# 规范背景与目的

MySQL数据库与Oracle、SQLServer等数据库相比，有其内核上的优势与劣势。我们在使用MySQL数据库的时候需要遵循一定规范，扬长避短。本规范旨在帮助或指导RD、QA、OP等技术人员做出适合线上业务的数据库设计。在数据库变更和处理流程、数据库表设计、SQL编写等方面予以规范，从而为公司业务系统稳定、健康地运行提供保障。

# 数据库设计规范

以下所有规范会按照【高危】、【强制】、【建议】三个级别进行标注，遵守优先级从高到低。对于不满足【高危】和【强制】两个级别的设计，需要进行修改。

## 通用命名约定

1.【强制】使用小写，有助于提高打字速度，避免因大小写敏感而导致的错误。

2.【强制】没有空格，使用下划线代替。

3.【强制】名称中没有数字，只有英文字母。

4.【强制】有效的可理解的名称。

5.【强制】名称应该是自我解释的。

6.【强制】名称不应超过32个字符。

7.【强制】避免使用前缀。

## 库

1.【强制】遵守以上全部一般命名规则。

2.【强制】使用单数。

4.【强制】库的名称格式：业务系统名称\_子系统名。

5.【强制】一般分库名称命名格式是`库通配名\_编号`，编号从0开始递增，比如`northwind\_001`，以时间进行分库的名称格式是`库通配名\_时间`。

6.【强制】创建数据库时必须显式指定字符集，并且字符集只能是utf8或者utf8mb4。创建数据库SQL举例：

```sql

createdatabasedb\_namedefaultcharactersetutf8;

```

## 表

1.【强制】遵守以上全部一般命名规则。

2.【强制】使用单数。

3.【强制】相关模块的表名与表名之间尽量体现join的关系，如`user`表和`user\_login`表。

4.【强制】创建表时必须显式指定字符集为utf8或utf8mb4。

5.【强制】创建表时必须显式指定表存储引擎类型，如无特殊需求，一律为InnoDB。当需要使用除InnoDB/MyISAM/Memory以外的存储引擎时，必须通过DBA审核才能在生产环境中使用。因为InnoDB表支持事务、行锁、宕机恢复、MVCC等关系型数据库重要特性，为业界使用最多的MySQL存储引擎。而这是其它大多数存储引擎不具备的，因此首推InnoDB。

6.【强制】建表必须有comment。

7.【强制】关于主键：(1)命名为`id`；(2)标识表里每一行主体的字段不要设为主键，建议设为其它字段如`user\_id`，`order\_id`等，并建立`uniquekey`索引。因为如果设为主键且主键值为随机插入，则会导致InnoDB内部page分裂和大量随机I/O，性能下降。

8.【建议】核心表（如用户表，金钱相关的表）必须有行数据的创建时间字段`create\_time`和最后更新时间字段`update\_time`，便于排查问题。

9.【建议】表中所有字段必须都是`NOT NULL`属性，业务可以根据需要定义`DEFAULT`值。因为使用`NULL`值会存在每一行都会占用额外存储空间、数据迁移容易出错、聚合函数计算结果偏差等问题。

10.【建议】建议对表里的`blob`、`text`等大字段，垂直拆分到其它表里，仅在需要读这些对象的时候才去select。

11.【建议】反范式设计：把经常需要join查询的字段，在其它表里冗余一份。如`username`属性在`user\_account`，`user\_login\_log`等表里冗余一份，减少join查询。

12.【强制】中间表用于保留中间结果集，名称必须以`tmp\_`开头。备份表用于备份或抓取源表快照，名称必须以`bak\_`开头。中间表和备份表定期清理。

13.【强制】对于超过100W行的大表进行`alter table`，必须经过DBA审核，并在业务低峰期执行。因为`altertable`会产生表锁，期间阻塞对于该表的所有写入，对于业务可能会产生极大影响。

## 字段名

1.【强制】遵守以上全部一般命名规则。

2.【建议】尽可能选择短的或一两个单词。

3.【强制】避免使用保留字作为字段名称：`order`，`date`，`name`是数据库的保留字，避免使用它。可以为这些名称添加前缀使其易于理解，如`user\_name`，`signup\_date`等。

4.【强制】避免使用与表名相同的字段名，这会在编写查询时造成混淆。

5.【强制】在数据库模式上定义外键。

6.【强制】避免使用缩写或基于首字母缩写词的名称。

7.【强制】外键列必须具有表名及其主键，例如：`blog\_id`表示来自表博客的主键id。

## 字段数据类型优化

1.【建议】表中的自增列（`auto\_increment`属性），推荐使用`bigint`类型。因为无符号`int`存储范围为`0~4,294,967,295`（不到43亿），溢出后会导致报错。

