6. 表的增删改查

CRUD : Create(创建), Retrieve(读取)，Update(更新)，Delete（删除）

6.1 Create

语法:

```
INSERT [INTO] table_name
    [(column [, column] ...)]
    VALUES (value_list) [, (value_list)] ...

value_list: value, [, value] ...
```

案例:

```
-- 创建一张学生表
CREATE TABLE students (
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    sn INT NOT NULL UNIQUE COMMENT '学号',
    name VARCHAR(20) NOT NULL,
    qq VARCHAR(20)
);
```

6.1.1 单行数据 + 全列插入

```
-- 插入两条记录，value_list 数量必须和定义表的列的数量及顺序一致
-- 注意，这里在插入的时候，也可以不用指定id(当然，那时候就需要明确插入数据到那些列了)，那么mysql会使用默认的值进行自增。
INSERT INTO students VALUES (100, 10000, '唐三藏', NULL);
Query OK, 1 row affected (0.02 sec)

INSERT INTO students VALUES (101, 10001, '孙悟空', '11111');
Query OK, 1 row affected (0.02 sec)

-- 查看插入结果
SELECT * FROM students;
+-----+-------+-----------+-------+
| id  | sn    | name      | qq    |
+-----+-------+-----------+-------+
| 100 | 10000 | 唐三藏    | NULL  |
| 101 | 10001 | 孙悟空    | 11111 |
+-----+-------+-----------+-------+
2 rows in set (0.00 sec)
```

6.1.2 多行数据 + 指定列插入

```
-- 插入两条记录，value_list 数量必须和指定列数量及顺序一致
```

```
INSERT INTO students (id, sn, name) VALUES
    (102, 20001, '曹孟德'),
    (103, 20002, '孙仲谋');
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

-- 查看插入结果
SELECT * FROM students;
+-----+-------+-----------+-------+
| id  | sn    | name      | qq    |
+-----+-------+-----------+-------+
| 100 | 10000 | 唐三藏     | NULL  |
| 101 | 10001 | 孙悟空     | 11111 |
| 102 | 20001 | 曹孟德     | NULL  |
| 103 | 20002 | 孙仲谋     | NULL  |
+-----+-------+-----------+-------+
4 rows in set (0.00 sec)
```

6.1.3 插入否则更新

由于 **主键** 或者 **唯一键** 对应的值已经存在而导致插入失败

```
-- 主键冲突
INSERT INTO students (id, sn, name) VALUES (100, 10010, '唐大师');
ERROR 1062 (23000): Duplicate entry '100' for key 'PRIMARY'

-- 唯一键冲突
INSERT INTO students (sn, name) VALUES (20001, '曹阿瞒');
ERROR 1062 (23000): Duplicate entry '20001' for key 'sn'
```

可以选择性的进行同步更新操作 语法：

```
INSERT ... ON DUPLICATE KEY UPDATE
    column = value [, column = value] ...
```

```
INSERT INTO students (id, sn, name) VALUES (100, 10010, '唐大师')
    ON DUPLICATE KEY UPDATE sn = 10010, name = '唐大师';
Query OK, 2 rows affected (0.47 sec)

-- 0 row affected:      表中有冲突数据，但冲突数据的值和 update 的值相等
-- 1 row affected:      表中没有冲突数据，数据被插入
-- 2 row affected:      表中有冲突数据，并且数据已经被更新

-- 通过 MySQL 函数获取受到影响的数据行数
SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           2 |
+-------------+
```

```
1 row in set (0.00 sec)

-- ON DUPLICATE KEY 当发生重复key的时候
```

6.1.4 替换

```
-- 主键 或者 唯一键 没有冲突，则直接插入；
-- 主键 或者 唯一键 如果冲突，则删除后再插入

REPLACE INTO students (sn, name) VALUES (20001, '曹阿瞒');
Query OK, 2 rows affected (0.00 sec)

-- 1 row affected:      表中没有冲突数据，数据被插入
-- 2 row affected:      表中有冲突数据，删除后重新插入
```

6.2 Retrieve

语法：

```
SELECT
    [DISTINCT] {* | {column [, column] ...}
    [FROM table_name]
    [WHERE ...]
    [ORDER BY column [ASC | DESC], ...]
    LIMIT ...
```

案例：

```
-- 创建表结构
CREATE TABLE exam_result (
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL COMMENT '同学姓名',
    chinese float DEFAULT 0.0 COMMENT '语文成绩',
    math float DEFAULT 0.0 COMMENT '数学成绩',
    english float DEFAULT 0.0 COMMENT '英语成绩'
);

-- 插入测试数据
INSERT INTO exam_result (name, chinese, math, english) VALUES
    ('唐三藏', 67, 98, 56),
    ('孙悟空', 87, 78, 77),
    ('猪悟能', 88, 98, 90),
    ('曹孟德', 82, 84, 67),
    ('刘玄德', 55, 85, 45),
    ('孙权', 70, 73, 78),
    ('宋公明', 75, 65, 30);
Query OK, 7 rows affected (0.00 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

6.2.1 SELECT 列

6.2.1.1 全列查询

