# Mysql

## php阶段将数据库分为三个阶

### 基础阶段：

mysql数据库的基本操作（增删改查），以及一些高级操作（视图，触发器，函数，存储过程等），PHP操作没有sql数据库。

### 优化阶段：

如何提高数据库效率，如索引，分表等。

### 部署阶段：

如何搭建真实的系统环境，如服务集群，负载均衡。

## 数据库基础

### 什么是数据库：

数据库：database ，存储数据的仓库（高效的存储和处理数据库的介质如磁盘、内存）

### 数据库分类：

1. 关系型数据库：（sql型）
2. 非关系型数据库（nomysql型）not only

### 不同数据卡产品：

关系型数据库：大型：Oracle ，DB2

中型：SQLserver，Mysql

小型：access

非关系型数据库：mencahed，moogodb，redis2

### 数据库的区别->两种阵营

关系型database：安全（保存在磁盘，不会丢失），浪费空间。

非关系型数据库：效率高（不够安全）

## 关系型数据库：

### 什么是关系型数据库：

建立在关系模型上的数据库（数学模式）

关系模型：建立在关系上的模型，三个方面。

1. 数据结构：数据存储问题，二维表的行列结构
2. Sql指令集合：sql语句。
3. 完整性约束：表内数据约束，表与表之间的约束。

### 关系型数据库的设计：

关系型数据库：从需要存储的数据需求分析，如果是一类数据（实体）->设计成二维表->表头（字段名field）和数据部分（实际存储单元）；

二维表：行与列，

例：

|  |  |  |  |
| --- | --- | --- | --- |
| 表头 | 字段1 | 字段2 | 字段3 |
| 数据单元 | 数据1 | 数据2 | 数据3 |

实际案例：教师负责讲学，教学生，在教室。

1. 找出实体。教师表，学生表，教室表。
2. 找出实体数据信息： （教师）姓名，性别，年龄，工资

（学生）姓名，性别，年龄，学号

（班级）班级编号，教室号

### 3，关系型数据库：维护实体内部、实体与实体之间的联系。

1，实体内部的联系：每个学生都有姓名、性别、学科、学号。

学生表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓名 | 性别 | 学号 | 学科 | 班名 |
| 张三 | 男 | 001 | 软件工程 | 软件工程001 |
| 小强 | 女 | 001 | 机械工程 | 机械工程001 |

第二行所有字段：都是在描述张三这个学生（内部联系）第二列只能是性别（内部约束）。

关系型数据库特点之一：如果表对应的某个字段没有值（数据），但是系统依然要分配存储空间；关系型数据库比较浪费空间。

实体与实体之间的联系：每个学生肯定属于某个班级，每个班级肯定有多个学生。

|  |  |
| --- | --- |
| 班名 | 教室编号 |
| 软件工程001 | 001 |
| 机械工程001 | 002 |

班级实体与学生实体之间的联系：实体与实体之间的联系

### 关键字说明：

数据库：database

数据库系统：DBS （database system）：虚拟系统，将多种内容关联起来的称呼

DBS=DBMS+DB

DBMS:database mangerment system数据库管理系统，专门管理数据库。

DBA：database Administrator 数据库管理员。

行/记录：row/record本质是一个东西->表中的一个记录，行是从结构角度出发，记录从数据角度出发。

列/字段 column/filed，本质是一个东西。

## SQL：

Structured Query language(结构化查询语言）数据库以查询为主，98%都是查询操作。

### SQL分为三个部分：

DDL：data Definition Language 数据定义语言-->用来维护存储数据的结构（数据库，表）

代表指令：create ，drop ，alter

DML：data manipulation Language 数据库操作语言-->用来对数据进行操作（数据表中的内容）

代表指令:insert delete updata等其中DML内部又进行分类：DQL（database Query Language）数据查询语言，如select。

DCL：data contral Language 数据控制语言-->主要负责权限的管理（用户）

代表指令：grant ，revoke等。

### SQL简介：

Sql是关系型数据库的操作指令

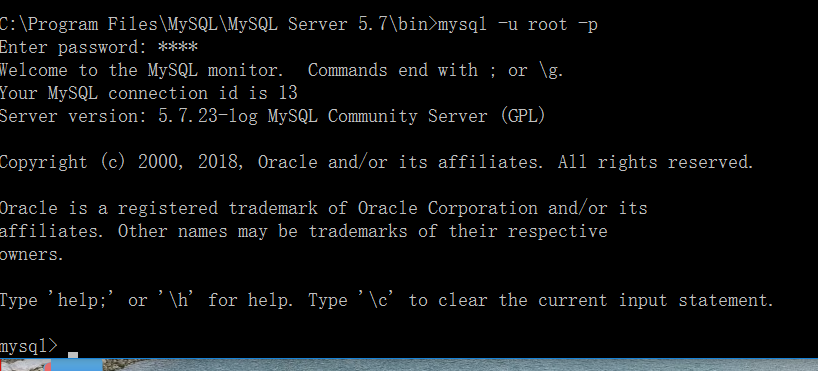
Sql是一种约束，但不强制（类似玩w3c组织）如Oracle，mysql内部有细微的区别。

### Mysql：

Mysql数据库是一种c/s结构，客户端/服务端若想要访问服务器必须通过客户端（服务器一直运行，客户端在需要的时候运行），

#### 交互方式

1. 客户端连接认证：连接服务器认证身份
2. 发送sql指令
3. 服务器接受服务器指令，处理sql指令返回操作结果。
4. 客户端接受操作结果，显示结果。
5. 断开连接（释放资源，服务器并发限制）。