2.【建议】业务中选择性很少的状态`status`、类型`type`等字段推荐使用`tinytint`或者`smallint`类型节省存储空间。

3.【建议】业务中IP地址字段推荐使用`int`类型，不推荐用`char(15)`。因为`int`只占4字节，可以用如下函数相互转换，而`char(15)`占用至少15字节。

SQL:

```sql

selectinet\_aton('13.107.21.200');

selectinet\_ntoa(225121736);

```

PHP:

```php

ip2long('192.168.2.12');

long2ip(3530427185);

```

Java:

```java

publicstaticlongipToLong(Stringaddr)

{

String[]addrArray=addr.split("\\.");

longnum=0;

for(inti=0;i<addrArray.length;i++)

{

intpower=3-i;

num+=((Integer.parseInt(addrArray[i])%256\*Math.pow(256,power)));

}

returnnum;

}

publicstaticStringlongToIp(longi)

{

return((i>>24)&0xFF)+"."+

((i>>16)&0xFF)+"."+

((i>>8)&0xFF)+"."+

(i&0xFF);

}

```

4.【建议】不推荐使用`blob`，`text`等类型。它们都比较浪费硬盘和内存空间。在加载表数据时，会读取大字段到内存里从而浪费内存空间，影响系统性能。InnoDB中当一行记录超过8098字节时，会将该记录中选取最长的一个字段将其768字节放在原始page里，该字段余下内容放在`overflow-page`里。不幸的是在`compact`行格式下，原始`page`和`overflow-page`都会加载。

5.【建议】存储金额的字段，建议用`int`类型以「分」而不是「元」为单位存储，存储千兆级别的金额可以使用`bigint`类型，程序端乘以100和除以100进行存取，如1元在数据库中用整型类型100存储。

>不建议使用`decimal`类型存储金额，`decimal`是通过二进制实现的一种编码方式，计算效率不如整型。

>使用整型的字段是定长字段，存储高效，而`decimal`根据定义的宽度决定，在数据库设计中，定长存储更高效。

6.【建议】文本数据尽量用`varchar`存储。因为`varchar`是变长存储，比`char`更省空间。MySQLserver层规定一行所有文本最多存65535字节，因此在utf8字符集下最多存21844个字符，超过会自动转换为`mediumtext`字段。而`text`在utf8字符集下最多存21844个字符，`mediumtext`最多存2^24/3个字符，`longtext`最多存2^32个字符。一般建议用`varchar`类型，字符数不要超过2700。

7.【建议】时间类型尽量选取`timestamp`。因为`datetime`占用8字节，`timestamp`仅占用4字节，但是范围为`1970-01-0100:00:01`到`2038-01-0100:00:00`。更为高阶的方法，选用`int`来存储时间，使用SQL函数`unix\_timestamp()`和`from\_unixtime()`来进行转换。

## 索引设计

1.【强制】InnoDB表必须主键为`id`，且主键值禁止被更新。

2.【建议】主键的名称以`pk\_`开头，唯一键以`uk\_`开头，普通索引以`idx\_`开头，一律使用小写格式，以表名/字段的名称或缩写作为后缀。

3.【强制】InnoDB和MyISAM存储引擎表，索引类型必须为`BTREE`；MEMORY表可以根据需要选择`HASH`或者`BTREE`类型索引。

4.【强制】单个索引中每个索引记录的长度不能超过64KB。

5.【建议】单个表上的索引个数不能超过5个。

6.【建议】在建立索引时，多考虑建立联合索引，并把区分度最高的字段放在最前面。如列`user\_id`的区分度可由`select count(distinct user\_id)`计算出来。

