，用于数据操作的 INSERT、UPDATE 和 DELETE 以及用于数据控制的 GRANT 和 REVOKE。

学习数据库必须掌握 SQL 语言。但教材不同于参考手册，本书只涉及 SQL 语言中最常用的部分，不仅符合 SQL-92 标准，也得到绝大多数商业 DBMS 的支持。

本章的示例 SQL 语句都假设在本书实例“人员数据库”上执行，有关人员数据库的说明请参考附录 A。

并非所有关系数据库管理系统都支持 SQL。在非正式场合，或者为了强调关系数据库管理系统对 SQL 的支持，也将支持 SQL 的关系数据库管理系统称为 SQL 数据库。本书有时为了将讨论限制在支持 SQL 的关系型数据库范围内，也使用此术语。

3.1.2　SQL 的术语

在 SQL-92 标准中使用了一些与传统关系模型不同的术语，这些术语在数据库领域使用的频度甚至超过对应关系模型的术语，因此必须加以介绍。

在 SQL 数据库中与关系模型中关系(Relation)相对应的术语是表(Table)。SQL-92 的标准文本对表有如下描述。

表是行(Row)的集合。行是值(Value)的非空序列。行与表具有相同的基数，其值的数量与表的列(Column)的数量相等。每行中的第 i 个值与表的第 i 个列相对应。行是对表执行插入和删除操作时最小的数据单位[5]。

可见，在 SQL 数据库中与元组相对应的术语是行，与属性相对应的术语是列，与分量相对应的术语是值。

在 SQL 数据库中关系可以有多种存在形式，最常见的有三种：

(1) 持久存储的表(Table)，也称为基表(Base Table)、基本表、数据库表或库表等。数据库中的持久数据存储在基表中，用户可以对基表中的行进行查询和更新。

(2) 逻辑表，称为视图(View)，是通过运算定义的关系。这种关系并不实际存储数据，它只是在需要的时候被完整或部分地生成。视图将在第 4 章讨论。

(3) 临时表，是在执行数据查询和更新等操作时生成的临时关系，用于存储操作的中间数据，操作结束后即被删除。读者将在第 14 章看到临时表是如何生成、使用和删除的。

上述最常用的是基表，在不会混淆的情况下，就简单地称基表为表。

除上述三种表之外，有些关系数据库系统中可能还有外部表(数据存储在外部数据文件中)、索引表(按照结构化主键排序存储数据)等形式。

3.1.3 RDBMS 的体系结构

DBMS 体系结构的概念如此重要，因此在讨论完关系数据模型的基本概念并对 SQL 有了初步了解之后，很有必要重新从关系模型的角度来考查关系数据库管理系统(RDBMS)的体系结构。

关系模型定义了 RDBMS 体系结构的逻辑模式层，SQL 定义了 RDBMS 的接口，因此 RDBMS 对逻辑模式层和外模式层有了更加明确的定义，1.3 节介绍的 DBMS 体系结构在 RDBMS 中具体化为如图 3.1 所示。

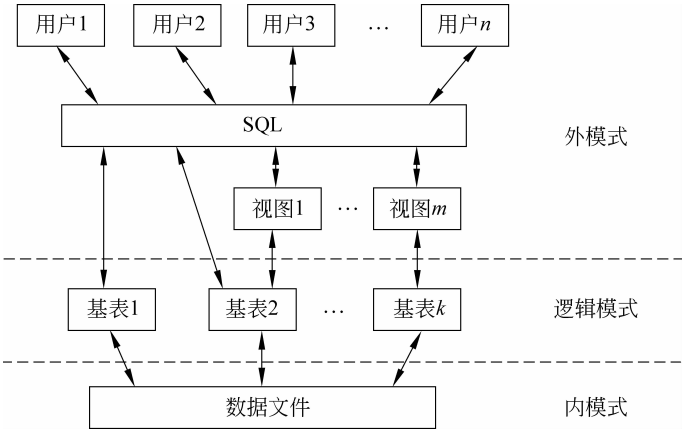

图 3.1 支持 SQL 的 RDBMS 体系结构

支持 SQL 的 RDBMS 体系结构如图 3.1 所示。视图虽然是关系的一种，但被看做外模式的一部分，这是由其性质和作用决定的。

1. 逻辑模式

SQL 数据库的逻辑模式是基表的集合，基表之间通过外键相互关联。对 SQL 数据库的逻辑模式进行设计，主要工作就是设计基表的结构及各类完整性约束。

2. 外模式

所有用户，无论是最终用户还是应用程序，都通过 SQL 来使用数据库。单一的用户可以透过 SQL 看到部分基表，或者通过视图访问基表的部分行、列。外模式还定义了对单用户视图中数据的操作。

