MySQL 数据库设计开发规范

创建人	刘龙威
部门	哔哩哔哩游戏事业部-SRE
创建时间	2018 年 12 月 20 日
版本号	V1.1
更新时间	

更新时间	更新内容	备注

目录

一、基础规范

1.1 使用 Innodb 存储引擎

所有表必须使用 InnoDB 存储引擎。

1.2 字符集统一使用 utf8 或 utf8mb4，首选 utf8

utf8mb4 是 utf8 的超集，utf8 比 utf8mb4 少一个字节,能够节约空间。同时 utf8mb4 会加大相应的索引长度，降低索引效率。当有 Emoji 和不常用的汉字等特殊需求时，再选用 utf8mb4。

不用指定校对规则，默认 utf8_general_ci 即可。

1.3 所有表都要添加注释

所有表需要添加注释，自研项目表中所有字段也需要添加注释，独代及本地化项目视情况而定。

1.4 控制单表字段数量

单表字段数上限 50 左右，再多的话考虑垂直分表，一是冷热数据分离，二是大字段分离，三是常在一起做条件和返回列的不分离。

表字段控制少而精，可以提高 IO 效率，内存缓存更多有效数据，从而提高响应速度和并发能力，后续 ALTER TABLE 也更快。

1.5 不使用外键

外键严重影响数据库性能，增加表维护复杂度，所以禁用外键。

1.6 所有表都必须要显式指定主键

InnoDB 表实际是一棵索引组织表，顺序存储可以提高存取效率，充分利用磁盘空间。

1.7 非日志类型业务表单表数据量控制在 1000 万以内

二、对象命名规范

2.1 采用有意义的命名

命名只能使用英文字母、数字和下划线三类字符组合。尽量使用英文单词，不要使用拼音，多个单词组成时，中间以下划线“_”分割。

2.2 英文单词统一使用单数的形式

无论表名和字段名，统一使用英文的单数形式。

2.3 表名字段名统一使用小写

虽然 MySQL 在服务器端可以设置不区分大小写，但是创建表的时候，表名和字段名统一使用小写。

2.4 表名字段名不适合过长，尽量不要超过 30 个字符

2.5 特殊用途的表名需要特殊标识，如临时表以 tmp_开头，备份表以 bak_开头

2.6 判断、标示性字段

凡表示"是否"一类的判断字段，都使用 is_开头，如 is_delete。

2.7 索引的命名

普通索引以 idx_各个列名简称，唯一索引以 uk_各个列名简称命名，中间用_隔开。如 idx_col1_col2_col3(col1,col2,col3)，如果列过长，用简写。

2.8 存储过程、函数、视图命名

- 索引统一以 idx_开头，唯一索引可用 uk_开头，如：idx_tablename_colname,idx_tbname_c1_c2(联合索引)。
- 存储过程以 sp_开头，后接功能描述。
- 函数命名以 fn_开头，后接功能描述。
- 视图命名以 vw_开头，vw_table_name+内容标识，物化视图以 mv_开头。
- 触发器命名以 tr_+触发标识+相应的表名。

禁止：

- 禁止使用 MySQL 系统保留字。
- 禁止对象名使用非字母开头。
- 禁止使用除字母、数字、下划线外的其他字符。

备注：

以上数据库对象命名规范主要适用于自研项目，独代、本地化项目需要视实际情况而定。

三、字段设计规范

3.1 尽量使用最小的数据类型

更小的数据类型会占用更少的磁盘、内存、和 CPU 缓存，处理时使用的 CPU 周期也更少，例如，存储‘bilibili’时，使用 varchar(8)和 varchar(100)，占用磁盘空间是一样的，但是 varchar(100)效率更低，使用时会耗用更大的内存，这对于排序和临时表操作影响比较大。

3.2 优先选择整型，而不是字符类型

整型比字符的代价更低，字符集在做比较时的校对规则比整型复杂，比如整型存储 IP 使用 int unsingned，手机号使用 bigint，都不用使用 varchar。IPv6 使用 varchar(39)。

