第 5 章

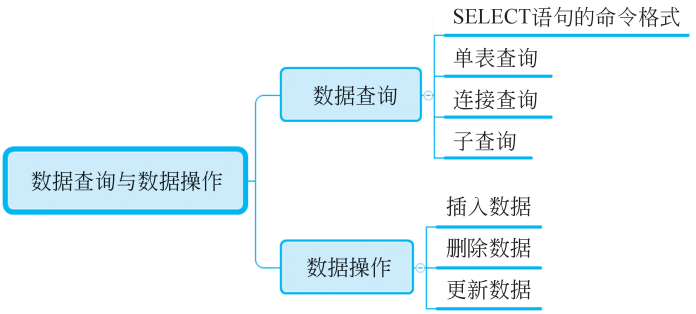
数据查询与数据操作

学习目标

（1）熟练掌握数据查询语句。

（2）熟练掌握数据操作语句。

思维导图



将数据存储到数据库之后，用户就可以对数据库中的数据进行数据查询和数据操作，其中，数据操作包括插入数据、删除数据和更新数据。本章主要介绍如何使用 SQL 语言进行数据查询和数据操作。

5.1 数据查询

5.1.1 SELECT 语句的命令格式

数据库查询是数据库的核心操作。SQL 语言提供了 SELECT 语句进行数据库的查询，该语句具有灵活的使用方式和丰富的功能。其一般格式如下：

```
[ WITH [ RECURSIVE ] with_query [, …] ]
SELECT [ ALL | DISTINCT[ ON ( expression [, …] ) ] ] select_list
                                                        /* 需要哪些列 */
[ FROM from_item [, …] ]                                /* 来自哪些表 */
[ WHERE search_condition ]                             /* 根据什么条件筛选元组 */
[ GROUP BY group_by_expression [ HAVING search_condition] ]   /* 按什么分组 */
[ ORDER BY order_expression [ ASC | DESC ] ] ]          /* 查询结果按什么排序 */
[ LIMIT { [offset,] count | ALL } ]
[ OFFSET start [ ROW | ROWS ] ]
```

```
[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
[ {FOR { UPDATE | SHARE } [ OF table_name [, …] ] [ NOWAIT ]} […] ];
```

下面对一些常用关键字语法进行简单介绍，详细的语法说明请参见 GaussDB 的相关文档。

(1) WITH 子句用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。它常用于复杂查询或递归查询。

(2) SELECT 子句描述查询结果中要返回的列及其限定，其中，ALL 是默认值，说明不去掉重复元组，DISTINCT 说明要去掉重复元组，DISTINCT ON 按指定列/表达式分组返回每个分组中的第一行数据，select_list 一般是表中的属性列表(也可以是表达式)，多个列之间用英文逗号(,)分隔，如果要查询表中的所有列，可以使用“*”表示。

(3) FROM 子句说明查询是基于哪个(些)表或视图，如果是连接查询，可以使用 JOIN 关键字，详细解释见 5.1.3 节。

(4) WHERE 子句说明查询条件，可以用于查询条件的运算符也非常丰富，表 5-1 列出了常用的运算符。

(5) GROUP BY 子句说明查询分组，与之配套的 HAVING 子句说明分组条件，GROUP BY 分组通常用于分组的汇总查询。

(6) ORDER BY 子句说明查询结果的排序方式。

(7) LIMIT 或 FETCH 子句指定查询的输出结果行，通常与 ORDER BY 子句一起使用。

(8) FOR UPDATE 子句将对 SELECT 检索出来的行进行加锁，这样避免它们在当前事务结束前被其他事务修改或者删除。

SELECT 查询命令的使用非常灵活，用它可以构造各种各样的查询。本节将以网络购物系统中的表为例，介绍 SELECT 语句的各种用法。

5.1.2 单表查询

单表查询是指仅涉及一张表的查询。

1. 选择表中若干列

1) 查询指定列

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

例 5.1 查询所有商品的商品代码和商品名称。

```
SELECT gid,gname
FROM goods;
```

例 5.2 查询所有用户的用户代码、兴趣爱好和用户姓名。

```
SELECT uid,hobby,uname
FROM users;
```

说明：<目标列表达式>中各列的先后顺序可以与表中的顺序不一致。用户可以根据应用的需要改变列的显示顺序。本例中先列出用户代码，再列出兴趣爱好，最后列出用户姓名。查询结果中各列的顺序与<目标列表达式>中各列的顺序是一致的。

