# 社区MySQL数据库设计规范



# 数据库表设计

# 配置规范

1. MySQL 数据库默认使用 InnoDB 存储引擎。
2. 保证字符集设置统一，MySQL 数据库相关系统、数据库、表和字段的字符集都用 UTF8，应用程序连接、展示、客户端等可以设置字符集的地方也都统一设置为 UTF8 字符集。
3. Mysql数据库的事务隔离级别默认为 RR（Repeatable-Read），建议初始化时统一设置为 RC（Read-Committed），对于 OLTP 业务更合适。
4. 数据库中的表要合理规划，控制单表数据量，对于 MySQL 数据库来说，建议单表记录数控制在2000万条数据之内。
5. MySQL 实例下，数据库、表数量尽可能少：数据库一般不超过50个，每个数据库下，数据表数量一般不超过500个（包括分区表）。

## 建表规范

1. 表和列的名称建议控制在12个字符以内，表名只能使用字母、数字和下划线，一律小写。
2. 表名要求以 tb_ 为前缀,并与模块名强相关，如师资系统采用”tb_teacher”作为前缀，渠道系统采用”tb_channel”作为前缀等(禁止使用拼音全称或缩写来命名表名！！！)。
3. 建表时关于主键：主键字段建议为 int 或 bigint， 如果配置表可以为auto_increment。如果是业务表，数据量大的表可用统一id生成工具生成
4. InnoDB 禁止使用外键约束，可以使用虚拟外键
5. 核心表必须有创建时间字段 create_date 和最后更新时间字段 update_date，便于查问题和数据清洗。
   创建时间设置默认值，更新时间设置默认值，且每次更新自动更新，如果是可以为空的 datetime 类型字段则设置默认值1970-01-01 00:00:00
6. 存储精确浮点数必须使用 decimal 代替 float 和 double。
7. 整形定义中无需定义显示宽度，比如：使用int，而不是int(4)，如确认定长则需要宽度。
   int(4) 不是指存储范围是0-9999，如果存储范围只要0-9999，可以改用 smallint 型，具体各数据类型长度请查看附录表格。括号里面的数字仅在字段加上 zerofill 属性时显示宽度，否则 int(2)、int(4) 和 int(10) 无任何差别。
8. 不建议使用 enum 枚举类型，可使用 tinyint 来代替。
   enum 类型属于字符型数据，在表定义时就把枚举值写死了，且浪费存储空间。如果增加枚举值需要执行相应的 DDL 操作。TINYINT 则是数值型，存储只占用1个字节，对于类型扩展很方便操作，不需要执行 DDL。
9. 尽可能不使用 text、blob 类型，如果必须使用，建议将过大字段或是不常用的描述型大字段拆分到其他表中；另外，禁止用关系型数据库存储图片或文件。
10. 全部字段都为not null，但如果确实业务不要求为空的，设置默认值，为了后期查询语句方便，is not null是不走索引的。
11. 反范式设计：把经常需要 join 查询的字段，在其他表里冗余一份。如：user_name 属性在 user_account，user_login_log 等表里冗余一份，减少 join 查询。
12. 业务中IP地址字段推荐使用 `int` 类型，不推荐使用 `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。一旦表数据行数到了1亿行以上，那么要多用 1.1G 存储空间。
    `select inet_aton('192.168.1.1'); select inet_ntoa(3232235777);`
13. 订单表 varchar 单个字段类型最大建议长度为1024 即 varchar（1024）,建议是 varchar（255）以下
14. 单表字段数不超过30个，超过需要拆分父子表。
15. 不要用 json 字段类型。

## 命名规范

1. 库、表、字段全部采用小写。
2. 库名、表名、字段名、索引名称均使用小写字母，并以“_”分隔。
3. 库名、表名、字段名建议不超过12个字符。（对多支持64个字符）
4. 库名、表名、字段名见名知意，并且添加注释。
5. 库名、表名、字段不允许出现数字，需要全英文命名。
6. 命名时避开 MySQL 关键字。

## 索引规范

1. 索引建议命名规则：idx_col1_col2[_colN]、uk_col1_col2[_colN]（如果字段过长建议采用缩写）。
2. 索引中的字段数建议不超过5个。
3. 单张表的索引个数控制在5个以内。
4. InnoDB 表强制要求有主键列。
5. 建立复合索引时，优先将选择性高的字段放在前面。
6. UPDATE、DELETE 语句需要根据 WHERE 条件添加索引。
7. 不建议使用%前缀模式查询。例如 LIKE ‘%WEIBO’，无法使用索引，会导致全表扫描。
8. 合理利用覆盖索引。例如：SELECT email,uid FROM user_email WHERE uid=xxx，如果 uid 不是主键，可以创建覆盖索引 idx_uid_email (uid,email) 来提高查询效率。
9. 避免在索引字段上使用函数，否则会导致查询时索引失效。
10. 避免不必要的重复或冗余索引，根据索引最左前缀使用原则，如表中的存在索引 `key(a,b)`，则索引 `key(a)`

# 代码编写

## DAO

1. 推荐使用手动拼 SQL+ 绑定变量传入参数的方式。
2. 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. 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. 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. 对于 log 或 history 类型的表，随时间增长容易越来越大，因此上线前开发和 DBA 必须建立表数据清理或归档方案。
6. 在应用程序设计阶段，开发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. 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. 业务逻辑之间加锁顺序尽量保持一致，否则会导致死锁。
9. 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mecache或redis），加快访问速度，降低 MySQL 压力。

