目录

[一、 数据库命令规范 1](#_Toc40645777)

[**1.** **所有数据库对象名称必须使用小写字母并用下划线分割** 1](#_Toc40645778)

[**2.** **所有数据库对象名称禁止使用 MySQL 保留关键字（如果表名中包含关键字查询时，需要将其用单引号括起来）** 1](#_Toc40645779)

[**3.** **数据库对象的命名要能做到见名识意，并且最后不要超过 32 个字符** 1](#_Toc40645780)

[**4.** **临时库表必须以 tmp\_为前缀并以日期为后缀，备份表必须以 bak\_为前缀并以日期 (时间戳) 为后缀** 1](#_Toc40645781)

[**5.** **所有存储相同数据的列名和列类型必须一致（一般作为关联列，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）** 1](#_Toc40645782)

[二、 数据库基本设计规范 1](#_Toc40645783)

[**1. 所有表必须使用 Innodb 存储引擎** 1](#_Toc40645784)

[**2. 数据库和表的字符集统一使用 UTF8** 2](#_Toc40645785)

[**3. 所有表和字段都需要添加注释** 2](#_Toc40645786)

[**4. 尽量控制单表数据量的大小,建议控制在 500 万以内。** 2](#_Toc40645787)

[**5. 谨慎使用 MySQL 分区表** 2](#_Toc40645788)

[**6.尽量做到冷热数据分离,减小表的宽度** 2](#_Toc40645789)

[**7. 禁止在表中建立预留字段** 3](#_Toc40645790)

[**8. 禁止在数据库中存储图片,文件等大的二进制数据** 3](#_Toc40645791)

[**9. 禁止在线上做数据库压力测试** 3](#_Toc40645792)

[**10. 禁止从开发环境,测试环境直接连接生成环境数据库** 3](#_Toc40645793)

[三、 数据库字段设计规范 3](#_Toc40645794)

[**1. 优先选择符合存储需要的最小的数据类型** 3](#_Toc40645795)

[**2. 避免使用 TEXT,BLOB 数据类型，最常见的 TEXT 类型可以存储 64k 的数据** 4](#_Toc40645796)

[**3. 避免使用 ENUM 类型** 4](#_Toc40645797)

[**4. 尽可能把所有列定义为 NOT NULL** 5](#_Toc40645798)

[**5. 使用 TIMESTAMP(4 个字节) 或 DATETIME 类型 (8 个字节) 存储时间** 5](#_Toc40645799)

[**6. 同财务相关的金额类数据必须使用 decimal 类型** 5](#_Toc40645800)

[四、 索引设计规范 6](#_Toc40645801)

[**1. 限制每张表上的索引数量,建议单张表索引不超过 5 个** 6](#_Toc40645802)

[**2. 禁止给表中的每一列都建立单独的索引** 6](#_Toc40645803)

[**3. 每个 Innodb 表必须有个主键** 6](#_Toc40645804)

[**4. 常见索引列建议** 6](#_Toc40645805)

[**5.如何选择索引列的顺序** 7](#_Toc40645806)

[**6. 避免建立冗余索引和重复索引（增加了查询优化器生成执行计划的时间）** 7](#_Toc40645807)

[**7. 对于频繁的查询优先考虑使用覆盖索引** 7](#_Toc40645808)

[**8.索引 SET 规范** 8](#_Toc40645809)

[五、 数据库 SQL 开发规范 8](#_Toc40645810)

[**1. 建议使用预编译语句进行数据库操作** 8](#_Toc40645811)

[**2. 避免数据类型的隐式转换** 8](#_Toc40645812)

[**3. 充分利用表上已经存在的索引** 8](#_Toc40645813)

[**4. 数据库设计时，应该要对以后扩展进行考虑** 9](#_Toc40645814)

[**5. 程序连接不同的数据库使用不同的账号，进制跨库查询** 9](#_Toc40645815)

[**6. 禁止使用 SELECT \* 必须使用 SELECT <字段列表> 查询** 9](#_Toc40645816)

[**7. 禁止使用不含字段列表的 INSERT 语句** 9](#_Toc40645817)

[**8. 避免使用子查询，可以把子查询优化为 join 操作** 9](#_Toc40645818)

[**9. 避免使用 JOIN 关联太多的表** 10](#_Toc40645819)

[**10. 减少同数据库的交互次数** 10](#_Toc40645820)

[**11. 对应同一列进行 or 判断时，使用 in 代替 or** 10](#_Toc40645821)

