**MySQL数据库开发规范（2020版）**

**修订记录表**

## 选型规范

## 设计规范

### 使用innodb存储引擎，禁止使用myisam引擎

5.5版本开始MySQL默认存储引擎就是InnoDB，5.7版本开始，系统表放弃MyISAM。

### 库名、表名、字段名、索引名须采用小写英文字母，并以下划线分割

* 1. MySQL有配置参数lower\_case\_table\_names=1，即库表名以小写存储，大小写不敏感。如果是0，则库表名以实际情况存储，大小写敏感；如果是2，以实际情况存储，但以小写比较。
  2. 如果大小写混合使用，可能存在abc，Abc，ABC等多个表共存，容易导致混乱。
  3. 字段名显示区分大小写，但实际使⽤时不区分，即不可以建立两个名字一样但大小写不一样的字段。
  4. 主键索引名为 pk\_ 字段名；唯一索引名为 uk \_字段名 ; 普通索引名则为 idx \_字段名。如: pk\_ 即 primary key；uk \_ 即 unique key；idx \_ 即 index 的简称。
  5. 表达是与否概念的字段，必须使用 is\_xxx 的方式命名。POJO 类的 boolean 属性不能加 is。

### 库名、表名、字段名、索引名长度禁止超过32位，需见名知意

库名、表名、字段名支持最多64个字符，但为了统一规范、易于辨识以及减少传输量。

应能够见名知意，能够体现其存储的数据内容，需使用名词而非动词，但不要使用复数名词。库名与应用名称关联。

相关模块的表名之间尽量体现join的关系，如user表和user\_login表。

同一模块使用的表名尽量使用统一前缀。

### 库名、表名、字段名、索引名禁止使用数据库保留关键字

### 表必须有主键，不建议字符串列和联合主键，建议使用自增或全局生成

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。还有对一些复杂查询可能需要自连接来优化时需要用到。

如果没有主键或唯一索引，update/delete是通过所有字段来定位操作的行，相当于每行就是一次全表扫描。

强制要求主键为id，类型为int或bigint，且为auto\_increment。

标识表里每一行主体的字段不要设为主键，建议设为其他字段如user\_id，order\_id等，并建立unique key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。

### 禁止使用外键，数据完整性在业务端完成

### 禁止在数据库中存储图片、视频等大容量文件

### 字段尽可能不使用blob、text类型，建议使用varchar(N)

### 存储精确浮点数必须使用decimal代替float和double

### 表名、字段设计需要添加备注

如表示状态的字段需指明主要值的含义，如”0-离线，1-在线”

### 若字段只有true or false，请使用tinyint（范围-128~127）

tinyint使用1个字节，一般用于status,type,flag的列。

### 存储时间（精确到秒）建议使用timestamp类型

因timestamp使用4字节且自动赋值和自动更新，datetime是8**字节。**

### 关键业务数据表，禁止硬删除，加上is\_deleted状态字段来标记删除操作

### 单表数据控制在3000w以内，超过容量需考虑业务分表

建议单表行数超过3000w行或者单表容量超过 2 GB 才进行分库分表；如果预计三年后的数据量根本达不到这个级别,请不要在创建表时就分库分表。

### 需控制单表字段数量，上限为30

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。

### 数据库中不允许存储明文密码

一旦数据库发生泄漏，攻击者就可以用密码去其他网站尝试登陆（因为用户往往会将多个网站的密码根据习惯设成一样的），一旦登陆成功，就会造成更严重的后果。

### 选择符合存储需要的最小的数据类型

在数据库中，数据处理以页为单位，列越小，在一个页中所能存储的数据量越多。当加载相同数据量时，列越小，加载的页越少，磁盘IO性能越好。

将字符串转换成数字类型存储，如：将IP地址转换成整型数据，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。一旦表数据行数到了1亿，那么要多用1.1G存储空间。

对于非负型的数据，必须是 unsigned。

对于值范围为0-255的字段，使用tinyint unsigned类型。

如果存储的字符串长度相等,必须使用 char 定长字符串类型。

varchar是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长度大于此值，定义字段类型为 text ，独立出来一张表，用主键来对应，避免影响其它字段索引效率。

