新整理，MySQL 开发规范

作者：c3gen
原文链接：https://ld246.com/article/1501228017914
来源网站：链滴

最近在对公司数据库进行分库和重新架构，，现整理一份规范，仅供一起学习。

命名规范

基本命名原则

- 使用有意义的英文词汇，词汇中间以下划线分隔。（不要用拼音）
- 只能使用英文字母，数字，下划线，并以英文字母开头。
- 库、表、字段全部采用小写，不要使用驼峰式命名。
- 避免用ORACLE、MySQL的保留字，如desc，关键字如index。
- 命名禁止超过32个字符，须见名之意，建议使用名词不是动词
- 数据库，数据表一律使用前缀
  - 临时库、表名必须以tmp为前缀，并以日期为后缀
  - 备份库、表必须以bak为前缀，并以日期为后缀

为什么库、表、字段全部采用小写?

在 MySQL 中，数据库和表对就于那些目录下的目录和文件。因而，操作系统的敏感性决定数据库和命名的大小写敏感。

- Windows下是不区分大小写的。
- Linux下大小写规则：
  - 数据库名与表名是严格区分大小写的；
  - 表的别名是严格区分大小写的；
  - 列名与列的别名在所有的情况下均是忽略大小写的；
  - 变量名也是严格区分大小写的；

如果已经设置了驼峰式的命名如何解决？需要在MySQL的配置文件my.ini中增加 lower_case_table_nmes = 1即可。

表命名

- 同一个模块的表尽可能使用相同的前缀，表名称尽可能表达含义。所有日志表均以 log_ 开头

字段命名

- 表达其实际含义的英文单词或简写。布尔意义的字段以“is_”作为前缀，后接动词过去分词。
- 各表之间相同意义的字段应同名。各表之间相同意义的字段，以去掉模块前缀的表名_字段名命名。
- 外键字段用表名_字段名表示其关联关系。
- 表的主键一般都约定成为id，自增类型，是别的表的外键均使用xxx_id的方式来表明。

索引命名

- 非唯一索引必须按照“idx_字段名称_字段名称[_字段名]”进行命名

- 唯一索引必须按照“uniq_字段名称_字段名称[_字段名]”进行命名

约束命名

- 主键约束：pk_表名称。
- 唯一约束：uk_表名称_字段名。（应用中需要同时有唯一性检查逻辑。）

触发器命名

- trg_表名_操作。

函数过程命名

- 采用动词+名词的形式表达其含义。

序列命名

- seq_表名

表设计规范

1、表引擎取决于实际应用场景；日志及报表类表建议用myisam，与交易，审核，金额相关的表建议innodb引擎。如无说明，建表时一律采用innodb引擎。

2、默认使用utf8mb4字符集，数据库排序规则使用utf8mb4_general_ci，（由于数据库定义使用了认，数据表可以不再定义，但为保险起见，建议都写上）。

为什么字符集不选择utf8，排序规则不使用utf8_general_ci?

采用utf8编码的MySQL无法保存占位是4个字节的Emoji表情。为了使后端的项目，全面支持客户端入的Emoji表情，升级编码为utf8mb4是最佳解决方案。对于JDBC连接串设置了characterEncoding utf8或者做了上述配置仍旧无法正常插入emoji数据的情况，需要在代码中指定连接的字符集为utf8m4。

3、所有表、字段均应用 comment 列属性来描述此表、字段所代表的真正含义，如枚举值则建议将字段中使用的内容都定义出来。

4、如无说明，表中的第一个id字段一定是主键且为自动增长，禁止在非事务内作为上下文作为条件行数据传递。禁止使用varchar类型作为主键语句设计。

5、如无说明，表必须包含create_time和modify_time字段，即表必须包含记录创建时间和修改时间字段

6、如无说明，表必须包含is_del，用来标示数据是否被删除，原则上数据库数据不允许物理删除。

7、用尽量少的存储空间来存数一个字段的数据

- 能用int的就不用char或者varchar
- 能用tinyint的就不用int
- 使用UNSIGNED存储非负数值。
- 不建议使用ENUM、SET类型，使用TINYINT来代替

