MySQL数据库开发规范

**1. 命名规范**

1. **库名、表名、字段名必须使用小写字母，并采用下划线分割**
   * 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。
   * 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。
   * 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。
   * 为了统一规范， 库名、表名、字段名使用小写字母，不允许 - 号。
2. **库名以 d\_ 开头，表名以 t\_ 开头，字段名以 f\_ 开头**
   * 比如表 t\_crm\_relation，中间的 crm 代表业务模块名
   * 库名，如果不是分库，两个不同db实例里面的db名，不能相同，以免混淆
   * 视图以view\_开头，事件以event\_开头，触发器以trig\_开头，存储过程以proc\_开头，函数以func\_开头
   * 普通索引以idx\_col1\_col2命名，唯一索引以uk\_col1\_col2命名（可去掉f\_公共部分）。如 idx\_companyid\_corpid\_contacttime(f\_company\_id,f\_corp\_id,f\_contact\_time)
   * 如果某些特殊情况需要在sql里面指定索引，select \* from t\_test using index(idx\_i\_abc)，这种所以如果可以，命名的时候加上 i 分隔，如idx\_i\_corpid, uk\_i\_user，方便DBA在修改索引的时候会注意到这个 i 标识，不能随意修改这个索引(名称)，否则查询会出错。当然这种情况尽量不要出现。
3. **库名、表名、字段名禁止超过32个字符，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，禁止超过32个字符

1. **临时用的库、表名须以tmp位前缀，日期为后缀**

如 tmp\_t\_crm\_relation\_0425。备份表也类似，形如 bak\_t\_xxxx\_20160425 ，这样便于查找和知道有效期。  
正常业务里用的临时表、中间表，后缀尽量不要包含 tmp 命名，以免造成歧义。

1. **按日期时间分表须符合\_YYYY[MM][DD]格式**

这也是为将来有可能分表做准备的，比如t\_crm\_ec\_record\_201403，但像 t\_crm\_contact\_at201506就打破了这种规范。  
不具有时间特性的，直接以 t\_tbname\_001 这样的方式命名。

**2. 库表基础规范**

1. **使用Innodb存储引擎**

5.5版本开始mysql默认存储引擎就是InnoDB，5.7版本开始，系统表都放弃MyISAM了。

1. **表字符集统一使用UTF8MB4**
   * UTF8字符集存储汉字占用3个字节，存储英文字符占用一个字节
   * 校对字符集使用默认的 utf8mb4\_general\_ci。特别对于使用GUI设计表结构时，要检查它生成的sql定义
   * 连接的客户端也使用utf8，建立连接时指定charset或SET NAMES UTF8;。（对于已经在项目中长期使用latin1的，救不了了）
   * 如果遇到EMOJ等表情符号的存储需求，可申请使用UTF8MB4字符集
2. **所有表都要添加注释**
   * 尽量给字段也添加注释
   * 类status型需指明主要值的含义，如”0-离线，1-在线”
3. **控制单表字段数量**
   * 单表字段数上限30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。
4. **所有表都必须要显式指定主键**
   * 主键尽量采用自增方式，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。
   * 只有需要全局唯一主键时，使用外部自增id服务
   * 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
   * 少数情况可以使用联合唯一主键，需与DBA协商
   * 对于主键字段值是从其它地方插入（非自己使用AUTO\_INCREMENT生产），去掉auto\_increment定义。比如一些31天表、历史月份表上，不要auto\_increment属性；再必须全局id服务获取的主键。
5. **不强制使用外键参考**

即使2个表的字段有明确的外键参考关系，也不使用 FOREIGN KEY ，因为新纪录会去主键表做校验，影响性能。

1. **适度使用存储过程、视图，禁止使用触发器、事件**

* 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。
  + 使用视图一定程度上也是为了降低代码里SQL的复杂度，但有时候为了视图的通用性会损失性能（比如返回不必要的字段）。
* 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。
* 事件（event）也是一种偷懒的表现，目前已经遇到数次由于定时任务执行失败影响业务的情况，而且mysql无法对它做失败预警。建立专门的 job scheduler 平台。

1. **单表数据量控制在5000w以内**

表字段数量不要超过20个，如果有需要建立主副表，主键一一关联，避免单行数据过多以及修改记录binlog ROW模式导致文件过大。  
特别对于有一个text/blob或很大长度的varchar字段时，更应考虑单独存储。但也要注意查询条件尽量放在一个表上。