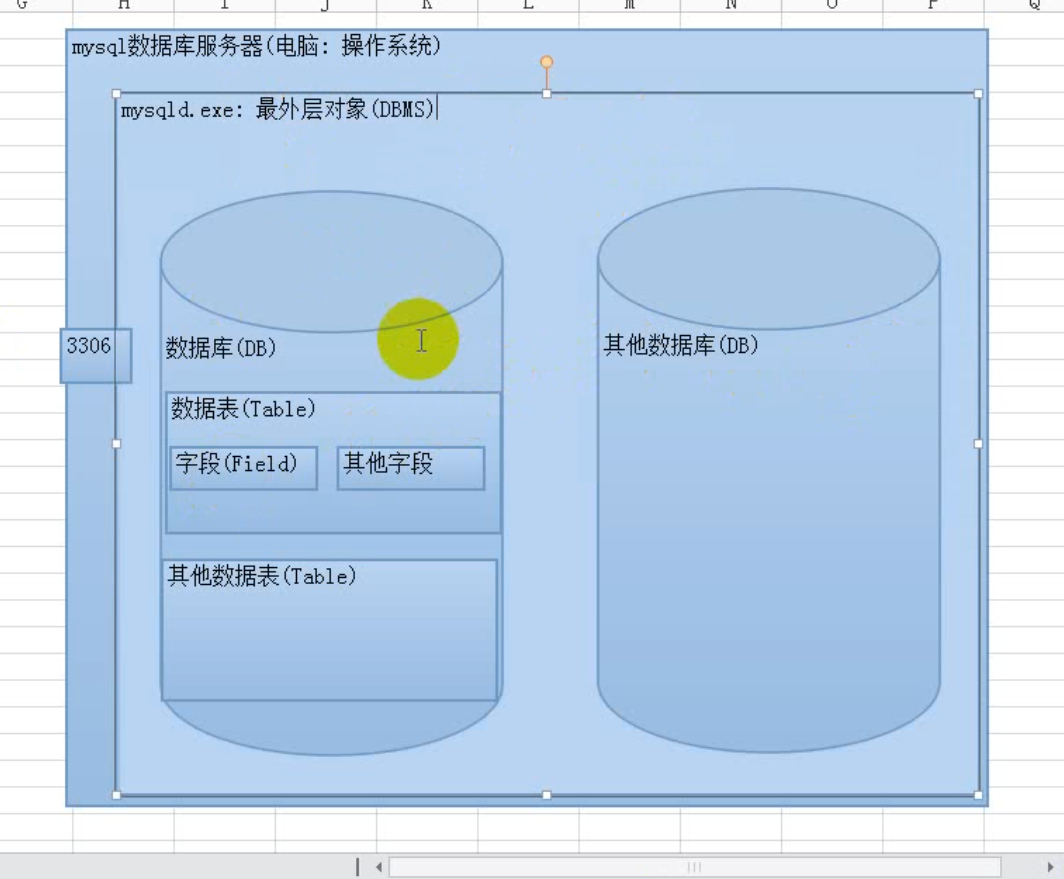
### Mysql服务器对象

没有办法完全理解服务器内部的内容：

只能粗略的分析数据库服务器内部结构

将mysql数据库分为四层：

系统(DBMS)->数据库(BD)->数据表(table)->字段（filed）



## SQL基本操作：

基本操作：CRUD

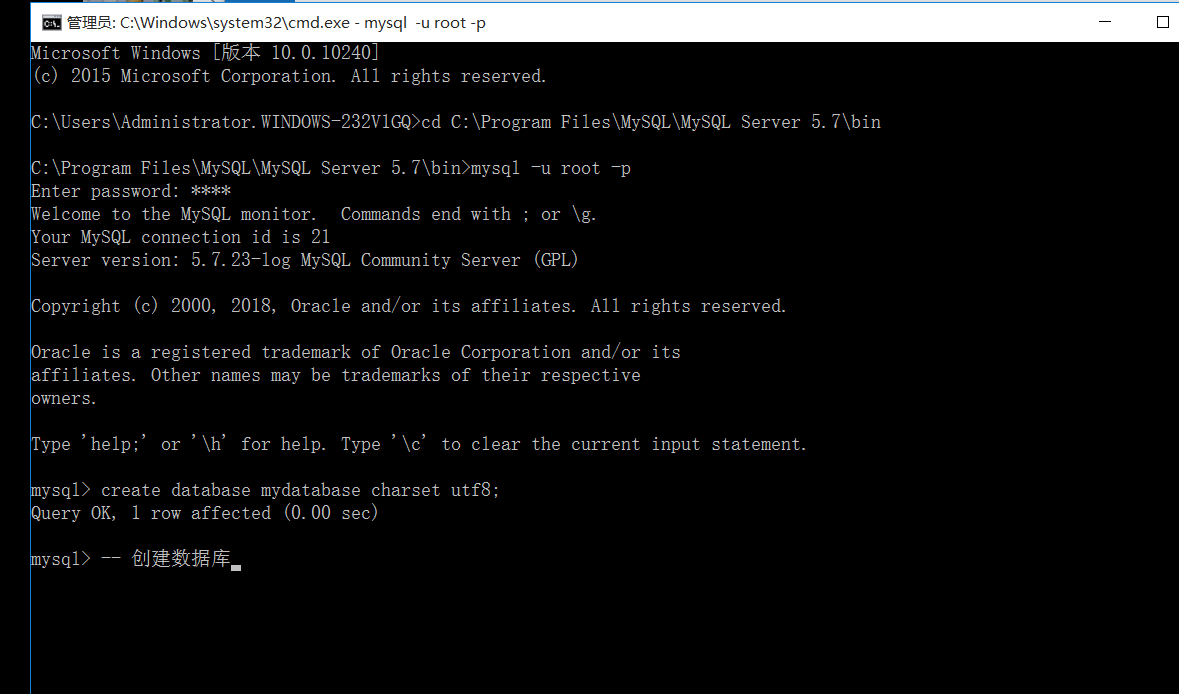
将sql的基本操作根据对象分为三类

1. 库操作2，表操作3，数据操作

### 库操作：

数据库的增删改查。

新增数据库：create database 库名 [库选项]



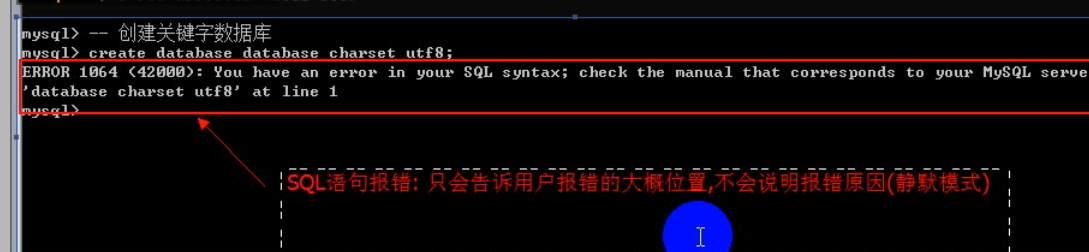
库选项：约束数据库，分为两个选项。

字符集的设定：charset /character set +具体字符集（数据存储的编码格式，一般两种格式GBK和utf8）

校对集设定：collate +具体校对集（数据库比较规则）

-- 双中下划线注释（单行注释，也可以用#）

其中数据库不能用关键字（已经被使用的字符或保留字（将来可能会用到））。



#### 查看数据库：

Show databases;



#### 查看指定部分数据库：模糊查询

Show databases like ‘patten’;--paatten是匹配模式

%：表示是匹配模式

\_：表示匹配单个字符；



#### 查看数据库创建语句：

Show create database 数据库名[库选项];

#### 更新数据库：

数据库名字不可以修改；

数据库的修改仅限于库选项：字符集和校对集（校对集依赖于字符集）；

Alter database 数据库名 [库选项];

Charset /character set [=] 字符集;

Collate [=] 校对集;

#### 删除数据库：

所有操作中删除最简单；

Drop database 数据库名;

当删除数据库语句执行以后发生了什么？

1. 在数据库颞部看不见对呀数据库。
2. 对应数据库存储文件夹内数据库名字对应的文件夹表也被删除。

注意：不要随便删除数据库

### 表操作：

表与字段（filed）密不可分

#### 新增数据表：

Create table 表名(字段名1 数据类型1，

字段名1 数据类型1

)[表选项]

表选项：控制表的选项

字符集：sharacter set /character 具体字符集;

校对集：collate 具体校对集;

存储引擎：engine 具体的存储引擎(innodb和myisam);

进入数据库环境 use 数据库名称

当创建表的sql指令执行以后发生了什么？

1. 数据库下存在相应的表，
2. 在数据库对应的文件夹下会产生对应的表结构文件。

#### 查看数据表：

数据库能查看的方式，表都能查看。

#### 查看所有表：

Show tables：

#### 查看部分表：

模糊匹配：show tables like ‘patten’;

#### 查看表的创建语句：

Show create table 表名;

#### 查看表结构:

Desc class;

Describe class;

Show colums from class;

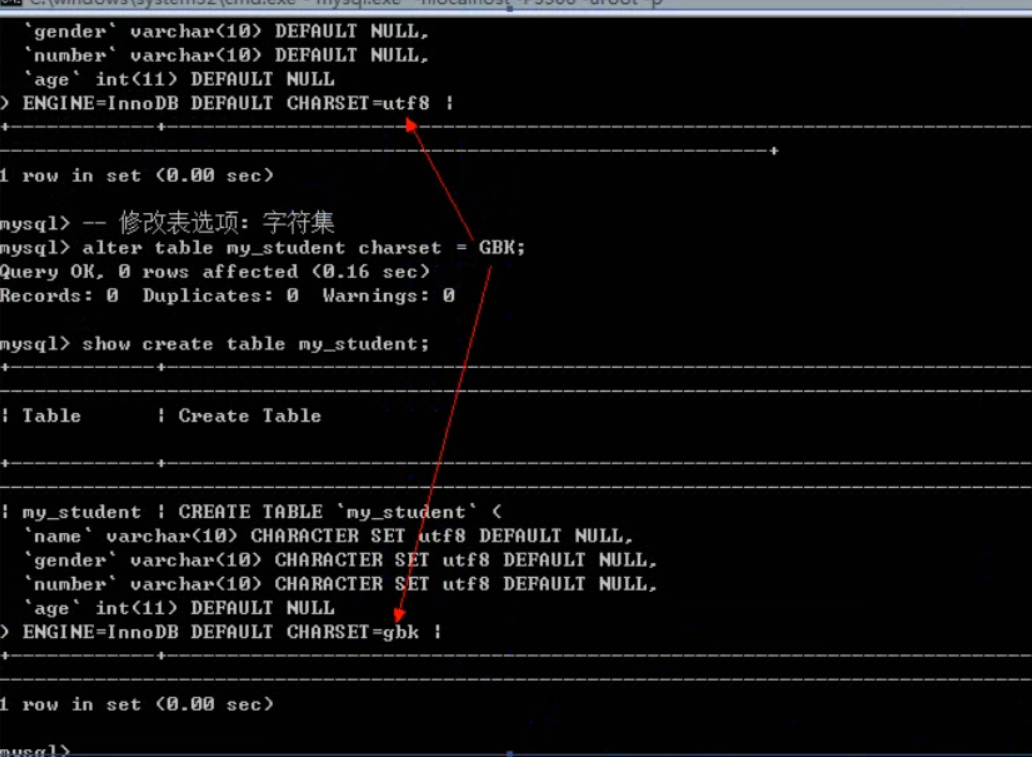
#### 修改数据表：

俩部分，修改表本身，修改字段

Rename table 老表名 to 新表名;

修改表选项：字符集

Alter table 表名，表选项 值;



##### 修改字段：

字段的操作很多新增、修改、修改、重名、删除。

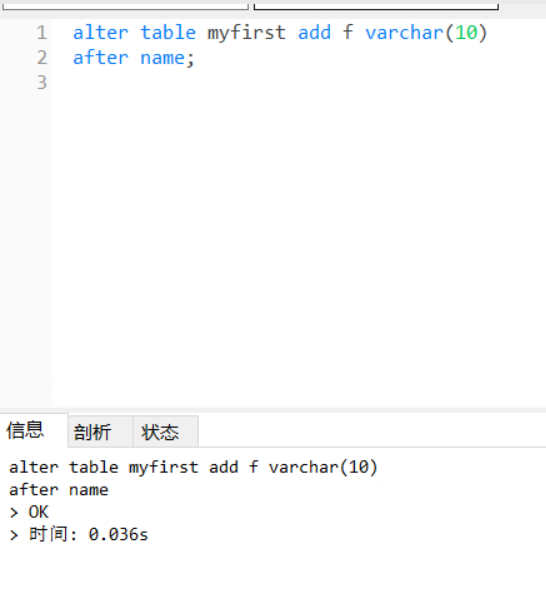
##### 新增字段：

Alter table 表名 add 字段名 数据类型 [列熟悉][位置];

位置：字段名可以存放在表中任何位置

First：第一位置。

Alter：在哪个字段之后，alter 字段名；默认在最后一个字段。



##### 修改字段：

通常是修改属性或数据类型

Alter table 表名 modify 字段名 数据类型[属性][位置];

##### 重命名字段：

Alter table 表名 change 旧字段 新字段 数据类型 [属性][位置];

##### 删除字段：

Alter table 表名 drop 字段名;

#### 删除数据表：

Drop table 表名1 表2--一次性多张

当删除数据表指令以后发生了什么？

1. 表空间没有指定表（数据没有了）；
2. 数据库对应的文件夹下对应的文件（与存储引擎有关）也会被删除

注意：删除有风险，操作不可逆。

### 数据操作：

#### 新增数据（两种）

方案1------给全字段插入数据，不需要指定列表：需要数据的值与表中设计字段出现顺序一致

Insert into 表名 values();

方案2------给部分字段插入数据，需要选定字段列表，字段列表顺序与字段顺序一致；

但是值列表顺序必须与选定字段顺序一致

Insert into 表名 （字段列表） values ([值列表]， [值列表]);

#### 查看数据

Select \*/字段列表 from 表名[while 条件];

#### 更新数据

Update 表名 set 字段 =值[while 条件];

#### 删除数据：不可逆

Delete from表名 [while 条件]

