

# MYSQL单表

## 学习目标

1. 能够使用SQL语句操作数据库
2. 能够使用SQL语句操作表结构
3. 能够使用SQL语句进行数据的添加修改和删除的操作
4. 能够使用SQL语句简单查询数据
5. 能够使用SQL语句进行排序
6. 能够使用聚合函数
7. 能够使用SQL语句进行分组查询
8. 能够完成数据的备份和恢复
9. 能够使用SQL语句添加主键、唯一、非空约束

## 第1章 SQL语句

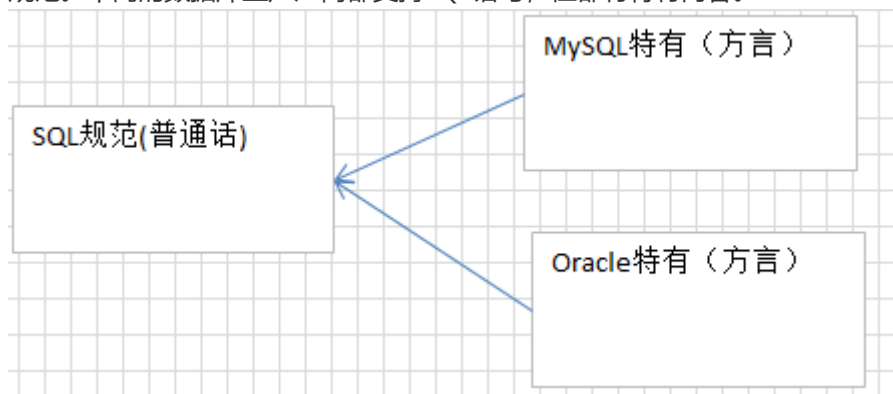
### 1.1 SQL的概念

#### 1.1.1 什么是SQL

结构化查询语言(Structured Query Language)简称SQL,SQL语句就是对数据库进行操作的一种语言。

#### 1.1.2 SQL作用

通过SQL语句我们可以方便的操作数据库中的数据、表、数据库。 SQL是数据库管理系统都需要遵循的规范。不同的数据库生产厂商都支持SQL语句，但都有特有内容。



#### 1.1.2 SQL语句分类

1. DDL(Data Definition Language) 数据定义语言 用来定义数据库对象：数据库，表，列等。关键字： `create`, `drop`, `alter` 等
2. DML(Data Manipulation Language)数据操作语言 用来对数据库中表的数据进行增删改。关键字： `insert`, `delete`, `update` 等
3. DQL(Data Query Language) 数据查询语言 (掌握)  
DQL语言并不是属于MYSQL官方的分类，但是对数据库的操作最多就是查询，所以我们的程序员把查询语句的语句称作为DQL语言
4. DCL(Data Control Language)数据控制语言(了解)

用来定义数据库的访问权限和安全级别，及创建用户。关键字：GRANT，REVOKE 等

## 5. TCL(Transaction Control Language) 事务控制语言

用于控制数据库的事务操作，关键字：COMMIT，SAVEPOINT，ROLLBACK 等

## 1.2 SQL通用语法

1. SQL语句可以单行或多行书写，以分号结尾。
2. 可使用空格和缩进来增强语句的可读性。
3. MySQL数据库的SQL语句不区分大小写，关键字建议使用大写。

```
SELECT * FROM student;
```

4. 3种注释 单行注释: -- 注释内容 或 # 注释内容(mysql特有) 多行注释: /\* 注释 \*/

## 1.3 DDL语句

### 目标

通过DDL操作数据库：创建库，显示库，修改库，删除库操作

通过DDL操作数据库表：创建表，显示表，修改表，删除表操作

通过DDL操作数据库表中的列：添加列，修改列，删除列

### 1.3.1 DDL操作数据库

#### 1.3.1.1 创建数据库

1. 直接创建数据库 CREATE DATABASE 数据库名;
2. 判断是否存在并创建数据库(了解) CREATE DATABASE IF NOT EXISTS 数据库名;
3. 创建数据库并指定字符集(了解) CREATE DATABASE 数据库名 CHARACTER SET 字符集;
4. 具体操作：

- 直接创建数据库db1

```
CREATE DATABASE db1;
```

```
mysql> create database db1;  
Query OK, 1 row affected (0.00 sec)
```

- 判断是否存在并创建数据库db2

```
CREATE DATABASE IF NOT EXISTS db2;
```

```
mysql> CREATE DATABASE IF NOT EXISTS db2;  
Query OK, 1 row affected (0.00 sec)
```

- 创建数据库并指定字符集为gbk

```
CREATE DATABASE db2 CHARACTER SET gbk;
```

```
mysql> CREATE DATABASE db3 CHARACTER SET gbk;  
Query OK, 1 row affected (0.01 sec)
```

### 1.3.1.2 查看数据库

1. 查看所有的数据库 `SHOW databases;`

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| performance_schema |
| test       |
+-----+
4 rows in set (0.00 sec)
```

2. 查看某个数据库的定义信息 `SHOW CREATE DATABASE 数据库名;`

```
mysql> show create database test;
+-----+-----+
| Database | Create Database |
+-----+-----+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+-----+-----+
1 row in set (0.00 sec)
```

### 1.3.1.3 修改数据库(了解)

修改数据库字符集格式

`ALTER DATABASE 数据库名 DEFAULT CHARACTER SET 字符集;`

注意: `DEFAULT` 关键字可以不写

具体操作:

- 将db3数据库的字符集改成utf8

```
ALTER DATABASE db3 DEFAULT CHARACTER SET utf8;
```

```
mysql> SHOW CREATE DATABASE db3;
+-----+-----+
| Database | Create Database |
+-----+-----+
| db3      | CREATE DATABASE `db3` /*!40100 DEFAULT CHARACTER SET gbk */ |
+-----+-----+
1 row in set (0.00 sec)

mysql> ALTER DATABASE db3 DEFAULT CHARACTER SET utf8;
Query OK, 1 row affected (0.00 sec)

mysql> ALTER DATABASE db3 DEFAULT CHARACTER SET utf8;
Query OK, 1 row affected (0.00 sec)

mysql> SHOW CREATE DATABASE db3;
+-----+-----+
| Database | Create Database |
+-----+-----+
| db3      | CREATE DATABASE `db3` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+-----+-----+
1 row in set (0.00 sec)
```

### 1.3.1.4 删除数据库

`DROP DATABASE 数据库名;`

具体操作:

- 删除db2数据库

```
DROP DATABASE db2;
```

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| day24      |
| db2        |
| db3        |
| mysql      |
| performance_schema |
| test       |
+-----+
7 rows in set (0.00 sec)

mysql> DROP DATABASE db2;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| day24      |
| db3        |
| mysql      |
| performance_schema |
| test       |
+-----+
6 rows in set (0.00 sec)

mysql>
```

db2数据库已经被删除了

### 1.3.1.5 使用数据库

1. 查看正在使用的数据库 `SELECT DATABASE();`
2. 使用/切换数据库 `USE 数据库名;`

具体操作:

- 查看正在使用的数据库

```
SELECT DATABASE();
```

```
mysql> select database();
+-----+
| database() |
+-----+
| NULL       |
+-----+
1 row in set (0.00 sec)
```

- 使用db1数据库

```
USE db1;
```

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| NULL      |
+-----+
1 row in set (0.00 sec)

mysql> USE db1;
Database changed
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| db1       |
+-----+
1 row in set (0.00 sec)

mysql>
```

## 小结

数据库操作中使用的语法:

create database 数据库名称; 创建数据库

alter database 数据库名 default character set 字符集; 数据库修改在开发中不建议使用

drop database 数据库名称; 删除数据库

use 数据库名称; 使用数据库

## 1.3.2 DDL操作表

对数据库表进行 创建 查看 修改 删除操作

前提先使用某个数据库

### 1.3.2.1 创建表

```
CREATE TABLE 表名 (字段名1 字段类型1, 字段名2 字段类型2...);
```

建议写成如下格式:

```
CREATE TABLE 表名 (
    字段名1 字段类型1,
    字段名2 字段类型2
);
```

### 1.3.2.2 数据库类型

**MySQL数据类型** MySQL中的我们常使用的数据类型如下:

类型	描述
int	整型
varchar	字符类型
float	浮点数类型（小数）
double	浮点数类型（小数）
decimal	浮点数类型（小数）， decimal类型的字段没有精度丢失的情况，和钱有关的数据都要使用decimal类型，不能使用float和double
date	日期类型，格式： yyyy-MM-dd
datetime	日期类型，格式： yyyy-MM-dd HH:mm:ss， 占用8字节的存储空间
timestamp	日期类型：格式： yyyy-MM-dd HH:mm:ss， 取值范围 1970-2038， 占用4字节存储， 性能要优于datetime

详细的数据类型如下(不建议详细阅读！)

分类	类型名称	说明
整数类型	tinyint	很小的整数
	smallint	小的整数
	mediumint	中等大小的整数
	int	普通大小的整数
小数类型	float	单精度浮点数
	double	双精度浮点数
	decimal (m,d)	压缩严格的定点数
日期类型	year	YYYY 1901~2155
	time	HH:MM:SS -838:59:59~838:59:59
	date	YYYY-MM-DD 1000-01-01~9999-12-3
	datetime	YYYY-MM-DD HH:MM:SS 1000-01-01 00:00:00~ 9999-12-31 23:59:59

分类	类型名称	说明
	timestamp	YYYY-MM-DD HH:MM:SS 1970~01~01 00:00:01 UTC~2038-01-19 03:14:07UTC
文本、二进制类型	CHAR(M)	M为0~255之间的整数
	VARCHAR(M)	M为0~65535之间的整数
	TINYBLOB	允许长度0~255字节
	BLOB	允许长度0~65535字节
	MEDIUMBLOB	允许长度0~167772150字节
	LOBLOB	允许长度0~4294967295字节
	TINYTEXT	允许长度0~255字节
	TEXT	允许长度0~65535字节
	MEDIUMTEXT	允许长度0~167772150字节
	LONGTEXT	允许长度0~4294967295字节
	VARBINARY(M)	允许长度0~M个字节的变长字节字符串
	BINARY(M)	允许长度0~M个字节的定长字节字符串

具体操作:

创建student表包含id,name,birthday字段

```
CREATE TABLE student (
    id INT,
    name VARCHAR(20),
    birthday DATE
);
```

### 1.3.2.3 查看表


1. 查看某个数据库中的所有表 `SHOW TABLES;`
2. 查看表结构 `DESC 表名;`
3. 查看创建表的SQL语句 `SHOW CREATE TABLE 表名;`

具体操作:

- 查看mysql数据库中的所有表