2）查询全部列

查询表中的所有属性列有两种方法：第一种方法是在 SELECT 子句后面列出所有列名；第二种方法是若列的显示顺序与其在原表中的顺序相同，也可以简单地将＜目标列表达式＞指定为“*”。

例 5.3 查询所有商品的商品代码、商品名称、商品类别、上下架状态、库存量、成本价格、销售价格。

```
SELECT gid, gname, category, gstatus, inventory, cprice, sprice
FROM goods;
```

等价于：

```
SELECT *
FROM goods;
```

3）查询经过计算的列

SELECT 子句的＜目标列表达式＞不仅可以是表中的属性列，也可以是算术表达式、字符串常量、函数等。

例 5.4 从用户表中查询用户代码、用户名和注册年份。

```
SELECT uid,uname,date_part('year',reg)
FROM users;
```

经过计算的列、函数产生的列和常量列在显示结果中不直观，通过定义列别名可以改变查询结果的列标题，这对于含算术表达式、常量、函数名的目标列尤为有用。定义列别名的语法格式如下：

```
列名|表达式 [AS] 列别名
```

例 5.4 可以定义如下列别名：

```
SELECT uid,uname,date_part('year',reg) AS regYear
FROM users;
```

2. 选择表中的若干元组

1）消除取值相同的行

例 5.5 查询购物车表中存在的商品代码。

```
SELECT gid
FROM cart;
```

说明：从本例可以看出，多个本来并不完全相同的元组，投影到指定的某些列上后，可能变成相同的行了。例如，本例的结果集中出现了 2 行“G001”、2 行“G006”、2 行“G008”、2 行“G014”等。本例要查询的是购物车表中存在的商品代码，取值相同的行在结果中是没有意义的，因此应消除掉。使用 DISTINCT 关键字可以解决这个问题，它的作用是去掉结果集中的重复行。注意，DISTINCT 关键字必须紧跟在 SELECT 关键字后面书写。因此本例应该用如下代码实现：

```
SELECT DISTINCT gid
FROM cart;
```

由于上例的查询结果只集中在一列，所以很容易误认为，DISTINCT 关键字是消除 DISTINCT 关键字后面单列的重复值，这种认识是错误的。DISTINCT 关键字是消除查询

结果集中的重复行。

例 5.6 查询 TT 表中 Col1 和 Col2 两列，要求去掉重复行，TT 表实例如下所示：

	Col1	Col2	Col3
1	a	b	c
2	a	b	d
3	a	c	e

```
SELECT DISTINCT Col1,Col2
FROM TT;
```

2）查询满足条件的元组

查询满足指定条件的元组可以通过 WHERE 子句实现。WHERE 子句常用的运算符如表 5-1 所示。

表 5-1 WHERE 子句中常用的运算符

查 询 方 式	运 算 符
比较	=、>、>=、<、<=、!=、<>
确定范围	BETWEEN AND、NOT BETWEEN AND
确定集合	IN、NOT IN
字符匹配	LIKE、NOT LIKE
空值	IS NULL、IS NOT NULL
否定	NOT

（1）比较大小。用于比较大小的运算符一般包括=（等于），>（大于），<（小于），>=（大于或等于），<=（小于或等于），!=、^=或<>（不等于）。有些产品还包括!>（不大于），!<（不小于），GaussDB 目前不支持。

例 5.7 查询订单状态为“未支付”的订单表的订单代码、用户代码和订单总金额。

```
SELECT oid,uid,osprice
FROM orders
WHERE ostatus='未支付';
```

说明：订单表中的订单总金额此时还未赋值，所以显示为空。赋值语句详见例 5.81。

例 5.8 查询库存量大于或等于 10 000 的商品代码、商品名称和库存量。

```
SELECT gid,gname,inventory
FROM goods
WHERE inventory>=10000;
```

例 5.9 查询注册时间在 2022 年 1 月 1 日之前的用户信息。

```
SELECT *
FROM users
WHERE reg<'2022-1-1';
```

说明：日期时间型的常量必须要加单引号。

例 5.10 查询订单表中在 2023 年 10 月 15 日之前下单的用户代码。

```
SELECT DISTINCT uid
FROM orders
WHERE createdate<'2023-10-15';
```

说明：由于一个用户可能在 2023 年 10 月 15 日之前有多次下单记录，所以在查询语句中应该通过使用 DISTINCT 关键字来消除重复的记录行，即消除重复的用户代码。

（2）逻辑查询。逻辑查询是由逻辑运算符 AND、OR、NOT 及其组合作为条件的查询。AND 和 OR 用于连接 WHERE 子句中的多个查询条件（布尔表达式）。NOT 用于反转查询条件的结果。当一个语句中使用了多个逻辑运算符时，计算顺序依次为 NOT＞AND＞OR。一般建议用户使用括号改变优先级，这样可以提高查询的可读性，并减少出现细微错误的可能性。使用括号不会造成重大的性能损失。