### 中文数据问题：

中文数据问题本质就说字符集问题，

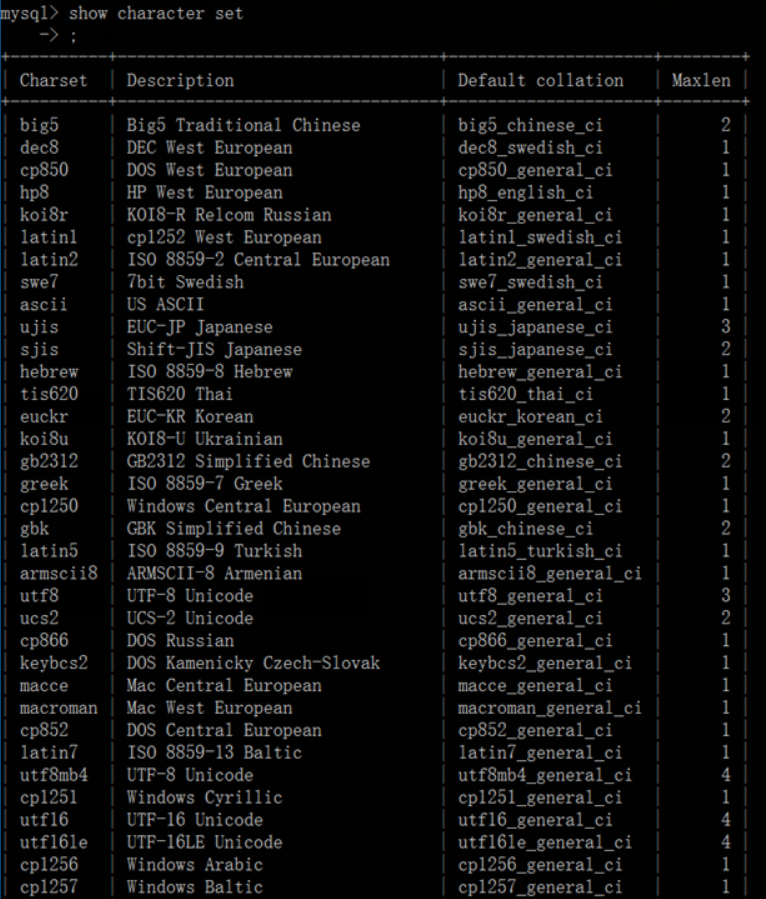
计算机只识别二进制，人类识别符号；需要友谊个二进制与字符对应关系（字符集）。

报错：服务器没有识别对应的四个字节。

服务器认为的数据是utf—8，一个汉字有三个字节，读取三个字节转换成汉字（失败），剩余的再读三个字节，最终失败！

所有数据库服务器认为（表现）的一些特征都是通过服务器端的变量来存储，系统先读取自己的变量，看看应该怎么表现。

Show viriables like ‘character\_set %’;（查看字符集）；对外处理默认字符集



基本上服务器是万能的，什么字符集都能识别。问题的根源：客户端只能识别GBK而服务器是utf-8，矛盾产生。

#### **解决方案1：**

改变服务器。默认字符集为GBK

Set character\_set\_client =GBK;

注意：数据的来源是服务器，解析数据是客户端（客户端只识别GBK；两个字节一个汉字）；但服务器给的数据却是utf8，三个字节一个汉字，乱码！！！！！！

#### 解决方案2：

修改服务器给客户端的数据数据集为GBK

Set character\_set\_result=GBK;

Set 变量=值 ；修改只会是会话级别（当前客户端：当次连接有效：关闭失效）

#### 快捷方式：

设置服务器对客户端的字符集的认识：

Set names 字符集;

例：set names gbk;

==>cheracter\_set\_client

==>character\_set\_result;

==>character\_set\_connection;

Connection: 连接层;字符集中间者统一了效率更高：

### 校对集问题：

校对集：数据比较的方式（三种方式）

\_bin:binary;二进制比较取出二进制位，一位一位进行比较，区分大小写。

\_ci:case sensitive; 大小写敏感，区分大小写；

\_ci:case insensitive; 大小写不敏感。不区分大小写。

检查数据库支持的校对集

Show collation;

对比：使用utf\_8的\_bin和\_ci来验证不同校对集的效果。

校对集:必须在没有数据之间声明好。

### Web乱码问题：

动态网站：部分构成；

浏览器：appache 服务器（php）数据库服务器

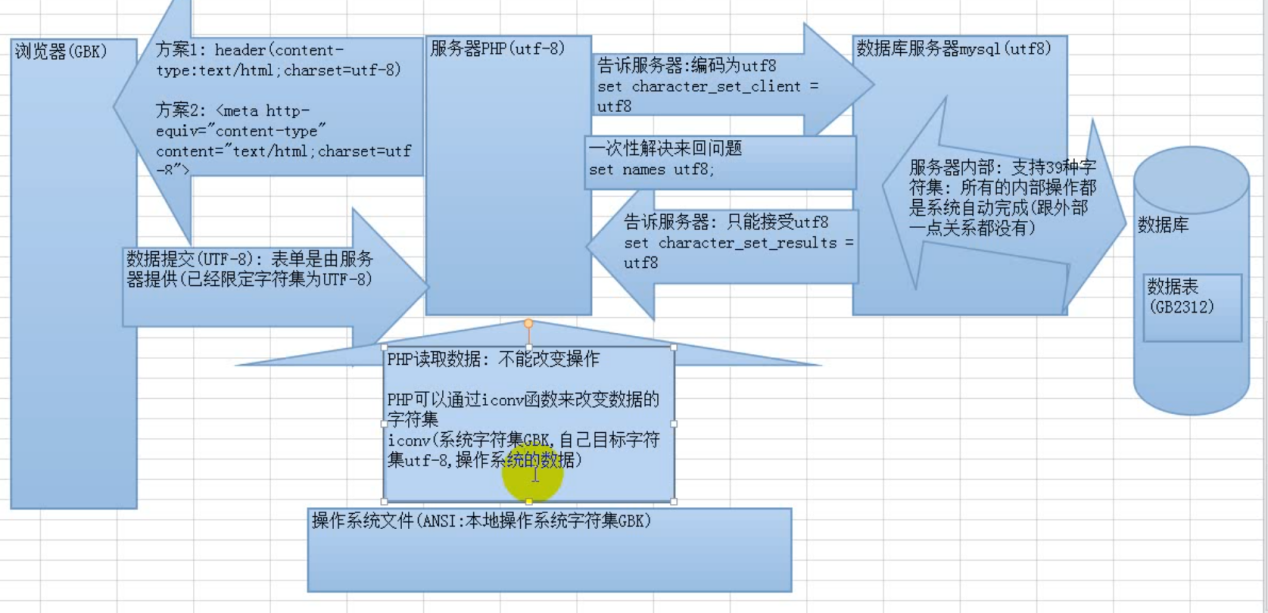
三个部分都有自己的字符集；数据在三个部分之间传递很容易产生乱码；

#### 如何解决乱码问题：

统一编码（三码合一）；

事实上不可能：浏览器用户管路（不能控制）

但是必须解决这些问题

，

## 回顾1

数据库基本知识：关系型数据库与非关系型数据库

关系型数据库：安全（磁盘）

非关系型数据库：高效（内存）

关系型数据库：建立在关系模型上的数据库，

数据结构：二维表（浪费空间）

数据库操作指令：sql（DDL,DML,DCL）

完整性约束：表内和表之间（实体）

### Mysql关系型数据库：

C/S结构（直接认证发送sql指令，服务器处理指令返回结果，客户端接受结果，分析结果）

Mysql服务器对象：DBMS----->DATABASE------->TABLE------>FIELD

数据库基本操作：库操作，表操作和数据操作。

字符集问题：

### 中文数据问题

1. 改变服务器接受数据字符集character\_set\_client
2. 改变服务器返回数据字符集character\_set\_result
3. 快捷方式：set names 字符集；

### 乱码问题：

浏览器解析：php处理

数据库处理

### 校对集问题：

比较规则：\_bin二进制,\_cs，\_ci大小写

算法：快速排序。

## 数据类型：

### 列类型：

所谓数据类型：对数据进行统一分类

从系统数据角度出发就是为了使用统一方式进行管理

更好的利用有限的空间

Sql将数据类型分为三类：

数值类型，字符串类型，时间日期型。

#### 数值型：

数值型数据：都是数据，系统将数据分为整数型和小数型

##### 整数型：

存放的都是数据

在sql因为要考虑节省磁盘

Tingint ：迷你整型，使用一个存储字节（常用），最多状态有256种

Smallint： 小整型，使用俩个字节，表示状态最多有65535种

Mediunint：中整型，使用三个字节存储

Int：表准整型，使用四个字节存储（常用）

Bigint ：大整型，使用八个字节存储。

在sql中数据库全部都是默认有符号的；分正负，使用无符号数据：给约定的数据类型限定

Int unsigned --无符号，从零开始。

查看表结构的时候，发现每个字段的数据类型之后都会带一个括号，里面有指定数值

显示宽度：数据最终显示的位数（包含符号）如-123是四位，253是三位，

显示宽度：没什么特别意义只是默认告诉用户显示的形式而已。实际上用户可以控制，不会改变数据的实际大小。

显示宽度的意义：在于当数据不够显示宽度的时候，会自动让数据变成对应的显示宽度：

通常需要搭配一个前导0来增加宽度，不改变值的大小：zerofill（零填充）

零填充的意义（显示宽度）：保证数据格式。

##### 小数型：

带小数点或者范围超过整型的数值类型。

Sql中将小数型分为俩种：浮点型和定点型

浮点型：小数点浮动，精度有限，会丢失精度。

定点型：小数点固定，精度有限，不会丢失精度。

###### 浮点型：

浮点型数据是一种精度型数据：因为超出规定范围之后会丢失精度（四舍五入）。

浮点型分为俩种：float：单精度，占用四个字节存储数据，大概七位。

Double：双精度，占用八个字节存储数据，大概十五位左右。

创建浮点数表：浮点的使用方式：直接float。

表示没有小数部分；float（M,D）M表示总长度，D表示小数部分长度，整数部分长度为（M-1）。 浮点型数据的插入：整数部分不能超过长度，但是小数部分可以超过长度（系统会自动四舍五入）结果：浮点数一定会进行四舍五入（超出精度范围）。