手机号使用varchar存储。

int使用固定4个字节存储。

int(4)与int(5)后的括号中的字符表示显示宽度，整数列的显示宽度与MySQL需要用多少个字符来显示该列数值，与该整数需要的存储空间的大小都没有关系，int类型的字段能存储的数据上限还是2147483647(有符号型)和4294967295(无符号型)。其实当我们在选择使用INT的类型的时候，不论是int(4)还是int(5)，它在数据库里面存储的都是4个字节的长度。

### 金额字段的设计

设计为Decimal类型，存储单位是分。

### 字段都定义为NOT NULL

* 1. 如果是索引字段，一定要定义为not null 。因为null值会影响cordinate统计，影响优化器对索引的选择。
  2. 如果不能保证insert时一定有值过来，定义时使用default ‘’ ，或 0。
  3. 业务可以根据需要定义DEFAULT值。

### 同一意义的字段定义必须相同

比如不同表中都有 f\_user\_id 字段，那么它的类型、字段长度要设计成一样。如果查询时关联列类型不一致会自动进行数据类型隐式转换，会造成列上的索引失效，导致查询效率降低）。

### 每张表必有的字段

每张表都需要有id,createtime,updatetime，便于查询问题。

### 小数类型为 decimal ,禁止使用 float 和 double

float 和 double 在存储的时候,存在精度损失的问题,很可能在值的比较时,得到不正确的结果。如果存储的数据范围超过 decimal 的范围,建议将数据拆成整数和小数分开存储。

### WHERE条件中必须使用合适的类型

避免MySQL进行隐式类型转化（导致索引失效）。WHERE条件中的非等值条件（IN、BETWEEN、<、<=、>、>=）会导致后面的条件使用不了索引。

### 禁止使用order by rand 进行随机排序

### 禁止使用不含字段列表的INSERT语句

如：insert into values (a,b,c); 应使用insert into t(c1,c2,c3) values (a,b,c);

### 对同一列进行or判断时，使用in代替or

in操作可以更有效的利用索引，or大多数情况下很少能利用到索引。

### 拆分复杂的大SQL为多个小SQL

### 禁止批量写（UPDATE、DELETE、INSERT）操作

禁止程序中批量写（UPDATE、DELETE、INSERT）操作。如果业务需要，采用上报DBA执行脚本的方式。

### 禁止在表中建立预留字段

### 禁止为程序使用的账号赋予super权限；对于程序连接数据库账号，遵循权限最小原则

### 字段允许适当冗余，以提高查询性能，但必须考虑数据一致

冗余字段应遵循：  
（1）不是频繁修改的字段  
（2）不是 varchar 超长字段，更不能是 text 字段  
正例：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存储类目名称,避免关联查询。

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。

### 禁止使用外键与级联,一切外键概念必须在应用层解决

### 枚举类型

推荐使用tinyint或smallint存储，并在字典表中明确每个值对应的意义。

### insert语句特殊要求

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。

### where条件里等号左右字段类型必须一致

否则无法利用索引。

### SELECT|UPDATE|DELETE|REPLACE要有WHERE子句，且WHERE子句的条件必需使用索引查找

### 禁止在业务的更新类SQL语句中使用join

比如update t1 join t2不可取。

### 建议少用子查询

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。

## 索引规范

### 禁止在低基数列上建索引。禁止将更新频繁的列作为索引

比如性别字段不适合做索引。

### 单张表的索引数不超过5，复合索引中的字段数不超过5

### 禁止在索引列进行各式运算

### 禁止查询不走索引导致的全表扫描

### SQL语句执行计划extra列尽量避免出现：Using File Sort, Using Temporary

### 合理创建联合索引（避免冗余），(a,b,c)相当于(a)、(a,b)、(a,b,c)

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f\_crm\_id作为主键，联合索引(f\_user\_id,f\_crm\_id)上的f\_crm\_id就完全多余。

(a,b,c)、(a,b)，后者为冗余索引。可以利用前缀索引来达到加速目的，减轻维护负担。

### 合理利用覆盖索引，比如select uid, email fromm user\_email where uid=xx，索引为(uid,email)

### 索引尽量建在选择性高的列上

* 1. 不在低基数列上建立索引，例如性别、类型。但有一种情况，idx\_feedbackid\_type (f\_feedback\_id,f\_type)，如果经常用 f\_type=1 比较，而且能过滤掉90%行，那这个组合索引就值得创建。有时候同样的查询语句，由于条件取值不同导致使用不同的索引，也是这个道理。
  2. 索引选择性计算方法（基数 ÷ 数据行数）  
     Selectivity = Cardinality / Total Rows = select count(distinct col1)/count(\*) from tbname，越接近1说明col1上使用索引的过滤效果越好
  3. 走索引扫描行数超过30%时，改全表扫描。