使用逻辑运算符 AND 的一般格式如下：

```
布尔表达式 1 AND 布尔表达式 2 AND …AND 布尔表达式 n
```

用 AND 连接的条件表示，只有当全部的布尔表达式的结果均为 True 时，整个表达式的结果才为 True；只要有一个布尔表达式的结果为 False，则整个表达式的结果为 False。

使用逻辑运算符 OR 的一般格式如下：

```
布尔表达式 1 OR 布尔表达式 2 OR …OR 布尔表达式 n
```

用 OR 连接的条件表示，只要其中一个布尔表达式为 True，则整个表达式的结果为 True；只有当全部布尔表达式的结果均为 False 时，整个表达式结果为 False。

使用逻辑运算符 NOT 的一般格式如下：

```
NOT 布尔表达式
```

当布尔表达式的结果为 True 时，整个表达式的结果为 Flase；当布尔表达式的结果为 False 时，整个表达式的结果为 True。

例 5.11 查询兴趣爱好为“阅读”且性别为“女”的用户代码和用户名。

```
select uid,uname
from users
where hobby='阅读' and gender='女';
```

例 5.12 查询兴趣爱好为“阅读”或者“音乐”的用户代码和用户名。

```
select uid,uname
from users
where hobby='阅读' or hobby='音乐';
```

例 5.13 查询库存量在 3000 以上（包括 3000），且商品类别为“图书”或”“办公用品”的商品信息。

```
SELECT *
FROM goods
WHERE (inventory >=3000) and ((category='图书')OR (category='办公用品'));
```

（3）确定范围。谓词 BETWEEN…AND…和 NOT　BETWEEN…AND…可以用来查找属性值在（或不在）指定范围内的元组，其中，BETWEEN 后面是范围的下限（即低值），AND 后面是范围的上限（即高值）。

使用 BETWEEN…AND…的一般格式如下：

```
列名|表达式 BETWEEN 下限值 AND 上限值
```

使用 BETWEEN…AND…的条件表达式的结果等价于下面条件表达式的结果：

```
(列名|表达式>=下限值) AND (列名|表达式<=上限值)
```

使用 NOT BETWEEN…AND…的一般格式如下：

```
列名|表达式 NOT BETWEEN 下限值 AND 上限值
```

使用 NOT BETWEEN…AND…的条件表达式的结果等价于下面条件表达式的结果：

```
(列名|表达式<下限值) OR (列名|表达式>上限值)
```

BETWEEN…AND…和 NOT BETWEEN…AND…一般用于对数值型数据和日期型数据进行比较。列名或表达式的类型要与下限值或上限值的类型相同。

例 5.14 查询库存量在 3000 至 6000(包括 3000 和 6000)之间的商品的名称、成本价格、销售价格和库存量。

```
SELECT gname ,cprice ,sprice ,inventory
FROM goods
WHERE inventory BETWEEN 3000 AND 6000;
```

等价于：

```
SELECT gname ,cprice ,sprice ,inventory
FROM goods
WHERE (inventory>=3000) AND (inventory<=6000);
```

例 5.15 查询库存量不在 3000 至 6000(包括 3000 和 6000)之间的商品的名称、成本价格、销售价格和库存量。

```
SELECT gname ,cprice ,sprice ,inventory
FROM goods
WHERE inventory NOT BETWEEN 3000 AND 6000;
```

等价于：

```
SELECT gname ,cprice ,sprice ,inventory
FROM goods
WHERE (inventory<3000) OR (inventory>6000);
```

(4) 确定集合。谓词 IN 可以用来查找属性值指定集合的元组。

使用 IN 的一般格式如下：

```
列名|表达式 IN (常量 1,常量 2,…,常量 n)
```

当列值(或表达式值)与 IN 集合中的某个常量值相等时，结果为 True；当列值(或表达式值)与 IN 集合中的任何一个常量值都不相等时，结果为 False。使用 IN 的条件表达式的结果等价于下面条件表达式的结果：

```
(列名|表达式=常量 1) OR (列名|表达式=常量 2 ) OR … OR (列名|表达式=常量 n)
```