3. 内模式

SQL 的标准文本并未对内模式给出建议，这部分留给具体的数据库产品去定义。本书第 3 部分将会对 CXDB 内模式的具体实现方法进行描述。

3.2 数据定义

数据库管理主要是对(基)表和表中的数据进行管理。关系数据库的数据定义也主要体

现在表上面，包括表的创建、修改和删除三种操作。

3.2.1　数据类型

关系模型中的一个重要概念是域，每个属性的取值必须是某个域的值，在 SQL 中域的概念用数据类型来实现。关系数据库中存储的数据包括文字、数值、日期时间等，SQL-92 标准文本对数据类型有如下描述。

数据类型是一些可描述值的集合。这些值在逻辑上用文本形式描述，在物理上依赖具体的实现。这里值是基本的，即在逻辑上不能再分。SQL 用如下关键字作为数据类型名称：CHARACTER、CHARACTER VARYING、BIT、BIT VARYING、NUMERIC、DECIMAL、INTEGER、SMALLINT、FLOAT、REAL、DOUBLE PRECISION、DATE、TIME、TIMESTAMP 和 INTERVAL[5]。

下面介绍 SQL-92 标准中常用的数据类型。

1. 固定长度字符串

类型 CHARACTER，在有些数据库产品中用 CHAR(n)表示，是最多可有 n 个字符的固定长度字符串。如果用户给出的字符串长度不足 n，系统会在后面补以空格。使用固定长度字符串优点是可以提高查询速度，缺点是补充的空格浪费存储空间。为了避免空间的过度浪费，系统一般限制 n 的值不能过大，如不能超过 255。

2. 可变长度字符串

类型 CHARACTER VARYING，在有些数据库产品中用 VARCHAR(n)表示，是最多可有 n 个字符的字符串。如果用户给出的字符串长度不足 n，后面也不补充空格。使用可变长度字符串的缺点是处理速度较慢，优点是节省空间，且 n 的允许值可以很大，如 4000。

随着计算机硬件处理速度的加快和数据库设计水平的提高，可变长度字符串在处理速度上的劣势已不再明显，很多数据库系统对 CHAR 和 VARCHAR 型数据已经采用相同的技术进行处理，长度都可变，二者不再区分。

另外，如果需要存储 Unicode 字符集的字符型数据，可以使用 NCHAR 和 NVARCHAR 类型。

3. 整数

整数类型包括 INTEGER(INT)和 SMALLINT，其中 SMALLINT 的取值范围小一些，具体的取值范围取决于具体产品的实现。

4. 浮点数

浮点数类型包括 FLOAT、REAL 和 DOUBLE PRECISION，其中 DOUBLE PRECISION 的精度更高一些，同样取决于具体实现。

5. 数值

数值类型包括 DECIMAL 和 NUMERIC。数值类型数据的取值既可以是整数，也可以

是浮点数，但浮点数的小数位需要预先确定。与前两种数据类型相比，数值类型具有更大的取值范围。最重要的，使用数值类型可以得到运算的“十进制精度”。

例 3.1 请看下面的 C 程序段：

```
float sum=0;
int i;
for (i=1;i<=10000;i++)
    sum+=(float)0.01;
printf("%f",sum);
```

在作者的计算机上，用 Visual C++ 6.0 编译执行上述程序，结果为 100.002953，也就是说 10000 个 1 分钱加起来不等于 100 元。可见浮点数的精度不足以支持银行、商场等业务信息的处理，对这些信息的处理必须提供十进制精度。

6. 日期和时间

日期和时间类型包括 DATE、TIME 和 TIMESTAMP，其中 TIMESTAMP 型数据中既有日期又有时间。在逻辑上日期和时间数据表现为按一定格式显示的字符串，如'2012-10-01'或'12:00:00'，在数据库中的物理存储由具体实现决定。一般数据库产品都支持在字符串与日期和时间型数据之间转换。

不同的关系数据库产品，其产品定位各不相同，实现方法差别甚大，都会在 SQL-92 的基础之上进行取舍。每个关系数据库产品所支持的数据类型都相似但又各不相同，以 CXDB 为例，所支持的数据类型包括 VARCHAR、NUMBER、DATE、CLOB 和 BLOB 等，见表 3.1。

表 3.1 CXDB 支持的数据类型

名　　称	含　　义
VARCHAR	字符型。用于描述变长的字符型数据，长度≤4000 字节。系统还支持使用别名 VARCHAR2 和 CHAR，内部的处理没有差别
NUMBER	数值型。类似于 SQL-92 的 DECIMAL 和 NUMERIC，用来存储整型或者浮点型数值，长度≤38 位
DATE	日期型。类似于 SQL-92 的 TIMESTAMP，用来存储日期数据。CXDB 的日期型其实是“日期时间型”，同时包含日期和时间，时间的精度到秒
CLOB	字符型大对象。用来存储多达 1GB 的字符数据
BLOB	二进制大对象。用来存储多达 1GB 的非结构化二进制数据