### 最左前缀原则

* 1. 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\_c1\_c2\_c3 (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
  2. 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
  3. 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

### 前缀索引

对超过30个字符长度的列创建索引时，考虑使用前缀索引，如 idx\_cs\_guid2 (f\_cs\_guid(26))表示截取前26个字符做索引，既可以提高查找效率，也可以节省空间

前缀索引也有它的缺点是，如果在该列上 ORDER BY 或 GROUP BY 时无法使用索引，也不能把它们用作覆盖索引(Covering Index)

如果在varbinary或blob这种以二进制存储的列上建立前缀索引，要考虑字符集，括号里表示的是字节数。

### 合理使用覆盖索引减少IO

INNODB存储引擎中，secondary index(非主键索引，又称为辅助索引、二级索引)没有直接存储行地址，而是存储主键值。

如果用户需要查询secondary index中所不包含的数据列，则需要先通过secondary index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 email，uid FROM user\_email WHERE uid=xx，如果uid不是主键，适当时候可以将索引添加为index(uid，email)，以获得性能提升。

### 业务上具有唯一特性的字段必须建成唯一索引

不要以为唯一索引影响了insert速度，这个速度损耗可以忽略，但提高查找速度是明显的 ；另外，即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

### 在varchar字段上建立索引时，必须指定索引长度

varchar类型没必要对全字段建立索引，根据实际文本区分度决定索引长度。

### 索引字段的顺序需要考虑字段值去重之后的个数，个数多的放在前面

### 如果有 order by 的场景，请注意利用索引的有序性

order by 最后的字段是组合索引的一部分,并且放在索引组合顺序的最后。

正例：where a =? and b =? order by c; 索引: a \_ b \_ c

反例：索引中有范围查找，那么索引有序性无法利用，如： WHERE a >10 ORDER BY b; 索引a \_ b 无法排序。

### 其他

区分度最高的列放在联合索引的最左侧；尽量把字段长度小的列放在联合索引的最左侧；尽量把使用最频繁的列放到联合索引的左侧。

存在非等号和等号混合时,在建索引时,请把等号条件的列前置。示例: where c >? andd =? 那么即使 c 的区分度更高,也必须把 d 放在索引的最前列,即索引 idx\_d\_c。（导致索引失效）。

explain sql 进行优化自查。

SQL 性能优化的目标：至少要达到 range 级别， 要求是 ref 级别， 如果可以是 consts

最好。

说明：

1） consts 单表中最多只有一个匹配行（主键或者唯一索引），在优化阶段即可读取到数据。

2） ref 指的是使用普通的索引。（ normal index）

3） range 对索引进范围检索。

必须主键为id int/bigint auto\_increment,且主键值禁止被更新。

索引类型必须为BTREE。

单个索引中每个索引记录的长度不能超过64KB。

单个表上的索引个数不能超过7个。

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。

## 使用规范

### 禁止使用存储过程、自定义函数、触发器

存储过程（procedure）虽然可以简化业务端代码，在传统企业写复杂逻辑时可能会用到，而在互联网企业变更是很频繁的，在分库分表的情况下要升级一个存储过程相当麻烦。又因为它是不记录log的，所以也不方便debug性能问题。如果使用过程，一定考虑如果执行失败的情况。

触发器（trigger）也是同样，但也不应该通过它去约束数据的强一致性，MySQL只支持“基于行的触发”，也就是说，触发器始终是针对一条记录的，而不是针对整个sql语句的，如果变更的数据集非常大的话，效率会很低。掩盖一条sql背后的工作，一旦出现问题将是灾难性的，但又很难快速分析和定位。再者需要ddl时无法使用pt-osc工具。放在transaction执行。