```
-- 通常情况下不建议使用 * 进行全列查询
-- 1. 查询的列越多，意味着需要传输的数据量越大;
-- 2. 可能会影响到索引的使用。（索引待后面课程讲解）

SELECT * FROM exam_result;
+----+-----------+-------+--------+---------+
| id | name      | chinese | math | english |
+----+-----------+-------+--------+---------+
|  1 | 唐三藏     |    67 |     98 |      56 |
|  2 | 孙悟空     |    87 |     78 |      77 |
|  3 | 猪悟能     |    88 |     98 |      90 |
|  4 | 曹孟德     |    82 |     84 |      67 |
|  5 | 刘玄德     |    55 |     85 |      45 |
|  6 | 孙权       |    70 |     73 |      78 |
|  7 | 宋公明     |    75 |     65 |      30 |
+----+-----------+-------+--------+---------+
7 rows in set (0.00 sec)
```

6.2.1.2 指定列查询

```
-- 指定列的顺序不需要按定义表的顺序来

SELECT id, name, english FROM exam_result;
+----+-----------+---------+
| id | name      | english |
+----+-----------+---------+
|  1 | 唐三藏     |      56 |
|  2 | 孙悟空     |      77 |
|  3 | 猪悟能     |      90 |
|  4 | 曹孟德     |      67 |
|  5 | 刘玄德     |      45 |
|  6 | 孙权       |      78 |
|  7 | 宋公明     |      30 |
+----+-----------+---------+
7 rows in set (0.00 sec)
```

6.2.1.3 查询字段为表达式

```
-- 表达式不包含字段

SELECT id, name, 10 FROM exam_result;
+----+-----------+----+
| id | name      | 10 |
+----+-----------+----+
|  1 | 唐三藏     | 10 |
|  2 | 孙悟空     | 10 |
|  3 | 猪悟能     | 10 |
|  4 | 曹孟德     | 10 |
|  5 | 刘玄德     | 10 |
|  6 | 孙权       | 10 |
```

```
|  7 | 宋公明     | 10 |
+----+-----------+----+
7 rows in set (0.00 sec)
```

```
-- 表达式包含一个字段

SELECT id, name, english + 10 FROM exam_result;
+----+-----------+-------------+
| id | name      | english + 10 |
+----+-----------+-------------+
|  1 | 唐三藏     |          66 |
|  2 | 孙悟空     |          87 |
|  3 | 猪悟能     |         100 |
|  4 | 曹孟德     |          77 |
|  5 | 刘玄德     |          55 |
|  6 | 孙权       |          88 |
|  7 | 宋公明     |          40 |
+----+-----------+-------------+
7 rows in set (0.00 sec)
```

```
-- 表达式包含多个字段

SELECT id, name, chinese + math + english FROM exam_result;
+----+-----------+-------------------------+
| id | name      | chinese + math + english |
+----+-----------+-------------------------+
|  1 | 唐三藏     |                     221 |
|  2 | 孙悟空     |                     242 |
|  3 | 猪悟能     |                     276 |
|  4 | 曹孟德     |                     233 |
|  5 | 刘玄德     |                     185 |
|  6 | 孙权       |                     221 |
|  7 | 宋公明     |                     170 |
+----+-----------+-------------------------+
7 rows in set (0.00 sec)
```

6.2.1.4 为查询结果指定别名

语法:

```
SELECT column [AS] alias_name [...] FROM table_name;
```

```
SELECT id, name, chinese + math + english 总分 FROM exam_result;
+----+-----------+--------+
| id | name      | 总分   |
+----+-----------+--------+
|  1 | 唐三藏     |    221 |
|  2 | 孙悟空     |    242 |
|  3 | 猪悟能     |    276 |
|  4 | 曹孟德     |    233 |
```

```
|  5 | 刘玄德      |    185 |
|  6 | 孙权        |    221 |
|  7 | 宋公明      |    170 |
+----+-----------+--------+
7 rows in set (0.00 sec)
```

6.2.1.5 结果去重

```
-- 98 分重复了

SELECT math FROM exam_result;
+--------+
| math |
+--------+
|     98 |
|     78 |
|     98 |
|     84 |
|     85 |
|     73 |
|     65 |
+--------+
7 rows in set (0.00 sec)
```