```
SHOW TABLES;
```

```
mysql> use mysql
Database changed
mysql> show tables;
+-----+
| Tables_in_mysql |
+-----+
| columns_priv     |
| db               |
| event            |
| func             |
| general_log      |
| help_category    |
| help_keyword     |
| help_relation    |
| help_topic       |
| host             |
| ndb_binlog_index |
| plugin           |
| proc             |
| procs_priv       |
| proxies_priv     |
| servers          |
| slow_log         |
| tables_priv      |
| time_zone        |
| time_zone_leap_second |
| time_zone_name   |
| time_zone_transition |
| time_zone_transition_type |
| user             |
+-----+
24 rows in set (0.00 sec)
```



- 查看student表的结构

```
DESC student;
```

```
mysql> desc student;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id     | int(11)       | YES  |     | NULL    |       |
| name   | varchar(20)   | YES  |     | NULL    |       |
| birthday | date         | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

- 查看student的创建表SQL语句

```
SHOW CREATE TABLE student;
```



```
mysql> show create table student;
+-----+-----+
| Table | Create Table |
+-----+-----+
| student | CREATE TABLE `student` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `birthday` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----+-----+
1 row in set (0.00 sec)
```

#### 1.3.2.4 快速创建一个表结构相同的表

CREATE TABLE 新表名 LIKE 旧表名;

具体操作:

- 创建s1表, s1表结构和student表结构相同

```
CREATE TABLE s1 LIKE student;
```

```
mysql> create table s1 like student;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW CREATE TABLE s1;
+-----+-----+
| Table | Create Table |
+-----+-----+
| s1 | CREATE TABLE `s1` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `birthday` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----+-----+
1 row in set (0.00 sec)
```

#### 1.3.2.5 删除表

1. 直接删除表 DROP TABLE 表名;
2. 存在则删除表(了解) DROP TABLE IF EXISTS 表名;

具体操作:

- 直接删除表s1表

```
DROP TABLE s1;
```

```
mysql> show tables;
+-----+
| Tables_in_db1 |
+-----+
| s1             |
| s2             |
| s3             |
| student        |
+-----+
4 rows in set (0.00 sec)

mysql> DROP TABLE s1;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+-----+
| Tables_in_db1 |
+-----+
| s2             |
| s3             |
| student        |
+-----+
3 rows in set (0.00 sec)
```

*s1表被删除了*

- 判断表是否存在并删除s1表

```
DROP TABLE IF EXISTS s1;
```

```
mysql> show tables;
+-----+
| Tables_in_db1 |
+-----+
| s2             |
| s3             |
| student        |
+-----+
3 rows in set (0.00 sec)

mysql> DROP TABLE s1;
ERROR 1051 (42S02): Unknown table 's1'
mysql> DROP TABLE IF EXISTS s1;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

*s1表不存在，直接删除会报错*

*增加了判断后表不存在不会报错*

### 1.3.2.6 修改表结构

修改表结构使用不是很频繁，只需要知道下，等需要使用的时候再回来查即可

1. 添加表列 `ALTER TABLE 表名 ADD 列名 类型;`

具体操作：

- 为学生表添加一个新的字段remark,类型为varchar(20)

```
ALTER TABLE student ADD remark VARCHAR(20);
```

```
mysql> desc s1;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)   | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| birthday | date     | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> alter table s1 add remark varchar(20);
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc s1;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)   | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| birthday | date     | YES  |     | NULL    |       |
| remark | varchar(20) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

2. 修改列类型 `ALTER TABLE 表名 MODIFY 列名 新的类型;` 具体操作:

- 将student表中的remark字段的改成varchar(100)

```
ALTER TABLE student MODIFY remark VARCHAR(100);
```

```
mysql> desc s1;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)   | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| birthday | date     | YES  |     | NULL    |       |
| remark | varchar(20) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> alter table s1 modify remark varchar(100);
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc s1;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)   | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| birthday | date     | YES  |     | NULL    |       |
| remark | varchar(100) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

3. 修改列名 `ALTER TABLE 表名 CHANGE 旧列名 新列名 类型;` 具体操作:

- 将student表中的remark字段名改成intro, 类型varchar(30)

```
ALTER TABLE student CHANGE remark intro varchar(30);
```

```
mysql> desc s1;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)       | YES  |     | NULL    |       |
| name  | varchar(20)   | YES  |     | NULL    |       |
| birthday | date        | YES  |     | NULL    |       |
| remark | varchar(100)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> alter table s1 change remark intro varchar(30);
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc s1;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)       | YES  |     | NULL    |       |
| name  | varchar(20)   | YES  |     | NULL    |       |
| birthday | date        | YES  |     | NULL    |       |
| intro | varchar(30)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

4. 删除列 `ALTER TABLE 表名 DROP 列名;` 具体操作:

- 删除student表中的字段intro

```
ALTER TABLE student DROP intro;
```

```
mysql> desc s1;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)       | YES  |     | NULL    |       |
| name  | varchar(20)   | YES  |     | NULL    |       |
| birthday | date        | YES  |     | NULL    |       |
| intro | varchar(30)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

mysql> alter table s1 drop intro;
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc s1;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)       | YES  |     | NULL    |       |
| name  | varchar(20)   | YES  |     | NULL    |       |
| birthday | date        | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

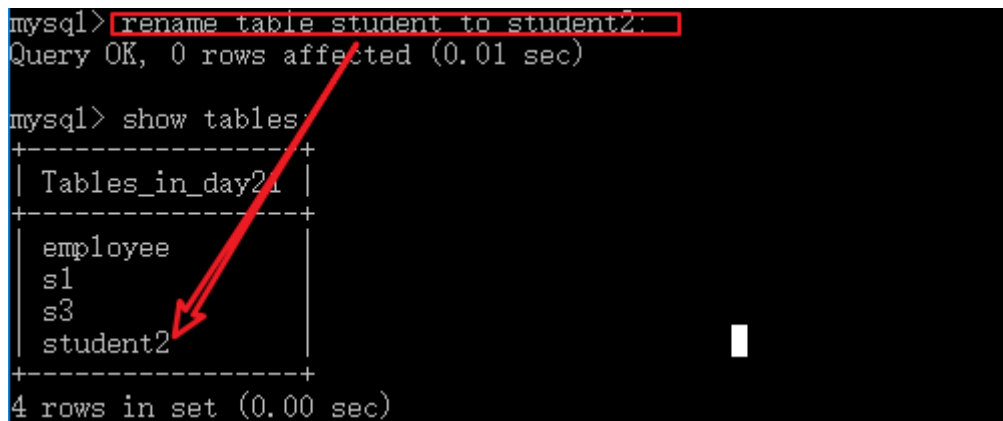
5. 修改表名 `RENAME TABLE 表名 TO 新表名;` 具体操作:

- 将学生表student改名成student2

```
RENAME TABLE student TO student2;
```

```
mysql> rename table student to student2;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+-----+
| Tables_in_day21 |
+-----+
| employee        |
| s1              |
| s3              |
| student2        |
+-----+
4 rows in set (0.00 sec)
```



#### 6. 修改字符集 ALTER TABLE 表名 character set 字符集; 具体操作:

- 将student2表的编码修改成gbk

```
ALTER TABLE student2 character set gbk;
```

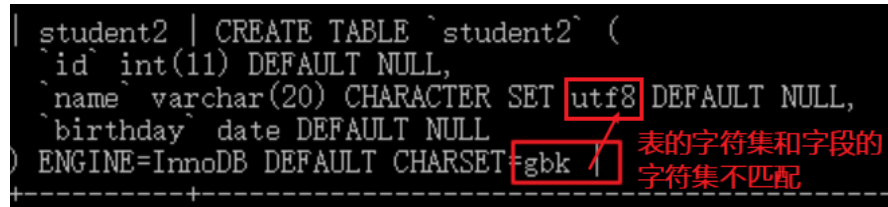
```
mysql> show create table student2;
+-----+-----+
| Table | Create Table
+-----+-----+
| student2 | CREATE TABLE `student2` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `birthday` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
+-----+-----+
1 row in set (0.00 sec)

mysql> alter table student2 character set gbk;
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> show create table student2;
+-----+-----+
| Table | Create Table
+-----+-----+
| student2 | CREATE TABLE `student2` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) CHARACTER SET utf8 DEFAULT NULL,
  `birthday` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=gbk
+-----+-----+
1 row in set (0.00 sec)
```

**注意：**修改表的字符集后，字段的字符集不会自动的修改，企业开发中通常不会修改表的字符集，如果需要修改，也是想将表的数据进行备份，然后删除表，在重新创建表并指定新的字符集，然后在恢复数据。

```
| student2 | CREATE TABLE `student2` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) CHARACTER SET utf8 DEFAULT NULL,
  `birthday` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=gbk
```



表的字符集和字段的字符集不匹配

## 小结

我们想存储数据必须先有表，其次是表结构，当然，在开发过程中，我们很少会修改表，一般都是提前约定好的。所以我们重点学会创建表操作以及类型

创建表: create table 表名 ( 字段名1 字段类型1, 字段名2 字段类型2 ); 查看表 : show tables 快速创建表 : create table 新表 like 旧表; 修改表结构 :

alter table 表名 add 列名 类型 alter table 表名 modify 列名 类型 alter table 表名 change 旧列名 新列名 类型 alter table 表名 drop 列名

## 1.4 DML语句

### 目标

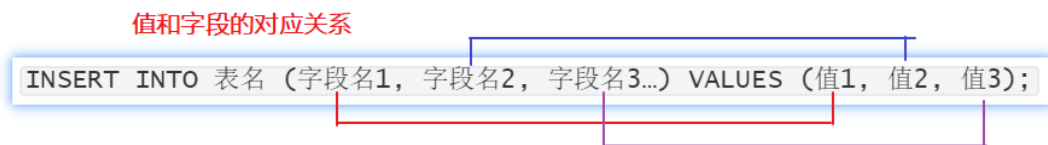
掌握增删改的语法

### 1.4.1 插入记录

#### 1.4.1.1 插入全部字段

- 所有的字段名都写出来 `INSERT INTO 表名 ( 字段名1, 字段名2, 字段名3...) VALUES (值1, 值2, 值3);`

**注意：**列出所有字段名的情况下，values后面值的顺序要和前面字段的顺序顺序要保持一致



- 不写字段名 `INSERT INTO 表名 VALUES (值1, 值2, 值3...);`

**注意：**不写字段名的情况下，values后面值的顺序要和数据内字段的定义顺序要保持一致。

#### 1.4.1.2 插入部分数据

`INSERT INTO 表名 ( 字段名1, 字段名2, ...) VALUES (值1, 值2, ...);` 没有添加数据的字段会使用 NULL

##### 1. 关键字说明

`INSERT INTO` 表名 - 表示往哪张表中添加数据  
( 字段名1, 字段名2, ...) -- 要给哪些字段设置值  
`VALUES` (值1, 值2, ...); -- 设置具体的值

##### 2. 注意

- 值与字段必须对应，个数相同，类型相同
- 值的数据大小必须在字段的长度范围内
- 除了数值类型外，其它的字段类型的值必须使用引号引起。（建议单引号）
- 如果要插入空值，可以不写字段，或者插入null

##### 3. 具体操作:

- 插入部分数据，往学生表中添加 id, name, age, sex数据