[**12. 禁止使用 order by rand() 进行随机排序** 10](#_Toc40645822)

[**13. WHERE 从句中禁止对列进行函数转换和计算** 10](#_Toc40645823)

[**14. 在明显不会有重复值时使用 UNION ALL 而不是 UNION** 11](#_Toc40645824)

[**15. 拆分复杂的大 SQL 为多个小 SQL** 11](#_Toc40645825)

[六、 数据库操作行为规范 11](#_Toc40645826)

[**1. 超 100 万行的批量写 (UPDATE,DELETE,INSERT) 操作,要分批多次进行操作** 11](#_Toc40645827)

[**2. 对于大表使用 pt-online-schema-change 修改表结构** 12](#_Toc40645828)

[**3. 禁止为程序使用的账号赋予 super 权限** 12](#_Toc40645829)

[**4. 对于程序连接数据库账号,遵循权限最小原则** 12](#_Toc40645830)

# 数据库命令规范

1. **所有数据库对象名称必须使用小写字母并用下划线分割**
2. **所有数据库对象名称禁止使用 MySQL 保留关键字（如果表名中包含关键字查询时，需要将其用单引号括起来）**
3. **数据库对象的命名要能做到见名识意，并且最后不要超过 32 个字符**
4. **临时库表必须以 tmp\_为前缀并以日期为后缀，备份表必须以 bak\_为前缀并以日期 (时间戳) 为后缀**
5. **所有存储相同数据的列名和列类型必须一致（一般作为关联列，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）**

# 数据库基本设计规范

**1. 所有表必须使用 Innodb 存储引擎**

没有特殊要求（即 Innodb 无法满足的功能如：列存储，存储空间数据等）的情况下，所有表必须使用 Innodb 存储引擎（MySQL5.5 之前默认使用 Myisam，5.6 以后默认的为 Innodb）。

Innodb 支持事务，支持行级锁，更好的恢复性，高并发下性能更好。

**2. 数据库和表的字符集统一使用 UTF8**

兼容性更好，统一字符集可以避免由于字符集转换产生的乱码，不同的字符集进行比较前需要进行转换会造成索引失效，如果数据库中有存储 emoji 表情的需要，字符集需要采用 utf8mb4 字符集。

