第 5 章

SQL 数据定义与操纵

学习目标：

- 了解 SQL
- 掌握 MYSQL 的基本数据类型
- 了解 SQL 数据定义
- 掌握创建数据表
- 掌握数据操纵(增删改)的方法

5.1 SQL 概 述

5.1.1 SQL 的发展历史

SQL(structured query language,结构化查询语言)是关系数据库的标准语言,是专门用来与数据库通信的语言,其利用一些简单的句子构成基本的语法来存取数据库中的内容,便于用户从数据库中获得和操作所需数据。

1972 年,IBM 公司开始研制实验型关系数据库管理系统 System R,并且为其配置了 SQUARE(specifying queries as relational expression)查询语言。1974 年,Boyce 和 Chamberlin 在此基础上对其进行改进,将 SQUARE 语言改为 SEQUEL(structured english query language),后来 SEQUEL 简称为 SQL,即“结构化查询语言”,并首先在 IBM 公司研制的关系数据库管理系统 System R 上实现。1986 年 10 月,经美国国家标准局(ANSI)的数据库委员会 X3H2 批准,将 SQL 作为关系数据库语言的美国标准,同年公布了标准 SQL。1987 年 6 月,国际标准化组织(International Organization for Standardization, ISO)将其采纳为国际标准。这两个标准现在称为“SQL 86”。ANSI 在 1989 年 10 月颁布了增强完整性特征的 SQL 89 标准,1992 年又公布了 SQL 92 标准,1999 年发布了 SQL 99 标准,以后每隔几年会推出一个新版本,目前最近的版本是 SQL 2022 标准。

SQL 是关系数据库的公共语言。用户可将使用 SQL 的应用从一个关系型数据库管理系统转移到另一系统。SQL 主要由数据定义语言、数据操纵语言和数据控制语言组成。

数据定义语言(data definition language,DDL)用来定义数据的结构,是对数据的格式和形态进行定义的语言,主要用于创建、修改和删除数据库对象、表、索引、视图及角色等,常用的语句有 CREATE、ALTER 和 DROP 等。每个数据库要建立时首先要面对一些问题,例如,数据与哪些表有关、表内有什么栏目主键,以及表与表之间互相参照的关系等,这些都需要在设计开始时就预先规划好,所以,DDL 是数据库管理员和数据库拥有者才有权操作的用于生成与改变存储结构的命令语句。

数据操纵语言(data manipulation language,DML)用于读取和操纵数据,数据定义完成后接下来就是对数据的操作。数据的操作主要有插入数据(INSERT)、查询数据(SELECT)、更改数据(UPDATE)和删除数据(DELETE)4 种方式,即数据操纵主要用于数据的更新、插入等操作。

数据控制语言(data control language,DCL)用于安全性控制,如权限管理、定义数据访问权限、进行完整性规则描述及事务控制等,其主要内容包括授予或回收操作数据库的某种特权、控制数据库操纵事务发生的时间及效果、对数据库实行监视三方面。

5.1.2 SQL 的特点

SQL 具有以下 4 个特点。

(1) 功能完备并且一体化。数据库的主要功能就是通过数据库支持的数据语言来实现的。SQL 不但具有数据定义、数据查询、数据操作、数据控制等功能,而且这些功能已被集成到一个语言系统中,只要用 SQL 就可以实现数据库生命周期中的全部活动。

(2) 语言简洁,易学易用。尽管 SQL 的功能很强,但语言十分简洁,SQL 的核心功能通过表 5-1 展示的 9 个语句来实现。

表 5-1 SQL 的功能及实现语句

SQL 的功能	语 句
数据定义	CREATE、DROP、ALTER
数据查询	SELECT
数据操纵	INSERT、UPDATE、DELETE
数据控制	GRANT、REVOKE

(3) 高度非过程化语言。SQL 允许用户在高层的数据结构上工作,而不对单个记录进行操作,可以操作记录集。SQL 语句接受集合作为输入,返回集合作为输出。在 SQL 中,用户只需要在程序中说明“做什么”,无须说明“怎样做”,即无须用户指定对数据存放的方法。

(4) 统一的语法结构。SQL 可用于所有用户的模型,包括系统管理员、数据库管理员、应用程序员及终端用户,这些用户可以通过自含式语言和嵌入式语言两种方式对数据库进行访问,这两种方式使用统一的语法结构。

