

**数据库规范**

拟制人 李晓愚 日期 2019年12月30日

审核人 李晓愚 日期 2019年12月30日

目录

[数据库规范 1](#_Toc28347)

[目的 4](#_Toc13948)

[数据库定义规范 4](#_Toc29059)

[一、 数据库命名规范 4](#_Toc15360)

[二、 对象命名规范 4](#_Toc4737)

[MYSQL数据库基本设计规范 5](#_Toc13060)

[引擎 5](#_Toc21401)

[字符集 5](#_Toc24815)

[字段设计 6](#_Toc19295)

[数据库字段设计规范 7](#_Toc30118)

[1. 关于数据长度 7](#_Toc5952)

[2. 禁止使用TEXT/BLOB类型，禁止在数据库中存储图片，文件等大的二进制数据 7](#_Toc31125)

[3. 避免使用ENUM(枚举)类型 8](#_Toc17376)

[4. 尽可能把所有列定义为NOT NULL 8](#_Toc32015)

[5. 使用TIMESTAMP（4个字节）或DATETIME类型（5个字节）存储时间 8](#_Toc1406)

[6. 同财务相关的金额类数据{设计使用小数}必须使用decimal类型 8](#_Toc7086)

[7. 同一意义的字段定义必须相同 8](#_Toc10330)

[8. 修改浮点型、定点型字段定义，千万谨慎 9](#_Toc15247)

[9. 增加字段时禁止指定after 9](#_Toc8565)

[10. 整形定义中不添加长度，比如使用INT，而不是INT[4] 9](#_Toc28581)

[11. VARCHAR(N)，N尽可能小 9](#_Toc10785)

[12. 数值型字段，default值建议选用0 9](#_Toc23930)

[索引设计规范 9](#_Toc25912)

[1. 创建表一定要有主键（PRIMARY KEY），要求使用整型 9](#_Toc18962)

[2. 限制每张表上的索引数量 9](#_Toc17146)

[3. 避免建立冗余索引和重复索引 10](#_Toc20174)

[4. 优先考虑覆盖索引 10](#_Toc16605)

[5. 一定要在表与表之间的关联键上建立索引 10](#_Toc32385)

[SQL开发规范 10](#_Toc19666)

[1. 建议使用预编译语句进行数据库操作 10](#_Toc17429)

[2. 避免数据类型的隐式转换 10](#_Toc4051)

[3. 充分利用表上已经存在的索引 10](#_Toc20341)

[4. 避免使用双%号的查询条件 10](#_Toc16816)

[5. 一个SQL只能利用到复合索引中的一列进行范围查询 10](#_Toc19085)

[6. 使用join(left join)来优化子查询 11](#_Toc30838)

[7. 数据库设计时，应该要对以后扩展进行考虑 11](#_Toc8108)

[8. 程序连接不同的数据库使用不同的账号，禁止跨库查询 11](#_Toc14313)

[9. 禁止使用SELECT \* 必须使用SELECT <字段列表> 查询 11](#_Toc9095)

[10. 禁止使用不含字段列表的INSERT语句 11](#_Toc27105)

[11. 避免使用JOIN关联太多的表 11](#_Toc18597)

[12. 减少同数据库的交互次数 12](#_Toc28307)

[13. 对应同一列进行or判断时，使用in代替or 12](#_Toc28429)

[14. 禁止使用order by rand()进行随机排序 12](#_Toc11267)

[15. WHERE从句中禁止对列进行函数转换和计算 12](#_Toc14944)

[16. 在明显不会有重复值时使用UNION ALL而不是UNION 12](#_Toc32315)

[17. 拆分复杂的大SQL为多个小SQL 12](#_Toc27079)

[18. 每个表增加create\_time、update\_time两个字段 12](#_Toc13814)

[数据库操作行为规范 13](#_Toc18533)

[1. 超1万行的批量写（UPDATE、DELETE、INSERT）操作，要分批多次进行操作 13](#_Toc475)

[2. 对于大表使用pt-online-schema-change修改表结构或者使用mysql的online ddl 13](#_Toc16928)

[3. 同一张表的结构多次修改，语句尽量整合 13](#_Toc16011)

[4. DDL操作中，add和modify建议拆分，或者有相应说明 14](#_Toc18439)

[数据库权限管理规范 14](#_Toc19462)

[1. 禁止为程序使用的账号赋予super权限 14](#_Toc18596)

