SQL

1. 介绍

数据库是什么呢？**数据库其实就是用来存储和管理数据的仓库。**

在生活中，也要一些岗位（例如财务、会计等）都是通过Excel来管理数据的。其实数据库基本上就是和Excel是类似的，是通过一张一张表格来存储和管理数据的。

数据库有多种产品，按照大类来区分，有两类：

- 关系型数据库

  > 什么是关系型数据库呢？值这类数据库不仅仅能存储数据，还能存储数据与数据之间的关系。

  > 关系型数据库一般是把数据基于磁盘来存储的。（读写速度慢）
  >
  > **SQL：Structured Query Language，结构化的查询语言。其实SQL就是SQL标准委员会给众多的关系型数据库制定的一个统一的语法。这个统一的语法其实就是SQL标准语法。**

  - MySQL
  - Oracle
  - SQL server
  - Access
  - MariaDB
  - SQL lite
  - OceanBase
- 非关系型数据库

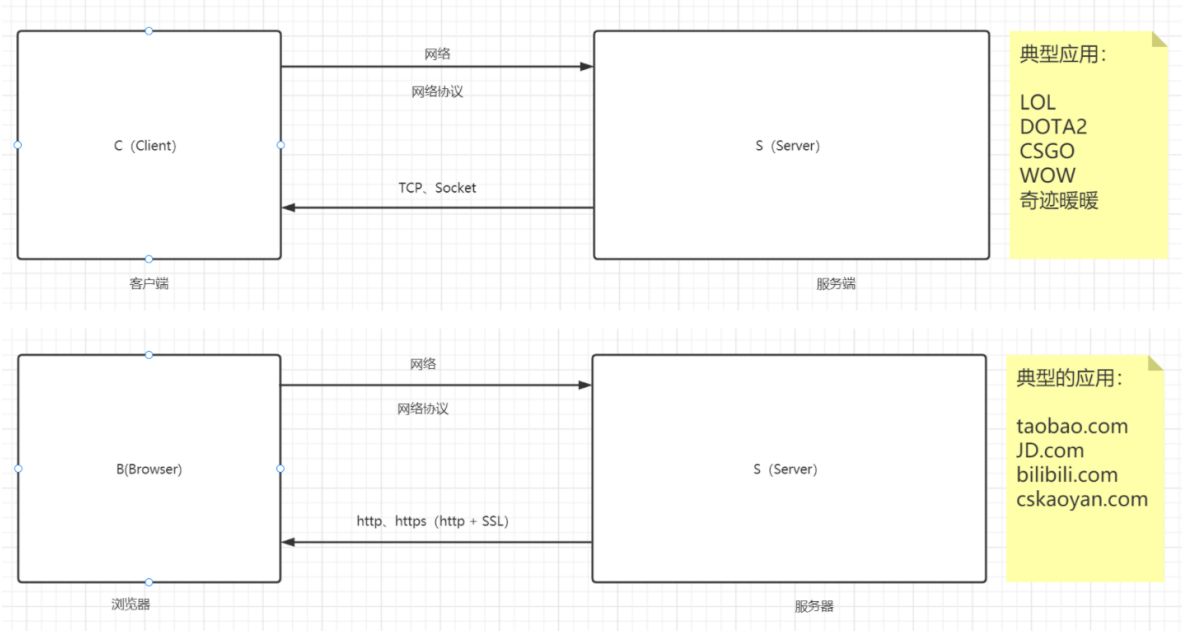
  > 仅仅只能存储数据。

  > 非关系型数据库一般是基于内存来存储的。（读写速度快）是对关系型数据库的一个补充。
  >
  > 非关系型数据库一般也叫作NoSQL。Not only SQL。不仅仅是SQL。

  - Redis
  - Memcache
  - Hbase
  - MongoDB

2. 安装

2.1 软件的架构



我们需要安装的MySQL是一个C/S架构的应用。

2.2 MySQL安装

[中文官网]

2.2.1 安装服务端

安装的版本要求： 5.6+、5.7、5.8

安装的密码：123456

> - 程序的路径：C:\Program Files\MySQL\MySQL 5.7\
> - 数据的路径：C:\ProgramData\MySQL\MySQL Server 5.7\data

