第 5 章

使用 SQL 管理表数据

对数据库中表数据执行添加、删除、修改和查询操作是必不可少的工作。查询是指从数据库中获取用户所需要的数据，查询操作在数据库操作中经常用到，而且也是最重要的操作之一。添加是向数据库表中添加不存在的记录，修改是对已经存在的记录进行更新，删除则是删除数据库中已存在的记录。

查询数据库中的记录有多种方式，可以查询所有的数据，也可以根据用户自己的需要进行查询，还可以借助集合函数进行查询。通过不同的查询方式，可以获取不同的数据。

5.1 SQL 的数据操纵功能

SQL 语言的数据操纵语句 DML 主要包括插入数据、修改数据和删除数据三种语句。

5.1.1 插入数据记录

插入数据是把新的记录插入到一个已有表中。插入数据使用语句 INSERT INTO，可分为以下几种情况。

1. 插入一行新记录

语法格式为：

```
INSERT INTO<表名>[(<列名 1>[,<列名 2>…])]VALUES(<值>)
```

其中：

- ＜表名＞是指要插入新记录的表。
- ＜列名＞是可选项，指定待添加数据的列，列出列名，则 VALUES 子句中值的排列顺序必须和列名表中的列名排列顺序一致，个数相等，数据类型一一对应；若省略列名，则 VALUES 子句中值的排列顺序必须和定义表时的列名排列顺序一致，个数相等，数据类型一一对应。
- VALUES 子句指定待添加数据的具体值。

【例 5-1】 在 student 表中插入一条学生记录(学号：'202011070339'，姓名：梦欣怡，性别：女，出生年月：2002-06-18，专业号：1102，所在班：大数据 2001)。

```
INSERT INTO student VALUES ('202011070339', '梦欣怡', '女', '2002-06-18', '1102',
'大数据 2001');
```

注意：

- 必须用逗号将各个数据分隔开，字符型数据要用单引号括起来。
- INTO子句中没有指定列名，则新插入的记录必须在每个属性列上均有值，且VALUES子句中值的排列顺序要和表中各属性列的排列顺序一致。

2. 插入一行的部分数据值

【例 5-2】 在sc表中插入一条选课记录('202011070339','58130540')。

```
INSERT INTO sc(Sno, Cno) VALUES ('202011070339 ', '58130540');
```

语句将VALUES子句中的值按照INTO子句中指定列名的顺序插入到表中。

对于INTO子句中没有出现的列，新插入的记录在这些列上将取空值，如sc表中的grade列即赋空值(NULL)。

但对于那些在表定义时有NOT NULL约束的属性列，则不能取空值。

3. 插入多行记录

用户可以从一个表中抽取数据插入另一表中，这可通过子查询来实现。

插入数据的命令语法格式为：

```
INSERT INT0<表名>[(<列名 1>[,<列名 2>…])]
子查询
```

【例 5-3】 建一点名表studentlist(Sno, Sname, Ssex)，其中字段含义分别是学生学号，学生姓名，学生性别，并把学生表中的相关数据插入到点名表中。

```
CREATE TABLE studentlist
(
    Sno CHAR(10),
    Sname VARCHAR(20),
    Ssex VARCHAR(10)
);
INSERT INTO studentlist(Sno,Sname,Ssex) SELECT Sno,Sname,Ssex FROM student;
```

5.1.2　修改数据记录

SQL语言可以使用UPDATE语句对表中的一行或多行记录的某些列的值进行修改，其语法格式为：

```
UPDATE<表名>
    SET<列名>=<表达式>[,<列名>=<表达式>…]
    [WHERE<条件>]
```

其中：

- ＜表名＞：是指要修改的表。
- SET子句：给出要修改的列及其修改后的值。
- WHERE子句指定待修改的记录应当满足的条件，WHERE子句省略时，则修改

表中的所有记录。

下面的示例修改一行记录。

【例 5-4】 把转专业的“郭爽”同学从“供应链 2001”转到“区块链 2001”。

```
UPDATE student SET Sclass='区块链 2001' WHERE Sname='郭爽';
```