浮点数如果是因为系统进位导致整数部分超出指定长度：那么系统也会允许成立。

###### 定点型：

绝对的保持整数部分不会被四舍五入（不会丢失精度），小数部分有可能丢失（理论上小数部分也不会丢失精度）

浮点数如果进位导致长度溢出，没有问题。但是定点数不行、

###### 时间日期型：

Datetime:时间日期，格式是YYYY-MM-DD HH：ii：ss

Date ：日期就是datetime的date部分

Time：时间段，指定某个区间之间，时间到时间

Timestamp：时间戳，是从1970开始的格式与datatime一致，

Year：年份，两种格式，year（2）。

插入数据：

时间time可以说负数，year可以使用2位数插入，也可以使用四位数。

Timestamp字段：只要是当前所在记录别更新，该字段一定会自动更新成当前时间。

网站以php位实现主要操作对象，php有非常强大的日期处理函数：date，只需要一个时间戳就可以转换成任意类型函数的时间：以php为主的时候，都是在数据库使用时间戳（整型）来存储时间。

##### 字符串类型：

在sql中将字符串分为了6类

Char ，varchar，text，blob,set和enum

###### Char：定长字符串

定长字符串。磁盘在定义结果的时候就已经确定了最终数据的存储长度。

Char[L]:L代表length，可以存储长度，单位为字符最大长度值为255

例：char[4]在utf8环境下需要4\*3=12个字节

###### Varchar：变长字符串

Varchar在分配空间长度的时候，按照最大空间分配，但是实际上最终用了多少是根据具体的数据来确定，

Varchar（L）:L代表字符长度理论上长度65536个字符

但是会多1到2个字节来确定存储的实际长度

例：varchar（10）确实存了十个汉字，在utf8的环境下10\*3+1=31字节

如何选择定长还是变成字符串？

定长的磁盘空间比较浪费空间，但是效率高；

变长的磁盘空间比较节省但是，效率低；

如果数据基本上确定长度一样，就是定长如果数据不能确定长度（不同数据有变化）

###### 文本字符串：

如果数据量特别大，通常超过255个字符，就会使用文本字符串，文本字符串会根据存储数据的格式进行分类：text（文字存储）和blob（二进制数据，二进制存储路径）

###### 枚举字符串：enum

枚举：enum，事先将可能出现的结构都设计好，实际上就是存储数据必须是规定好的数据中的一个。

枚举使用方式

定义：enum（可能出现的元素列表）

如：enum(‘女’,’男’,’人妖’,’保密’,);

使用：存储数据，只能存储上面定义好的数据

例：创建枚举表，create table my\_enum(

Gender enum(‘男’,’女’,’人妖’) character utf8

加入数据；作用之一就是规范数据格式

数据只能是规定数据中的一个

Insert into my\_enum values(‘男’),(‘人妖’);

作用之二节省空间（枚举 别名 单选框）（存储的是数值）

在mysql中系统也是自动转换数据格式的，

证明字段存储的是数值：将取出来的+0

就可以判断出原来的数据到底是字符串还是数值，如果是字符串结果为0，否则为其他值。

例：select gender+0,gender from My\_enum;

找出枚举元素的规律，按照元素出现顺序从1开始编写

枚举原理：枚举在进行数据规范的时候，系统会自动建立一个数值与枚举元素的对应关系（关系放在日志中）然后在进行数据插入的时候，系统自动将字符串转换成对应的数字存储，然后进行数据提取的时候，系统自动将数值转换成对应的字符串显示。

###### 集合字符串：

集合跟枚举很类似：实际存储的是数值而不是字符串（集合是多选）

例：create table my\_set (hobby set (‘篮球’,’足球’,’乒乓球’))charset utf8;

插入数据：可以使用多元素字符串进行组合，也可以数值。

例：insert into my\_set values (‘足球’)

Insert into my\_set values(2)

集合中：每一个元素对应一个二进制位，被选中为一，没有则为零，最后方向反过来

例：110------>011=3

Select hobby +0 ,hobby from my\_set;

集合元素的顺序没有关系，最终系统会匹配顺序；

集合的强大功能在于能够规范数据和节省空间。

Php也可以规范数据但是对于php来说效率优先；而且数据的维护可以通过数值进行增加php的维护成本；php根本没有办法判断数据在数据库的形式。

### MySql记录长度

Mysql规定：任何一条记录最长不能超过65535个字节（varchar永远达不到理论值）

Varchar实际存储长度能达到多少呢，看字符集

在utf8下varchar的实际顶配varchar（21844）21844个字符-----21844\*3+2=65532+2=65534

Gbk下实际顶配varchar（32766）32766个字符 ------32766\*2+2=65532+2=65534

Mysql记录中如果任何一个字段允许为空，那么系统会自动从整个记录中保留一个字节存储在null（若想释放null所占用的字节，必须保证所有字段不为空）

Mysql中text文本字符串，不占用记录长度；额外存储，但是text文本字符串也是记录的一部分一定需要占据部分长度：10字节（保留数据的地址以及长度）

### 列属性：

列属性：真正约束字段的是数据类型，但是数据类型的约束很单一，需要一些额外的约束来保证数据的完整性

列属性：null/not null

Default

Primary key

Unque key

Auto\_increment

Comment

#### 空属性：null

Null（默认的和not null不为空）虽然是默认的，数据基本上都是字段为空，但是实际上真正开发的时候，尽肯能地保证所有的数据都不为空；空数据没有意义，空数据不能参加运算。

#### 列描述：comment

没有实际意义，专门用来描述字段，会根据表的创建语句保存（用来给dba）了解的。

例：comment’姓名’;

#### 默认值：default

某种数据会经常出现在某个具体的值

想要使用某个默认值，可以不一定指定列表，不使用字段列表，可以使用default关键字代替值

例：insert into my\_default values(‘张三’,18,default);

## 回顾2

字段类型（列类型）：数值型，时间日期型，字符串类型

数值型：整型，小数型（浮点与定点）

时间日期型：datetime，time，datetimestamp，year

字符串类型:定长，变成，

文本字符串（text和blob）

枚举与集合

Mysql记录长度：65535个字节

Null占一个字节

text文本不占用记录长度，本身十个字节

字段属性：空属性，列描述，默认值

## 字段属性：

主键，唯一键，自增长。

### 主键：

Primary key 主要的键，一张表只能有一个字段能使用对应的键，用来唯一约束该字段里面的数据不能重复，称之为主见。

一张表最多只有一个主键。

#### 增加主键：

Sql操作当中有很多方式给表增加主键，大体上有三种

##### 方案一：

创建表的时候，直接加在字段之后跟primary key

例：name varcahr(20) primary key;

##### 方案二：

复合主键

##### 方案三：

当表已经创建好之后，再次额外追加主键，通过修改表字段的属性，也可以直接追加

Alter table 表名 add primary key (字段列表) ;

前提是：字段名不重复；

#### 主键约束：

主键定义的字段中不允许重复；一旦重复数据操作失败(改，增)

更改主键&删除主键

Alter table 表名 drop primary

#### 主键分类：

在实际创建表的过程中，很少使用真业务数据作为主键字段（业务主键）

大部分使用逻辑性字段（字段没有业务含义）将这种字段称之为逻辑主键。

### 自动增长：

#### 自增长：

定义字段给的是默认值，不给值为null，会自动被系统触发，系统会从以前字段已有的最大值进行+1操作得到行的不同的字段。

自增长通常与主键搭配

#### **特点：**

auto\_increment

1. 任何一个字段要做自增长必须前提是本身是一个索引（key一栏有值）
2. 自增长字段必须死数字（int 整型）
3. 一张表最多只有一个自增长。

例：ID int primary key auto\_increment comment’自动增长’;

#### 自增长得使用

当自增长被给定的值为null 或者默认值的时候会自动增长

自增长如果对应的字段输入了值，那么自增长失效

下一次还是能够正确的自增长（最大值+1）

#### 修改自增长：

自增长如果涉及到字段改变，必须先删除自增长（一张表只能有一个自增长）。

修改当前自增长已经存在的值，修改只能比当前已有的自增长的最大值大，不能为null。

Alter table 表名 atuo\_increment=值

思考？为什么自增长都是从一开始为什么每次都是自增长一呢

所有系统的实现（如字符集，校对集）都是由系统内部变量控制的，查看自增长对应的变量

Show variables like ‘auto\_increment’;

可以修改变量实现不同的效果；修改是对整个数据而不是单张表（修改只是会话级）

Set auto\_incretment\_increment=值---自增值

#### 删除自增长:

自增长是字段的属性，通过modify来进行修改（保证字段没有auto\_increment）

Alter table 表名 modify 字段类型;

### 唯一键：

一张表往往有很多字段需要具有唯一性，数据不能重复；但是一张表只能有一个主键。唯一键（unque key）就可以解决表中有很多字段需要唯一性约束。

本质与主键一致，唯一键允许自动为空（空字段不参与唯一性比较）

#### 增加唯一键（三种）

##### 方案一：

在创建表的时候字段后直接跟unque/unque key

例：name char(10) unque;

##### 方案二：

在所有字段之后自动增加unque key（字段列表）---复合唯一键

Unque key (name);

##### 方案三：

在创建表之后增加唯一键

Alter table 表名 add unque key (字段);