```
-- 去重结果

SELECT DISTINCT math FROM exam_result;
+--------+
| math |
+--------+
|     98 |
|     78 |
|     84 |
|     85 |
|     73 |
|     65 |
+--------+
6 rows in set (0.00 sec)
```

6.2.2 WHERE 条件

比较运算符：

运算符	说明
>, >=, <, <=	大于，大于等于，小于，小于等于
=	等于，NULL 不安全，例如 NULL = NULL 的结果是 NULL
<=>	等于，NULL 安全，例如 NULL <=> NULL 的结果是 TRUE(1)
!=, <>	不等于
BETWEEN a0 AND a1	范围匹配，[a0, a1]，如果 a0 <= value <= a1，返回 TRUE(1)
IN (option, ...)	如果是 option 中的任意一个，返回 TRUE(1)
IS NULL	是 NULL
IS NOT NULL	不是 NULL
LIKE	模糊匹配。% 表示任意多个（包括 0 个）任意字符；_ 表示任意一个字符

逻辑运算符：

运算符	说明
AND	多个条件必须都为 TRUE(1)，结果才是 TRUE(1)
OR	任意一个条件为 TRUE(1), 结果为 TRUE(1)
NOT	条件为 TRUE(1)，结果为 FALSE(0)

案例：

6.2.2.1 英语不及格的同学及英语成绩 (< 60)

```
-- 基本比较

SELECT name, english FROM exam_result WHERE english < 60;
+-----------+---------+
| name      | english |
+-----------+---------+
| 唐三藏    |      56 |
| 刘玄德    |      45 |
| 宋公明    |      30 |
+-----------+---------+
3 rows in set (0.01 sec)
```

6.2.2.2 语文成绩在 [80, 90] 分的同学及语文成绩

```
-- 使用 AND 进行条件连接

SELECT name, chinese FROM exam_result WHERE chinese >= 80 AND chinese <= 90;
+-----------+-------+
| name      | chinese |
+-----------+-------+
| 孙悟空    |    87 |
| 猪悟能    |    88 |
| 曹孟德    |    82 |
+-----------+-------+
3 rows in set (0.00 sec)
```

```
-- 使用 BETWEEN ... AND ... 条件

SELECT name, chinese FROM exam_result WHERE chinese BETWEEN 80 AND 90;
+-----------+-------+
| name      | chinese |
+-----------+-------+
| 孙悟空    |    87 |
| 猪悟能    |    88 |
| 曹孟德    |    82 |
+-----------+-------+
3 rows in set (0.00 sec)
```

6.2.2.3 数学成绩是 58 或者 59 或者 98 或者 99 分的同学及数学成绩

```
-- 使用 OR 进行条件连接

SELECT name, math FROM exam_result
    WHERE math = 58
    OR math = 59
    OR math = 98
    OR math = 99;
+-----------+--------+
| name      | math |
+-----------+--------+
| 唐三藏    |     98 |
| 猪悟能    |     98 |
+-----------+--------+
2 rows in set (0.01 sec)
```

```
-- 使用 IN 条件

SELECT name, math FROM exam_result WHERE math IN (58, 59, 98, 99);
+-----------+--------+
| name      | math |
+-----------+--------+
| 唐三藏    |     98 |
| 猪悟能    |     98 |
+-----------+--------+
2 rows in set (0.00 sec)
```

6.2.2.4 姓孙的同学 及 孙某同学

```
-- % 匹配任意多个（包括 0 个）任意字符

SELECT name FROM exam_result WHERE name LIKE '孙%';
+-----------+
| name      |
+-----------+
| 孙悟空    |
| 孙权      |
+-----------+
2 rows in set (0.00 sec)
```

```
-- _ 匹配严格的一个任意字符

SELECT name FROM exam_result WHERE name LIKE '孙_';
+--------+
| name   |
+--------+
| 孙权   |
+--------+
1 row in set (0.00 sec)
```

6.2.2.5 语文成绩好于英语成绩的同学

```
-- WHERE 条件中比较运算符两侧都是字段

SELECT name, chinese, english FROM exam_result WHERE chinese > english;
+-----------+-------+---------+
| name      | chinese | english |
+-----------+-------+---------+
| 唐三藏    |    67 |      56 |
| 孙悟空    |    87 |      77 |
| 曹孟德    |    82 |      67 |
| 刘玄德    |    55 |      45 |
| 宋公明    |    75 |      30 |
+-----------+-------+---------+
5 rows in set (0.00 sec)
```

6.2.2.6 总分在 200 分以下的同学

```
-- WHERE 条件中使用表达式
-- 别名不能用在 WHERE 条件中

SELECT name, chinese + math + english 总分 FROM exam_result
    WHERE chinese + math + english < 200;
+-----------+--------+
| name      | 总分   |
+-----------+--------+
| 刘玄德    |    185 |
| 宋公明    |    170 |
+-----------+--------+
2 rows in set (0.00 sec)
```

6.2.2.7 语文成绩 > 80 并且不姓孙的同学

```
-- AND 与 NOT 的使用

SELECT name, chinese FROM exam_result
    WHERE chinese > 80 AND name NOT LIKE '孙%';
+----+-----------+-------+--------+--------+
| id | name      | chinese | math | english |
+----+-----------+-------+--------+--------+
|  3 | 猪悟能    |    88 |     98 |     90 |
|  4 | 曹孟德    |    82 |     84 |     67 |
+----+-----------+-------+--------+--------+
2 rows in set (0.00 sec)
```

6.2.2.8 孙某同学，否则要求总成绩 > 200 并且 语文成绩 < 数学成绩 并且 英语成绩 > 80