5.1.3 SQL体系结构

SQL支持关系数据库体系结构，即外模式、模式和内模式。利用SQL可以实现对三级模式的定义、修改和数据的操作功能，在此基础上形成了SQL的体系结构，如图5-1所示。

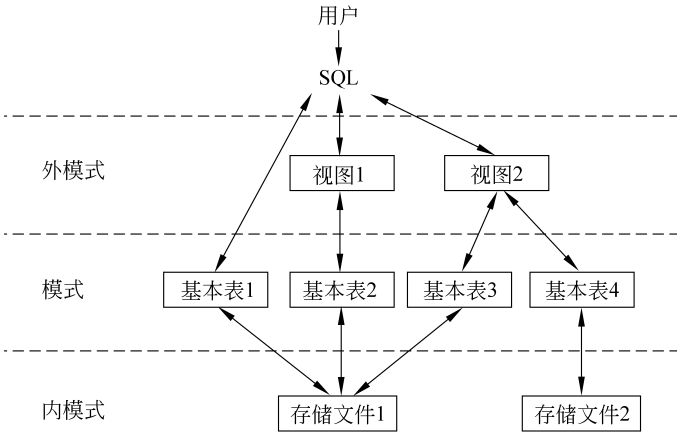

图5-1 SQL的体系结构

SQL的体系结构中包含用户、基本表、视图和存储文件。

(1) 用户。SQL用户可以是应用程序，也可以是终端用户。SQL语句可嵌入在宿主语言的程序中使用，也可以作为独立的用户接口，供交互环境下的终端用户使用。

(2) 基本表。基本表简称基表，它是数据库中实际存在的表，在SQL中一个关系对应于一个基本表。

(3) 视图。SQL用视图概念支持非标准的外模式概念。视图是从一个或几个基表导出的表，虽然它也是关系形式，但它本身不实际存储在数据库中，只存放对视图的定义信息(没有对应的数据)。因此，视图是一个虚表或虚关系，而基表是一种实关系。

(4) 存储文件。每个基表对应一个存储文件，每个存储文件都与外部存储器上一个物理文件对应。一个基表还可以带一个或几个索引，存储文件和索引一起构成了关系数据库的内模式。

因此可以看出，一个基本表可以存放在多个存储文件中，一个存储文件也可以存放多个基本表的数据；一个视图可以来自多个基本表，一个基本表可以构造多个视图；一个用户可以查询多个视图，一个视图也可以被多个用户访问。

5.2 MySQL的基本数据类型

数据类型是定义列中可以存储什么数据以及该数据实际怎样存储的基本规则。在创建表时，表中的每个字段都有数据类型，用来指定数据的存储格式、约束和有效范围。选择合

适的数据类型可以有效地节省存储空间,同时可以提升数据的计算性能。在设计表时,应该特别重视所用的数据类型。使用错误的数据类型可能会严重地影响应用程序的功能和性能。更改包含数据的列不是一件小事(而且这样做可能会导致数据丢失)。MySQL 提供了多种数据类型,常用的主要包括数值类型(包括整数类型和小数类型)、字符串类型、日期时间类型和二进制类型。

1. 数值类型

MySQL 支持多种数值数据类型,每种存储的数值具有不同的取值范围。显然,支持的取值范围越大,所需存储空间越多。此外,有的数值数据类型支持使用十进制小数点,有的则只支持整数。表 5-2 列出了常用的 MySQL 数值数据类型。

表 5-2 常用的 MySQL 数值数据类型

数据类型	说明
BIT	位字段,1～64 位(在 MySQL 5 之前,BIT 在功能上等价于 TINYINT)
BIGINT	整数值,支持 -9223372036854775808 ～ 9223372036854775807(如果是 UNSIGNED,为 0～18446744073709551615)范围内的数值
BOOLEAN(或 BOOL)	布尔标志,值为 0 或 1,主要用于开/关(on/off)标志
DECIMAL(或 DEC)	精度可变的浮点值
DOUBLE	双精度浮点值
FLOAT	单精度浮点值
INT(或 INTEGER)	整数值,支持 -2147483648 ～ 2147483647(如果是 UNSIGNED 为 0～4294967295)范围内的数值
MEDIUMINT	整数值,支持-8388608～8388607(如果是 UNSIGNED,为 0～16777215)范围内的数值
REAL	4 字节的浮点值
SMALLINT	整数值,支持-32768～32767(如果是 UNSIGNED,为 0～65535)范围内的数值
TINYINT	整数值,支持-128～127(如果是 UNSIGNED,为 0～255)范围内的数值