1. **数据库中不允许存储明文密码**

**3. 字段规范**

1. **char、varchar、text等字符串类型定义**
   * 对于长度基本固定的列，如果该列恰好更新又特别频繁，适合char。 utf8mb4字符集下，尽量使用varchar
   * varchar虽然存储变长字符串，但不可太小也不可太大。UTF8最多能存21844个汉字，或65532个英文
   * varbinary(M)保存的是二进制字符串，它保存的是字节而不是字符，所以没有字符集的概念，M长度0-255（字节）。只用于排序或比较时大小写敏感的类型，不包括密码存储
   * TEXT类型与VARCHAR都类似，存储可变长度，最大限制也是2^16，但是它20bytes以后的内容是在数据页以外的空间存储（row\_format=dynamic），对它的使用需要多一次寻址，没有默认值。  
     一般用于存放容量平均都很大、操作没有其它字段那样频繁的值。  
     网上部分文章说要避免使用text和blob，要知道如果纯用varchar可能会导致行溢出，效果差不多，但因为每行占用字节数过多，会导致buffer\_pool能缓存的数据行、页下降。另外text和blob上面一般不会去建索引，而是利用sphinx之类的第三方全文搜索引擎，如果确实要创建（前缀）索引，那就会影响性能。凡事看具体场景。  
     另外尽可能把text/blob拆到另一个表中
   * BLOB可以看出varbinary的扩展版本，内容以二进制字符串存储，无字符集，区分大小写，有一种经常提但不用的场景：不要在数据库里存储图片。
2. **int、tinyint、decimal等数字类型定义**
   * 使用tinyint来代替 enum和boolean  
     ENUM类型在需要修改或增加枚举值时，需要在线DDL，成本较高；ENUM列值如果含有数字类型，可能会引起默认值混淆  
     tinyint使用1个字节，一般用于status,type,flag的列
   * 建议使用 UNSIGNED 存储非负数值  
     相比不使用 unsigned，可以扩大一倍使用数值范围
   * int使用固定4个字节存储，int(11)与int(4)只是显示宽度的区别。但是定义是bigint(20), int(11)，不要随便改动这个显示宽度，c++里面需要这个长度去截取字段
   * 使用Decimal 代替float/double存储精确浮点数  
     对于货币、金额这样的类型，使用decimal，如 decimal(9,2)。float默认只能能精确到6位有效数字
3. **timestamp与datetime选择**
   * datetime 和 timestamp类型所占的存储空间不同，前者5个字节(5.5是8字节)，后者4个字节，这样造成的后果是两者能表示的时间范围不同。前者范围为1000-01-01 00:00:00 ~ 9999-12-31 23:59:59，后者范围为 1970-01-01 08:00:01 到 2038-01-19 11:14:07 。所以 TIMESTAMP 支持的范围比 DATATIME 要小。
   * timestamp可以在insert/update行时，自动更新时间字段（如 f\_set\_time timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP），但一个表只能有一个这样的定义。
   * timestamp显示与时区有关，内部总是以 UTC 毫秒 来存的。还受到严格模式的限制
   * 优先使用timestamp，datetime也没问题
   * 默认时间，要么current\_timestamp，要么’1970-01-02 01:01:01’，不要设置为’’或0
   * where条件里不要对时间列上使用时间函数
   * 如果使用int的型存储时间戳，约定统一使用 int unsigned default 0
4. **建议字段都定义为NOT NULL**
   * 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
   * 如果不能保证insert时一定有值过来，定义时使用default ‘’ ，或 0
5. **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

**4. 索引规范**

1. **任何新的select,update,delete上线，都要先explain，看索引使用情况**

尽量避免extra列出现：Using File Sort，Using Temporary，rows超过1000的要谨慎上线。  
explain解读

* + type：ALL, index, range, ref, eq\_ref, const, system, NULL（从左到右，性能从差到好）
  + possible\_keys：指出MySQL能使用哪个索引在表中找到记录，查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询使用
  + key：表示MySQL实际决定使用的键（索引）  
    如果没有选择索引，键是NULL。要想强制MySQL使用或忽视possible\_keys列中的索引，在查询中使用FORCE INDEX、USE INDEX或者IGNORE INDEX
  + ref：表示选择 key 列上的索引，哪些列或常量被用于查找索引列上的值
  + rows：根据表统计信息及索引选用情况，估算的找到所需的记录所需要读取的行数
  + Extra
    - Using temporary：表示MySQL需要使用临时表来存储结果集，常见于排序和分组查询
    - Using filesort：MySQL中无法利用索引完成的排序操作称为“文件排序”