```
-- 综合性查询

SELECT name, chinese, math, english, chinese + math + english 总分
FROM exam_result
WHERE name LIKE '孙_' OR (
    chinese + math + english > 200 AND chinese < math AND english > 80
);
+-----------+-------+--------+--------+--------+
| name      | chinese | math | english | 总分   |
+-----------+-------+--------+--------+--------+
| 猪悟能    |    88 |     98 |     90 |    276 |
| 孙权      |    70 |     73 |     78 |    221 |
+-----------+-------+--------+--------+--------+
2 rows in set (0.00 sec)
```

6.2.2.9 NULL 的查询

```
-- 查询 students 表
+-----+-------+-----------+-------+
| id  | sn    | name      | qq    |
+-----+-------+-----------+-------+
| 100 | 10010 | 唐大师    | NULL  |
```

```
| 101 | 10001 | 孙悟空     | 11111 |
| 103 | 20002 | 孙仲谋     | NULL  |
| 104 | 20001 | 曹阿瞒     | NULL  |
+-----+-------+-----------+-------+
4 rows in set (0.00 sec)

-- 查询 qq 号已知的同学姓名

SELECT name, qq FROM students WHERE qq IS NOT NULL;
+-----------+-------+
| name      | qq    |
+-----------+-------+
| 孙悟空     | 11111 |
+-----------+-------+
1 row in set (0.00 sec)

-- NULL 和 NULL 的比较，= 和 <=> 的区别

SELECT NULL = NULL, NULL = 1, NULL = 0;
+-------------+----------+----------+
| NULL = NULL | NULL = 1 | NULL = 0 |
+-------------+----------+----------+
|        NULL |     NULL |     NULL |
+-------------+----------+----------+
1 row in set (0.00 sec)

SELECT NULL <=> NULL, NULL <=> 1, NULL <=> 0;
+---------------+------------+------------+
| NULL <=> NULL | NULL <=> 1 | NULL <=> 0 |
+---------------+------------+------------+
|             1 |          0 |          0 |
+---------------+------------+------------+
1 row in set (0.00 sec)
```

6.2.3 结果排序

语法：

```
-- ASC 为升序（从小到大）
-- DESC 为降序（从大到小）
-- 默认为 ASC

SELECT ... FROM table_name [WHERE ...]
    ORDER BY column [ASC|DESC], [...];
```

注意：没有 ORDER BY 子句的查询，返回的顺序是未定义的，永远不要依赖这个顺序

案例：

6.2.3.1 同学及数学成绩，按数学成绩升序显示

```
SELECT name, math FROM exam_result ORDER BY math;
+-----------+--------+
| name      | math |
+-----------+--------+
| 宋公明     |     65 |
| 孙权       |     73 |
| 孙悟空     |     78 |
| 曹孟德     |     84 |
| 刘玄德     |     85 |
| 唐三藏     |     98 |
| 猪悟能     |     98 |
+-----------+--------+
7 rows in set (0.00 sec)
```

6.2.3.2 同学及 qq 号，按 qq 号排序显示

```
-- NULL 视为比任何值都小，升序出现在最上面

SELECT name, qq FROM students ORDER BY qq;
+-----------+-------+
| name      | qq    |
+-----------+-------+
| 唐大师     | NULL  |
| 孙仲谋     | NULL  |
| 曹阿瞒     | NULL  |
| 孙悟空     | 11111 |
+-----------+-------+
4 rows in set (0.00 sec)
```

```
-- NULL 视为比任何值都小，降序出现在最下面

SELECT name, qq FROM students  ORDER BY qq DESC;
+-----------+-------+
| name      | qq    |
+-----------+-------+
| 孙悟空     | 11111 |
| 唐大师     | NULL  |
| 孙仲谋     | NULL  |
| 曹阿瞒     | NULL  |
+-----------+-------+
4 rows in set (0.00 sec)
```

6.2.3.3 查询同学各门成绩，依次按 数学降序，英语升序，语文升序的方式显示

```
-- 多字段排序，排序优先级随书写顺序

SELECT name, math, english, chinese FROM exam_result
    ORDER BY math DESC, english, chinese;
+-----------+--------+---------+---------+
| name      | math | english | chinese |
+-----------+--------+---------+---------+
```

```
| 唐三藏      |      98 |      56 |     67 |
| 猪悟能      |      98 |      90 |     88 |
| 刘玄德      |      85 |      45 |     55 |
| 曹孟德      |      84 |      67 |     82 |
| 孙悟空      |      78 |      77 |     87 |
| 孙权        |      73 |      78 |     70 |
| 宋公明      |      65 |      30 |     75 |
+-----------+---------+---------+--------+
7 rows in set (0.00 sec)
```

6.2.3.4 查询同学及总分，由高到低