下面的示例修改多行记录。

【例 5-5】 将所有课程的学分增加 1。

```
UPDATE course SET Ccredit=Ccredit+1;
```

【例 5-6】 把选修表中每个同学的成绩提高 5 分。

```
UPDATE sc SET Grade=Grade+5;
```

5.1.3 删除数据记录

使用 DELETE 语句可以删除表中的一行或多行记录，其语法格式为：

```
DELETE
FROM<表名>
[WHERE<条件>]
```

其中：

- ＜表名＞：要删除数据的表。
- WHERE 子句：指定待删除的记录应当满足的条件，WHERE 子句省略时，则删除表中的所有记录。

下面的示例删除一行记录。

【例 5-7】 删除“马琦”同学的记录。

```
DELETE FROM student WHERE Sname='马琦';
```

下面的示例删除多行记录。

【例 5-8】 从学生表中删除所有工商 1401 班的同学记录。

```
DELETE FROM student WHERE Sclass='工商 1401';
```

【例 5-9】 删除学生表中的所有记录。

```
DELETE FROM student;
```

执行此语句后，student 表即为一个空表，但其定义仍存在于数据字典中。

5.1.4 使用 TRUNCATE 清空表数据

TRUNCATE 用于完全清空一个表，基本语法格式如下：

```
TRUNCATE[table]表名
```

【例 5-10】　清除 sc 表。

```
TRUNCATE table sc;
```

注意：TRUNCATE TABLE 与 DELETE 的区别如下。

- TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同：两者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少。DELETE 语句每次删除一行，会在事务日志中为所删除的每行记录一项。
- TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据，且只在事务日志中记录页的释放。

TRUNCATE、DELETE、DROP 的比较如下。

- TRUNCATE TABLE：删除内容、释放空间但不删除定义。
- DELETE TABLE：删除内容不删除定义，不释放空间。
- DROP TABLE：删除内容和定义，释放空间。

5.2　SQL 的数据查询功能

查询功能是 SQL 语言的核心功能，是数据库中使用最多的操作，查询语句也是最复杂的一个语句。本节中的例子都是基于 jxgl 数据库的，数据表中的数据请参见 5.3 节。

5.2.1　查询语句 SELECT 的基本结构

SELECT 语句可以有效地从数据库的表或视图中访问和提取数据，并具有强大的单表和多表查询功能，正因为如此，SELECT 语句的可选项很多，语法也比较复杂。其一般格式为：

```
SELECT[ALL|DISTINCT]<目标列表达式>[[AS]<新列名>][,…n]
    FROM<表名或视图名>[[AS]<别名>][…n]
    [WHERE<条件表达式>]
    [GROUP BY<分组依据列>]
    [HAVING<条件表达式>]
    [ORDER BY<排序依据列>[ASC|DESC]][,…n]
    [LIMIT N,M]
```

其中：

- [ALL|DISTINCT]：指定在结果集中是否显示重复行。ALL 表示显示，为默认值；DISTINCT 表示不显示。
- ＜目标列表达式＞[[AS]＜新列名＞][,…n]：指定为结果集选定的列。特别地，如果该处为“*”，则表示输出所有列。
- ＜表名或视图名＞[[AS]＜别名＞][,…n]：指定从其中检索数据的表或视图。
- [WHERE＜条件表达式＞]：指定数据检索的条件。

- [GROUP BY＜分组依据列＞]：实现对数据的分组查询。
- [HAVING＜条件表达式＞]：用于分组后的筛选条件。
- [ORDER BY＜排序依据列＞[ASC|DESC]][,…n]：对结果集按＜排序依据列＞指定的列的值排序。其中 ASC 表示升序排列，为默认值；DESC 表示降序排列。
- [LIMIT N,M]：表示只从查询结果集中输出从 N 到 M 行。

整个 SELECT 语句的含义是：根据 WHERE 子句的条件表达式，从 FROM 子句指定的基本表或视图中找出满足条件的元组，再按 SELECT 子句中的目标列表达式，选出元组中的属性值，形成结果表。如果有 GROUP 子句，则将结果按＜分组依据列＞的值进行分组，该属性列的值相等的元组为一个组，每个组产生结果表中的一条记录。通常会在分组中使用集函数。如果 GROUP 子句带 HAVING 短语，则只有满足指定条件的组才予以输出。如果有 ORDER 子句，则还需对结果按＜排序依据列＞的值升序或降序排列。