整型中注意 tinyint 选项，不要创建 tinyint(1)，tinyint(1)在 java 中会隐士转换成 boolean 类型。

3.3 整型类型，默认一般都加上 unsigned 属性

一般不用负数，这样可以使正数的上限提升一倍。

3.4 主键列尽量使用有序整型

innodb 的主键就是聚簇索引，最通俗的解释是:聚簇索引的顺序就是数据的物理存储顺序。所以有序的主键在插入和查找都能获得更好的性能，也有效地避免页分裂问题。不建议使用 varchar 类型的主键。

3.5 在对小数进行精准计算时使用 decimal，如金额等账务数据

3.6 字段若无特殊情况，需设置为非空，并设置默认值，整型默认 0，字符串默认''（空）

设置非空的目的是避免 NULL 值的出现。NULL 在 MySQL 中，类似黑洞，是未知的，且占用空间。NULL 使得索引、索引统计和值都更复杂，并且影响优化器的判断。并且在 count()某列的时候，NULL 是不会被统计到的。

3.7 对于 text/blob 类型，当范围允许的时候，能用 varchar 还是用 varchar。

MySQL 会把这两种类型的值当做一个独立的对象处理，存储引擎在存储时通常会做特殊处理，某些会带来性能影响。

四、索引设计规范

4.1 索引创建原则

选择索引列时尽量选择索引选择性高的列，索引选择性高即数据重复度低。索引选择性计算方法：select count(distinct(column_name))/count(*) from table_name，越接近 1 说明 col1 上使用索引的过滤效果越好。另外如果某列为

sex，数据倾斜比较大，male 占比 99%，female 占比 1%，而每次查询都为 female，则需要考虑创建该列的索引。

4.2 最左前缀原则

mysql 使用联合索引时，从左向右匹配，遇到断开或者范围查询时，无法用到后续的索引列比如索引 idx_c1c2c3 (c1,c2,c3)，相当于创建了(c1)、(c1,c2)、(c1,c2,c3)三个索引，where 条件包含上面三种情况的字段比较则可以用到索引，但像 where c1=a and c3=c 只能用到 c1 列的索引，像 c2=b and c3=c 等情况就完全用不到这个索引。

遇到范围查询(>、<、between、like)也会停止索引匹配，比如 c1=a and c2 > 2 and c3=c，只有 c1,c2 列上的比较能用到索引，(c1,c2,c3)排列的索引才可能会都用上。

where 条件里面字段的顺序与索引顺序无关，mysql 优化器会自动调整顺序。

联合索引中，尽量将选择性高的列放在索引的左侧，比如 A、B 两个列，B 的选择性比 A 高，这样建议索引创建时 B 放到 A 的左边：idx_B_A(B,A)。

4.3 前缀索引

当列长度比较长时，可以考虑添加前缀索引。

MySQL 的前缀索引能有效减小索引文件的大小，提高索引的速度。

前缀索引也有它的坏处：不能在 ORDER BY 或 GROUP BY 中使用前缀索引，也不能把它们用作覆盖索引(Covering Index)。

4.4 覆盖索引

INNODB 存储引擎中，主键是和数据放在一起的聚集索引，普通索引最终指向的是主键地址，如果用户需要查询普通索引中所不包含的数据列，则需要先通过普通索引查找到主键值，然后再通过主键查询到其他数据列，因此需要查询两次。覆盖索引则可以在一个索引中获取所有需要的数据列，从而避免回表进行二次查找，节省 IO 因此效率较高。例如 SELECT a,b FROM table WHERE a=xx，如果 uid 不是主键，适当时候可以将索引添加为 index(a,b)，以获得性能提升。