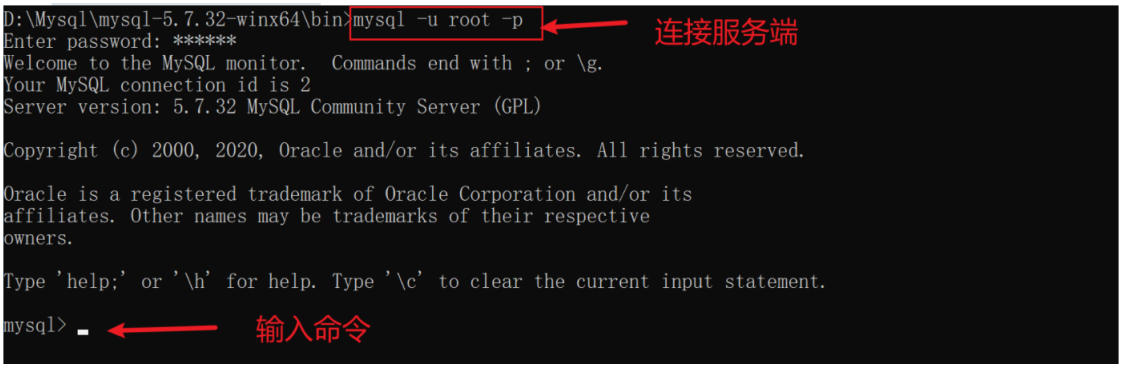
MySQL安装好了之后唯一的标识：

2.2.2 安装客户端

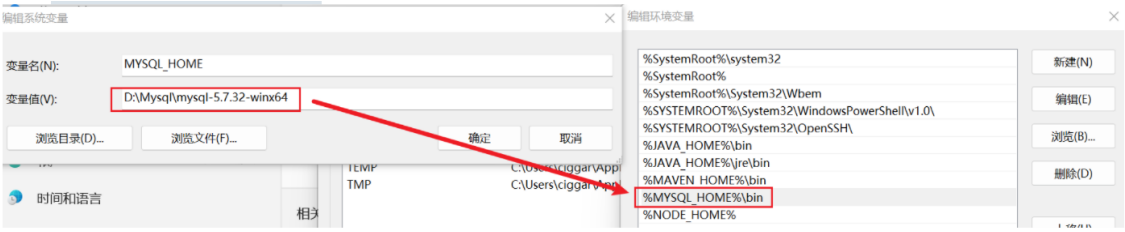
- 命名行客户端

  这个是一个默认安装的客户端，是不用我们特地再来安装的。

  `C:\Program Files\MySQL\MySQL 5.7\bin`



- 配置环境变量



- 图形化界面客户端
  - SQL yog
  - Workbench
  - **Navicat**
  - Dbeaver

  以安装Navicat为例：

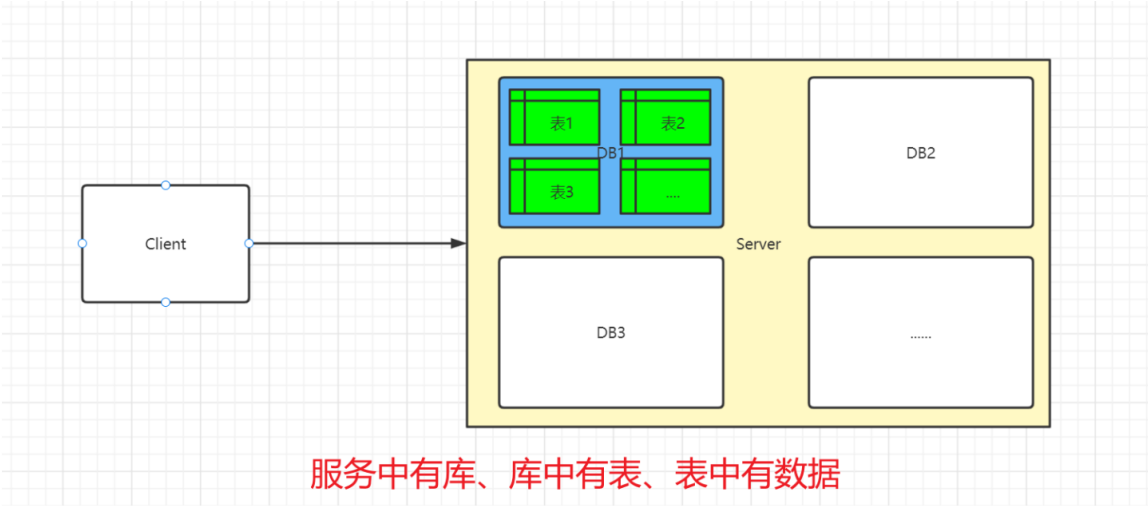
3. SQL

介绍MySQL的使用其实就是介绍SQL标准语法。

- 非常简单，由一些关键字组成
- 关键字不区分大小写 ， 而一些表名、列名等需要看服务端的配置

  （在实际操作的时候，建议大家都使用小写的）

3.1 MySQL的组成结构



3.2 库的操作

> 在命令行客户端中，每一个SQL语句写完之后，后面要加一个分号，表示SQL语句的结束。

增

```
-- 新增数据库
create database dbName;

-- 新增库的时候指定字符集
create database dbName character set utf8 collate utf8_bin;

-- 字符集：字符集就是一套符号和编码的集合。
-- 校对规则：校对规则是这套字符集中用于比较的这么一套规则。
-- 常规的字符集和校对规则：
-- utf8 | utf8_bin(区分大小写)  utf8_general_ci(不区分大小写)
-- utf8mb4 | utf8mb4_bin  utf8mb4_general_ci
-- latin1(不支持中文)
```

删

```
-- 删除数据库(慎重)
drop database dbName;
```

改

```
-- 改名字
-- SQL标注语法不支持对数据改名

-- 改字符集和校对规则
alter database dbName character set utf8 collate utf8_bin;
```

查

```
-- 查询所有的数据库
show databases;

-- 查询数据库的建库语句
show create database dbName;
```

3.3 表的操作

> 因为表都是在数据库中的，所以在操作表之前需要指定数据库。

```
-- 指定数据库
use dbName;
```

增

```
create table tableName(
    columnName1 dataType1,
    columnName2 dataType2,
    columnName3 dataType3,
    ...
    columnName4 dataType4
)character set utf8 collate utf8_bin;
```

> 数据类型：
>
> - 整型
>
>   int | smallint | bigint |
> - 浮点型
>
>   float | double| decimal
> - 日期类型
>
>   year | date | time | datetime | timestamp
> - 字符串类型
>
>   char | **varchar** | text
>
>   enum | set
>
>   (一般在使用数据库的时候要遵循单一职责，即数据库仅仅是用来存储和管理数据的，不用来做数据的正确性的逻辑判断，所以enum和set一般不使用)
> - 二进制类型

> 练习：

```
-- 练习
create table employee(
    id int,
    name varchar(20),
    gender varchar(10),
    birthday date,
    entry_date date,
    job varchar(20),
    salary decimal,
    resume text
)character set utf8 collate utf8_bin;
```