#### 唯一键约束

唯一键与主键本质相同，唯一键区别就是唯一键默认允许为空，而且是多个为空

#### 更新唯一键和删除唯一键：

先删除后新增（唯一键可以有多个）

Alter table 表名 drop index 索引名字

### 索引：

几乎所有的索引都建立在字段之上。

索引：系统根据某种算法将已有的数据（未来可能增加的数据），单独建立一个文件，文件能够实现快速匹配数据并且能够快速地找到对应表中的记录

#### 意义：

1. 提升查询数据的效率
2. 约束数据的有效性（唯一性等）

增加索引的前提条件：索引本身会产生索引文件（有时候可能比数据文件还大）会非常耗费磁盘空间

如果某个字段作为查询条件经常使用你们可以使用索引（一定会想办法增加）

#### Mysql中提供了多种索引：

1. 主键索引
2. 唯一索引
3. 全文索引
4. 普通索引

##### 全文索引：

针对文章内部关键字进行索引，

全文索引最大的问题：在于如何确定关键字

英文很容易：英文单词与单词之间有空格，中文很难，没有空格，而中文可以多种随意组合（分词：spinx）

## 关系

将实体与实体的关系，反应到最终数据表的设计上来，将关系分为三种，一对多，多对多，多对多。

所有关系都是表与表之间的关系。

### 一对一:

一张表的一条记录一定只对应另外一张表的一条记录，反之亦然。

例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | 姓名 | 性别 | 年龄 | 电话号码 |
| 1 | 张三 | 男 | 23 | 13320848263 |

### 一对多

一张表中有一条记录，对应另一张表多条记录但是反过来，另一张表的一条记录只能对应这张表的一条记录。

例母亲与孩子的关系

### 多对多

一张表中的一条记录对应另外一张表的多条记录，同时另一张表的一条记录对应另外一张表的多条记录。

## 范式

Normal format

是一找种离散型数学中的知识，为解决数据存储的与优化问题，

数据保存以后，凡是能够通过关系找出来的数据，坚决不在重复存储：终极目标是减少数据的冗余。

范式是一种分层的结构规范，分为六层，每一层都比上一层严。若要满足下一范式，前提是满足上一层范式。

六层范式：1NF,2NF,3NF.4NF,5NF,6NF最高

Mysql属于关系型数据库有空间浪费：致力于节省存储空间，与范式所解决的问题不谋而活：在设计数据库，会利用范式来指导设计，但是数据不单是解决空间问题，要保护效率问题，范式只为解决空间问题，所以数据库的设计又不可能完全按照范式要求实现，一般只有前三种范式需求满足。

范式在数据库中有指导意义：但是不强制规范。

### 第一范式：1NF

在设计表存储数据的时候，如果表中设计的字段存储的数据，再取出来的使用之前还需要额外的处理（拆分）那么说表的设计不满足第一范式=>第一范式要求字段的数据具有原子性=>不可再分。

### 第二范式：2NF

再数据表设计过程中，如果有复合主键（多字段主键），且表中有字段并不是由整个主键来确定，而是依靠主键的某个字段（主键的一部分）=>存在字段约束主键的部分问题，称之为部分依赖=>第二范式就是要解决表中不允许出现部分依赖。

取消复合主键，使用逻辑主键。

### 第三范式：3NF

要满足第三范式，必须满足第二范式。

第三范式：理论上讲，y应该一张表中所有字段都应该有直接依赖主键（逻辑主键代表业务主键），如果表设计中存在一个字段，并不直接依赖主键而是通过某个非主键字段依赖，最终实现依赖主键。把这种不是直接依赖主键而是依赖非主键字段的依赖关系称之为依赖传递

### 逆规范化

磁盘利用率与效率的对抗，有时候再设计表的时候如果一张表中有就几个字段需要从别的表中去获取信息，理论上讲的确可以获得想要的数据，但是效率低一点，会刻意的在某些表中，不去保存外表的主键（逻辑主键）而是直接保存想要的数据信息这样一来在查询数据的时候，一张表可以直接提供数据，而不需要多表查询（效率低）。

## 数据的高级操作：

数据的操作：增删查改。

### 高级新增数据

Insert into 表名(字段列表) values (值列表)；

在数据插入的时候，假设主键对应的值已经存在，插入一定会失败。

主键冲突：

当主键存在冲突的时候（Duplicate key）

可以选择性进行处理：更新和替换

#### 高级操作更新：

Insert into 表名 [field主键] values (值列表) on duplicate key update 字段=新值;

主键冲突例子：insert into my\_sql values (‘php’,’B101’);

更新:insert into my\_sql values (‘php’,B101’) on duplicate key update room =’B102’;

#### 替换

Replace into 表名[字段主键] values (值列表)

例：replace into my\_sql values (‘Java’,’102’);

#### 蠕虫复制：

蠕虫复制：从已有的数据表中获取数据，然后将数据又进行新增操作，数据成部的增加。

表创建高级操作：从已有表创建新表（复制表结构）

Create table 表名 like 数据库.表名;

蠕虫复制：先查出数据，然后再将查出的数据新增n遍

Insert into 表名(字段) select 字段列表/\* from 数据表名；;

#### 蠕虫复制的意义：

1. 从已有表拷贝数据到新表中
2. 可以迅速让表中数据膨胀到一定的数量级；测试表的压极效率；

### 高级更新数据

Update

Update 表名 set 字段 =值[where条件];

### 高级新增语法

Update 表名 set 字段 =值[where条件][limit 更新数量];

例：update my\_copy set name =’c’ where name =’a’ limit =3;

### 删除数据与更新类似

Delete from表名 [where条件][limit 数量];

例：delete from my\_copy where name =’a’ limit=3;

删除：如果表中存在主键自增长，那么当删除之后自增长不会还原。

思路：数据的删除是不会改变表结构，只能删除表后重建

Truncate 表名; 先删除改变后新增改变

清空表，重置自增长。

### 查询数据

Select 字段列表/\* from表名 [where条件];

完整语法：

Select [select选项] 字段列表 [字段别名] /\* from数据源 [where条件子句] [group by 子句] [ having 子句] [order by 子句] [limit 子句];

#### select选项

Select对查询出来的结果处理方式

All：默认的保留所有结果。

Distinct：去重，查出来的结果，将重复的结构去除（所有字段相同）。

#### 字段别名

当数据查询出来以后，有时候名字不一定就满足需求（多表查询的时候，会有同名的字段）

就需要对字段名进行重命名：别名：字段名 [as] 别名;

#### 数据源

数据源：数据的来源，关系型数据库的来源都是数据表，本质上可以保证数据类似二维表，最终都可以作为数据源，

数据源分为多种：单标数据源，多表数据源，查询语句。

##### 单标数据源

Select \* from 表名;

##### 多表数据源

Select \*from 表名1，表名2.......;

从一张表中取出一条记录，去另外一张表中匹配所有记录，而且全部保留（记录数和字段数），将这种结果称之为笛卡尔积（交叉连接）

笛卡尔积没什么用，应该尽量避免。

#### 子查询

数据的来源是一条查询语句（查询语句结果是二维表）

Select \* from (select 语句) as 表名;

#### Where 子句

Where 子句，用来判断数据，筛选数据，

Where子句会返回结果：0/1；true/false

判断条件：运算符：<,>,<=,>=,!=,<>,like between and ,notin/in

逻辑运算符：&&（and），or

##### where原理

Where 是唯一一个直接从磁盘读取数据的时候就开始判断的条件，从磁盘取出一条记录，开始进行where 判断，成立保存到内存，失败则放弃

#### 条件查询

要求找：

#### Group by 子句

分组

根据某个字段分组（同为一组）

基本语法：group by 字段名

意义：统计数据（按分组字段进行数组统计）

SQL提供了一系列统计函数

Count(c):统计分组后记录数，每组多少记录。

Max():最大值

MIn():最小值  
Avg():平均数

Sum():求和

Count函数：里面可以使用两种参数，\*代表统计记录

字段名代表统计对应字段（null不统计）

分组会自动排序，根据分组字段：默认排序

Group by 字段[desc/asc] ；多分组结果然后合并之后的整个结果进行排序；

多字段分组：先根据一个字段进行分组，然后对分组厚度结果按照其他字段进行分组。

有一个函数：可以对分组结果中的某个字段进行字符串连接（保留该字段所有的字段）：

Group \_concat(字段)

回溯统计：with rollup ：任何一个分组后都会有一个小组，最后都需要向上级分组进行汇报统计：根据当前分组字段，回溯统计的时候会将分组字段置空

多字段回溯：考虑第一层分组会有此回溯：第二次分组要看第一层分组的组数，组数是多少回溯就是多少，然后加上第一层

#### Having子句

与where 子句一样；进行条件判断的，where是对磁盘数据进行判断，进入到内存以后经常分组操作，分组结果就需要having 来处理。

Having 能做where能做的几乎所有事情，但是where不能做having能做的很多事情。

1. 分组统计的结果就是或者是说统计函数都只有having 能够使用
2. Having 能够使用做的别名，where不能=>where是从磁盘读取数据，而名字可能是字段进入到内存后产生的。

#### Order by子句

Order by子句：排序，根据某个字段进行升序或者降序，排序依赖校对集。

基本语法：order by 字段名 [asc/desc]

Asc：是升序（默认的）

Desc：降序

排序可以进行多字段排序：先根据某个字段进行排序然后排序好内部。再按照某个数据进行排序。

Limit子句

Limit 子句是限制结果的语句，限制数量

Limit有两种使用方式

方案1：只用来限制长度（数据量）

Limit 数据量；

方案2：

限制起始位置，限制数量：limit 起始位置，长度

主要用来实现数据的分页：为当前用户节省时间提高服务器的响应效率，减少资源的浪费。

对于用户来讲：可以点击的分页按钮1,2,3,4

对于服务器来讲：根据用户选择的页码来获取不同的数据：limit offset，

Length;