```
INSERT INTO student (id, NAME, age, sex) VALUES (1, '张三', 20, '男');
```

```
INSERT INTO student (id, NAME, age, sex) VALUES (1, '张三', 20, '男');
```

○ 向表中插入所有字段

- 所有的字段名都写出来

```
INSERT INTO student (NAME, id, age, sex, address) VALUES ('李四', 2, 23, '女', '广州');
```

```
INSERT INTO student (NAME, id, age, sex, address) VALUES ('李四', 2, 23, '女', '广州');
```

id	name	age	sex	address
1	张三	20	男	(NULL)
2	李四	23	女	广州

所有字段都添加数据

- 不写字段名

```
INSERT INTO student VALUES (3, '王五', 18, '男', '北京');
```

```
-- 不需要写字段名,就是添加所有字段
-- INSERT INTO 表名 VALUES (值1, 值2, 值3...);
INSERT INTO student VALUES (3, '王五', 18, '男', '吉山');
```

id	name	age	sex	address
1	张三	20	男	(NULL)
2	李四	23	女	广州
3	王五	18	男	吉山
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

### 1.4.2.3 批量插入数据

批量插入就是 values 后面跟多组值，有多少组值，后面就会插入多少行数据

```
INSERT INTO 表名 (字段名1, 字段名2, ...) VALUES (值1, 值2, ...), (值3, 值4, ...);
```

-- 一次性插入2行数据

```
INSERT INTO student(id, NAME, birthday) VALUES (10, '小美', '2019-10-01'), (11, '小爱', '2018-10-01');
```

```
INSERT INTO student(id, NAME, birthday) VALUES (10, '小美', '2019-10-01'), (11, '小爱', '2018-10-01');
```

1 信息 2 表数据 3 Info 4 历史

```
(2 row(s) affected)
Execution Time : 00:00:00:078
Transfer Time : 00:00:00:000
Total Time : 00:00:00:078
```

一次性插入2行数据

### 1.4.2.4 DOS命令窗口中文乱码问题

个别同学在使用DOS窗口去操作数据库时，可能会出现中文乱码的情况，出现中文乱码的问题主要是因为客户端的字符集和服务器的字符集不匹配造成的。

查看 MySQL 内部设置的编码 `show variables like 'character%';`

Variable_name	Value
character_set_client	utf8
character_set_connection	utf8
character_set_database	utf8
character_set_filesystem	binary
character_set_results	utf8
character_set_server	utf8
character_set_system	utf8

解决方案：修改client、connection、results的编码为GBK，保证和DOS命令行编码保持一致

#### 1. 单独设置

```
set character_set_client=gbk;
set character_set_connection=gbk;
set character_set_results=gbk;
```

#### 2. 快捷设置

```
set names gbk;
```

注意：以上2种方式为临时方案，退出DOS命令行就失效了，需要每次都配置

后期我们主要是用MySQL的客户端工具，使用工具后，将不会再次类似的问题

### 1.4.2.5 蠕虫复制

什么是蠕虫复制：在已有的数据基础之上，将原来的数据进行复制，插入到对应的表中 语法格式：

`INSERT INTO 表名1 SELECT * FROM 表名2;` 作用:将 表名2 中的数据复制到 表名1 中

具体操作:

- 创建student2表，student2结构和student表结构一样

```
CREATE TABLE student2 LIKE student;
```

- 将student表中的数据添加到student2表中

```
INSERT INTO student2 SELECT * FROM student;
```



注意：如果只想复制student表中name,age字段数据到student2表中使用如下格式 `INSERT INTO student2(NAME, age) SELECT NAME, age FROM student;`

`INSERT INTO student2(NAME, age) SELECT NAME, age FROM student;`

放到目标表的哪些字段      复制原表的哪些字段

id	name	age	sex	address	math	english
1	张三	20	男	(NULL)	67	96
2	李四	23	女	广州	88	92
3	王五	18	男	北京	56	65
4	赵六	28	男	南京	99	98

只复制了指定字段的数据，没有指定字段的数据没有复制

id	name	age	sex	address	math	english
1	张三	20	男	(NULL)	67	96
2	李四	23	女	广州	88	92
3	王五	18	男	北京	56	65
4	赵六	28	男	南京	99	98
(NULL)	张三	20	(NULL)	(NULL)	(NULL)	(NULL)
(NULL)	李四	23	(NULL)	(NULL)	(NULL)	(NULL)
(NULL)	王五	18	(NULL)	(NULL)	(NULL)	(NULL)
(NULL)	赵六	28	(NULL)	(NULL)	(NULL)	(NULL)

## 1.4.2 更新表记录

1. 不带条件修改数据 `UPDATE 表名 SET 字段名=值;`
2. 带条件修改数据 `UPDATE 表名 SET 字段名=值 WHERE 字段名=值;`
3. 关键字说明

**UPDATE**：修改数据  
**SET**：修改哪些字段  
**WHERE**：指定条件

### 4. 具体操作：

- 不带条件修改数据，将所有的性别改成女

```
UPDATE student SET sex='女';
```

修改前

id	name	age	sex	address
1	张三	20	男	(NULL)
2	李四	23	女	广州
3	王五	18	男	北京
4	赵六	28	男	南京

修改后

id	name	age	sex	address
1	张三	20	女	(NULL)
2	李四	23	女	广州
3	王五	18	女	北京
4	赵六	28	女	南京

- 带条件修改数据，将id号为2的学生性别改成男

```
UPDATE student SET sex='男' WHERE id=2;
```

修改前

id	name	age	sex	address
1	张三	20	女	(NULL)
2	李四	23	女	广州
3	王五	18	女	北京
4	赵六	28	女	南京

修改后

id	name	age	sex	address
1	张三	20	女	(NULL)
2	李四	23	男	广州
3	王五	18	女	北京
4	赵六	28	女	南京

- 一次修改多个列，把id为3的学生，年龄改成26岁，address改成北京

```
UPDATE student SET age=26, address='北京' WHERE id=3;
```

修改前

id	name	age	sex	address
1	张三	20	女	(NULL)
2	李四	23	男	广州
3	王五	18	女	北京
4	赵六	28	女	南京

修改后      一次性修改2个字段

id	name	age	sex	address
1	张三	20	女	(NULL)
2	李四	23	男	广州
3	王五	26	女	北京
4	赵六	28	女	南京

## 1.4.3 删除表记录

1. 不带条件删除数据 `DELETE FROM 表名;`
2. 带条件删除数据 `DELETE FROM 表名 WHERE 字段名=值;`

### 3. truncate删除表记录 TRUNCATE TABLE 表名;

truncate和delete的区别:

- delete是将表中的数据一条一条删除
- truncate是将整个表摧毁, 重新创建一个新的表, 新的表结构和原来表结构一模一样

```
mysql> select * from student;
```

id	name	age	sex	address	math	english
1	张三	20	女	(NULL)	67	96
2	李四	23	男	广州	88	92
3	王五	18	男	北京	56	65
4	赵六	28	女	南京	99	98

```
4 rows in set (0.00 sec)

mysql> TRUNCATE TABLE student;
```

Query OK, 0 rows affected (0.01 sec)

```
mysql> select * from student;
```

Empty set (0.00 sec)

mysql>

truncate 也将数据库中的数据全部删除了  
truncate和delete的区别:  
1.delete是将表中的数据一条一条删除  
2.truncate是将整个表摧毁, 重新创建一个新的表

### 4. 具体操作:

- 带条件删除数据, 删除id为3的记录

```
DELETE FROM student WHERE id=3;
```

删除前

id	name	age	sex	address
1	张三	20	女	(NULL)
2	李四	23	男	广州
3	王五	26	女	北京
4	赵六	28	女	南京

删除后

id	name	age	sex	address
1	张三	20	女	(NULL)
2	李四	23	男	广州
4	赵六	28	女	南京

满足条件的记录被删除了

- 不带条件删除数据, 删除表中的所有数据

```
DELETE FROM student;
```

删除前

id	name	age	sex	address
1	张三	20	女	(NULL)
2	李四	23	男	广州
4	赵六	28	女	南京

删除后

id	name	age	sex	address
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

所有的记录都被删除

## 小结

增加

insert into 表名(列名1, 列名2, 列名3 ...) values (值1, 值2, 值3)

insert into 表名 values (值1, 值2, 值3...) 此语法需要将所有字段添加上

修改

update 表名 set 列名=值, 列名=值 ...

update 表名 set 列名=值, 列名=值 ... where 条件(用于锁定数据)

删除

delete from 表名

delete from 表名 where 条件(用于锁定数据)

## 1.5 DQL

### 目标

在真正开发中，我们的查询操作比增删改操作更多，而且查询也比增删改更加复杂，所以，我们务必需要将查询拿下，而查询分为单表和多表查询。我们先熟练应用单表查询，再实现多表，多表是鉴于单表查询的基础之上，所以需要同学们好好练习。

查询不会对数据库中的数据进行修改.只是一种显示数据的方式 准备数据

```
CREATE TABLE student3 (  
    id int,  
    name varchar(20),  
    age int,  
    sex varchar(5),  
    address varchar(100),  
    math int,  
    english int  
);
```

```
INSERT INTO student3(id,NAME,age,sex,address,math,english) VALUES (1,'马云',55,'男','杭州',66,78),(2,'马化腾',45,'女','深圳',98,87),(3,'马景涛',55,'男','香港',56,77),(4,'柳岩',20,'女','湖南',76,65),(5,'柳青',20,'男','湖南',86,NULL),(6,'刘德华',57,'男','香港',99,99),(7,'马德',22,'女','香港',99,99),(8,'德玛西亚',18,'男','南京',56,65);
```

### 1.5.1 简单查询

#### 1.5.1.1 查询表所有数据

1. 使用 \* 表示所有列 `SELECT * FROM 表名;` 具体操作:

