第 4 章　数据表的基本操作

在数据库中，数据表是数据库中最重要、最基本的操作对象，是数据存储的基本单位。数据表被定义为列的集合，数据在表中是按照行和列的格式来存储的。每一行代表一条唯一的记录，每一列代表记录中的一个域。

本章将详细介绍数据表的基本操作，主要内容包括创建数据表、修改数据库、删除数据表。通过本章的学习，读者能够熟练掌握数据表的基本概念，理解约束的含义并且学会运用，能够在图形界面模式和命令行模式下熟练地完成有关数据表的常用操作。

- 掌握创建数据表的方法
- 掌握查看数据表的属性的方法
- 掌握修改数据表的方法
- 熟悉删除数据表的方法
- 熟练操作综合案例数据表的基本操作

4.1　创建数据表

在创建完数据库之后，接下来的工作是创建数据表。所谓创建数据表，指的是在已经创建好了的数据库中建立新表。创建数据表的过程是规定数据列的属性的过程，同时也是实施数据完整性（包括实体完整性、引用完整性和域完整性等）约束的过程。本节将介绍创建数据表的语法形式、如何添加主键约束、外键约束、非空约束等。

4.1.1　创建数据表

数据表属于数据库，在创建数据表之前，应该先在对象浏览器中选择在哪个数据库中进行，如果没有选择数据库，则不能创建数据表。常见的创建数据表的方法有以下两种：

1. 使用对象浏览器创建数据表

【例 4.1】创建数据表 ppo1。具体操作步骤如下：

步骤 01　在对象管理器中依次展开【mytest】➢【模式】➢【public】节点，右击【表】节点并在弹出的快捷菜单中选择【创建】➢【表】菜单命令，如图 4-1 所示。

步骤 02　弹出【创建 - 表】对话框，默认选择【通常】选项卡，在【名称】文本框中输入“ppo1”，在【所有者】下拉列表中选择数据表的拥有者，如图 4-2 所示。

图 4-1　选择【创建】➢【表】菜单命令

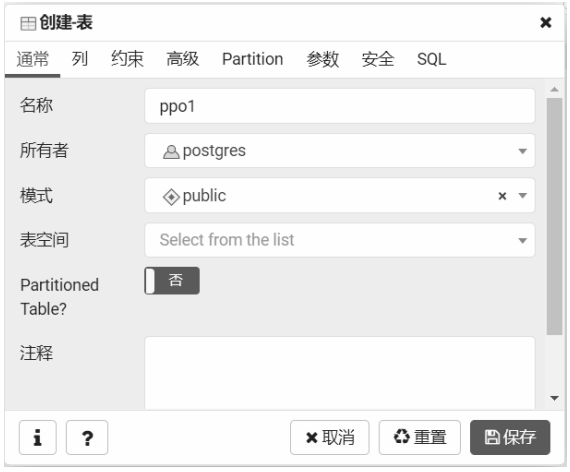

图 4-2　【创建 - 表】对话框

步骤 03　选择【列】选项卡，用户可以在这里添加数据表的字段。单击【添加】按钮，在下方可以设置字段的名称、数据类型、长度、精度等属性，这里在【名称】文本框中输入字段名称为“idnumbl”，在【数据类型】下拉列表中选择【integer】，如图 4-3 所示。

步骤 04　采用同样的操作，可以创建数据表的多个字段，如图 4-4 所示。

图 4-3　添加“idnumbl”字段

图 4-4　添加其他字段

步骤 05　选择【约束】选项卡，用户可以创建各种约束，包括主键、外键、排他约束、唯一和检查等，默认选择【主键】选项，单击【添加】按钮，在【名称】文本框中输入“numb”，

如图 4-5 所示。

步骤 06 单击【编辑】按钮，在【通常】选项下可设置主键的名称和注释，如图 4-6 所示。

图 4-5 【约束】选项卡

图 4-6 【通常】选项

步骤 07 选择【定义】选项，可设置作为主键的字段、主键的表空间和填充因子等参数，这里在【列】下拉列表中选择【idnumbl】字段，如图 4-7 所示。

步骤 08 选择【高级】选项卡，用户可以设置类型、填充因子等数据表的属性，如图 4-8 所示。

图 4-7 【定义】选项

图 4-8 【高级】选项卡

步骤 09 选择【参数】选项卡，即可设置关于整理数据表的相关内容，如图 4-9 所示。

步骤 10 选择【安全】选项卡，单击【添加】按钮，可以设置用户对数据表的权限，如图 4-10 所示。

图 4-9　【参数】选项卡

图 4-10　【安全】选项卡

步骤 11　选择【SQL】选项卡，即可看到上述操作对应的 SQL 语句，如图 4-11 所示。

步骤 12　单击【保存】按钮，即可创建数据表，在对象浏览器中依次展开【表】➢【ppo1】➢【列】节点，即可在右侧的属性窗口中查看新建表中各字段的属性，如图 4-12 所示。

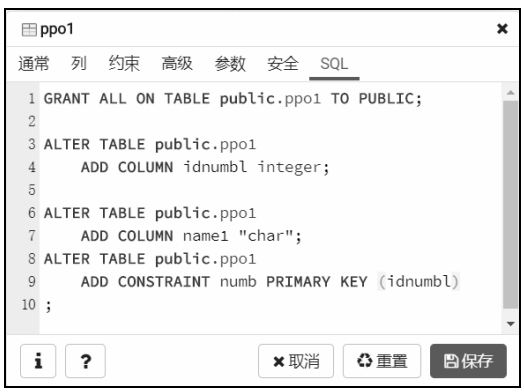

图 4-11　【SQL】选项卡

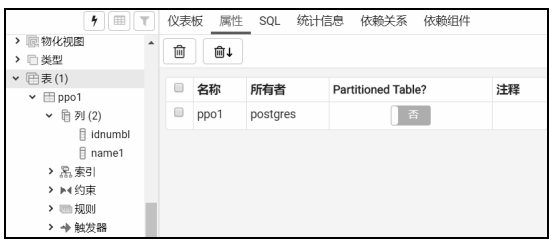

图 4-12　查看字段的属性

2. 使用 SQL 语句创建数据表

创建数据表的语句为 CREATE　TABLE，语法规则如下：

```
CREATE　TABLE <表名>
(
  字段名 1  数据类型  [列级别约束条件] [默认值],
  字段名 2  数据类型  [列级别约束条件] [默认值],
  ……
  [表级别约束条件]
);
```

使用 CREATE TABLE 创建表时，必须指定以下信息：

（1）要创建的表名称，不区分大小写，不能使用 SQL 语言中的关键字，如 DROP、ALTER、INSERT 等。

（2）数据表中每一个列（字段）的名称和数据类型，如果创建多个列，要用逗号隔开。

【例 4.2】创建员工表 tb_emp1，结构如表 4.1 所示。

表 4.1　tb_emp1 表结构

字段名称	数据类型	备注
id	INT	员工编号
name	VARCHAR(25)	员工名称
deptId	INT	所在部门编号
salary	FLOAT	工资

在对象浏览器中选择 mytest 数据库，新建立一个当前连接查询，在查询编辑器中输入以下 SQL 语句，如图 4-13 所示。

```
CREATE TABLE tb_emp1
(
  id          INT,
  name      VARCHAR(25),
  deptId    INT,
  salary    FLOAT
);
```

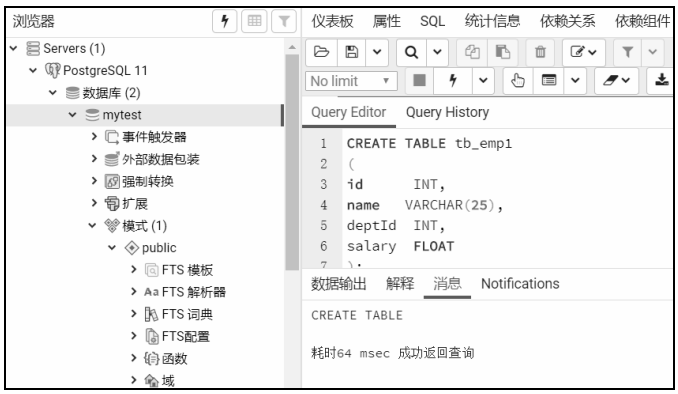

图 4-13　输入 SQL 语句

语句执行后，便创建了一个名称为 tb_emp1 的数据表。在对象浏览器中依次展开【mytest】➢【模式】➢【public】➢【表】➢【tb_emp1】➢【列】节点，即可在右侧的属性窗口中查看新建表的字段，如图 4-14 所示。

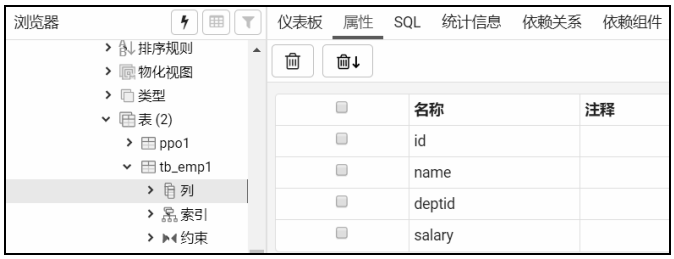

图 4-14　成功创建数据表

4.1.2　使用主键约束

主键又称主码，是表中一列或多列的组合。主键约束（Primary Key Constraint）要求主键列的数据唯一，并且不允许为空。主键能够唯一标识表中的一条记录，可以结合外键来定义不同数据表之间的关系，并且可以加快数据库查询的速度。主键和记录之间的关系如同身份证和人之间的关系，它们之间是一一对应的。主键分为两种类型：单字段主键，多字段联合主键。

1. 单字段主键

主键由一个字段组成，SQL 语句格式分为以下两种情况。

（1）在定义列的同时指定主键，语法规则如下：

```
字段名  数据类型  PRIMARY KEY
```

【例 4.3】定义数据表 tb_emp 2，其主键为 id，SQL 语句如下：

```
CREATE TABLE tb_emp2
(
   id          INT PRIMARY KEY,
   name      VARCHAR(25),
   deptId    INT,
   salary    FLOAT
);
```

在对象浏览器中选择新创建的数据表，在右侧的窗口中选择【依赖组件】选项卡，即可看到主键的类型，如图 4-15 所示。

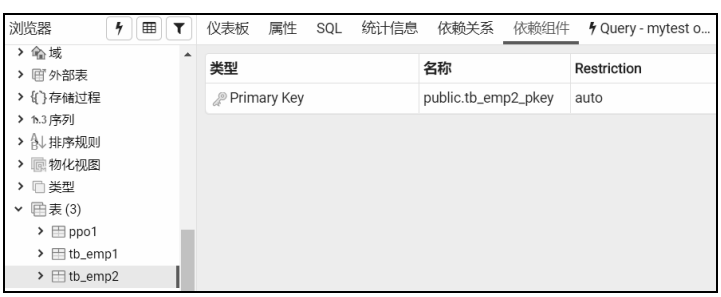

图 4-15　看到主键的类型

（2）在定义完所有列之后指定主键。

```
[CONSTRAINT <约束名>] PRIMARY KEY [字段名]
```

【例 4.4】定义数据表 tb_emp3，其主键为 id，SQL 语句如下：

```
CREATE TABLE tb_emp3
(
id INT,
name VARCHAR(25),
deptId INT,
salary FLOAT,
PRIMARY KEY(id)
);
```

上述两个例子执行后的结果是一样的，都会在 id 字段上设置主键约束。

2. 多字段联合主键

主键由多个字段联合组成，语法规则如下：

```
PRIMARY KEY [字段 1, 字段 2,…, 字段 n]
```

【例 4.5】定义数据表 tb_emp4，假设表中间没有主键 id，为了唯一确定一个员工，可以把 name、deptId 联合起来作为主键，SQL 语句如下：

```
CREATE TABLE tb_emp4
(
name VARCHAR(25),
deptId INT,
salary FLOAT,
PRIMARY KEY(name,deptId)
);
```

语句执行后，便创建了一个名称为 tb_emp4 的数据表，name 字段和 deptId 字段组合在一起成为 tb_emp4 的多字段联合主键。

4.1.3 使用外键约束

外键用来在两个表的数据之间建立链接，可以是一列或者多列。一个表可以有一个或多个外键。外键对应的是参照完整性，可以为空值，若不为空值，则每一个外键值必须等于另一个表中主键的某个值。

外键：表中的一个字段，可以不是本表的主键，但必须对应另外一个表的主键。外键的主要作用是保证数据引用的完整性。定义外键后，不允许删除在另一个表中具有关联关系的行。例如，部门表 tb_dept 的主键是 id，在员工表 tb_emp5 中有一个键 deptId 与这个 id 关联。

主表（父表）：对于两个具有关联关系的表而言，相关联字段中主键所在的那个表就是主表。

从表（子表）：对于两个具有关联关系的表而言，相关联字段中外键所在的那个表就是从表。

使用对象浏览器创建外键的具体操作步骤如下：

步骤 01　展开需要创建外键约束的数据表，右击【约束】节点，并在弹出的快捷菜单中选择【创建】➢【外键】菜单命令，如图 4-16 所示。

步骤 02　弹出【创建-外键】对话框，默认选择【通常】选项卡，在【名称】文本框中输入外键的名称，如图 4-17 所示。

图 4-16　选择【外键】菜单命令

图 4-17　【创建-外键】对话框

步骤 03　选择【列】选项卡，在 3 个参数的下拉列表中分别选择字段和表，单击【添加】按钮 +，即可添加外键的参考字段，如图 4-18 所示。

步骤 04　设置完成后，单击【保存】按钮，然后在对象浏览器中刷新【约束】节点，即可看到新创建的外键约束，如图 4-19 所示。

图 4-18　【列】选项卡

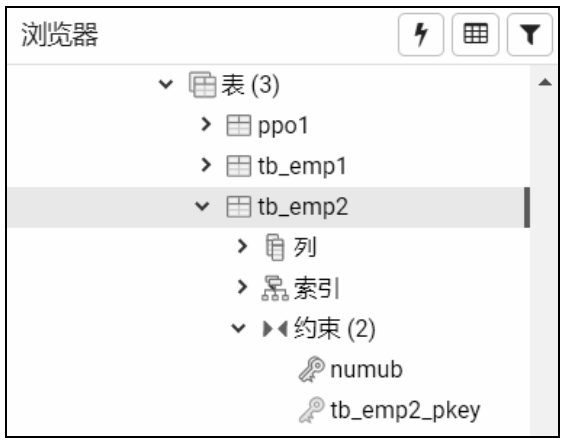

图 4-19　创建的外键约束

使用 SQL 语言可以更灵活地创建外键约束，创建外键的语法规则如下：

```
[CONSTRAINT <外键名>] FOREIGN KEY  字段名 1 [ ,字段名 2,…]
REFERENCES <主表名>  主键列 1 [ ,主键列 2,…]
```

外键名为定义的外键约束的名称，一个表中不能有相同名称的外键；字段名表示从表需要添加外键约束的字段列；主表名，即被从表外键所依赖的表的名称；主键列表示主表中定义的主键字段或者字段组合。

【例 4.6】定义数据表 tb_emp5，并在 tb_emp5 表上创建外键约束。

创建一个部门表 tb_dept1，表结构如表 4.2 所示，SQL 语句如下：

```
CREATE TABLE tb_dept1
(
id          int PRIMARY KEY,
name        VARCHAR(22)    NOT NULL,
location    VARCHAR(50)
);
```

表 4.2　tb_dept1 表结构

字段名称	数据类型	备注
id	INT	部门编号
name	VARCHAR(22)	部门名称
location	VARCHAR(50)	部门位置

定义数据表 tb_emp5，让它的键 deptId 作为外键关联到 tb_dept1 的主键 id，SQL 语句为：

```
CREATE TABLE tb_emp5
(
id          INT PRIMARY KEY,
name        VARCHAR(25),
deptId      INT,
salary      FLOAT,
CONSTRAINT fk_emp_dept1 FOREIGN KEY(deptId) REFERENCES tb_dept1(id)
);
```

以上语句执行成功之后，在表 tb_emp5 上添加了名称为 fk_emp_dept1 的外键约束，外键名称为 deptId，其依赖于表 tb_dept1 的主键 id。

提　示

关联指的是在关系型数据库中相关表之间的联系。它是通过相容或相同的属性或属性组来表示的。子表的外键必须关联父表的主键，且关联字段的数据类型必须匹配，如果类型不一样，那么创建子表时就会出现错误。

4.1.4　使用非空约束

非空约束（Not Null Constraint）指字段的值不能为空。对于使用了非空约束的字段，如果用户在添加数据时没有指定值，数据库系统会报错。

非空约束的语法规则如下：

```
字段名 数据类型 not null
```

【例 4.7】定义数据表 tb_emp6，指定员工的名称不能为空，SQL 语句如下：

```
CREATE TABLE tb_emp6
(
id        INT PRIMARY KEY,
name      VARCHAR(25) NOT NULL,
deptId    INT,
salary    FLOAT,
CONSTRAINT fk_emp_dept2    FOREIGN KEY (deptId) REFERENCES tb_dept1(id)
);
```

执行后在 tb_emp6 中创建了一个 Name 字段，其插入值不能为空（NOT NULL）。

4.1.5　使用唯一性约束

唯一性约束（Unique Constraint）要求添加该约束的列字段的值唯一，允许为空，但只能出现一个空值。唯一约束可以确保一列或者几列不出现重复值。

使用对象浏览器创造唯一性约束的具体操作步骤如下。

步骤 01　展开需要创建唯一性约束的数据表，右击【约束】节点，并在弹出的快捷菜单中选择【创建】➢【唯一约束】菜单命令，如图 4-20 所示。

步骤 02　弹出【创建 – 唯一约束】对话框，在【名称】文本框中输入唯一性约束的名称，如图 4-21 所示。

图 4-20　选择【唯一约束】菜单命令

图 4-21　【创建–唯一约束】对话框

步骤 03 选择【定义】选项卡，在【列】下拉列表中选择需要添加唯一性约束的字段，如图 4-22 所示。

步骤 04 设置完成后，单击【保存】按钮，此时在对象浏览器中即可看到新创建的唯一约束，如图 4-23 所示。

图 4-22 【定义】选项卡

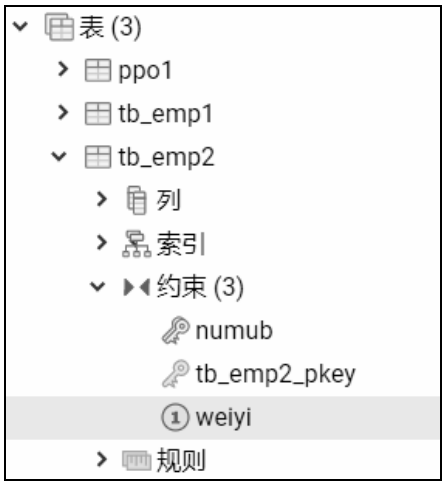

图 4-23 新建唯一约束

使用 SQL 语句也可以轻松创建唯一约束，具体的语法规则如下：

（1）在定义完列之后直接指定唯一约束，语法规则如下：

```
字段名 数据类型 UNIQUE
```

【例 4.8】定义数据表 tb_dept2，指定部门的名称唯一，SQL 语句如下：

```
CREATE TABLE tb  dept2
(
id          INT PRIMARY KEY,
name        VARCHAR(22) UNIQUE,
location    VARCHAR(50)
);
```

（2）在定义完所有列之后指定唯一约束，语法规则如下：

```
[CONSTRAINT <约束名>] UNIQUE(<字段名>)
```

【例 4.9】定义数据表 tb_dept3，指定部门的名称唯一，SQL 语句如下：

```
CREATE TABLE tb_dept3
(
id          INT PRIMARY KEY,
name        VARCHAR(22),
location    VARCHAR(50),
```

```
CONSTRAINT STH UNIQUE(name)
);
```

UNIQUE 和 PRIMARY KEY 的区别：一个表中可以有多个字段声明为 UNIQUE，但只能由一个 PRIMARY KEY 声明；声明为 PRIMAY KEY 的列不允许有空值，但是声明为 UNIQUE 的字段允许有空值（NULL）的存在。

4.1.6　使用默认约束

默认约束（Default Constraint）指定某列的默认值。例如，男性同学较多，性别就可以默认为‘男’。如果插入一条新的记录时没有为这个字段赋值，那么系统会自动为这个字段赋值为‘男’。

默认约束的语法规则如下：

```
字段名 数据类型 DEFAULT 默认值
```

【例 4.10】定义数据表 tb_emp7，指定员工的部门编号默认为 1111，SQL 语句如下：

```
CREATE TABLE tb_emp7
(
id          INT PRIMARY KEY,
name      VARCHAR(25) NOT NULL,
deptId    INT DEFAULT 1111,
salary    FLOAT,
CONSTRAINT fk_emp_dept3    FOREIGN KEY (deptId) REFERENCES tb_dept1(id)
);
```

以上语句执行成功之后，表 tb_emp7 上的字段 deptId 拥有了一个默认的值 1111。新插入的记录如果没有指定部门编号，则默认的都为 1111。

4.2　修改数据表

修改表指的是修改数据库中已经存在的数据表的结构。常用的修改表的操作有：修改表名、修改字段数据类型或字段名、增加和删除字段、修改字段的排列位置、更改表的存储引擎、删除表的外键约束等。本节将对和修改表有关的操作进行讲解。

4.2.1　修改表名

在对象浏览器中修改表名的方法比较简单，选择需要修改名称的数据表右击，在弹出的快捷菜单中选择【属性】菜单命令，即可在弹出的对话框中修改数据表的名称，单击【保存】按钮即可，如图 4-24 所示。

PostgreSQL 是通过 ALTER TABLE 语句来实现表名修改的，具体的语法规则如下：

```
ALTER TABLE <旧表名> RENAME    TO    <新表名>;
```

【例 4.11】将数据表 tb_dept3 改名为 tb_department3。

使用 ALTER TABLE 将表 tb_dept3 改名为 tb_department3，SQL 语句如下：

```
ALTER TABLE tb_dept3 RENAME TO tb_department3;
```

语句执行之后，检验表 tb_dept3 是否改名成功。在对象浏览器中执行刷新操作，即可看到数据表的名称已修改，如图 4-25 所示。

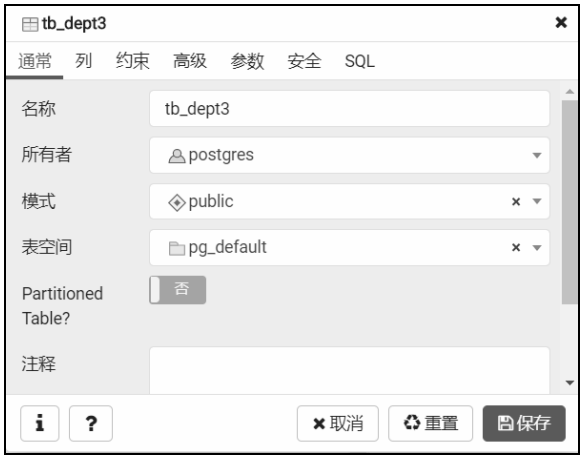

图 4-24 【表 - tb_dept3】对话框

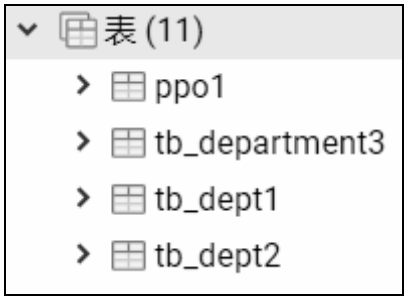

图 4-25 修改数据表的名称

读者可以在修改表名称时查看修改前后两个表的结构。修改表名并不修改表的结构，因此修改名称后的表和修改名称前的表的结构必然是相同的。

4.2.2 修改字段的数据类型

对于创建好的字段，用户可以修改它的数据类型。使用 SQL 语言可以修改字段的数据类型。修改字段的数据类型，就是把字段的数据类型转换成另一种数据类型。在 PostgreSQL 中修改字段数据类型的语法规则如下：

```
ALTER TABLE <表名> ALTER COLUMN <字段名> TYPE <数据类型>
```

其中，“表名”指要修改数据类型的字段所在表的名称，“字段名”指需要修改的字段，“数据类型”指修改后字段的新数据类型。

【例 4.12】将数据表 tb_dept1 中 name 字段的数据类型由 VARCHAR(22)修改成 VARCHAR(30)。

输入如下 SQL 语句并执行：

```
ALTER TABLE tb_dept1
    ALTER COLUMN name TYPE text;
```

在对象浏览器中选择 tb_dept1 数据表的【name】字段，执行刷新操作，然后右击并在弹出的快捷菜单中选择【属性】菜单命令，即可在弹出的对话框中看到数据类型发生了变化，如图 4-26 所示。

