第二章 MySQL数据的增删改查

课前回顾

1. 在数据库exercise中创建课程表stu_course，包含字段课程编号(number)，类型为整数，长度为11，是主键，自增长，非空、课程名称(name)，类型为字符串，长度为20，非空、学分(score)，类型为浮点数，小数点后面保留2位有效数字， 长度为5，非空

```
-- 如果数据库不存在 就创建数据库
CREATE DATABASE IF NOT EXISTS exercise DEFAULT CHARACTER SET UTF8 COLLATE
UTF8_GENERAL_CI;
-- 使用数据库
USE exercise;
-- 在数据库中创建数据表stu_course
CREATE TABLE IF NOT EXISTS stu_course(
    `number` INT(11) AUTO_INCREMENT PRIMARY KEY NOT NULL COMMENT '课程编号',
    name VARCHAR(20) NOT NULL COMMENT '课程名称',
    score DOUBLE(5, 2) NOT NULL COMMENT '学分'
)ENGINE=InnoDB CHARSET=UTF8 COMMENT '课程表';
```

2. 将课程表重命名为course

```
ALTER TABLE stu_course RENAME AS course;
```

3. 在课程表中添加字段学时(time)，类型为整数，长度为3，非空

```
ALTER TABLE course ADD `time` INT(3) NOT NULL COMMENT '学时';
```

4. 修改课程表学分类型为浮点数，小数点后面保留1位有效数字，长度为3，非空

```
ALTER TABLE course MODIFY score DOUBLE(3, 1) NOT NULL COMMENT '学分';
```

5. 删除课程表

```
DROP TABLE IF EXISTS course;
```

6. 删除数据库exercise

```
DROP DATABASE IF EXISTS exercise;
```

章节内容

- DML--插入语句 **重点**
- DML--修改语句 **重点**
- DML--删除语句 **重点**
- DQL -- 查询语句 **重点**
- 聚合函数 **重点**
- 分组查询 **重点**
- 查询排序 **重点**

- 分页查询 **重点**

章节目标

- 掌握DML--插入语句
- 掌握DML--修改语句
- 掌握DML--删除语句
- 掌握DQL--查询语句
- 掌握聚合函数的使用
- 掌握分组查询
- 掌握查询排序
- 掌握分页查询

第一节 DML语句

1. 什么是DML

DML全称为Data Manipulation Language，表示数据操作语言。主要体现于对表数据的增删改操作。因此DML仅包括INSERT、UPDATE和DELEETE语句。

2. INSERT语句

```
-- 需要注意，VALUES后的字段值必须与表名后的字段名一一对应
INSERT INTO 表名(字段名1, 字段名2, ..., 字段名n) VALUES(字段值1, 字段值2, ..., 字段值n);
-- 需要注意，VALUES后的字段值必须与创建表时的字段顺序保持一一对应
INSERT INTO 表名 VALUES(字段值1, 字段值2, ..., 字段值n);

-- 一次性插入多条数据
INSERT INTO 表名(字段名1, 字段名2, ..., 字段名n) VALUES(字段值1, 字段值2, ..., 字段值n),(字段值1, 字段值2, ..., 字段值n), ... , (字段值1, 字段值2, ..., 字段值n);
INSERT INTO 表名 VALUES(字段值1, 字段值2, ..., 字段值n), (字段值1, 字段值2, ..., 字段值n), ..., (字段值1, 字段值2, ..., 字段值n);
```

示例

向课程表中插入数据

```
INSERT INTO course(`number`, name, score, `time`) VALUES (1, 'Java基础', 4, 40);
INSERT INTO course VALUES (2, '数据库', 3, 20);
INSERT INTO course(`number`, score,  name, `time`) VALUES (3, 5, 'Jsp', 40);
INSERT INTO course(`number`, name, score, `time`) VALUES (4, 'Spring', 4, 5),(5, 'Spring Mvc', 2, 5);
INSERT INTO course VALUES (6, 'SSM', 2, 3), (7, 'Spring Boot', 2, 2);
```

3. UPDATE语句

```
UPDATE 表名 SET 字段名1=字段值1[,字段名2=字段值2, ..., 字段名n=字段值n] [WHERE 修改条件]
```

3.1 WHERE条件子句

在Java中，条件的表示通常都是使用关系运算符来表示，在SQL语句中也是一样，使用 >, <, >=, <=, != 来表示。不同的是，除此之外，SQL中还可以使用SQL专用的关键字来表示条件。这些将在后面的DQL语句中详细讲解。