- 使用短数据类型，比如取值范围为0-80时，使用TINYINT UNSIGNED
- 存储精确浮点数必须使用DECIMAL替代FLOAT和DOUBLE
- 时间字段，除特殊情况一律采用int来记录unix_timestamp
  - 存储年使用YEAR类型。
  - 存储日期使用DATE类型。
  - 存储时间（精确到秒）建议使用TIMESTAMP类型，因为TIMESTAMP使用4字节，DATETIME用8个字节。
- 建议使用INT UNSIGNED存储IPV4。
- 尽可能不使用TEXT、BLOB类型
- 禁止在数据库中使用VARBINARY、BLOB存储图片、文件等。建议使用其他方式存储（TFS/SFS）MySQL只保存指针信息。
- 单条记录大小禁止超过8k（列长度(中文)*3(UTF8)+列长度(英文)*1）

datetime与timestamp有什么不同?

相同点：TIMESTAMP列的显示格式与DATETIME列相同。显示宽度固定在19字符，并且格式为YYYY MM-DD HH:MM:SS。

不同点：

- TIMESTAMP
  - 4个字节储存，时间范围：1970-01-01 08:00:01 ~ 2038-01-19 11:14:07
  - 值以UTC格式保存，涉及时区转化 ，存储时对当前的时区进行转换，检索时再转换回当前的区。
- datetime
  - 8个字节储存，时间范围：1000-01-01 00:00:00 ~ 9999-12-31 23:59:59
  - 实际格式储存，与时区无关

如何使用TIMESTAMP的自动赋值属性?

- 将当前时间作为ts的默认值：ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP。
- 当行更新时，更新ts的值：ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP。
- 可以将1和2结合起来：ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRNT_TIMESTAMP。

如何使用INT UNSIGNED存储ip?

使用INT UNSIGNED而不是char(15)来存储ipv4地址，通过MySQL函数inet_ntoa和inet_aton来进转化。Ipv6地址目前没有转化函数，需要使用DECIMAL或者两个bigINT来存储。

8、如无备注，所有字段都设置NOT NULL，并设置默认值；

9、禁止在数据库中存储明文密码

10、如无备注，所有的布尔值字段，如is_hot、is_deleted，都必须设置一个默认值，并设为0；

11、如无备注，排序字段order_id在程序中默认使用降序排列；

12、整形定义中不添加长度，比如使用INT，而不是INT[4]

INT[M]**，M值代表什么含义?**

注意数值类型括号后面的数字只是表示宽度而跟存储范围没有关系。很多人他们认为INT(4)和INT(10其取值范围分别是 (-9999到9999)和(-9999999999到9999999999)，这种理解是错误的。其实对整中的 M值与 ZEROFILL 属性结合使用时可以实现列值等宽。不管INT[M]中M值是多少，其取值范围是 (-2147483648到2147483647 有符号时)，(0到4294967295无符号时)。

显示宽度并不限制可以在列内保存的值的范围，也不限制超过列的指定宽度的值的显示。当结合可选展属性ZEROFILL使用时默认补充的空格用零代替。例如：对于声明为INT(5) ZEROFILL的列，值4检为00004。请注意如果在整数列保存超过显示宽度的一个值，当MySQL为复杂联接生成临时表时会遇问题，因为在这些情况下MySQL相信数据适合原列宽度，如果为一个数值列指定ZEROFILL, MySQL动为该列添加UNSIGNED属性。

13、使用VARBINARY存储大小写敏感的变长字符串

什么时候用CHAR，什么时候用VARCHAR?

CHAR和VARCHAR类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保等方面也不同。CHAR和VARCHAR类型声明的长度表示你想要保存的最大字符数。例如，CHAR(30)以占用30个字符。

- CHAR列的长度固定为创建表时声明的长度。长度可以为从0到255的任何值。当保存CHAR值时在它们的右边填充空格以达到指定的长度。当检索到CHAR值时，尾部的空格被删除掉。在存储或检过程中不进行大小写转换。
- VARCHAR列中的值为可变长字符串。长度可以指定为0到65,535之间的值。(VARCHAR的最大效长度由最大行大小和使用的字符集确定。整体最大长度是65,532字节）。