4.5 避免在频繁更新的列上创建索引

更新索引列会增加索引维护成本，增加 IO 负担。

4.6 避免冗余索引

InnoDB 表是一棵索引组织表，主键是和数据放在一起的聚集索引，普通索引最终指向的是主键地址，所以把主键做最后一列是多余的。

如 id 作为主键，联合索引(user_id,id)上的 id 就完全多余。

(a,b,c)、(a,b)，后者为冗余索引。

(a,b)、(b,a)，其中(b,a)索引中的 a 列也是冗余的，直接创建 b 的单列索引即可。

可以利用前缀索引来达到加速目的，减轻维护负担。

4.7 索引个数限制

索引数量不用过多，结合我们游戏的业务类型来看，一张表最多 5 个索引。

虽然大部分情况下，索引会提高查询效率，但是维护索引成本较高，写入的时候会增加 IO 负担。另外，索引也会增加磁盘空间的开销。

4.8 可考虑创建索引的情况

- where 后面使用频繁的搜索列，但不是所有 where 后面的列都要创建索引。
- 索引即排序，可考虑在 group by、order by 后面字段加上索引，如 select c1,c2,c3,... from tablename where c1=xxx order by cn;可考虑创建(c1,cn)的复合索引。
- join 关联查询时，所有匹配 on 和 where 的字段应建立合适的索引。

4.9 存在索引但不会用到索引的情况

- 以%开头的 like 查询不能够使用 B-Tree 索引。
- 在发生隐式转换的时候也不会使用索引，特别是当列类型是字符串，请记得在 where 条件中把字符常量值用引号括起来。
- 符合索引中，不满足最左前缀原则的也不会使用复合索引，最多只能使用复合索引的部分列。
- 如果 MySQL 估计使用索引比全表扫更慢，则不使用索引，特别是表中数据量很小的时候。
- 用 OR 分开的条件，如果 OR 前的条件中的列有索引，而后面的列没有索引，那么涉及的索引都不会被用到。

- WHERE 条件列上使用了函数

五、SQL 开发及设计

5.1SQL 语句尽可能简单

尽量减少接口对后台 MySQL 的请求数。

少用 select *，最好直接列出所需的列，返回多余的数据会增加 IO 开销。

少用函数运算，如 SUM()、COUNT()等，如有需要放到应用程序或缓存中计算。

少用 join 查询，特别是 join 大表，可适当设置字段冗余，避免和减少 join 查询。

避免或谨慎使用 MySQL 的特殊功能，如：INSERT... ON DUPLICATE KEY UPDATE、LAST_INSERT_ID()等，INSERT... ON DUPLICATE KEY UPDATE 在 MySQL5.7 版本后可能会引发死锁问题，LAST_INSERT_ID()在 update 的时候性能很差。

5.2 保持代码洁癖，不要出现隐式类型转换。

隐式转换除了不会使用已创建的索引，还有可能会引发其他意想不到的问题。

5.3 避免大事务

一个事务中避免出现大事务，过多的 SELECT 和 UPDATE 等，因为只有最后 COMMIT\ROLLBACK 完成后，锁资源才会释放，可能会影响并发性能。

5.4 避免全表扫类型的 SQL

特别是大表的全变扫会对数据库带来毁灭性影响，建立合适索引，避免不走索引的查询，杜绝出现不加 WHERE 条件的全表查询。

5.5 常用到的优化技巧

- GROUP BY 不仅是分组，而且要排序。排序会消耗 CPU 资源，如果业务不需要排序可以在 GROUP BY c1 后面加上 ORDER BY NULL（只分组，不排序）。
- 能用 IN 的不要用 OR,能用 EXIST 的不要用 IN。
- 分页查询 MySQL 处理较慢，特别是数据量比较大的时候，可以用索引延迟策略，先查分页后的索引值，再 INNER JOIN 查出分页的记录。
- 尽量合并插入，如：INSERT INTO VALUES(),(),()。

六、通用平台表创建规则及建表示例

6.1 字段映射

6.1.1 登陆日志:

分表要求：日表

格式示例:

映射的字段	类型	注释
id	int(10)	自增主键 id

映射的字段	类型	注释
uid	bigint(20)	玩家 id
platform_id	tinyint(2)	平台 id 1.IOS 2.Bili 3.CPS 4.UO 5.WEB
channel_id	int(10)	渠道 id
type	tinyint(1)	类型 1.注册和 2.登录
created_at	int(10)	时间戳
server_id	int(10)	区服 id
ip	varchar(50)	ip 地址
app_version	varchar(20)	版本号
deviceinfo	varchar(500)	设备 info: 包含设备 ID、系统版本 / 固件类型(注意留有空格)、客户端来源、客户端版本 示例:(446ce1fa8577de4a5c05fc474157ad5d;Android OS 6.0.1 / API-23 (MXB48T/1);smartisan SM919;com.bilibili.fatego1)