删

```
-- 删除表
drop table tableName;
```

改

```
-- 修改表名
rename table employee to staff;

-- 修改表的字符集
-- 修改表的字符集的时候不会修改表中的字符串类型的列的字符集
alter table staff character set gbk collate gbk_bin;

-- 修改表中的列
-- 新增一列 | add
alter table staff add height int;
-- 删除一列 | drop
alter table staff drop height;
-- 修改一列（修改列名change | 修改列的类型 modify）
alter table staff modify resume char(20) [character set utf8 collate utf8_bin];
alter table staff change resume weight int;
alter table staff change weight weight float(6,2);
```

查

```
-- 查询当前数据库中所有的表
show tables;

-- 查询表的建表语句
show create table tableName;

CREATE TABLE `user` (
  `id` int(11) DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `name` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `hobby` set('唱','跳','rap','篮球') COLLATE utf8_bin DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin

-- 1. 在SQL标准语法中，双引号和单引号的作用是一样的
```

```
-- 2. `id`等列名和表名都被上引号包起来是为了让MySQL服务器把这些列名和表名当成纯文本来解析，而不是当成关键字来解析

-- 查看表的结构
describe tableName;
desc tableName;
```

3.4 数据的操作

增

```
-- 新增单条数据
insert into student values (1001,"张飞",30,"1999-10-30");

-- 新增多条数据
insert into student values (1002,"关羽",35,"1994-10-30"),(1003,"刘备",40,"1989-10-30"),(1004,"吕布",38,"1991-10-30");


-- 新增指定列的数据
insert into student (id,name) values (2001,"花无缺"),(2002,"刀郎");

insert into student -- 新增单条数据
insert into student values (1001,"张飞",30,"1999-10-30");

-- 新增多条数据
insert into student values (1002,"关羽",35,"1994-10-30"),(1003,"刘备",40,"1989-10-30"),(1004,"吕布",38,"1991-10-30");


-- 新增指定列的数据
insert into student (id,name) values (2001,"花无缺"),(2002,"刀郎");

insert into student values (3001,"陈奕迅",null,null); (3001,"陈奕迅",null,null);

-- values 和 value 是等价的，但是不建议使用value，因为适配性不好
```

删

```
-- 删除数据
delete from student;

-- 删除指定的数据 (where)
delete from student where id =1004;
delete from student where id > 2000;

delete from student where name = '刘备';
```

改

```
-- 修改数据
-- 修改单列
update student set name = '周杰伦';

-- 修改多列
update student set name = "疫严丁真",age = 50;


-- 修改指定的行
update student set name = '蔡徐坤',age = 30,birthday='2000-10-20' where id =
2002;
```

查

```
-- 查询

-- 查询单列
select  id from student;

-- 查询多列
select id,name from student;

-- 查询所有列
select * from student;

-- 查询指定行的数据
select * from student where id  >= 2000;
```

3.5 关键字

where

where关键字是筛选。通过条件去筛选指定的行。

- 算术运算符

  `+ - X / %`

- 比较运算符

  `> < >= <= != <>`

  `is null | is not null | in | not in | between and | like`

- 逻辑运算符

  `and | or`

- 位运算符

  一般不使用

```
-- 查询总成绩>200分的同学的信息
select * from t_students where (chinese + english + math) > 200;

-- 查询语文成绩不等于90分的同学信息
```

```
select * from t_students where chinese != 90;

select * from t_students where class  <> '一班';

update t_students set class  = null where id = 1;

-- 需要注意的是，空串不是null
-- is null
select * from t_students where class is null;

-- is not null
select * from t_students where class is not null;

-- in
-- 查询一班、二班和三班的所有同学信息
select * from t_students where class = '一班' or class = '二班' or class = '三班';

select * from t_students where class in ("一班",'二班','三班');


-- not in
select * from t_students where class not in ("一班",'二班','三班');

-- between and (闭区间)
-- 查询语文成绩大于等于60分小于等于90分的同学信息
select * from t_students where chinese >= 60 and chinese <= 90;

select * from t_students where chinese between 60 and 90;

-- like
-- 模糊查询
-- _:表示占位
-- %:表示通配

-- 查询姓黄的同学的信息
select * from t_students where name like "黄%";

-- 查询姓黄的、名字只有两个字的同学的信息
select * from t_students where name like "黄__";

-- 查询名字中带黄字的同学信息
select * from t_students where name like "%黄%";
```

distinct

去重。

```
-- 去重
select distinct(class) from t_students;

-- distinct要求两行数据是完全一致的情况下才可以去重
select distinct(class),name from t_students;
```

limit

限制结果集。

```
select * from t_students limit 0,5;


-- 查询第二行到第五行的学生信息
select * from t_students limit 1,4;

-- 假如某个查询有n条结果，n >> 10000
-- 现在页面上一页显示100个

-- 第一页：limit 0,100;
-- 第二页：limit 100,100;
-- 第三页：limit 200,100;
-- 第 n页：limit (n-1)*100,100

-- 分页的公式： 页码（pageNo），页面大小(pageSize)
select * from tableName limit (pageNo-1)*pageSize, pageSize;
```

计算字段

计算字符并不是一个关键字，而是一种用法；计算字段可以支持我们对**行内的数据进行计算。**横向的

```
-- 查询各个学生的总成绩
select *,chinese+english+math from t_students;
```

as

起别名。as关键字可以省略，但是不建议省略。

> **select查询语句 查询出来的结果是一个 没有名字的临时表。我们可以对这个临时表的表名和列名去起别名。**

```
-- 别名
select name as '姓名',(chinese+english+math) as '得分' from t_students;


-- 查询各个学生的总成绩并且筛选出总分大于250分的同学信息
select *,chinese+english+math as total from t_students
where (chinese+english+math) > 250;


select * from ((select *,chinese+english+math as total from t_students) as sss)
where total > 250;
```

order by

排序。

```
-- 排序
select * from t_students;

-- 查询出各个学生的成绩信息并且按照语文成绩进行排序（从低到高排）
-- ASC 升序(可以省略，默认就是升序)
-- DESC 降序
select * from t_students order by chinese ASC;

-- 按多字段进行排序
-- 查询出各个学生的成绩信息，并且按照语文成绩的降序进行排序；假如语文成绩一样，那么再按照英语成绩的降序进行排序

select * from t_students order by chinese DESC,english DESC;
```

group by

分组


```
-- 按照班级进行分组
select group_concat(id) as ids,group_concat(name) as nameList,class from student
group by class;
```