大多数 DBMS 都支持 CLOB 和 BLOB 类型数据，很有必要，但 CLOB 和 BLOB 并不是 SQL-92 的标准类型，而是 SQL-99 的新增类型。

3.2.2 定义基表

了解了 RDBMS 支持的数据类型，就可以用这些类型定义表的各列。

SQL 语言使用 CREATE TABLE 语句创建表，主要工作是定义各列的属性和表的完整性约束。CREATE TABLE 语句的一般形式为：

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

注意：SQL 语句都以动词开头，表明所要执行的操作，并以分号“;”结束。

对 SQL 语句语法的严格描述可以采用 BNF 范式（参见 3.6.1 节）或语法图等形式[6]，本章采用上述简单（但不规范）的描述形式。为使描述更清晰，将可进一步说明的语法成分用尖括号括起来，并在必要时给出说明。

1. 简单的表定义

创建表的主要工作是定义各列的名称和数据类型。

例 3.2　人员数据库的学位表（参见附录 A）的创建语句为：

```
CREATE TABLE degrees (
degree_id NUMBER(4),
degree_name VARCHAR(50)
);
```

所创建的表的表名为 degrees，共有两列。第 1 列 degree_id 为学位号，数值型，最多 4 位，未指定小数位，所以取值只能是整数。第 2 列 degree_name 为学位名称，字符型，最多使用 50 个字符（字节）。

SQL 语言不区分标识符字母的大小写，但本书的示例 SQL 语句尽量遵循一个原则，就是关键字用大写，用户自定义的标识符用小写，方便读者理解。

所有文献在讲解 SQL 语句时都会假设有一个界面，用户输入 SQL 语句，然后计算机显示执行结果，本书也不例外，假设 SQL 语句在 CXDBManager 的“执行 SQL 语句”窗口中执行。

在实际的数据库应用程序开发中，通常会选择一种开发语言，如 C、Java 等，对 SQL 来说称为宿主语言，SQL 语句由宿主语言调用并将结果返回给宿主语言处理。

例 3.3　人员数据库的人员表的创建语句为：

```
CREATE TABLE people (
p_id NUMBER(4),
p_name VARCHAR(50),
sex VARCHAR(4),
age NUMBER(4),
mgr_id NUMBER(4),
degree_id NUMBER(4),
dept_id NUMBER(5),
hire_date DATE
);
```

2. 定义域完整性约束

除关系数据模型中规定的三类完整性约束之外，关系数据库系统大多支持域完整性约束。域完整性是指表中的列必须满足某种特定的约束条件，如取值必须在某个范围内等。当表创建完成后，如果用户要操作表中的数据，DBMS 会自动检查这些约束条件是否满足，不满足的操作会被 DBMS 禁止。列的 NOT NULL、DEFAULT 和 CHECK 等定义都属于域完整性约束。

(1) 非空约束(NOT NULL)

SQL-92 标准文本对空值(NULL)有如下说明：空值是一个特殊的值或标记，用于表明“没有值”。空值是一个依赖于具体实现的特殊值，它与任意数据类型中的所有非空值相区别；空值仅仅是一个值，但它属于任何 SQL 数据类型。没有任何文本可以表示空值，尽管有时使用 NULL 关键字表明该处需要一个空值[5]。

空值表示某行某列“没有值”或“值未知(Value Unknown)”，并不是指该行该列的值是长度为 0 的字符串或数值 0。

如果表的某一列被指定具有 NULL 属性，那么就允许插入数据时省略该列的值。反之如果表的某一列被指定具有 NOT NULL 属性，且该列没有指定默认值，那么在插入数据行时必须为该列指定值。

在 CXDB 中，列的默认属性是 NULL。如果要限定某列不允许使用空值，可用如下的方式进行列定义：

```
p_name VARCHAR(50) NOT NULL
```

在数据库系统中一旦涉及空值情况就会变得比较复杂，读者会在本书的后续内容中看到多处针对空值的特别解释。但定义空值又非常必要，因为在数据库系统中有时必须指明某些值是未知的。

(2) 默认值(DEFAULT)

向表中插入数据时并非总是给它的每列都指定值，一般情况下系统会将未指定值的列置为 NULL；但如果该列定义了默认值，则系统将该列自动置为默认值。

一般情况下用户为列指定常量作为默认值，如指定“年龄”列的默认值为 0，可用如下的列定义：

```
age NUMBER(4) DEFAULT 0
```

也可以用表达式的结果值作为默认值，如指定“聘用日期”列的默认值为当前系统日期，可用如下的列定义：

```
hiredate DATE DEFAULT SYSDATE
```

