昨日回顾 2

数据类型 4

概览 4

整数类型 4

范围 5

通用设定形式 5

小数类型 5

时间日期类型 6

字符串类型 6

最基本最重要的2个： 6

2个二进制文本： 6

2个大文本类型： 6

2个有关“选项”的文本存储形式： 6

表定义语句 8

创建表： 8

基本形式 8

字段属性设置 8

索引设置 8

约束设置 9

表选项 10

修改表： 11

一般概述 11

修改表的基本形式： 12

删除表： 12

表的其他操作： 12

视图 12

什么是视图： 12

视图创建形式： 12

视图的使用 13

修改视图： 13

删除视图： 13

数据库（数据表）的设计思想介绍 13

数据库设计3范式（3NF）： 14

第一范式（１ＮＦ），原子性 14

第二范式（２ＮＦ）唯一性 14

第三范式（3NF）：独立性，消除传递依赖 16

最后的总结 18

昨日回顾

二分查找：

前提：1，针对索引数组

2，已排序

效率：

数据量： 计算的次数

1024 10

约100万 20

约1600万 24次

约20亿 31次

约40亿 32次

数据库基础

数据库的分类：网状数据库，层次数据库，关系数据库

关系数据库：

数据都是以“分门别类”的方式存储在一个一个“表”中，每个表理论上都只存储某类信息。

表跟表之间的数据可以建立一定的对应显示应用需求的关系。

关系数据库常见术语：

数据库操作的基本模式（流程）

1：客户端——建立连接——服务器端

2(反复）: 客户端——命令——服务器端，接受，执行——返回结果——接收结果——处理（显示）

3：断开连接

mysql系统级操作和基本语法规定

登录/退出基本操作

mysql -hlocalhost -uroot -p

mysql -h12.34.56.78 -utest1 -p

备份恢复数据库

备份：mysqldump -h服务器地址 -u用户名 -p 数据库名1 > 要存储为的文件名file1

恢复：mysql -h服务器地址 -u用户名 -p 数据库名2 < 要存储为的文件名file1

基本语法规定：

注释：

#注释

-- 注释

/\*注释\*/

语句结束符: 默认是“;”，可以使用delimiter 修改为其他

php中操作数据库的基本代码和流程

1，链接数据库：mysql\_connect(“数据库服务器地址”, “用户名”, “密码”);

2，设定链接编码：mysql\_set\_charset(“编码名”); //对应语句：set names 编码名；

3，选定数据库：mysql\_select\_db(数据库名); //对应语句： use 数据库名

4，执行sql语句：$result = mysql\_query(“sql语句”);

如果是非返回数据语句：返回值只有true和false

如果是返回数据语句，失败同样返回false，但成功就会返回数据集（结果集）

此时就需要进行“遍历取出”：

ｗｈｉｌｅ（$record = mysql\_fetch\_array( 结果集 ){

//$record是一个数组，里面装了一行数据，数字的下标就是行的字段名。

}

数据定义语句

数据库定义

语法形式

删除数据库：

其他数据库相关语句

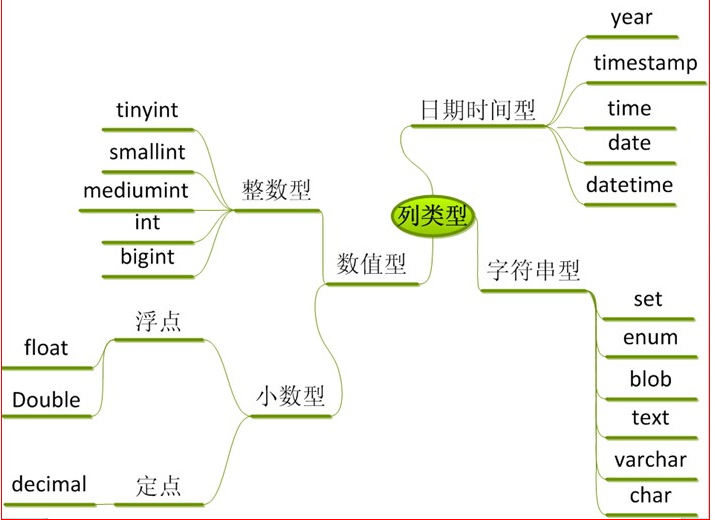
问题：在cmd使用set names utf8，然后得到乱码了？

1，cmd，必须使用gbk

2，php文件中，可以根据文件的编码来定：utf8编码就是用utf8，ANSI编码（gbk）就用gbk

数据类型

概览



整数类型

分为：

tinyint(1字节), smallint(2字节), mediumint(3字节),int(4字节), bigint(8字节)