```
-- ORDER BY 中可以使用表达式

SELECT name, chinese + english + math FROM exam_result
    ORDER BY chinese + english + math DESC;
+-----------+-------------------------+
| name      | chinese + english + math |
+-----------+-------------------------+
| 猪悟能      |                     276 |
| 孙悟空      |                     242 |
| 曹孟德      |                     233 |
| 唐三藏      |                     221 |
| 孙权        |                     221 |
| 刘玄德      |                     185 |
| 宋公明      |                     170 |
+-----------+-------------------------+
7 rows in set (0.00 sec)
```

```
-- ORDER BY 子句中可以使用列别名

SELECT name, chinese + english + math 总分 FROM exam_result
    ORDER BY 总分 DESC;
+-----------+--------+
| name      | 总分   |
+-----------+--------+
| 猪悟能      |    276 |
| 孙悟空      |    242 |
| 曹孟德      |    233 |
| 唐三藏      |    221 |
| 孙权        |    221 |
| 刘玄德      |    185 |
| 宋公明      |    170 |
+-----------+--------+
7 rows in set (0.00 sec)
```

6.2.3.5 查询姓孙的同学或者姓曹的同学数学成绩，结果按数学成绩由高到低显示

```
-- 结合 WHERE 子句 和 ORDER BY 子句

SELECT name, math FROM exam_result
    WHERE name LIKE '孙%' OR name LIKE '曹%'
    ORDER BY math DESC;
+-----------+--------+
| name      | math |
+-----------+--------+
| 曹孟德     |     84 |
| 孙悟空     |     78 |
| 孙权       |     73 |
+-----------+--------+
3 rows in set (0.00 sec)
```

6.2.4 筛选分页结果

语法：

```
-- 起始下标为 0

-- 从 0 开始，筛选 n 条结果
SELECT ... FROM table_name [WHERE ...] [ORDER BY ...] LIMIT n;

-- 从 s 开始，筛选 n 条结果
SELECT ... FROM table_name [WHERE ...] [ORDER BY ...] LIMIT s, n;

-- 从 s 开始，筛选 n 条结果，比第二种用法更明确，建议使用
SELECT ... FROM table_name [WHERE ...] [ORDER BY ...] LIMIT n OFFSET s;
```

建议：对未知表进行查询时，最好加一条 LIMIT 1，避免因为表中数据过大，查询全表数据导致数据库卡死

按 id 进行分页，每页 3 条记录，分别显示 第 1、2、3 页

```
-- 第 1 页

SELECT id, name, math, english, chinese FROM exam_result
    ORDER BY id LIMIT 3 OFFSET 0;
+----+-----------+--------+--------+-------+
| id | name      | math | english | chinese |
+----+-----------+--------+--------+-------+
|  1 | 唐三藏     |     98 |     56 |    67 |
|  2 | 孙悟空     |     78 |     77 |    87 |
|  3 | 猪悟能     |     98 |     90 |    88 |
+----+-----------+--------+--------+-------+
3 rows in set (0.02 sec)
```

```
-- 第 2 页

SELECT id, name, math, english, chinese FROM exam_result
    ORDER BY id LIMIT 3 OFFSET 3;
+----+-----------+--------+--------+-------+
| id | name      | math | english | chinese |
+----+-----------+--------+--------+-------+
|  4 | 曹孟德      |     84 |     67 |    82 |
|  5 | 刘玄德      |     85 |     45 |    55 |
|  6 | 孙权        |     73 |     78 |    70 |
+----+-----------+--------+--------+-------+
3 rows in set (0.00 sec)
```

```
-- 第 3 页，如果结果不足 3 个，不会有影响

SELECT id, name, math, english, chinese FROM exam_result
    ORDER BY id LIMIT 3 OFFSET 6;
+----+-----------+--------+--------+-------+
| id | name      | math | english | chinese |
+----+-----------+--------+--------+-------+
|  7 | 宋公明      |     65 |     30 |    75 |
+----+-----------+--------+--------+-------+
1 row in set (0.00 sec)
```

6.3 Update

语法：

```
UPDATE table_name SET column = expr [, column = expr ...]
    [WHERE ...] [ORDER BY ...] [LIMIT ...]
```

对查询到的结果进行列值更新

案例：

6.3.1 将孙悟空同学的数学成绩变更为 80 分

```
-- 更新值为具体值

-- 查看原数据
SELECT name, math FROM exam_result WHERE name = '孙悟空';
+-----------+--------+
| name      | math |
+-----------+--------+
| 孙悟空      |     78 |
+-----------+--------+
1 row in set (0.00 sec)

-- 数据更新
UPDATE exam_result SET math = 80 WHERE name = '孙悟空';
Query OK, 1 row affected (0.04 sec)
```