```
SELECT * FROM student3;
```

	id	NAME	age	sex	address	math	english
<input type="checkbox"/>	1	马云	55	男	杭州	66	78
<input type="checkbox"/>	2	马化腾	45	女	深圳	98	87
<input type="checkbox"/>	3	马景涛	55	男	香港	56	77
<input type="checkbox"/>	4	柳岩	20	女	湖南	76	65
<input type="checkbox"/>	5	柳青	20	男	湖南	86	(NULL)
<input type="checkbox"/>	6	刘德华	57	男	香港	99	99
<input type="checkbox"/>	7	马德	22	女	香港	99	99
<input type="checkbox"/>	8	德玛西亚	18	男	南京	56	65

2. 写出查询每列的名称 `SELECT 字段名1, 字段名2, 字段名3, ... FROM 表名;` 具体操作:

```
SELECT id, NAME ,age, sex, address, math, english FROM student3;
```

<input type="checkbox"/>	id	NAME	age	sex	address	math	english
<input type="checkbox"/>	1	马云	55	男	杭州	66	78
<input type="checkbox"/>	2	马化腾	45	女	深圳	98	87
<input type="checkbox"/>	3	马景涛	55	男	香港	56	77
<input type="checkbox"/>	4	柳岩	20	女	湖南	76	65
<input type="checkbox"/>	5	柳青	20	男	湖南	86	(NULL)
<input type="checkbox"/>	6	刘德华	57	男	香港	99	99
<input type="checkbox"/>	7	马德	22	女	香港	99	99
<input type="checkbox"/>	8	德玛西亚	18	男	南京	56	65

### 1.5.1.2 查询指定列

查询指定列的数据,多个列之间以逗号分隔 `SELECT 字段名1, 字段名2... FROM 表名;`

具体操作: 查询student3表中的id , name , age , sex , address 列

```
SELECT id, NAME ,age, sex, address FROM student3;
```

<input type="checkbox"/>	id	NAME	age	sex	address
<input type="checkbox"/>	1	马云	55	男	杭州
<input type="checkbox"/>	2	马化腾	45	女	深圳
<input type="checkbox"/>	3	马景涛	55	男	香港
<input type="checkbox"/>	4	柳岩	20	女	湖南
<input type="checkbox"/>	5	柳青	20	男	湖南
<input type="checkbox"/>	6	刘德华	57	男	香港
<input type="checkbox"/>	7	马德	22	女	香港
<input type="checkbox"/>	8	德玛西亚	18	男	南京

### 1.5.1.3 别名查询

1. 查询时给列、表指定别名需要使用AS关键字

2. 使用别名的好处是方便观看和处理查询到的数据 `SELECT 字段名1 AS 别名, 字段名2 AS 别名... FROM 表名;` `SELECT 字段名1 AS 别名, 字段名2 AS 别名... FROM 表名 AS 表别名;` 注意:

查询给表取别名目前还看不到效果, 需要到多表查询的时候才能体现出好处 AS关键字可以省略

3. 具体操作:

◦ 查询student3表中name 和 age 列, name列的别名为"姓名", age列的别名为"年龄"

```
SELECT NAME AS 姓名,age 年龄 FROM student3;
```

<input type="checkbox"/>	NAME	age
<input type="checkbox"/>	马云	55
<input type="checkbox"/>	马化腾	45
<input type="checkbox"/>	马景涛	55
<input type="checkbox"/>	柳岩	20
<input type="checkbox"/>	柳青	20
<input type="checkbox"/>	刘德华	57
<input type="checkbox"/>	马德	22
<input type="checkbox"/>	德玛西亚	18

无别名

<input type="checkbox"/>	姓名	年龄
<input type="checkbox"/>	马云	55
<input type="checkbox"/>	马化腾	45
<input type="checkbox"/>	马景涛	55
<input type="checkbox"/>	柳岩	20
<input type="checkbox"/>	柳青	20
<input type="checkbox"/>	刘德华	57
<input type="checkbox"/>	马德	22
<input type="checkbox"/>	德玛西亚	18

有别名

使用别名的好处是方便观看和处理查询到的数据

- 查询student3表中name和age列，student3表别名为s

```
SELECT NAME, age FROM student3 AS s;
```

<input type="checkbox"/>	NAME	age
<input type="checkbox"/>	马云	55
<input type="checkbox"/>	马化腾	45
<input type="checkbox"/>	马景涛	55
<input type="checkbox"/>	柳岩	20
<input type="checkbox"/>	柳青	20
<input type="checkbox"/>	刘德华	57
<input type="checkbox"/>	马德	22
<input type="checkbox"/>	德玛西亚	18

查询给表取别名目前还看不到效果，需要到多表查询的时候才能体现出好处

#### 1.5.1.4 清除重复值

为了演示重复值,我们再插入一条记录

```
INSERT INTO student3 (id,NAME,age) VALUES(9,'马云',12);
```

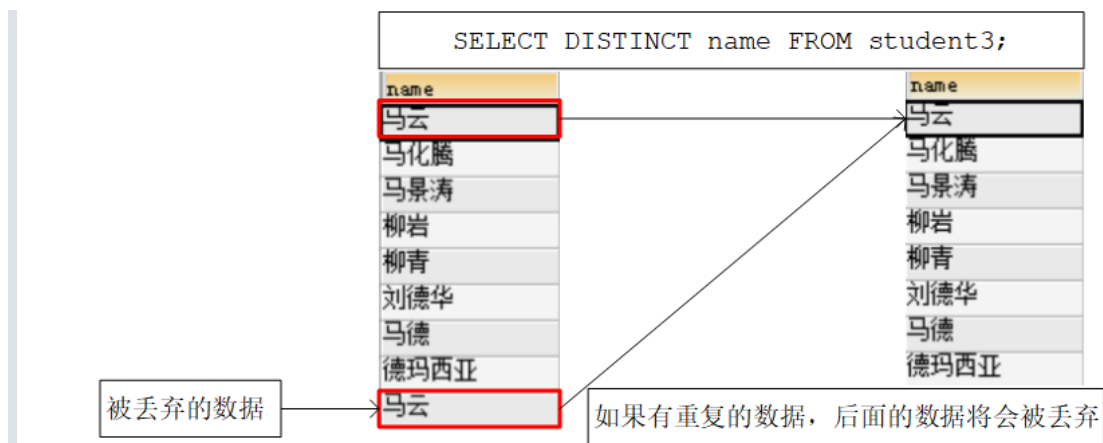
语法: `select distinct 字段... from 表;`

清除重复值的关键字为 `distinct`，一个 `select` 语句只能有一个 `distinct`，而且只能写在 `select` 关键字的后面

它的执行流程为，是先将查询出来字段的数据进行组合，然后在对组合的结果进行检查是否存在重复值，如果存在则只保留一条

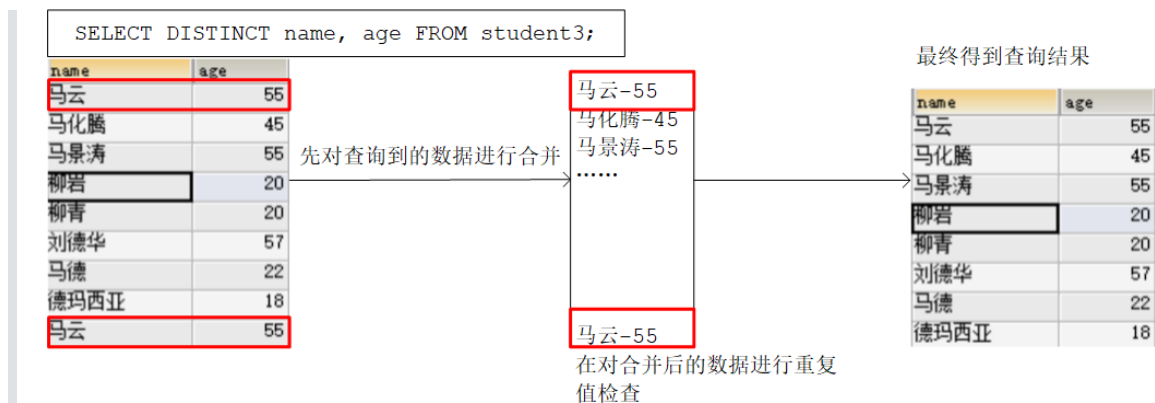
##### 1) distinct后面为1个字段的情况

```
select DISTINCT NAME FROM student3;
```



## 2) distinct后面的字段为多个的情况

```
SELECT DISTINCT NAME, age FROM student3;
```



测试完成之后，把刚刚插入的记录删除

```
DELETE FROM student3 WHERE id = 9;
```

### 1.5.1.5 查询结果参与运算

1. 某列数据和固定值运算 `SELECT 列名1 + 固定值 FROM 表名;`
2. 某列数据和其他列数据参与运算 `SELECT 列名1 + 列名2 FROM 表名;`

注意: 参与运算的必须是数值类型

3. 具体例子:

- 查询每个人的总成绩

```
SELECT NAME, math + english FROM student3;
```

name	math + english
马云	144
马化腾	185
马景涛	133
柳岩	141
柳青	(NULL)
刘德华	198
马德	198
德玛西亚	121

结果确实将每条记录的math和english相加，有两个小问题

- 1.效果不好看
- 2.柳青的成绩为NULL

我们来把这两个问题解决下

- 查询math + english的和并使用别名“总成绩”

```
SELECT NAME, math + english 总成绩 FROM student3;
```

NAME	总成绩
马云	144
马化腾	185
马景涛	133
柳岩	141
柳青	(NULL)
刘德华	198
马德	198
德玛西亚	121

- NULL值处理

注意：NULL参与算术运算结果还是NULL。

IFNULL(表达式1,表达式2)：若表达式1的值为NULL,则返回表达式2的结果;若表达式1的值不为NULL,则返回表达式1的值

因为english一列中有null值，我们可以使用IFNULL解决，写法 IFNULL(english,0)，意思为若english值为NULL，按0处理；否则还是使用原来english的值。

```
SELECT NAME, math + IFNULL(english,0) 总成绩 FROM student3;
```

NAME	总成绩
马云	144
马化腾	185
马景涛	133
柳岩	141
柳青	86
刘德华	198
马德	198
德玛西亚	121

- 查询所有列与math + english的和并使用别名“总成绩”