图 4-26　查看数据类型

提　示

由于不同类型的数据在机器中存储的方式及长度并不相同，修改数据类型可能会影响到数据表中已有的数据记录，因此当数据库表中已经有数据时，不要轻易修改数据类型。

4.2.3　修改字段名

使用对象浏览修改字段名的方法比较简单，只需要在字段属性对话框中修改即可。下面讲述如何使用 SQL 语句实现修改字段名。

在 PostgreSQL 中修改表字段名的语法规则如下：

```
ALTER TABLE <表名> RENAME <旧字段名> TO　<新字段名> <新数据类型>;
```

其中，“旧字段名”指修改前的字段名；“新字段名”指修改后的字段名；“新数据类型”指修改后的数据类型，如果不需要修改字段的数据类型，将新数据类型设置成与原来一样即可，但数据类型不能为空。

【例 4.13】将数据表 tb_dept1 中的 location 字段名称改为 loc，数据类型保持不变，SQL 语句如下：

```
ALTER TABLE tb_dept1 RENAME location TO loc;
```

命令执行后，选择【列】节点后刷新，即可看到字段的名称发生了变化，如图 4-27 所示。

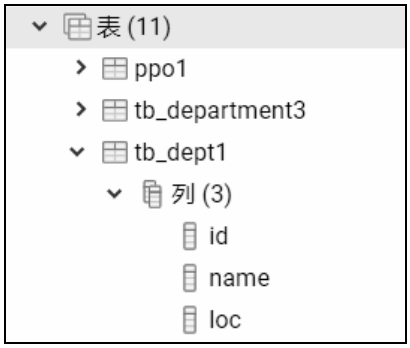

图 4-27　修改字段名称

提　示

用户也可以选择【字段名】后右击，并在弹出的快捷菜单中选择【属性】菜单命令，然后在弹出对话框的【名称】文本框中修改字段的名称。

4.2.4　添加字段

随着业务需求的变化，可能需要在已经存在的表中添加新的字段。在对象浏览器中添加字段的具体操作步骤如下：

步骤 01　选择需要添加字段的数据表，右击并在弹出的快捷菜单中选择【创建】➢【列】菜单命令，如图 4-28 所示。

步骤 02　弹出【创建–列】对话框，输入字段的名称和数据类型等基本参数后，单击【保存】按钮即可添加新的字段，如图 4-29 所示。

图 4-28　选择【列】菜单命令

图 4-29　【创建–列】对话框

一般情况下，一个完整字段包括字段名、数据类型、完整性约束。使用 SQL 语句添加字段的语法格式如下：

```
ALTER TABLE <表名> ADD  COLUMN  <新字段名> <数据类型>
```

1．添加无完整性约束条件的字段

【例 4.14】在数据表 tb_dept1 中添加一个没有完整性约束的 INT 类型的字段 managerid(部

门经理编号），SQL 语句如下：

```
ALTER TABLE tb_dept1 ADD COLUMN managerid INT;
```

命令执行后，选择【列】节点并刷新，即可看到新添加的字段 managerid，如图 4-30 所示。

图 4-30　创建字段 managerId

2．添加有完整性约束条件的字段

【例 4.15】在数据表 tb_dept1 中添加一个不能为空的 VARCHAR(12)类型的字段 column1，SQL 语句如下：

```
ALTER TABLE tb_dept1 ADD COLUMN column1 VARCHAR(12) not  null;
```

命令执行后，查看新添加字段的属性，即可看到【不为 NULL】选项为【是】，如图 4-31 所示。

图 4-31　查看字段的属性

4.2.5 删除字段

删除字段是将数据表中的某个字段从表中移除。对于不用的字段，可以进行删除操作。使用对象浏览器删除对象的具体操作如下：

步骤 01 选择需要删除的字段，右击并在弹出的快捷菜单中选择【删除/移除】菜单命令，如图 4-32 所示。

步骤 02 弹出【删除列么？】对话框，单击【OK】按钮，即可删除所选的字段，如图 4-33 所示。

图 4-32 选择【删除/移除】菜单命令

图 4-33 【删除列么？】对话框

删除字段的语法格式如下：

```
ALTER TABLE <表名> DROP <字段名>;
```

“字段名”指需要从表中删除的字段的名称。

【例 4.16】删除数据表 tb_dept1 表中的 managerid 字段。

删除 managerid 字段，SQL 语句如下：

```
ALTER TABLE tb_dept1 DROP managerid;
```

命令执行后，刷新【列】节点，可以看到 tb_dept1 表中已经不存在名称为 managerid 的字段，删除字段成功，如图 4-34 所示。

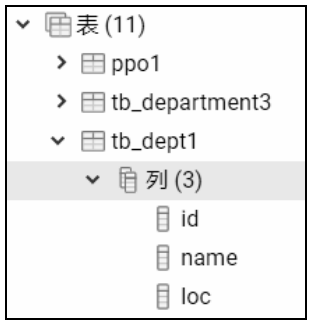

图 4-34 删除字段 managerid

4.2.6 删除表的外键约束

对于数据库中定义的外键，如果不再需要，可以将其删除。外键一旦删除，就会解除主表和从表间的关联关系。

步骤 01 选择需要删除的外键约束，右击并在弹出的快捷菜单中选择【删除/移除】菜单命令，如图 4-35 所示。

步骤 02　弹出【删除外键么?】对话框，单击【OK】按钮，即可删除所选的外键，如图 4-36 所示。

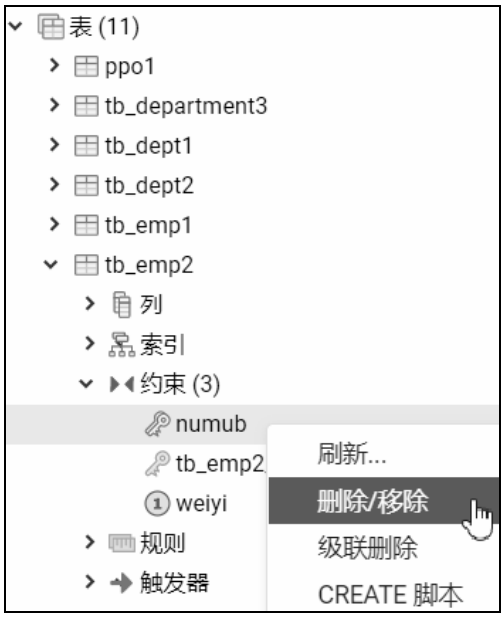

图 4-35　选择【删除/移除】菜单命令

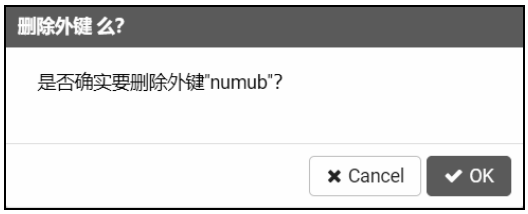

图 4-36　【删除外键么?】对话框

在 PostgreSQL 中，删除外键的语法格式如下：

```
ALTER TABLE <表名> DROP CONSTRAINT    <外键约束名>
```

【例 4.17】删除数据表 tb_emp9 中的外键约束。

首先创建表 tb_emp9，创建外键 deptId 关联 tb_dept1 表的主键 id，SQL 语句如下：

```
CREATE TABLE tb_emp9
(
id          INT PRIMARY KEY,
name      VARCHAR(25),
deptId    INT,
salary      FLOAT,
CONSTRAINT fk_emp_dept    FOREIGN KEY (deptId) REFERENCES tb_dept1(id)
);
```

执行完上面的 SQL 语句后，刷新【表】节点，然后展开新创建的数据表的【约束】节点，即可看到创建的外键，如图 4-37 所示。

图 4-37　创建的数据表的外键

可以看到，已经成功添加了表的外键。下面删除外键约束，SQL 语句如下：

```
ALTER TABLE tb_emp9 DROP CONSTRAINT fk_emp_dept;
```

执行完毕之后，将删除表 tb_emp9 的外键约束，刷新 tb_emp9 表的【约束】节点，即可看到原有的名称为 fk_emp_dept 的外键约束删除成功，如图 4-38 所示。

图 4-38　删除表 tb_emp9 的外键约束

4.3　删除数据表

删除数据表是将数据库中已经存在的表从数据库中删除。注意，删除表的同时，表的定义和表中所有的数据均会被删除，因此，在删除操作前，最好对表中的数据做个备份，以免造成无法挽回的后果。本节将详细讲解数据库表的删除方法。

4.3.1　删除没有被关联的表

在 PostgreSQL 中，使用 DROP TABLE 可以一次删除一个或多个没有被其他表关联的数据表。语法格式如下：

```
DROP TABLE [IF EXISTS]表 1, 表 2, ... ,表 n;
```

其中，“表 n”是指要删除的表的名称，后面可以同时删除多个表，只需要将要删除的表

名依次写在后面，相互之间用逗号隔开即可。如果要删除的数据表不存在，就会提示错误信息，如图 4-39 所示。

参数“IF EXISTS”用于在删除前判断删除的表是否存在，加上该参数后，再删除表的时候，如果表不存在，SQL 语句可以顺利执行，但是会发出警告信息(warning)，如图 4-40 所示。

图 4-39　提示错误信息

图 4-40　警告提示信息

在前面的例子中，已经创建了名为 tb_dept1 的数据表，如果没有，请读者输入语句，创建该表，SQL 语句如【例 4.8】。下面使用删除语句将该表删除。

【例 4.18】删除数据表 tb_dept2，SQL 语句如下：

```
DROP TABLE IF EXISTS tb_dept2;
```

语句执行完毕之后，数据表列表中已经不存在名称为 tb_dept2 的表，删除操作成功。

4.3.2　删除被其他表关联的主表

在数据表之间存在外键关联的情况下，如果直接删除父表，结果会显示失败，原因是直接删除将破坏表的参照完整性。如果必须要删除，可以先删除与之关联的子表，再删除父表。但是这样同时删除了两个表中的数据。有的情况下可能要保留子表，这时若要单独删除父表，只需将关联的表的外键约束条件取消，就可以删除父表了。下面讲解这种方法。

在数据库中创建两个关联表。首先，创建表 tb_dept2，SQL 语句如下：

```
CREATE TABLE tb_dept2
(
id          INT PRIMARY KEY,
name        VARCHAR(22),
location    VARCHAR(50)
);
```

接下来创建表 tb_emp，SQL 语句如下：

```
CREATE TABLE tb_emp
```

```
(
id          INT PRIMARY KEY,
name        VARCHAR(25),
deptId      INT,
salary      FLOAT,
CONSTRAINT fk_emp_dept    FOREIGN KEY (deptId) REFERENCES tb_dept2(id)
);
```

以上 SQL 语句创建了两个关联表 tb_dept2 和 tb_emp。其中，tb_emp 表为子表，具有名称为 fk_emp_dept 外键约束；tb_dept2 为父表，其主键 id 被子表 tb_emp 所关联。

【例 4.19】删除被数据表 tb_emp 关联的数据表 tb_dept2。

首先，直接删除父表 tb_dept2，输入如下删除语句：

```
DROP TABLE tb_dept2;
```

执行上述 SQL 语句后，【消息】窗口中显示错误信息，如图 4-41 所示。可以看到，如前所讲，在存在外键约束时，主表不能被直接删除。

图 4-41　错误信息

接下来，解除关联子表 tb_emp 的外键约束，SQL 语句如下：

```
ALTER TABLE tb_emp DROP CONSTRAINT fk_emp_dept;
```

语句成功执行后，将取消表 tb_emp 和 tb_dept2 之间的关联关系。此时，可以输入删除语句，将原来的父表 tb_dept2 删除，SQL 语句如下：

```
DROP TABLE tb_dept2;
```

再次执行上述 SQL 语句，提示执行成功，如图 4-42 所示。在对象浏览器中刷新【表】节点，可以看到数据表列表中已经不存在名称为 tb_dept2 的表。

图 4-42 执行删除操作

4.4 PostgreSQL 11 的新特性——新增带默认值的字段不再重写数据表

在 PostgreSQL 11 版本之前，新增一个不带默认值的字段时，只需要更新数据字典。如果新增一个非空带默认值的字段，则需要重写数据表，表越大，执行时间越长。例如：

```
ALTER TABLE tb_dept1 ADD COLUMN mmid INT DEFAULT 1001;
```

对于一个数据比较大的数据表，上述语句将执行以下两步：

- 第一步，添加不带默认值的字段。
- 第二步，批量刷新新增字段的默认值。

在 PostgreSQL 11 版本中，上述语句将不再重写数据表，从而提高了执行效率。
下面通过案例测试 PostgreSQL 11 版本中新增带默认值的字段时是否重写数据表。

步骤 01 新增测试数据表 tt，语句如下：

```
CREATE TABLE tt(id int, name text);
```

步骤 02 插入 500 万行测试数据，语句如下：

```
INSERT INTO tt (id,name ) SELECT n, n || '_ALTER TABLE TEST ' FROM generate_series (1,5000000) n;
```

步骤 03 查看数据表 tt 的 relfilenode 和 relpages 信息，语句如下：

```
SELECT relname,relfilenode, relpages FROM pg_class WHERE relname='tt';
```

执行结果如图 4-43 所示。

步骤 04 新增带默认值的字段 ffname，语句如下：

```
ALTER TABLE tt ADD COLUMN ffname text DEFAULT '默认值';
```

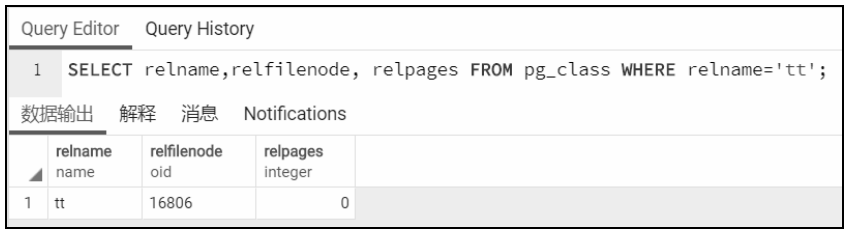

图 4-43　查看 relfilenode 和 relpages 信息

步骤 05　再次查看数据表 tt 的 relfilenode 和 relpages 信息，语句如下：

```
SELECT relname,relfilenode, relpages FROM pg_class WHERE relname='tt';
```

执行结果如图 4-44 所示。从结果可以看出，relfilenode 和 relpages 的数据并没有发生变化。

图 4-44　再次查看 relfilenode 和 relpages 信息

4.5　综合案例——数据表的基本操作

本章全面介绍了 PostgreSQL 中数据表的各种操作，如创建表、添加各类约束、查看表结构，以及修改和删除表。读者应该掌握这些基本的操作，为以后的学习打下坚实的基础。本章给出一个综合案例，让读者全面回顾一下本章的知识要点，并通过这些操作来检验自己是否已经掌握了数据表的常用操作。

1. 案例目的

创建、修改和删除表，掌握数据表的基本操作。

创建数据库 company，按照表 4.3 和表 4.4 给出的表结构在 company 数据库中创建两个数据表 offices 和 employees，按照操作过程完成对数据表的基本操作。

表 4.3　offices 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
officeCode	INT	是	否	是	是	否
city	VARCHAR(50)	否	否	是	否	否
address	VARCHAR(50)	否	否	是	否	否
country	VARCHAR(50)	否	否	是	否	否
postalCode	VARCHAR(15)	否	否	是	否	否

表 4.4 employees 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
employeeNumber	INT	是	否	是	是	是
lastName	VARCHAR(50)	否	否	是	否	否
firstName	VARCHAR(50)	否	否	是	否	否
mobile	VARCHAR(25)	否	否	是	否	否
officeCode	INT	否	是	是	否	否
jobTitle	VARCHAR(50)	否	否	是	否	否
birth	DATE	否	否	否	否	否
note	VARCHAR(255)	否	否	否	否	否
sex	VARCHAR(5)	否	否	否	否	否

2. 案例操作过程

步骤 01 登录数据库。

启动 pgAdmin 4，输入密码连接服务器。

步骤 02 创建数据库 company，执行过程如下：

```
CREATE DATABASE company;
```

命令执行后，刷新数据库，即可看到新创建的数据库 company，表明语句成功执行，如图 4-45 所示。

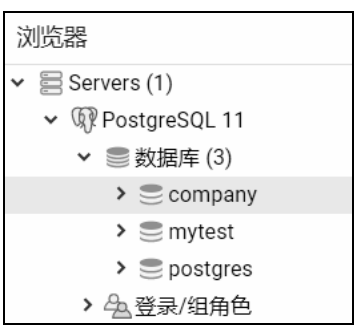

图 4-45 创建的数据库 company

步骤 03 创建表 offices。

创建表 offices 的语句如下：

```
CREATE TABLE offices
(
officeCode   INT NOT NULL UNIQUE,
city         VARCHAR(50) NOT NULL,
address      VARCHAR(50) NOT NULL,
```

```
country        VARCHAR(50) NOT NULL,
postalCode     VARCHAR(15) NOT NULL,
PRIMARY KEY    (officeCode)
);
```

语句执行后，便创建了一个名称为 offices 的数据表。在对象浏览器中选择【company】节点，然后执行刷新操作，依次展开【company】➢【模式】➢【public】➢【表】➢【offices】➢【列】节点，即可看到新建表的字段，如图 4-46 所示。

图 4-46　创建数据表 offices

步骤 04　创建表 employees。

创建表 employees 的语句如下：

```
CREATE TABLE employees
(
employeeNumber    INT NOT NULL PRIMARY KEY,
lastName           VARCHAR(50) NOT NULL,
firstName          VARCHAR(50) NOT NULL,
mobile             VARCHAR(25) NOT NULL,
officeCode         INT NOT NULL,
jobTitle           VARCHAR(50) NOT NULL,
birth              DATE,
note               VARCHAR(255),
sex                VARCHAR(5),
CONSTRAINT office_fk FOREIGN KEY(officeCode)    REFERENCES offices(officeCode)
);
```

语句执行后，便创建了一个名称为 employees 的数据表。在对象浏览器中选择【company】节点，然后执行刷新操作，依次展开【company】➢【模式】➢【public】➢【表】➢【employees】

➢【列】节点，即可看到新建表的字段，如图 4-47 所示。

图 4-47　创建数据表 employees

现在数据库中已经创建好了 employees 和 offices 两个数据表。要检查表的结构是否按照要求创建，选择需要查看的字段，在右侧的【属性】选项卡下可以查看详细信息，如图 4-48 所示。经过查看可以知道，两个表中的字段分别满足【表 4.3】和【表 4.4】中要求的数据类型和约束类型。

图 4-48　查看表的字段类型

步骤 05　将表 employees 的 birth 字段改名为 employee_birth。

修改字段名，需要用到 ALTER TABLE 语句，输入语句如下：

```
ALTER TABLE employees RENAME birth TO employee_birth;
```

SQL 命令执行后，选择【列（9）】节点后刷新，可以看到表中只有 employee_birth 字段，已经没有名称为 birth 的字段了，修改名称成功，如图 4-49 所示。

步骤 06 修改 sex 字段，数据类型为 CHAR(1)，非空约束。

修改字段数据类型，需要用到 ALTER TABLE 语句，输入语句如下：

```
ALTER TABLE employees ALTER COLUMN sex TYPE CHAR(1);
```

命令执行后，执行刷新操作，然后选择【sex】字段，在右侧的【属性】选项卡下即可看到 sex 字段的数据类型由前面的 VARCHAR(5)修改为 CHAR(1)，如图 4-50 所示。

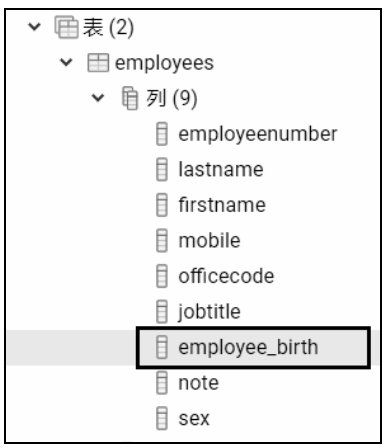

图 4-49 修改字段名称

图 4-50 修改 sex 字段的数据类型

步骤 07 修改 sex 字段为非空约束。

```
ALTER TABLE employees ALTER COLUMN sex SET NOT NULL;
```

命令执行后，执行刷新操作，然后选择【sex】字段，右击该字段并在弹出的快捷菜单中选择【属性】菜单命令，在弹出的【sex】对话框中选择【定义】选项卡，即可看到【不为 NULL】选项已被设置为【是】，表示该列不允许空值，修改成功，如图 4-51 所示。

步骤 08 删除字段 note。

删除字段，需要用到 ALTER TABLE 语句，输入语句如下：

```
ALTER TABLE employees DROP note;
```

命令执行后，刷新【列】节点，可以看到 employees 表中返回了 8 个列字段，note 字段已经不在表结构中，删除字段成功，如图 4-52 所示。

图 4-51　【sex】对话框

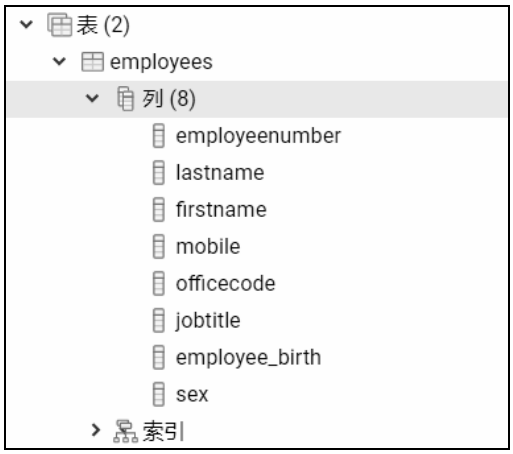

图 4-52　删除字段 note

步骤 09　增加字段名 favoriate_activity，数据类型为 VARCHAR(100)。

增加字段，需要用到 ALTER TABLE 语句，输入语句如下：

```
ALTER TABLE employees ADD COLUMN favoriate_activity VARCHAR(100);
```

命令执行后，选择【列】节点后刷新，即可看到新添加的字段 favoriate_activity，数据类型为 VARCHAR(100)，允许空值，添加新字段成功，如图 4-53 所示。

图 4-53　增加字段名 favoriate_activity

步骤 10　删除表 offices。

在创建表 employees 表时设置了表的外键，该表关联了其父表的 officeCode 主键，如前面所述，删除关联表时，要先删除子表 employees 的外键约束才能删除父表，因此必须先删除 employees 表的外键约束。

① 删除 employees 表的外键约束，输入如下语句：

```
ALTER TABLE employees DROP CONSTRAINT office_fk;
```

其中，office_fk 为 employees 表的外键约束名称，语句执行成功后即可删除 offices 父表。

② 删除表 offices，输入如下语句：

```
DROP TABLE offices;
```

语句执行成功后，刷新【表】节点，可以看到数据库中已经没有名称为 offices 的表了，删除表成功，如图 4-54 所示。

图 4-54　删除表 offices

步骤 11 将表 employees 名称修改为 employees_info。

修改数据表名，需要用到 ALTER TABLE 语句，输入语句如下：

```
ALTER TABLE employees RENAME TO employees_info;
```

语句执行之后，检验表 employees 是否改名成功。在对象浏览器中选择【表】节点并刷新，即可看到数据表的名称已成功修改为 employees_info，如图 4-55 所示。

图 4-55　修改数据表的名称

4.6 常见问题及解答

疑问 1：表删除和修改操作时需注意什么问题？

表删除操作将把表的定义和表中的数据一起删除，因此执行删除操时应当慎重，在删除表前，最好对表中的数据进行备份，这样当操作失误时可以对数据进行恢复，以免造成无法挽回的后果。

同样的，在使用 ALTER TABLE 进行表的基本修改操作时，在执行操作过程之前也应该确保对数据进行完整的备份，因为数据库的改变无法撤销，如果添加了一个不需要的字段，可以将其删除；相同的，如果删除了一个你需要的列，那么该列下面的所有数据将会丢失。

另外，在对表进行修改时，首先要查看该表是否和其他表存在依赖关系，如果存在依赖关系，就应先解除该表的依赖关系后再进行修改操作，否则会导致其他表出错。

疑问 2：每一个表中都要有一个主键吗？

并不是每一个表中都需要主键。一般，多个表之间进行连接操作时需要用到主键。因此，并不需要为每个表都建立主键，而且有些情况最好不使用主键。

4.7 经典习题

1．创建数据库 Market，在 Market 中创建数据表 customers（表结构如表 4.5 所示），并按要求进行操作。

表 4.5 customers 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
c_num	INT	是	否	是	是	是
c_name	VARCHAR(50)	否	否	否	否	否
c_contact	VARCHAR(50)	否	否	否	否	否
c_city	VARCHAR(50)	否	否	否	否	否
c_birth	DATE	否	否	是	否	否