1字节=8位（8个灯泡）

一个灯泡只能表达2个意思（2个数字）

2个灯泡可以表达4个意思

3个灯泡可以表达8个意思

。。。。。

8个灯泡（1个字节）可以表达256个数字。

默认情况下，这些整数类型都是可正可负的，那么：

tinyInt就只能存储: -128--127这些数

范围



通用设定形式

定义一个字段的时候的类型的写法。

比如：

create table tab1 (f1 数据类型 );

数据类型： 类型名[（长度n）] [unsigned] [zerofill]

长度n： 表示的意思是该数字的“显示形式上的长度”，

unsigned：设定为“无符号”数，则此时不能存储负数，正数几乎加倍。

zerofill：填充0，是指如果一个数字的长度不够指定长度的时候，可以在左边填充0以补到该长度。

注意： 如果设置了zerofill，则自动也就表示同时具备了unsigned修饰的含义

小数类型

可以分为：

单精度浮点型： float，非精确数，通常不设定长度

双精度浮点：double，非精确数，通常不设定长度

定点型：decimal，精确数，通常，定点型需要设定长度，形式为：decimal(总长, 小数位数）

时间日期类型

有如下：

date, time, datetime, year, timestamp

注意：

写入数据库时，直接的时间日期数据，应该用单引号引起了。

year类型可以是4位整数或4位纯数字字符串，也可以是2位整数或2位纯数字字符串

timestamp表示的含义是“时间戳”，其实就是指“当前时刻”，本质上是一个数字，代表从1970年1月1日0点0分0秒到某个时间之间的秒数数值。该类型的字段值无需赋值，而是会自动取得当前时间值。

字符串类型

最基本最重要的2个：

varchar类型：可变长度字符串类型。最多能存储65532个字节的字符串——也还要考虑字符编码。设定的长度只是最长长度，但可以不存满，则实际长度以数据长度为准。

char类型：定长字符串类型。最多能存储256个字符。如果存储的数据不足设定的长度，则会自动补空格填满。

设定时都需要给定长度，比如：varchar(20), char(6);

mysql，一行的所有内容的总的存储长度也有个限制，约65535个。

2个二进制文本：

binary： 类似char，只是里面不存“文本”，而是存“文本的二进制数据”

varbinary: ，类似varchar，同样，不存“文本”，而是存“文本的二进制数据”

2个大文本类型：

text： 可以存储“超大文本”，且其实际的长度并不占用一行的长度。相对char和varchar，效率低。

blob： 可以存储“超大二进制文本”，通常用于存储图片这种“二进制数据”

2个有关“选项”的文本存储形式：

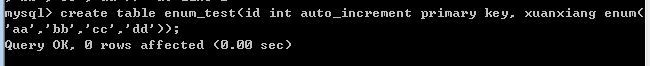
enum：

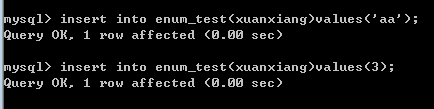
专门用于方便存储类似表单中的“单选项”的值。

形式：

enum(‘选项1’，‘选项2’，‘选项3’，......）

这些选项的值虽然是字符串，但其数据库内部存储其实是数字（效率高），他们的数字值是：1， 2， 3，4， 5，。。。。。最多6万多个。





结果：

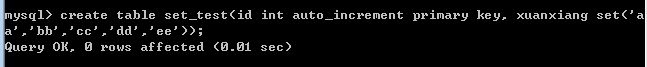


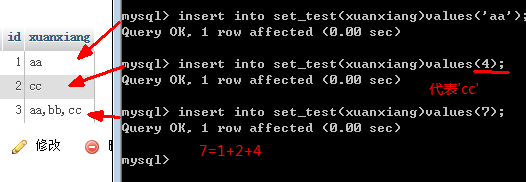
set：专门用于方便存储类似表单中的“多选项”的值。

形式：

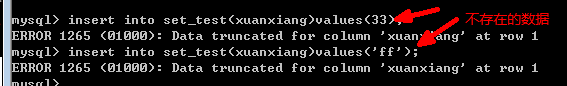
set (‘选项1’，‘选项2’，‘选项3’，......）

这些选项的值虽然是字符串，但其数据库内部存储其实是数字（效率高），他们对应的数字值是：1， 2，4， 8， 16，。。。。。最多6万多个





在看不存在的数据的情形：



可见，enum，set类型的字段，限制的“字符串”数据值。

表定义语句

创建表：

基本形式

create table [if not exists] 表名(字段列表， [约束或索引列表]) [表选项列表];