使用自定义函数容易将业务逻辑和DB耦合在一起。

### 禁止使用UUID()、USER()类似的函数，会严重影响MySQL性能与主备同步

### 每个表的主键不能更新频繁

### 禁止左模糊或者全模糊，如like ‘%abc’

会导致索引失效，有这种搜索需求是，考虑其它方案，如ES。

### 禁止在生产环境做数据库压力测试，及其他影响性能的操作。

### 尽量用单表查询，避免多表join，多表查询建议不超过3张。禁止跨db的join语句

这样可以减少模块间耦合，为数据库拆分奠定坚实基础。

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 …。

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。

### where条件必须使用合适的类型，避免MySQL隐式转换，如id字段为int类型，必须id=1，不能id='1'

### 合理使用sql语句来减少与数据库的交互，批量插入建议使用batch处理

### 严禁从开发、测试环境直连生产数据使用

### 能确定返回结果只有一条时，使用 limit 1

在保证数据不会有误的前提下，能确定结果集数量时，多使用limit，尽快的返回结果。

### 使用join时，where条件尽量使用充分利用同一表上的索引

* 1. 如 select t1.a,t2.b \* from t1,t2 and t1.a=t2.a and t1.b=123 and t2.c= 4 ，如果t1.c与t2.c字段相同，那么t1上的索引(b,c)就只用到b了。此时如果把where条件中的t2.c=4改成t1.c=4，那么可以用到完整的索引
  2. 这种情况可能会在字段冗余设计（反范式）时出现
  3. 正确选取inner join和left join

### 使用union all，少使用union，注意考虑去重

* 1. union all不去重，而少了排序操作，速度相对比union要快，如果没有去重的需求，优先使用union all
  2. 如果UNION结果中有使用limit，在2个子SQL可能有许多返回值的情况下，各自加上limit。如果还有order by，请找DBA。
  3. 在明显不会有重复值时使用UNION ALL而不是UNION。

### IN的内容尽量不超过500

超过500个值使用批量的方式，否则一次执行会影响数据库的并发能力，因为单SQL只能且一直占用单CPU，而且可能导致主从复制延迟。

### 禁止使用is null, is not null这样的比较

使用 ISNULL() 来判断是否为 NULL 值。

### 单个事务，处理的行数不要超过1000 row/s

### 禁止order by .. limit

这种查询更多的是通过索引去优化，但order by的字段有讲究，比如主键id与f\_time都是顺序递增，那就可以考虑order by id而非 f\_time。

### count计数

* 1. 首先count(\**)、count(1)、count(col1)是有区别的，count(*\*)表示整个结果集有多少条记录，count(1)表示结果集里以primary key统计数量，绝大多数情况下count(*)与count(1)效果一样的，但count(col1)表示的是结果集里 col1 列 NOT null 的记录数。优先采用count(*\*)
  2. 大数据量count是消耗资源的操作，甚至会拖慢整个库，查询性能问题无法解决的，应从产品设计上进行重构。例如当频繁需要count的查询，考虑使用汇总表
  3. 遇到distinct的情况，group by方式可能效率更高。
  4. 尽量避免使用count(\*)操作，如果可以，计数类统计在redis或离线解决。

### 禁止对enum类型数据进行order by操作

### 禁止全局扫描

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。

### 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;

### 线上禁止使用的SQL语句

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。

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。

禁用insert into …on duplicate key update…在高并发环境下，会造成主从不一致。

禁止联表更新语句，如update t1,t2 where t1.id=t2.id…。

## 开发规范

### 应用长连接必须具有重连机制，但避免每执行一个sql检查一次db可用性；

### 应用使用长连接具有连接的timeout检查机制，及时回收长时间没有使用的连接，timeout建议为20min；

### SQL采用preparedStatement技术，提高预编译性能且降低SQL注入风险；

### 禁止一些框架或定制化的底层类使用set autocommit=0; set autocommit=1;等控制事务，应由程序把控使用begin;和commit;来实现事务；

### xml 配置文件约定

#{}，#param# 不要使用，${} 此种方式容易出现 SQL 注入。

### 不允许直接拿HashMap与HashTable作为查询结果集的输出

### 程序层DAO设计建议

建议手写SQL，如果用工具生成，需要仔细检查SQL，避免工具自动生成的语句出问题。

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。

## 权限规范

### 应用账号的权限按最小化原则分配增删改查权限，额外权限需要找DBA申请

### 应用代码禁止DDL操作，如需要需与DBA协商同意后方可使用；