同CHAR对比，VARCHAR值保存时只保存需要的字符数，另加一个字节来记录长度(如果列声明的长超过255，则使用两个字节)。VARCHAR值保存时不进行填充。当值保存和检索时尾部的空格仍保留符合标准SQL。

char适合存储用户密码的MD5哈希值，它的长度总是一样的。对于经常改变的值，char也好于varchar因为固定长度的行不容易产生碎片，对于很短的列，char的效率也高于varchar。char(1)字符串对于字节字符集只会占用一个字节，但是varchar(1)则会占用2个字节，因为1个字节用来存储长度信息。

索引设计规范

MySQL的查询速度依赖良好的索引设计，因此索引对于高性能至关重要。合理的索引会加快查询速（包括UPDATE和DELETE的速度，MySQL会将包含该行的page加载到内存中，然后进行UPDATE或DELETE操作），不合理的索引会降低速度。MySQL索引查找类似于新华字典的拼音和部首查找，当音和部首索引不存在时，只能通过一页一页的翻页来查找。当MySQL查询不能使用索引时，MySQL进行全表扫描，会消耗大量的IO。索引的用途：去重、加速定位、避免排序、覆盖索引。

什么是覆盖索引?

InnoDB存储引擎中，secondary index（非主键索引）中没有直接存储行地址，存储主键值。如果用需要查询secondary index中所不包含的数据列时，需要先通过secondary index查找到主键值，然再通过主键查询到其他数据列，因此需要查询两次。覆盖索引的概念就是查询可以通过在一个索引中成，覆盖索引效率会比较高，主键查询是天然的覆盖索引。合理的创建索引以及合理的使用查询语句当使用到覆盖索引时可以获得性能提升。比如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。

索引的基本规范

1、索引数量控制，单张表中索引数量不超过5个，单个索引中的字段数不超过5个。

- 综合评估数据密度和分布
- 考虑查询和更新比例

为什么一张表中不能存在过多的索引?

InnoDB的secondary index使用b+tree来存储，因此在UPDATE、DELETE、INSERT的时候需要对b+ree进行调整，过多的索引会减慢更新的速度。

2、对字符串使用前缀索引，前缀索引长度不超过8个字符，建议优先考虑前缀索引，必要时可添加伪并建立索引。

- 不要索引blob/text等字段,不要索引大型字段,这样做会让索引占用太多的存储空间

什么是前缀索引?

前缀索引说白了就是对文本的前几个字符（具体是几个字符在建立索引时指定）建立索引，这样建立来的索引更小，所以查询更快。 前缀索引能有效减小索引文件的大小，提高索引的速度。但是前缀索也有它的坏处：MySQL 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引(Covering Index)。

建立前缀索引的语法：ALTER TABLE table_name ADD KEY(column_name(prefix_length));

3、主键准则

- 表必须有主键
- 不使用更新频繁的列
- 尽量不选择字符串列
- 不使用UUID MD5 HASH
- 默认使用非空的唯一键
- 建议选择自增或发号器

4、 重要的SQL必须被索引，核心SQL优先考虑覆盖索索引

- UPDATE、DELETE语句的WHERE条件列
- ORDER BY、GROUP BY、DISTINCT的字段
- 多表JOIN的字段

5、区分度最大的字段放在前面

- 选择筛选性更优的字段放在最前面，比如单号、userid等，type，status等筛选性一般不建议放在前面
- 索引根据左前缀原则，当建立一个联合索引(a,b,c)，则查询条件里面只有包含(a)或(a,b)或(a,b,c)的候才能走索引,(a,c)作为条件的时候只能使用到a列索引,所以这个时候要确定a的返回列一定不能太多不然语句设计就不合理,(b,c)则不能走索引