> 分组之后进行过滤：having

聚合函数

聚合函数是作用在列上的函数，可以求出某一列的总和，最大值，最小值，数量，平均值。**（纵向的）**

- max

  求最大值

- min

  求最小值

- sum

  求和

- avg

  求平均值

- count

  计数

```sql
-- 求各个学生中语文的最高分
select max(chinese) from  t_students;

-- 求各个学生中语文的最低分
select min(chinese) from  t_students;

-- 求语文的总分
select sum(chinese) from  t_students;

-- 求语文的平均分
select avg(chinese) from t_students;

-- 求语文成绩的个数
select count(chinese) from t_students;

-- 求表中的总行数
select count(id) from t_students;

select count(*) from t_students;

select count(*) from student;
```

> 聚合函数通常情况下是配合分组一起来使用的。

3.6 SQL语句的顺序

```sql
(5) SELECT column_name, ...
(1) FROM table_name, ...
(2) [WHERE ...]
(3) [GROUP BY ...]
(4) [HAVING ...]
(6) [ORDER BY ...];
(7) [Limit m,n];

-- select 后面的列的计算，包括聚合函数的计算，受到MySQL内部优化器的影响，可能会在having之前或者是之后执行。
```

3.7 数据完整性

数据完整性是MySQL在设计表的时候，给用户提供了一些规范，防止用户可能的错误的输入。

实体完整性

实体完整性是确保用户插入到表中的数据是唯一的，是不重复的。实体完整性通过**主键**来体现。

```
-- 创建主键
create table teacher(
    -- 声明id是主键
    id int PRIMARY KEY,
    name varchar(20),
    age int
)character set utf8 collate utf8_bin;

-- 声明为主键的这一列的值有两个特点：
-- 1. 不能重复
-- 2. 不能为null
```

- auto_increment

  自增的意思，其实就是让数据库来维护主键的值，数据库采取自增的策略。

  ```
  -- 创建主键
  create table teacher(
      -- 声明id是主键    声明自增
      id int PRIMARY KEY auto_increment,
      name varchar(20),
      age int
      -- 指定自增从多少开始
  ) [auto_increment = 100001] character set utf8 collate utf8_bin;
  ```

域完整性

域完整性是指数据库表中的每一列都必须有数据类型来约束。还提供了一些关键字来约束具体列的值。

- unique

  表示唯一，用unique修饰的列，值不能重复

- not null

  表示不为空，用not null修饰的列，值不能为空

```
create table teacher(
    -- 声明是主键
    id int PRIMARY KEY,

    -- 声明唯一
    name varchar(20) unique,

    -- 值不能为null
    age int not null

);
```

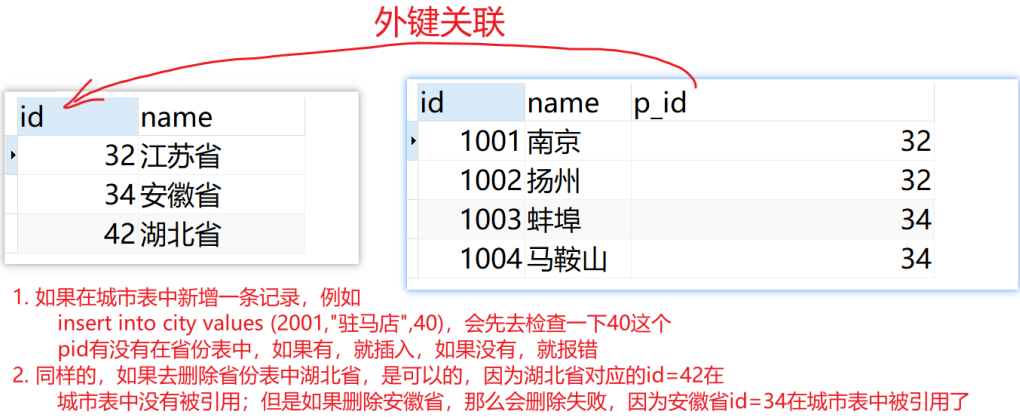
> 主键和unique的异同：
>
> 1. 一个表中只能有一个主键，一个表中可以出现多个unique
> 2. 主键和unique都能确保值不重复出现
> 3. 主键修饰的列值不能重复，并且不能为空；unique修饰的列值不能重复，但是可以为空，并且空值可以重复出现
> 4. 主键和unique在存储的原理上有区别

参照完整性

参照完整性指的就是**外键。**外键就是一种数据与数据之间的关系的具体表现形式。

```
-- 创建一个城市表
create table city (
    id int PRIMARY KEY,
    name varchar(20),
    p_id int,
    -- 声明外键
    constraint fk_pid foreign key(p_id) references province(id)

)character set utf8;
```



> 外键的优缺点：
>
> - 优点：确保数据的正确性
> - 缺点：会比较严重的影响效率（增删改）
>
> 因为外键会比较严重的影响性能，所以在公司里面，假如数据量比较大的话，一般不使用外键。

4. 多表设计

一对一

一对一是指两个表，表中的记录是一一对应的。

用户表				用户详情表					
id	name	password	——对应	id	weight	height	age	location	user_id
1001	张飞	俺也一样	→	1	180	180	30	蜀中	1001
1002	李云龙	开炮	→	2	150	170	40	晋西北	1002
1003	八戒	师傅被抓走了	→	3	200	175	200	高老庄	1003