1. **索引个数限制**
   * 索引是双刃剑，会增加维护负担，增大IO压力，索引占用空间是成倍增加的
   * 单张表的索引数量控制在5个以内，或不超过表字段个数的20%。若单张表多个字段在查询需求上都要单独用到索引，需要经过DBA评估。
2. **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
3. **没有特殊要求，使用自增id作为主键**
   * 主键是一种聚集索引，顺序写入。组合唯一索引作为主键的话，是随机写入，适合写少读多的表
   * 主键不允许更新
4. **索引尽量建在选择性高的列上**
   * 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。
   * 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
   * 走索引扫描行数超过30%时，改全表扫描
5. **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\_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
   * 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
6. **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
7. **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。

1. **尽量不要在频繁更新的列上创建索引**

如不在定义了 ON UPDATE CURRENT\_STAMP 的列上创建索引，维护成本太高（好在mysql有insert buffer，会合并索引的插入）

1. 修改表结构时 drop colum 时要注意，与这个字段相关的索引都会改变，变化是从原索引抽掉该字段定义。这种情况有可能导致部分索引重复或失效。

**5. SQL设计**

1. **杜绝直接 SELECT \* 读取全部字段**

即使需要所有字段，减少网络带宽消耗，能有效利用覆盖索引，表结构变更对程序基本无影响

1. **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。

1. **小心隐式类型转换**
   * 转换规则

a. 两个参数至少有一个是 NULL 时，比较的结果也是 NULL，例外是使用 <=> 对两个 NULL 做比较时会返回 1，这两种情况都不需要做类型转换  
b. 两个参数都是字符串，会按照字符串来比较，不做类型转换  
c. 两个参数都是整数，按照整数来比较，不做类型转换  
d. 十六进制的值和非数字做比较时，会被当做二进制串  
e. 有一个参数是 TIMESTAMP 或 DATETIME，并且另外一个参数是常量，常量会被转换为 timestamp  
f. 有一个参数是 decimal 类型，如果另外一个参数是 decimal 或者整数，会将整数转换为 decimal 后进行比较，如果另外一个参数是浮点数，则会把 decimal 转换为浮点数进行比较  
g. 所有其他情况下，两个参数都会被转换为浮点数再进行比较。

* + 如果一个索引建立在string类型上，如果这个字段和一个int类型的值比较，符合第 g 条。如f\_phone定义的类型是varchar，但where使用f\_phone in (098890)，两个参数都会被当成成浮点型。发生这个隐式转换并不是最糟的，最糟的是string转换后的float，mysql无法使用索引，这才导致了性能问题。如果是 f\_user\_id = ‘1234567’ 的情况，符合第 b 条,直接把数字当字符串比较。

1. **禁止在where条件列上使用函数**
   * 会导致索引失效，如lower(email)，f\_qq % 4。可放到右边的常量上计算
   * 返回小结果集不是很大的情况下，可以对返回列使用函数，简化程序开发
2. **使用like模糊匹配，%不要放首位**

会导致索引失效，有这种搜索需求是，考虑其它方案，如sphinx全文搜索

1. **涉及到复杂sql时，务必先参考已有索引设计，先explain**
   * 简单SQL拆分，不以代码处理复杂为由。
   * 比如 OR 条件： f\_phone=’10000’ or f\_mobile=’10000’，两个字段各自有索引，但只能用到其中一个。可以拆分成2个sql，或者union all。
   * 先explain的好处是可以为了利用索引，增加更多查询限制条件
2. **使用join时，where条件尽量使用充分利用同一表上的索引**
   * 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
   * 这种情况可能会在字段冗余设计（反范式）时出现
   * 正确选取inner join和left join。不允许滥用left join
3. **少用子查询，改用join**

小于5.6版本时，子查询效率很低，不像Oracle那样先计算子查询后外层查询。5.6版本开始得到优化

1. **考虑使用union all，少使用union，注意考虑去重**
   * 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
   * 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。
2. **IN的内容尽量不超过200个**