7.【建议】在多表join的SQL里，保证被驱动表的连接列上有索引，这样join执行效率最高。

8.【建议】建表或加索引时，保证表里互相不存在冗余索引。对于MySQL来说，如果表里已经存在`key(a,b)`，则`key(a)`为冗余索引，需要删除。

9.【建议】如果选择性超过20%，那么全表扫描比使用索引性能更优，即没有设置索引的必要。

## **分库**分表、分区表

1.【强制】分区表的分区字段（`partition-key`）必须有索引，或者是组合索引的首列。

2.【强制】单个分区表中的分区（包括子分区）个数不能超过1024。

3.【强制】上线前RD或者DBA必须指定分区表的创建、清理策略。

4.【强制】访问分区表的SQL必须包含分区键。

5.【建议】单个分区文件不超过2G，总大小不超过50G。建议总分区数不超过20个。

6.【强制】对于分区表执行`alter table`操作，必须在业务低峰期执行。

7.【强制】采用分库策略的，库的数量不能超过1024。

8.【强制】采用分表策略的，表的数量不能超过4096。

9.【建议】单个分表不超过500W行，ibd文件大小不超过2G，这样才能让数据分布式变得性能更佳。

10.【建议】水平分表尽量用取模方式，日志、报表类数据建议采用日期进行分表。

## 字符集

1.【强制】数据库本身库、表、列所有字符集必须保持一致，为`utf8`或`utf8mb4`。

2.【强制】前端程序字符集或者环境变量中的字符集，与数据库、表的字符集必须一致，统一为`utf8`。

## 程序层DAO设计建议

1.【建议】新的代码不要用model，推荐使用手动拼SQL+绑定变量传入参数的方式。因为model虽然可以使用面向对象的方式操作db，但是其使用不当很容易造成生成的SQL非常复杂，且model层自己做的强制类型转换性能较差，最终导致数据库性能下降。

2.【建议】前端程序连接MySQL或者Redis，必须要有连接超时和失败重连机制，且失败重试必须有间隔时间。

3.【建议】前端程序报错里尽量能够提示MySQL或Redis原生态的报错信息，便于排查错误。

4.【建议】对于有连接池的前端程序，必须根据业务需要配置初始、最小、最大连接数，超时时间以及连接回收机制，否则会耗尽数据库连接资源，造成线上事故。

5.【建议】对于`log`或`history`类型的表，随时间增长容易越来越大，因此上线前RD或者DBA必须建立表数据清理或归档方案。

6.【建议】在应用程序设计阶段，RD必须考虑并规避数据库中主从延迟对于业务的影响。尽量避免从库短时延迟（20秒以内）对业务造成影响，建议强制一致性的读开启事务走主库，或更新后过一段时间再去读从库。

7.【建议】多个并发业务逻辑访问同一块数据（InnoDB表）时，会在数据库端产生行锁甚至表锁导致并发下降，因此建议更新类SQL尽量基于主键去更新。

8.【建议】业务逻辑之间加锁顺序尽量保持一致，否则会导致死锁。

9.【建议】对于单表读写比大于10:1的数据行或单个列，可以将热点数据放在缓存里（如Memcached或Redis），加快访问速度，降低MySQL压力。

## 一个规范的建表语句示例

-一个较为规范的建表语句为：

```sql

create table user

(

`id` bigint(11) not null auto\_increment,

`user\_id` bigint(11) not null comment '用户ID',

`username` varchar(45) not null comment '登录名',

`email`varchar(30) not null comment '邮箱',

`nickname`varchar(45) not null comment '昵称',

`avatar` int(11) not null comment '头像',

`birthday` date not null comment '生日',

`gender`tiny int(4) default '0' comment '性别',

`intro` varchar(150) default null comment '简介',

`resume\_url` varchar(300) not null comment '简历存放地址',

`register\_ip` int not null comment '用户注册时的源IP',

`review\_status` tinyint not null comment '审核状态，1-通过，2-审核中，3-未通过，4-尚未提交审核',

`create\_time` timestamp not null comment '记录创建的时间',

`update\_time` timestamp not null comment '资料修改的时间',

primary key(`id`),

unique key `idx\_user\_id` (`user\_id`),

key `idx\_username` (`username`),

key `idx\_create\_time` (`create\_time`,`review\_status`)

)

engine=InnoDB

defaultcharset=utf8

comment='用户基本信息';

```

# SQL编写规范

## DML语句

1.【强制】select语句必须指定具体字段名称，禁止写成`\*`。因为`select\*`会将不该读的数据也从MySQL里读出来，造成I/O压力。

2.【强制】insert语句指定具体字段名称，不要写成`insert into t1 values(…)`，道理同上。

3.【建议】`insert into… values(xx),(xx),(xx)…`，这里xx的值不要超过5000个。值过多虽然上线很快，但会引起主从同步延迟。

4.【建议】select语句不要使用`union`，推荐使用`union all`，并且`union`子句个数限制在5个以内。因为`union all`不需要去重，节省数据库资源，提高性能。

5.【建议】in值列表限制在500以内。例如`select … where user\_id in(…500个以内…)`，这么做是为了减少底层扫描，减轻数据库压力从而加速查询。