```
Rows matched: 1  Changed: 1  Warnings: 0

-- 查看更新后数据
SELECT name, math FROM exam_result WHERE name = '孙悟空';
+-----------+--------+
| name      | math |
+-----------+--------+
| 孙悟空     |     80 |
+-----------+--------+
1 row in set (0.00 sec)
```

6.3.2 将曹孟德同学的数学成绩变更为 60 分，语文成绩变更为 70 分

```
-- 一次更新多个列

-- 查看原数据
SELECT name, math, chinese FROM exam_result WHERE name = '曹孟德';
+-----------+--------+-------+
| name      | math | chinese |
+-----------+--------+-------+
| 曹孟德     |     84 |    82 |
+-----------+--------+-------+
1 row in set (0.00 sec)

-- 数据更新
UPDATE exam_result SET math = 60, chinese = 70 WHERE name = '曹孟德';
Query OK, 1 row affected (0.14 sec)
Rows matched: 1  Changed: 1  Warnings: 0

-- 查看更新后数据
SELECT name, math, chinese FROM exam_result WHERE name = '曹孟德';
+-----------+--------+-------+
| name      | math | chinese |
+-----------+--------+-------+
| 曹孟德     |     60 |    70 |
+-----------+--------+-------+
1 row in set (0.00 sec)
```

6.3.3 将总成绩倒数前三的 3 位同学的数学成绩加上 30 分

```
-- 更新值为原值基础上变更

-- 查看原数据
-- 别名可以在ORDER BY中使用
SELECT name, math, chinese + math + english 总分 FROM exam_result
    ORDER BY 总分 LIMIT 3;
+-----------+--------+--------+
| name      | math | 总分    |
+-----------+--------+--------+
| 宋公明     |     65 |    170 |
| 刘玄德     |     85 |    185 |
| 曹孟德     |     60 |    197 |
```

```
+-----------+--------+--------+
3 rows in set (0.00 sec)

-- 数据更新，不支持 math += 30 这种语法
UPDATE exam_result SET math = math + 30
    ORDER BY chinese + math + english LIMIT 3;

-- 查看更新后数据
-- 思考：这里还可以按总分升序排序取前 3 个么?
SELECT name, math, chinese + math + english 总分 FROM exam_result
    WHERE name IN ('宋公明', '刘玄德', '曹孟德');
+-----------+--------+--------+
| name      | math | 总分   |
+-----------+--------+--------+
| 曹孟德    |     90 |    227 |
| 刘玄德    |    115 |    215 |
| 宋公明    |     95 |    200 |
+-----------+--------+--------+
3 rows in set (0.00 sec)

-- 按总成绩排序后查询结果
SELECT name, math, chinese + math + english 总分 FROM exam_result
    ORDER BY 总分 LIMIT 3;
+-----------+--------+--------+
| name      | math | 总分   |
+-----------+--------+--------+
| 宋公明    |     95 |    200 |
| 刘玄德    |    115 |    215 |
| 唐三藏    |     98 |    221 |
+-----------+--------+--------+
3 rows in set (0.00 sec)
```

6.3.4 将所有同学的语文成绩更新为原来的 2 倍

注意：更新全表的语句慎用！

```
-- 没有 WHERE 子句，则更新全表

-- 查看原数据
SELECT * FROM exam_result;
+----+-----------+-------+--------+--------+
| id | name      | chinese | math | english |
+----+-----------+-------+--------+--------+
|  1 | 唐三藏    |    67 |     98 |     56 |
|  2 | 孙悟空    |    87 |     80 |     77 |
|  3 | 猪悟能    |    88 |     98 |     90 |
|  4 | 曹孟德    |    70 |     90 |     67 |
|  5 | 刘玄德    |    55 |    115 |     45 |
|  6 | 孙权      |    70 |     73 |     78 |
|  7 | 宋公明    |    75 |     95 |     30 |
+----+-----------+-------+--------+--------+
7 rows in set (0.00 sec)
```

```
-- 数据更新
UPDATE exam_result SET chinese = chinese * 2;
Query OK, 7 rows affected (0.00 sec)
Rows matched: 7  Changed: 7  Warnings: 0

-- 查看更新后数据
SELECT * FROM exam_result;
+----+-----------+-------+--------+---------+
| id | name      | chinese | math | english |
+----+-----------+-------+--------+---------+
|  1 | 唐三藏     |   134 |     98 |      56 |
|  2 | 孙悟空     |   174 |     80 |      77 |
|  3 | 猪悟能     |   176 |     98 |      90 |
|  4 | 曹孟德     |   140 |     90 |      67 |
|  5 | 刘玄德     |   110 |    115 |      45 |
|  6 | 孙权       |   140 |     73 |      78 |
|  7 | 宋公明     |   150 |     95 |      30 |
+----+-----------+-------+--------+---------+
7 rows in set (0.00 sec)
```

6.4 Delete

6.4.1 删除数据

语法：

```
DELETE FROM  table_name [WHERE ...] [ORDER BY ...] [LIMIT ...]
```

案例：

6.4.1.1 删除孙悟空同学的考试成绩