```
SELECT *, math + IFNULL(english,0) 总成绩 FROM student3;
```

id	name	age	sex	address	math	english	总成绩
1	马云	55	男	杭州	66	78	144
2	马化腾	45	女	深圳	98	87	185
3	马景涛	55	男	香港	56	77	133
4	柳岩	20	女	湖南	76	65	141
5	柳青	20	男	湖南	86	(NULL)	86
6	刘德华	57	男	香港	99	99	198
7	马德	22	女	香港	99	99	198
8	德玛西亚	18	男	南京	56	65	121

- 查询姓名、年龄，将每个人的年龄增加10岁

```
SELECT NAME, age + 10 FROM student3;
```

<input type="checkbox"/>	NAME	age + 10
<input type="checkbox"/>	马云	65
<input type="checkbox"/>	马化腾	55
<input type="checkbox"/>	马景涛	65
<input type="checkbox"/>	柳岩	30
<input type="checkbox"/>	柳青	30
<input type="checkbox"/>	刘德华	67
<input type="checkbox"/>	马德	32
<input type="checkbox"/>	德玛西亚	28

## 小结

不管是什么查询，我们都不会修改原表的数据，而是在结果中获得自己需要的数据。

# 第2章 DQL语句增强

## 2.1 条件查询

前面我们的查询都是将所有数据都查询出来，但是有时候我们只想获取到 满足条件 的数据 语法格式：

`SELECT 字段名 FROM 表名 WHERE 条件;` 流程：取出表中的每条数据，满足条件的记录就返回，不满足条件的记录不返回

### 2.1.1 NULL值查询

作为一个程序员判断一个变量是否为 null 时，习惯性的使用 `变量 == null` 方式来判断，但是在数据库中并不能使用这种方式检测某个字段是否存在 null 值

数据库中提供了特定的语法

**查询字段值等于null的数据:** `字段 is null`

查询 english 字段值等于 null 的数据

```
SELECT * FROM student3 WHERE english IS NULL;
```

id	name	age	sex	address	math	english
5	柳青	20	男	湖南	86	(NULL)

**查询字段值不等于null的数据:** `字段 is not null`

查询 english 字段值不等于 null 的数据

```
SELECT * FROM student3 WHERE english IS NOT NULL;
```



id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

## 2.1.1 比较运算符

> 大于 < 小于 <= 小于等于 >= 大于等于 = 等于 <>、!= 不等于

具体操作：

- 查询math分数大于80分的学生

```
SELECT * FROM student3 WHERE math>80;
```

id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87
6	刘德华	57	男	香港	99	99

- 查询english分数小于或等于80分的学生

```
SELECT * FROM student3 WHERE english<=80;
```

id	name	age	sex	address	math	english
1	马云	51	男	杭州	66	78
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65

- 查询age等于20岁的学生

```
SELECT * FROM student3 WHERE age=20;
```

id	name	age	sex	address	math	english
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	(NULL)	(NULL)

- 查询age不等于20岁的学生

```
SELECT * FROM student3 WHERE age!=20;
SELECT * FROM student3 WHERE age<>20;
```

id	name	age	sex	address	math	english
1	马云	51	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
6	刘德华	57	男	香港	99	99

## 2.1.2 逻辑运算符

and(&&) 多个条件同时满足 or(||) 多个条件其中一个满足 not(!) 不满足

具体操作:

- 查询age大于35且性别为男的学生(两个条件同时满足)

```
SELECT * FROM student3 WHERE age>35 AND sex='男';
```

id	name	age	sex	address	math	english
1	马云	51	男	杭州	66	78
3	马景涛	55	男	香港	56	77
6	刘德华	57	男	香港	99	99

age大于35且性别为男的学生，两个条件同时满足)

- 查询age大于35或性别为男的学生(两个条件其中一个满足)

```
SELECT * FROM student333 WHERE age>35 OR sex='男';
```

id	name	age	sex	address	math	english
1	马云	51	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
5	柳青	20	男	湖南	(NULL)	(NULL)
6	刘德华	57	男	香港	99	99

age大于35或性别为男的学生，最少其中一个条件满足

- 查询id是1或3或5的学生

```
SELECT * FROM student3 WHERE id=1 OR id=3 OR id=5;
```

id	name	age	sex	address	math	english
1	马云	51	男	杭州	66	78
3	马景涛	55	男	香港	56	77
5	柳青	20	男	湖南	(NULL)	(NULL)

in关键字 语法格式: SELECT 字段名 FROM 表名 WHERE 字段 in (数据1, 数据2...); in里面的每个数据都会作为一次条件，只要满足条件的就会显示

具体操作:

- 查询id是1或3或5的学生

```
SELECT * FROM student3 WHERE id IN (1,3,5);
```

id	name	age	sex	address	math	english
1	马云	51	男	杭州	66	78
3	马景涛	55	男	香港	56	77
5	柳青	20	男	湖南	(NULL)	(NULL)

- 查询id不是1或3或5的学生

```
SELECT * FROM student3 WHERE id NOT IN (1,3,5);
```

id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87
4	柳岩	20	女	湖南	76	65
6	刘德华	57	男	香港	99	99

### 2.1.3 范围

`BETWEEN 值1 AND 值2` 表示从值1到值2范围，包头又包尾 比如：`age BETWEEN 80 AND 100` 相当于：`age>=80 && age<=100`

具体操作：

- 查询english成绩大于等于75，且小于等于90的学生

```
SELECT * FROM student3 WHERE english>=75 AND english<=90;
SELECT * FROM student3 WHERE english BETWEEN 75 AND 90;
```

id	name	age	sex	address	math	english
1	马云	51	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77

### 2.1.4 like

`LIKE` 表示模糊查询 `SELECT * FROM 表名 WHERE 字段名 LIKE '通配符字符串'`；满足 通配符字符串 规则的数据就会显示出来 所谓的 通配符字符串 就是 含有通配符的字符串 MySQL通配符有两个：`%`: 表示0个或多个字符(任意个字符) `_`: 表示一个字符

具体操作：

- 查询姓马的学生

```
SELECT * FROM student3 WHERE NAME LIKE '马%';
```

id	name	age	sex	address	math	english
1	马云	51	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77

- 查询姓名中包含'德'字的学生

```
SELECT * FROM student3 WHERE NAME LIKE '%德%';
```

id	name	age	sex	address	math	english
6	刘德华	57	男	香港	99	99
7	马德	22	女	(NULL)	(NULL)	(NULL)
8	德玛西亚	18	男	(NULL)	(NULL)	(NULL)

- 查询姓马，且姓名有三个字的学生

```
SELECT * FROM student3 WHERE NAME LIKE '马__';
```

id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77

马开头，后面还有2个字符

## 2.2 排序

通过 `ORDER BY` 子句，可以将查询出的结果进行排序(排序只是显示方式，不会影响数据库中数据的顺序) `SELECT 字段名 FROM 表名 WHERE 字段=值 ORDER BY 字段名 [ASC|DESC]`; ASC: 升序, 默认是升序  
DESC: 降序

### 2.2.1 单列排序

单列排序就是使用一个字段排序

具体操作:

- 查询所有数据,使用年龄降序排序

```
SELECT * FROM student3 ORDER BY age DESC;
```

id	name	age	sex	address	math	english
6	刘德华	57	男	香港	99	99
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
2	马化腾	45	女	深圳	98	87
7	马德	22	女	香港	99	99
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
8	德玛西亚	18	男	南京	56	65

### 2.2.2 组合排序

组合排序就是先按第一个字段进行排序，如果第一个字段相同，才按第二个字段进行排序，依次类推。上面的例子中，年龄是有相同的。当年龄相同再使用math进行排序 `SELECT 字段名 FROM 表名 WHERE 字段=值 ORDER BY 字段名1 [ASC|DESC], 字段名2 [ASC|DESC]`;

具体操作:

- 查询所有数据,在年龄降序排序的基础上，如果年龄相同再以数学成绩降序排序

```
SELECT * FROM student3 ORDER BY age DESC, math DESC;
```

id	name	age	sex	address	math	english
6	刘德华	57	男	香港	99	99
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
2	马化腾	45	女	深圳	98	87
7	马德	22	女	香港	99	99
5	柳青	20	男	湖南	86	(NULL)
4	柳岩	20	女	湖南	76	65
8	德玛西亚	18	男	南京	56	65

## 2.3 聚合函数

之前我们做的查询都是横向查询，它们都是根据条件一行一行的进行判断，而使用聚合函数查询是纵向查询，它是对一列的值进行计算，然后返回一个结果值。另外聚合函数会忽略空值

五个聚合函数：`count`：统计指定列记录数，记录为NULL的不统计 `sum`：计算指定列的数值和，如果不是数值类型，那么计算结果为0 `max`：计算指定列的最大值 `min`：计算指定列的最小值 `avg`：计算指定列的平均值，如果不是数值类型，那么计算结果为0

聚合函数的使用：写在 SQL 语句 `SELECT` 后 字段名 的地方 `SELECT 字段名... FROM 表名;` `SELECT COUNT(age) FROM 表名;`

具体操作：

- 查询学生总数

```
SELECT COUNT(english) FROM student3;
```

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

COUNT(english)
7

对于NULL的记录不会统计

我们发现对于NULL的记录不会统计

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

直接english查询这个字段的数据

```
SELECT english FROM student3;
```

english
78
87
77
65
(NULL)
99
99
65

查询english字段数据，如果为NULL则使用0

```
SELECT IFNULL(english,0) FROM student3;
```

IFNULL(english,0)
78
87
77
65
0
99
99
65

我们可以利用IFNULL()函数，如果记录为NULL，给个默认值，这样统计的数据就不会遗漏

```
SELECT COUNT(IFNULL(english,0)) FROM student3;
```

COUNT(IFNULL(english,0))
8

```
SELECT COUNT(*) FROM student3;
```

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

COUNT(*)
8

- 查询年龄大于40的总数

```
SELECT COUNT(*) FROM student3 WHERE age>40;
```

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

COUNT (\*)

4

- 查询数学成绩总分

```
SELECT SUM(math) FROM student3;
```

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

SUM(math)

636

- 查询数学成绩平均分

```
SELECT AVG(math) FROM student3;
```

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

AVG(math)

79.5000

- 查询数学成绩最高分

```
SELECT MAX(math) FROM student3;
```

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

MAX(math)

99

- 查询数学成绩最低分