一对一的关系模型有哪些常见的例子呢？

- 用户和用户详情
- 学生和学号
- 人和身份证

一对多

一对多是指存在表A和表B，表A中的一条记录对应表B中的多条记录，表B中的一条记录对应表A中的一条记录。

一对多其实还有很多常见的案例：

- 班级和学生
- 省份和城市

多对多

多对多的关系是指存在表A和表B，表A中的一条记录对应表B中的多条记录；表B中的一条记录也对应表A中的多条记录。

其实本质上就是互为一对多。

多对多生活中有很多常见的案例：

- 学生和课程
- 订单和商品

三大范式

三大范式是指我们在进行多表设计的时候，前人总结出来的三个规范。

第一范式

第一范式就是指的**原子性**。指每一列都是一个不可拆分的原子。

收获地址表

id	用户id	收货地址
1	1	湖北省武汉市洪山区花山街道软件新城C13

收获地址表

id	用户id	省份	城市	区/县	乡镇/街道	详细地址
1	1	湖北省	武汉市	洪山区	花山街道	软件新城C13

例如在上面的案例中，我们可以让收货地址进行拆分，拆分的更细，来保持原子性。

那么遵循了原子性之后有什么好处呢？可以便于业务的拓展。

第二范式

记录的**唯一性**。要求记录有唯一的标识，其实这个标识就是通过主键来体现。

订单号	产品号	产品数量	产品折扣	产品价格	订单金额	订单时间
2008003	205	100	0.9	10	4300	20181201
2008003	206	200	0.85	20	4300	20181201
2008005	207	300	0.8	30	7200	20181202
2008006	207	400	0.78	30	9360	20181202
2008008	207	500	0.75	30	58250	20181203
2008008	208	1000	0.6	50	58250	20181203
2008008	209	200	0.85	100	58250	20181203

第三范式

字段不要冗余。

班主任表

id	name	nickname
1001	长风	宝马哥
1002	景天	奶茶哥
1003	云天明	U盘哥

学生表

id	name	班主任id	班主任名字
1	张飞	1001	长风
2	关羽	1001	长风
3	刘备	1002	景天
4	大乔	1003	云天明
5	小乔	1003	云天明

冗余字段

> 数据冗余之后，有一些优缺点：
>
> - 优点：查询变得更加方便了
> - 缺点：增删改变麻烦了

> 在以后的工作中，一般要不要进行字段冗余呢？
>
> 假如查询的需求远大于增删改的需求，那么可以考虑适当的字段冗余。反之，不要进行冗余。
>
> 冗余字段的做法，一般叫做反范式化设计。(第一范式和第二范式必须要遵守，第三范式可以考虑不遵守...看情况)

5. 多表查询

5.1 连接查询

交叉连接

交叉连接的结果没有实际的意义。是两个表记录的笛卡尔积。

```
-- 交叉连
select * from province cross join city;
```

内连接

内连接的结果实际上是在交叉连接的结果之上按照条件进行筛选，结果是有意义的。

```
-- 内连接
-- 隐式
select * from province,city where province.id = city.p_id;

-- 显式
select * from province inner join city on province.id = city.p_id;
```

外连接

外连接的结果是在内连接的结果的基础之上，去和左表或者是右表做并集。

- 左外连接

  左外连接 会保留左表的全部数据。

- 右外连接

  右外连接会保留右表的全部数据

  ```
  -- 左外连接
  select * from province left outer join city on province.id = city.p_id;

  -- 右外连接
  select * from province right outer join city on province.id = city.p_id;

  -- outer可以省略
  ```

练习：

```
# 一对一
select * from user;
select * from user_detail;

-- 查询猪八戒的体重
select user_detail.weight from user left join user_detail on user.id =
user_detail.user_id where user.username = '猪八戒';


select * from clazz;
select * from student;
# 一对多
```

```sql
-- 查询一班的所有学生信息
select s.*
 from clazz as c  inner join student as s on c.id = s.clazz_id where c.name = '一
班';

-- 查询各个班级的人数
-- 连接
select clazz.id as cid,count(student.id)  from clazz left join student on
clazz.id = student.clazz_id
group by clazz.id;


-- 查询各个班级男学生的人数

-- 1. 连接
select * from clazz left join student on clazz.id = student.clazz_id;
-- 2. 筛选出男学生的记录
select * from clazz left join student on clazz.id = student.clazz_id
and student.gender = 'male';
-- 3. 分组
select clazz.id,count(student.id) from clazz left join student on clazz.id =
student.clazz_id
and student.gender = 'male' group by clazz.id;


# 多对多
select * from student;
select * from course;
select * from s_c;

-- 查询张三选了哪些课程
select c.* from student as s
    inner join s_c as sc on s.id = sc.sid
    inner join course as c on sc.cid = c.id
    where s.name = '张三';

-- 查询Java被哪些学生选了
select s.* from course as c
    inner join s_c as sc on c.id = sc.cid
    inner join student as s on sc.sid = s.id
    where c.name = 'JAVA';

-- 查询各个课程的选课人数并且按照人数排序（降序）

-- 1. 连接
select c.name as cname,s.name as sname from course as c
    left join s_c as sc on c.id = sc.cid
    left join student as s on sc.sid = s.id;
-- 2. 分组
select
    c.name as cname,
    count(s.name) as total,
    group_concat(s.name)
from course as c
```

```
    left join s_c as sc on c.id = sc.cid
    left join student as s on sc.sid = s.id
group by c.name;
-- 3. 排序
select
    c.name as cname,
    count(s.name) as total,
    group_concat(s.name)
from course as c
    left join s_c as sc on c.id = sc.cid
    left join student as s on sc.sid = s.id
group by c.name
order by total desc;
```