**3. 所有表和字段都需要添加注释**

使用 comment 从句添加表和列的备注，从一开始就进行数据字典的维护

**4. 尽量控制单表数据量的大小,建议控制在 500 万以内。**

500 万并不是 MySQL 数据库的限制，过大会造成修改表结构，备份，恢复都会有很大的问题。

可以用历史数据归档（应用于日志数据），分库分表（应用于业务数据）等手段来控制数据量大小

**5. 谨慎使用 MySQL 分区表**

分区表在物理上表现为多个文件，在逻辑上表现为一个表；

谨慎选择分区键，跨分区查询效率可能更低；

建议采用物理分表的方式管理大数据。

**6.尽量做到冷热数据分离,减小表的宽度**

MySQL 限制每个表最多存储 4096 列，并且每一行数据的大小不能超过 65535 字节。

减少磁盘 IO,保证热数据的内存缓存命中率（表越宽，把表装载进内存缓冲池时所占用的内存也就越大,也会消耗更多的 IO）；

更有效的利用缓存，避免读入无用的冷数据；

经常一起使用的列放到一个表中（避免更多的关联操作）。

**7. 禁止在表中建立预留字段**

预留字段的命名很难做到见名识义。

预留字段无法确认存储的数据类型，所以无法选择合适的类型。

对预留字段类型的修改，会对表进行锁定。

**8. 禁止在数据库中存储图片,文件等大的二进制数据**

通常文件很大，会短时间内造成数据量快速增长，数据库进行数据库读取时，通常会进行大量的随机 IO 操作，文件很大时，IO 操作很耗时。

通常存储于文件服务器，数据库只存储文件地址信息

**9. 禁止在线上做数据库压力测试**

**10. 禁止从开发环境,测试环境直接连接生成环境数据库**

# 数据库字段设计规范

**1. 优先选择符合存储需要的最小的数据类型**

**原因：**

列的字段越大，建立索引时所需要的空间也就越大，这样一页中所能存储的索引节点的数量也就越少也越少，在遍历时所需要的 IO 次数也就越多，索引的性能也就越差。

**方法：**

**a.将字符串转换成数字类型存储,如:将 IP 地址转换成整形数据**

MySQL 提供了两个方法来处理 ip 地址

•inet\_aton 把 ip 转为无符号整型 (4-8 位)•inet\_ntoa 把整型的 ip 转为地址

插入数据前，先用 inet\_aton 把 ip 地址转为整型，可以节省空间，显示数据时，使用 inet\_ntoa 把整型的 ip 地址转为地址显示即可。

**b.对于非负型的数据 (如自增 ID,整型 IP) 来说,要优先使用无符号整型来存储**

**原因：**

无符号相对于有符号可以多出一倍的存储空间

SIGNED INT -2147483648~2147483647UNSIGNED INT 0~4294967295

VARCHAR(N) 中的 N 代表的是字符数，而不是字节数，使用 UTF8 存储 255 个汉字 Varchar(255)=765 个字节。**过大的长度会消耗更多的内存。**

**2. 避免使用 TEXT,BLOB 数据类型，最常见的 TEXT 类型可以存储 64k 的数据**

**a. 建议把 BLOB 或是 TEXT 列分离到单独的扩展表中**

MySQL 内存临时表不支持 TEXT、BLOB 这样的大数据类型，如果查询中包含这样的数据，在排序等操作时，就不能使用内存临时表，必须使用磁盘临时表进行。而且对于这种数据，MySQL 还是要进行二次查询，会使 sql 性能变得很差，但是不是说一定不能使用这样的数据类型。

如果一定要使用，建议把 BLOB 或是 TEXT 列分离到单独的扩展表中，查询时一定不要使用 select \* 而只需要取出必要的列，不需要 TEXT 列的数据时不要对该列进行查询。

**2、TEXT 或 BLOB 类型只能使用前缀索引**

因为MySQL[1] 对索引字段长度是有限制的，所以 TEXT 类型只能使用前缀索引，并且 TEXT 列上是不能有默认值的

**3. 避免使用 ENUM 类型**

修改 ENUM 值需要使用 ALTER 语句

ENUM 类型的 ORDER BY 操作效率低，需要额外操作

禁止使用数值作为 ENUM 的枚举值

**4. 尽可能把所有列定义为 NOT NULL**

**原因：**

索引 NULL 列需要额外的空间来保存，所以要占用更多的空间

进行比较和计算时要对 NULL 值做特别的处理

**5. 使用 TIMESTAMP(4 个字节) 或 DATETIME 类型 (8 个字节) 存储时间**

TIMESTAMP 存储的时间范围 1970-01-01 00:00:01 ~ 2038-01-19-03:14:07

TIMESTAMP 占用 4 字节和 INT 相同，但比 INT 可读性高

超出 TIMESTAMP 取值范围的使用 DATETIME 类型存储

**经常会有人用字符串存储日期型的数据（不正确的做法）**

•缺点 1：无法用日期函数进行计算和比较•缺点 2：用字符串存储日期要占用更多的空间

**6. 同财务相关的金额类数据必须使用 decimal 类型**

•非精准浮点：float,double•精准浮点：decimal

Decimal 类型为精准浮点数，在计算时不会丢失精度

占用空间由定义的宽度决定，每 4 个字节可以存储 9 位数字，并且小数点要占用一个字节

可用于存储比 bigint 更大的整型数据

# 索引设计规范

**1. 限制每张表上的索引数量,建议单张表索引不超过 5 个**

索引并不是越多越好！索引可以提高效率同样可以降低效率。

索引可以增加查询效率，但同样也会降低插入和更新的效率，甚至有些情况下会降低查询效率。

因为 MySQL 优化器在选择如何优化查询时，会根据统一信息，对每一个可以用到的索引来进行评估，以生成出一个最好的执行计划，如果同时有很多个索引都可以用于查询，就会增加 MySQL 优化器生成执行计划的时间，同样会降低查询性能。

**2. 禁止给表中的每一列都建立单独的索引**

5.6 版本之前，一个 sql 只能使用到一个表中的一个索引，5.6 以后，虽然有了合并索引的优化方式，但是还是远远没有使用一个联合索引的查询方式好。

**3. 每个 Innodb 表必须有个主键**

Innodb 是一种索引组织表：数据的存储的逻辑顺序和索引的顺序是相同的。每个表都可以有多个索引，但是表的存储顺序只能有一种。

Innodb 是按照主键索引的顺序来组织表的

•不要使用更新频繁的列作为主键，不适用多列主键（相当于联合索引）•不要使用 UUID,MD5,HASH,字符串列作为主键（无法保证数据的顺序增长）•主键建议使用自增 ID 值