6.1.2 用户基础信息:

分表要求: 10 库 10 表

分库分表示例 : 根据用户 ID 的后 1 位分库, 后 2 位分表:

`db0:` xxx_0 , `table:` xxx_00 ~ xxx_90

`db1:` xxx_1 , `table:` xxx_01 ~ xxx_91

`... ...`

`db9:` xxx_9 , `table:` xxx_09 ~ xxx_99

映射的字段	类型	注释
id	int(10)	自增主键 id

映射的字段	类型	注释
uid	bigint(20)	玩家 id
platform_id	tinyint(2)	平台 id 1.IOS 2.Bili 3.CPS 4.UO 5.WEB
channel_id	int(10)	渠道 id
type	tinyint(1)	类型 1.注册和 2.登录
created_at	int(10)	时间戳
server_id	int(10)	区服 id
ip	varchar(50)	ip 地址
app_version	varchar(20)	版本号
deviceinfo	varchar(500)	设备 info: 包含设备 ID、系统版本 / 固件类型(注意留有空格)、客户端来源、客户端版本 示例:(446ce1fa8577de4a5c05fc474157ad5d;Android OS 6.0.1 / API-23 (MXB48T/1);smartisan SM919;com.bilibili.fatego1)

6.1.3 全局用户信息:

分表要求：单表 firstpaytime 和 regtime 要 有符号设置.

映射的字段	类型	注释
uid	bigint(20)	玩家 id
platform_id	tinyint(2)	平台 id 1.IOS 2.Bili 3.CPS 4.UO 5.WEB
channel_id	int(10)	渠道 id
account_name	varchar(50)	通行证名称 appname
account_uid	varchar(32)	通行证 uuid
sex	tinyint(1)	性别 性别 1：男 2：女 0:其他
level	int(10)	等级

映射的字段	类型	注释
uname	varchar(50)	玩家名字
firstpaytime	int(10)	首冲时间
regtime	int(10)	注册时间
lasttime	int(10)	最后登陆时间
pay_total	int(10)	充值总值(单位分)
server_id	int(10)	区服 id
reg_device_id	varchar(50)	注册设备 id
last_device_id	varchar(50)	最后一次设备 id

6.1.4 支付信息:

映射的字段	类型	注释
id	int(10)	主键 ID
platform_id	tinyint(2)	平台 id 1.IOS 2.Bili 3.CPS 4.UO 5.WEB
channel_id	int(10)	渠道 ID
uid	bigint(20)	玩家 id
out_trade_no	varchar(80)	订单号
total_fee	bigint(20)	支付金额(单位分)
started_at	int(10)	下单时间(时间戳)
ended_at	int(10)	结单时间(时间戳)
status	tinyint(1)	订单状态(0 未支付,1.已支付,已完成,2 手工修复,已完成)
server_id	int(10)	区服 id

6.2 SQL 文件发布功能

6.2.1 介绍:

将 SQL 文件推到服务器并执行.

6.2.2 规则:

上传文件包的格式必须是.zip。

文件包内文件命名规范示例 bili_ios_X_dbname.sql 即是 平台 1_平台 2_X_dbname_20180608.sql ， "_X_"是分隔符, dbname 是数据库名，文件命名必须严格按照规范。

压缩包内可以存在多个 2 中的文件. 一个 SQL 文件对应一个 db(可多个平台,但必须是同一个 dbname)。

单个文件中的 SQL 语句，开启事务,开启超时设置。

6.3 相关 ID 生成规范

6.3.1 platformId 和 channelId 即 平台 ID 和渠道 ID:

平台	平台 ID	渠道 (channel_id)	注释
uo	4	1-60	具体数值从 sdk 中获取
cps	3	10	
bili	2	1	
ios	1	1000	
小萌 ios	91	3003	

小萌 googleplay	92	3001	
小萌 mycard	93	3002	大法师

6.3.2 serverId 即 区服 id:

ios 渠道默认值 10000

bili 渠道默认值 20000

cps 渠道默认值 30000

uo 渠道默认值 40000

web 渠道默认值 50000

小萌 ios 渠道默认值 910001

小萌 googleplay 渠道默认值 920001

小萌 mycard 渠道默认值 930001

小萌 web 渠道默认值 950001

分小服规则 渠道默认值 + 服 id 例如 ios 一服：10000+1 = 10001，ios 二服：10000+2=10002。 Bili 一服：20000+1=20001，Bili 二服：20000+2=20002。

如不分服，serverId = 默认值+1

6.3.3 playerId 即 member 表 id 自增初始值:

ios 渠道：10 亿（1000000000）

bili 渠道：20 亿（2000000000）

cps 渠道：30 亿（3000000000）

uo 渠道：40 亿（4000000000）

web 渠道：50 亿（5000000000）

备注 1、如涉及到混服，id 自增仍然以各自渠道标示为准。 2、如涉及特殊情况，视具体情况制定。

6.3.4 orderId 即游戏订单号唯一生成规则：

第一段：线上(pr)、提审(ts)、cb(cb)、QA(qa)、开发(de)、测试(te)、版署(bs)、云测(yc) 第二段：i(ios)、b(bili)、c(cps)、u(uo)、w(web) i(ios)、g(googleplay)、m(mycard) 规则：用途(2)-渠道(1)-游戏内 id(10)-时间戳(11)-随机数(4)，一共 32 位 例如：de-b-1234567890-12345678900-1234 各个位置补齐各要求位数，共 32 位。

6.3.5 sdk:

国服 SDK 类型： Android B 服：1 iOS B 服：2 CPS 享游：3 UO 联运：4 繁体 FGO SDK 类型： 繁体 FGO_Google 支付版：1 繁体 FGO_iOS：2 繁体 FGO_MyCard 支付版：3 KOMOE SDK 类型： KOMOE_Google 支付版：1 KOMOE_iOS：2 KOMOE_Mycard 支付版：3 。

6.4 建表示例

6.4.1 登录 log 表:

```
CREATE TABLE xx_player_login_logs (
  id INT (10) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '主键',
  uid BIGINT (20) UNSIGNED NOT NULL COMMENT '玩家 id',
  platform_idtinyint (2) UNSIGNED NOT NULL COMMENT '平台 id 1.IOS 2.Bili 3.CPS
4.UO 5.WEB',
  channel_idint (10) UNSIGNED NOT NULL COMMENT '渠道 id',
  login_type TINYINT (2) NOT NULL COMMENT '类型 1.注册和 2.登录',
```

```
  server_id INT (10) DEFAULT NULL COMMENT '区服 id',
  deviceinfo VARCHAR (500) COMMENT '设备 info: 包含设备 ID、系统版本 / 固件类型(注意留有空格)、客户端来源、客户端版本 示例:( 446ce1fa8577de4a5c05fc474157ad5d;Android OS 6.0.1 / API-23 (MXB48T/1);smartisan SM919;com.bilibili.fatego1)',
  created_at INT (10) UNSIGNED NOT NULL COMMENT '时间戳',
  ip VARCHAR (50) COMMENT 'IP 地址 192.168.100.100',
  app_versionvarchar (20) COMMENT '版本号',
  PRIMARY KEY (id),
  INDEX idx_platform (platform_id) USING BTREE,
  INDEX idx_channel (channel_id) USING BTREE
) ENGINE = INNODB DEFAULT CHARSET = utf8mb4 COMMENT '登陆 log 表';
```