5.2 子查询

子查询又被称为嵌套查询。其实就是说SQL语句是可以嵌套的。

```
# 子查询
SELECT
    *
FROM
    student
WHERE
    clazz_id = ( SELECT id FROM clazz WHERE NAME = '一班' );


-- 查询二班和三班有哪些学生
select id from clazz where name = '二班' or name = '三班';
SELECT
    *
FROM
    student
WHERE
    clazz_id IN ( SELECT id FROM clazz WHERE NAME = '二班' OR NAME = '三班' );
```

> 子查询需要注意的是：
>
> 1. 嵌套的SQL语句的执行结果是单列值
> 2. 这个单列值的结果假如只有一个，可以直接用等于；假如有多个，可以考虑使用 in

5.3 联合查询

```
-- 查询一班和二班的信息
select * from clazz where name = '一班' or name = '二班';


select * from clazz where name = '一班'
union
select * from clazz where name = '二班';
```

> 一般不使用联合查询。如果or或者是in查询性能很差的时候，可以考虑使用联合查询。

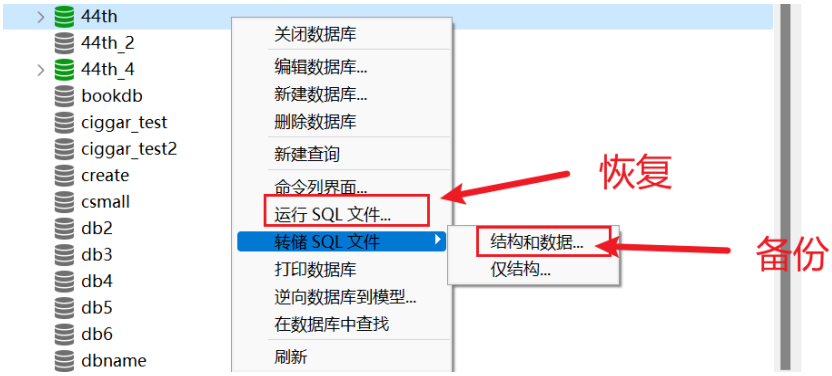
6. 数据库的备份与恢复

6.1 命令行客户端

```
# 备份
# 在命令行下面执行
mysqldump -uroot -p dbName>/path/dbName.sql

# 恢复(备份下来的文件中没有建库语句，只有建表语句和插入数据的语句)
# 0. 新建数据库
# 1. 选中数据库
# 2. 恢复数据 (source其实就是去执行这个SQL文件中的所有的SQL语句)
source /path/dbName.sql
```