5.2.2 单表查询

1. 选择表中的若干列

下面的示例查询全部列。

【例 5-11】 查询全体学生的详细记录。

```
SELECT * FROM student;
```

该 SELECT 语句实际上是无条件地把 student 表的全部信息都查询出来，所以也称为全表查询，这是很常用的也是最简单的一种查询。

输出结果如图 5-1 所示。

信息 | 结果 1 | 剖析 | 状态

sno	sname	ssex	sbirth	zno	sclass
202011070338	孙一凯	男	2000-10-11	1102	大数据2001
202011855228	唐晓	女	2002-11-05	1102	大数据2001
202011855321	蓝梅	女	2002-07-02	1102	大数据2001
202011855426	余小梅	女	2002-06-18	1102	大数据2001
202012040137	郑熙婷	女	2003-05-23	1214	区块链2001
202012855223	徐美利	女	2000-09-07	1214	区块链2001
202014070116	欧阳贝贝	女	2002-01-08	1407	健管2001
202014320425	曹平	女	2002-12-14	1407	健管2001
202014855302	李壮	男	2003-01-17	1409	智能医学2001
202014855308	马琦	男	2003-06-14	1409	智能医学2001
202014855328	刘梅红	女	2000-06-12	1407	健管2001
202014855406	王松	男	2003-10-06	1409	智能医学2001
202016855305	聂鹏飞	男	2002-08-25	1601	供应链2001
202016855313	郭爽	女	2001-02-14	1601	供应链2001
202018855212	李冬旭	男	2003-06-08	1805	智能感知2001
202018855232	王琴雪	女	2002-07-20	1805	智能感知2001

图 5-1 查询全体学生详细记录的结果

下面的示例查询指定列。

【例 5-12】 查询全体学生的学号与姓名。

```
SELECT Sno,Sname FROM student;
```

输出结果如图 5-2 所示。

目标列表达式中各个列的先后顺序可以与表中的顺序不一致。也就是说，用户在查询时可以根据应用的需要改变列的显示顺序。

【例 5-13】 查询全体学生的姓名、学号、所在班级。

```
SELECT Sname,Sno,Sclass FROM student;
```

输出结果如图 5-3 所示。

信息　结果 1　剖析　状态

Sno	Sname
202011070338	孙一凯
202011855228	唐晓
202011855321	蓝梅
202011855426	余小梅
202012040137	郑熙婷
202012855223	徐美利
202014070116	欧阳贝贝
202014320425	曹平
202014855302	李壮
202014855308	马琦
202014855328	刘梅红
202014855406	王松
202016855305	聂鹏飞
202016855313	郭爽
202018855212	李冬旭
202018855232	王琴雪

图 5-2　查询全体学生的学号与姓名的结果

信息　结果 1　剖析　状态

Sname	Sno	Sclass
孙一凯	202011070338	大数据2001
唐晓	202011855228	大数据2001
蓝梅	202011855321	大数据2001
余小梅	202011855426	大数据2001
郑熙婷	202012040137	区块链2001
徐美利	202012855223	区块链2001
欧阳贝贝	202014070116	健管2001
曹平	202014320425	健管2001
李壮	202014855302	智能医学2001
马琦	202014855308	智能医学2001
刘梅红	202014855328	健管2001
王松	202014855406	智能医学2001
聂鹏飞	202016855305	供应链2001
郭爽	202016855313	供应链2001
李冬旭	202018855212	智能感知2001
王琴雪	202018855232	智能感知2001

图 5-3　查询全体学生的姓名、学号、所在班级的结果

结果表中列的顺序与基表中不同，是按查询要求先列出姓名属性，再列出学号属性和所在系属性。

下面示例查询经过计算的值。

SELECT 子句的＜目标列表达式＞不仅可以是表中的属性列，也可以是有关表达式，即可以将查询出来的属性列经过一定的计算后列出结果。

【例 5-14】 查询全体学生的姓名及其年龄。

```
SELECT Sname,YEAR(now())-YEAR(Sbirth) FROM student;
```

本例中，＜目标列表达式＞中第二项不是通常的列名，而是一个计算表达式，是用当前的年份减去学生的出生日期，这样所得的即为学生的年龄。其中，YEAR()是输出年份的函数，now()是输出当前日期的函数。

输出结果如图 5-4 所示。

表达式不仅可以是算术表达式，还可以是字符串常量、函数等。

【例 5-15】 查询全体学生的姓名、出生年份。

```
SELECT Sname AS 学生姓名, YEAR(Sbirth) AS 出生年份 FROM student;
```