6.4.2 10 库 10 表用户表:

```
CREATE TABLE xx_player (
  uid BIGINT (20) UNSIGNED NOT NULL COMMENT '玩家 id',
  platform_id TINYINT (2) UNSIGNED NOT NULL COMMENT '平台 id 1.IOS 2.Bili 3.CPS 4.UO 5.WEB',
  channel_id INT (10) UNSIGNED NOT NULL COMMENT '渠道 id',
  LEVEL INT (10) UNSIGNED DEFAULT NULL COMMENT '等级',
  last_ac VARCHAR (50) DEFAULT NULL COMMENT '最后一次行为',
  last_ac_time INT (10) UNSIGNED DEFAULT NULL COMMENT '最后一次执行时间',
  server_idint (10) UNSIGNED DEFAULT NULL COMMENT '区服 id',
  regtime INT (10) DEFAULT NULL COMMENT '注册时间',
  PRIMARY KEY (uid),
  INDEX idx_last_ac_time (last_ac_time) USING BTREE
) ENGINE = INNODB DEFAULT CHARSET = utf8mb4 ;
```

6.4.3 用户总表:

```
CREATE TABLE xx_user_all (
  uid BIGINT (20) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '用户游戏自增 id',
  uname VARCHAR (50) COMMENT '玩家名字',
  platform_id TINYINT (2) UNSIGNED NOT NULL COMMENT '平台 id 1.IOS 2.Bili 3.CPS 4.UO 5.WEB',
  channel_id INT (10) UNSIGNED NOT NULL COMMENT '渠道 id',
  account_name VARCHAR (50) COMMENT '通行证名称 appname',
  account_uid VARCHAR (32) NOT NULL COMMENT '通行证 uuid',
  sex TINYINT (2) UNSIGNED DEFAULT 0 COMMENT '性别 性别 1: 男 2: 女 0:其他',
  user_level INT (10) UNSIGNED COMMENT '等级',
  firstpaytime INT (10) COMMENT '首冲时间',
  regtime INT (10) COMMENT '注册时间',
```

```
  lasttime INT (10) COMMENT '最后登陆时间',
  pay_total INT (10) COMMENT '充值总值(单位分)',
  server_id INT (10) UNSIGNED COMMENT '区服 id',
  reg_device_id VARCHAR (50) DEFAULT NULL COMMENT '注册设备 id',
  last_device_id VARCHAR (50) DEFAULT NULL COMMENT '最后登陆设备 ID',
  PRIMARY KEY (uid),
  INDEX idx_uname (uname) USING BTREE,
  INDEX idx_first_pay_time (firstpaytime) USING BTREE,
  INDEX idx_created_at (regtime) USING BTREE,
  INDEX idx_account_name (account_name) USING BTREE,
  UNIQUE account_uid_server_id (account_uid, server_id) USING BTREE
) ;
```

6.4.4 支付表:

```
CREATE TABLE xx_payment (
  id INT (10) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '主键',
  platform_id TINYINT (2) NOT NULL COMMENT '平台 id 1.IOS 2.Bili 3.CPS 4.UO
5.WEB',
  channel_id INT (10) UNSIGNED NOT NULL COMMENT '渠道 ID',
  uid BIGINT (20) UNSIGNED NOT NULL COMMENT '玩家 id',
  out_trade_no VARCHAR (80) NOT NULL COMMENT '订单号',
  total_feebigint (20) UNSIGNED DEFAULT NULL COMMENT '支付金额(单位分)',
  started_at INT (10) UNSIGNED DEFAULT NULL COMMENT '支付(下单)时间(时间戳)',
  ended_at INT (10) UNSIGNED DEFAULT NULL COMMENT '结单时间(时间戳)',
  order_status TINYINT (2) UNSIGNED DEFAULT NULL COMMENT '订单状态(0 未支付,1.已
支付,已完成,2 手工修复,已完成)',
  server_id INT (10) UNSIGNED DEFAULT NULL COMMENT '区服 id',
  PRIMARY KEY (id),
  INDEX idx_pay_time (started_at) USING BTREE,
) ENGINE = INNODB DEFAULT CHARSET = utf8mb4 COMMENT '支付表';
```