Length：每页显示的数据量，基本不变；

Offset： offset=（页码-1）\*每页显示量。

## 回顾3

列属性：主键，自增长，唯一键。

关系：一对一，一对多，多对多

三层范式：

1NF：字段设计必须符合原子性

2NF：不存在部分依赖（没有复合主键）

3NF：不存在传递依赖（实体单独成表）

逆规范化：效率磁盘空间博弈；

高级数据操作：

新增操作：主键冲突（更新与替换），蠕虫复制

更新操作：限制更新数量：limit

删除操作：限制删除数量：limit。清空表结构：truncate

查询操作：select选项；

字段别名： 数据源（单表，多表，子查询[别名]）

Where子句（条件判断：从磁盘）

Group by子句（判断结构，针对分组统计结果）

Order by子句（排序，多字段排序）

Having 子句 （判断结果，针对分组进行统计结果）

Limit子句 （限制记录数，分页）

## 连接查询

连接查询：将多张表（可以大于2张）进行记录的连接（按照某个指定条件进行数据拼接）

最终结果：记录数有可能变化，字段数一定会增加（至少两张表合并）

连接查询的意义：用户查看数据的时候，需要现实的数据来自多张表。

### 连接查询分类

Sql将连接查询分为四类：外链接，自然连接，内连接，交叉连接

连接查询：join，使用方式 左表join 右表;

左表；在join关键字左边的表；

右表：在join 关键字右边的表;

### 交叉连接

Cross join从一张表中循环取出每一条记录，每张记录都去另一张表进行匹配。

匹配一定保留（没有条件匹配），而连接本身字段就全增加（保留），最终会形成的结果叫做笛卡尔积

#### 基本语法：

左表 cross join 右表;==from 左表,右表;

笛卡尔积没有意义：应该尽量避免（交叉查询没用）交叉连接的价值：保证连接结构的完整性。

例：select \* from my\_stu cross join my\_class;

### 内连接：

内连接：[inner] join;

从左表中取出没一条记录去右表中所有记录进行匹配，匹配必须是某个条件，在左表中与右表中相同的最终会保留，否则不保留。

#### 基本语法

左表 [inner] 右表 on 左表.字段=右边.字段;

On 表示连接条件;

条件字段就是代表相同的业务含义

如：内连接

Select \*from my\_stu a join my\_class b

On a.c\_id=b.id;

在php中会覆盖：

字段别名以及表别名对的使用：在查询数据的时候，不同表有同名字段，这个时候需要加上表名才能区分，而表名太长，通常使用别名。

内连接可以使用where 代替on关键字（where没有on效率高）

### 外链接：

外链接：outer join ，以某张表为主，取出里面所有的记录，然后每条与另一表进行连接，不管能不能匹配上条件，最终都会保留。

能匹配，正确保留，不能匹配其他字段制空null；

外连接分为两种：以某张表为主

以左表为主： left join ：左外连接

以右表为主 right join ：右外连接

#### 基本语法

左表 left/right join 右表 on 左表.字段=右表.字段；

例： select \* from My\_stu a left/right join My\_class b

On a.c\_id=b.id；

虽然左连接和右连接有主表差异，但是显示的结果永远是左表数据在左边，右表数据在右边，左连接和右连接可以相互转

### 自然连接

Natural join ：自动匹配连接条件，系统以字段名为匹配模式（同名字段作为条件，多个同名字段作为条件）

自然连接分为自然内连接。自然外连接

自然内连接 ：select \* from My\_stu

Natural join my\_class;

自然外连接 ：左表 natural left/right join 右表

例：select \* from my\_stu natural left/right join my\_class;

其实 内连接与外连接都可以模拟自然连接！！！！！！！！！

## 回顾4

### 连接查询:

连接多张表到一起，不管记录数如何，字段数一定会增加。

分类：内连接，外连接。自然连接，交叉连接，

交叉连接：cross join （笛卡尔积）

内连接：inner join，左右两张表有连接条件匹配（不匹配自动忽略）

外连接：left/right join 主表的一条记录一定会存在匹配保留副表数据，否则置空

自然连接：natural join 自动匹配（相同字段名）

PHP操作：mysql

Php充当客户端，开启mysql扩展

连接认证：mysql\_connect;执行sql

获取结果:mysql\_query;

解析结果：mysql\_fetch系列

释放资源：mysql\_free\_result和mysql\_close

错误处理：mysql\_ermo和mysql\_error

## 外键

Foreign key 外面的键

如果一张表中有一个字段（非字段）指向另一张表的主键，那么将该字段称之为外键

### 增加外键：

外键可以在创建表的时候或者在创建表之后增加（但是要考虑数据问题）一张表有多个外键

1. 创建表的时候增加外键：在所有的表字段之后使用：

Foreign key （外键字段） references 外部表（主键字段）

例：

Crate table My\_c (

c\_id int

Foreign key (c\_id) references

My\_class

) character utf8;

在新增之后增加外键；修改表结构

Alter table表名 add foreignkey(外键字段)

References 父表(主键字段);

例

Alter table my\_foreign add

Constraint student\_class\_1

Foreign key (c\_id)

Reference my\_class(id);

### 修改&删除外键

外键不可修改；只能先删除后新增

删除外键

Alter table 表名 drop foreignerkey 外键名;--一张表有个外键但是外键名称不能一样。

例：alter table my\_foreigner drop foreigner key

My\_foreigner\_id;

注意：外键删除是不能通过查看表结构体现出来，查看创建语句查看；

### 外键作用

外键默认作用：一个对父表，一个对子表（外键所在表）

对字表的约束：字表数据进行操作的时候如果对应的外键字段在父表找不到对应的匹配操作时便（约束字表数据）

对父表约束：父表数据进行写操作（删与改：主键）如果对应主键在字表中已经被数据引用，那么不允许操作。

### 外键条件

1. 外表要存在：首先要保证表的存储引擎是innodb（默认存储引擎）如果不是innodb，外键创建成功，但是没有约束效果。
2. 外键字段的字段类型（列属性）必须与父表的主键类型完全一致。
3. 一张表中外键名称不能重复
4. 增加外键的字段（数据已经存在），必须保证数据与父表主键要求一致。

### 外表约束

所谓外表约束就是指外键的作用

之前将外键作用：是默认作用，其实可以通过对外键的需求爱，进行定制操作。

外键约束有三种模式：针对父表。

District ：严格模式（默认的）父表不能删除或者更新一个已经被子表数据引用的数据。

Cascade：级联模式，父表的操作，对应子表关联的数据也跟着操作。

Set null: 置空模式，父表的操作之后，子表的数据被置空。

### 约束模式：

删除的时候子表置空，更新的时候子表级联操作。

指定模式的语法：foreign 可以（外表字段）references 父表（主键字段）on delete 模式 update 模式;

### 更新操作：级联操作

删除操作：置空

删除置空前提条件：外键字段允许为空（如果不满足条件），外键无法创建。

外键很强大，能够进行各种约束；但对php来讲，外键约束降低了php对数据的可控性，但是实际开发中，很少用外键来处理。

## 联合查询

联合查询：将多次查询（多条select语句）在记录上进行拼接（字段不会增加）。

语法：多条select语句构成，每条select语句获取的字段必须严格一致（但是字段类型无关）。

Select 语句1

Union [union选项]

Select语句2........

Union 选项有两个

All：保留所有

Distinct：去重。默认的

联合查询只要求字段一样，与数据类型无关。

意义：两种

1. 查询一张表但是需求不同。
2. 多表查询，多张表的结构完全一样，保存的数据也一样

### Order by

使联合查询中order by 不能直接使用需要对查询语句使用括号。

例：

(select \* from my-class where sex=’男’ order by age asc limit 999)

Union

(select \* from my-class where sex=’女’ order by age desc limit 999);

若要order by生效，必须搭配limit ，limit最大值。

## 子查询

子查询：subquery 查询是在某个查询结果之上进行的（一条select语句含另一条select语句）

子查询分为两种：按位置分，按结果分

### 按位置分类：子查询select语句在外部查询

Select语句出现的位置

Form子查询：子查询跟在form后面。

Where 子查询：子查询出现在where条件中

Exists子查询:子查询出现在exists里面

### 按结果分类：根据子查询得到的数据进行分类（理论上讲任何一个查询得到的结构都可以理解为二维表）

标量查询：子查询得到的结果是一行一列

列子查询：子查询得到的结果是一行多列

行子查询：子查询得到的结果是一行多列（多行多列）

表子查询：子查询得到的结果是多行多列（出现位置在form之后）

### 标量子查询

例：需求：知道名字php获取学生

Select \*from my\_class where c\_id =(select id from my\_student where c\_name=php);

### 列子查询

例：需求查询所有在读班级的学生（班级表存在的班级）

1. 确定数据源：学生

Select \* from my\_sutdent where c\_id in (?)

2,确定有效班级id，所有班级id

Select id from my\_class;

合：Select \* from my\_sutdent where c\_id in (Select id from my\_class);

列子查询返回的结果比较：一行多列，需要使用in作为条件匹配，其实mysql还有几个类似：all ，some，any===in；

### 行子查询：

行子查询返回的结果可以是多行多列（一行多列）、

例：需求:要求查询整个学生中年龄最大，身高最高的学生

1. 确定数据源

Select \*from my\_student where age =? and heigth=?;

2,确定最大的年龄和最高的身高

Select max（age），max（height） from my\_student;

行子查询：需要构造行元素，行元素多个字段构成

Select \* from my\_student where (age,height)=(select max(age),max (height)from my\_student);

表子查询：

表子查询：子查询返回的结果是多行多列的二维表，子查询的结果当做二维表来使用

例：需求：找出每班最后一个的学生

1. 确定数据源；先将学生安装身高进行降序或者是升序

Select \*from my\_student order by height desc ;

1. 从每个班选出第一个学生

Select \* from my\_student group by c\_id;

表子查询：from子查询得到的结果作为from的数据源。

### Exists 子查询

Exists 是否存在，exists子查询就是用来判断某些条件是否满足（跨表）exists是接在where之后，exists返回的结果只有0和1

例：查询所有的学生，前提条件是班级存在

1. 确定数据源

Select \*form my\_student where?