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

6、索引禁忌

- 不在低基数列上建立索引，例如“性别”
- 不在索引列进行数学运算和函数运算
- 不要索引常用的小型表

7、 尽量不使用外键

- 外键用来保护参照完整性，可在业务端实现
- 对父表和子表的操作会相互影响，降低可用性
- INNODB本身对online DDL的限制

MYSQL 中索引的限制

- MYISAM 存储引擎索引长度的总和不能超过 1000 字节
- BLOB 和 TEXT 类型的列只能创建前缀索引
- MYSQL 目前不支持函数索引
- 使用不等于 (!= 或者 <>) 的时候, MYSQL 无法使用索引。
- 过滤字段使用函数运算 (如 abs (column)) 后, MYSQL无法使用索引。
- join语句中join条件字段类型不一致的时候MYSQL无法使用索引
- 使用 LIKE 操作的时候如果条件以通配符开始 (如 ‘%abc...’)时, MYSQL无法使用索引。
- 使用非等值查询的时候, MYSQL 无法使用 Hash 索引。

语句设计规范

1、使用预编译语句

- 只传参数，比传递SQL语句更高效
- 一次解析，多次使用
- 降低SQL注入概率

2、避免隐式转换

- 会导致索引失效

3、充分利用前缀索引

- 必须是最左前缀
- 不可能同时用到两个范围条件
- 不使用%前导的查询，如like “%ab”

4、不使用负向查询，如not in/like

- 无法使用索引，导致全表扫描

- 全表扫描导致buffer pool利用率降低

5、避免使用存储过程、触发器、UDF、events等

- 让数据库做最擅长的事
- 降低业务耦合度，为sacle out、sharding留有余地
- 避开BUG

6、避免使用大表的JOIN

- MySQL最擅长的是单表的主键/二级索引查询
- JOIN消耗较多内存，产生临时表

7、避免在数据库中进行数学运算

- MySQL不擅长数学运算和逻辑判断
- 无法使用索引

7、减少与数据库的交互次数

- INSERT ... ON DUPLICATE KEY UPDATE
- REPLACE INTO、INSERT IGNORE 、INSERT INTO VALUES(),(),()
- UPDATE ... WHERE ID IN(10,20,50,...)

8、合理的使用分页

- 限制分页展示的页数
- 只能点击上一页、下一页
- 采用延迟关联

如何正确的使用分页?

mysql中分页查询有两种方式, 一种是使用COUNT(*)的方式,具体代码如下

```
SELECT COUNT(*) FROM foo WHERE b = 1;

SELECT a FROM foo WHERE b = 1 LIMIT 100,10;
```

另外一种是使用SQL_CALC_FOUND_ROWS

```
SELECT SQL_CALC_FOUND_ROWS a FROM foo WHERE b = 1 LIMIT 100, 10;

SELECT FOUND_ROWS();
```

第二种方式调用SQL_CALC_FOUND_ROWS之后会将WHERE语句查询的行数放在FOUND_ROWS()中，第二次只需要查询FOUND_ROWS()就可以查出有多少行了。

两种方式的比较?

首先原子性讲，第二种肯定比第一种好。第二种能保证查询语句的原子性，第一种当两个请求之间有外的操作修改了表的时候，结果就自然是不准确的了。而第二种则不会。但是非常可惜，一般页面需

进行分页显示的时候，往往并不要求分页的结果非常准确。即分页返回的total总数大1或者小1都是所谓的。所以其实原子性不是我们分页关注的重点。

下面看效率。这个非常重要，分页操作在每个网站上的使用都是非常大的，查询量自然也很大。由于论哪种，分页操作必然会有两次sql查询，于是就有很多很多关于两种查询性能的比较：

SQL_CALC_FOUND_ROWS真的很慢么?

http://hi.baidu.com/thinkinginlamp/item/b122fdaea5ba23f614329b14

To SQL_CALC_FOUND_ROWS or not to SQL_CALC_FOUND_ROWS?

http://www.mysqlperformanceblog.com/2007/08/28/to-sql_calc_found_rows-or-not-to-sql_calc_found_rows/

老王这篇文章里面有提到一个covering index的概念，简单来说就是怎样才能只让查询根据索引返回果，而不进行表查询

具体看他的另外一篇文章：

MySQL之Covering Index

http://hi.baidu.com/thinkinginlamp/item/1b9aaf09014acce0f45ba6d3

所以我得出的结论是如果数据库是InnoDB的话，我还是倾向于使用SQL_CALC_FOUND_ROWS

结论：SQL_CALC_FOUND_ROWS和COUNT(*)的性能在都使用covering index的情况下前者高，在使用covering index情况下后者性能高。所以使用的时候要注意这个。