说明：列表都是表示“多个”，相互之间用逗号分开。

字段基本形式： **字段名 类型 [字段修饰属性]**；

字段属性设置

not null： 不为空，表示该字段不能放“null”这个值。不写，则默认是可以为空

auto\_increment: 设定int类型字段的值可以“自增长”，即其值无需“写入”，而会自动获得并增加

此属性必须随同 primary key 或 unique key 一起使用。

[primary] key： 设定为主键。是唯一键“加强”：也不能重复并且不能使用null，并且可以作为确定任意一行数据的“关键值”，最常见的类似：where id= 8; 或 where user\_name = ‘zhangsan’;

通常，每个表都应该有个主键，而且大多数表，喜欢使用一个id并自增长类型作为主键。

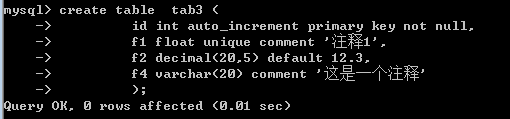
但：一个表只能设定一个主键。

unique [key] : 设定为唯一键：表示该字段的所有行的值不可以重复（唯一性）。

default ‘默认值’： 设定一个字段在没有插入数据的时候自动使用的值。

comment ‘字段注释’：

举例：



索引设置

什么是索引：

索引是一个“内置表”，该表的数据是对某个真实表的某个（些）字段的数据做了“排序”之后的存储形式。

其作用是：极大提高表查找数据的速度！——其效率（速度）可以匹敌二分查找。

注意：索引在提供查找速度的同时，降低增删改的速度。

对创建（设计）表来说，建立索引是非常简单的事，形式如下：

索引类型 (字段名1，字段名2， .... ） //可以使用多个字段建立索引，但通常是一个

有以下几种索引：

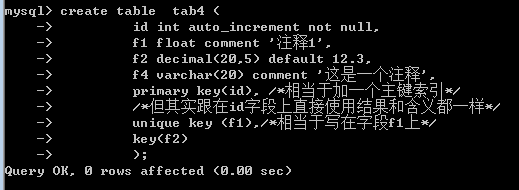
普通索引：key(字段名1，字段名2， .... ）：它只具有索引的基本功能——提速

唯一索引：unique key (字段名1，字段名2， .... ）

主键索引：primary key (字段名1，字段名2， .... ）

全文索引：fulltext (字段名1，字段名2， .... ）

举例：



约束设置

什么叫约束：约束就是一种限定数据以符合某种要求的形式（机制）

约束主要有：

**主键约束**：primary key (字段名1，字段名2， .... ）

其实就是主键索引，也是主键属性。即primary key有3个角度的理解（说法）：字段属性设置为主键，或建立的主键索引，或设定一个主键约束，但他们的本质是一样

**唯一约束**：unique key (字段名1，字段名2， .... ），其实也是“3体合一”（类似primary key)

**外键约束**：

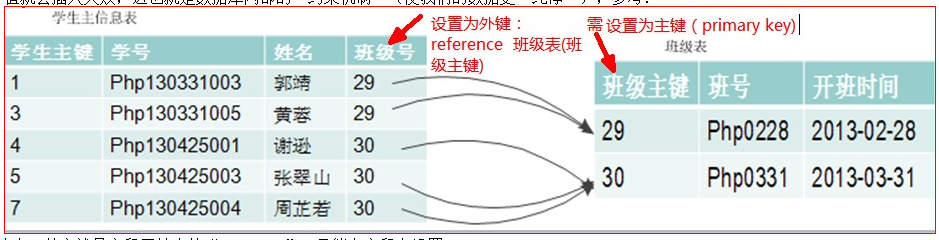
什么叫外键：就是设定一个表中的某个字段的值，必须“来源于”另一个表的某个主键字段的值。