默认情况下,整数类型既可以表示正整数,也可以表示负整数。如果只希望表示正整数,则可以使用关键字 UNSIGNED 来修饰。例如,要将学生表中的学生年龄字段定义为无符号整数,可以使用 SQL 语句 AGE TINYINT UNSIGNED 来实现。

对于整数类型还可以指定其显示宽度,例如 int(8)表示当数值宽度小于 8 位时在数字前面填满宽度。如果在数字位数不够需要用 0 填充时,则可以使用关键字 zerofill。但是在插入的整数位数大于指定的显示宽度时,将按照整数的实际值进行存储。

2. 字符串类型

最常用的数据类型是串数据类型。它们存储串,如名字、地址、电话号码、邮政编码等。

有两种基本的串类型，分别为定长串和变长串，如表 5-3 所示。

表 5-3 字符串类型

数 据 类 型	说 明
CHAR	1～255 个字符的定长串。它的长度必须在创建时指定，否则 MySQL 假定为 CHAR(1)
VARCHAR	长度可变，最多不超过 255 字节。如果在创建时指定为 VARCHAR(n)，则可存储 0 到 n 个字符的变长串(其中 $n \leqslant 255$)
TINYTEXT	最大长度为 255B 的变长文本
TEXT	最大长度为 64KB 的变长文本
MEDIUMTEXT	最大长度为 16KB 的变长文本
LONGTEXT	最大长度为 4GB 的变长文本
ENUM	枚举类型，接受最多 64KB 个串组成的一个预定义集合的某个串
SET	集合类型，接受最多 64B 个串组成的一个预定义集合的零个或多个串

定长串接受长度固定的字符串，其长度是在创建表时指定的。例如，名字列可允许 30 个字符，而身份证号列允许 18 个字符。定长列不允许多于指定的字符数目。它们分配的存储空间与指定的一样多。

变长串存储可变长度的文本。有些变长数据类型具有最大的定长，而有些则是完全变长的。不管是哪种，只有指定的数据得到保存(额外的数据不保存)。

既然变长数据类型这样灵活，为什么还要使用定长数据类型？是因为性能。MySQL 处理定长列远比处理变长列快得多。此外，MySQL 不允许对变长列(或一个列的可变部分)进行索引，这也会极大地影响性能。

3. 日期和时间数据类型

MySQL 使用专门的数据类型来存储日期和时间值，见表 5-4。

表 5-4 日期和时间数据类型

数 据 类 型	说 明
DATE	表示 1000-01-01～9999-12-31 的日期，格式为 YYYY-MM-DD
DATETIME	DATE 和 TIME 的组合
TIMESTAMP	功能和 DATETIME 相同(但范围较小)
TIME	格式为 HH：MM：SS
YEAR	用 2 位数字表示，范围是 70(1970 年)～69(2069 年)；用 4 位数字表示，范围是 1901～2155 年

4. 二进制数据类型

二进制数据类型主要存储二进制数据，如图像、多媒体、字处理文档等，如表 5-5 所示。

表 5-5 二进制数据类型

数 据 类 型	说 明
BLOB	Blob 最大长度为 64KB
MEDIUMBLOB	Blob 最大长度为 16MB
LONGBLOB	Blob 最大长度为 4GB
TINYBLOB	Blob 最大长度为 255B

5.3 SQL 数据定义

5.3.1 基本表的定义

1. 表结构的定义

建立数据库最重要的一步就是定义基本表的结构。SQL 用于创建基本表的语法结构为

```
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>]);
```

说明：

① 表名是所要定义的基本表的名字，表可以由一个或多个属性(列)组成。

② 定义表的各个列时需要指明其数据类型及长度。

③ 完整性约束条件。关系完整性约束包括实体完整性、参照完整性和用户定义完整性。这三种完整性约束条件都可以在表的定义中给出。其中，实体完整性定义表的主关键字，参照完整性定义外关键字，用户定义完整性根据具体应用对关系模式提出要求，主要包括对数据类型、数据格式、取值范围、空值约束等的定义。

完整性约束又可分为列完整性、元组完整性和表级完整性三个级别。在关系模式的定义中，最常定义的是列完整性约束和表级完整性约束。用户定义的完整性规则属于列级完整性约束，而实体完整性和参照完整性都属于表级完整性约束。