9、拒绝大SQL，拆分成小SQL

- 充分利用QUERY CACHE
- 充分利用多核CPU

10、使用in代替or，in的值不超过1000个

11、禁止使用order by rand()

12、使用EXPLAIN诊断，避免生成临时表

EXPLAIN语句（在MySQL客户端中执行）可以获得MySQL如何执行SELECT语句的信息。通过对SELET语句执行EXPLAIN，可以知晓MySQL执行该SELECT语句时是否使用了索引、全表扫描、临时表、序等信息。尽量避免MySQL进行全表扫描、使用临时表、排序等。详见官方文档。

13、用union all而不是union

union all与 union有什么区别?

union和union all关键字都是将两个结果集合并为一个，但这两者从使用和效率上来说都有所不同。

union在进行表链接后会筛选掉重复的记录，所以在表链接后会对所产生的结果集进行排序运算，删

重复的记录再返回结果。如：

select * from test_union1

union

select * from test_union2

这个SQL在运行时先取出两个表的结果，再用排序空间进行排序删除重复的记录，最后返回结果集，果表数据量大的话可能会导致用磁盘进行排序。

而union all只是简单的将两个结果合并后就返回。这样，如果返回的两个结果集中有重复的数据，那返回的结果集就会包含重复的数据了。

从效率上说，union all要比union快很多，所以，如果可以确认合并的两个结果集中不包含重复的数的话，那么就使用union all，如下：

select * from test_union1

union all

select * from test_union2

14、程序应有捕获SQL异常的处理机制

15、禁止单条SQL语句同时更新多个表

16、不使用select *，SELECT语句只获取需要的字段

- 消耗CPU和IO、消耗网络带宽
- 无法使用覆盖索引
- 减少表结构变更带来的影响
- 因为大，select/join 可能生成临时表

17、UPDATE、DELETE语句不使用LIMIT

18、INSERT语句必须显式的指明字段名称，不使用INSERT INTO table()

19、INSERT语句使用batch提交（INSERT INTO table VALUES(),(),()……），values的个数不超过500

20、统计表中记录数时使用COUNT(*)，而不是COUNT(primary_key)和COUNT(1) 备注：仅针对Myiam

21、数据更新建议使用二级索引先查询出主键，再根据主键进行数据更新

22、禁止使用跨库查询

23、禁止使用子查询，建议将子查询转换成关联查询

24、针对varchar类型字段的程序处理，请验证用户输入，不要超出其预设的长度；

分表规范

单表一到两年内数据量超过500w或数据容量超过10G考虑分表，需提前考虑历史数据迁移或应用自删除历史数据，采用等量均衡分表或根据业务规则分表均可。要分表的数据表必须与DBA商量分表策略

- 用HASH进行散表，表名后缀使用十进制数，下标从0开始
- 按日期时间分表需符合YYYY[MM][DD][HH]格式
- 采用合适的分库分表策略。例如千库十表、十库百表等
- 禁止使用分区表，分区表对分区键有严格要，分区表在表变大后执行DDL、SHARDING、单表恢复都变得更加困难。
- 拆分大字段和访问频率低的字段，分离冷热数据

行为规范

- 批量导入、导出数据必须提前通知DBA协助观察
- 禁止在线上从库执行后台管理和统计类查询
- 禁止有super权限的应用程序账号存在
- 产品出现非数据库导致的故障时及时通知DBA协助排查
- 推广活动或上线新功能必须提前通知DBA进行流量评估
- 数据库数据丢失，及时联系DBA进行恢复
- 对单表的多次alter操作必须合并为一次操作
- 不在MySQL数据库中存放业务逻辑
- 重大项目的数据库方案选型和设计必须提前通知DBA参与
- 对特别重要的库表，提前与DBA沟通确定维护和备份优先级
- 不在业务高峰期批量更新、查询数据库其他规范
- 提交线上建表改表需求，必须详细注明所有相关SQL语句

其他规范

日志类数据不建议存储在MySQL上，优先考虑Hbase或OceanBase，如需要存储请找DBA评估使用缩表存储。