数据操纵：插入、更新和删除数据

准备工作

- 创建数据库 stu_info

```
CREATE DATABASE stu_info;
USE stu_info;

CREATE TABLE `student` (`id` INT(4) PRIMARY KEY,
                        `name` VARCHAR(20) NOT NULL,
                        `grade` FLOAT);
```

一、插入数据

基本语法

```
INSERT INTO [<db_name>.]<table_name> (column1, column2, ...)
VALUES (value1,value2, ...)[,(...)];
```

> 注意：
>
> - 字段名可以省略，但是此时要按照数据定义时的字段顺序插入值；
> - 可以一次插入多行，各行之间用 , 隔开。

或者

```
INSERT INTO [<db_name>.]<table_name>
SET column1=value1, column2=value2, ...;
```

示例1：运用 `INSERT` 语句插入数据时，指定所有字段名

```
INSERT INTO student(`id`, `name`, `grade`)
VALUES(2, 'zhangsan', 98.5);

SELECT * FROM student; -- 查看student表中的所有数据
```

示例2：运用 `INSERT` 语句插入数据时，指定所有字段名，打乱顺序

```
INSERT INTO student(`name`, `grade`, `id`)
VALUES('lisi', 95, 2);

SELECT * FROM student;
```

示例3：运用 `INSERT` 语句插入数据时，不指定字段名,需严格按字段的顺序提供值

```
INSERT INTO student
VALUES(3, 'wangwu', 61.5);

SELECT * FROM student;

INSERT INTO student
VALUES('wanqi', 4, 61.5); # 如果不按指定顺序，则可能发生插入错误
```

示例4：运用 `INSERT` 语句插入数据时，向指定字段中添加值。

```
INSERT INTO student(`id`, `name`)
VALUES(4, 'zhaoliu');

SELECT * FROM student;
```

示例5：运用 `INSERT` 语句插入数据时，向指定字段中添加值。(注意字段的约束)

```
-- name字段上的非空约束
INSERT INTO student(`id`,`grade`)
VALUES(5,'97');
-- 主键约束
INSERT INTO student(`id`, `name`, `grade`)
VALUES(4, 'longli', 50); # 键值4已存在

INSERT INTO student(`name`, `grade`)
VALUES('xiaofang', 99); # 未设置auto_increment的情况下，必须提供主键值
```

- `AUTO_INCREMENT` 主键字段值的插入
  请完成以下试验

```
 -- 创建表temp
 CREATE TABLE temp (id int primary key auto_increment,
                    name varchar(20));

 -- 然后尝试以下操作理解auto-increment约束的相关运行规则
 INSERT INTO temp (name)
 VALUES ('Huang');  -- id为1，说明初始值为1

```

```sql
INSERT INTO temp
VALUES (3, 'Jone');  -- id为3, 说明可以直接指定值

INSERT INTO temp(name)
VALUES ('Zhang');  -- id为4, 说明是在前一个值的基础上加1

DELETE FROM temp
WHERE id = 4;

INSERT INTO temp(name)
VALUES ('Du');  -- id为5, 说明是在前一个值的基础上加1

SELECT * FROM temp;

INSERT INTO temp(id, name)
VALUES (2, 'Liu'); -- 插入成功, 可以直接指定值

TRUNCATE temp;  -- 截断表, 把所有行和相关操作记录清空

INSERT INTO temp(name)
VALUES ('Huang'); -- 插入成功, id值为1
```

示例6：运用 `INSERT ... SET` 语句为表中指定字段或全部字段添加数据

```sql
INSERT INTO student
SET `id`=5, `name`='boya', `grade`=99;
```

示例7：运用 `INSERT` 语句为 `student` 表中所有字段添加三条数据

```sql
INSERT INTO student
VALUES (6,'lilei',99),
(7,'hanmei',100),
(8,'poly',40.5);
```

示例8：运用 `INSERT` 语句为 `student` 表中指定字段增加多条记录

```sql
INSERT INTO student (`id`,`name`)
VALUES (9,'liubei'),
       (10,'guanyu'),
       (11,'zhangfei');
```

课堂练习1

往 `product` 表中插入以下数据

product_id	product_name	product_code	product_place	product_date	price	unit	detail	subsort_id	sort_id
1035	'商务型U盘128M'	1314027	'上海'	'2010/9/10'	325	'片'	'1片*1盒'	'1314'	'13'
1048	'索尼CD-RW刻录盘'	1314040	'上海'	'2012/12/1'	15	'片'	'1片*1盒'	'1314'	'13'
1058	'LG刻录机'	1314050	'惠州'	'2015/3/9'	410	'台'	'1*1'	'1314'	'13'
1100	'东芝2868复印机'	1101011	'日本'	'2016/6/20'	17250	'台'	'1*1'	'1101'	'11'
1170	'柯达牌喷墨专用纸'	'2205027'	'美国'	'2012/12/12'	56	'包'	'100张/10包/箱'	'2205'	'22'