由于完整性约束条件也是关系模式定义的一部分，所以下面给出部分完整性约束条件的定义方法。这些完整性约束条件被存入系统的数据字典中，当用户操作表中数据时由 DBMS 自动检查该操作是否违背这些完整性约束条件。

例 5-1 建立一个“学生”表 S，它由学号 Sno、姓名 Sname、性别 Ssex、年龄 Sage、籍贯 Shome 五个属性组成。其中学号不能为空，值是唯一的。

```
CREATE TABLE S
  (Sno  CHAR(5)  NOT NULL  UNIQUE,
   Sname  CHAR(8),
   Ssex   CHAR(2),
   Sage   INT,
   Shome  CHAR(15));
```

上述 SQL 语句执行后，将建立一个新的空“学生”表 S(见图 5-2)。其中，NOT NULL 和 UNIQUE 分别说明学号 Sno 不能取空值和重复的值，该约束等同于主码的约束。

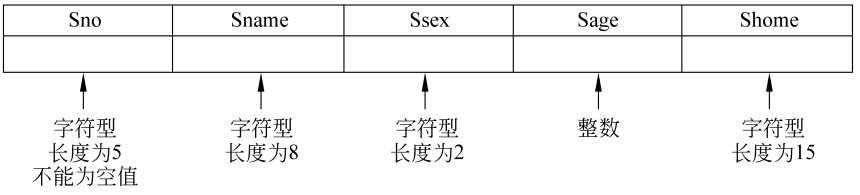

图 5-2 “学生”表 S

2. 主关键字的定义

一个关系可能有多个候选关键字，但在定义基本表时只能定义一个主关键字。一个关系的主关键字由一个或几个属性构成，在 CREATE TABLE 中声明主关键字的方法如下：

(1) 在列出关系模式的属性时，在属性及其类型后加上保留字 PRIMARY KEY，表示该属性是主关键字。

(2) 在列出关系模式的所有属性后，再附加一个声明：

```
PRIMARY KEY(<属性 1>[,<属性 2>,…])
```

说明：如果关键字由多个属性构成，则必须使用方法(2)。

例 5-2 建立一个“学生”表 S，它由学号 Sno、姓名 Sname、性别 Ssex、年龄 Sage、籍贯 Shome 五个属性组成。其中学号是主关键字。

方法 1：

```
CREATE TABLE S
  (Sno  CHAR(5)  PRIMARY KEY,
   Sname  CHAR(8) ,
   Ssex   CHAR(2) ,
   Sage   INT,
   Shome  CHAR(15));
```

方法 2：

```
CREATE TABLE S
  (Sno  CHAR(5) ,
   Sname  CHAR(8) ,
   Ssex   CHAR(2) ,
```

```
  Sage   INT,
  Shome  CHAR(15)
  PRIMARY  KEY(Sno));
```

例 5-3 建立一个“班级”表 C，它由班级号 Cno、班级名 Cname、班级人数 Csize、所在年级 Cgrade、所在院系 Cdept 共 5 个属性组成，其中，课程号 Cno 为主关键字。

```
CREATE TABLE C
  (Cno  CHAR(8)  NOT NULL  UNIQUE,
   Cname  CHAR(15),
   Csize  SMALLINT,
   Cgrade  CHAR(4),
   Cdept  CHAR(15),
   PRIMARY  KEY(Cno));
```

从例 5-3 可以看出，虽然非空(NOT NULL)约束和唯一(UNIQUE)约束结合在一起的作用等同于主键(PRIMARY KEY)约束，但是，二者是可以重复定义的。同时，虽然主键的声明是可选的，但为每个关系指定一个主键会更好些。

例 5-4 建立一个“学生分班”表 SC，它由学号 Sno、班级号 Cno，分班成绩 Grade 组成，其中(Sno，Cno)为主码。

```
CREATE TABLE SC
  (Sno CHAR(5),
   Cno CHAR(4),
   Grade   INT,
   PRIMARY KEY (Sno, Cno));
```

3. 外部关键字的定义

外部关键字的定义是建立参照完整性的约束，它是关系模式的另一种重要约束。根据参照完整性的概念，在 SQL 中，有两种方法用于说明一个外部关键字。

(1) 如果外部关键字只有一个属性，可以在它的属性名和类型后面直接用 REFERENCES 说明它参照了某个表的某些属性(必须是主关键字)。其语法格式为

```
REFERENCES<表名>(<属性>)
```