（1）创建数据库 Market。

（2）创建数据表 customers，在 c_num 字段上添加主键约束和自增约束，在 c_birth 字段上添加非空约束。

（3）将 c_name 字段数据类型改为 VARCHAR(70)。

（4）将 c_contact 字段改名为 c_phone。

（5）增加 c_gender 字段，数据类型为 CHAR(1)。

（6）将表名修改为 customers_info。

（7）删除字段 c_city。

2．在 Market 中创建数据表 orders（表结构如表 4.6 所示），并按要求进行操作。

表 4.6　orders 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
o_num	INT	是	否	是	是	否
o_date	DATE	否	否	否	否	否
c_id	VARCHAR(50)	否	否	否	否	否

（1）创建数据表 orders，在 o_num 字段上添加主键约束和自增约束，在 c_id 字段上添加外键约束，关联 customers 表中的主键 c_num。

（2）删除 orders 表的外键约束，然后删除表 customers。

第 5 章　数据类型和运算符

数据表由多列字段构成，每一个字段指定了不同的数据类型。指定字段的数据类型之后，也就决定了向字段插入的数据内容。例如，当要插入数值的时候，可以将它们存储为整数类型，也可以将它们存储为字符串类型；不同的数据类型决定了 PostgreSQL 在存储它们的时候使用的方式，也决定了在使用它们的时候选择什么运算符号进行运算。本章将介绍 PostgreSQL 中的数据类型和常见运算符。

- 熟悉常见数据类型的概念和区别
- 掌握选择数据类型的方法
- 熟悉常见运算符的概念和区别
- 掌握综合案例中运算符的运用方法

5.1　PostgreSQL 数据类型介绍

PostgreSQL 支持多种数据类型，主要有整数类型、浮点数类型、任意精度数值、日期/时间类型、字符串类型、二进制类型、布尔类型和数组类型等。本节主要介绍这些常见类型的使用方法。

5.1.1　整数类型

数值型数据类型主要用来存储数字。PostgreSQL 提供多种数值数据类型，不同的数据类型提供不同的取值范围，可以存储的值范围越大，其所需要的存储空间越大。PostgreSQL 主要提供的整数类型有 MALLINT，INT（INTEGER）和 BIGINT。

表 5.1 列出了 PostgreSQL 中的整数类型。

表 5.1　PostgreSQL 中的整数型数据类型

类型名称	说明	存储空间
MALLINT	小范围的整数	2 字节
INT（INTEGER）	普通大小的整数	4 字节
BIGINT	大整数	8 字节

从中可以看到，不同类型整数存储所需的字节数是不同的，占用字节数最小的是 MALLINT 类型，占用字节最大的是 BITINT 类型。相应的，占用字节越多的类型所能表示的数值范围越大。根据占用字节数可以求出每一种数据类型的取值范围，例如 MALLINT 需要 2 字节（16 bits）来存储，那么 MALLINT 的最大值为 2^{15}-1，即 32767。其他类型的整数的取值范围计算方法相同，如表 5.2 所示。

表 5.2　不同整数类型的取值范围

数据类型	取值范围
SMALLINT	-32768 到 32767
INT（INTEGER）	-2147483648 到 2147483647
BIGINT	-9223372036854775808 到 9223372036854775807

【例 5.1】创建表 tmp1，其中字段 x、y、z 的数据类型依次为 SMALLINT、INT、BIGINT，SQL 语句如下：

```
CREATE TABLE tmp1 (x SMALLINT, y INT, z BIGINT );
```

语句执行成功之后，创建 3 种整数类型的字段。

不同的整数类型有不同的取值范围，并且需要不同的存储空间，因此应该根据实际需要选择合适的类型，这样有利于提高查询的效率和节省存储空间。整数类型是不带小数部分的数值，现实生活中很多地方需要用到带小数的数值，下面将介绍 PostgreSQL 中支持的小数类型。

5.1.2　浮点数类型

PostgreSQL 中使用浮点数来表示小数。浮点类型有两种：REAL 和 DOUBLE PRECISION。它们的含义如表 5.3 所示。

表 5.3　PostgreSQL 中的小数类型

类型名称	说明	存储需求
REAL	6 位十进制数字精度	4 字节
DOUBLE PRECISION	15 位十进制数字精度	8 字节

在大多数系统平台上，REAL 类型的范围是 1E-37 到 1E+37，精度至少是 6 位小数。DOUBLE PRECISION 的范围通常是 1E-307 到 1E+308，精度至少是 15 位数字，太大或者太小的数值都会导致错误。

PostgreSQL 也支持 SQL 标准表示法 float 和 float(p)，用于声明非精确的数值类型。其中，p 声明以二进制位表示的最低可接受精度。在选取 real 类型的时候，PostgreSQL 接受 float(1)到 float(24)，在选取 double precision 的时候，接受 float(25)到 float(53)。在允许范围之外的 p 值将导致一个错误。没有声明精度的 float 将被当作 double precision。

【例 5.2】创建表 tmp2，其中字段 x、y、z 的数据类型依次为 FLOAT(5)、REAL 和 DOUBLE PRECISION，SQL 语句如下：

```
CREATE TABLE tmp2 (x FLOAT(5),  y REAL,  z DOUBLE PRECISION );
```

语句执行成功之后，创建 3 种浮点类型的字段。

在 PostgreSQL 中，在浮点类型中有几个特殊值：Infinity 表示正无穷大，-Infinity 表示负无穷大，NaN 表示不是一个数字。

5.1.3 任意精度类型

在 PostgreSQL 中，使用 NUMERIC（M，N）表示任意精度类型的数值。其中，M 称为精度，表示总共的位数；N 称为标度，表示小数的位数。例如，563.186 的精度为 6、标度为 3。

NUMERIC 的有效取值范围由 M 和 D 的值决定。如果改变 M 而固定 D，则其取值范围将随 M 的变大而变大。另外，如果用户指定的精度超出精度外，就会进行四舍五入处理。

【例 5.3】创建表 tmp3，其中字段 x、y 数据类型依次为 NUMERIC(5,1) 和 NUMERIC (5,2)，向表中插入数据 9.12、9.15。创建表 tmp3，SQL 语句如下：

```
CREATE TABLE tmp3 ( x NUMERIC (5,1), y NUMERIC (5,2));
```

向表中插入数据，SQL 语句如下：

```
INSERT INTO tmp3 VALUES(9.12, 9.15);
```

查看表中的数据，SQL 语句如下：

```
SELECT * FROM tmp3;
```

语句执行后，结果如图 5-1 所示。可以看到 PostgreSQL 对插入的数据 9.12 进行了四舍五入处理。

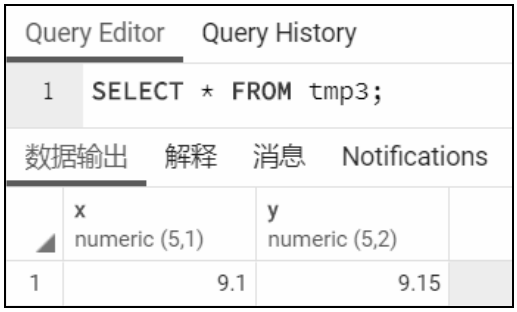

图 5-1 查看表中的数据

5.1.4 日期与时间类型

PostgreSQL 中有多种表示日期的数据类型，主要有 TIME、DATE、TIMESTAMP 和 INTERVAL。每一个类型都有合法的取值范围，

当指定确实不合法的值时系统将“零”值插入数据库中。本节将介绍 PostgreSQL 日期和时间数据类型的使用方法。表 5.4 列出了 PostgreSQL 中的日期与时间类型。

表 5.4　日期/时间数据类型

类型名称	含义	日期范围	存储需求
TIME	只用于一日内的时间	00:00:00～24:00:00	8 字节
DATE	只用于日期	4713 BC～5874897 AD	4 字节
TIMESTAMP	日期和时间	4713 BC～5874897 AD	8 字节

在格里高利历法里没有零年，所以数字上的 1BC 是公元零年。

提　示

另外，对于 TIME 和 TIMESTAMP 类型，默认情况下为 without time zone（不带时区），如果需要，可以设置为带时区（with time zone）。

1．TIME

TIME 类型用于只需要时间信息的值，在存储时需要 8 字节，格式为 HH:MM:SS。其中，HH 表示小时；MM 表示分钟；SS 表示秒。TIME 类型的取值范围为 00:00:00～24:00:00。

【例 5.4】创建数据表 tmp4，定义数据类型为 TIME 的字段 t，向表中插入值‘10:05:05’、‘23:23’。

创建表 tmp4，SQL 语句如下：

```
CREATE TABLE tmp4( t TIME );
```

向表中插入数据，SQL 语句如下：

```
INSERT INTO tmp4 values('10:05:05 '), ('23:23');
```

查看结果，SQL 语句如下：

```
SELECT * FROM tmp4;
```

语句执行后，结果如图 5-2 所示。

由结果可以看到，‘10:05:05’被转换为 10:05:05；‘23:23’被转换为 23:23:00。

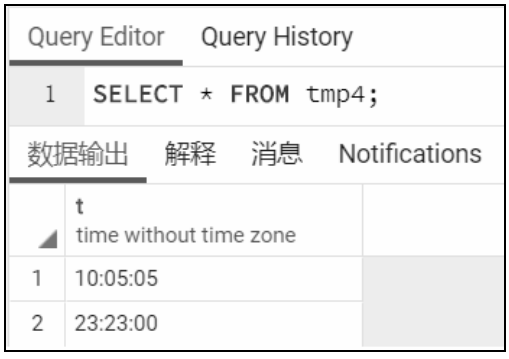

图 5-2　SQL 语句执行结果

【例 5.5】向表 tmp4 中插入值‘101112’，SQL 语句如下：

向表中插入数据，SQL 语句如下：

```
INSERT INTO tmp4 values('101112');
```

查看结果，SQL 语句如下：

```
SELECT * FROM tmp4;
```

语句执行后，结果如图 5-3 所示。

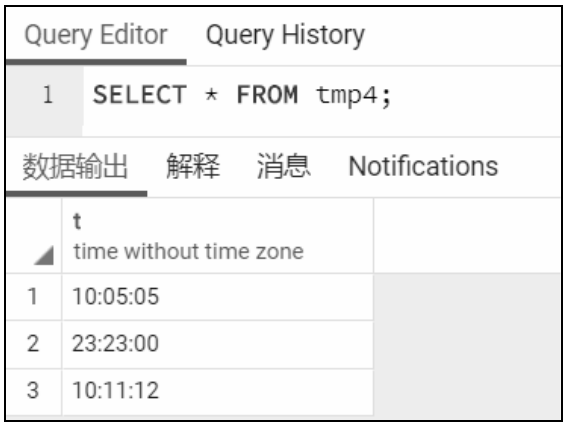

图 5-3 SQL 语句执行结果

由结果可以看到，‘101112’被转换为 10:11:12。

也可以是用系统日期函数向 TIME 字段列插入值。

【例 5.6】向 tmp4 表中插入系统当前时间，SQL 语句如下：

由于由时间函数获得的时间是带时区的，因此需要先将字段属性修改为带时区类型的时间：

```
ALTER TABLE tmp4
    ALTER COLUMN t TYPE time without time zone;
```

删除表中的数据：

```
DELETE FROM tmp4;
```

向表中插入数据，SQL 语句如下：

```
INSERT INTO tmp4 values (CURRENT_TIME) ,(NOW());
```

查看结果，SQL 语句如下：

```
SELECT * FROM tmp4;
```

语句执行后，结果如图 5-4 所示。

由结果可以看到，获取系统当前的日期时间插入到 TIME 类型列，因为读者输入语句的时间不确定，所以获取的值与这里的可能不同，但都是系统当前的日期时间值，并显示所在的时区。

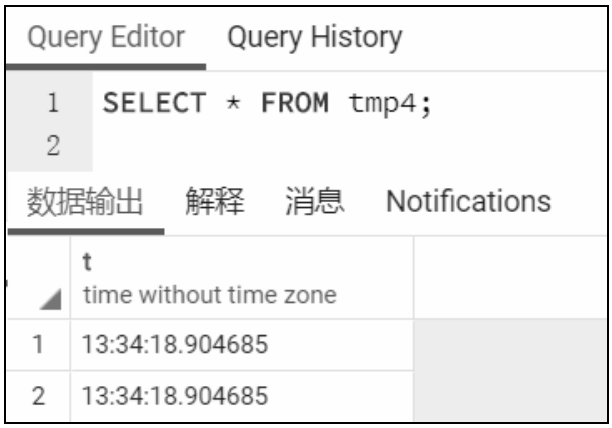

图 5-4　SQL 语句执行结果

2．DATE 类型

DATE 类型用在仅需要日期值时，没有时间部分，在存储时需要 4 字节，日期格式为‘YYYY-MM-DD’。其中，YYYY 表示年；MM 表示月；DD 表示日。在给 DATE 类型的字段赋值时，可以使用字符串类型或者数字类型的数据插入，只要符合 DATE 的日期格式即可。

（1）以‘YYYY-MM-DD’或者‘YYYYMMDD’字符串格式表示的日期。例如，输入‘2012-12-31’或者‘20121231’，插入数据库的日期都为 2012-12-31。

（2）以‘YY-MM-DD’或者‘YYMMDD’字符串格式表示的日期。在这里，YY 表示两位的年值，包含两位年值的日期会令人模糊，因为不知道世纪。PostgreSQL 使用以下规则解释两位年值：‘00～69’范围的年值转换为‘2000～2069’；‘70～99’范围的年值转换为‘1970～1999’。例如，输入‘12-12-31’，插入数据库的日期为 2012-12-31；输入‘981231’，插入数据的日期为 1998-12-31。

（3）利用 CURRENT_DATE 或者 NOW()插入当前系统日期。

【例 5.7】创建数据表 tmp5，定义数据类型为 DATE 的字段 d，向表中插入“YYYY-MM-DD”和“YYYYMMDD”字符串格式日期，SQL 语句如下：

首先，创建表 tmp5：

```
CREATE TABLE tmp5(d DATE);
```

向表中插入“YYYY-MM-DD“和“YYYYMMDD”格式日期：

```
INSERT INTO tmp5 values('1998-08-08'),('19980808'),('20101010');
```

查看插入结果：

```
SELECT * FROM tmp5;
```

语句执行后，结果如图 5-5 所示。

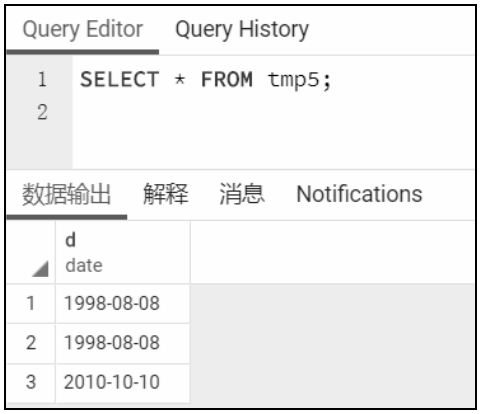

图 5-5 SQL 语句执行结果

可以看到各个不同类型的日期值都正确地插入到了数据表中。

【例 5.8】向 tmp5 表中插入“YY-MM-DD“和“YYMMDD”字符串格式日期，SQL 语句如下：

首先，删除表中的数据：

```
DELETE FROM tmp5;
```

向表中插入“YY-MM-DD“和“YYMMDD”格式日期：

```
INSERT INTO tmp5 values('99-09-09'),( '990909'), ( '000101') ,( '121212');
```

查看插入结果：

```
SELECT * FROM tmp5;
```

语句执行后，结果如图 5-6 所示。

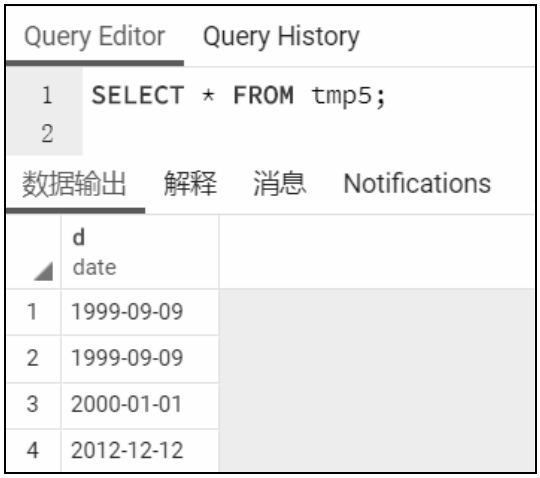

图 5-6 SQL 语句执行结果

【例 5.9】向 tmp5 表中插入系统当前日期，SQL 语句如下：

删除表中的数据：

```
DELETE FROM tmp5;
```

向表中插入系统当前日期：

```
INSERT INTO tmp5 values(NOW() );
```

查看插入结果：

```
SELECT * FROM tmp5;
```

语句执行后，结果如图 5-7 所示。

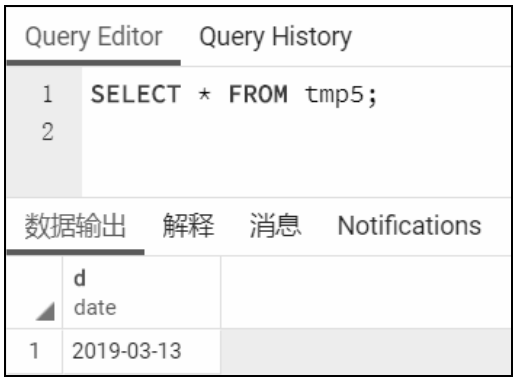

图 5-7　SQL 语句执行结果

NOW()函数返回日期和时间值，在保存到数据库时，只保留了其日期部分。

3．TIMESTAMP

TIMESTAMP 的日期格式为 YYYY-MM-DD HH:MM:SS。在存储时需要 8 字节，因此在插入数据时要保证在合法的取值范围内。

【例 5.10】创建数据表 tmp7，定义数据类型为 TIMESTAMP 的字段 ts，向表中插入值‘1996-02-02 02:02:02’、NOW()，SQL 语句如下：

创建数据表和字段：

```
CREATE TABLE tmp7( ts TIMESTAMP);
```

向表中插入数据：

```
INSERT INTO tmp7 values ('1996-02-02 02:02:02'),( NOW() );
```

查看插入结果：

```
SELECT * FROM tmp7;
```

语句执行后，结果如图 5-8 所示。

图 5-8 SQL 语句执行结果

由结果可以看到，‘1996-02-02 02:02:02’被转换为 1996-02-02 02:02:02；NOW()被转换为系统当前日期时间 2019-03-13 13:38:55。

4. 创建带时区的日期和时间类型

【例 5.11】创建数据表 tmp7h，定义数据类型为 TIME 的字段 th，向表中插入值‘10:05:05 PST’、‘10:05:05’。

创建表 tmp7h，SQL 语句如下：

```
CREATE TABLE tmp7h( t TIME with time zone);
```

向表中插入数据，SQL 语句如下：

```
INSERT INTO tmp7h values('10:05:05 PST '), ('10:05:05');
```

查看结果，SQL 语句如下：

```
SELECT * FROM tmp7h;
```

语句执行后，结果如图 5-9 所示。

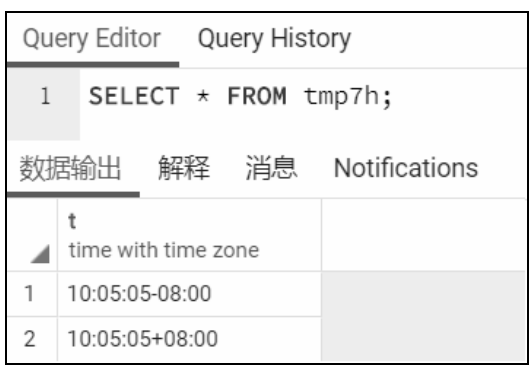

图 5-9 SQL 语句执行结果

由结果可以看到，创建了带时区的时间类型，其中 PST 为西 8 区。如果不指定时区，默认为东 8 区。另外，带时区的日期类型的创建与之相似，这里不再重复讲述。

5.1.5 字符串类型

字符串类型用来存储字符串数据，除了可以存储字符串数据之外，还可以存储其他数据，比如图片和声音的二进制数据。字符串可以进行区分或者不区分大小写的串比较，另外还可以进行模式匹配查找。在 PostgreSQL 中，字符串类型是指 CHAR、VARCHAR 和 TEXT。表 5.5 列出了 PostgreSQL 中的字符串数据类型。

表 5.5 PostgreSQL 中字符串数据类型

类型名称	说明
CHAR(n) CHARACTER (n)	固定长度非二进制字符串，不足补空白
VARCHAR(n) CHARACTER VARYING(n)	变长非二进制字符串，有长度限制
TEXT	变长非二进制字符串，无长度限制

1. CHARACTER(n)和 CHARACTER VARYING(n)

其中，n 是一个正整数。CHARACTER(n)和 CHARACTER VARYING(n)都可以存储最多 n 个字符的字符串。试图存储更长的字符串到这些类型的字段里会产生一个错误，除非超出长度的字符都是空白，这种情况下该字符串将被截断为最大长度。如果要存储的字符串比声明的长度短，类型为 character 的数值将会用空白填满；而类型为 CHARACTER VARYING 的数值将只存储短些的字符串。

【例 5.12】创建 tmp8 表，定义字段 ch 和 vch 数据类型依次为 CHARACTER（4）、CHARACTER VARYING（4），向表中插入不同长度的字符串，SQL 语句如下：

创建表 tmp8：

```
CREATE TABLE tmp8(
ch   CHARACTER (4),   vch   CHARACTER VARYING (4)
);
```

输入数据：

```
INSERT INTO tmp8 VALUES('ab', 'ab'),
('abcd', 'abcd'),
('ab  ', 'ab  ');
```

查询结果：

```
SELECT concat('(', ch, ')'), concat('(',vch,')') FROM tmp8;
```

语句执行后，结果如图 5-10 所示。

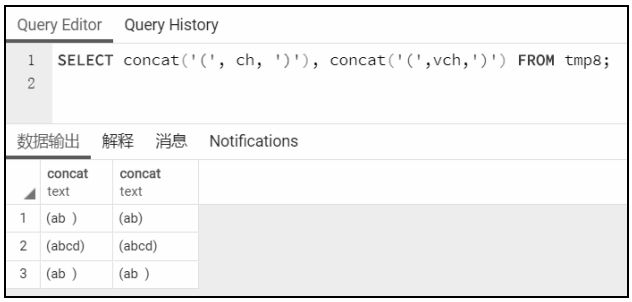

图 5-10 SQL 语句执行结果

从查询结果可以看到，ch 在保存“ab”时在右侧填充空格以达到指定的长度，而 vch 字段仅仅保留了“ab”。

CHARACTER 类型中填充的空白是无意义的。例如，在比较两个 CHARACTER 值的时候，填充的空白都会被忽略，在转换成其他字符串类型的时候，CHARACTER 值里面的空白会被删除。注意，在 CHARACTER VARYING 和 TEXT 数值里，结尾的空白是有意思的。

提 示

上例中的 concat()函数的语法为 CONCAT(str1,str2,…)，返回结果为连接参数产生的字符串。

如果插入的字符长度超过规定的长度，例如插入字符为“abcde”，代码如下：

```
INSERT INTO tmp8 VALUES('abcde', 'abcde')
```

语句执行后，【消息】窗口中显示错误信息，如图 5-11 所示。可见系统会阻止这个数值的插入，并且提示语法错误，说明插入的字符串长度已经大于可以插入的最大值。

图 5-11 SQL 语句执行结果

2. TEXT 类型

PostgreSQL 中的 TEXT 类型可以存储任何长度的字符串。尽管类型 TEXT 不是 SQL 标准，但是许多其他 SQL 数据库系统中也有。

【例 5.13】创建 tmp9 表，定义字段 te，数据类型为 TEXT，向表中插入不同长度的字符串，SQL 语句如下：

创建表 tmp9：

```
CREATE TABLE tmp9(te    TEXT);
```

输入数据：

```
INSERT INTO tmp9 VALUES('ab'),('abcd'),('ab    ');
```

查询结果：

```
SELECT concat('(', te, ')') FROM tmp9;
```

语句执行后，结果如图 5-12 所示。

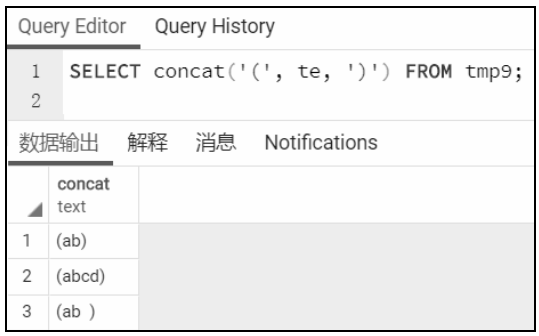

图 5-12　SQL 语句执行结果

5.1.6　二进制类型

PostgreSQL 支持两类字符型数据：文本字符串和二进制字符串。前面讲解了存储文本的字符串类型，这一节将讲解 PostgreSQL 中存储二进制数据的数据类型。