```
SELECT MIN(math) FROM student3;
```

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

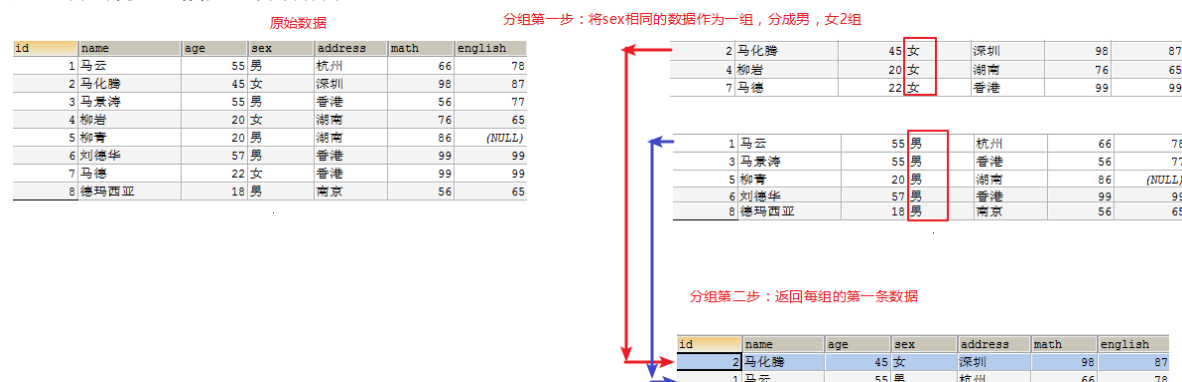
MIN (math)  
56

## 2.4 分组

分组查询是指使用 `GROUP BY` 语句对查询信息进行分组，相同数据作为一组 `SELECT 字段1, 字段2...`  
`FROM 表名 GROUP BY 分组字段 [HAVING 条件];`

GROUP BY怎么分组的？将分组字段结果中相同内容作为一组 `SELECT * FROM student3 GROUP BY sex;`

这句话会将sex相同的数据作为一组



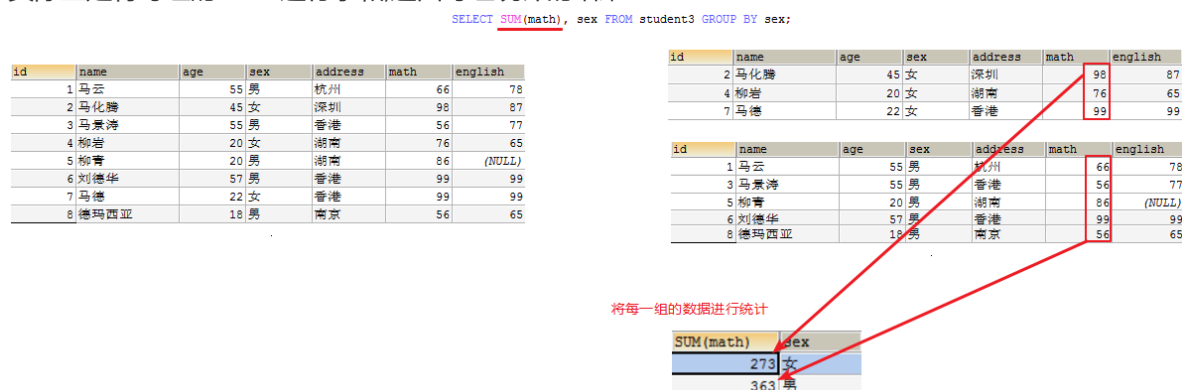
`GROUP BY` 将分组字段结果中相同内容作为一组，并且返回每组的第一条数据，所以单独分组没什么用处。分组的目的是为了统计，一般分组会跟聚合函数一起使用。

分组后聚合函数的作用不是操作所有数据，而是操作一组数据。 `SELECT SUM(math), sex FROM`

`student3 GROUP BY sex;` 效果如下：

SUM(math)	sex
273	女
363	男

实际上是将每组的math进行求和,返回每组统计的结果



注意事项：当我们使用某个字段分组,在查询的时候也需要将这个字段查询出来,否则看不到数据属于哪组的

- 查询的时候没有查询出分组字段

`SELECT SUM(math) FROM student3 GROUP BY sex;`

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87
4	柳岩	20	女	湖南	76	65
7	马德	22	女	香港	99	99

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
8	德玛西亚	18	男	南京	56	65

SUM(math)
273
363

不知道数据是属于哪组的？

- 查询的时候查询出分组字段

`SELECT SUM(math), sex FROM student3 GROUP BY sex;`

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87
4	柳岩	20	女	湖南	76	65
7	马德	22	女	香港	99	99

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
8	德玛西亚	18	男	南京	56	65

SUM(math)	sex
273	女
363	男

具体实现：

- 按性别分组

`SELECT sex FROM student3 GROUP BY sex;`

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

sex
女
男

相同的数据作为一组

- 查询男女各多少人

1. 查询所有数据,按性别分组。
2. 统计每组人数

`SELECT sex, COUNT(*) FROM student3 GROUP BY sex;`

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

sex	COUNT(*)
女	3
男	5

- 查询年龄大于25岁的人,按性别分组,统计每组的人数

1. 先过滤掉年龄小于25岁的人。
2. 再分组。
3. 最后统计每组的人数

`SELECT sex, COUNT(*) FROM student3 WHERE age > 25 GROUP BY sex;`



```
SELECT sex, COUNT(*) FROM student3 WHERE age > 25 GROUP BY sex;
```

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

### 1.先过滤掉年龄小于25岁的人。

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
6	刘德华	57	男	香港	99	99

### 2.再分组。

id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
6	刘德华	57	男	香港	99	99

### 3.最后统计每组的人数

sex	COUNT(*)
女	1
男	3

- 查询年龄大于25岁的人,按性别分组,统计每组的人数,并只显示性别人数大于2的数据 有很多同学可能会将SQL语句写出这样: `SELECT sex, COUNT(*) FROM student3 WHERE age > 25 GROUP BY sex WHERE COUNT(*) >2;`

注意: "显示性别人数大于2的数据" 属于分组后的条件,对于分组后的条件需要使用 `having` 子句

```
SELECT sex, COUNT(*) FROM student3 WHERE age > 25 GROUP BY sex HAVING COUNT(*) >2;
```

只有分组后人数大于2的`男`这组数据显示出来

查询年龄大于25岁的人,按性别分组,统计每组的人数,并只显示性别人数大于2的数据

```
SELECT sex, COUNT(*) FROM student3 WHERE age > 25 GROUP BY sex HAVING COUNT(*) >2;
```

分组前的条件 → **WHERE age > 25** ← 分组后的条件

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

### 1.先过滤掉年龄小于25岁的人。

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
6	刘德华	57	男	香港	99	99

### 2.再分组。

id	name	age	sex	address	math	english
2	马化腾	45	女	深圳	98	87

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
3	马景涛	55	男	香港	56	77
6	刘德华	57	男	香港	99	99

### 3.统计每组的人数

sex	COUNT(*)
女	1
男	3

### 4.显示性别人数大于2的数据

sex	COUNT(*)
男	3

having与where的区别

- having是在分组后对数据进行过滤.
- where是在分组前对数据进行过滤
- having后面可以使用聚合函数
- where后面不可以使用聚合函数

## 2.5 limit语句

`LIMIT` 是 限制 的意思, 所以 `LIMIT` 的作用就是限制查询记录的条数。

现在表内的数据不多,为了更好的演示 `LIMIT` 语句,这里我们先初始化一下数据

```
INSERT INTO student3(id,NAME,age,sex,address,math,english) VALUES
(9,'唐僧',25,'男','长安',87,78),
(10,'孙悟空',18,'男','花果山',100,66),
(11,'猪八戒',22,'男','高老庄',58,78),
(12,'沙僧',50,'男','流沙河',77,88),
(13,'白骨精',22,'女','白虎岭',66,66),
(14,'蜘蛛精',23,'女','盘丝洞',88,88);
```

SELECT \* | 字段列表 [as 别名] FROM 表名 [WHERE子句] [GROUP BY子句] [HAVING子句] [ORDER BY子句] [LIMIT子句]; 思考: limit子句为什么排在最后? 因为前面所有的限制条件都处理完了, 只剩下显示多少条记录的问题了!

**LIMIT语法格式:** LIMIT offset,length; 或者LIMIT length; offset 是指跳过的记录数量 ( 学名: 偏移量 ), 默认为0 length 是指需要显示的总记录数

具体步骤:

- 查询学生表中数据, 从第三条开始显示, 显示6条

我们可以认为跳过前面2条, 取6条数据

```
SELECT * FROM student3 LIMIT 2,6;
```

所有数据

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65
9	唐僧	25	男	长安	87	78
10	孙悟空	18	男	花果山	100	66
11	猪八戒	22	男	高老庄	58	78
12	沙僧	50	男	流沙河	77	88
13	白骨精	22	女	白虎岭	66	66
14	蜘蛛精	23	女	盘丝洞	88	88

SELECT \* FROM student3 LIMIT 2,6;

跳过2条记录 → 显示6条记录

id	name	age	sex	address	math	english
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65

**LIMIT的使用场景:** 分页 比如我们登录京东, 淘宝, 返回的商品信息可能有几万条, 不是一次全部显示出来。是一页显示固定的条数。假设我们一每页显示5条记录的方式来分页, SQL语句如下:

- 每页显示5条
- 第一页: LIMIT 0,5; 跳过0条, 显示5条
- 第二页: LIMIT 5,5; 跳过5条, 显示5条
- 第三页: LIMIT 10,5; 跳过10条, 显示5条