6.2 Navicat



课堂SQL

2022-08-22

```sql
create table t_int(
    t1 tinyint,
    t2 smallint,
    t3 int
);


create table t_f(
    t1 float(4,2),
    t2 double(6,2)
);



create table t_date(
    t1 year,
    t2 time,
    t3 date,
    t4 datetime,
    t5 timestamp
);



create table t_str(
    t1 char(10) character set latin1,
    t2 varchar(20),
    t3 text
) character set utf8 collate utf8_bin;

create table user(
    id int,
    age int,
    name varchar(20),
    gender enum('男','女')
)character set utf8 collate utf8_bin;


create table user(
    id int,
    age int,
    name varchar(20),
    hobby set("唱","跳","rap","篮球")
)character set utf8 collate utf8_bin;

-- 查看表的建表语句
show create table user;
CREATE TABLE `user` (
  `id` int(11) DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `name` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `hobby` set('唱','跳','rap','篮球') COLLATE utf8_bin DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
```

```sql
create database `create`;

-- 查看表的结构
desc user;
describe user;



-- 练习
create table employee(
    id int,
    name varchar(20),
    gender varchar(10),
    birthday date,
    entry_date date,
    job varchar(20),
    salary decimal,
    resume text
)character set utf8 collate utf8_bin;


show create table staff;
CREATE TABLE `staff` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `gender` varchar(10) COLLATE utf8_bin DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `entry_date` date DEFAULT NULL,
  `job` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `salary` decimal(10,0) DEFAULT NULL,
  `resume` text COLLATE utf8_bin
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin


CREATE TABLE `staff` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `gender` varchar(10) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `entry_date` date DEFAULT NULL,
  `job` varchar(20) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `salary` decimal(10,0) DEFAULT NULL,
  `resume` text CHARACTER SET utf8 COLLATE utf8_bin
) ENGINE=InnoDB DEFAULT CHARSET=gbk COLLATE=gbk_bin


-- 修改表名
rename table employee to staff;

-- 修改表的字符集
-- 修改表的字符集的时候不会修改表中的字符串类型的列的字符集
alter table staff character set gbk collate gbk_bin;


desc staff;
```

```sql
-- 修改表中的列
-- 新增一列 | add
alter table staff add height int;
-- 删除一列 | drop
alter table staff drop height;
-- 修改一列（修改列名change | 修改列的类型 modify）
alter table staff modify resume char(20);
alter table staff change resume weight int;
alter table staff change weight weight float(6,2);
-- 修改表中列的字符集
CREATE TABLE `staff` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `gender` varchar(10) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `entry_date` date DEFAULT NULL,
  `job` varchar(20) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `salary` decimal(10,0) DEFAULT NULL,
  `weight` float(6,2) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=gbk COLLATE=gbk_bin

alter table staff modify name varchar(20) character set latin1 collate
latin1_bin;


CREATE TABLE `staff` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(20) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  `gender` varchar(10) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `entry_date` date DEFAULT NULL,
  `job` varchar(20) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `salary` decimal(10,0) DEFAULT NULL,
  `weight` float(6,2) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=gbk COLLATE=gbk_bin;

drop table t_f;

drop table t_int;


-- 数据的新增
create table student(
    id int,
    name varchar(20),
    age int,
    birthday date
)character set utf8 collate utf8_bin;


-- 新增单条数据
insert into student values (1001,"张飞",30,"1999-10-30");

-- 新增多条数据
```

```
insert into student values (1002,"关羽",35,"1994-10-30"),(1003,"刘备",40,"1989-10-30"),(1004,"吕布",38,"1991-10-30");


-- 新增指定列的数据
insert into student (id,name) values (2001,"花无缺"),(2002,"刀郎");

insert into student values (3001,"陈奕迅",null,null);


-- 删除数据
delete from student;

-- 删除指定的数据 (where)
delete from student where id =1004;
delete from student where id > 2000;

delete from student where name = '刘备';


-- 修改数据
-- 修改单列
update student set name = '周杰伦';

-- 修改多列
update student set name = "疫严丁真",age = 50;


-- 修改指定的行
update student set name = '蔡徐坤',age = 30,birthday='2000-10-20' where id = 2002;

-- 查询

-- 查询单列
select  id from student;

-- 查询多列
select id,name from student;

-- 查询所有列
select * from student;

-- 查询指定行的数据
select * from student where id  >= 2000;


CREATE TABLE t_students(
    id        INT       PRIMARY KEY    AUTO_INCREMENT,
    name      CHAR(20)  NOT NULL,
    class     CHAR(10)  NOT NULL,
    chinese   FLOAT,
    english   FLOAT,
    math      FLOAT
);
```

```sql
INSERT INTO t_students (name, class, chinese, english, math)
VALUES ('东邪', '一班', 90, 90, 90);
INSERT INTO t_students (name, class, chinese, english, math)
VALUES ('西毒', '二班', 80, 80, 90);
INSERT INTO t_students (name, class, chinese, english, math)
VALUES ('南帝', '三班', 80, 90, 80);
INSERT INTO t_students (name, class, chinese, english, math)
VALUES ('北丐', '四班', 60, 60, 60);
INSERT INTO t_students (name, class, chinese, english, math)
VALUES ('中神通', '五班', 100, 100, 100);
INSERT INTO t_students (name, class, chinese, english, math)
VALUES ('郭靖', '一班', 59, 59, 59);
INSERT INTO t_students (name, class, chinese, english, math)
VALUES ('黄蓉', '一班', 90, 90, 90);
INSERT INTO t_students (name, class, chinese, english, math)
VALUES ('黄药师', '二班', 92, 91, 89);


select * from t_students;


-- 查询总成绩>200分的同学的信息
select * from t_students where (chinese + english + math) > 200;

-- 查询语文成绩不等于90分的同学信息
select * from t_students where chinese != 90;

select * from t_students where class  <> '一班';

update t_students set class  = null where id = 1;

-- 需要注意的是，空串不是null
-- is null
select * from t_students where class is null;

-- is not null
select * from t_students where class is not null;

-- in
-- 查询一班、二班和三班的所有同学信息
select * from t_students where class = '一班' or class = '二班' or class = '三班';

select * from t_students where class in ("一班",'二班','三班');


-- not in
select * from t_students where class not in ("一班",'二班','三班');

-- between and (闭区间)
-- 查询语文成绩大于等于60分小于等于90分的同学信息
select * from t_students where chinese >= 60 and chinese <= 90;

select * from t_students where chinese between 60 and 90;
```

```
-- like
-- 模糊查询
-- _:表示占位
-- %:表示通配

-- 查询姓黄的同学的信息
select * from t_students where name like "黄%";

-- 查询姓黄的、名字只有两个字的同学的信息
select * from t_students where name like "黄__";

-- 查询名字中带黄字的同学信息
select * from t_students where name like "%黄%";
```