PostgreSQL 提供了 BYTEA 类型，用于存储二进制字符串。BYTEA 类型存储空间为 4 字节加上实际的二进制字符串。

【例 5.14】创建表 tmp10，定义 BYTEA 类型的字段 b，向表中插入二进制数据“E'\\000'”。

创建表 tmp10，SQL 语句如下：

```
CREATE TABLE tmp10( b BYTEA );
```

插入数据：

```
INSERT INTO tmp10 VALUES(E'\\000');
```

查询插入结果：

```
SELECT * FROM tmp10;
```

语句执行后，结果如图 5-13 所示。

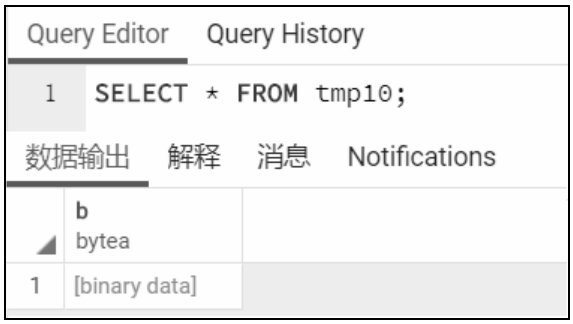

图 5-13 SQL 语句执行结果

5.1.7 布尔类型

PostgreSQL 提供了 BOOLEAN 布尔数据类型。BOOLEAN 用 1 字节来存储，提供了 TRUE（真）和 FALSE（假）两个值。

另外，用户可以使用其他有效文本值替代 TRUE 和 FALSE。替代 TRUE 的文本值为't'、'true'、'y'、'yes'和'1'，替代 FALSE 的文本值为 'f'、'false'、'n' 、'no' 和'0'。

【例 5.15】创建表 tmp11，定义 BYTEA 类型的字段 b，向表中插入布尔型数据“TRUE”和“FALSE”。

创建表 tmp11，SQL 语句如下：

```
CREATE TABLE tmp11( b BOOLEAN );
```

插入数据：

```
INSERT INTO tmp11 VALUES(TRUE),(FALSE),('y'),('no'),('0');
```

查询插入结果：

```
SELECT * FROM tmp11;
```

语句执行后，结果如图 5-14 所示。

图 5-14 SQL 语句执行结果

5.1.8 数组类型

PostgreSQL 允许将字段定义成定长或变长的一维或多维数组。数组类型可以是任何基本类型或用户定义类型。

1.声明数组

在 PostgreSQL 中，一个数组类型是通过在数组元素类型名后面附加方括弧来命名的。例如：

```
numb    INT[],
```

```
xuehao    TEXT[][];
zuoye    TEXT[4][4];
```

其中，numb 字段为一维 INT 数组，xuehao 字段为二维 TEXT 数组，zuoye 字段为二维 TEXT 数组，并且声明了数组的长度。不过，目前 PostgreSQL 并不强制数组的长度，所以声明长度和不声明长度是一样的。

另外，对于一维数组，也可以使用 SQL 标准声明，SQL 语句如下：

```
PAY_BY_QUARTER    INT ARRAY[5];
```

此种声明方式仍然不强制数组的长度。

2. 插入数组数值

插入数组元素时，用大括号把数组元素括起来并且用逗号将它们分开。

【例 5.16】创建表 tmp12，定义数组类型的字段 bt，向表中插入一些数组数值。

创建表 tmp12，SQL 语句如下：

```
CREATE TABLE tmp12( bt int[]);
```

插入数据：

```
INSERT INTO tmp12 VALUES('{{1,1,1},{2,2,2},{3,3,3}}');
```

查询插入结果：

```
SELECT * FROM tmp12;
```

语句执行后，结果如图 5-15 所示。

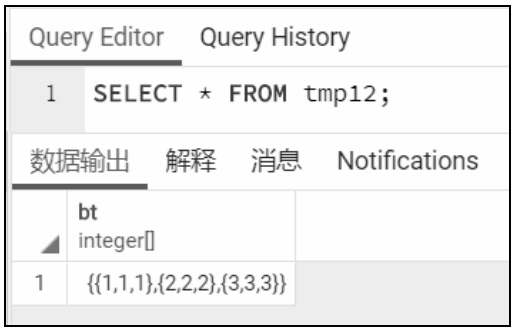

图 5-15　SQL 语句执行结果

5.2　如何选择数据类型

PostgreSQL 提供了大量的数据类型，为了优化存储、提高数据库性能，在任何情况下均应使用最精确的类型，即在所有可以表示该列值的类型中该类型使用的存储空间最少。

1. 整数和浮点数

如果不需要小数部分，就使用整数来保存数据；如果需要表示小数，就使用浮点数类型。对于浮点数据列，存入的数值会被该列定义的小数位进行四舍五入。例如，列值范围为1~99999，若使用整数，则 INT 是最好的类型；若需要存储小数，则使用浮点数类型。

2. 日期与时间类型

PostgreSQL 对于不同种类的日期和时间有很多数据类型，比如 TIME 和 DATE。如果只需要记录时间，就使用 TIME 类型，如果只记录日期，就使用 DATE 类型。

如果同时需要记录日期和时间，就可以使用 TIMESTAMP 类型。默认情况下，当插入一条记录但并没有指定 TIMESTAMP 这个列值时，PostgreSQL 会把 TIMESTAMP 列设为当前的时间，因此当需要插入记录时同时插入当前时间，使用 TIMESTAMP 是方便的。

3. CHAR 与 VARCHAR 之间的特点与选择

CHAR 和 VARCHAR 的区别是，CHAR 是固定长度字符，VARCHAR 是可变长度字符。对于插入数据长度不够时，CHAR 会自动填充插入数据的尾部空格，VARCHAR 不会自动填充尾部空格。

CHAR 是固定长度，所以它的处理速度比 VARCHAR 的速度快，缺点是浪费存储空间。所以对存储不大但在速度上有要求的可以使用 CHAR 类型，反之可以用 VARCHAR 类型来实现。

5.3 常见运算符介绍

运算符连接表达式中的各个操作数，用来指明对操作数所进行的运算。常见的运算有数学运算、比较运算、位运算或者逻辑运算。利用运算符可以更加灵活地使用表中的数据。常见的运算符类型有算术运算符、比较运算符、逻辑运算符和位运算符。本节将介绍各种操作符的特点和使用方法。

5.3.1 运算符概述

运算符是告诉 PostgreSQL 执行特定算术或逻辑操作的符号。PostgreSQL 的内部运算符很丰富，主要有四大类，分别是算术运算符、比较运算符、逻辑运算符、位操作运算符。

1. 算术运算符

用于各类数值运算，包括加（+）、减（-）、乘（*）、除（/）、求余（或称模运算，%）。

2. 比较运算符

比较运算符用于比较运算，包括大于（>）、小于（<）、等于（=）、大于等于（>=）、小于等于（<=）、不等于（!=）以及 IN、BETWEEN AND、GREATEST、LEAST、LIKE 等。

3. 逻辑运算符

逻辑运算符的求值所得结果均为 t（TRUE）、f（FALSE）。这类运算符有逻辑非（NOT）、逻辑与（AND）、逻辑或（OR）。

4. 位操作运算符

参与位操作运算的操作数，按二进制位进行运算。位操作运算符包括位与（&）、位或（|）、位非（~）、位异或（^）、左移（<<）、右移（>>）6 种。

接下来将对 PostgreSQL 中各种运算符的使用进行详细的介绍。

5.3.2 算术运算符

算术运算符是 SQL 中最基本的运算符。PostgreSQL 中的算术运算符如表 5.6 所示。

表 5.6 PostgreSQL 中的算术运算符

运算符	作用
+	加法运算
-	减法运算
*	乘法运算
/	除法运算，返回商
%	求余运算，返回余数

下面分别讨论不同算术运算符的使用方法。

【例 5.16】创建表 tmp14，定义数据类型为 INT 的字段 num，插入值 64，对 num 值进行算术运算：

创建表 tmp14，输入语句如下：

```
CREATE TABLE tmp14 ( num INT);
```

向字段 num 插入数据 64：

```
INSERT INTO tmp14 VALUES (64);
```

对 num 值进行加法和减法运算：

```
SELECT num, num+10, num-10, num+5-3, num+36.5 FROM tmp14;
```

语句执行后，结果如图 5-16 所示。

由计算结果可以看到，可以对 num 字段的值进行加法和减法的运算，而且由于‘+’和‘-’的优先级相同，因此先加后减或者先减后加之后的结果是相同的。

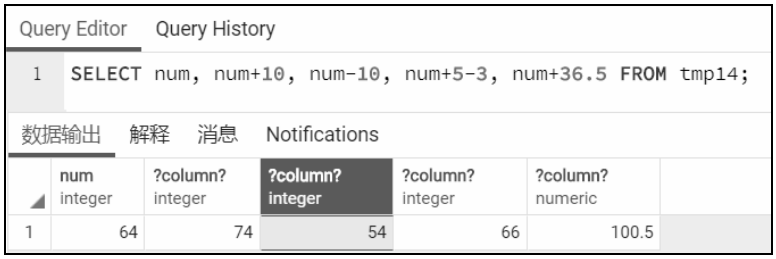

图 5-16 SQL 语句执行结果

【例 5.17】对 tmp14 表中的 num 进行乘法、除法运算。

```
SELECT num, num *2, num /2, num/3, num%3 FROM tmp14;
```

语句执行后，结果如图 5-17 所示。

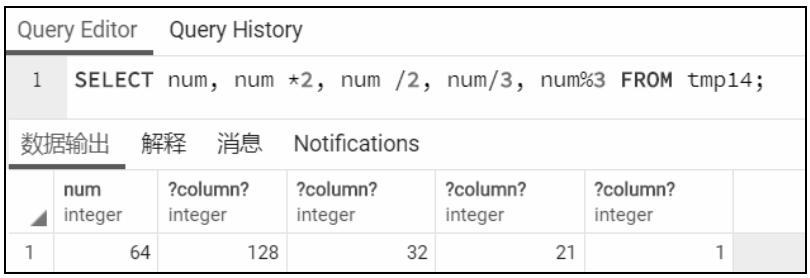

图 5-17 SQL 语句执行结果

由计算结果可以看到，对 num 进行除法运算时，由于 64 无法被 3 整除，因此 PostgreSQL 对 num/3 求商的结果保留到了小数点后面四位，结果为 21.3333；64 除以 3 的余数为 1，因此取余运算 num%3 的结果为 1。

在数学运算时，除数为 0 的除法是没有意义的，因此除法运算中的除数不能为 0，如果被 0 除，就会弹出错误警告。

【例 5.18】用 0 除 num。

```
SELECT num, num / 0, num %0 FROM tmp14;
```

语句执行后，结果如图 5-18 所示。由于存在错误，【数据输出】窗口中并没有显示出相应的结果，在【消息】窗口显示错误信息。

图 5-18 SQL 语句执行结果

5.3.3 比较运算符

一个比较运算符的结果总是 t、f 或者是空值，比较运算符经常在 SELECT 的查询条件子句中使用，用来查询满足指定条件的记录。PostgreSQL 中的比较运算符如表 5.7 所示。

表 5.7 PostgreSQL 中的比较运算符

运算符	作用
=	等于
<> (!=)	不等于
<=	小于等于
>=	大于等于
>	大于
<	小于
LEAST	在有两个或多个参数时，返回最小值
GREATEST	当有两个或多个参数时，返回最大值
BETWEEN . . . AND. . .	判断一个值是否落在两个值之间
IN	判断一个值是 IN 列表中的任意一个值
LIKE	通配符匹配

下面分别讨论不同比较运算符的使用方法。

1．等于运算符（=）

等于“=”用来判断数字、字符串和表达式是否相等。如果相等，返回值为 1，否则返回 0。

【例 5.19】使用“=”进行相等判断，SQL 语句如下：

```
SELECT 1=0, '2'=2, 2=2,'b'='b', (1+3) = (2+1),NULL=NULL;
```

语句执行后，结果如图 5-19 所示。

由结果可以看到，在进行判断时 2=2 和‘2’=2 的返回值相同，都为 true，因为在进行判断时，PostgreSQL 自动进行了转换，把字符‘2’转换成了数字 2；‘b’=‘b’为相同的字符比较，因此返回值为 true；表达式 1+3 和表达式 2+1 的结果不相等，返回值为 false；由于‘=’不能用于空值 NULL 的判断，因此返回值为空。

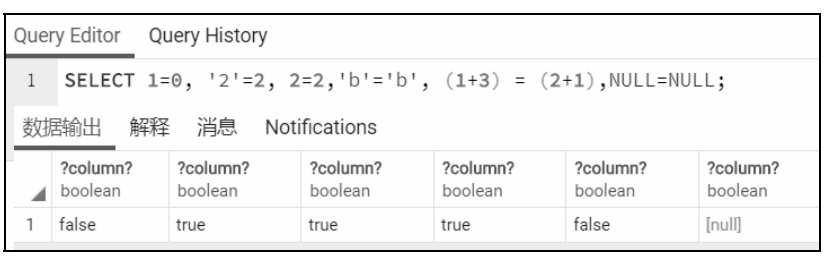

图 5-19 SQL 语句执行结果

等于运算符在进行数值比较时有如下规则：

（1）若有一个或两个参数为 NULL，则比较运算的结果为空。
（2）若同一个比较运算中的两个参数都是字符串，则按照字符串进行比较。
（3）若两个参数均为整数，则按照整数进行比较。
（4）若一个字符串和数字进行相等判断，则 PostgreSQL 可以自动将字符串转换为数字。

2．不等于运算符（<>或者 !=）

‘<>’或者‘!=’用于判断数字、字符串、表达式不相等的判断，如果不相等，返回 t；否则返回 f。这两个运算符不能用于判断空值 NULL。

【例 5.20】使用‘<>’和‘!=’进行不相等的判断，SQL 语句如下：

```
SELECT 'good'<>'god', 1<>2, 4!=4, 5.5!=5, (1+3)!=(2+1),NULL<>NULL;
```

语句执行后，结果如图 5-20 所示。

Query Editor　Query History

```
1  SELECT 'good'<>'god', 1<>2, 4!=4, 5.5!=5, (1+3)!=(2+1),NULL<>NULL;
```

数据输出　解释　消息　Notifications

	?column? boolean	?column? boolean	?column? boolean	?column? boolean	?column? boolean	?column? boolean
1	true	true	false	true	true	[null]

图 5-20　SQL 语句执行结果

由结果可以看到，两个不等于运算符的作用相同，都可以判断数字、字符串、表达式的比较判断。

3．小于等于运算符（<=）

‘<=’用来判断左边的操作数是否小于或者等于右边的操作数。如果小于或者等于，返回值为 t，否则返回值为 f。‘<=’不能用于判断空值 NULL。

【例 5.21】使用‘<=’进行比较判断，SQL 语句如下：

```
SELECT 'good'<='god', 1<=2, 4<=4, 5.5<=5, (1+3) <= (2+1),NULL<=NULL;
```

语句执行后，结果如图 5-21 所示。

Query Editor　Query History

```
1  SELECT 'good'<='god', 1<=2, 4<=4, 5.5<=5, (1+3) <= (2+1),NULL<=NULL;
```

数据输出　解释　消息　Notifications

	?column? boolean	?column? boolean	?column? boolean	?column? boolean	?column? boolean	?column? boolean
1	false	true	true	false	false	[null]

图 5-21　SQL 语句执行结果

由结果可以看到，左边操作数小于或者等于右边时，返回值为 t，例如 4<=4；当左边操作数大于右边时，返回 0，例如‘good’第三个位置的‘o’字符在字母表中的顺序大于‘god’中的第三个位置的‘d’字符，因此返回 f；同样比较 NULL 值时返回空。

4．小于运算符（<）

‘<’运算符用来判断左边的操作数是否小于右边的操作数，如果小于，返回值为 t，否则返回 f。‘<’不能用于判断空值 NULL。

【例 5.22】使用‘<’进行比较判断，SQL 语句如下：

```
SELECT 'good'<'god', 1<2, 4<4, 5.5<5, (1+3) < (2+1),NULL<NULL;
```

语句执行后，结果如图 5-22 所示。

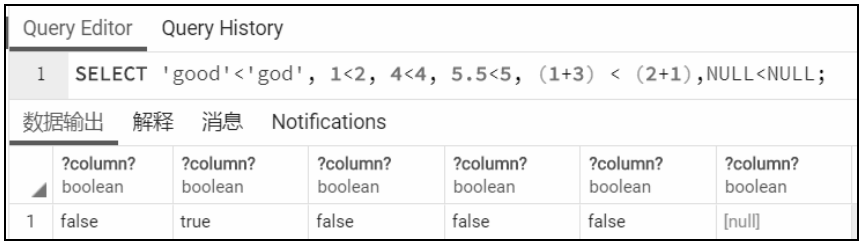

图 5-22　SQL 语句执行结果

由结果可以看到，左边操作数小于或者等于右边时，返回值为 t，例如 1<2；当左边操作数大于右边时，返回 0，例如‘good’第三个位置的‘o’字符在字母表中的顺序大于‘god’中的第三个位置的‘d’字符，因此返回 f；同样比较 NULL 值时返回 NULL。

5．大于等于运算符（>=）

‘>=’运算符用来判断左边的操作数是否大于或者等于右边的操作数，如果大于或者等于，返回值为 t，否则返回 f。‘>=’不能用于判断空值 NULL。

【例 5.23】使用‘>=’进行比较判断，SQL 语句如下：

```
SELECT 'good'>='god', 1>=2, 4>=4, 5.5>=5, (1+3) >= (2+1),NULL>=NULL;
```

语句执行后，结果如图 5-23 所示。

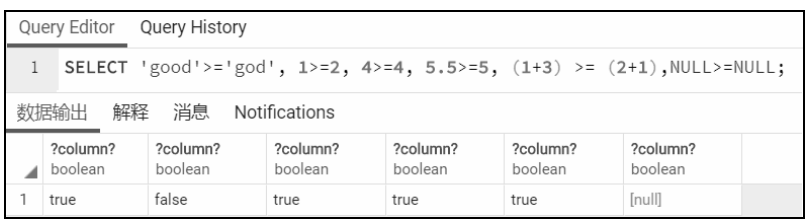

图 5-23　SQL 语句执行结果

由结果可以看到，左边操作数大于或者等于右边时，返回值为 t，例如 4>=4；当左边操作

数小于右边时，返回 f，例如 1>=2；同样比较 NULL 值时返回空值。

6．大于运算符（>）

‘>’运算符用来判断左边的操作数是否大于右边的操作数，如果大于，返回值为 1，否则返回 0。‘>’不能用于判断空值 NULL。

【例 5.24】使用‘>’进行比较判断，SQL 语句如下：

```
SELECT 'good'>'god', 1>2, 4>4, 5.5>5, (1+3) > (2+1),NULL>NULL;
```

语句执行后，结果如图 5-24 所示。

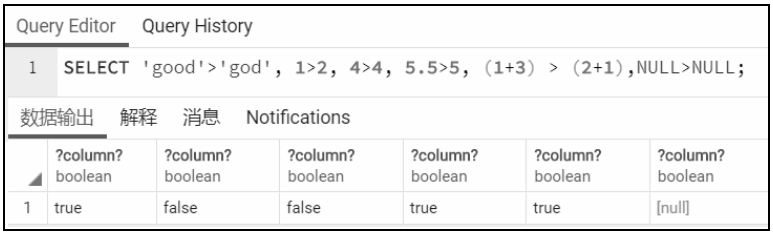

图 5-24　SQL 语句执行结果

由结果可以看到，左边操作数大于或者等于右边时，返回值为 t，例如 5.5>5；当左边操作数小于右边时，返回 f，例如 1>2；同样比较 NULL 值时返回空值。

7．BETWEEN AND 运算符

BETWEEN AND 运算符的语法格式为：expr BETWEEN min AND max。假如 expr 大于或等于 min 且 expr 小于或等于 max，则 BETWEEN 的返回值为 t，否则返回 f。

【例 5.25】使用 BETWEEN AND 进行值区间判断，输入 SQL 语句如下：

```
SELECT 4 BETWEEN 2 AND 5, 4 BETWEEN 4 AND 6,12 BETWEEN 9 AND 10;
```

语句执行后，结果如图 5-25 所示。

由结果可以看到，4 在端点值区间内或者等于其中一个端点值时，BETWEEN AND 表达式返回值为 t；12 并不在指定区间内，因此返回 f。

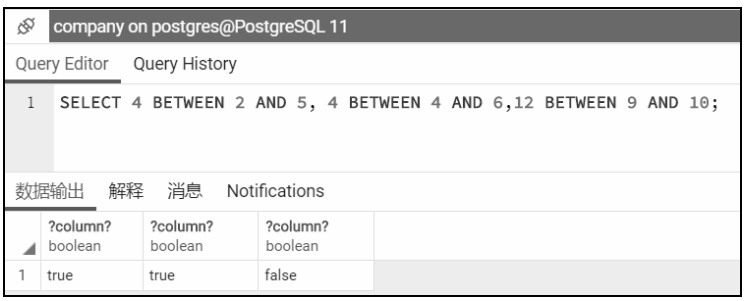

图 5-25　SQL 语句执行结果

【例 5.26】使用 BETWEEN AND 进行字符串的比较，输入 SQL 语句如下：

```
SELECT   'x' BETWEEN 'f' AND 'g', 'b' BETWEEN 'a' AND 'c';
```

语句执行后，结果如图 5-26 所示。

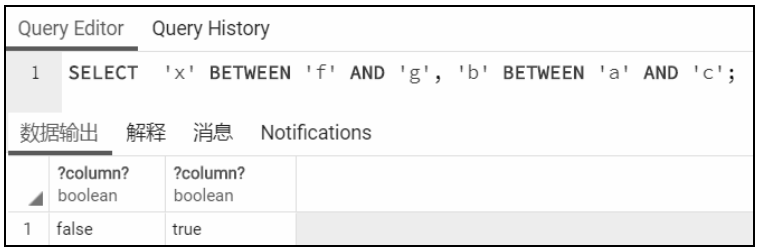

图 5-26　SQL 语句执行结果

对于字符串类型的比较，按字母表中的字母顺序进行比较，‘x’不在指定的字母区间内，因此返回 f，而‘b’位于指定字母区间内，因此返回 t。

8．LEAST 运算符

LEAST 运算符的语法格式为：LEAST(值 1,值 2,...,值 n)。其中，值 n 表示参数列表中有 n 个值。在有两个或多个参数的情况下，返回最小值，假如任意一个值为 NULL，则在比较中忽略不计。

【例 5.27】使用 LEAST 运算符进行大小判断，SQL 语句如下：

```
SELECT least(2,0), least(20.0,3.0,100.5), least('a','c','b'),least(10,NULL);
```

语句执行后，结果如图 5-27 所示。

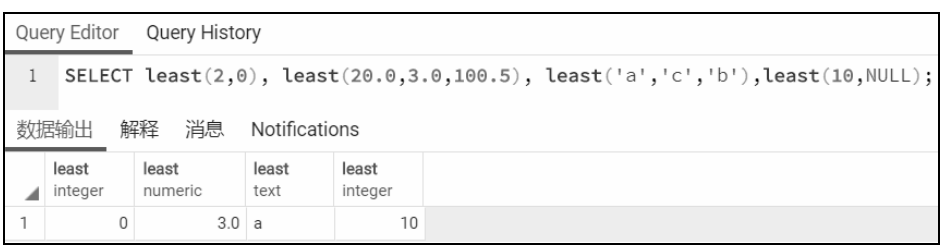

图 5-27　SQL 语句执行结果

由结果可以看到，当参数中是整数或者浮点数的时候，LEAST 将返回其中最小的值；当参数为字符串时，返回字母表中顺序最靠前的字符；当比较值列表中有 NULL 时，忽略不计，返回值为 10。

9．GREATEST (value1,value2,...)