6.【建议】事务里批量更新数据需要控制数量，进行必要的sleep，做到少量多次。

7.【强制】事务涉及的表必须全部是InnoDB表。否则一旦失败不会全部回滚，且易造成主从库同步中断。

8.【强制】写入和事务的SQL发往主库，只读SQL发往从库。

9.【强制】除静态表或小表（100行以内），dml语句必须有where条件，且使用索引查找。

10.【强制】生产环境禁止使用`hint`，如`sql\_no\_cache`，`force index`，`ignore key`，`straight join`等。因为`hint`是用来强制sql按照某个执行计划来执行，但随着数据量变化我们无法保证自己当初的预判是正确的，因此我们要相信MySQL优化器。

11.【强制】where条件里等号左右字段类型必须一致，否则无法利用索引。

12.【建议】`select | update | delete | replace`要有where子句，且where子句的条件必需使用索引查找。

13.【强制】生产数据库中强烈不推荐大表上发生全表扫描，但对于100行以下的静态表可以全表扫描。查询数据量不要超过表行数的25%，否则不会利用索引。

14.【强制】where子句中禁止只使用全模糊的like条件进行查找，必须有其它等值或范围查询条件，否则无法利用索引。

15.【建议】索引列不要使用函数或表达式，否则无法利用索引。如`where length(name)='admin'`或`where user\_id + 2 = 10023`。

16.【建议】减少使用or语句，可将or语句优化为union，然后在各个where条件上建立索引。如`where a=1 or b=2`优化为`where a=1…union…where b=2 , key(a),key(b)`。

17.【建议】分页查询，当`limit`起点较高时，可先用过滤条件进行过滤。如`select a, b, c from t1 limit 10000 ,20;`优化为:`select a, b, c from t1 where id > 10000 limit 20;`。

## 多表连接

1.【强制】禁止跨DB的join语句。因为这样可以减少模块间耦合，为数据库拆分奠定坚实基础。

2.【强制】禁止在业务的更新类SQL语句中使用join，比如`update t1 join t2…`。

3.【建议】不建议使用子查询，建议将子查询SQL拆开结合程序多次查询，或使用join来代替子查询。

4.【建议】线上环境，多表join不要超过3个表。

5.【建议】多表连接查询推荐使用别名，且select列表中要用别名引用字段，数据库.表格式，如`select a from db1.table1 alias1 where…`。

6.【建议】在多表join中，尽量选取结果集较小的表作为驱动表，来join其它表。

## 事务

1.【建议】事务中`insert|update|delete|replace`语句操作的行数控制在2000以内，以及where子句中in列表的传参个数控制在500以内。

2.【建议】批量操作数据时，需要控制事务处理间隔时间，进行必要的sleep，一般建议值5-10秒。

3.【建议】对于有`auto\_increment`属性字段的表的插入操作，并发需要控制在200以内。

4.【强制】程序设计必须考虑“数据库事务隔离级别”带来的影响，包括脏读、不可重复读和幻读。线上建议事务隔离级别为`repeatable-read`。

5.【建议】事务里包含SQL不超过5个（支付业务除外）。因为过长的事务会导致锁数据较久，MySQL内部缓存、连接消耗过多等雪崩问题。

6.【建议】事务里更新语句尽量基于主键或`unique key`，如`update … where id = XX;`，否则会产生间隙锁，内部扩大锁定范围，导致系统性能下降，产生死锁。

7.【建议】尽量把一些典型外部调用移出事务，如调用WebService，访问文件存储等，从而避免事务过长。

8.【建议】对于MySQL主从延迟严格敏感的select语句，请开启事务强制访问主库。

## 排序和分组

1.【建议】减少使用`order by`，和业务沟通能不排序就不排序，或将排序放到程序端去做。`order by`、`group by`、`distinct`这些语句较为耗费CPU，数据库的CPU资源是极其宝贵的。

2.【建议】`order by`、`group by`、`distinct`这些SQL尽量利用索引直接检索出排序好的数据。如`where a = 1 order by`可以利用`key(a,b)`。

3.【建议】包含了`order by`、`group by`、`distinct`这些查询的语句，where条件过滤出来的结果集请保持在1000行以内，否则SQL会很慢。

## 线上禁止使用的SQL语句

1.【高危】禁用`update|delete t1 … where a = XX limit XX ;`这种带limit的更新语句。因为会导致主从不一致，导致数据错乱。建议加上`order by PK`。

2.【强制】禁用procedure、function、trigger、views、event、外键约束。因为他们消耗数据库资源，降低数据库实例可扩展性。推荐都在程序端实现。

3.【强制】禁用`insert into … on duplicate key update…`在高并发环境下，会造成主从不一致。

4.【强制】禁止联表更新语句，如`update t1 , t2 where t1.id = t2.id…`。