在Java中，条件之间的衔接通常都是使用逻辑运算符来表示，在SQL语句中也是一样，但通常使用AND来表示逻辑与(&&)，使用OR来表示逻辑或(||)

示例

```
WHERE time > 20 && time < 40;  <=>  WHERE time > 20 and time <40;
```

3.2 UPDATE语句

示例

将数据库的学分更改为4，学时更改为15

```
UPDATE course SET score=4, `time`=15 WHERE name='数据库';
```

4. DELETE语句

```
DELETE FROM 表名 [WHERE 删除条件];
```

示例

删除课程表中课程编号为1的数据

```
DELETE FROM course WHERE `number`=1;
```

5. TRUNCATE语句

```
-- 清空表中数据
TRUNCATE [TABLE] 表名;
```

示例

清空课程表数据

```
TRUNCATE course;
```

6. DELETE与TRUNCATE区别

- DELETE语句根据条件删除表中数据，而TRUNCATE语句则是将表中数据全部清空；如果DELETE语句要删除表中所有数据，那么在效率上要低于TRUNCATE语句。
- 如果表中有自增长列，TRUNCATE语句会重置自增长的计数器，但DELETE语句不会。
- TRUNCATE语句执行后，数据无法恢复，而DELETE语句执行后，可以使用事务回滚进行恢复。

第二节 DQL语句

1. 什么是DQL

DQL全称是Data Query Language，表示数据查询语言。体现在数据的查询操作上，因此，DQL仅包括SELECT语句。

2. SELECT语句

```
SELECT ALL/DISTINCT * | 字段名1 AS 别名1[,字段名1 AS 别名1, ..., 字段名n AS 别名n]
FROM 表名 WHERE 查询条件
```

解释说明

ALL表示查询所有满足条件的记录，可以省略；DISTINCT表示去掉查询结果中重复的记录

AS可以给数据列、数据表取一个别名

示例： 从课程表中查询课程编号小于5的课程名称

```
SELECT name FROM course WHERE `number`<5;
```

从课程表中查询课程名称为"Java基础"的学分和学时

```
SELECT score, `time` FROM course WHERE name='Java基础';
```

3. 比较操作符

操作符	语法	说明
IS NULL	字段名 IS NULL	如果字段的值为NULL，则条件满足
IS NOT NULL	字段名 IS NOT NULL	如果字段的值不为NULL，则条件满足
BETWEEN...AND	字段名 BETWEEN 最小值 AND 最大值	如果字段的值在最小值与最大值之间（能够取到最小值和最大值），则条件满足
LIKE	字段名 LIKE '%匹配内容%'	如果字段值包含有匹配内容，则条件满足
IN	字段名 IN(值1，值2，...， 值n)	如果字段值在值1,值2, ...，值n中，则条件满足

示例：从课程表查询课程名为NULL的课程信息

```
SELECT * FROM course WHERE name IS NULL;
```

示例：从课程表查询课程名不为NULL的课程信息

```
SELECT * FROM course WHERE name IS NOT NULL;
```

示例：从课程表查询学分在2~4之间的课程信息

```
SELECT * FROM course WHERE score BETWEEN 2 AND 4;
```

示例：从课程表查询课程名包含"V"的课程信息

```
SELECT * FROM course WHERE name LIKE '%v%';
```

示例：从课程表查询课程名以"J"开头的课程信息

```
SELECT * FROM course WHERE name LIKE 'J%';
```

示例：从课程表查询课程名以"p"结尾的课程信息

```
SELECT * FROM course WHERE name LIKE '%p';
```

示例：从课程表查询课程编号为1,3,5的课程信息

```
SELECT * FROM course WHERE `number` IN (1, 3, 5);
```

4. 分组

数据表准备：新建学生表student，包含字段学号（no），类型为长整数，长度为20，是主键，自增长，非空；姓名（name），类型为字符串，长度为20，非空；性别（sex），类型为字符串，长度为2，默认值为"男"；年龄（age），类型为整数，长度为3，默认值为0；成绩（score），类型为浮点数，长度为5，小数点后面保留2位有效数字

```
DROP TABLE IF EXISTS student;
CREATE TABLE student(
    no BIGINT(20) AUTO_INCREMENT NOT NULL PRIMARY KEY COMMENT '学号，主键',
    name VARCHAR(20) NOT NULL COMMENT '姓名',
    sex VARCHAR(2) DEFAULT '男' COMMENT '性别',
    age INT(3) DEFAULT 0 COMMENT '年龄',
    score DOUBLE(5, 2) COMMENT '成绩'
)ENGINE=InnoDB CHARSET=UTF8 COMMENT='学生表';
```