语法格式为：GREATEST(值 1, 值 2,...,值 n)。其中，n 表示参数列表中有 n 个值。当有两个或多个参数时，返回值为最大值，假如任意一个自变量为 NULL，则在比较中忽略不计。

【例 5.28】使用 GREATEST 运算符进行大小判断，SQL 语句如下：

```
SELECT greatest(2,0), greatest(20.0,3.0,100.5), greatest('a','c','b'),greatest(10,NULL);
```

语句执行后，结果如图 5-28 所示。

图 5-28　SQL 语句执行结果

由结果可以看到，当参数中是整数或者浮点数的时候，GREATEST 将返回其中最大的值；当参数为字符串时，返回字母表中顺序最靠后的字符；当比较值列表中有 NULL 时，忽略不计，返回值为 10。

10．IN、NOT IN 运算符

IN 运算符用来判断操作数是否为 IN 列表中的其中一个值，如果是，那么返回值为 t，否则返回值为 f。

NOT IN 运算符用来判断操作数是否为 IN 列表中的其中一个值，如果不是，那么返回值为 t，否则返回值为 f。

【例 5.29】使用 NOT IN 运算符进行判断，SQL 语句如下：

```
SELECT 2 IN (1,2,5,8), 3 IN (1,2,5,8);
```

语句执行后，结果如图 5-29 所示。

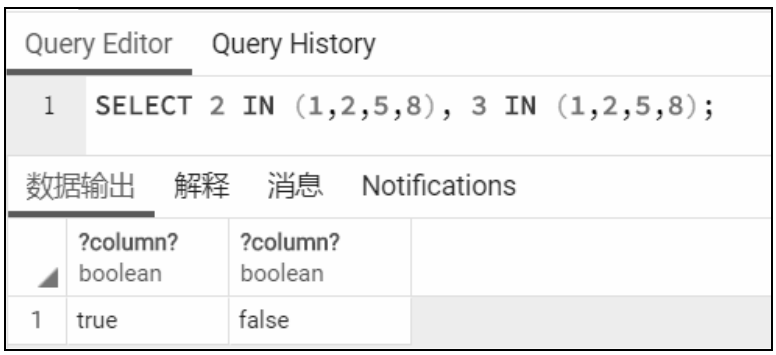

图 5-29　SQL 语句执行结果

【例 5.30】使用 NOT IN 运算符进行判断，SQL 语句如下：

```
SELECT 2 NOT IN (1,2,5,8), 3 NOT IN (1,2,5,8);
```

语句执行后，结果如图 5-30 所示。

图 5-30　SQL 语句执行结果

由结果可以看到，IN 和 NOT IN 的返回值正好相反。

在左侧表达式为 NULL 的情况下，或者表中找不到匹配项并且表中有一个表达式为 NULL 的情况下，IN 的返回值均为空值。

【例 5.31】存在 NULL 值时的 IN 运算，SQL 语句如下：

```
SELECT NULL IN (1,3,5),10 IN (1,3,NULL);
```

语句执行后，结果如图 5-31 所示。

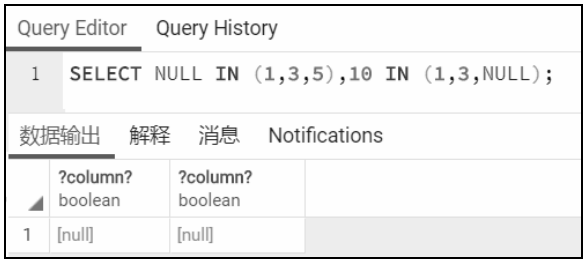

图 5-31　SQL 语句执行结果

IN()运算符也可用于在 SELECT 语句中进行嵌套子查询，在后面的章节中将会讲到。

11．LIKE

LIKE 运算符用来匹配字符串，语法格式为：expr LIKE 匹配条件。如果 epxr 满足匹配条件，就返回 t（TRUE）；如果不匹配，则返回 f（FALSE）。若 expr 或匹配条件中任何一个为 NULL，则结果为空值。

LIKE 运算符在进行匹配时，可以使用下面两种通配符：

（1）‘%’，匹配任何数目的字符，甚至包括零字符。

（2）‘_’，只能匹配一个字符。

【例 5.32】使用运算符 LIKE 进行字符串匹配运算，SQL 语句如下：

```
SELECT 'stud' LIKE 'stud', 'stud' LIKE 'stu_','stud' LIKE '%d','stud' LIKE 't___', 's' LIKE NULL;
```

语句执行后，结果如图 5-32 所示。

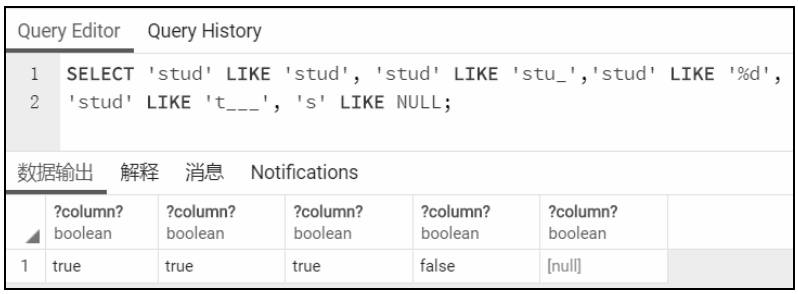

图 5-32 SQL 语句执行结果

由结果可以看到，指定匹配字符串为“stud”。“stud”表示直接匹配“stud”字符串，满足匹配条件，返回 true；“stu_”表示匹配以 stu 开头的长度为 4 个字符的字符串，“stud”正好是 4 个字符，满足匹配条件，因此匹配成功，返回 true；“%d”表示匹配以字母“d”结尾的字符串，“stud”满足匹配条件，匹配成功，返回 true；“t_ _ _”表示匹配以‘t’开头的长度为 4 个字符的字符串，“stud”不满足匹配条件，因此返回 false；当字符‘s’与 NULL 匹配时，结果为空值。

5.3.4 逻辑运算符

在 SQL 中，所有逻辑运算符的求值所得结果均为 TRUE、FALSE 或空值。不同数据库中的表示方法相同。PostgreSQL 中包含表 5.8 所示的逻辑运算符。

表 5.8 PostgreSQL 中的逻辑运算符

运算符	作用
NOT	逻辑非
AND	逻辑与
OR	逻辑或

逻辑运算符的操作参数为布尔型数据。接下来，分别讨论不同的逻辑运算符的使用方法。

1．NOT

逻辑非运算符 NOT 表示当操作数为 TRUE 时，所得值为 f；当操作数为 FALSE 时，所得值为 t，当操作数为 NULL 时，所得的返回值为空值。

【例 5.33】分别使用非运算符“NOT”进行逻辑判断，SQL 语句如下：

```
SELECT NOT '1', NOT 'y', NOT '0',   NOT NULL, NOT 'n';
```

语句执行后，结果如图 5-33 所示。

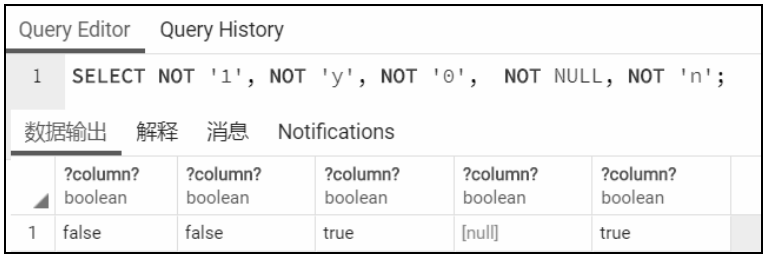

图 5-33　SQL 语句执行结果

由结果可以看到，布尔值'1'的 NOT 值为 false，对于 NULL 的运算结果为空值。

逻辑运算符的参数必须是布尔变量，如果随意输入其他类型的数值，就将会弹出错误提示信息。

2．AND

逻辑与运算符 AND 表示当所有操作数均为 TRUE 并且不为 NULL 时，计算所得结果为 t，当一个或多个操作数为 FALSE 时，所得结果为 f，其余情况返回值为空值。

【例 5.34】分别使用与运算符“AND”进行逻辑判断，SQL 语句如下：

```
SELECT   '1'AND 'y','1'AND '0','1'AND NULL, '0'AND NULL;
```

语句执行后，结果如图 5-34 所示。

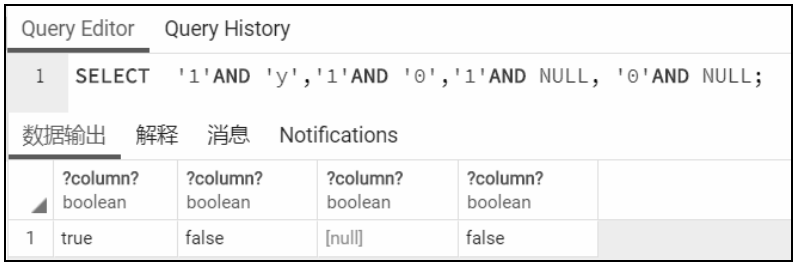

图 5-34　SQL 语句执行结果

由结果可以看到，“'1'AND 'y'”中没有 FALSE 或者 NULL，因此结果为 true；“'1'AND '0'”中有操作数'0'，因此结果为 false；“'1'AND NULL”中有 NULL，返回结果为空值；“'0'AND NULL”中虽然也有 NULL，但是有'0'值，所得结果为 false。

“AND”运算符可以有多个操作数，但要注意：多个操作数运算时，AND 两边一定要使用空格隔开，不然会影响结果的正确性。

3．OR

逻辑或运算符 OR 表示当两个操作数均为非 NULL 值时，任意一个操作数为 TRUE，则结

果为 t，否则结果为 f；有一个操作数为 NULL 时，若另一个操作数为 TRUE，则结果为 t，否则结果为空值；当两个操作数均为 NULL 时，所得结果为空值。

【例 5.35】使用或运算符“OR”进行逻辑判断，SQL 语句如下：

```
SELECT   '1' OR 't' OR '0', '1'OR 'y','1' OR NULL, '0'OR NULL, NULL OR NULL;
```

语句执行后，结果如图 5-35 所示。

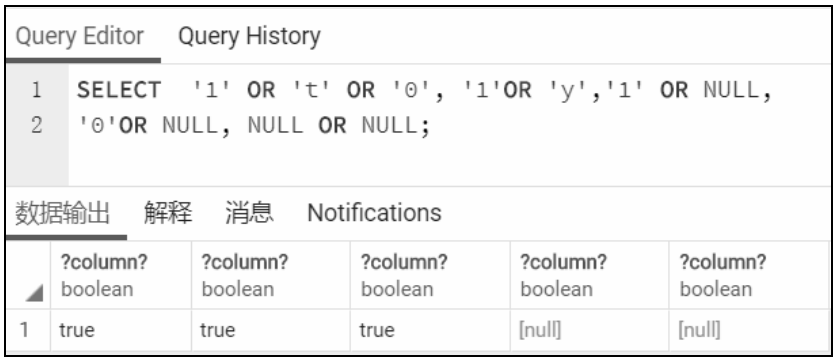

图 5-35 SQL 语句执行结果

由结果可以看到，“'1' OR 't' OR '0'”中有'0'，但同时包含有'1'和 't'，返回结果为 t；“'1'OR 'y'”中没有 FALSE 值，返回结果为 t；“'1' OR NULL”中虽然有 NULL，但是有操作数'1'，返回结果为 t；“'0' OR NULL”中没有 FALSE 值，并且有 NULL，返回结果为空值；“NULL OR NULL”中只有 NULL，返回结果为空值。

5.3.5 运算符的优先级

运算的优先级决定了不同的运算符在表达式中计算的先后顺序。表 5.9 列出了 PostgreSQL 中的各类运算符及优先级。

表 5.9 运算符按优先级由低到高排列

优先级	运算符
最低	=（赋值运算），:=
	|OR
	AND
	NOT
	BETWEEN，CASE，WHEN，THEN，ELSE
	=（比较运算），>=，>，<=，<，<>，!=，IS，LIKE， IN
	-，+
	*，/(DIV)，%(MOD)
	-（负号）
最高	!

可以看到，不同运算符的优先级是不同的。一般情况下，级别高的运算符先进行计算，如果级别相同，PostgreSQL 按表达式的顺序从左到右依次计算。当然，在无法确定优先级的情况下，可以使用圆括号（）来改变优先级，并且这样会使计算过程更加清晰。

5.4 综合案例——运算符的使用

本章首先介绍了 PostgreSQL 中各种数据类型的特点和使用方法，以及如何选择合适的数据类型；接着详细介绍了 PostgreSQL 中各类常见的运算符号的使用，学习了如何使用这些运算符对不同的数据进行运算，包括算术运算、比较运算、逻辑运算等，同时介绍了不同运算符的优先级别。在本章的综合案例中，我们将执行各种常见的运算操作。

1. 案例目的

创建数据表，并对表中的数据进行运算操作，掌握各种运算符的使用方法。

创建表 tmp15，其中包含 VARCHAR 类型的字段 note 和 INT 类型的字段 price，使用运算符对表 tmp15 中不同的字段进行运算；使用逻辑操作符对数据进行逻辑操作。

2. 案例操作过程

下面讲述具体的操作过程。

步骤 01 创建表 tmp15，SQL 语句如下：

```
CREATE TABLE tmp15 (note VARCHAR(100), price INT);
```

步骤 02 向表中插入一条记录，note 值为 “Thisgood”，price 值为 50，SQL 语句如下：

```
INSERT INTO tmp15 VALUES ('Thisgood',50);
```

步骤 03 对 tmp15 表中的整数值字段 price 进行算术运算，执行过程如下：

```
SELECT price, price + 10, price -10, price * 2, price /2, price%3 FROM tmp15 ;
```

语句执行后，结果如图 5-36 所示。

Query Editor　Query History

```
1  SELECT price, price + 10, price -10, price * 2, price /2, price%3 FROM tmp15 ;
```

数据输出　解释　消息　Notifications

	price integer	?column? integer	?column? integer	?column? integer	?column? integer	?column? integer
1	50	60	40	100	25	2

图 5-36　SQL 语句执行结果

步骤 04 对 tmp15 中的整型数值字段 price 进行比较运算，执行过程如下：

```
SELECT price, price> 10, price<10, price != 10, price =10,price <>10 FROM tmp15 ;
```

语句执行后，结果如图 5-37 所示。

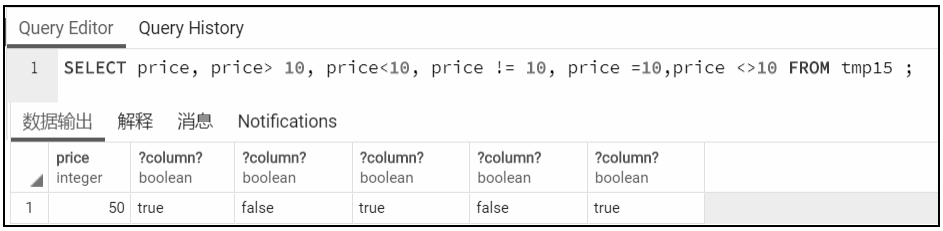

图 5-37 SQL 语句执行结果

步骤 05 判断 price 值是否落在 30~80 区间；返回与 70、30 相比最大的值，判断 price 是否为 IN 列表（10, 20, 50, 35）中的某个值，执行过程如下：

```
SELECT price, price BETWEEN 30 AND 80, GREATEST(price, 70,30), price IN (10, 20, 50,35) FROM tmp15 ;
```

语句执行后，结果如图 5-38 所示。

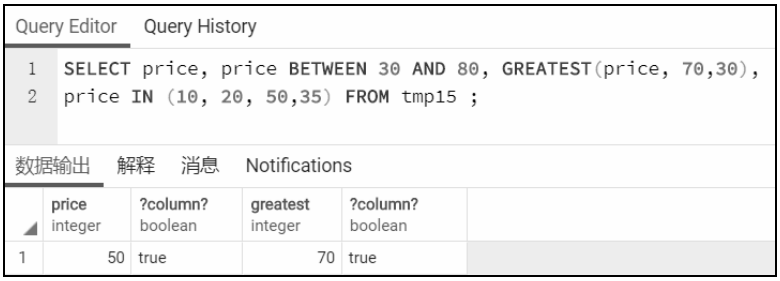

图 5-38 SQL 语句执行结果

步骤 06 对 tmp15 中的字符串数值字段 note 进行比较运算，判断表 tmp15 中 note 字段是否为空；使用 LIKE 判断是否以字母 ‘d’ 开头。执行过程如下：

```
SELECT note, note IS NULL, note LIKE 't%' FROM tmp15 ;
```

语句执行后，结果如图 5-39 所示。

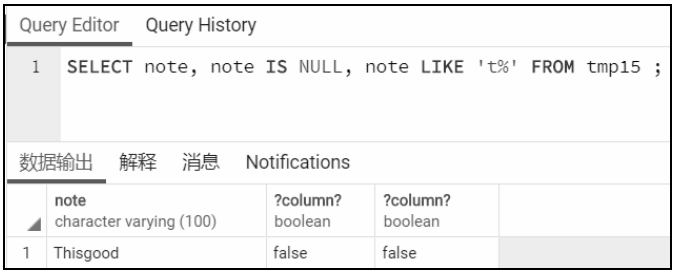

图 5-39 SQL 语句执行结果

5.5 常见问题及解答

疑问 1：PostgreSQL 中可以存储文件吗？

PostgreSQL 中的 TEXT 字段类型可以存储数据量较大的文件，可以使用这些数据类型存储图像、声音或者是大容量的文本内容，例如网页或者文档。虽然使用 TEXT 可以存储大容量的数据，但是对这些字段的处理会降低数据库的性能。如果不是非必要，可以选择只储存文件的路径。

疑问 2：二进制和普通字符串的区别是什么？

二进制字符串和普通字符串的区别有两个：

（1）二进制字符串完全可以存储字节零值以及其他“不可打印的”字节（定义在 32~126 范围之外的字节）。字符串不允许字节零值，并且也不允许那些不符合选定的字符集编码的非法字节值或者字节序列。

（2）对二进制字符串的处理实际上就是处理字节，而对字符串的处理则取决于区域设置。简单地说，二进制字符串适用于存储那些程序员认为是“原始字节”的数据，而字符串适合存储文本。

5.6 经典习题

（1）PostgreSQL 中的小数如何表示？不同表示方法之间有什么区别？

（2）CHAR 和 TEXT 分别适合于存储什么类型的数据？

（3）说明选择常用数据类型的方法。

（4）在 PostgreSQL 中执行如下算术运算：(9-7)*4，8+15/3，17DIV2，39%12。

（5）在 PostgreSQL 中执行如下比较运算：36>27，15>=8， 40<50，15<=15。

（6）在 PostgreSQL 中执行如下逻辑运算：'1'AND 'y'，'1'AND '0'，'1'AND NULL，'0'AND NULL。

第 6 章　PostgreSQL 函数

PostgreSQL 提供了众多功能强大、方便易用的函数。使用这些函数，可以极大地提高用户对数据库的管理。PostgreSQL 中的函数包括数学函数、字符串函数、日期和时间函数、条件判断函数、系统信息函数和加密函数等其他函数。本章将介绍 PostgreSQL 中这些函数的功能和用法。

- 了解什么是 PostgreSQL 的函数
- 掌握各种数学函数的用法
- 掌握各种字符串函数的用法
- 掌握时间和日期函数的用法
- 掌握条件函数的用法
- 掌握系统信息函数的用法
- 掌握加密函数的用法
- 掌握其他特殊函数的用法
- 熟练掌握综合案例中函数操作方法和技巧

6.1　PostgreSQL 函数简介

函数表示对输入参数值返回一个具有特定关系的值，PostgreSQL 提供了大量丰富的函数，在进行数据库管理以及数据的查询和操作时将会经常用到各种函数。通过对数据的处理，数据库功能可以变得更加强大，可以更加灵活地满足不同用户的需求。各类函数从功能方面主要分为数学函数、字符串函数、日期和时间函数、条件判断函数、系统信息函数和加密函数等。下面将分类介绍不同函数的使用方法。

6.2　数学函数

数学函数主要用来处理数值数据，主要的数学函数有绝对值函数、三角函数（包括正弦函

数、余弦函数、正切函数、余切函数等）、对数函数、随机数函数等。在有错误产生时，数学函数将会返回空值 NULL。本节将介绍各种数学函数的功能和用法。

6.2.1 绝对值函数 ABS(x)和返回圆周率的函数 PI()

ABS(x)返回 x 的绝对值。

【例 6.1】求 2、-3.3 和-33 的绝对值，输入语句如下：

```
SELECT ABS(2), ABS(-3.3), ABS(-33);
```

语句执行后，结果如图 6-1 所示。

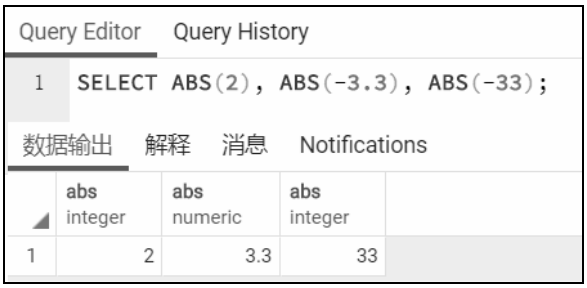

图 6-1 SQL 语句执行结果

正数的绝对值为其本身，所以 2 的绝对值为 2；负数的绝对值为其相反数，所以-3.3 的绝对值为 3.3、-33 的绝对值为 33。

PI()返回圆周率 π 的值，默认的显示小数位数是 6 位。

【例 6.2】返回圆周率值，输入语句如下：

```
SELECT pi();
```

语句执行后，结果如图 6-2 所示，返回结果保留了 15 位有效数字。

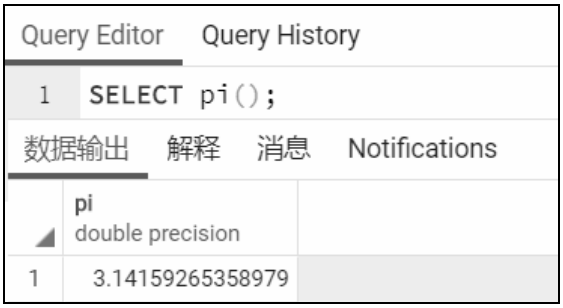

图 6-2 SQL 语句执行结果

6.2.2 平方根函数 SQRT(x)和求余函数 MOD(x,y)

SQRT(x)返回非负数 x 的二次方根。

【例 6.3】求 9、40 的二次平方根，输入语句如下：

```
SELECT SQRT(9), SQRT(40);
```

语句执行后，结果如图 6-3 所示。

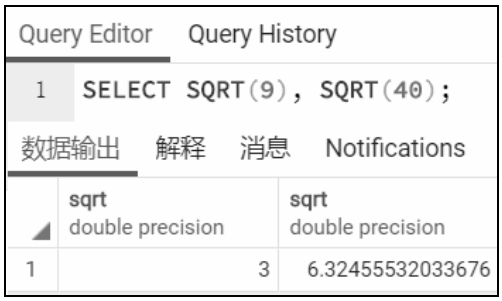

图 6-3 SQL 语句执行结果

注 意

3 的平方等于 9，因此 9 的二次平方根为 3；40 的平方根为 6.32455532033676，负数没有平方根，如果所求值为负数，将会提示错误信息。

MOD(x,y)返回 x 被 y 除后的余数。MOD() 对于带有小数部分的数值也起作用，返回除法运算后的精确余数。

【例 6.4】对(31,8)、(234,10)、(45.5,6)进行求余运算，输入语句如下：

```
SELECT MOD(31,8),MOD(234, 10),MOD(45.5,6);
```

语句执行后，结果如图 6-4 所示。

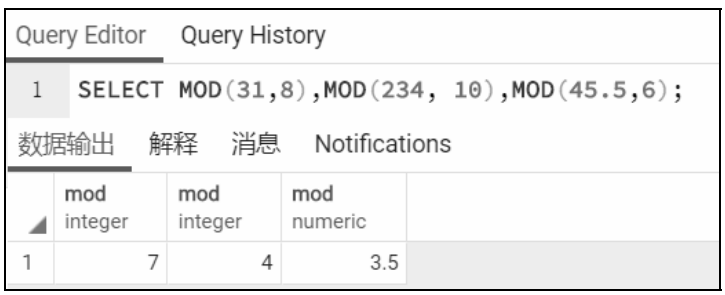

图 6-4 SQL 语句执行结果

6.2.3 获取整数的函数 CEIL(x)、CEILING(x)和 FLOOR(x)

CEIL(x)和 CEILING(x)的意义相同，返回不小于 x 的最小整数值，返回值转化为一个 BIGINT。

【例 6.5】使用 CEIL 和 CEILING 函数返回最小整数，输入语句如下：

```
SELECT   CEIL(-3.35),CEILING(3.35);
```