使用 NOT IN 的一般格式如下：

```
列名|表达式 NOT IN (常量 1,常量 2,…,常量 n)
```

当列值(或表达式值)与 IN 集合中的任何一个常量值都不相等时，结果为 True；当列值(或表达式值)与 IN 集合中的某个常量值相等时，结果为 False；使用 NOT IN 的条件表达式的结果等价于下面条件表达式的结果：

```
(列名|表达式<>常量 1) AND (列名|表达式<>常量 2 ) AND … AND (列名|表达式<>常量 n)
```

例 5.16 查询商品类别为“图书”“办公用品”“数码”的商品名称、类别和库存量。

```
SELECT gname ,category,inventory
FROM goods
WHERE category IN('图书','办公用品','数码');
```

等价于：

```
SELECT gname ,category,inventory
FROM goods
WHERE (category='图书') OR (category='办公用品') OR (category='数码');
```

例 5.17 查询商品类别不为“图书”“办公用品”“数码”的商品名称、类别和库存量。

```
SELECT gname ,category,inventory
FROM goods
WHERE category NOT IN('图书','办公用品','数码');
```

等价于：

```
SELECT gname ,category,inventory
FROM goods
WHERE (category<>'图书') AND (category<>'办公用品') AND (category<>'数码');
```

(5) 字符匹配。谓词 LIKE 确定特定字符串是否与指定匹配串相匹配。匹配串可以包含常规字符和通配符。匹配过程中，常规字符必须与字符串中指定的字符完全匹配。但是，通配符可以与字符串的任意部分相匹配。与使用＝和!＝字符串比较运算符相比，使用通配符可使 LIKE 运算符更加灵活。其一般语法格式如下：

```
列名|字符串表达式 [NOT] LIKE '<匹配串>'
```

其含义是查找指定的列值与＜匹配串＞相匹配(或不匹配)的元组。＜匹配串＞可以是一个完整的字符串，也可以含有通配符，常用通配符如表 5-2 所示。

表 5-2 常用通配符及其含义

通 配 符	含 义
_	匹配任何单个字符
%	匹配包含零个或多个字符串的任意字符串

例 5.18 查询用户表中姓“王”的用户信息。

```
SELECT *
FROM users
WHERE uname LIKE '王%';
```

例 5.19 查询收货地址为“北京”的用户代码和收货地址。

```
SELECT uid,addressinfo
FROM address
WHERE addressinfo LIKE '北京%'
```

例 5.20 查询收货地址不为“北京”的用户代码和收货地址。

```
SELECT uid,addressinfo
FROM address
WHERE addressinfo NOT LIKE '北京%'
```

例 5.21 查询收货地址中含有“河西区”的用户代码和收货地址。

```
SELECT uid,addressinfo
FROM address
WHERE addressinfo LIKE '%河西区%'
```

例 5.22 查询用户表中姓名第 2 个字为“晓”的用户信息。

```
SELECT *
FROM users
WHERE uname LIKE '_晓%';
```

(6) 空值。空值表示值未知。空值不同于空白或零值。没有两个相等的空值。比较两个空值或将空值与任何其他值相比均返回未知,这是因为每个空值均为未知。空值一般表示数据未知、不适用或将在以后添加数据。

若要在查询中测试空值,则在 WHERE 子句中使用 IS NULL 或 IS NOT NULL。具体格式如下:

```
列名|表达式 IS [NOT] NULL
```

不能使用普通的比较运算符(=、!=等)来判断某个列或表达式是否为 NULL 值。

例 5.23 查询没有填写兴趣爱好的用户代码和用户名。

```
SELECT uid,uname
FROM users
WHERE hobby IS NULL ;
```

例 5.24 查询填写了兴趣爱好的用户代码和用户名。

```
SELECT uid,uname
FROM users
WHERE hobby IS NOT NULL ;
```

3. 对查询结果进行排序

用户可以用 ORDER BY 子句对查询结果按照一个或多个属性列的升序(ASC)或降序(DESC)排序,省略值为升序。ORDER BY 之所以重要,是因为关系理论规定除非已经指定 ORDER BY,否则不能假设查询结果集中的行带有任何序列。如果查询结果集中行的顺序对 SELECT 语句很重要,那么在 SELECT 语句中就必须使用 ORDER BY 子句。ORDER BY 子句的一般格式如下:

```
ORDER BY <列名>[ASC|DESC][,…n]
```

空值被视为最低的可能值。

例 5.25 指定一列作为排序依据列。查询商品表信息,要求查询结果按销售价格的升序排列。

```
SELECT *
FROM goods
ORDER BY sprice ASC
```

说明: “ORDER BY sprice ASC”中的 ASC 是可以省略的。

例 5.26 指定一列作为排序依据列。查询商品表信息,要求查询结果按销售价格的降序排列。

```
SELECT *
FROM goods
ORDER BY sprice DESC
```