其中 SYSDATE 是 CXDB 系统函数，用于取系统当前的日期时间。

(3) 检查(CHECK)

检查约束指定一个条件表达式。如果对某列施加了 CHECK 约束，当为该列赋一个新值时，系统首先检查新值是否符合指定的条件。如果符合就可以将新值放入该列，如果不符

合就拒绝接受该值。如限定“年龄”列的值必须大于 18,可用如下的列定义:

```
age NUMBER(4) CHECK(age>18)
```

3. 定义实体完整性约束

通过在表上定义主键实现关系的实体完整性约束。在表上定义主键有两种方式:

(1) 定义列时声明其为主键;

(2) 当列定义完成之后,在表定义的后部专门声明一组列为主键。

如果主键由多个列组成,则只能使用第 2 种方式。

例 3.4　例 3.2 定义的表 degrees,在定义时要指定 degree_id 列为主键,采用第 1 种方式的 SQL 语句为:

```
CREATE TABLE degrees (
degree_id NUMBER(4) PRIMARY KEY,
degree_name VARCHAR(50)
);
```

采用第 2 种方式的等价 SQL 语句为:

```
CREATE TABLE degrees (
degree_id NUMBER(4),
degree_name VARCHAR(50),
PRIMARY KEY (degree_id)
);
```

无论采用哪种方式,都可以使用 CONSTRAINT 关键字为主键指定约束名:

```
CREATE TABLE degrees (
degree_id NUMBER(4),
degree_name VARCHAR(50),
CONSTRAINT degrees_pk PRIMARY KEY (degree_id)
);
```

如果未指定主键名称,系统会自动生成一个名称。

一旦定义了主键,系统会自动保证所有行在主键列上不能有完全相同的值,且属于主键的各列都不能为空。

例 3.5　创建人员表时同样可以为其指定主键:

```
CREATE TABLE people (
p_id NUMBER(4) PRIMARY KEY,
p_name VARCHAR(50) NOT NULL,
⋮
);
```

很多情况下可以在表中找到具有实际意义的列作为该表的主键。例如,如果数据库中人员的姓名都不相同,在上述人员表中可以指定 p_name 列为主键。但实用的数据库设计中更常用的方法是使用虚拟主键,如例 3.5 人员表中就是创建了一个列 p_id 专门用来做主

键，例 3.4 的学位表也采用了同样的方法。这样做可以得到多方面的好处：

- 单纯从数据角度考虑，这样做可以更安全地对主键值做出需要的假定，如在人员表中就不用再担心有重名的人员，不需要为这种情况准备特殊的处理；
- 在应用开发中主键往往被用做查询的依据，使用简单的主键列（取值为整数，不会是多列）可以为程序开发带来方便；
- 从数据库实现的角度，使用简单的主键列可以节省索引存储空间，提高查询速度。

与主键类似的约束是唯一性约束，除用 UNIQUE 关键字代替 PRIMARY KEY 之外，在定义上没有差别，但在约束规则上有差别。唯一约束条件同样要求在约束列上不能有完全相同的值，但可以有空值，且可以有多个空值，也就是说数据库系统认为空值之间互不相等。

4. 定义参照完整性约束

通过在表上定义外键实现关系的参照完整性约束，外键的概念见 2.4.2 节。外键的定义只能在列定义完成之后，在表定义的后部进行。

例 3.6 人员表的 degree_id 列参照了学位表的 degree_id 列，可以在人员表中将该列定义为外键，表的创建语句为：

```
CREATE TABLE people (
p_id NUMBER(4) PRIMARY KEY,
⋮
degree_id NUMBER(4),
CONSTRAINT people_fk1 FOREIGN KEY (degree_id)
    REFERENCES degrees(degree_id)
);
```

上面例句中使用了 CONSTRAINT 关键字指定外键名称，如果不指定，系统会自动生成一个名称。

3.2.3 修改与删除基表

SQL 语言使用 ALTER TABLE 语句修改表结构。不同 DBMS 产品对修改表结构的支持差别很大，例如 CXDB 支持对表名、列名的修改，支持对约束条件的增加和删除。ALTER TABLE 语句的一般形式为：

```
ALTER TABLE <表名>
[RENAME TO <新表名>]
[RENAME COLUMN <旧列名>TO <新列名>]
[RENAME CONSTRAINTS <旧约束名>TO <新约束名>]
[ADD <表级完整性约束条件>]
[DROP PRIMARY KEY]
[DROP CONSTRAINTS <约束名>]
;
```

例 3.7 要将学位表的表名改为 degree，其 SQL 语句为：

```
ALTER TABLE degrees
```

```
RENAME TO degree;
```