语句执行后，结果如图 6-5 所示。

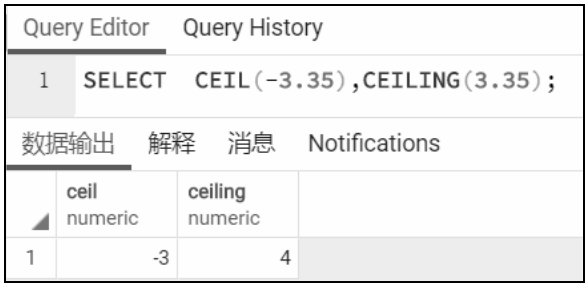

图 6-5　SQL 语句执行结果

-3.35 为负数，不小于-3.35 的最小整数为-3，因此返回值为-3；不小于 3.35 的最小整数为 4，因此返回值为 4。

FLOOR(x)返回不大于 x 的最大整数值，返回值转化为一个 BIGINT。

【例 6.6】使用 FLOOR 函数返回最大整数，输入语句如下：

```
SELECT FLOOR(-3.35), FLOOR(3.35);
```

语句执行后，结果如图 6-6 所示。

-3.35 为负数，不大于-3.35 的最大整数为-4，因此返回值为-4；不大于 3.35 的最大整数为 3，因此返回值为 3。

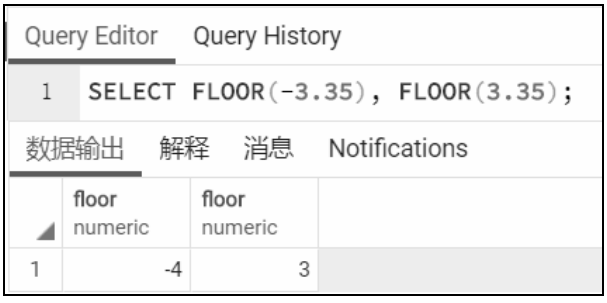

图 6-6　SQL 语句执行结果

6.2.4　四舍五入函数 ROUND(x)和 ROUND(x,y)

ROUND(x)返回最接近于参数 x 的整数，对 x 值进行四舍五入。

【例 6.7】使用 ROUND(x)函数对操作数进行四舍五入操作，输入语句如下：

```
SELECT ROUND(-1.14),ROUND(-1.67), ROUND(1.14),ROUND(1.66);
```

语句执行后，结果如图 6-7 所示。

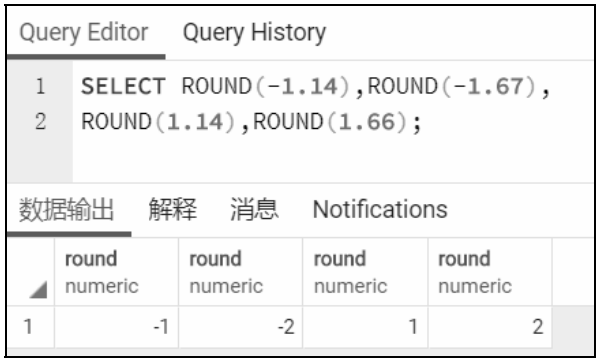

图 6-7　SQL 语句执行结果

可以看到，四舍五入处理之后，只保留了各个值的整数部分。

ROUND(x,y)返回最接近于参数 x 的数，其值保留到小数点后面 y 位，若 y 为负值，则将保留 x 值到小数点左边 y 位。

【例 6.8】使用 ROUND(x,y)函数对操作数进行四舍五入操作，结果保留小数点后面指定 y 位，输入语句如下：

```
SELECT ROUND(1.38, 1), ROUND(1.38, 0), ROUND(232.38, -1), ROUND (232.38,-2);
```

语句执行后，结果如图 6-8 所示。

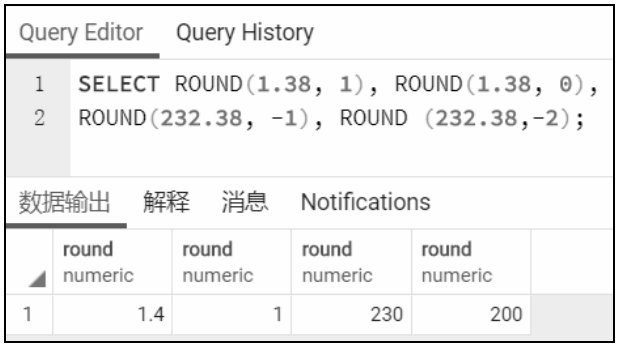

图 6-8　SQL 语句执行结果

ROUND(1.38, 1)保留小数点后面 1 位，四舍五入的结果为 1.4；ROUND(1.38, 0) 保留小数点后面 0 位，即返回四舍五入后的整数值； ROUND(23.38, -1)和 ROUND (232.38,-2)分别保留小数点左边 1 位和 2 位。

提　示

y 值为负数时，保留的小数点左边的相应位数直接保存为 0，同时进行四舍五入。

6.2.5　符号函数 SIGN(x)

SIGN(x)返回参数的符号，x 的值为负、零或正时返回结果依次为-1、0 或 1。

【例 6.9】使用 SIGN 函数返回参数的符号，输入语句如下：

```
SELECT SIGN(-21),SIGN(0), SIGN(21);
```

语句执行后，结果如图 6-9 所示。

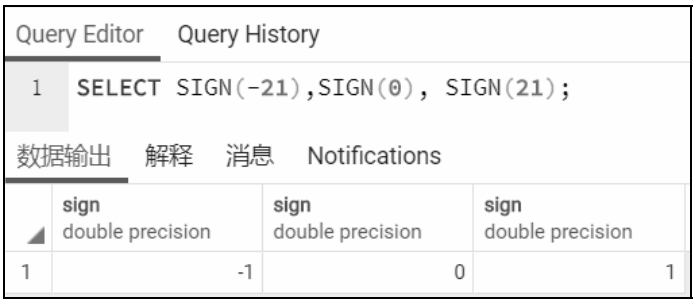

图 6-9　SQL 语句执行结果

SIGN(-21)返回-1；SIGN(0)返回 0；SIGN(21)返回 1。

6.2.6　幂运算函数 POW(x,y)、POWER(x,y)和 EXP(x)

POW(x,y)或者 POWER(x,y)函数返回 x 的 y 次乘方的结果值。

【例 6.10】使用 POW 和 POWER 函数进行乘方运算，输入语句如下：

```
SELECT POW(2,2), POWER(2,2),POW(2,-2), POWER(2,-2);
```

语句执行后，结果如图 6-10 所示。

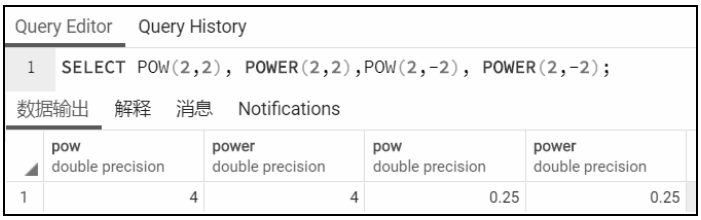

图 6-10　SQL 语句执行结果

可以看到，POW 和 POWER 的结果是相同的，POW(2,2)和 POWER(2,2)返回 2 的 2 次方，结果都是 4；POW(2,-2)和 POWER(2,-2)都返回 2 的-2 次方，结果为 4 的倒数，即 0.25。

EXP(x)返回 e 的 x 乘方后的值。

【例 6.11】使用 EXP 函数计算 e 的乘方，输入语句如下：

```
SELECT EXP(3),EXP(-3),EXP(0);
```

语句执行后，结果如图 6-11 所示。

图 6-11 SQL 语句执行结果

EXP(3)返回以 e 为底的 3 次方，结果为 20.0855369231877；EXP(-3)返回以 e 为底的-3 次方，结果为 0.0497870683678639；EXP(0)返回以 e 为底的 0 次方，结果为 1。

6.2.7 对数运算函数 LOG(x)

LOG(x)返回 x 的自然对数，x 相对于基数 e 的对数。对数定义域不能为负数，因此数组为负数时将会弹出错误信息。

【例 6.12】使用 LOG(x)函数计算自然对数，输入语句如下：

```
SELECT LOG(3);
```

语句执行后，结果如图 6-12 所示。

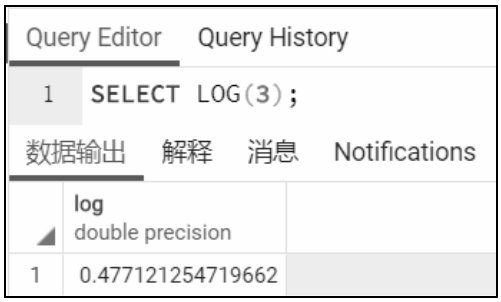

图 6-12 SQL 语句执行结果

6.2.8 角度与弧度相互转换的函数 RADIANS(x)和 DEGREES(x)

RADIANS(x)将参数 x 由角度转化为弧度。

【例 6.13】使用 RADIANS 将角度转换为弧度，输入语句如下：

```
SELECT RADIANS(90),RADIANS(180);
```

语句执行后，结果如图 6-13 所示。

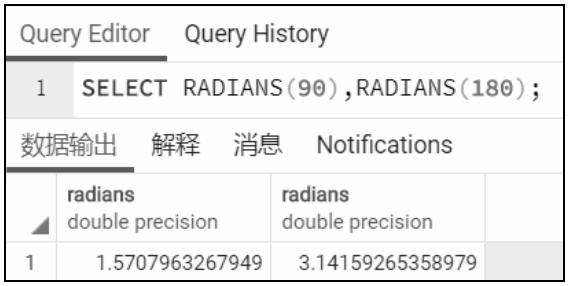

图 6-13　SQL 语句执行结果

DEGREES(x)将参数 x 由弧度转化为角度。

【例 6.14】使用 DEGREES 将弧度转换为角度，输入语句如下：

```
SELECT DEGREES(PI()), DEGREES(PI() / 2);
```

语句执行后，结果如图 6-14 所示。

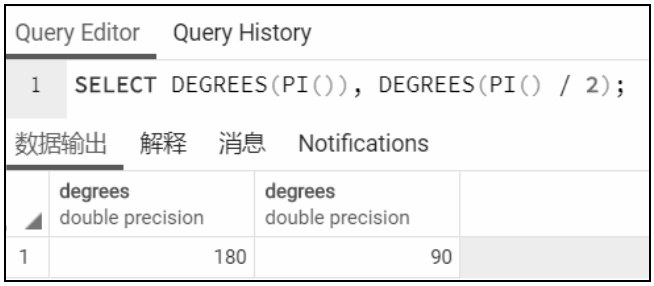

图 6-14　SQL 语句执行结果

6.2.9　正弦函数 SIN(x)和反正弦函数 ASIN(x)

SIN(x)返回 x 正弦，其中 x 为弧度值。

【例 6.15】使用 SIN 函数计算正弦值，输入语句如下：

```
SELECT SIN(1), ROUND(SIN(PI()));
```

语句执行后，结果如图 6-15 所示。

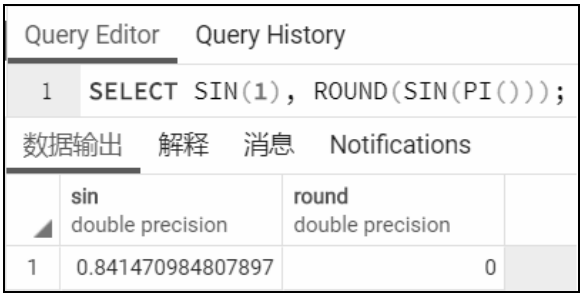

图 6-15　SQL 语句执行结果

ASIN(x)返回 x 的反正弦，即正弦为 x 的值。若 x 不在-1 到 1 的范围之内，则会弹出错误信息“输入超出范围”。

【例 6.16】使用 ASIN 函数计算反正弦值，输入语句如下：

```
SELECT ASIN(0.8414709848078965);
```

语句执行后，结果如图 6-16 所示。

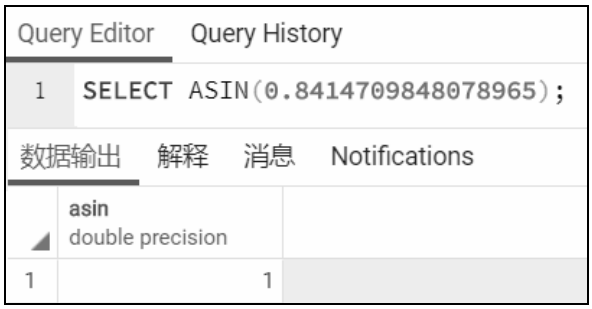

图 6-16 SQL 语句执行结果

由结果可以看到，函数 ASIN 和 SIN 互为反函数。

6.2.10 余弦函数 COS(x)和反余弦函数 ACOS(x)

COS(x)返回 x 的余弦，其中 x 为弧度值。

【例 6.17】使用 COS 函数计算余弦值，输入语句如下：

```
SELECT COS(0),COS(PI()),COS(1);
```

语句执行后结果如图 6-17 所示。

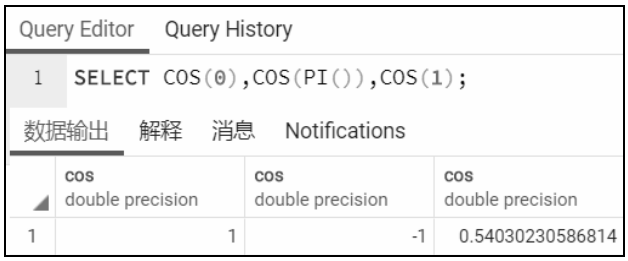

图 6-17 SQL 语句执行结果

由结果可以看到，COS(0)值为 1；COS(PI())值为-1；COS(1)值为 0.54030230586814。

ACOS(x)返回 x 的反余弦值，即余弦是 x 的值。若 x 不在-1 到 1 的范围之内，则会弹出错误信息。

【例 6.18】使用 ACOS 计算反余弦值，输入语句如下：

```
SELECT ACOS(1),ACOS(0), ROUND(ACOS(0.5403023058681398));
```

语句执行后，结果如图 6-18 所示。

由结果可以看到，函数 ACOS 和 COS 互为反函数。

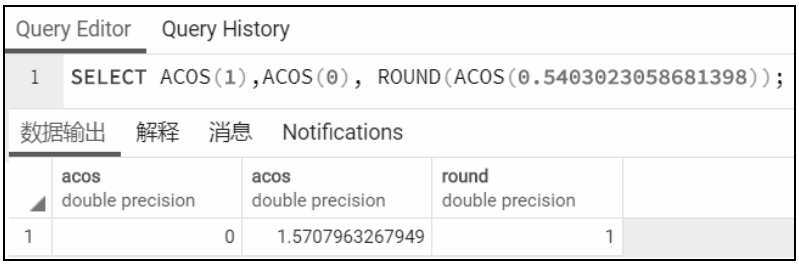

图 6-18　SQL 语句执行结果

6.2.11　正切函数、反正切函数和余切函数

TAN(x)返回 x 的正切，其中 x 为给定的弧度值。

【例 6.19】使用 TAN 函数计算正切值，输入语句如下：

```
SELECT TAN(0.3), ROUND(TAN(PI()/4));
```

语句执行后，结果如图 6-19 所示。

图 6-19　SQL 语句执行结果

ATAN(x)返回 x 的反正切，即正切为 x 的值。

【例 6.20】使用 ATAN 函数计算反正切值，输入语句如下：

```
SELECT ATAN(0.30933624960962325), ATAN(1);
```

语句执行后，结果如图 6-20 所示。

由结果可以看到，函数 ATAN 和 TAN 互为反函数。

COT(x)返回 x 的余切。

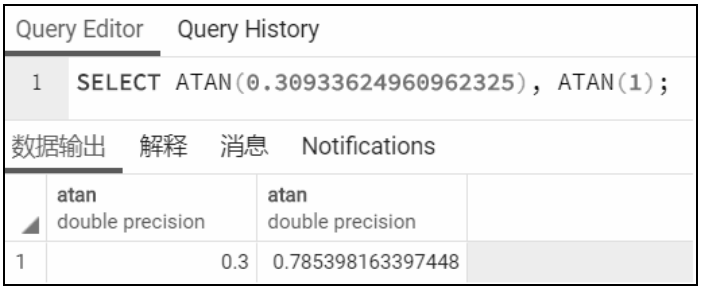

图 6-20 SQL 语句执行结果

【例 6.21】使用 COT()函数计算余切值，输入语句如下：

```
SELECT COT(0.3), 1/TAN(0.3),COT(PI() / 4);
```

语句执行后，结果如图 6-21 所示。

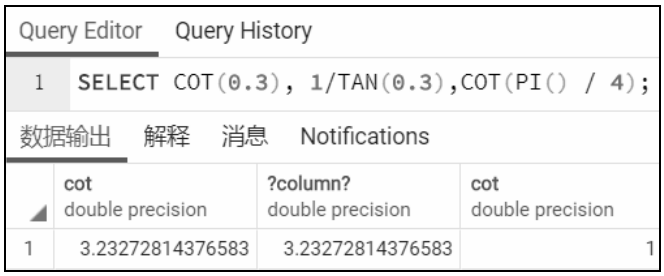

图 6-21 SQL 语句执行结果

由结果可以看到，函数 COT 和 TAN 互为倒函数。

6.3 字符串函数

字符串函数主要用来处理数据库中的字符串数据。PostgreSQL 中的字符串函数有计算字符串长度函数、字符串合并函数、字符串替换函数、字符串比较函数、查找指定字符串位置函数等。本节将介绍各种字符串函数的功能和用法。

6.3.1 计算字符串字符数的函数和字符串长度的函数

CHAR_LENGTH(str)返回值为字符串 str 所包含的字符个数。一个多字节字符算作一个单字符。

【例 6.21】使用 CHAR_LENGTH 函数计算字符串字符个数，输入语句如下：

```
SELECT CHAR_LENGTH('date'), CHAR_LENGTH('egg');
```

语句执行后结果如图 6-22 所示。

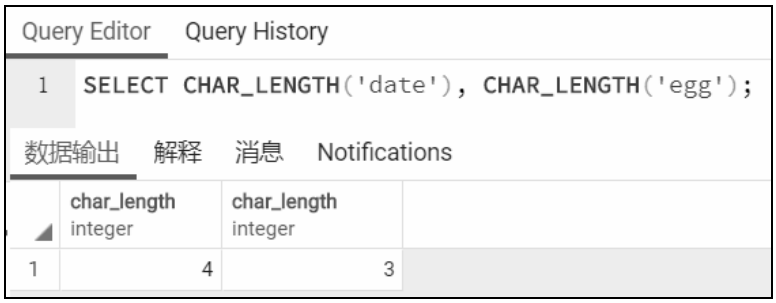

图 6-22　SQL 语句执行结果

LENGTH(str)返回值为字符串的字节长度，使用 utf8 编码字符集时，一个汉字是 3 个字节，一个数字或字母算一个字节。

【例 6.23】使用 LENGTH 函数计算字符串长度，输入语句如下：

```
SELECT LENGTH('date'), LENGTH('egg');
```

语句执行后，结果如图 6-23 所示。

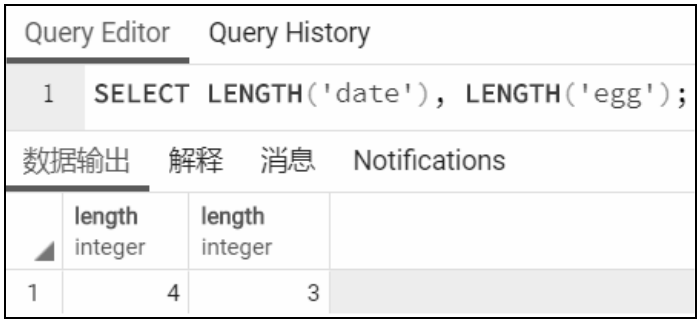

图 6-23　SQL 语句执行结果

可以看到，计算的结果与 CHAR_LENGTH 相同，因为英文字符的个数和所占的字节相同，一个字符占一个字节。

6.3.2　合并字符串函数 CONCAT(s1,s2,…)、CONCAT_WS(x,s1,s2,…)

CONCAT(s1,s2,…)的返回结果为连接参数产生的字符串。若有任何一个参数为 NULL，则返回值为 NULL。如果所有参数均为非二进制字符串，则结果为非二进制字符串。如果自变量中含有任一二进制字符串，则结果为一个二进制字符串。

【例 6.24】使用 CONCAT 函数连接字符串，输入语句如下：

```
SELECT CONCAT('PostgreSQL', '11.2'),CONCAT('Postgre',NULL, 'SQL');
```

语句执行后，结果如图 6-24 所示。

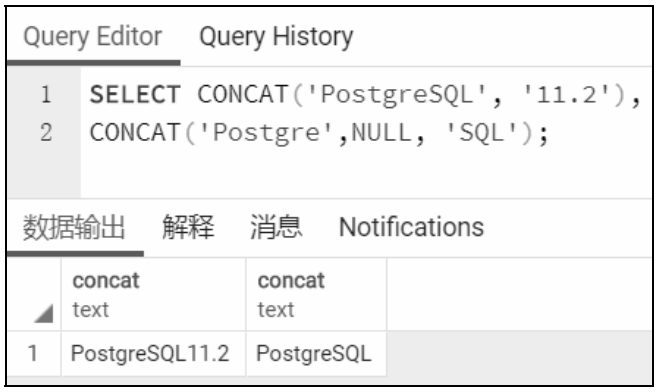

图 6-24 SQL 语句执行结果

CONCAT('PostgreSQL', '11.2')返回两个字符串连接后的字符串；CONCAT('Postgre',NULL, 'SQL')中有一个参数为 NULL，因此合并的时候忽略不计。

在 CONCAT_WS(x,s1,s2,…)中，CONCAT_WS 代表 CONCAT With Separator，是 CONCAT()的特殊形式。第一个参数 x 是其他参数的分隔符。分隔符的位置放在要连接的两个字符串之间。分隔符可以是一个字符串，也可以是其他参数。如果分隔符为 NULL，则结果为 NULL。函数会忽略任何分隔符参数后的 NULL 值。

【例 6.25】使用 CONCAT_WS 函数连接带分隔符的字符串，输入语句如下：

```
SELECT CONCAT_WS('-', '1st','2nd', '3rd'), CONCAT_WS('*', '1st', NULL, '3rd');
```

语句执行后，结果如图 6-25 所示。

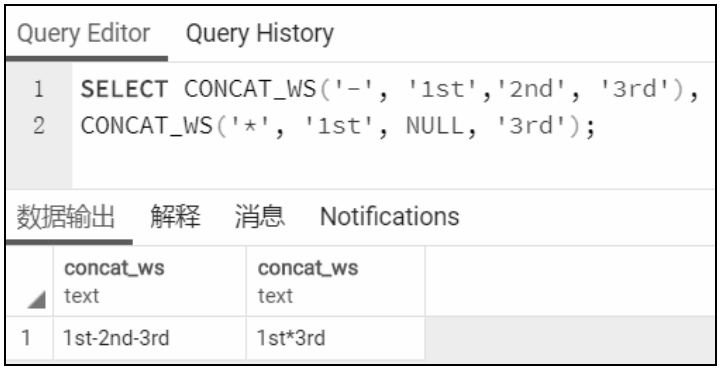

图 6-25 SQL 语句执行结果

CONCAT_WS('-', '1st','2nd', '3rd')使用分隔符‘-’将 3 个字符串连接成一个字符串，结果为“1st-2nd-3rd”；CONCAT_WS('*', '1st', NULL, '3rd')使用分隔符‘*’将两个字符串连接成一个字符串，同时忽略 NULL 值。

6.3.3 获取指定长度的字符串的函数 LEFT(s,n)和 RIGHT(s,n)

LEFT(s,n)返回字符串 s 最左边的 n 个字符。

【例 6.26】使用 LEFT 函数返回字符串中左边的字符，输入语句如下：

```
SELECT LEFT('football', 5);
```

语句执行后，结果如图 6-26 所示。

图 6-26　SQL 语句执行结果

函数返回字符串“football”从左边开始长度为 5 的子字符串，结果为“footb”。
RIGHT(s,n)返回字符串 s 最右边的 n 个字符。

【例 6.27】使用 RIGHT 函数返回字符串中右边的字符，输入语句如下：

```
SELECT RIGHT('football', 4);
```

语句执行后，结果如图 6-27 所示。

图 6-27　SQL 语句执行结果

函数返回字符串“football”从右边开始长度为 4 的子字符串，结果为“ball”。

6.3.4　填充字符串的函数 LPAD(s1,len,s2)和 RPAD(s1,len,s2)