七、范例数据库

7.1 XX_mdb:固定配置库包括卡片，装备，产品等固定配置表，表内数据跟用户无关系，只跟游戏版本有关

7.2 XX_peyment:订单支付库，包括

订单充值支付表按月分表

payment_201802

payment_201803

payment_201804

7.3 XX_member:用户登录库，包括用户登录表，记录玩家登陆帐号，渠道，第三方关联账号，用户机型，用户 mac 等

7.4 XX_udb:用户数据库划分为 10 库每库内分 10 表，Uid 最后一位决定落在哪个库，后两位决定落在那张表，比如 10000321 落在 1 库的 21 表

XX_udb_00 库包括

用户基础数据 user_00~user_90

用户状态数据 user_state_00~user_state_90

用户装备数据 user_item_00~user_item_90

用户卡片数据 user_card_00~user_card_90

等表数据随用户进度而增加

~

XX_udb_09 库包括

用户基础数据 user_09~user_99，

用户状态数据 user_state_09~user_state_99，

用户装备数据 user_item_09~user_item_99，

用户卡片数据 user_card_09~user_card_99

等表数据随用户进度而增加

7.5 XX_log：日志库，包括用户登录，用户任务等流水数据，需要按天进行分表

登录日志表 login_20180202

登录日志表 login_20180203

登录日志表 login_20180204

任务日志表 task_20180202

任务日志表 task_20180203

任务日志表 task_20180204

如果日志库内表过多需要拆分到不同的日志库

附录一：常用类型所占字节以及取值范围：

日期和时间数据类型

MySQL 数据类型	含义（有符号）
date	3 字节，日期，格式：2014-09-18
time	3 字节，时间，格式：08:42:30
datetime	8 字节，日期时间，格式：2014-09-18 08:42:30

MySQL 数据类型	含义（有符号）
timestamp	4 字节，自动存储记录修改的时间
year	1 字节，年份

由于 timestamp 只占 4 字节，所以性能会比 datetime 更占优势，所以可以考虑用 timestamp 取代 datetime。但是 timestamp 有两个需要注意的，一是取值范围是’1970-01-01 00:00:00’到 2037 年，二是如果数据库修改时区，所在实例的所有 timestamp 会受到影响。

数值数据类型

整型

MySQL 数据类型	含义（有符号）
tinyint	1 字节，范围（-128~127）
smallint	2 字节，范围（-32768~32767）
mediumint	3 字节，范围（-8388608~8388607）
int	4 字节，范围（-2147483648~2147483647）
bigint	8 字节，范围（+-9.22*10 的 18 次方）

上面定义的都是有符号的，当然了，也可以加上 unsigned 关键字，定义成无符号的类型，那么对应的取值范围就要翻翻了，比如：

tinyint unsigned 的取值范围为 0~255。

浮点型

MySQL 数据类型	含义
float(m, d)	4 字节，单精度浮点型，m 总个数，d 小数位

MySQL 数据类型	含义
double(m, d)	8 字节，双精度浮点型，m 总个数，d 小数位
decimal(m, d)	decimal 是存储为字符串的浮点数

在使用浮点型的时候，还是要注意陷阱的，要以插入数据库中的实际结果为准。

字符串数据类型

MySQL 数据类型	含义
char(n)	固定长度，最多 255 个字符
varchar(n)	可变长度，最多 65535 个字符
tinytext	可变长度，最多 255 个字符
text	可变长度，最多 65535 个字符
mediumtext	可变长度，最多 2 的 24 次方-1 个字符
longtext	可变长度，最多 2 的 32 次方-1 个字符

char（n）和 varchar（n）中括号中 n 代表字符的个数，并不代表字节个数，所以当使用了中文的时候(UTF8)意味着可以插入 m 个中文，但是实际会占用 m*3 个字节。

同时 char 和 varchar 最大的区别就在于 char 不管实际 value 都会占用 n 个字符的空间，而 varchar 只会占用实际字符应该占用的空间+1，并且实际空间+1<=n。

超过 char 和 varchar 的 n 设置后，字符串会被截断。

char 的上限为 255 字节，varchar 的上限 65535 字节，text 的上限为 65535 字节。