1. 数值类型字段定义
2. <= 255 统一采用 tinyint(4) 类型定义
3. 255 统一采用 bigint(20) 类型定义
4. 字符串类型定义
5. 字符串类型统一采用 varchar(xx) 类型定义( xx 表示字符串最大长度 )
6. 每个表必须包含 status, created\_date, last\_modified\_date 三个字段 (方便底层作统一处理)  
   eg:

CREATE TABLE `tb\_hist\_player\_level` (  
 `id` bigint(20) unsigned NOT NULL AUTO\_INCREMENT,  
 `player\_id` bigint(20) unsigned NOT NULL COMMENT '玩家ID',  
 `oper\_type` bigint(20) unsigned NOT NULL COMMENT '操作类型',  
 `old\_level` bigint(20) unsigned NOT NULL COMMENT '操作前等级',  
 `level` bigint(20) unsigned NOT NULL COMMENT '操作后等级',  
 `time` datetime NOT NULL COMMENT '玩家升级的时间',  
 `status` tinyint(4) unsigned NOT NULL,  
 `created\_date` datetime NOT NULL,  
 `last\_modified\_date` datetime NOT NULL,  
 PRIMARY KEY (`id`),  
 KEY `Index\_1` (`player\_id`) USING BTREE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci COMMENT='玩家升级历史表';

1. 底层在插入记录时, created\_date和last\_modified\_date会自动赋值为当前时间;
2. 底层在更新记录时, last\_modified\_date会自动赋值为当前时间;
3. 涉及到 服务器读、写的数据库表，修改过结构的，需要通知重新生成 program.exe 才会生效
4. 数据库表名定义规范

tb\_ad\_\*, tb\_rp\_\* php相关表  
 tb\_hist\_\* 历史表, 不加载到内存  
 tb\_player\* 玩家公有数据表, 全部加载到内存  
 tb\_player\_pd\_\* 玩家私有数据表, 只加载活跃玩家的记录  
 tb\_sys\_\* 系统数据表(例如帮派、活动...) 全部加载到内存

1. 玩家私有数据表
2. 数据库表名以 tb\_player\_ 开头 并且 \_pd 结尾的表
3. 此一类表在进程启动时, 不会将所有数据加载至内存(即dbmgr只会加载一部份数据 – 根据 tb\_player 的 last\_modified\_date 获取最后 N 个活跃的玩家)
4. (2)中提到的 N 在 machine.conf 中的 [app settings] -- preload 配置
5. 此一类表的数据通常只有玩家在线时才会被加载至内存 (在 dbmgr 通过 dbmgr\_global.load\_player\_private\_data())
6. dbmgr\_global.load\_player\_private\_data() 会先判断玩家是否已加载过所有tb\_player\_xxx\_pd表, 如果已经加载过则直接返回; 否则会将表数据从 MySql 加载至 内存
7. 单服每增加一张表都必须向 tb\_sys\_tableinfo 表新增相应的记录

tb\_sys\_tableinfo 表定义  
 CREATE TABLE `tb\_sys\_tableinfo` (  
 `table\_name` varchar(100) COLLATE utf8\_unicode\_ci NOT NULL DEFAULT '0' COMMENT '数据库表名',  
 `combine\_type` tinyint(4) unsigned NOT NULL DEFAULT '0' COMMENT '合服方式,1表示当A,B两个服合服时,保留A服数据;2表示清空表;3表示两个服数据直接合并;',  
 `delete\_player\_type` tinyint(4) unsigned NOT NULL DEFAULT '0' COMMENT '删除玩家方式,1表示不操作;2表示根据delete\_player\_condition进行操作;',  
 `delete\_player\_condition` varchar(200) COLLATE utf8\_unicode\_ci NOT NULL DEFAULT '0' COMMENT '删除玩家条件,例如 player\_id\_1 = ? or player\_id\_2 = ?, player\_id\_1 和 player\_id\_2 是字段名',  
 `delete\_player\_sequence` tinyint(4) unsigned NOT NULL DEFAULT '0' COMMENT '删除玩家时的操作顺序,值越大,越后处理',  
 `status` tinyint(4) unsigned NOT NULL DEFAULT '1' COMMENT '状态码',  
 `created\_date` datetime NOT NULL DEFAULT '1990-01-01 00:00:00' COMMENT '创建时间',  
 `last\_modified\_date` datetime NOT NULL DEFAULT '1990-01-01 00:00:00' COMMENT '最后修改时间',  
 PRIMARY KEY (`table\_name`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci COMMENT='数据库表合服删除玩家逻辑信息';

1. 数据库表记录的增、删、改、查操作
2. 以 tb\_example 表名为例, 下面代码演示增加记录操作;
3. 在对数据库所有字段进行赋值之前，必须先调用 tb\_example:reset()
4. 在对数据库所有字段进行赋值之后，必须调用 tb\_example:insert() 插入记录

local tb\_example = daodb.tb\_example()  
tb\_example:reset()  
tb\_example.test = 1  
tb\_example:insert()

1. 以 tb\_example 表名为例, 下面代码演示查找并更新记录操作

local tb\_example = daodb.tb\_example()  
tb\_example:reset()  
tb\_example:prepare("test=? and status=?")  
tb\_example:setNumber(1, 1)  
tb\_example:setNumber(2, DB\_RECORD\_STATUS.DB\_RECORD\_STATUS\_VALID)  
tb\_example:find()  
while tb\_example:next() do  
 tb\_example.test = 2  
 tb\_example:update()  
end

1. 将tb\_example表中, 所有 test 为 1 的记录的 test 字段的值, 修改为 2
2. 调用完tb\_example:find()后必须至少调用一次tb\_example:next()后，才可以读取字段值
3. 对表字段进行赋值后，必须调用 update() 方法才会生效
4. 以 tb\_example 表名为例, 下面代码演示查找并删除记录操作

local tb\_example = daodb.tb\_example()  
tb\_example:reset()  
tb\_example:prepare("test=? and status=?")  
tb\_example:setNumber(1, 1)  
tb\_example:setNumber(2, DB\_RECORD\_STATUS.DB\_RECORD\_STATUS\_VALID)  
tb\_example:find()  
while tb\_example:next() do  
 tb\_example:remove()  
end

1. 在对数据库表进行操作后, 协议返回之前，必须调用以下代码才会生效

transaction:commit()

1. 判断find()结构的记录数

local t = daodb.tb\_example()  
t:reset()  
t:prepare("test=? and status=?")  
t:setNumber(1, 1)  
t:setNumber(2, DB\_RECORD\_STATUS.DB\_RECORD\_STATUS\_VALID)  
t:find()  
  
--   
if t:count() == 0 then  
 --- 没有记录时的处理  
end  
  
--   
if t:count() ~= 0 then  
 --- 当存在记录时的处理  
end  
  
while tb\_example:next() do  
 -- 对记录进行操作  
end