LPAD(s1,len,s2)返回字符串 s1，其左边由字符串 s2 填充，填充长度为 len。假如 s1 的长度大于 len，则返回值被缩短至 len 字符。

【例 6.28】使用 LPAD 函数对字符串进行填充操作，输入语句如下：

```
SELECT LPAD('hello',4,'??'), LPAD('hello',10,'??');
```

语句执行后，结果如图 6-28 所示。

图 6-28 SQL 语句执行结果

字符串“hello”长度大于 4，不需要填充，因此 LPAD('hello',4,'??')只返回被缩短的长度为 4 的子串“hell”；字符串“hello”长度小于 10，LPAD('hello',10,'??')返回结果为“?????hello”，左侧填充‘？’，长度为 10。

RPAD(s1,len,s2)返回字符串 s1，其右边被字符串 s2 填补至 len 字符长度。假如字符串 s1 的长度大于 len，则返回值被缩短到与 len 字符相同的长度。

【例 6.29】使用 RPAD 函数对字符串进行填充操作，输入语句如下：

```
SELECT RPAD('hello',4,'?'), RPAD('hello',10,'?');
```

语句执行后，结果如图 6-29 所示。

图 6-29 SQL 语句执行结果

字符串“hello”长度大于 4，不需要填充，因此 RPAD('hello',4,'??')只返回被缩短的长度为 4 的子串“hell”；字符串“hello”长度小于 10，RPAD('hello',10,'??')返回结果为“hello?????”，右侧填充‘？’，长度为 10。

6.3.5 删除空格的函数 LTRIM(s)、RTRIM(s)和 TRIM(s)

LTRIM(s)返回字符串 s，字符串左侧空格字符被删除。

【例 6.30】使用 LTRIM 函数删除字符串左边的空格，输入语句如下：

```
SELECT '(    book    )',CONCAT('(',LTRIM('    book    '),')');
```

语句执行后，结果如图 6-30 所示。

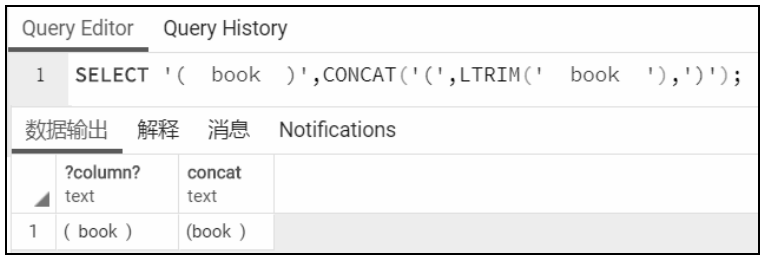

图 6-30　SQL 语句执行结果

LTRIM 只删除字符串左边的空格，而右边的空格不会被删除，“　book　”删除左边空格之后的结果为‘book　’。

RTRIM(s)返回字符串 s，字符串右侧空格字符被删除。

【例 6.31】使用 RTRIM 函数删除字符串右边的空格，输入语句如下：

```
SELECT '(　book　)',CONCAT('(', RTRIM ('　book　'),')');
```

语句执行后，结果如图 6-31 所示。

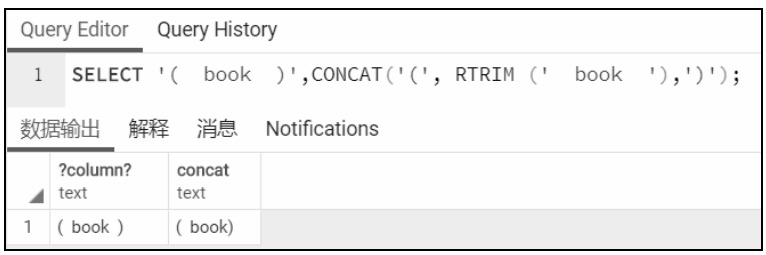

图 6-31　SQL 语句执行结果

RTRIM 只删除字符串右边的空格，左边的空格不会被删除，“　book　”删除右边空格之后的结果为“　book”。

TRIM(s)删除字符串 s 两侧的空格。

【例 6.32】使用 TRIM 函数删除指定字符串两端的空格，输入语句如下：

```
SELECT '(　book　)',CONCAT('(', TRIM('　book　'),')');
```

语句执行后，结果如图 6-32 所示。

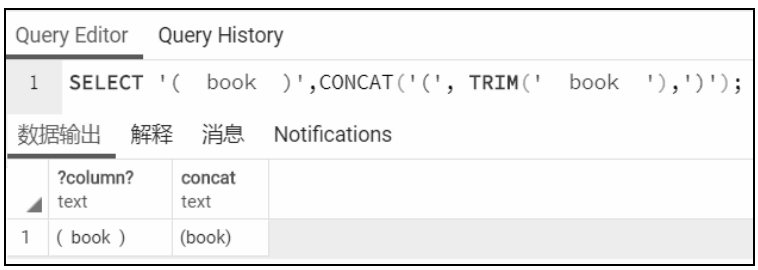

图 6-32　SQL 语句执行结果

可以看到，函数执行之后字符串“　book　”两边的空格都被删除，结果为“book”。

6.3.6　删除指定字符串的函数 TRIM(s1 FROM s)

TRIM(s1 FROM s)删除字符串 s 中两端所有的子字符串 s1。s1 为可选项，在未指定情况下，删除空格。

【例 6.33】使用 TRIM(s1 FROM s)函数删除字符串中两端指定的字符，输入语句如下：

```
SELECT TRIM('xy' FROM 'xyboxyokxyxy') ;
```

语句执行后，结果如图 6-33 所示。

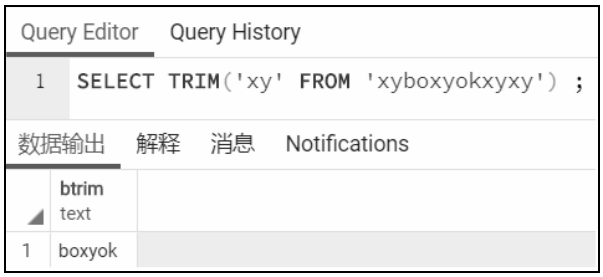

图 6-33　SQL 语句执行结果

删除字符串“xyboxyokxyxy”两端的重复字符串“xy”，而中间的“xy”并不删除，结果为“boxyok”。

6.3.7　重复生成字符串的函数 REPEAT(s,n)

REPEAT(s,n)返回一个由重复的字符串 s 组成的字符串，n 表示重复生成的次数。若 n<=0，则返回一个空字符串；若 s 或 n 为 NULL，则返回 NULL。

【例 6.34】使用 REPEAT 函数重复生成相同的字符串，输入语句如下：

```
SELECT REPEAT('PostgreSQL', 3);
```

语句执行后，结果如图 6-34 所示。

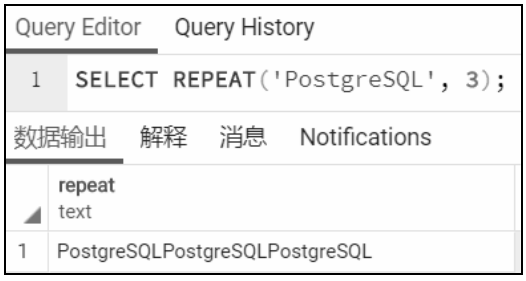

图 6-34　SQL 语句执行结果

REPEAT('PostgreSQL', 3)函数返回的字符串由 3 个重复的“PostgreSQL”字符串组成。

6.3.8 替换函数 REPLACE(s,s1,s2)

REPLACE(s,s1,s2)使用字符串 s2 替代字符串 s 中所有的字符串 s1。

【例 6.35】使用 REPLACE 函数进行字符串替代操作，输入语句如下：

```
SELECT REPLACE('xxx.PostgreSQL.com', 'x', 'w');
```

语句执行后，结果如图 6-35 所示。

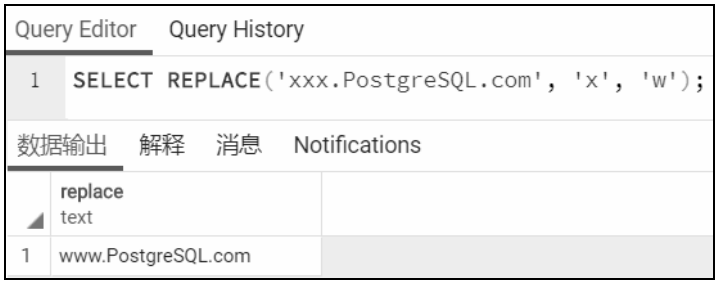

图 6-35 SQL 语句执行结果

REPLACE('xxx.PostgreSQL.com', 'x', 'w')将“xxx.PostgreSQL.com”字符串中的‘x’字符替换为‘w’字符，结果为“www.PostgreSQL.com”。

6.3.9 获取子串的函数 SUBSTRING(s,n,len)

SUBSTRING(s,n,len)表示从字符串 s 返回一个长度为 len 的子字符串，起始于位置 n。也可能对 n 使用一个负值，假若这样，则子字符串的位置起始于字符串结尾的 n 字符，即倒数第 n 个字符。

【例 6.36】使用 SUBSTRING 函数获取指定位置处的子字符串，输入语句如下：

```
SELECT SUBSTRING('breakfast',5) AS col1,
SUBSTRING('breakfast',5,3) AS col2,
 SUBSTRING('lunch', -3) AS col3;
```

语句执行后，结果如图 6-36 所示。

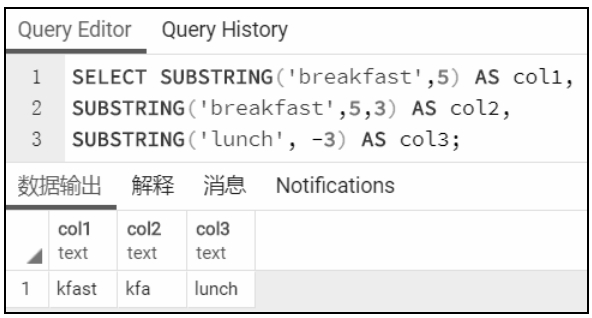

图 6-36 SQL 语句执行结果

SUBSTRING('breakfast',5)返回从第 5 个位置开始到字符串结尾的子字符串，结果为“kfast”；SUBSTRING('breakfast',5,3)返回从第 5 个位置开始长度为 3 的子字符串，结果为“kfa”；SUBSTRING('lunch', -3)返回整个字符串。

提 示

如果对 len 使用的是一个小于 1 的值，则结果始终为整个字符串。

6.3.10 匹配子串开始位置的函数 POSITION(str1 IN str)

POSITION(str1 IN str)函数的作用是返回子字符串 str1 在字符串 str 中的开始位置。

【例 6.37】使用 POSITION 函数查找字符串中指定子字符串的开始位置，输入语句如下：

```
SELECT POSITION('ball'IN 'football');
```

语句执行后，结果如图 6-37 所示。

图 6-37 SQL 语句执行结果

子字符串“ball”在字符串“football”中从第 5 个字母位置开始，因此函数返回结果为 5。

6.3.11 字符串逆序的函数 REVERSE(s)

REVERSE(s)将字符串 s 反转，返回的字符串顺序和 s 字符串顺序相反。

【例 6.38】使用 REVERSE 函数反转字符串，输入语句如下：

```
SELECT REVERSE('abc');
```

语句执行后，结果如图 6-38 所示。

图 6-38 SQL 语句执行结果

由结果可以看到，字符串“abc”经过 REVERSE 函数处理之后所有字符串顺序被反转，结果为“cba”。

6.4 日期和时间函数

日期和时间函数主要用来处理日期和时间值，一般的日期函数除了使用 DATE 类型的参数外，也可以使用 DATETIME 或者 TIMESTAMP 类型的参数，但会忽略这些值的时间部分。相同的，以 TIME 类型值为参数的函数，可以接受 TIMESTAMP 类型的参数，但会忽略日期部分。许多日期函数可以同时接受数字和字符串类型的两种参数。本节将介绍各种日期和时间函数的功能和用法。

6.4.1 获取当前日期的函数和获取当前时间的函数

CURRENT_DATE 函数的作用是将当前日期按照‘YYYY-MM-DD’格式的值返回，具体格式根据函数用在字符串或是数字语境中而定。

【例 6.39】使用日期函数获取系统当前日期，输入语句如下：

```
SELECT CURRENT_DATE;
```

语句执行后，结果如图 6-39 所示。

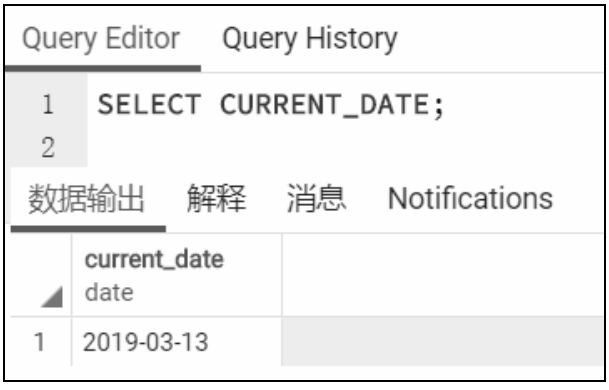

图 6-39 SQL 语句执行结果

可以看到，返回了相同的系统当前日期。

CURRENT_TIME 函数的作用是将当前时间以‘HH:MM:SS’的格式返回，具体格式根据函数用在字符串或是数字语境中而定。

【例 6.40】使用时间函数获取系统当前日期，输入语句如下：

```
SELECT CURRENT_TIME;
```

语句执行后，结果如图 6-40 所示。可以看到，函数返回了系统当前时间。

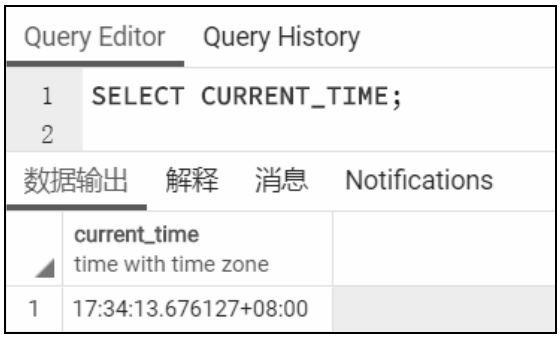

图 6-40 SQL 语句执行结果

LOCALTIME 函数的作用是将当前时间以‘HH:MM:SS’的格式返回，唯一和 CURRENT_TIME 函数不同的是返回时不带时区值。

【例 6.41】使用时间函数获取系统当前日期，输入语句如下：

```
SELECT LOCALTIME;
```

语句执行后，结果如图 6-41 所示。可以看到，函数返回了系统当前时间，但是不带时区。

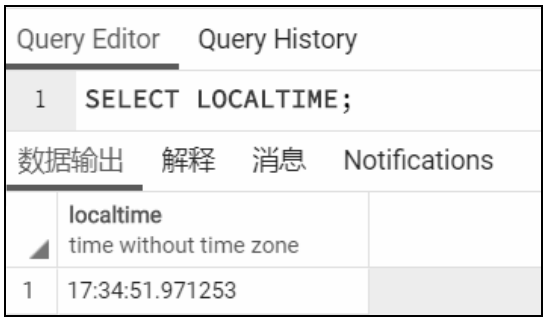

图 6-41 SQL 语句执行结果

6.4.2 获取当前日期和时间的函数

CURRENT_TIMESTAMP、LOCALTIMESTAMP 和 NOW()3 个函数的作用相同，返回当前日期和时间值，格式为‘YYYY-MM-DD HH:MM:SS’或 YYYYMMDDHHMMSS，具体格式根据函数是否用在字符串或数字语境中而定。

【例 6.42】使用日期时间函数获取当前系统日期和时间，输入语句如下：

```
SELECT CURRENT_TIMESTAMP,LOCALTIMESTAMP,NOW();
```

语句执行后，结果如图 6-42 所示。

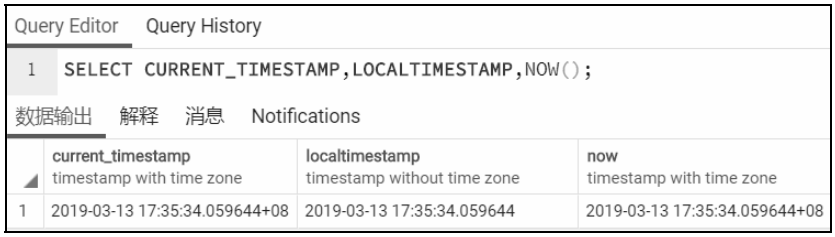

图 6-42　SQL 语句执行结果

可以看到，3 个函数返回的日期和时间是相同的。唯一不同的是，LOCALTIMESTAMP 函数的返回值不带时区。

6.4.3　获取日期的指定值的函数 EXTRACT(type FROM d)

EXTRACT(type FROM date)函数从日期中提取其部分，而不是执行日期运算。

【例 6.43】使用 EXTRACT 函数从日期中提取一个月中的第几天，输入语句如下：

```
SELECT EXTRACT(DAY FROM TIMESTAMP '2017-09-10 10:18:40');
```

语句执行后，结果如图 6-43 所示。

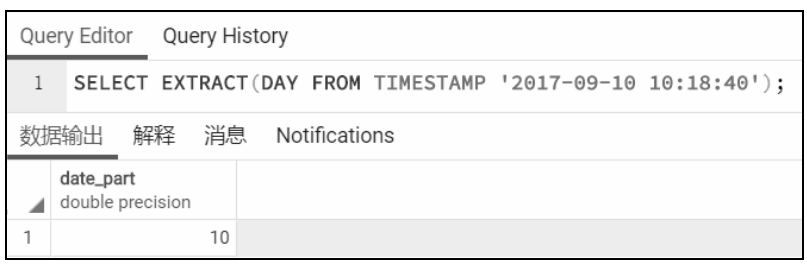

图 6-43　SQL 语句执行结果

【例 6.44】使用 EXTRACT 函数从日期中提取月份，输入语句如下：

```
SELECT EXTRACT(MONTH FROM TIMESTAMP '2019-09-10 10:18:40');
```

语句执行后，结果如图 6-44 所示。

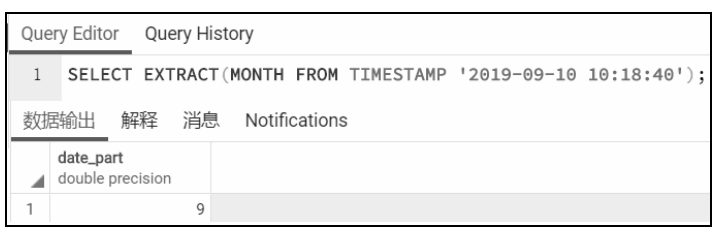

图 6-44　SQL 语句执行结果

【例 6.45】使用 EXTRACT 函数从日期中提取年份，输入语句如下：

```
SELECT EXTRACT(YEAR FROM TIMESTAMP '2019-09-10 10:18:40');
```

语句执行后，结果如图 6-45 所示。

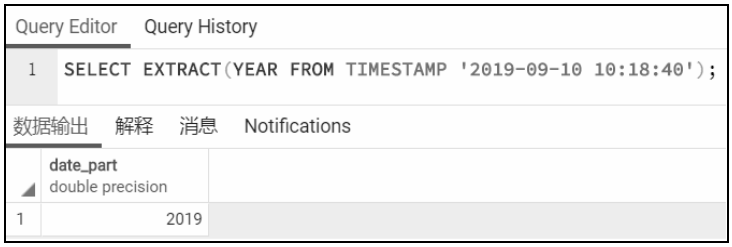

图 6-45 SQL 语句执行结果

【例 6.46】使用 EXTRACT 函数查询指定日期是一年中的第几天，输入语句如下：

```
SELECT EXTRACT(DOY FROM TIMESTAMP '2019-09-10 10:18:40');
```

语句执行后，结果如图 6-46 所示。

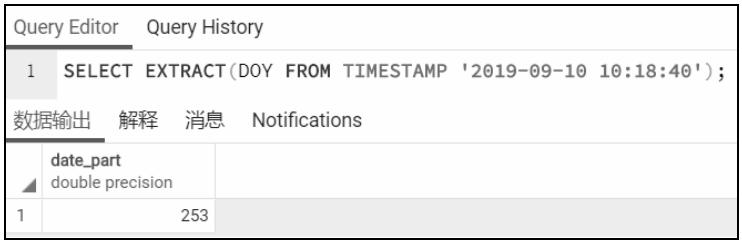

图 6-46 SQL 语句执行结果

【例 6.47】使用 EXTRACT 函数查询指定日期是一周中的星期几，输入语句如下：

```
SELECT EXTRACT(DOW FROM TIMESTAMP '2019-09-10 10:18:40');
```

语句执行后，结果如图 6-47 所示。

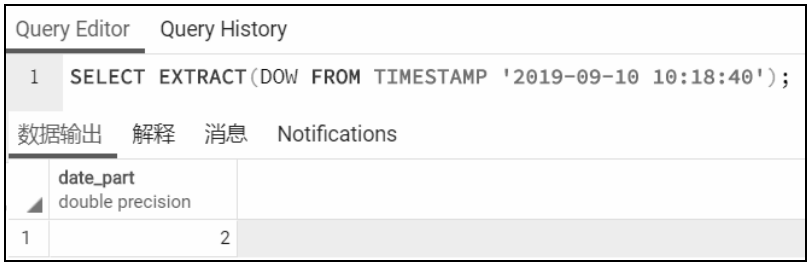

图 6-47 SQL 语句执行结果

从结果可以看出，2019-09-10 是星期日。需要读者注意的是，此函数的星期编号为 0~6，星期日的返回结果为 0。

【例 6.48】使用 EXTRACT 函数查询指定日期是该年的第几季度（1~4），输入语句如下：

```
SELECT EXTRACT(QUARTER FROM TIMESTAMP '2019-09-10 10:18:40');
```

语句执行后，结果如图 6-48 所示。

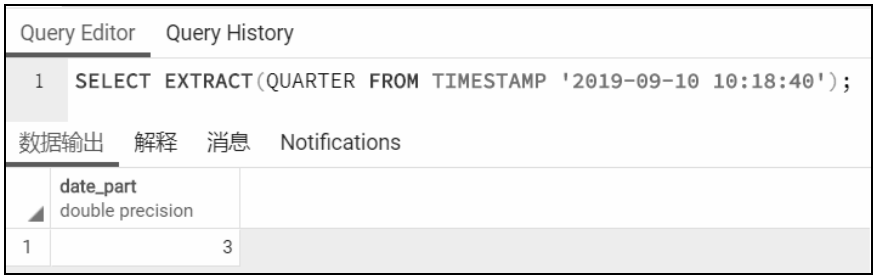

图 6-48　SQL 语句执行结果

从结果可以看出，2019-09-10 是该年中的第 3 季度。

6.4.4　日期和时间的运算操作

日期和时间之间可以有加、减、乘、除运算操作。本节主要讲述这些操作的方法和技巧。

【例 6.49】 计算指定日期加上间隔天数后的结果，输入语句如下：

```
SELECT DATE '2019-09-28' + integer '10';
```

语句执行后，结果如图 6-49 所示。

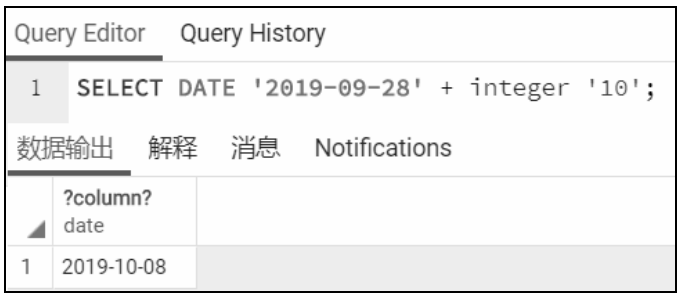

图 6-49　SQL 语句执行结果

【例 6.50】 计算指定日期加上间隔小时后的结果，输入语句如下：

```
SELECT DATE '2019-09-28' + interval '3 hour';
```

语句执行后，结果如图 6-50 所示。

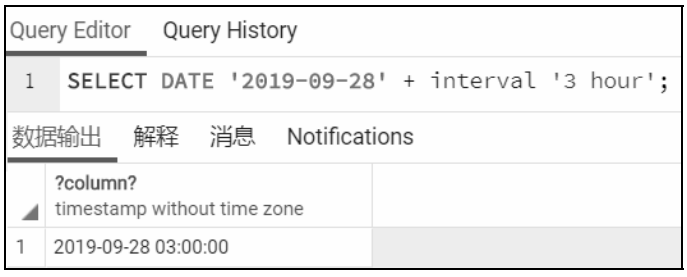

图 6-50　SQL 语句执行结果

【例 6.51】 计算指定日期加上指定时间后的结果，输入语句如下：

```
SELECT DATE '2019-09-28' + time '06:00';
```