语法形式：

foreign key (字段名1，字段名2， .... ) references 表名2(字段名1，字段名2， .... )

说明：

对某个（些）字段设定外键，则其相对应的其他表的对应字段需要设置为主键。



**非空约束**：就是要求该字段的值不能为空，其只能在字段上当作字段属性来设定。

**默认约束**：就是要求该字段的值在“空”的时候会自动填充该设定的默认值，也只能字段上设定。

**检查约束**：就是使用一个表达式（逻辑判断）来决定数据是否有效，比如年龄字段，可以使用

tinyint，则可能会超过127就不合适了。

tinynit unsigned，则0-255是可以的。

但：如果考虑现实情况，假设某保险公司只作150岁以下的人的保险。则我们就可以继续对该字段可能存储的数据进行“约束”。比如类似：if（age > 150){return false}

可惜的是：ｍｙｓｑｌ不支持检查约束的语法和功能。

表选项

表选项就是对一个表的有关属性的设定，通常都不需要。如果不设定，则有其默认值。

有以下几个可用：

comment = ‘表的注释’;

charset = 字符编码名称； //跟数据的字符编码设定一个意思。

字符编码设定的范围及继承关系：

系统级设定：安装时确定了。

库级设定：建库时设定；

表级设定：就是这里的charset = 字符编码名称

字段级设定：作为字段属性出现。

他们都只对“字符类型”的字段有效。每一级如果没有设定，就会“继承使用”其上一级的设定。

auto\_increment = 起始整数； //自增长类型值的初值，默认是1

engine = “表的存储引擎名”； //

存储引擎就是将数据存入硬盘（或其他媒介）的方式方法。通常就几个可用，默认是InnoDB

存储引擎决定一个数据表的各方面的信息：功能和性能。



总的示例：



修改表：

一般概述

通常，创建一个表，能搞定（做到）的事情，修改表也能做到。大体来说，就可以做到：

增删改字段：

增：alter table 表名 add [column] 字段名 字段类型 字段属性；

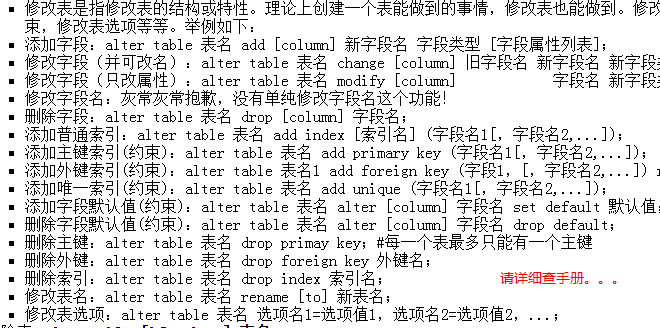
删： alter table 表名 drop 字段名

改：alter table 表名 change 原字段名 新字段名 新字段类型 新字段属性；

增删索引：

增删约束：

修改表选项：



修改表的基本形式：

alter table 表名 修改语句1，修改语句2， ..... ；

删除表：

drop table [if exists] 表名；

表的其他操作：

显示所有表show tables：

显示表结构desc 表名；

显示表的创建语句: show create table 表名；。

从已有表复制表结构：create table [if not exists] 新表名 like 原表名;

从已有表复制表结构：create table [if not exists] 新表名 select \* from 原表名 where 1<>1;（不推荐）

视图

单词：view

什么是视图：

视图可以看作是一个“临时存储的数据所构成的表”（非真实表），其实本质上只是一个select语句。只是将该select语句（通常比较复杂）进行一个“包装”，并设定了一个名字，其后就可以通过该名字并把该名字当作一个表来使用。

如果一个ｓｅｌｅｃｔ语句比较复杂，又在多个页面需要使用它，则可以将它做成一个视图，方便使用。

又如果，某个数据表中的某些字段不想给别人看（不同公司之间的数据业务交换的时候），但另一个又需要给人看，此时也可以使用视图。

视图创建形式：

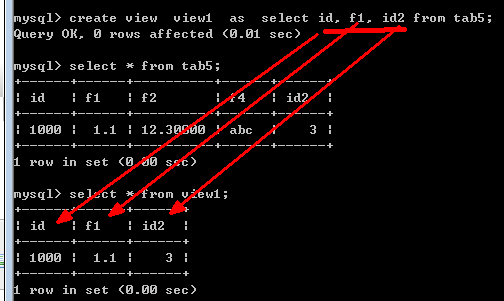
create view 视图名 [（列名1，列名2，...）] as 一条复杂select语句；