```
-- 查看原数据
SELECT * FROM exam_result WHERE name = '孙悟空';
+----+-----------+-------+--------+---------+
| id | name      | chinese | math | english |
+----+-----------+-------+--------+---------+
|  2 | 孙悟空     |   174 |     80 |      77 |
+----+-----------+-------+--------+---------+
1 row in set (0.00 sec)

-- 删除数据
DELETE FROM exam_result WHERE name = '孙悟空';
Query OK, 1 row affected (0.17 sec)

-- 查看删除结果
SELECT * FROM exam_result WHERE name = '孙悟空';
Empty set (0.00 sec)
```

6.4.1.2 删除整张表数据

注意：删除整表操作要慎用！

```
-- 准备测试表
CREATE TABLE for_delete (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(20)
);
Query OK, 0 rows affected (0.16 sec)

-- 插入测试数据
INSERT INTO for_delete (name) VALUES ('A'), ('B'), ('C');
Query OK, 3 rows affected (1.05 sec)
Records: 3  Duplicates: 0  Warnings: 0

-- 查看测试数据
SELECT * FROM for_delete;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
+----+------+
3 rows in set (0.00 sec)
```

```
-- 删除整表数据

DELETE FROM for_delete;
Query OK, 3 rows affected (0.00 sec)

-- 查看删除结果
SELECT * FROM for_delete;
Empty set (0.00 sec)
```

```
-- 再插入一条数据，自增 id 在原值上增长
INSERT INTO for_delete (name) VALUES ('D');
Query OK, 1 row affected (0.00 sec)

-- 查看数据
SELECT * FROM for_delete;
+----+------+
| id | name |
+----+------+
|  4 | D    |
+----+------+
1 row in set (0.00 sec)

-- 查看表结构，会有 AUTO_INCREMENT=n 项
SHOW CREATE TABLE for_delete\G
*************************** 1. row ***************************
       Table: for_delete
Create Table: CREATE TABLE `for_delete` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
```

```
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

6.4.2 截断表

语法：

```
TRUNCATE [TABLE] table_name
```

注意：这个操作慎用

1. 只能对整表操作，不能像 DELETE 一样针对部分数据操作；
2. 实际上 MySQL 不对数据操作，所以比 DELETE 更快，但是TRUNCATE在删除数据的时候，并不经过真正的事物，所以无法回滚
3. 会重置 AUTO_INCREMENT 项

```
-- 准备测试表
CREATE TABLE for_truncate (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(20)
);
Query OK, 0 rows affected (0.16 sec)

-- 插入测试数据
INSERT INTO for_truncate (name) VALUES ('A'), ('B'), ('C');
Query OK, 3 rows affected (1.05 sec)
Records: 3  Duplicates: 0  Warnings: 0

-- 查看测试数据
SELECT * FROM for_truncate;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
+----+------+
3 rows in set (0.00 sec)
```

```
-- 截断整表数据，注意影响行数是 0，所以实际上没有对数据真正操作
TRUNCATE for_truncate;
Query OK, 0 rows affected (0.10 sec)

-- 查看删除结果
SELECT * FROM for_truncate;
Empty set (0.00 sec)
```

```
-- 再插入一条数据，自增 id 在重新增长
INSERT INTO for_truncate (name) VALUES ('D');
Query OK, 1 row affected (0.00 sec)
```

```
-- 查看数据
SELECT * FROM for_truncate;
+----+------+
| id | name |
+----+------+
|  1 | D    |
+----+------+
1 row in set (0.00 sec)

-- 查看表结构，会有 AUTO_INCREMENT=2 项
SHOW CREATE TABLE for_truncate\G
*************************** 1. row ***************************
       Table: for_truncate
Create Table: CREATE TABLE `for_truncate` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

6.5 插入查询结果

语法：

```
INSERT INTO table_name [(column [, column ...])] SELECT ...
```

案例：删除表中的的重复复记录，重复的数据只能有一份

```
-- 创建原数据表

CREATE TABLE duplicate_table (id int, name varchar(20));
Query OK, 0 rows affected (0.01 sec)

-- 插入测试数据
INSERT INTO duplicate_table VALUES
    (100, 'aaa'),
    (100, 'aaa'),
    (200, 'bbb'),
    (200, 'bbb'),
    (200, 'bbb'),
    (300, 'ccc');
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

思路：

```
-- 创建一张空表 no_duplicate_table，结构和 duplicate_table 一样
CREATE TABLE no_duplicate_table LIKE duplicate_table;
Query OK, 0 rows affected (0.00 sec)
```

```
-- 将 duplicate_table 的去重数据插入到 no_duplicate_table
INSERT INTO no_duplicate_table SELECT DISTINCT * FROM duplicate_table;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

-- 通过重命名表，实现原子的去重操作
RENAME TABLE duplicate_table TO old_duplicate_table,
    no_duplicate_table TO duplicate_table;
Query OK, 0 rows affected (0.00 sec)

-- 查看最终结果
SELECT * FROM duplicate_table;
+------+------+
| id   | name |
+------+------+
|  100 | aaa  |
|  200 | bbb  |
|  300 | ccc  |
+------+------+
3 rows in set (0.00 sec)
```