例 3.8　在上述人员表中，mgr_id(上级人员编号)列参照的是本表的 p_id 列，可以用下面的 SQL 语句为人员表增加外键：

```
ALTER TABLE people
ADD CONSTRAINT people_fk3 FOREIGN KEY (mgr_id)
    REFERENCES people(p_id);
```

多数关系数据库产品允许本表内字段的相互参照，并允许在创建表时定义相应外键。CXDB 要求定义外键时被参照的表必须已经存在，因此只能用上述增加定义的方法定义此类外键。

有的 DBMS 产品还支持修改字段类型，修改字段大小，增加/删除字段等功能，但不同 DBMS 产品的语法差别甚大，本书不一一介绍。

SQL 语言使用 DROP TABLE 语句删除已经存在的表。DROP TABLE 语句的一般形式为：

```
DROP TABLE <表名>;
```

例 3.9　要删除学位表，其 SQL 语句为：

```
DROP TABLE degrees;
```

3.3　基本数据查询

对于绝大多数数据库系统而言，查询都是其最常用的数据管理操作。SQL 的查询语句只有一个——SELECT，但 SELECT 无疑是所有 SQL 语句中功能最强、最灵活也最复杂的一个。

3.3.1　SELECT 语句的语法

SELECT 语句的一般格式为：

```
SELECT <选择表达式>[,<选择表达式>...]
FROM <数据源>[,<数据源>...]
[WHERE <条件表达式>]
[GROUP BY <分组表达式>[,<分组表达式>...] [HAVING <条件表达式>]]
[ORDER BY <排序条件>[,<排序条件>...]];
```

一个 SELECT 语句至少包含 SELECT 和 FROM 两个子句。SELECT 子句中包含一个或多个选择表达式，指明要查询的列。FROM 子句中指定一个或多个数据源(关系)，SELECT 语句将从这些数据源中取数据。所谓数据源可以是基表，也可以是视图。从基表或视图中查询数据的语法完全相同，因为本章的示例都从基表查询数据，所以在本章后文中使用“表”而不再使用数据源或关系，请读者正确理解。

SELECT 语句的其他子句待用到时再介绍。本节对 SELECT 语句的严格语法不做详

细介绍，而是通过一些 SELECT 语句示例进行说明。

注意：以下的 SELECT 语句仍然假设在本书实例“人员数据库”上执行。

3.3.2 单表查询

使用 SELECT 语句既可从单一表查询，也可以对多个表进行连接查询。先讨论比较简单的情况——单表查询。

1. 查询基表中的全部数据

例 3.10 查询学位表中的所有行和所有列，即全部数据。

```
SELECT * FROM degrees;
```

执行结果为：

degree_id	degree_name
1	博士
2	硕士
3	学士

其中选择表达式“*”代表所有列。本例未使用 WHERE 子句，所以查询结果中包含源表的所有行。

其他例句：

```
SELECT * FROM departments;
SELECT * FROM people;
```

2. 查询表中的若干列

只查询表中的部分列，其数学基础是关系代数的投影(Projection)运算，关系代数相关的内容见 2.2 节和 2.3 节。

例 3.11 查询所有的学位名称。

```
SELECT degree_name FROM degrees;
```

执行结果为：

degree_name
博士
硕士
学士

其中选择表达式 degree_name 指明了要查询的是“学位名称”列。

其他例句：

```
SELECT dept_name,dept_phone FROM departments;
SELECT p_name,sex,age FROM people;
```

3. 查询经过计算的值

查询结果不一定只能是表的原始列，可以是任意合法表达式，包括常量、函数调用等，原始列只是表达式的一个特例但最常用。

例 3.12　请看下面 SELECT 语句，虽无实际意义，但语句合法。

```
SELECT degree_id+10,15,-abs(degree_id) FROM degrees;
```

执行结果为：

Column0	Column1	Column2
11	15	−1
12	15	−2
13	15	−3

其中 abs()为求绝对值函数。

例 3.13　再看下面 SELECT 语句。

```
SELECT p_name||sex||age FROM people;
```

其中运算符 ‖ 表示字段串的连接操作。

默认情况下，对于原始的表列，查询结果的列标题就是源表的列名；但对于经过计算的列，列标题由系统自动生成，生成方法不同的 DBMS 产品各不相同。如果用户想指定与默认情况不同的列名，可在选择表达式后使用 AS 关键字指定别名。

例 3.14　为例 3.12 的各输出列指定别名。

```
SELECT degree_id+10 AS col1,15 AS col2,-abs(degree_id) AS col3
FROM degrees;
```