**4. 常见索引列建议**

•出现在 SELECT、UPDATE、DELETE 语句的 WHERE 从句中的列•包含在 ORDER BY、GROUP BY、DISTINCT 中的字段•并不要将符合 1 和 2 中的字段的列都建立一个索引， 通常将 1、2 中的字段建立联合索引效果更好•多表 join 的关联列

**5.如何选择索引列的顺序**

建立索引的目的是：希望通过索引进行数据查找，减少随机 IO，增加查询性能 ，索引能过滤出越少的数据，则从磁盘中读入的数据也就越少。

•区分度最高的放在联合索引的最左侧（区分度=列中不同值的数量/列的总行数）•尽量把字段长度小的列放在联合索引的最左侧（因为字段长度越小，一页能存储的数据量越大，IO 性能也就越好）•使用最频繁的列放到联合索引的左侧（这样可以比较少的建立一些索引）

**6. 避免建立冗余索引和重复索引（增加了查询优化器生成执行计划的时间）**

•重复索引示例：primary key(id)、index(id)、unique index(id)•冗余索引示例：index(a,b,c)、index(a,b)、index(a)

**7. 对于频繁的查询优先考虑使用覆盖索引**

覆盖索引：就是包含了所有查询字段 (where,select,ordery by,group by 包含的字段) 的索引

**覆盖索引的好处：**

•**避免 Innodb 表进行索引的二次查询:** Innodb 是以聚集索引的顺序来存储的，对于 Innodb 来说，二级索引在叶子节点中所保存的是行的主键信息，如果是用二级索引查询数据的话，在查找到相应的键值后，还要通过主键进行二次查询才能获取我们真实所需要的数据。而在覆盖索引中，二级索引的键值中可以获取所有的数据，避免了对主键的二次查询 ，减少了 IO 操作，提升了查询效率。•**可以把随机 IO 变成顺序 IO 加快查询效率:** 由于覆盖索引是按键值的顺序存储的，对于 IO 密集型的范围查找来说，对比随机从磁盘读取每一行的数据 IO 要少的多，因此利用覆盖索引在访问时也可以把磁盘的随机读取的 IO 转变成索引查找的顺序 IO。

**8.索引 SET 规范**

**尽量避免使用外键约束**

•不建议使用外键约束（foreign key），但一定要在表与表之间的关联键上建立索引•外键可用于保证数据的参照完整性，但建议在业务端实现•外键会影响父表和子表的写操作从而降低性能

# 数据库 SQL 开发规范

**1. 建议使用预编译语句进行数据库操作**

预编译语句可以重复使用这些计划，减少 SQL 编译所需要的时间，还可以解决动态 SQL 所带来的 SQL 注入的问题。

只传参数，比传递 SQL 语句更高效。

相同语句可以一次解析，多次使用，提高处理效率。

**2. 避免数据类型的隐式转换**

隐式转换会导致索引失效如:

select name,phone from customer where id = '111';

**3. 充分利用表上已经存在的索引**

避免使用双%号的查询条件。如：a like '%123%'，（如果无前置%,只有后置%，是可以用到列上的索引的）

一个 SQL 只能利用到复合索引中的一列进行范围查询。如：有 a,b,c 列的联合索引，在查询条件中有 a 列的范围查询，则在 b,c 列上的索引将不会被用到。

在定义联合索引时，如果 a 列要用到范围查找的话，就要把 a 列放到联合索引的右侧，使用 left join 或 not exists 来优化 not in 操作，因为 not in 也通常会使用索引失效。

**4. 数据库设计时，应该要对以后扩展进行考虑**

**5. 程序连接不同的数据库使用不同的账号，进制跨库查询**

•为数据库迁移和分库分表留出余地•降低业务耦合度•避免权限过大而产生的安全风险

**6. 禁止使用 SELECT \* 必须使用 SELECT <字段列表> 查询**

**原因：**

•消耗更多的 CPU 和 IO 以网络带宽资源•无法使用覆盖索引•可减少表结构变更带来的影响

**7. 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');

**8. 避免使用子查询，可以把子查询优化为 join 操作**

通常子查询在 in 子句中，且子查询中为简单 SQL(不包含 union、group by、order by、limit 从句) 时,才可以把子查询转化为关联查询进行优化。

**子查询性能差的原因：**

子查询的结果集无法使用索引，通常子查询的结果集会被存储到临时表中，不论是内存临时表还是磁盘临时表都不会存在索引，所以查询性能会受到一定的影响。特别是对于返回结果集比较大的子查询，其对查询性能的影响也就越大。