2,确定条件是否满足

Exists (select \* from my\_class)--是否成立

## 视图

视图：view是一种有结构的但是没有结构来源的虚拟表，虚拟表的结构来源不是自己定义的而是从对应的基表中产生（来源）

### 创建视图

基本语法：

Create view 视图名字 as select 语句

可以是普通查询，也可以是连接查询，联合查询，子查询。

#### 创建单表视图：

基表只有一个

例：create view my\_view as select \*from my\_class;

#### 创建多表查询

基表有多个

### 查看视图：

查看视图结构

视图是一张虚拟表，表的查看方式适用于

视图：show tables ,desc 表名

Show create table 表名\G

视图与表还是有一个关键字的区别：view

查看表（视图）的创建语句的时候可以使用view关键字

视图一旦创建：系统就会在视图对应的数据库文件夹下创建一个对应的结构文件：frm文件

### 使用视图

使用视图：主要是为了子查询：将视图做为表一样

查询：视图的执行：其实本质就是执行封装的select语句

### 修改视图：

视图本身不可修改，但是视图的来源可以修改

修改视图：修改视图本身的来源语句（select 语句）

Alter view视图名字 as 新的select语句

例：alter view my\_view as select id,name,age,sex from my\_stu;

### 删除视图

Drop view 视图名字;

### 视图的意义

1. 视图可以节省sql语句：将一条复杂的查询语句使用视图进行保存，以后可以直接对视图进行操作；
2. 数据安全：视图操作是主要针对查询，如果对视图结果进行处理(删除)不会影响基表数据（相对安全）
3. 视图往往是在大型项目中使用，而且是多系统使用，对外提供有效有用的数据隐藏关键（无用的）数据，数据安全。
4. 视图对外提供友好型，不同的视图提供不同的数据，对外好像是专门设计
5. 视图可以更好地进行权限控制

### 视图数据操作

视图可以进行很多数据写操作：但是有限制，将数据直接在视图上进行操作

#### 新增数据

数据新增直接对数据进行新增

1. 多表数据不能进行新增操作
2. 可以向单表视图插入数据，但是视图中包含的字段必须有基表所不能为空（或者没有默认值）字段
3. 视图可以向基表插入数据（不应该给权限）

#### 删除数据

1. 多表视图不能删除数据
2. 单表视图可以删除数据（对基表进行操作）‘delete from my\_view2 where id=2’

#### 更新数据

理论上单表与多表视图都可以更新数据库

Update my\_view1 set id=3 where id=2;

更新限制：with check option 如果对视图在新增的时候，限定了某个字段有限制；那么在对视图进行数据更新操作之后，数据依然可以被实体查询出来，否则不更新。

例：select \* from my\_stu where age>30 with check option---表视图数据来源年龄不能>30岁，with check option 决定更新的时候不能将age >30改成<30。

### 视图算法

系统对视图以及外部查询视图select语句的一种解析方法。

视图算法有三种

1. undefind：未定义（默认的），这不是实际使用算法，推卸责任的算法告诉系统，视图没有定义算法，自己看着办吧！
2. Temptable:临时算法，系统应该先执行视图的select语句，后执行外部查询语句
3. Merge：合并算法，系统应该先将视图对应的select语句与外部查询视图的select语句进行合并；然后执行（效率高，常态）；

算法指定：创建视图的时候

Create algorithm =指定算法 view 视图名字 as select 语句;

#### 视图算法的选择

如果视图的select语句中包含一个查询子句（五子句？？？）而且很有肯比外部查询语句要靠后，一定要使用temptable 其他情况不指定。

## 数据备份与还原

备份：将当前已有的数据保留。

还原：将已经保留的数据恢复到对应表中

为什么要做数据备份

1. 防止数据丢失，被盗，误操作
2. 保护数据记录

数据备份还原方式有多种：数据表备份

单表数据备份

Sql备份

增量备份

### 数据表备份：

数据不需要sql来备份，直接进入到数据库文件夹复制对应结构以及数据文件，以后还原的时候，直接将备份的内容放入即可

数据表备份有前提条件：根据不同的存储引擎有不同的区别。

存储引擎：mysql进行数据存储的方式主要有两种：innodb，myisam（免费）

Innodb：只有表结构，数据存储到ibdatal文件夹中

Myisam：表，数据和索引单独存储

Myisam：Fim是结构，myd是数据，myi是索引

这种备份通常适用于myisam存储引擎直接复制三个文件夹即可，放到对应的数据库可以使用，innodb可以复制但是不能使用。

### 单表数据备份

每次只能备份一张表，只能备份数据（表结构不能备份）

通常使用：将表中的数据进行到出到文件

备份：从表中选出一部分数据保存到外部文件中。

Select \*/字段列表 into outfile 文件路径 from 数据源;--前提是外部文件不存在

Select \* into outfile “d:new.txt” from myfirst;

高级备份：自己制定字段和行的处理方式：

Select \*/字段列表 into outfile 文件路径 fields 字段处理 lines行处理 from 数据源。

Fileds ：字段处理

Enclosed by ：字段使用什么内容包裹，默认是：空字符串

Terminate by：字段以什么结束，默认是“\t”“tab”键

Escaped by： 特殊字符用什么方式处理，默认是“\\”使用反斜杠转译

Lines ：行处理

Start by：每行以什么开始默认是空字符串

Terminate by: 每行以什么结束，默认是“\r\n” 换行符

------------指定备份处理方式

Select \* into outfile “D:new.txt”

Fields enclosed by “ ”--字段处理

Lines starting by “ ”--行处理

From my\_class;

数据还原：将一个外部的数据重新恢复到表中（如果表结构不存在）

Load data infile 文件路径 into

Table 表名[字段列表]

Fileds 字段处理

Lines 行处理

怎么备份怎么还原。

例：

Load data infile “d:new.txt”

Into table my\_class

Fields encosed by” ”

Lines starting by ‘start ’;

### Sql备份：

备份sql语句：系统会对表结构以及数据进行数据处理，变成对应的sql语句然后进行备份；还原度时候执行SQL指令即可（主要是针对表结构）

备份：

Mysql 没有提供指令需要利用mysql提供的软件mysqlDump.exe

Mysqldump.exe也是一种客户端，需要操作服务器，必须连接认证。

Mysqldump / mysqldump.exe -u -p 数据库名字[数据表名字1数据表名字2]>外部文件目录（建议使用.sql结尾）

#### Sql还原数据

两种方式还原，

1. 使用mysql.exe客户端还原：mysql.exe / mysql -u -p 数据库名字 <备份文件目录
2. Sql指令还原sql备份 source备份文件所在路径

#### Sql备份优缺点

1. 优点，可以备份结构，尽管结果不管过程
2. 确定，会浪费空间（额外增加sql指令）

增量备份（大项目用增量备份）

不是针对数据或者sql指令进行备份：是针对mysql服务器日志文件备份。

增量备份：指定时间开始进行备份，备份数据不会重复，而且所有的操作都会备份。

## 回顾

外键：关联关系（表与表之间，表中字段指向另外一张表的主键）

条件：字段类型一致，存储引擎必须是innodb

约束：子表约束，不能插入父表不存在的记录

父表约束，三种约束模式。

（district ，cascade，setnull）

On delete set null on update cascade;