## SQL规范

### DML语句

1. 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，造成网卡压力。且表字段一旦更新，但 model 层没有来得及更新的话，系统会报错。
2. insert 语句指定具体字段名称，不要写成 insert into t1 values(…)，道理同上。
3. insert into…values(XX),(XX),(XX)…。这里 XX 的值不要超过5000个。值过多虽然上线很很快，但会引起主从同步延迟。
4. select 语句慎重使用 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. in 值列表限制在500以内。例如 select… where userid 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. 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. 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终端。
8. 写入和事务发往主库，只读 sql 发往从库。
9. 除静态表或小表（1000行以内），DML 语句必须有 where 条件，且使用索引查找。
10. 生产环境禁止使用 hint，如 sql_no_cache，force index，ignore key，straight join 等。
11. 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. 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. 生产数据库中强烈不推荐大表上发生全表扫描，但对于1000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. where 子句中禁止只使用全模糊的 like 条件进行查找，必须有其他等值或范围查询条件，否则无法利用索引。
15. 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. 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. 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. 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. 禁止在业务的更新类 SQL 语句中使用 join，比如 update t1 join t2…。
3. 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. 线上环境，多表 join 不要超过3个表。
5. 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. 在多表 join 中，尽量选取结果集较小的表作为驱动表，来 join 其他表。

## 事务

1. 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. 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. 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. 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 Read-Commit。
5. 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. 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. 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. 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。

### 排序分组

1. 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. 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. 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

1. 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 主键。
2. 禁止使用关联子查询，如 update t1 set … where name in(select name from user where…); 效率极其低下。
3. 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。
4. 禁用 insert into …on duplicate key update… 在高并发环境下，会造成主从不一致。
5. 禁止联表更新语句，如 update t1,t2 where t1.id=t2.id…。

# 附录

## DDL示例

一个比较符合规范的ddl示例

```
CREATE TABLE `tb_phone_type` (
 `id` BIGINT(20) NOT NULL AUTO_INCREMENT COMMENT 'id',
 `type` TINYINT(2) NOT NULL DEFAULT '0' COMMENT '0ios1android',
 `name` VARCHAR(255) NOT NULL DEFAULT '' COMMENT '',
 `high` INT(11) NOT NULL DEFAULT '0' COMMENT '',
 `width` INT(11) NOT NULL DEFAULT '0' COMMENT '',
 `STATUS` TINYINT(1) NOT NULL DEFAULT '1' COMMENT '0/1',
 `create_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '',
 `create_by` BIGINT(20) NOT NULL DEFAULT '1' COMMENT '',
 `update_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
 `update_by` BIGINT(20) NOT NULL DEFAULT '1' COMMENT '',
 PRIMARY KEY (`id`)
)
COMMENT=''
COLLATE='utf8mb4_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=0;
```

新建表时需要赋予相应权限

```
grant select,update,insert,delete on db.tb_user to dbuser@'%';
```

## MySQL 数据类型

### 整型

| 类型 | 大小 | 范围（有符号） | 范围（无符号） | 用途 |
|---|---|---|---|---|
| TINYINT | 1 字节 | (-128，127) | (0，255) | 小整数值 |
| SMALLINT | 2 字节 | (-32 768，32 767) | (0，65 535) | 大整数值 |
| MEDIUMINT | 3 字节 | (-8 388 608，8 388 607) | (0，16 777 215) | 大整数值 |
| INT或INTEGER | 4 字节 | (-2 147 483 648，2 147 483 647) | (0，4 294 967 295) | 大整数值 |
| BIGINT | 8 字节 | (-9,223,372,036,854,775,808，9 223 372 036 854 775 807) | (0，18 446 744 073 709 551 615) | 极大整数值 |
| FLOAT | 4 字节 | (-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38) | 0，(1.175 494 351 E-38，3.402 823 466 E+38) | 单精度<br>浮点数值 |
| DOUBLE | 8 字节 | (-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308) | 0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308) | 双精度<br>浮点数值 |
| DECIMAL | 对DECIMAL(M,D) ，如果M>D，为M+2否则为D+2 | 依赖于M和D的值 | 依赖于M和D的值 | 小数值 |

### 日期和时间类型

| 类型 | 大小<br>(字节) | 范围 | 格式 | 用途 |
| --- | --- | --- | --- | --- |
| DATE | 3 | 1000-01-01/9999-12-31 | YYYY-MM-DD | 日期值 |
| TIME | 3 | '-838:59:59'/'838:59:59' | HH:MM:SS | 时间值或持续时间 |
| YEAR | 1 | 1901/2155 | YYYY | 年份值 |
| DATETIME | 8 | 1000-01-01<br>00:00:00/9999-12-31<br>23:59:59 | YYYY-MM-DD HH:MM:SS | 混合日期和时间值 |
| TIMESTAMP | 4 | 1970-01-01 00:00:00/2038<br><br>结束时间是第 2147483647 秒，北京时间 2038-1-19 11:14:07，格林尼治时间 2038年1月19日 凌晨 03:14:07 | YYYYMMDD HHMMSS | 混合日期和时间值，时间戳 |

## 字符串类型

| 类型 | 大小 | 用途 |
| --- | --- | --- |
| CHAR | 0-255字节 | 定长字符串 |
| VARCHAR | 0-65535 字节 | 变长字符串 |
| TINYBLOB | 0-255字节 | 不超过 255 个字符的二进制字符串 |
| TINYTEXT | 0-255字节 | 短文本字符串 |
| BLOB | 0-65 535字节 | 二进制形式的长文本数据 |
| TEXT | 0-65 535字节 | 长文本数据 |
| MEDIUMBLOB | 0-16 777 215字节 | 二进制形式的中等长度文本数据 |
| MEDIUMTEXT | 0-16 777 215字节 | 中等长度文本数据 |
| LONGBLOB | 0-4 294 967 295字节 | 二进制形式的极大文本数据 |
| LONGTEXT | 0-4 294 967 295字节 | 极大文本数据 |