执行结果为：

COL1	COL2	COL3
11	15	−1
12	15	−2
13	15	−3

4. 查询符合条件的行

仅查询表中符合条件的行，其数学基础是关系代数的选择(Selection)运算。

SELECT 语句中，可选的 WHERE 子句包含条件表达式，指明从表中选择哪些行。

条件表达式可以是一个单独的谓词，也可以是对谓词进行并且(AND)、或者(OR)或求

反(NOT)运算后的结果。这里的谓词可以是关系谓词、LIKE 谓词和 IS NULL 等谓词。

例 3.15 查询姓名为“张一”的人员的信息。

```
SELECT * FROM people
WHERE p_name='张一';
```

执行结果为:

p_id	p_name	sex	age	mgr_id	degree_id	dept_id	hire_date
1001	张一	男	35		2	1	2001-01-01

例 3.16 查询年龄小于 38 且性别为女的人员的信息。

```
SELECT * FROM people
WHERE age<38 AND sex='女';
```

执行结果为:

p_id	p_name	sex	age	mgr_id	degree_id	dept_id	hire_date
1003	张三	女	33	1001		1	2003-04-05
1005	李二	女	31	1004	1	2	
1006	李三	女	36	1004		2	2005-06-07

例 3.17 查询年龄在 35～40 之间的人员的信息。

```
SELECT * FROM people
WHERE age BETWEEN 35 AND 40;
```

执行结果为:

p_id	p_name	sex	age	mgr_id	degree_id	dept_id	hire_date
1001	张一	男	35		2	1	2001-01-01
1006	李三	女	36	1004		2	2005-06-07
1009	王三	女	38	1007			2003-02-01

例 3.18 查询所有姓王的人员的信息。

```
SELECT * FROM people
WHERE p_name LIKE '王%';
```

执行结果为:

p_id	p_name	sex	age	mgr_id	degree_id	dept_id	hire_date
1007	王一	男	46	1001		3	2006-10-22
1008	王二	男	33	1007	2	3	
1009	王三	女	38	1007			2003-02-01

例 3.19　查询所有拥有学位(学位号字段不为空)的人员的信息。

```
SELECT * FROM people
WHERE degree_id IS NOT NULL;
```

执行结果为：

p_id	p_name	sex	age	mgr_id	degree_id	dept_id	hire_date
1001	张一	男	35		2	1	2001-01-01
1004	李一	男	44	1001	1	2	2004-05-06
1005	李二	女	31	1004	1	2	
1008	王二	男	33	1007	2	3	

关于如何使用 LIKE 关键字进行模式匹配和如何对空值进行判断，3.6 节有更为详细的说明。

5. 对查询结果进行排序

SELECT 语句中可选的 ORDER BY 子句包含一个或多个排序条件，系统根据这些排序条件对查询结果进行排序。这是 SQL 对关系代数的扩展，关系代数无论是基于集合还是基于包，元组都无先后次序，这显然不能满足实际的数据管理需求，因此 SQL 在此对关系代数进行了扩展。＜排序条件＞的格式为：

```
<排序表达式>[ASC|DESC]
```

根据排序表达式的值进行排序，如果指明 ASC，则按从小到大排序；如果指明 DESC，则按从大到小排序；如果不指明 ASC 或 DESC，默认从小到大。

例 3.20　查询所有姓王的人员的信息，并按年龄从小到大排序。

```
SELECT * FROM people
WHERE p_name LIKE '王%'
ORDER BY age;
```

执行结果为：

p_id	p_name	sex	age	mgr_id	degree_id	dept_id	hire_date
1008	王二	男	33	1007	2	3	
1009	王三	女	38	1007			2003-02-01
1007	王一	男	46	1001		3	2006-10-22

3.3.3　分组统计查询

可以使用聚合函数对数据库中的数据进行统计查询，这也是 SQL 对关系代数的扩展。

SQL-92 规定了 5 个聚合函数，见表 3.2。

表 3.2 常用的聚合函数

函数名	函 数 说 明	函数名	函 数 说 明
AVG	AVG(value)，求平均值	MIN	MIN(value)，求更小值
COUNT	COUNT(value)，计数	SUM	SUM(value)，求和
MAX	MAX(value)，求最大值		

例 3.21 查询人员数量、年龄最大和最小的人员的年龄。

```
SELECT COUNT(*),MAX(age),MIN(age) FROM people;
```

执行结果为：

Column0	Column1	Column2
9	46	31

例 3.21 对表中所有的数据进行统计。实际的数据库应用中对数据的统计往往与分组结合使用。SELECT 语句中可选的 GROUP BY 子句包含一个或多个分组表达式。所谓分组，就是将表达式值相同的行分为一组，在查询结果中表现为一行。

例 3.22 查询男女人员的数量和不同性别中年龄最大的人员的年龄。

```
SELECT sex,COUNT(*),MAX(age)
FROM people
GROUP BY sex;
```