语句执行后，结果如图 6-51 所示。

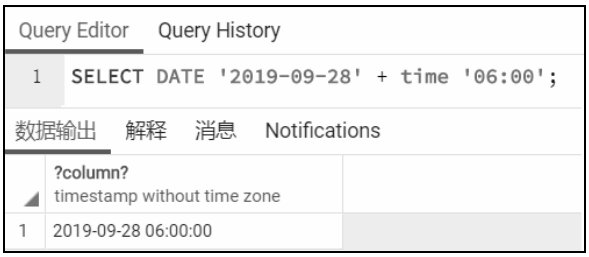

图 6-51 SQL 语句执行结果

【例 6.52】 计算指定日期和时间加上间隔时间后的结果，输入语句如下：

```
SELECT TIMESTAMP '2019-09-28 02:00:00' + interval '10 hours';
```

语句执行后，结果如图 6-52 所示。

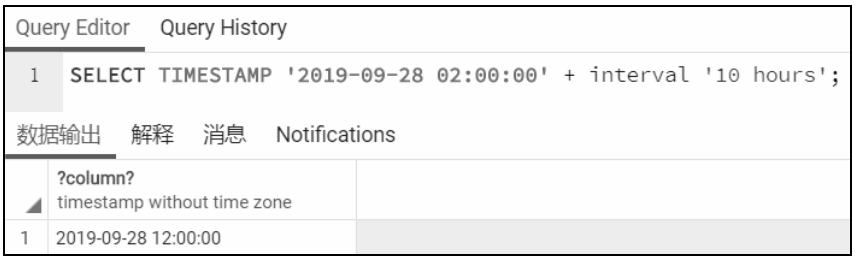

图 6-52 SQL 语句执行结果

【例 6.53】 计算指定日期之间的间隔天数，输入语句如下：

```
SELECT date '2019-11-01' - date '2019-09-10';
```

语句执行后，结果如图 6-53 所示。

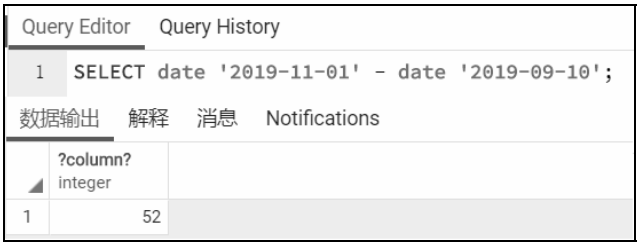

图 6-53 SQL 语句执行结果

【例 6.54】 计算指定日期减去间隔天数后的结果，输入语句如下：

```
SELECT DATE '2019-09-28' - integer '10';
```

语句执行后，结果如图 6-54 所示。

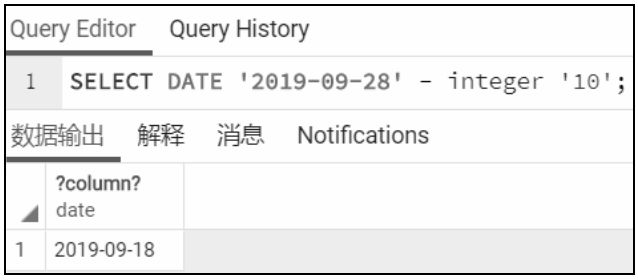

图 6-54　SQL 语句执行结果

【例 6.55】 计算整数与天数相乘的结果，输入语句如下：

```
SELECT 15 * interval '2 day';
```

语句执行后，结果如图 6-55 所示。

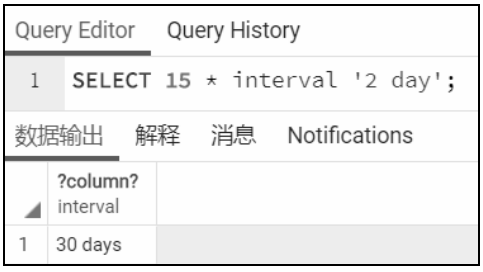

图 6-55　SQL 语句执行结果

【例 6.56】 计算整数与秒数相乘的结果，输入语句如下：

```
SELECT 50 * interval '2 second';
```

语句执行后，结果如图 6-56 所示。

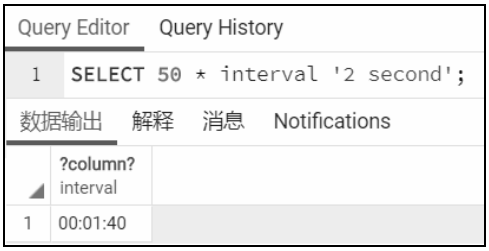

图 6-56　SQL 语句执行结果

【例 6.57】 计算小时数与整数相除的结果，输入语句如下：

```
SELECT interval '1 hour' / integer   '2';
```

语句执行后，结果如图 6-57 所示。

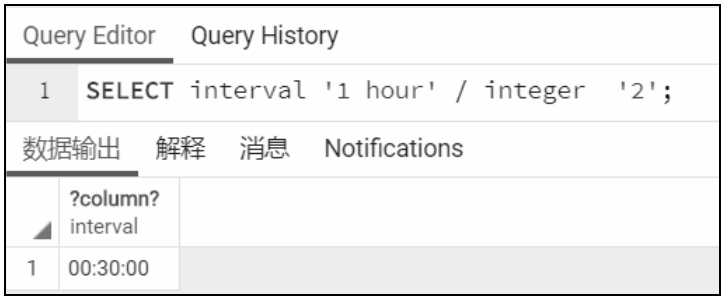

图 6-57 SQL 语句执行结果

6.5 条件判断函数

条件判断函数亦称为控制流程函数，根据满足的条件而执行相应的流程。在 PostgreSQL 中，进行条件判断的函数为 CASE。

1. CASE expr WHEN v1 THEN r1 [WHEN v2 THEN r2] [ELSE rn] END

该函数表示，如果 expr 值等于某个 vn，则返回对应位置 THEN 后面的结果；如果与所有值都不相等，则返回 ELSE 后面的 rn。

【例 6.58】使用 CASE value WHEN 语句执行分支操作，输入语句如下：

```
SELECT CASE 2 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END;
```

语句执行后，结果如图 6-58 所示。

图 6-58 SQL 语句执行结果

CASE 后面的值为 2，与第二条分支语句 WHEN 后面的值相等，因此返回结果为“two”。

2. CASE WHEN v1 THEN r1 [WHEN v2 THEN r2] ELSE rn] END

该函数表示，某个 vn 值为 TRUE 时，返回对应位置 THEN 后面的结果；如果所有值都不为 TRUE，则返回 ELSE 后的 rn。

【例 6.59】使用 CASE WHEN 语句执行分支操作，输入语句如下：

```
SELECT CASE WHEN 1<0 THEN 'true' ELSE 'false' END;
```

语句执行后，结果如图 6-59 所示。

图 6-59　SQL 语句执行结果

1<0 的结果为 FALSE，因此函数返回值为 ELSE 后面的“false”。

提　示

一个 CASE 表达式的默认返回值类型是任何返回值的相容集合类型，具体情况视其所在语境而定。用在字符串语境中，返回结果为字符串；用在数字语境中，返回结果为十进制值、实值或整数值。

6.6　系统信息函数

PostgreSQL 中的系统信息有数据库的版本号、当前用户名和链接数、系统字符集、最后一个自动生成的 ID 值等。本节将介绍各个函数的使用方法。

6.6.1　获取 PostgreSQL 版本号

VERSION()返回指示 PostgreSQL 服务器版本的字符串。这个字符串使用 utf8 字符集。

【例 6.60】查看当前 PostgreSQL 版本号，输入语句如下：

```
SELECT VERSION();
```

语句执行后，结果如图 6-60 所示。

图 6-60　SQL 语句执行结果

6.6.2 获取用户名的函数

USER 和 CURRENT_USER 函数返回当前被 PostgreSQL 服务器验证的用户名。这个值符合确定当前登录用户存取权限的 PostgreSQL 账户。一般情况下，这两个函数的返回值是相同的。

【例 6.61】获取当前登录用户名称，输入语句如下：

```
SELECT USER, CURRENT_USER;
```

语句执行后，结果如图 6-61 所示。

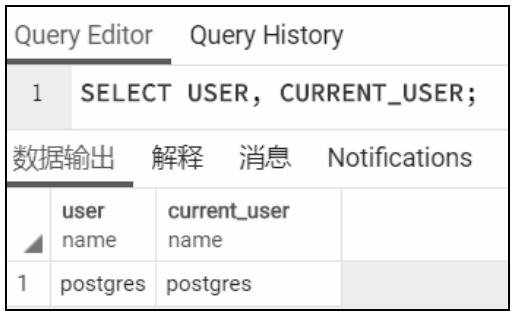

图 6-61 SQL 语句执行结果

返回结果值指示了当前账户连接服务器时的用户名，postgres 为当前登录的用户名。

6.7 加密函数

加密函数主要用来对数据进行加密和界面处理，以保证某些重要数据不被别人获取。这些函数在保证数据库安全时非常有用。本章将介绍各种加密和解密函数的作用和使用方法。

6.7.1 加密函数 MD5(str)

MD5(str)为字符串算出一个 MD5 128 比特检查和。该值以 32 位十六进制数字的二进制字符串的形式返回，若参数为 NULL 则会返回 NULL。

【例 6.62】使用 MD5 函数加密字符串，输入语句如下：

```
SELECT MD5 ('mypwd');
```

语句执行后，结果如图 6-62 所示。

可以看到，“mypwd”经 MD5 加密后的结果为 318bcb4be908d0da6448a0db76908d78

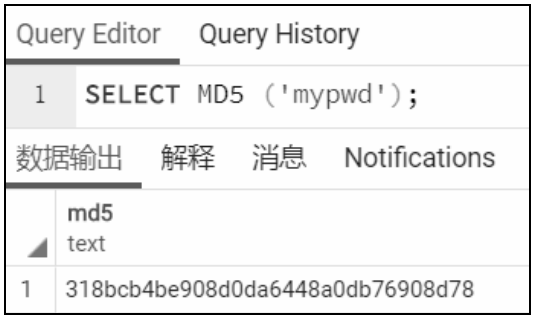

图 6-62 SQL 语句执行结果

6.7.2 加密函数 ENCODE(str,pswd_str)

ENCODE(str,pswd_str)使用 pswd_str 作为加密编码，加密 str，常见的加密编码包括 base64、hex 和 escape。

【例 6.63】使用 ENCODE 加密字符串，输入语句如下：

```
SELECT ENCODE('secret','hex'), LENGTH(ENCODE('secret','hex'));
```

语句执行后，结果如图 6-63 所示。可以看到，加密后的长度为 12。

图 6-63 SQL 语句执行结果

6.7.3 解密函数 DECODE(crypt_str,pswd_str)

DECODE(crypt_str,pswd_str)将 pswd_str 作为密码，解密加密字符串 crypt_str。crypt_str 是由 ENCODE()返回的字符串。

【例 6.64】使用 DECODE 函数解密被 ENCODE 加密的字符串，输入语句如下：

```
SELECT DECODE(ENCODE('secret','hex'),'hex');
```

语句执行后，结果如图 6-64 所示。

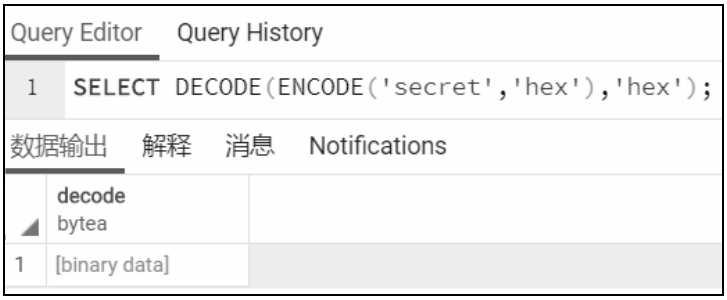

图 6-64 SQL 语句执行结果

可以看到，使用相同解密字符串进行解密之后的结果正好为 ENCODE 函数中被加密的字符串。DECODE 函数和 ENCODE 函数互为反函数。

6.8 改变数据类型的函数

CAST(x , AS type)函数将一个类型的值转换为另一个类型的值。

【例 6.65】使用 CAST 函数进行数据类型的转换，SQL 语句如下：

```
SELECT CAST(100 AS CHAR(2));
```

语句执行后，结果如图 6-65 所示。

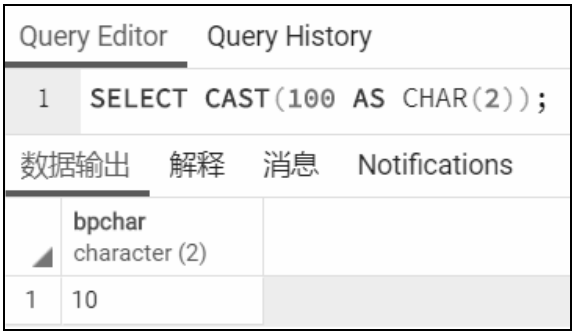

图 6-65 SQL 语句执行结果

可以看到，CAST(100 AS CHAR(2))将整数数据 100 转换为带有 2 个显示宽度的字符串类型，结果为‘10’。

6.9 综合案例——PostgreSQL 函数的使用

本章为读者介绍了大量的 PostgreSQL 函数，包括数学函数、字符串函数、日期和时间函数、条件判断函数、系统函数、加密函数以及其他函数。读者应该在实践过程中深入了解、掌握这些函数。不同版本的 PostgreSQL 之间的函数可能会有微小的差别，使用时需要查阅对应版本的参考手册，但大部分函数功能在不同版本的 PostgreSQL 之间是一致的。接下来，将给

出一个使用各种 PostgreSQL 函数的综合案例。

1. 案例目的

使用各种函数操作数据，掌握各种函数的作用和使用方法。

2. 案例操作过程

步骤 01 使用 sin()、cos()、tan()、cot()函数计算三角函数值，并将计算结果值转换成整数值。

在 PostgreSQL 中，三角函数计算出来的值并不一定是整数值，需要使用数学函数将其转换为整数。可以使用的数学函数有 ROUND()、FLOOR 等。执行过程如下:

```
SELECT PI(), sin(PI()/2),cos(PI()), ROUND(tan(PI()/4)), FLOOR(cot(PI()/4));
```

语句执行后结果如图 6-66 所示。

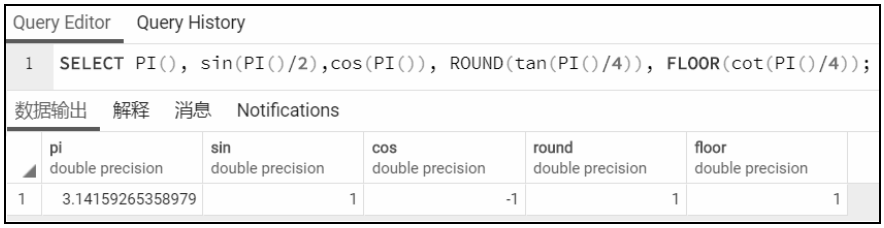

图 6-66 SQL 语句执行结果

步骤 02 创建表，并使用字符串和日期函数对字段值进行操作。

（1）创建表 member，其中包含 5 个字段，分别为 INT 类型的 m_id 字段、VARCHAR 类型的 m_FN 字段、VARCHAR 类型的 m_LN 字段，DATA 类型的 m_birth 字段和 VARCHAR 类型的 m_info 字段。

（2）插入一条记录，m_id 值为“10110”，m_FN 值为“Halen”，m_LN 值为“Park”，m_birth 值为 1970-06-29，m_info 值为“GoodMan”。

（3）返回第一条记录中的人的全名，将 m_info 字段值转换成小写字母。将 m_info 的值反向输出。

（4）计算第一条记录中人的年龄，并计算 m_birth 字段中的日期值在那一年中的位置，按照“Saturday October 4th 1997”格式输出日期时间值。

（5）插入一条新的记录，m_FN 值为“Samuel”，m_LN 值为“Green”，m_birth 值为系统当前时间，m_info 为空。使用 LAST_INSERT_ID()查看最后插入的 ID 值。

操作过程如下：

（1）创建表 member，输入语句如下：

```
CREATE TABLE member
(
```

```
  m_id      INT PRIMARY KEY,
  m_FN      VARCHAR(100),
  m_LN      VARCHAR(100),
  m_birth   DATE,
  m_info    VARCHAR(255) NULL
);
```

（2）插入一条记录，输入语句如下：

```
INSERT INTO member VALUES (10110, 'Halen ', 'Park', '1970-06-29', 'GoodMan ');
```

使用 SELECT 语句查看插入结果：

```
SELECT * FROM member;
```

语句执行后结果如图 6-67 所示。

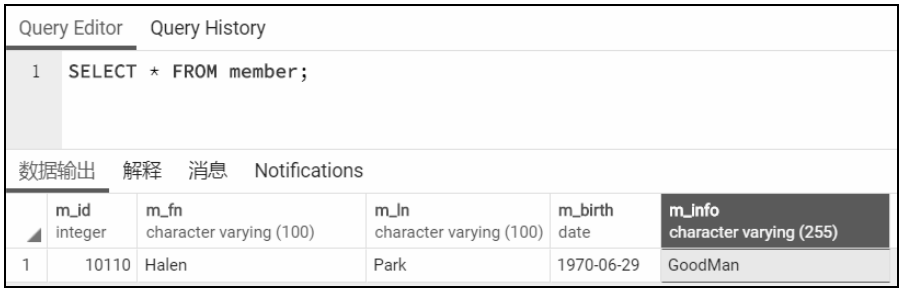

图 6-67　SQL 语句执行结果

（3）返回 m_FN 的长度，返回记录中人的全名，将 m_info 字段值转换成小写字母，将 m_info 的值反向输出。

```
SELECT LENGTH(m_FN), CONCAT(m_FN, m_LN),
LOWER(m_info), REVERSE(m_info) FROM member;
```

语句执行后，结果如图 6-68 所示。

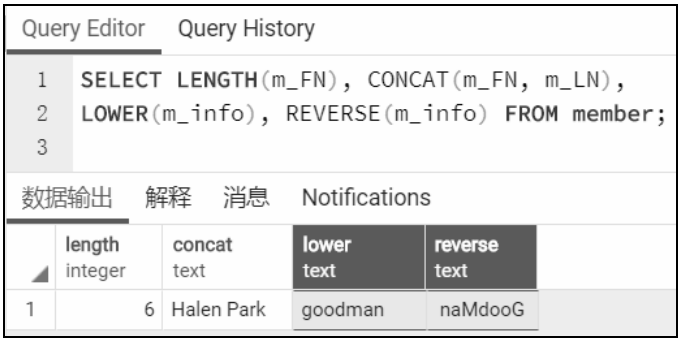

图 6-68　SQL 语句执行结果

（4）计算第一条记录中人的年龄，并计算 m_birth 字段中的值在那一年中的位置。

```
SELECT EXTRACT(YEAR FROM CURRENT_DATE)-EXTRACT
(YEAR FROM m_birth)AS age,
EXTRACT(DOY FROM m_birth)AS days FROM member;
```

语句执行后，结果如图 6-69 所示。

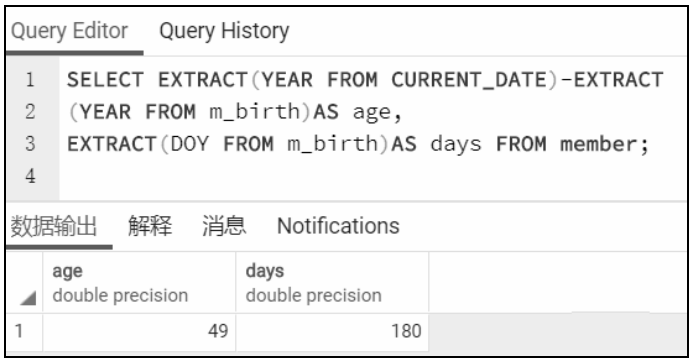

图 6-69　SQL 语句执行结果

（5）插入一条新的记录，m_FN 值为“Samuel”，m_LN 值为“Green”，m_birth 值为系统当前时间，m_info 为空。使用 LAST_INSERT_ID()查看最后插入的 ID 值。

```
INSERT INTO member VALUES (10112, 'Samuel', 'Green', NOW(),NULL);
```

使用 SELECT 语句查看插入结果：

```
SELECT * FROM member;
```

语句执行后，结果如图 6-70 所示。可以看到，表中现在有两条记录。

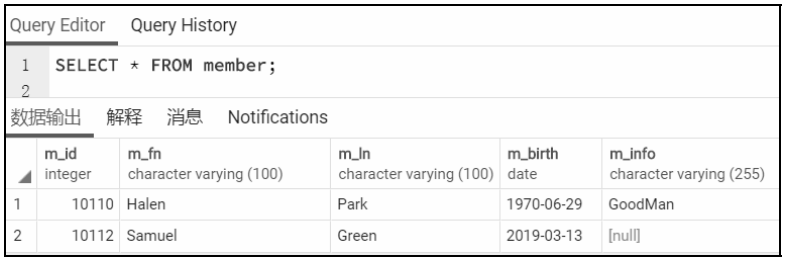

图 6-70　SQL 语句执行结果

步骤 04　使用 CASE 进行条件判断：如果 m_birth 小于 2000 年，显示“old”；如果 m_birth 大于 2000 年，则显示“young”。输入语句如下：

```
SELECT m_birth, CASE WHEN EXTRACT (YEAR FROM m_birth) < 2000    THEN    'old'
WHEN EXTRACT (YEAR FROM m_birth) > 2000 THEN    'young'
ELSE 'not born' END AS status FROM member;
```

语句执行后，结果如图 6-71 所示。

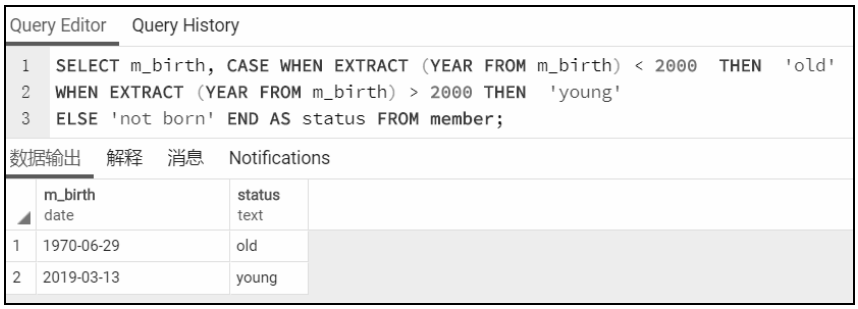

图 6-71 SQL 语句执行结果

6.10 常见问题及解答

疑问 1：如何从日期时间值中获取年、月、日等部分日期或时间值？

在 PostgreSQL 中，日期时间值以字符串形式存储在数据表中，因此可以使用字符串函数分别截取对日期时间值的不同部分，例如某个名称为 dt 的字段有值“2010-10-01 12:00:30”，如果只需要获得年值，可以输入 LEFT(dt, 4)，这样就获得了字符串左边开始长度为 4 的子字符串，即 YEAR 部分的值；如果要获取月份值，可以输入 MID(dt,6,2)，从字符串第 6 个字符开始，长度为 2 的子字符串正好为 dt 中的月份值。同理，读者可以根据其他日期和时间的位置，计算并获取相应的值。

疑问 2：如何计算年龄？

年龄是通过当前年份减去出生年份来计算的。例如，在本章的综合案例中的 m_birth 字段包含了年、月和日，这就需要从 m_birth 字段中获取年份。PostgreSQL 中提供了 EXTRACT 函数获取年份，例如 EXTRACT(YEAR FROM m_birth)的返回结果是 1970。然后通过 EXTRACT(YEAR FROM CURRENT_DATE)获取当前的年份，两者相减，即可得到年龄。

6.11 经典习题

1．使用数学函数进行如下运算：

（1）计算 18 除以 5 的商和余数。
（2）将弧度值 PI()/4 转换为角度值。
（3）计算 9 的 4 次方值。
（4）保留浮点值 3.14159 小数点后面 2 位。

2．使用字符串函数进行如下运算：

（1）分别计算字符串“Hello World!”和“University”的长度。
（2）从字符串“Nice to meet you!”中获取子字符串“meet”。

（3）重复输出 3 次字符串“Cheer!”。

（4）将字符串“voodoo”逆序输出。

（5）4 个字符串“PostgreSQL”“not”“is”“great”按顺序排列，从中选择 1、3 和 4 位置处的字符串组成新的字符串。

3．使用日期和时间函数进行如下运算：

（1）计算当前日期是一年的第几周。

（2）计算当前日期是一周中的第几个工作日。

（3）计算“1929-02-14”与当前日期之间相差的年份。

（4）从当前日期时间值中获取时间值，并将其转换为秒值。