```
SELECT * FROM student3 LIMIT 0,5;
SELECT * FROM student3 LIMIT 5,5;
SELECT * FROM student3 LIMIT 10,5;
```

所有数据

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65
9	唐僧	25	男	长安	87	78
10	孙悟空	18	男	花果山	100	66
11	猪八戒	22	男	高老庄	58	78
12	沙僧	50	男	流沙河	77	88
13	白骨精	22	女	白虎岭	66	66
14	蜘蛛精	23	女	盘丝洞	88	88

SELECT \* FROM student3 LIMIT 0,5;

id	name	age	sex	address	math	english
1	马云	55	男	杭州	66	78
2	马化腾	45	女	深圳	98	87
3	马景涛	55	男	香港	56	77
4	柳岩	20	女	湖南	76	65
5	柳青	20	男	湖南	86	(NULL)

SELECT \* FROM student3 LIMIT 5,5;

id	name	age	sex	address	math	english
6	刘德华	57	男	香港	99	99
7	马德	22	女	香港	99	99
8	德玛西亚	18	男	南京	56	65
9	唐僧	25	男	长安	87	78
10	孙悟空	18	男	花果山	100	66

SELECT \* FROM student3 LIMIT 10,5;

id	name	age	sex	address	math	english
11	猪八戒	22	男	高老庄	58	78
12	沙僧	50	男	流沙河	77	88
13	白骨精	22	女	白虎岭	66	66
14	蜘蛛精	23	女	盘丝洞	88	88

注意: LIMIT 10, 5; -- 不够5条, 有多少显示多少

注意：

- 如果第一个参数是0可以简写：`SELECT * FROM student3 LIMIT 0,5;` `SELECT * FROM student3 LIMIT 5;`
- `LIMIT 10, 5;` -- 不够5条，有多少显示多少

## 第3章 数据库备份

### 目标

完成数据的备份操作

### 3.1 备份的应用场景

在服务器进行数据传输、数据存储和数据交换，就有可能产生数据故障。比如发生意外停机或存储介质损坏。这时，如果没有采取数据备份和数据恢复手段与措施，就会导致数据的丢失，造成的损失是无法弥补与估量的。

### 3.2 source命令备份与还原

**备份格式：** `mysqldump -u用户名 -p密码 数据库 > 文件的路径`

**还原格式：** `SOURCE 导入文件的路径;`

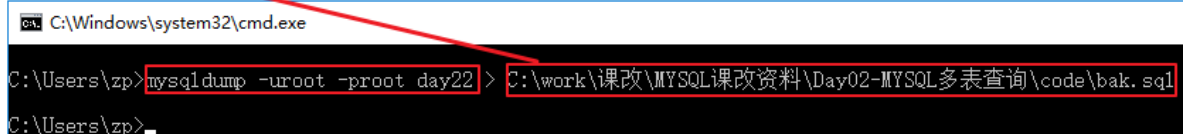
注意：还原的时候需要先登录MySQL,并选中对应的数据库

具体操作：

- 备份day22数据库中的数据

```
mysqldump -uroot -proot day22 > C:\work\课改\MYSQL课改资料\Day02-MYSQL多表查询\code\bak.sql
```

1.sql	2018/1/26 18:23	SQL 文件	5 KB
bak.sql	2018/1/26 18:23	SQL 文件	4 KB



```
C:\Windows\system32\cmd.exe
C:\Users\zp>mysqldump -uroot -proot day22 > C:\work\课改\MYSQL课改资料\Day02-MYSQL多表查询\code\bak.sql
C:\Users\zp>
```

## 数据库中的所有表和数据都会导出成SQL语句

day22  
表  
employee  
student3

数据库中的所有表都会导出

```
DROP TABLE IF EXISTS `employee`;  
/*!40101 SET @saved_cs_client      = @@character_set_client */;  
/*!40101 SET character_set_client = utf8 */;  
CREATE TABLE `employee` (  
  `id` int(11) DEFAULT NULL,  
  `name` varchar(20) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
/*!40101 SET character_set_client = @saved_cs_client */;  
DROP TABLE IF EXISTS `student3`;  
/*!40101 SET @saved_cs_client      = @@character_set_client */;  
/*!40101 SET character_set_client = utf8 */;  
CREATE TABLE `student3` (  
  `id` int(11) DEFAULT NULL,  
  `name` varchar(20) DEFAULT NULL,  
  `age` int(11) DEFAULT NULL,  
  `sex` varchar(5) DEFAULT NULL,  
  `address` varchar(100) DEFAULT NULL,  
  `math` int(11) DEFAULT NULL,  
  `english` int(11) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
/*!40101 SET character set client = @saved cs client */;
```

- 还原day22数据库中的数据

- 删除day22数据库中的所有表

```
mysql> drop table employee;  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> drop table student3;  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> show tables; 删除day22数据库中的所有表  
Empty set (0.00 sec)
```

- 登录MySQL

```
mysql -uroot -proot
```

- 选中数据库

```
use day22;  
select database();
```

```
mysql> select database();  
+-----+  
| database() |  
+-----+  
| day22      |  
+-----+
```

- 使用SOURCE命令还原数据

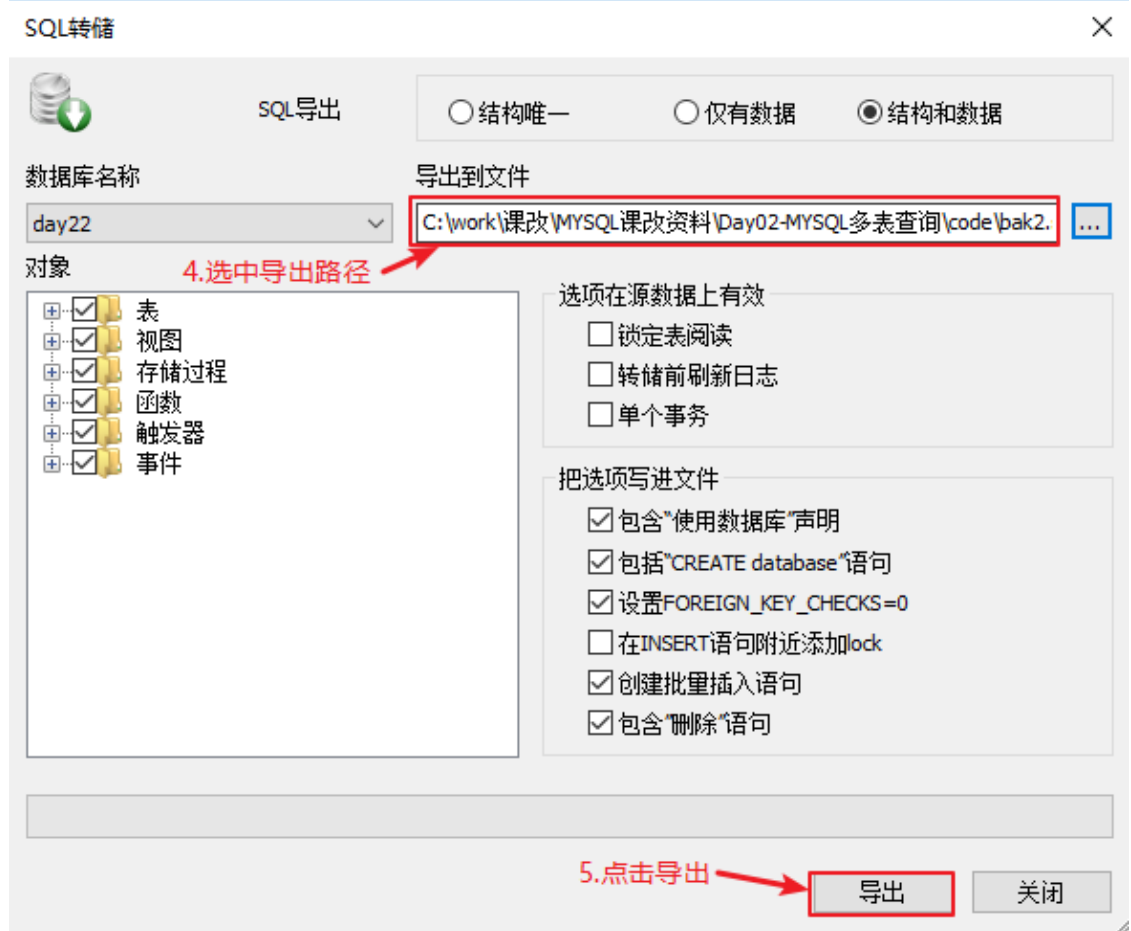
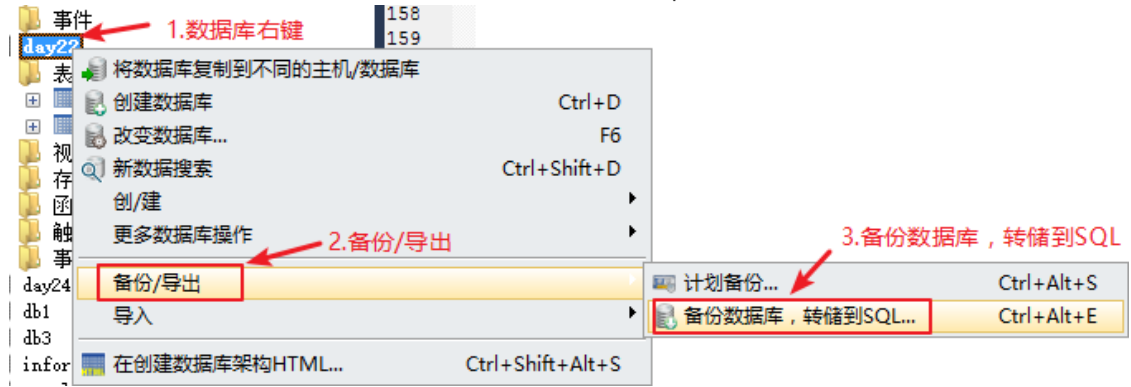
```
source C:\work\课改\MYSQL课改资料\Day02-MYSQL多表查询\code\bak.sql
```

```
mysql> source C:\work\课改\MYSQL课改资料\Day02-MYSQL多表查询\code\bak.sql  
Query OK, 0 rows affected (0.00 sec)  
  
Query OK, 0 rows affected (0.00 sec)  
  
Query OK, 0 rows affected (0.00 sec)  
  
Query OK, 0 rows affected (0.00 sec)
```

## 3.3 图形化界面备份与还原

- 备份day22数据库中的数据

选中数据库，右键“备份/导出”，指定导出路径，保存成.sql文件即可。



### 包含创建数据库的语句

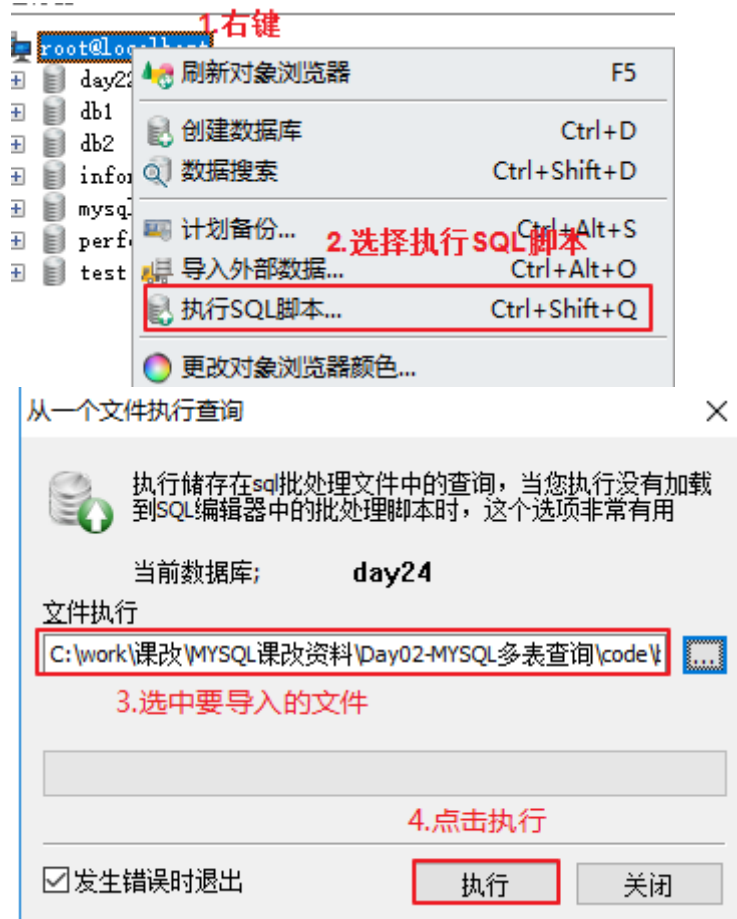
```
/*!40101 SET @OLD_SQL_MODE=@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@SQL_NOTES, SQL_NOTES=0 */;
CREATE DATABASE /*!32312 IF NOT EXISTS*/ `day22` /*!40100 DEFAULT CHARACTER SET utf8 */;
USE `day22`;