6.6 聚合函数

函数	说明
COUNT([DISTINCT] expr)	返回查询到的数据的 数量
SUM([DISTINCT] expr)	返回查询到的数据的 总和，不是数字没有意义
AVG([DISTINCT] expr)	返回查询到的数据的 平均值，不是数字没有意义
MAX([DISTINCT] expr)	返回查询到的数据的 最大值，不是数字没有意义
MIN([DISTINCT] expr)	返回查询到的数据的 最小值，不是数字没有意义

案例：

6.6.1 统计班级共有多少同学

```
-- 使用 * 做统计，不受 NULL 影响

SELECT COUNT(*) FROM students;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)

-- 使用表达式做统计

SELECT COUNT(1) FROM students;
+----------+
```

```
| COUNT(1) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

6.6.2 统计班级收集的 qq 号有多少

```
-- NULL 不会计入结果

SELECT COUNT(qq) FROM students;
+-----------+
| COUNT(qq) |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)
```

6.6.3 统计本次考试的数学成绩分数个数

```
-- COUNT(math) 统计的是全部成绩

SELECT COUNT(math) FROM exam_result;
+---------------+
| COUNT(math) |
+---------------+
|             6 |
+---------------+
1 row in set (0.00 sec)

-- COUNT(DISTINCT math) 统计的是去重成绩数量
SELECT COUNT(DISTINCT math) FROM exam_result;
+------------------------+
| COUNT(DISTINCT math) |
+------------------------+
|                      5 |
+------------------------+
1 row in set (0.00 sec)
```

6.6.4 统计数学成绩总分

```
SELECT SUM(math) FROM exam_result;
+-------------+
| SUM(math) |
+-------------+
|         569 |
+-------------+
1 row in set (0.00 sec)

-- 不及格 < 60 的总分，没有结果，返回 NULL
SELECT SUM(math) FROM exam_result WHERE math < 60;
+-------------+
```

```
| SUM(math) |
+-------------+
|        NULL |
+-------------+
1 row in set (0.00 sec)
```

6.6.4 统计平均总分

```
SELECT AVG(chinese + math + english) 平均总分 FROM exam_result;
+--------------+
| 平均总分     |
+--------------+
|        297.5 |
+--------------+
```

6.6.5 返回英语最高分

```
SELECT MAX(english) FROM exam_result;
+-------------+
| MAX(english) |
+-------------+
|          90 |
+-------------+
1 row in set (0.00 sec)
```

6.6.6 返回 > 70 分以上的数学最低分

```
SELECT MIN(math) FROM exam_result WHERE math > 70;
+-------------+
| MIN(math) |
+-------------+
|          73 |
+-------------+
1 row in set (0.00 sec)
```

6.7 group by子句的使用

在select中使用group by 子句可以对指定列进行分组查询

```
select column1, column2, .. from table group by column;
```

案例：

- 准备工作，创建一个雇员信息表（来自oracle 9i的经典测试表）
  - EMP员工表
  - DEPT部门表
  - SALGRADE工资等级表
- 如何显示每个部门的平均工资和最高工资

```
select deptno,avg(sal),max(sal) from EMP group by deptno;
```

- 显示每个部门的每种岗位的平均工资和最低工资

```
select avg(sal),min(sal),job, deptno from EMP group by deptno, job;
```

- 显示平均工资低于2000的部门和它的平均工资
  - 统计各个部门的平均工资

```
select avg(sal) from EMP group by deptno
```

  - having和group by配合使用，对group by结果进行过滤

```
select avg(sal) as myavg from EMP group by deptno having myavg<2000;

--having经常和group by搭配使用，作用是对分组进行筛选，作用有些像where。
```

6.8 实战OJ

- 牛客：批量插入数据
- 牛客：找出所有员工当前(to_date='9999-01-01')具体的薪水salary情况，对于相同的薪水只显示一次,并按照逆序显示
- 牛客：查找最晚入职员工的所有信息
- 牛客：查找入职员工时间排名倒数第三的员工所有信息
- 牛客：查找薪水涨幅超过15次的员工号emp_no以及其对应的涨幅次数t
- 牛客：获取所有部门当前manager的当前薪水情况，给出dept_no, emp_no以及salary，当前表示to_date='9999-01-01'
- 牛客：从titles表获取按照title进行分组,每组个数大于等于2，给出title以及对应的数目t
- leetcode: duplicate-emails
- leetcode: big-countries
- leetcode: nth-highest-salary

> 面试题：SQL查询中各个关键字的执行先后顺序 from > on> join > where > group by > with > having > select > distinct > order by > limit