(2) 在 CREATE TABLE 语句的属性列表后面增加一个或几个外部关键字说明，其格式为

```
FOREIGN KEY(<属性 1>)REFERENCES<表名>(<属性 2>)
```

其中，“属性 1”是外部关键字，“属性 2”是被参照的属性。

例 5-5 为例 5-4 中的“学生分班”表 SC 建立外码，分别参照例 5-2 学生表中的学号和例 5-3 班级表中的班级号。

```
CREATE TABLE SC
  (Sno CHAR(5),
```

```
    Cno CHAR(4),
    Grade   INT,
    PRIMARY KEY (Sno, Cno)
    FOREIGN KEY(Sno)  REFERENCES  S(Sno)
    FOREIGN KEY(Cno)  REFERENCES  C(Cno));
```

该例中定义了两个外关键字：学生分班表中的学号和班级号。根据参照完整性规则，学生分班表中的学号要么取空值，要么取学生表中的学号值。但是，由于学生分班表中的学号又是该关系主关键字中的属性，根据实体完整性约束条件，主属性不能取空值。所以，学生分班表中的外关键字学号只能取学生表中学号的值，不能取空值。另一个外关键字班级号的取值亦然。

4. 默认值的定义

可以在定义属性时增加保留字 DEFAULT 为表中某列的取值定义一个默认值。例如：

```
CREATE TABLE C
  (Cno  CHAR(8)  NOT NULL  UNIQUE,
   Cname  CHAR(15),
   Csize  SMALLINT  DEFAULT  50,
   Cgrade  CHAR(4),
   Cdept  CHAR(15),
   PRIMARY  KEY(Cno));
```

5.3.2 基本表的修改

随着应用环境和应用需求的变化，有时需要修改已经建立好的基本表，如增加列、增加新的完整性约束条件、修改原有的列定义或删除已有的完整性约束条件等，此处仅列出其中的部分语法。

语法格式：

```
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>]
|[CHANGE <列名><新列名><新数据类型>];
```

其中，<表名>是要修改的基本表；ADD 子句用于增加新列和新的完整性约束；DROP 子句用于删除某一列；DROP CONSTRAINT 子句用于删除指定的完整性约束；CHANGE 子句用于修改列名和数据类型。

例 5-6 在学生表 S 中增加“技能 Sskill”属性，类型为 CHAR。

```
ALTER TABLE S ADD COLUMN Sskill CHAR(20);
```

例 5-7 在班级表 C 中修改“所在院系 Cdept”属性的长度为 20 位。

```
ALTER TABLE C CHANGE COLUMN Cdept Cdept CHAR(20);
```

例 5-8 在学生表 S 中删除“技能 Sskill”属性。

```
ALTER TABLE S DROP COLUMN Sskill;
```

注意：

① 可以增加或减少某一列的长度，但是修改后的长度不能小于该列原有数据的长度；

② 对于 NULL 值约束进行修改的问题。该问题产生于将某列的约束从 NULL 改变为 NOT NULL 时，要求指定的字段中不能有 NULL 值。如果包含值为 NULL 的字段，则必须先删掉所发现的任何 NULL 值，然后使用 ALTER TABLE 命令进行修改。

5.3.3 基本表的删除

当不仅要删除表中的数据而且要删除表的结构时，可以使用 DROP TABLE 语句。该语句的格式如下：

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

RESTRICT 表示如果有视图或约束条件涉及要删除的表时，就禁止 DBMS 执行该命令；而 CASCADE 选项则将该表与其涉及的对象一起删除。

例 5-9 假设存在表 Temp，现将其删除，并将与该表有关的其他数据库对象一起删除。对应的 SQL 语句如下：

```
DROP TABLE Temp CASCADE;
```

5.4 数 据 操 纵

SQL 对数据的操纵是指用 INSERT、UPDATE、DELETE 语句来进行插入、更新和删除数据库表中记录行的数据，由 DML 语言实现，它们是数据库的主要功能之一。

5.4.1 插入数据

1. 插入单个元组

插入语句的一般格式为：

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

如果某些属性列在 INSERT INTO 子句中没有出现，则新记录在这些列上将取空值。但必须注意的是，在表定义时说明了 NOT NULL 的属性列不能取空值，否则会出错。

如果 INSERT INTO 子句中没有指明任何列名，则新插入的记录必须在每个属性列上