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。

1. **拒绝大事务**

比如在一个事务里进行多个select，多个update，如果是高频事务，会严重影响MySQL并发能力，因为事务持有的锁等资源只在事务rollback/commit时才能释放。但同时也要权衡数据写入的一致性。  
不要再事务里面做除数据库以外的操作。

1. **避免使用is null, is not null这样的比较**
2. **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 。

1. **c1 < a order by c2**

与上面不同的是，order by之前有个范围查询，由前面的内容可知，用不到类似(c1,c2)的索引，但是可以利用(c2,c1)索引。另外还可以改写成join的方式实现。

1. **分页优化**

建议使用合理的分页方式以提高分页效率，大页情况下不使用跳跃式分页  
假如有类似下面分页语句:  
SELECT *FROM table1 ORDER BY ftime DESC LIMIT 10000,10;  
这种分页方式会导致大量的io，因为MySQL使用的是提前读取策略。  
推荐分页方式：  
SELECT*FROM table1 WHERE ftime < last\_time ORDER BY ftime DESC LIMIT 10  
即传入上一次分页的界值

SELECT \* FROM table as t1 inner JOIN (SELECT id FROM table ORDER BY time LIMIT 10000，10) as t2 ON t1.id=t2.id

1. **count计数**
   * 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(*)
   * 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
   * 遇到distinct的情况，group by方式可能效率更高。
2. **delete,update语句改成select再explain**

select最多导致数据库慢，写操作才是锁表的罪魁祸首

1. **减少与数据库交互的次数，尽量采用批量SQL语句**
   * INSERT ... ON DUPLICATE KEY UPDATE ...，插入行后会导致在一个UNIQUE索引或PRIMARY KEY中出现重复值，则执行旧行UPDATE，如果不重复则直接插入，影响1行。
   * REPLACE INTO类似，但它是冲突时删除旧行。INSERT IGNORE相反，保留旧行，丢弃要插入的新行。
   * INSERT INTO VALUES(),(),()，合并插入。
2. **杜绝危险SQL**
   * 去掉where 1=1 这样无意义或恒真的条件，如果遇到update/delete或遭到sql注入就恐怖了
   * SQL中不允许出现DDL语句。一般也不给予create/alter这类权限，但阿里云RDS只区分读写用户
3. 是否应该 order by 主键  
   许多排序的场景，如果主键id是增长的，如果 order by f\_create\_time 查询慢，有可能使用了filesort，此时最简单的办法是看能否换成 order by id，因为id作为主键是递增的，并且附带在了每个二级索引后面。  
   但是也要谨慎使用 order by id，特别是在explain结果看到filesort的情况下，优化器极有可能放弃这个filesort，而选择了它所认为更高效的扫描方式，实则更慢。
4. 使用正确的表  
   比如要统计昨天的数据这类业务较多，是否可以设计一个昨天表，不在31天表上统计，在月份表上统计也行。  
   或者其它组已经有“半统计”的数据，从他们那抽取数据，而不是在原始数据上统计

**6. 行为规范**

* 不允许在DBA不知情的情况下导现网数据
* 大批量更新，如修复数据，避开高峰期，并通知DBA。直接执行sql的由运维或DBA同事操作
* 及时处理已下线业务的SQL
* 复杂sql上线审核  
  因为目前还没有SQL审查机制，复杂sql如多表join,count,group by，主动上报DBA评估。
* 重要项目的数据库方案选型和设计必须提前通知DBA参与

**7. DBA运维规范**

这部分可另起一篇规范了，这里先开个头。

* 安装初始化MySQL，严格安装约定的目录、配置文件、权限、参数来初始化
* 第一时间添加监控、备份
* 定期巡检
* 对先上过期废弃的表，及时与开发确认归档清除，同时也要注意与它相关的视图、过程、事件等。

**本文参考**

* [互联网MySQL开发规范](http://wangwei007.blog.51cto.com/68019/1709769) 这个基本也是《去哪儿MySQL开发规范.pdf》版本
* MySQL数据库开发的三十六条军规\_石展\_完整.pdf
* [老叶观点：MySQL开发规范之我见](http://imysql.com/2015/07/23/something-important-about-mysql-design-reference.shtml)
* [MySQL开发规范与使用技巧总结](http://blog.csdn.net/xlgen157387/article/details/48086607)
* <http://highdb.com/mysql%E5%BC%80%E5%8F%91%E8%A7%84%E8%8C%83/>

本文链接地址：<http://seanlook.com/2016/05/11/mysql-dev-principle-ec/>