插入测试数据：

```
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '张三', '男', 20,
59);
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '李四', '女', 19,
62);
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '王五', '其他',
21, 62);
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '龙华', '男', 22,
75);
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '金凤', '女', 18,
80);
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '张华', '其他',
27, 88);
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '李刚', '男', 30,
88);
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '潘玉明', '女',
28, 81);
INSERT INTO student(no, name, sex, age, score) VALUES (DEFAULT, '凤飞飞', '其他',
32, 90);
```

4.1 分组查询

```
SELECT ALL/DISTINCT * | 字段名1 AS 别名1[,字段名1 AS 别名1, ..., 字段名n AS 别名n]
FROM 表名 WHERE 查询条件 GROUP BY 字段名1，字段名2,..., 字段名n
```

分组查询所得的结果只是该组中的第一条数据。

示例：从学生表查询成绩在80分以上的学生信息并按性别分组

```
SELECT * FROM student WHERE score>80 GROUP BY sex;
```

示例：从学生表查询成绩在60~80之间的学生信息并按性别和年龄分组

```
SELECT * FROM student WHERE score BETWEEN 60 AND 80 GROUP BY sex, age;
```

4.2 聚合函数

- **COUNT()：**统计满足条件的数据总条数

  示例：从学生表查询成绩在80分以上的学生人数

  ```
  SELECT COUNT(*) total FROM student WHERE score>80;
  ```

- **SUM()：**只能用于数值类型的字段或者表达式，计算该满足条件的字段值的总和

  示例：从学生表查询不及格的学生人数和总成绩

  ```
  SELECT COUNT(*) totalCount, SUM(score) totalScore FROM student WHERE
  score<60;
  ```

- **AVG()：**只能用于数值类型的字段或者表达式，计算该满足条件的字段值的平均值

  示例：从学生表查询男生、女生、其他类型的学生的平均成绩

  ```
  SELECT sex, AVG(score) avgScore FROM student GROUP BY sex;
  ```

- **MAX()：**只能用于数值类型的字段或者表达式，计算该满足条件的字段值的最大值

  示例：从学生表查询学生的最大年龄

  ```
  SELECT MAX(age) FROM student;
  ```

- **MIN()：**只能用于数值类型的字段或者表达式，计算该满足条件的字段值的最小值

  示例：从学生表查询学生的最低分

  ```
  SELECT MIN(score) FROM student;
  ```

4.3 分组查询结果筛选

```
SELECT ALL/DISTINCT * | 字段名1 AS 别名1[,字段名1 AS 别名1, ..., 字段名n AS 别名n]
FROM 表名 WHERE 查询条件 GROUP BY 字段名1，字段名2,..., 字段名n HAVING 筛选条件
```

分组后如果还需要满足其他条件，则需要使用HAVING子句来完成。

示例：从学生表查询年龄在20~30之间的学生信息并按性别分组，找出组内平均分在74分以上的组

```
SELECT * FROM student WHERE age BETWEEN 20 AND 30 GROUP BY sex HAVING
avg(score)>74;
```

5. 排序

```
SELECT ALL/DISTINCT * | 字段名1 AS 别名1[,字段名1 AS 别名1, ..., 字段名n AS 别名n]
FROM 表名 WHERE 查询条件 ORDER BY 字段名1 ASC|DESC，字段名2 ASC|DESC,..., 字段名n
ASC|DESC
```

ORDER BY 必须位于WHERE 条件之后。

示例：从学生表查询年龄在18~30岁之间的学生信息并按成绩从高到低排列，如果成绩相同，则按年龄从小到大排列

```
SELECT * FROM student WHERE age BETWEEN 18 AND 30 ORDER BY score DESC, age ASC;
```

6. 分页

```
SELECT ALL/DISTINCT * | 字段名1 AS 别名1[,字段名1 AS 别名1, ..., 字段名n AS 别名n]
FROM 表名 WHERE 查询条件 LIMIT 偏移量, 查询条数
```

LIMIT的第一个参数表示偏移量，也就是跳过的行数。

LIMIT的第二个参数表示查询返回的最大行数，可能没有给定的数量那么多行。

示例：从学生表分页查询成绩及格的学生信息，每页显示3条，查询第2页学生信息

```
SELECT * FROM student WHERE score>=60 LIMIT 3, 3;
```

注意：

如果一个查询中包含分组、排序和分页，那么它们之间必须按照**分组->排序->分页**的先后顺序排列。