二、更新数据

基本语法：

```
UPDATE [<db_name>.]<table_name>
SET column1=value1, column2=value2, ....
[WHERE子句];
```

> 注意：
>
> - 如果不加where子句，将更新表中所有行对应的字段值
> - MySQL默认模式是安全更新模式，在该模式下，会导致非主键条件下无法执行更新。要把MySQL数据库设置为非安全更新模式

示例10：更新 `student` 表中 `id` 字段值为1的记录。将该记录的 `name` 字段的值更新为 `'caocao'` ， `grade` 字段的值更新为 `50` 。

```
USE stu_grade;

SELECT * FROM student WHERE `id` = 1;

UPDATE student
SET `name` = 'caocao', `grade` = 50
WHERE `id`=1;

SELECT * FROM student WHERE `id` = 1;
```

示例11：更新 `student` 表中 `id` 字段值小于 `4` 的记录。将些记录的 `grade` 字段的值都更新为 `100`

```
 SELECT * FROM student WHERE `id` < 4;

 UPDATE student
 SET `grade` = 100
 WHERE id < 4;

 SELECT * FROM student WHERE `id` < 4;

 -- MySQL默认模式是安全更新模式，在该模式下，会导致非主键条件下无法执行更新。要把MySQL数据库设置为非安全更新模式
 SET SQL_SAFE_UPDATES=0;  -- 重置SQL模式
 UPDATE student
 SET `grade`=90
 WHERE `grade`=100;
```

课堂练习2

(1) 将 `product` 数据表中产地字段值 `'惠州'` 改为 `'广东'` ；

(2) 将 `product` 数据表中所有产品的价格下调 `10%` ；

(3) 将 `product` 数据表中 `2012` 以后的产品价格增加 `20%` 。

> 注意：此练习用到 `year` 函数，提示： `year(Product_Date)`

三、删除数据

基本语法：

```
DELETE FROM [<db_name>.]<table_name>
[WHERE子句];
```

> 注意：

```
- 如果未定义`where`子句，将逐一删除所有行；
```

示例12：在 `student_bak1` 表中，删除 `id` 字段值为 `11` 的记录。

```
-- 首先将student表复制为student_bak1表
USE stu_info;
CREATE TABLE student_bak1 SELECT * FROM student;

DELETE FROM student_bak1
WHERE `id` = 11;

SELECT * FROM student_bak1;
```

示例13：在 `student_bak1` 表中，删除 `id` 字段值大于 `5` 的所有记录。

```
DELETE FROM student_bak1
WHERE `id` > 5;

SELECT * FROM student_bak1;
```

示例14：删除 `student_bak1` 表中所有记录。

```
DELETE FROM student_bak1;
```

- `TRUNCATE`
  清空所有数据表所有数据

```
TRUNCATE <table_name>;
```

示例15：在 `student_bak2` 表中，删除 `id` 字段值2到5之间的所有记录(其中值包含2，不包含5)。

```
-- 首先将student表复制为student_bak2表。
CREATE TABLE student_bak2 SELECT * FROM student;

SELECT * FROM student_bak2;

DELETE FROM student_bak2
WHERE `id` >= 2 and `grade` < 5;
```

示例16：使用 `TRUNCATE` 删除 `student_bak2` 表中所有记录。

```
TRUNCATE TABLE student_bak2;

SELECT * FROM student_bak2;
```

- `DELETE` 和 `TRUNCATE` 完全删除表记录的区别：
  - (1) 使用 `TRUNCATE` 语句删除表中的数据，再向表中添加记录时，自动增加字段的默认初始值重新由1开始，使用 `DELETE` 语句删除表中所有记录，再向表中添加记录时，自动增加字段的值为删除时该字段的最大值加

1(*因为 `TRUNCATE` 把操作日志删除了*);

- (2) 使用 `DELETE` 语句时，每删除一条记录都会在日志中记录，而使用 `TRUNCATE` 语句时，不会在日志中记录删除的内容，因此 `TRUNCATE` 语句的执行效率比 `DELETE` 语句高。

课堂练习3

(1) 删除 `product` 数据表中产地字段为'美国'的记录；

(2) 删除 `product` 数据表中产品的价格在100和500之间的记录；

(3) 删除 `product` 数据表中上半年生产的产品（注意：使用 `month` 函数）；

(4) 用 `TRUNCATE` 语句删除 `product_bak` 数据表中所有记录。