执行结果为：

sex	Column1	Column2
男	4	46
女	5	42

如果使用了 GROUP BY 子句，还可以使用 HAVING 子句对分组后的结果进行筛选。

例 3.23 请看下面 SELECT 语句。

```
SELECT sex,COUNT(*),MAX(age)
FROM people
GROUP BY sex
HAVING COUNT(*)>4;
```

执行结果为：

sex	Column1	Column2
女	5	42

3.4　连接查询

在设计数据库时，为了满足数据库的规范化(见第 6 章)要求，数据分布在不同的表中。但是在使用数据库时，经常需要把分布在不同表中的数据作为一个统一的结果集提供给用户。

前面讨论的数据查询都是从单一基表中查询数据。在实际应用中经常需要同时从两个或两个以上的表中查询数据，这就需要使用多表连接查询技术。在执行连接查询时，需要指定多表之间用某个或某些列进行连接的条件，也就是说查询结果既要满足连接条件，又要满足选择(WHERE)条件。连接查询是完善数据库信息查询的重要手段，是数据库规范化理论之所以合理的关键原因。

ANSI SQL 规定了多表连接查询的方法，即使用 JOIN 关键字，称为 **ANSI 连接**。但由于历史原因绝大多数 DBMS 还支持一种老式的连接方式，不使用 JOIN 关键字，称为**旧式连接**。本书主要介绍 ANSI 连接方式，兼顾旧式连接，但对旧式连接方式不做过多说明。

SELECT 语句是所有 SQL 语句中功能最强、最灵活也最复杂的一个，而多表连接查询又是 SELECT 语句中最灵活也最复杂的部分。要想真正掌握多表连接查询的方法需要大量实践，本书仅通过一些实例进行简单介绍，目的是为读者今后的数据库实践打下基础。

在本节的内容中，交叉连接的数学基础是关系代数的笛卡儿积运算，内连接的数学基础是关系代数的自然连接(Natural Join)运算和 θ 连接(θ Join)运算，而外连接则是在关系代数基本运算基础之上的扩展。

本节先从两个表之间的连接查询开始讨论。

3.4.1　交叉连接

连接查询的数学基础是关系代数的笛卡儿积运算，也就是说所有连接运算的结果都是笛卡儿积的子集。

所谓交叉连接，就是不对连接条件进行限制，所得结果集就是参与连接的两个表的笛卡儿积本身。

例 3.24　请看下面 SELECT 语句。

```
SELECT * FROM people CROSS JOIN degrees;
```

执行结果包括 9 列(人员表的 7 列加学位表的 2 列)、27 行(人员表的 9 行乘以学位表的 3 行)。结果集就是两个表的笛卡儿积。

等价的旧式连接例句：

```
SELECT * FROM people,degrees;
```

可以对多表中的指定列进行查询，相当于在笛卡儿积上再做投影运算。

例 3.25　请看下面 SELECT 语句。

```
SELECT p_name,degrees.degree_id,degree_name
FROM people CROSS JOIN degrees;
```

执行结果仍为 27 行，但只有指定的 3 列，其中 degree_id 列来自于表 degrees。

需要注意的是，在进行多表查询时，使用表名限定列名可以使 SELECT 语句更明了，甚至可能提高数据查询的效率。对于同时存在于多个表中的同名列，有的 DBMS 规定必须限定表名，有的 DBMS 无此规定（系统自动确定表，但不一定符合用户的期望）。无论如何，在进行多表连接查询时适当地限定各列的表名是好习惯。

3.4.2 内连接

内连接把两个表连接成一个表，结果表中仅包含原表中满足连接条件的行。内连接主要有两种形式，相等连接和非相等连接。相等连接指定两个表间进行比较的列，结果集中只包含比较列值相等的行；非相等连接则使用其他类型的连接条件。

1. 相等连接

例 3.26 查询人员的姓名和学位。单从人员表查询只能查到学位号，不能查到学位名称。如果要同时查询人员的姓名和学位（名称），就需要使用连接查询。

```
SELECT p.p_name,d.degree_name FROM people p INNER JOIN degrees d
ON p.degree_id=d.degree_id;
```

执行结果为：

p_p_name	d_degree_name
张一	硕士
李一	博士
李二	博士
王二	硕士

在例 3.26 中为参与连接的表指定了别名，people 的别名为 p，degrees 的别名为 d。由于需要用表名来限定列名，因此为较长的表名指定较短的别名可以简化 SELECT 语句的书写。

其中关键字 INNER 表示内连接。连接条件是两个表的 degree_id 列值相等。INNER 是默认连接方式，因此 INNER 关键字可以省略。

等价例句（查询结果与上面例句完全相同）：

```
SELECT p.p_name,d.degree_name FROM people p JOIN degrees d
ON p.degree_id=d.degree_id;
```