输出结果如图 5-5 所示。

信息 | 解释 1 | 结果 1 | 剖析 | 状态

Sname	YEAR(now())-YEAR(Sbirth)
孙一凯	21
唐晓	19
蓝梅	19
余小梅	19
郑熙婷	18
徐美利	21
欧阳贝贝	19
曹平	19
李壮	18
马琦	18
刘梅红	21
王松	18
聂鹏飞	19
郭爽	20
李冬旭	18
王琴雪	19

图 5-4 查询全体学生的姓名及其年龄的结果

信息 | 解释 1 | 结果 1 | 剖析 | 状态

学生姓名	出生年份
孙一凯	2000
唐晓	2002
蓝梅	2002
余小梅	2002
郑熙婷	2003
徐美利	2000
欧阳贝贝	2002
曹平	2002
李壮	2003
马琦	2003
刘梅红	2000
王松	2003
聂鹏飞	2002
郭爽	2001
李冬旭	2003
王琴雪	2002

图 5-5 查询全体学生的姓名及其出生年份（列取别名）的结果

从上例可以看出用户可以通过指定别名来改变查询结果的列标题，这对于含算术表达式、常量、函数名的目标列表达式尤其有用。

2. 选择表中的若干元组

下面的示例消除取值重复的行。

【例 5-16】 查询所有选修过课的学生学号。

```
SELECT Sno FROM sc;
```

执行上面的 SELECT 语句后，输出结果如图 5-6 所示。

该查询结果里包含了许多重复行。如果想去掉结果表中的重复行，应使用 DISTINCT 短语：

```
SELECT DISTINCT Sno FROM sc;
```

输出结果如图 5-7 所示。

要查询满足条件的元组，可以通过 WHERE 子句实现。WHERE 子句常用的查询条件如表 5-1 所示。

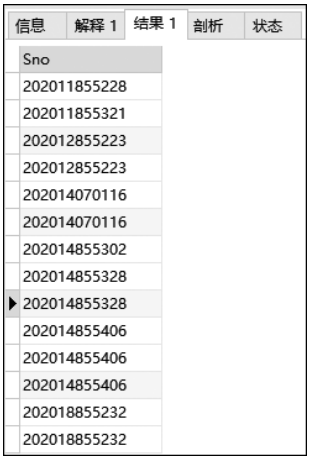


图 5-6　查询所有选修过课的学生学号的结果

图 5-7　查询所有选修过课的学生学号(去除重复行的结果)

表 5-1　常用的查询条件

查 询 条 件	谓　　词
比较(比较运算符)	=、>、>=、<、<=、<>(!=)、NOT
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
多重条件	AND、OR

3. 比较大小

下面的示例比较大小。

【例 5-17】 查询“大数据 2001”班的全体学生名单。

```
SELECT Sname FROM student WHERE Sclass='大数据 2001';
```

输出结果如图 5-8 所示。

【例 5-18】 查询所有“2001”年以前出生学生的姓名及其出生日期。

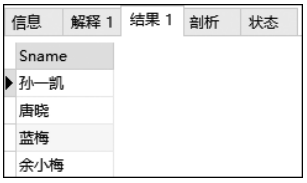
```
SELECT Sname,Sbirth FROM student WHERE Sbirth<'2001-01-01';
```

输出结果如图 5-9 所示。

【例 5-19】 查询考试成绩低于 75 的学生学号。

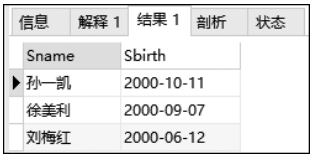
```
SELECT DISTINCT Sno FROM sc WHERE Grade<75;
```

这里使用了 DISTINCT 短语,会在结果集中去掉重复行。用在此处,使得当某个学生有多门课程不及格时,其学号也只列一次。



信息	解释 1	结果 1	剖析	状态
Sname				
孙一凯				
唐晓				
蓝梅				
余小梅				

图 5-8　查询“大数据 2001”班的全体学生名单的结果



信息	解释 1	结果 1	剖析	状态
Sname	Sbirth			
孙一凯	2000-10-11			
徐美利	2000-09-07			
刘梅红	2000-06-12			

图 5-9　查询所有“2001”年以前出生学生的姓名及其出生日期的结果

输出的结果如图 5-10 所示。

【例 5-20】　查询在'2000-01-01' 和'2002-12-31'之间出生的学生的姓名、班级和出生日期。

```
SELECT Sname,Sclass,sbirth FROM student WHERE sbirth
    BETWEEN '2000-01-01' AND '2002-12-31' ;
```