联合查询：union 多表合并和单表不同查询条件，联合查询使用order by：select语句使用括号：还必须配合：limit

子查询：查询语句中有查询语句，

分类：按位置：from子查询，where子查询，exists子查询

按返回结果：表量子查询，列子查询，行子查询，表长查询。

视图：view

节省SQL语句：安全级别

本质：虚拟表，有结构，无数据

视图数据操作：多表只能改，单表可以增加（有限制）

视图算法：undifined 未定义

Temptable 临时表

Merge 合并

文件备份与还原

文件备份:存储引擎（myisam）

单表数据备份：只能备份数据

Sql备份：备份sql指令（mysqldump.exe客户端）

增量备份：备份系统日志文件。

## 事物：

一系列要发生的联续的操作

### 事物安全：

一种保护连贯操作同时满足实现的机制

事物安全的意义：保证数据操作的完整性。

事物操作：分为两种：自动事物（默认的），手动事物。

### 手动事物：

操作流程

1. 开启事物：告诉系统所有操作系统（写）不要写入到数据表，先存放到事物日志。

Start transaction；

1. 进行事物操作：一系列操作
2. 关闭事物：选择性将日志文件中操作的结果同步到数据表（同步保存）或者说字节清空事物日志。

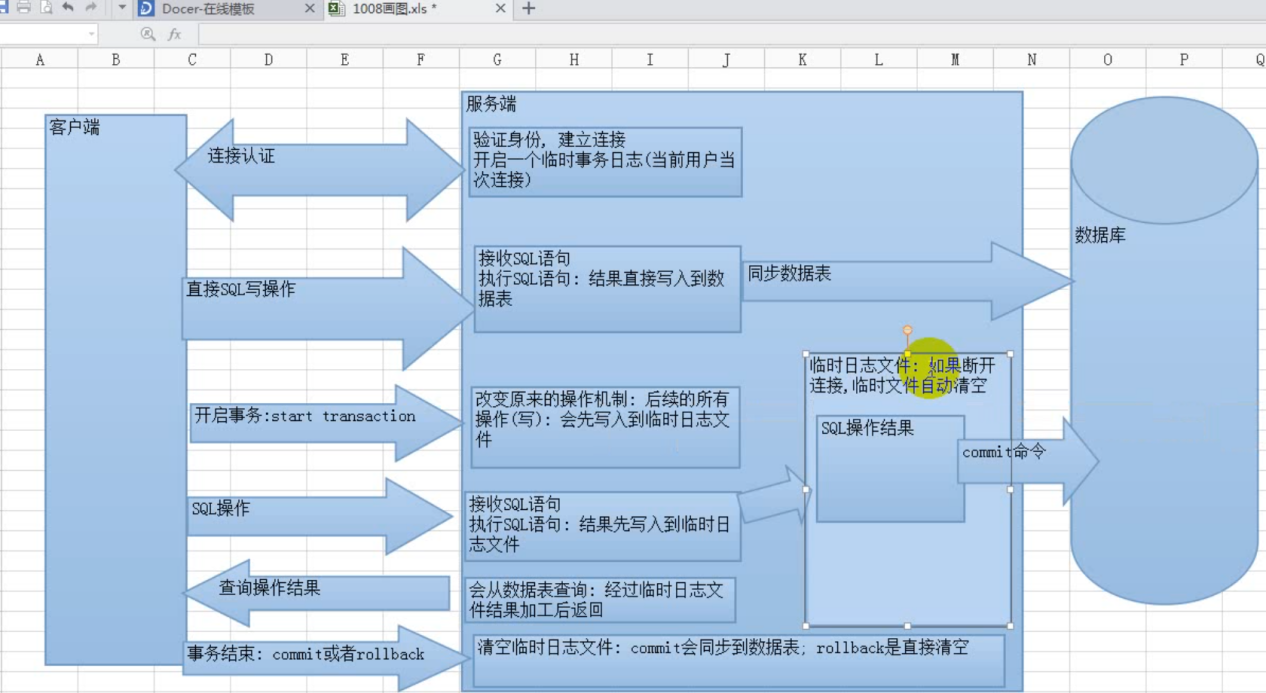
Commit ：提交事物，同步数据表（操作成功）

Rollback ：回滚事物，直接清空日志表（操作失败）。

### 事物原理

事物开启之后，所有操作会保存到事务日志，事务日志只有得到commit命令才会同步到数据表。

其他任何情况都会清空（rollback，断电，断开连接）。



### 回滚点

在某个成功完成之后，有可能成功，也有可能失败，但是不管成功还是失败，前面的操作已经成功，在当前成功的位置设置一个点，可以供后续失败操作返回该位置，而不是返回所有操作。这个点称之为回滚点

语法：sava point 回滚点名字；

回到回滚点语法：rollback to 回滚点名字

### 自动事物处理：

在mysql中默认的是自动事物处理，用户操作完全会立即同步到数据表

自动事物：系统通过autocommit控制：show variables like ‘autocommit’;

关闭自动提交事物：set autocommit=off/0;

自动关闭之后，需要手动选择处理：commit，rollback。

注：通常使用自动事物

### 事物特性

ACID四大特性

A：Atonic原子性；事物的整个操作是一个整体，不可分割要么全部成功要么全部失败。

C：Consistency一致性：事物操作前后，书记表中数据没有变化。

I: Isolation 隔离性，事物操作是相互隔离不受影响的

D：Durability 永久性，数据一旦提交不可改变。

### 锁机制

Innodb默认是行锁，但是事物如果是在操作过程中，如果没有使用到索引，那么系统会自动全表检索，自动升级为表锁。

行锁：只有当前行被锁住，别的用户不能操作

表锁：整张表被锁住，别的用户不能操作

## 变量：

变量分为两种，系统变量和自定义变量

### 系统变量：

系统定义好的，大部分用户更本不需要使用系统变量，系统变量是用来控制服务器表现的：如autoCommit 等atuo\_increament\_increment

查看系统变量show variables;查看所有系统变量，查看具体变量值，任何一个有数据返回的内容都是由select 查看。

Select@@变量名

### 修改系统变量

修改系统变量分为两种方式：会话级别和全局级别

会话级别：临时修改，当次连接有效 set 变量名=值

全局级别：一次修改，永久生效（所有客户端生效）set global 变量名=值

例：set global autocommit=0；

如果对方（其他）客户端已经连上服务器，那么当次修改无效，要退出重新登录才会有效

自定义变量：

定义变量

系统为了区分系统变量，规定用户自定义变量，必须使用一个@符号 set@变量名=值

用户定义变量类似于系统变量查看 Select @变量名;

在sql中“=”会默认的当做比较符号处理。Mysql为了区分比较和赋值的概念：重新定义一个新的赋值符号“:=”

mysql允许从数据表中获取数据然后赋值给变量(两种方式）

1. 变赋值变查看结果

Select @变量名：=字段名 from 数据源;--从字段取赋值变量名如果使用=会变成比较

1. 只有赋值，不看结果：要求很严格，数据记录最多允许获取一条：mysql不支持数组

select 字段列表 from表名 into 变量列表

注：所有自变量都是会话级别；当前客户端当次连接有效

所有自定义变量不区分数据库（会话级别）

## 触发器

Trigger 实现为某张表绑定一段代码，当表中的某些内容发生改变的时候（增删改）

系统会自动触发代码执行。

触发器：事件类型，触发对象，触发时间

事件类型：三种（增删改）insert ，delete，update

触发时间：前后，before ，after

触发对象：表中每一条记录（行）

一张表只能有一种触发时间的一种类型的触发器，最多一张表能有6个触发器

### 创建触发器

在mysql高级结构中，没有大括号，都是用对应的字母符号代替 --临时修改语句结束符

触发器的基本语法

Delimiter 自定义符号：后续代码碰到自定义符号结束

Create trigger 触发器名字 触发时间（事件），触发事件（类型）， on 表名 for each row

Begin --代表大括号：开始

--里面是触发器内容：每行必须使用语句结束符；分号

End --代表大括号：结束

自定义符号

Delimiter； --将临时符号修改过来

### 查看触发器：

Show trigger [like ‘patten’];

查看触发器创建语句

Show create trigger 触发器名字

所有的触发器都会保存在一张表中

information\_Schema.trigger

### 使用触发器

触发器：不需要手动触发，而是当某种情况发生时会自动触发

### 修改触发器&&删除触发器

Drop trigger 触发器名字;

### 触发器记录;

不管触发器是与否触发，只要当某种操作准备执行：系统就会将当前操作的记录的当前状态和即将执行之后新的状态分别保留下来，供触发器使用。

其中要操作的当前状态保存到old中，操作之后的可能形态保存到new，删除的时候没有new的，插入的时候没有old。

Old与new都是代表记录本身，任何一条记录除了有数据，还有字段名字。

使用方式：old字段/new 字段名（new代表假设发生之后的结果）

如果触发器内部只有一条要执行的sql指令，可以省略大括号（begin与end）

Create trigger 触发器名字 触发时间 时间类型 on 表名 for each row 一条sql指令;

触发器：可以很好的协调内部的数据处理顺序和关系，但是从php角度出发，触发器会增加数据库维护难度，所以较少使用触发器。

## 代码执行结构

三种，顺序结构，分支结构，循环结构。

### 分支结构：

事先准备多个代码块按照条件选择性执行某段代码。

在mysql中只有if分支

If 判断条件 then

-- 满足条件执行的代码

Else

-- 不满足

End if

### 循环结构：

某段代码在指定条件执行重复执行

While （循环）

While 条件判断 do

--满足条件执行代码

--变更循环条件

End while;

循环控制：在循环内部判断和控制

Mysql没有对应continue 与break

Iterate:迭代 类似于continue，后面的代码不执行，循环重新来过

Leave：离开类似于break，循环结束

循环名字：while条件do

--循环体

--控制

Leave/iterate 循环名字

End while;

## 函数

将一段代码块分装到一个结构中，在需要执行代码块的时候，调用结构执行即可；

函数分为两类：系统函数和自定义函数

### 系统函数

系统定义好的函数直接调用即可任何函数都有函数返回值因此函数的调用是通过select 调用，mysql中字符串是基本操作单位名称（字符）

subString ：字符串截取（字符为单位）

例

Select subString（@name ,1,1）

Set @name=”您好”;

截取位置从1开始：mysql中字符串下表从1开始截取单位为字符

Char-length:字符串长度

Length：字节长度

Instr：判断字符串是否在某个具体的字符串存在

Lpad：左填充。将字符串按照某个指定的填充方式填充到指定长度（字符）

Insert：替换，找到目标位置，指定长度字符串替换成目标字符串

Strcmp：compare，字符串比较

### 自定义函数

函数：函数名，参数列表（形参与实参）

返回值，函数体（作用域）

创建函数：create function 函数名（形参列表）return 数据类型 --规定返回的数据类型

Begin

---函数体

--返回值：return类型（指定数据类型）

End

自定义函数与系统函数调试方式是一样的

Select 函数名([实参列表])

### 查看函数

查看所有函数：show function status\G

查看函数创建语句：show create function 函数名

删除函数

Drop

Function

函数名

### 函数参数

参数：定义时叫形参，调用时叫实参（实参可以使数值，可以使变量）

形参：指定数据类型

Function 函数名称（形参名字 字段类型） returns 数据类型

注：任何变量要修改必须是使用set关键字

### 作用域：

Mysql 中作用域与js的作用域一样全局变量可以向任何地方使用，局部变量只能在函数内部使用

全局变量：使用set关键字定义，使用@符号标志

局部变量：使用declare关键字声明没有@符号。

所有局部变量声明，必须是在函数体开始之前

## 存储过程

存储过程：过程procedure是一种用来处理数据的方式（增删改）

存储过程是一种没有返回值的函数

### 创建过程

Create produce 过程名字（参数列表）

Begin

--过程体

End

查看过程：关键字procedure

Show procedure statue [like ‘pattern’];

查看过程创建语句：show create procedure 过程名;

调用过程：过程没有返回值，select语句不能访问

过程有一个专门的调用关键字：call

Call 过程名。

修改过程&删除过程

过程只能删除不能修改

Drop procedure 过程名;

Out与intout 是一种引用类型，内部修改会影响到外部。

### 过程参数

函数的参数需要类型指定，过程比函数跟严格

过程需要自己的类型限定（三种）

In：数据只能是从外部传入内部使用（值传递）可以使数值，可以是变量。

Out：只允许过程内部使用（不用外部数据）给外部使用（引用传递：外部数据会被清空才进入到内部）：只能是变量。

Inout：外部可以在内部使用，颞部修改可以给外部使用，典型的引用传递；只能传变量。

#### 基本使用

Create procedure 过程名 （in 形参名字，数据类型，out 形参名字 数据类型，inout 形参名字 数据类型）

注：调用过程名，inout，out必须是变量

存储过程对于变量的操作（返回）是滞后的，是在存储过程调用结束的时候，才会将内部修改的值赋值给外部传入的变量。

# Mysql基础学习到此结束谢谢大家