可以将select语句所取得的列重新命名，但也可以不重新命名，则使用select语句中的给定列名。

视图的使用

其实就是当作一个查询表来用（通常只用于select）

select \* from 视图名 where 条件 order by .....。



修改视图：

alter view 视图名 [（列名1，列名2，...）] as select语句；

删除视图：

drop view [if exists] 视图名；

数据库（数据表）的设计思想介绍

设计数据库，其实就是设计表（多个表）

数据库设计3范式（3NF）：

范式，就是规范，就是指设计数据库需要（应该）遵循的原则。

每个范式，都是用来规定某种结构或数据要求——后一范式都是在前一范式已经满足的情况用来“加强要求”

第一范式（１ＮＦ），原子性

原子性：

存储的数据应该具有“不可再分性”。

不良做法示例：



可见，其违反了原则性范式：学生字段的数据存储了多个可分的数据。

修改后为：



第二范式（２ＮＦ）唯一性

**需要实现每一行数据具有唯一可区分的特性，并不能有部分依赖关系**。

通常，给一个表加主键（也是推荐做法），就可以做到“唯一可区分”。

但主键有这样情况：

设定一个字段为主键：此时，表示该一个字段的值就可以明确确定一行数据。

设定多个字段为主键：表示只有这多个字段的值都确定后才能确定一行数据。此时也称为“联合主键”

**什么叫依赖：**

如果确定一个表中的某个数据（A），则就可以确定该表中的其他另一个数据（B），则我们说：B依赖于A。

实际上，一个表只要有主键，则其他非主键一定是依赖于主键的。

**什么叫“部分依赖”：**

如果确定一个表中的某个数据组合（A，B），则就可以确定该表中的其他另一个数据（C），则我们说：C依赖于（A，B）（此时A，B通常就是做出主键）。

但：如果某个数据D，它只依赖于数据A，或者说，A一确定，则D也可以确定，此时我们就称为“数据D部分依赖于数据A——可见部分依赖是指某个非主键字段，依赖于联合主键字段的其中部分字段。

不良做法：



改良之后：



第三范式（3NF）：独立性，消除传递依赖

在一个具有主键的表中，假设主键为A，其必然其他非主键都依赖于该主键，比如：B依赖于A，C依赖于A，D依赖于A。。。。。。

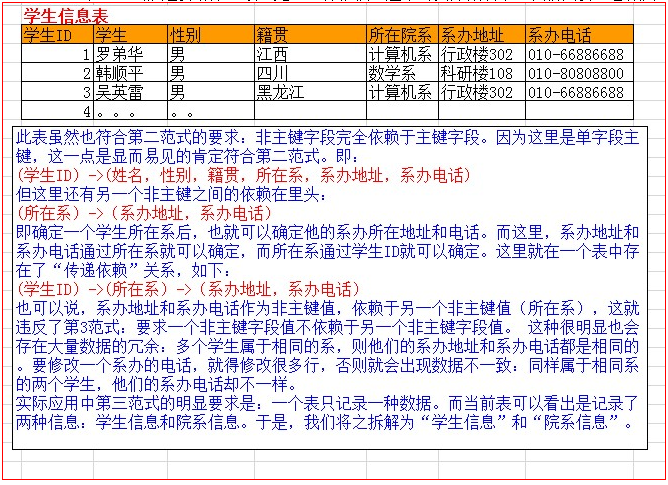
但同时：如果该表中的某个字段B的值一确定，就能够确定另一个字段的值C，则我们称为C依赖于B。

那么，就出现了：

C依赖B，B依赖A——这就是传递依赖。

则消除该传递依赖的的通常做法，就是将C依赖于B的数据，分离到另一个表中。

不良例子：



改良之后：



最后的总结

通常，在实践中，满足3范式只要做到“**一个表只存一种数据”**基本就可以实现。

另外，范式不是绝对要求，有时候我们为了数据的使用方便，还会（需要）故意违反范式。