由于子查询会产生大量的临时表也没有索引，所以会消耗过多的 CPU 和 IO 资源，产生大量的慢查询。

**9. 避免使用 JOIN 关联太多的表**

对于 MySQL 来说，是存在关联缓存的，缓存的大小可以由 join\_buffer\_size 参数进行设置。

在 MySQL 中，对于同一个 SQL 多关联（join）一个表，就会多分配一个关联缓存，如果在一个 SQL 中关联的表越多，所占用的内存也就越大。

如果程序中大量的使用了多表关联的操作，同时 join\_buffer\_size 设置的也不合理的情况下，就容易造成服务器内存溢出的情况，就会影响到服务器数据库性能的稳定性。

同时对于关联操作来说，会产生临时表操作，影响查询效率，MySQL 最多允许关联 61 个表，建议不超过 5 个。

**10. 减少同数据库的交互次数**

数据库更适合处理批量操作，合并多个相同的操作到一起，可以提高处理效率。

**11. 对应同一列进行 or 判断时，使用 in 代替 or**

in 的值不要超过 500 个，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。

**12. 禁止使用 order by rand() 进行随机排序**

order by rand() 会把表中所有符合条件的数据装载到内存中，然后在内存中对所有数据根据随机生成的值进行排序，并且可能会对每一行都生成一个随机值，如果满足条件的数据集非常大，就会消耗大量的 CPU 和 IO 及内存资源。

推荐在程序中获取一个随机值，然后从数据库中获取数据的方式。

**13. WHERE 从句中禁止对列进行函数转换和计算**

对列进行函数转换或计算时会导致无法使用索引

**不推荐：**

where date(create\_time)='20190101'

**推荐：**

where create\_time >= '20190101' and create\_time < '20190102'

**14. 在明显不会有重复值时使用 UNION ALL 而不是 UNION**

•UNION 会把两个结果集的所有数据放到临时表中后再进行去重操作•UNION ALL 不会再对结果集进行去重操作

**15. 拆分复杂的大 SQL 为多个小 SQL**

•大 SQL 逻辑上比较复杂，需要占用大量 CPU 进行计算的 SQL•MySQL 中，一个 SQL 只能使用一个 CPU 进行计算•SQL 拆分后可以通过并行执行来提高处理效率

# 数据库操作行为规范

**1. 超 100 万行的批量写 (UPDATE,DELETE,INSERT) 操作,要分批多次进行操作**

**大批量操作可能会造成严重的主从延迟**

主从环境中,大批量操作可能会造成严重的主从延迟，大批量的写操作一般都需要执行一定长的时间， 而只有当主库上执行完成后，才会在其他从库上执行，所以会造成主库与从库长时间的延迟情况

**binlog 日志为 row 格式时会产生大量的日志**

大批量写操作会产生大量日志，特别是对于 row 格式二进制数据而言，由于在 row 格式中会记录每一行数据的修改，我们一次修改的数据越多，产生的日志量也就会越多，日志的传输和恢复所需要的时间也就越长，这也是造成主从延迟的一个原因

**避免产生大事务操作**

大批量修改数据，一定是在一个事务中进行的，这就会造成表中大批量数据进行锁定，从而导致大量的阻塞，阻塞会对 MySQL 的性能产生非常大的影响。

特别是长时间的阻塞会占满所有数据库的可用连接，这会使生产环境中的其他应用无法连接到数据库，因此一定要注意大批量写操作要进行分批

**2. 对于大表使用 pt-online-schema-change 修改表结构**

•避免大表修改产生的主从延迟•避免在对表字段进行修改时进行锁表

对大表数据结构的修改一定要谨慎，会造成严重的锁表操作，尤其是生产环境，是不能容忍的。

pt-online-schema-change 它会首先建立一个与原表结构相同的新表，并且在新表上进行表结构的修改，然后再把原表中的数据复制到新表中，并在原表中增加一些触发器。把原表中新增的数据也复制到新表中，在行所有数据复制完成之后，把新表命名成原表，并把原来的表删除掉。把原来一个 DDL 操作，分解成多个小的批次进行。

**3. 禁止为程序使用的账号赋予 super 权限**

•当达到最大连接数限制时，还运行 1 个有 super 权限的用户连接•super 权限只能留给 DBA 处理问题的账号使用

**4. 对于程序连接数据库账号,遵循权限最小原则**

•程序使用数据库账号只能在一个 DB 下使用，不准跨库•程序使用的账号原则上不准有 drop 权限