输出结果如图 5-11 所示。

信息	解释 1	结果 1	剖析	状态
Sno				
202011855321				
202012855223				
202014070116				

图 5-10　查询考试成绩低于 75 的学生学号的结果

信息	解释 1	结果 1	剖析	状态
Sname	Sclass	sbirth		
孙一凯	大数据2001	2000-10-11		
唐晓	大数据2001	2002-11-05		
蓝梅	大数据2001	2002-07-02		
余小梅	大数据2001	2002-06-18		
徐美利	区块链2001	2000-09-07		
欧阳贝贝	健管2001	2002-01-08		
曹平	健管2001	2002-12-14		
刘梅红	健管2001	2000-06-12		
聂鹏飞	供应链2001	2002-08-25		
郭爽	供应链2001	2001-02-14		
王琴雪	智能感知2001	2002-07-20		

图 5-11　查询在两个日期之间出生学生的姓名、班级和出生日期的结果

与 BETWEEN…AND…相对的谓词是 NOT BETWEEN…AND…。

【例 5-21】　查询不在'2000-01-01' 和'2002-12-31'之间出生的学生的姓名、班级和出生日期。

```
SELECT Sname,Sclass,sbirth FROM student WHERE sbirth
    NOT BETWEEN '2000-01-01' AND '2002-12-31' ;
```

输出结果如图 5-12 所示。

【例 5-22】　查询“区块链 2001”和“供应链 2001”班学生的姓名和性别。

```
SELECT Sname,Ssex FROM student WHERE Sclass IN('区块链 2001','供应链 2001');
```

输出结果如图 5-13 所示。

信息 | 解释 1 | 结果 1 | 剖析 | 状态

Sname	Sclass	sbirth
郑熙婷	区块链2001	2003-05-23
李壮	智能医学2001	2003-01-17
马琦	智能医学2001	2003-06-14
王松	智能医学2001	2003-10-06
李冬旭	智能感知2001	2003-06-08

图 5-12　查询不在两个日期之间出生学生的姓名、班级和出生日期的结果

信息 | 解释 1 | 结果 1 | 剖析 | 状态

Sname	Ssex
郑熙婷	女
徐美利	女
聂鹏飞	男
郭爽	女

图 5-13　查询指定两个班级学生的姓名和性别的结果

与 IN 相对的谓词是 NOT IN，用于查找属性值不属于指定集合的元组。

【例 5-23】 查询既不是“区块链 2001”也不是“供应链 2001”班的学生的姓名和性别。

```
SELECT Sname, Ssex FROM student WHERE Sclass NOT IN('区块链 2001','供应链 2001');
```

输出结果如图 5-15 所示。

谓词 LIKE 可以用来进行字符串的匹配。其语法格式如下：

```
[NOT]LIKE '<匹配串>'
```

其含义是查找指定的属性列值与＜匹配串＞相匹配的元组。

＜匹配串＞可以是一个完整的字符串，也可以含有以下通配符：

- %(百分号)：代表任意长度(长度可以为 0)的字符串。
- _(下横线)：代表任意单个字符。
- []：匹配[]中的任意一个字符。
- [^]：不匹配[]中的任意一个字符。

信息 | 解释 1 | 结果 1 | 剖析 | 状态

Sname	Ssex
孙一凯	男
唐晓	女
蓝梅	女
余小梅	女
欧阳贝贝	女
曹平	女
李壮	男
马琦	男
刘梅红	女
王松	男
李冬旭	男
王琴雪	女

图 5-14　查询不在指定两个班级中的学生姓名和性别的结果

当＜匹配串＞中没有通配符时，“LIKE”的作用等同于“＝”。

【例 5-24】 查询所有姓李的学生的姓名、学号和性别。

```
SELECT Sname,Sno,Ssex FROM student WHERE Sname LIKE '李%';
```

输出结果如图 5-15 所示。

【例 5-25】 查询姓名中第二个字为“小”的学生的姓名和学号。

```
SELECT Sname,Sno FROM student WHERE Sname LIKE'_小%';
```

输出结果如图 5-16 所示。

【例 5-26】 查询所有不姓李的学生的姓名。

```
SELECT Sname FROM student WHERE Sname NOT LIKE'李%';
```