例 5.27 指定多列作为排序依据列。查询商品表信息，要求查询结果按商品类别的升序排列，同一类别的再按库存量的降序排列。

```
SELECT *
FROM goods
ORDER BY category ASC, inventory DESC
```

4. 限制查询结果的数量

在查询数据时可以使用 LIMIT 子句或 FETCH 子句限制查询结果的数量。其语法格式如下：

```
[LIMIT { [offset,] count | ALL }]
[OFFSET start [ ROW | ROWS ]]
[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
```

其中，LIMIT 表示最多返回 count 行结果，参数 offset 为可选项，指定从哪一行开始显示，offset 为 0（默认值）表示从第一条记录开始显示，offset 为 1 表示从第二条记录开始显示，以此类推；OFFSET 表示跳过指定的 start 行数（在显示查询结果时，忽略前 start 条记录，从 start+1 条记录开始显示），默认为 0；FETCH 表示最多返回 count 行结果，默认为 1；ROW 和 ROWS 是同义词，FIRST 和 NEXT 是同义词；ONLY 表示不返回更多的数据。

注意事项：

（1）如果 SELECT 语句中既有 LIMIT（或 FETCH）子句，又有 ORDER BY 子句，则 LIMIT（或 FETCH）子句必须在 ORDER BY 子句后面，否则将报语法错误。

（2）LIMIT ALL 和没有 LIMIT 子句作用是一样的，表示不限定查询返回的记录数。

（3）OFFSET 0 和没有 OFFSET 子句作用是一样的。

例 5.28 查询运动类商品的销售价格最高前 3 名的商品代码、商品名称、商品类别和销售价格。

```
SELECT gid,gname,category,sprice
FROM goods
WHERE category ='运动'
ORDER BY sprice DESC
LIMIT 3;
```

该查询也可以用 FETCH 子句，查询语句如下：

```
SELECT gid,gname,category,sprice
FROM goods
WHERE category ='运动'
ORDER BY sprice DESC
FETCH FIRST 3 rows ONLY ;
```

例 5.29 查询销售价格最高的忽略前 2 名的商品代码、商品名称、商品类别和销售价格。

```
SELECT gid,gname,category,sprice
FROM goods
ORDER BY sprice DESC
OFFSET 2 rows;
```

例 5.30 查询销售价格最高的第 3 名到第 6 名的商品代码、商品名称、商品类别和销售价格。

```
SELECT gid,gname,category,sprice
FROM goods
ORDER BY sprice DESC
OFFSET 2 rows
FETCH NEXT 4 rows ONLY;
```

5. 分组与汇总查询

SQL SELECT 查询可以直接对查询结果进行汇总计算，也可以对查询结果进行分组计算。在查询中完成汇总计算的函数称为聚合函数，实现分组查询的子句为 GROUP BY 子句。

1）聚合函数与汇总查询

聚合函数对一组值执行计算，并返回单个值。常用的聚合函数如表 5-3 所示。

表 5-3 常用的聚合函数

聚 合 函 数	含 义
COUNT(*)	统计元组的个数
COUNT([DISTINCT|ALL]<列名|表达式>)	统计一列中值的个数
SUM([DISTINCT|ALL]<列名|表达式>)	计算一列值的总和(此列必须是数值型)
AVG([DISTINCT|ALL]<列名|表达式>)	计算一列值的平均值(此列必须是数值型)
MAX([DISTINCT|ALL]<列名|表达式>)	求一列值中的最大值
MIN([DISTINCT|ALL]<列名|表达式>)	求一列值中的最小值

说明：如果指定 DISTINCT 关键字，则表示在统计时要取消指定列的重复值。如果不指定 DISTINCT 关键字或指定 ALL 关键字(默认选项)，则表示不取消重复值。

除了 COUNT(*) 以外，聚合函数都会忽略空值。

例 5.31 查询用户的总数。

```
SELECT COUNT(*)
FROM users ;
```

例 5.32 查询有过订单记录的用户总数。

```
SELECT COUNT(distinct uid)
FROM orders;
```

例 5.33 查询 2022 年 1 月 1 日之后下单的所有订单的订单总金额和平均订单总金额。

```
SELECT SUM(osprice),AVG(osprice)
FROM orders
WHERE createdate>='2022/1/1';
```

查询结果如下所示：

	sum	avg
1	NULL	NULL

说明：由于这里销售金额没有赋值，所以查询结果为 NULL。

执行完例 5.81 给订单表的销售金额列赋值后，此题执行结果如下：