如果两个表中用于判断相等的列名相同，且两个表中没有其他列名相同，则可以使用自然（NATURAL）连接。

等价例句：

```
SELECT p.p_name,d.degree_name
FROM people p NATURAL INNER JOIN degrees d;
```

无论两个表中是否有其他列名相同，都可以用 USING 关键字显式地指定用于相等连接的列。

等价例句：

```
SELECT p.p_name,d.degree_name FROM people p JOIN degrees d
USING (degree_id);
```

等价的旧式连接例句：

```
SELECT p.p_name,d.degree_name FROM people p,degrees d
WHERE p.degree_id=d.degree_id;
```

读 SELECT 语句的技巧：先读 FROM 子句，再读 SELECT 子句。

2. 非相等连接

连接条件不一定非得是两个表的某些列相等(虽然这种情况最为常见)，任何合法的表达式都可以作为连接条件。

例 3.27　下面语句虽然没有实际意义，但是合法的 SELECT 语句。

```
SELECT p.p_name,d.degree_name FROM people p INNER JOIN degrees d
ON p.degree_id<>d.degree_id;
--<>意为“不等于”
```

SQL 语言一般支持用/* */括起来的多行注释和以--开头的单行注释，本书也会使用。

例 3.28　很多参考文献中都会用到一个著名的“部门-雇员”数据库示例，在该示例数据库中，如果要查询雇员的姓名、工资及其工资级别，可使用下面语句：

```
SELECT e.ename,e.sal,s.grade FROM emp e JOIN salgrade s
ON e.sal BETWEEN s.losal AND s.hisal;
```

3.4.3　外连接

读者也许已经注意到，例 3.26 使用内连接查询人员的姓名和学位(名称)时，结果集只有 4 行，那些没有学位的人员并没有列出。

在进行内连接查询时，两个表中都可能存在不符合连接条件的行，内连接查询将会放弃这些行。但如果这些行对查询结果是重要的则需要使用外连接，外连接分为左外连接、右外连接和完全连接。

1. 左外连接

例 3.29　要查询“所有”人员的学位情况。

```
SELECT p.p_name,d.degree_name
FROM people p LEFT OUTER JOIN degrees d
ON p.degree_id=d.degree_id;
```

执行结果为：

p_p_name	d_degree_name
张一	硕士
张二	
张三	
李一	博士
李二	博士
李三	
王一	
王二	硕士
王三	

查询结果中包含了所有人员，包括那些没有学位的人员，这些人员对应的学位名称为空。这样的查询结果可能更符合某些应用需求。

由于左连接肯定是外连接，所以句中的 OUTER 关键字可以省略。等价例句：

```
SELECT p.p_name,d.degree_name
FROM people p NATURAL LEFT JOIN degrees d;
```

其他等价例句：

```
SELECT p.p_name,d.degree_name FROM people p LEFT JOIN degrees d
USING (degree_id);
```

等价的旧式连接例句：

```
SELECT p.p_name,d.degree_name FROM people p,degrees d
WHERE p.degree_id=d.degree_id(+);
```

注意使用旧式连接方式时(＋)的位置。

2. 右外连接

例 3.30 要查询所有的学位和具有相应学位的人员。

```
SELECT p.p_name,d.degree_name FROM people p RIGHT JOIN degrees d
ON p.degree_id=d.degree_id;
```

执行结果为：

p_p_name	d_degree_name
张一	硕士
李一	博士
李二	博士
王二	硕士
	学士

查询结果中包含了所有的学位。如果某个学位有多人拥有，则该学位在结果集中有多行；如果某个学位没人拥有，则该学位在结果集中对应的人员姓名为空。

等价的旧式连接例句：

```
SELECT p.p_name,d.degree_name FROM people p,degrees d
WHERE p.degree_id(+)=d.degree_id;
```

3. 完全连接

将内连接查询的结果集同时进行左外连接和右外连接的扩展，就得到完全连接的结果。

例 3.31　请看下面 SELECT 语句。

```
SELECT p.p_name,d.degree_name FROM people p FULL JOIN degrees d
ON p.degree_id=d.degree_id;
```

执行结果为：

p_p_name	d_degree_name
张一	硕士
张二	
张三	
李一	博士
李二	博士
李三	
王一	
王二	硕士
王三	
	学士

旧式连接方式中没有完全连接。

3.4.4　多表连接查询

对多个(超过两个)表进行连接查询的实质与两个表的连接查询相同。

例 3.32　要查询人员的姓名、学位(名称)和单位(名称)，需要同时从 3 个表中查询数据。

```
SELECT p.p_name,d.degree_name,t.dept_name
FROM people p JOIN degrees d ON p.degree_id=d.degree_id
JOIN departments t ON p.dept_id=t.dept_id;
```

执行结果为：