/*Table structure for table `employee` */
DROP TABLE IF EXISTS `employee`;

CREATE TABLE `employee` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- 还原day22数据库中的数据
  - 删除day22数据库

- 数据库列表区域右键“执行SQL脚本”，指定要执行的SQL文件，执行即可



## 第4章 数据库约束

### 目标

掌握数据库的约束创建

### 4.1 约束分类

对表中的数据进行进一步的限制，保证数据的**正确性**、**有效性**和**完整性**。约束种类：

- PRIMARY KEY：主键
- UNIQUE：唯一
- NOT NULL：非空
- DEFAULT：默认
- FOREIGN KEY：外键

### 4.2 主键

#### 4.2.1 主键的作用

用来唯一标识一条记录，每个表都应该有一个主键，并且每个表只能有一个主键。有些记录的name,age,score字段的值都一样时,那么就没法区分这些数据,造成数据库的记录不唯一,这样就不方便

管理数据	NAME	age	score	id	NAME	age	score
	张三	20	80	1	张三	20	80
	李四	21	90	2	李四	21	90
	王五	19	70	3	王五	19	70
	张三	20	80	4	张三	20	80

哪个字段应该作为表的主键？通常不用业务字段作为主键，单独给每张表设计一个id的字段，把id作为主键。主键是给数据库和程序使用的，不是给最终的客户使用的。所以主键有没有含义没有关系，只要不重复，非空就行。

## 4.2.2 创建主键

主键：PRIMARY KEY 主键的特点：

- 主键必须包含唯一的值
- 主键列不能包含NULL值

创建主键方式：

1. 在创建表的时候给字段添加主键 字段名 字段类型 PRIMARY KEY
2. 在已有表中添加主键 ALTER TABLE 表名 ADD PRIMARY KEY(字段名);

具体操作：

- 创建表学生表st5, 包含字段(id, name, age)将id做为主键

```
CREATE TABLE st5 (  
    id INT PRIMARY KEY, -- id是主键  
    NAME VARCHAR(20),  
    age INT  
);
```

1 信息 2 表数据 3 信息		
Format: <input checked="" type="radio"/> HTML <input type="radio"/> 文本/详细 刷新		
表示主键		
Field	Type	
 id	int(11) NOT NULL	
NAME	varchar(20) NULL	
age	int(11) NULL	

- 添加数据

```
INSERT INTO st5 (id, NAME) VALUES (1, '唐伯虎');  
INSERT INTO st5 (id, NAME) VALUES (2, '周文宾');  
INSERT INTO st5 (id, NAME) VALUES (3, '祝枝山');  
INSERT INTO st5 (id, NAME) VALUES (4, '文征明');
```

- 插入重复的主键值

```
-- 主键是唯一的不能重复: Duplicate entry '1' for key 'PRIMARY'  
INSERT INTO st5 (id, NAME) VALUES (1, '文征明2');
```

- 插入NULL的主键值

```
-- 主键是不能为空的: column 'id' cannot be null
INSERT INTO st5 (id, NAME) VALUES (NULL, '文征明3');
```

注意：一张表中只有一个主键，主键可以为多个字段，不过我们一般设置一个自增的id字段作为主键。

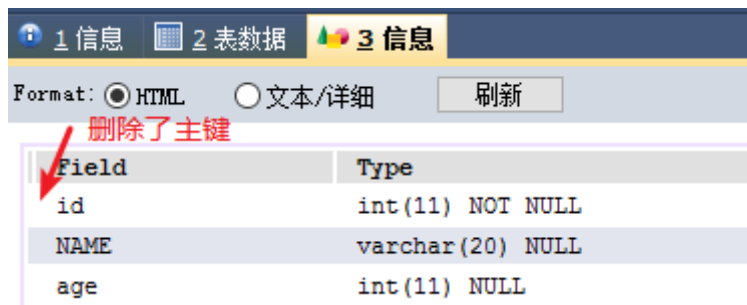
### 4.2.3 删除主键

```
ALTER TABLE 表名 DROP PRIMARY KEY;
```

具体操作：

- 删除st5表的主键

```
ALTER TABLE st5 DROP PRIMARY KEY;
```



Field	Type
id	int(11) NOT NULL
NAME	varchar(20) NULL
age	int(11) NULL

### 4.2.4 主键自增

主键如果让我们自己添加很有可能重复,我们通常希望在每次插入新记录时,数据库自动生成主键字段的值 `AUTO_INCREMENT` 表示自动增长(字段类型是整型数字)

具体操作：

- 创建学生表st6, 包含字段(id, name, age)将id做为主键并自动增长

```
CREATE TABLE st6 (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    NAME VARCHAR(20),  
    age INT  
);
```

- 插入数据

```
-- 主键默认从1开始自动增长
INSERT INTO st6 (NAME, age) VALUES ('唐僧', 22);
INSERT INTO st6 (NAME, age) VALUES ('孙悟空', 26);
INSERT INTO st6 (NAME, age) VALUES ('猪八戒', 25);
INSERT INTO st6 (NAME, age) VALUES ('沙僧', 20);
```



id	NAME	age
1	唐僧	22
2	孙悟空	26
3	猪八戒	25
4	沙僧	20



## DELETE和TRUNCATE的区别

- DELETE 删除表中的数据，但不重置AUTO\_INCREMENT的值。

### 原本数据

id	NAME	age
1	唐僧	22
2	孙悟空	26
3	猪八戒	25
4	沙僧	20

### DELETE 删除数据

```
DELETE FROM st6;
```

id	NAME	age
(Auto)	(NULL)	(NULL)

### 插入数据

```
INSERT INTO st6 (NAME, age) VALUES ('唐僧', 22);
```

id	NAME	age
5	唐僧	22

### 主键在原来的基础之上增加1

- TRUNCATE 摧毁表，重建表，AUTO\_INCREMENT重置为1

### 原本数据

id	NAME	age
1	唐僧	22
2	孙悟空	26
3	猪八戒	25
4	沙僧	20

### TRUNCATE摧毁表重建

```
TRUNCATE st6;
```

id	NAME	age
(Auto)	(NULL)	(NULL)

### 插入数据

```
INSERT INTO st6 (NAME, age) VALUES ('唐僧', 22);
```

id	NAME	age
5	唐僧	22

id	NAME	age
1	唐僧	22

### TRUNCATE 摧毁表，重建表， AUTO\_INCREMENT重置为1

## 4.3 唯一

在这张表中这个字段的值不能重复

### 4.3.1 唯一约束的基本格式

字段名 字段类型 UNIQUE

## 4.3.2 实现唯一约束

具体步骤:

- 创建学生表st7, 包含字段(id, name),name这一列设置唯一约束,不能出现同名的学生

```
CREATE TABLE st7 (  
    id INT,  
    NAME VARCHAR(20) UNIQUE  
);
```

- 添加一个学生

```
INSERT INTO st7 VALUES (1, '貂蝉');  
INSERT INTO st7 VALUES (2, '西施');  
INSERT INTO st7 VALUES (3, '王昭君');  
INSERT INTO st7 VALUES (4, '杨玉环');  
  
-- 插入相同的名字出现name重复: Duplicate entry '貂蝉' for key 'name'  
INSERT INTO st7 VALUES (5, '貂蝉');  
  
-- 出现多个null的时候会怎样? 因为null是没有值, 所以不存在重复的问题  
INSERT INTO st3 VALUES (5, NULL);  
INSERT INTO st3 VALUES (6, NULL);
```

## 4.4 非空

这个字段必须设置值,不能是NULL

### 4.4.1 非空约束的基本语法格式

字段名 字段类型 NOT NULL

具体操作:

- 创建表学生表st8, 包含字段(id,name,gender)其中name不能为NULL

```
CREATE TABLE st8 (  
    id INT,  
    NAME VARCHAR(20) NOT NULL,  
    gender CHAR(2)  
);
```

- 添加一条完整的记录

```
INSERT INTO st8 VALUES (1, '郭富城', '男');  
INSERT INTO st8 VALUES (2, '黎明', '男');  
INSERT INTO st8 VALUES (3, '张学友', '男');  
INSERT INTO st8 VALUES (4, '刘德华', '男');  
  
-- 姓名不赋值出现姓名不能为null: Column 'name' cannot be null  
INSERT INTO st8 VALUES (5, NULL, '男');
```

## 4.4.2 默认值

往表中添加数据时,如果不指定这个字段的数据,就使用默认值

默认值格式 `字段名 字段类型 DEFAULT 默认值`

具体步骤:

- 创建一个学生表 st9, 包含字段(id,name,address), 地址默认值是广州

```
CREATE TABLE st9 (  
  id INT,  
  NAME VARCHAR(20),  
  address VARCHAR(50) DEFAULT '广州'  
);
```

- 添加一条记录,使用默认地址

```
INSERT INTO st9 (id, NAME) VALUES (1, '刘德华');
```

添加一条数据, address不指定数据, 会使用创建表时指定的默认值

```
INSERT INTO st9 (id, NAME) VALUES (1, '刘德华');
```

<input type="checkbox"/>	id	NAME	address
<input type="checkbox"/>	1	刘德华	广州
*	(NULL)	(NULL)	广州

从这里也可以看出表的默认值

- 添加一条记录,不使用默认地址

```
INSERT INTO st9 VALUES (2, '张学友', '香港');
```

## 小结

不管是哪种约束,其最终的目的就是约束我们的数据有效性和完整性。防止我们的数据在数据库产生冗余数据

- PRIMARY KEY: 主键
- UNIQUE: 唯一
- NOT NULL: 非空
- DEFAULT: 默认