2022-08-23

```
select * from t_students limit 3,4;

-- 查询总成绩大于180分同学信息
select * from t_students where (chinese + english + math) > 180;

-- 查询数学成绩在[80,90]之间的同学姓名
select name from t_students where math BETWEEN 80 and 90;

-- 查询各科都及格的同学姓名
select name from t_students where
(chinese >= 60 and math >= 60 and english >= 60);

-- 查询一班和二班的同学信息
select * from t_students where class = '一班' or class = '二班';


select *  from t_students where class in ('一班','二班');


select class from t_students;

-- 去重
select distinct(class) from t_students;


select distinct(class),name from t_students;


select * from t_students limit 0,5;


-- 查询第二行到第五行的学生信息
select * from t_students limit 1,4;

-- 假如某个查询有n条结果，n >> 10000
```

```sql
-- 现在页面上一页显示100个

-- 第一页：limit 0,100;
-- 第二页：limit 100,100;
-- 第三页：limit 200,100;
-- 第 n页：limit (n-1)*100,100

-- 分页的公式： 页码（pageNo），页面大小(pageSize)
select * from tableName limit (pageNo-1)*pageSize, pageSize;




-- 查询各个学生的总成绩
select name,chinese+english+math from t_students;


-- 别名
select name  '姓名',(chinese+english+math)  '得分' from t_students;


-- 查询各个学生的总成绩并且筛选出总分大于250分的同学信息
select *,chinese+english+math as total from t_students
where (chinese+english+math) > 250;


select * from ((select *,chinese+english+math as total from t_students) as sss)
where total > 250;

-- 排序
select * from t_students;

-- 查询出各个学生的成绩信息并且按照语文成绩进行排序（从低到高排）
-- ASC 升序(可以省略，默认就是升序)
-- DESC 降序
select * from t_students order by chinese ASC;

-- 按多字段进行排序
-- 查询出各个学生的成绩信息，并且按照语文成绩的降序进行排序；假如语文成绩一样，那么再按照英语成绩的降序进行排序

select * from t_students order by chinese DESC,english ASC;


drop table student;

create table student(
    id int,
    name varchar(20),
    class varchar(20)
)character set utf8 collate utf8_bin;

insert into student values (1001,"张飞",'一班');
insert into student values (1002,"关羽",'一班');
insert into student values (1003,"刘备",'一班');
```

```
insert into student values (2001,"张苞",'二班');
insert into student values (2002,"阿斗",'二班');
insert into student values (2003,"关索",'二班');
insert into student values (3001,"曹操",'三班');
insert into student values (3002,"荀彧",'三班');
insert into student values (3003,"郭嘉",'三班');

select * from student;


-- 按照班级进行分组
select group_concat(id) as ids,group_concat(name) as nameList,class from student
group by class;


select * from t_students;

-- 求各个学生中语文的最高分
select max(chinese) from  t_students;

-- 求各个学生中语文的最低分
select min(chinese) from  t_students;

-- 求语文的总分
select sum(chinese) from  t_students;

-- 求语文的平均分
select avg(chinese) from t_students;

-- 求语文成绩的个数
select count(chinese) from t_students;

-- 求表中的总行数
select count(id) from t_students;

select count(*) from t_students;

select count(*) from student;




-- 求各个班级的人数
select * from student;

select class,count(id) from student group by class;



select * from t_students;

-- 查询每个同学的总成绩、平均成绩，并且用别名来表示
select *,(chinese+english+math) as total, (chinese+english+math) / 3 as '平均成
绩'from t_students;
```

```sql
-- 查询数学最大值，并且用别名表示
select max(math) as '数学最高分' from t_students;

select math as '数学最高分' from t_students order by math desc limit 0,1;

-- 查询全体学生各科的平均成绩，用别名表示
SELECT
    avg( chinese ) AS '语文平均分',
    avg( math ) AS '数学平均分',
    avg( english ) AS '英语平均分',
    sum(math) as '数学总分'
FROM
    t_students;

-- 查询人数大于2的班级

-- 先分组
select * from t_students group by class;

-- 计算各个班级的人数
select class,count(id) as total from t_students group by class;

-- 过滤出人数大于2的班级
select class,count(id) as total from t_students group by class having total > 2;

-- 查询人数少于10的班级，并且按照人数的降序进行排序

-- 分组
select * from t_students group by class;

-- 求各个班级的人数
select class,count(id) as total from t_students group by class;

-- 筛选出人数少于10的班级
select class,count(id) as total from t_students group by class having total <
10;

-- 按照人数进行排序
select class,count(id) as total from t_students group by class having total < 10
order by total desc;


drop table teacher;

-- 创建主键
create table teacher(
    -- 声明id是主键
    id int PRIMARY KEY auto_increment,
    name varchar(20),
    age int
)auto_increment = 100001 character set utf8 collate utf8_bin;
```

```
insert into teacher values (null,'天明',33);

show create table teacher;

CREATE TABLE `teacher` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2010 DEFAULT CHARSET=utf8 COLLATE=utf8_bin



CREATE TABLE `teacher` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3001 DEFAULT CHARSET=utf8 COLLATE=utf8_bin


select * from teacher;

insert into teacher values (1001,"天明",30);
insert into teacher values (1002,"景天",40);


insert into teacher values (null,"长风4号",30);

insert into teacher values (3000,"鲁智深xxx",33);



drop table teacher;


create table teacher(

    id int PRIMARY KEY,
    name varchar(20) unique,
    age int not null

);

insert into teacher values (1001,'天明',30);

insert into teacher values (1002,'天明1号',35);
insert into teacher values (1004,null,35);
insert into teacher values (1005,"长风",null);


select * from teacher;


drop table province;
```

```sql
drop table city;

-- 创建一个省份表
create table province(

    id int PRIMARY KEY,
    name varchar(20)
)character set utf8;

-- 创建一个城市表
create table city (
    id int PRIMARY KEY,
    name varchar(20),
    p_id int,
    -- 声明外键
    constraint fk_pid foreign key(p_id) references province(id)
)character set utf8;


select * from province;
select * from city;


insert into province values (32,'江苏省');
insert into province values (34,'安徽省');
insert into province values (42,'湖北省');

insert into city values (1001,'南京',32);
insert into city values (1002,'扬州',32);

insert into city values (1003,'蚌埠',34);
insert into city values (1004,'马鞍山',34);


insert into city values (2001,'驻马店',40);

delete from city where id = 1003 or id = 1004;

delete from province where id = 34;

delete from province where id = 42;

delete from province where id = 32;

select * from province;
select * from city;

-- 交叉连接
select * from province cross join city;

-- 内连接
-- 隐式
select * from province,city where province.id = city.p_id;

-- 显式
```

```
select * from province inner join city on province.id = city.p_id;

-- 左外连接
select * from province left  join city on province.id = city.p_id;

-- 右外连接
select * from province right outer join city on province.id = city.p_id;
```

```
# 一对一
select * from user;
select * from user_detail;

-- 查询猪八戒的体重
select user_detail.weight from user left join user_detail on user.id =
user_detail.user_id where user.username = '猪八戒';


select * from clazz;
select * from student;
# 一对多

-- 查询一班的所有学生信息
select s.*
 from clazz as c  inner join student as s on c.id = s.clazz_id where c.name = '一
班';

-- 查询各个班级的人数
-- 连接
select clazz.id as cid,count(student.id)  from clazz left join student on
clazz.id = student.clazz_id
group by clazz.id;


-- 查询各个班级男学生的人数

-- 1. 连接
select * from clazz left join student on clazz.id = student.clazz_id;
-- 2. 筛选出男学生的记录
select * from clazz left join student on clazz.id = student.clazz_id
and student.gender = 'male';
-- 3. 分组
select clazz.id,count(student.id) from clazz left join student on clazz.id =
student.clazz_id
and student.gender = 'male' group by clazz.id;


# 多对多
```