[2. 对于程序连接数据库账号，遵循权限最小原则 14](#_Toc15146)

[3. 账号授权精确到网段的前三位 14](#_Toc25578)

[4. 线上数据查询使用查询系统 14](#_Toc31846)

**目的**

为加强北京中交兴路信息科技有限公司（以下简称“公司”）数据库管理，规范数据库相关操作，降低优化成本，增强数据安全性，特制定本管理制度。

**数据库定义规范**

1. **数据库命名规范**
2. 采用小写字母、数字（通常不需要）和下划线组成。禁止使用’-’
3. 命名简洁、含义明确。
4. **对象命名规范**
5. **表命名**
   1. 根据类型不同，采用不同的前缀。
   2. 采用小写字母、数字（通常不需要）和下划线组成。
   3. 长度控制在30个字符之内。

推荐命名规则如下：

|  |  |  |
| --- | --- | --- |
| 表类型 | 表前缀 | 说明 |
| 业务表 | TB\_ |  |
| 关系表 | TR\_ |  |
| 历史表 | TH\_ |  |
| 统计表 | TS\_ |  |
| 日志表 | TL\_XX\_LOG |  |
| 系统表、字典表、码表 | SYS\_ |  |
| 临时表 | TMP\_ | 禁用-临时用 |
| 备份表 | BAK\_XX\_YYYYMMDD | 如当日创建多个，则在日期后增加数字后缀。研发人员禁止使用 |
| 业务分区表 | 表名\_YYYY（MMDD） | 业务分区采用以上命名规则+时间结尾，根据年月日采用不同结尾长度 |
|  |  |  |

1. **视图**

|  |  |  |
| --- | --- | --- |
| 类型 | 前缀 | 说明 |
| 视图 | view\_ | 尽量避免使用 |

1. **索引**
2. 根据类型不同，采用不同的前缀。
3. 采用小写字母、数字（通常不需要）和下划线组成。
4. 长度控制在30个字符之内。

|  |  |  |
| --- | --- | --- |
| 类型 | 前缀 | 说明 |
| 主键名 | PK\_ | 以PK\_+表名+主键字段名构成。例如PK\_SYS\_CAR\_ID |
| 外键名 | FK\_ | 禁止使用 |
| 普通索引 | IDX\_ | 以IDX\_+表名+索引字段名构成。例如IDX\_SYS\_CAR\_DIN；如果复合索引的构成字段较多，则只包含第一个字段 |
| 唯一索引 | UNIQ\_ | 以UNIQ\_+表名+索引字段名构成。表名可以去掉前缀。例如UNIQ\_SYS\_CAR\_DIN。 |

**MYSQL数据库基本设计规范**

**引擎**

1. 没有特殊要求mysql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（5.5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

**字符集**

1. 数据库和表的字符集统一,尽量使用UTF8（根据业务需求）

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

#### UTF8和UTF8MB4字段进行关联，会导致索引失效

#### 除非特殊情况，禁止建表指定字符集（采用库默认字符集），降低出现字符集不统一导致性能问题的风险。

#### 无特殊要求，禁止指定表COLLATE

COLLATE主要的作用是排序的规则以及检索的规则，utf8字符集默认的是 utf8\_unicode\_ci ，utf8mb4字符集默认的是utf8mb4\_unicode\_ci，结尾的ci意思是不区分大小写。

COLLATE会影响到ORDER BY语句的顺序，会影响到WHERE条件中大于小于号筛选出来的结果，比如：select \* from test where name like 'A%',在 utf8\_bin字符集下，是无法检索出 ‘abc’字段的，并且排序的情况下Abc和abc所在的顺序是不一致的。

#### 慎重选择row\_format

db默认的innodb\_file\_format 为 Barracuda，默认的innodb\_default\_row\_format为 dynamic；其中COMPRESSED 压缩比经测试最大也就 1/2，但读取和写入会有额外cpu开销，并且申请内存是按照解压后的原大小申请，在高并发情况下容易导致性能问题。

**字段设计**

#### 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

#### 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

#### 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，另外，对于表结构维护，分区表的维护造成的开销更集中，建议采用物理分表的方式管理大数据

#### 建议将大字段，访问频度低的字段拆分到单独的表中存储，分离冷热数据，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。为减少磁盘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）。对于非常用字段，建议采用扩展表的方式进行分表。

注意：每一行数据的65535字节中，utf8字符集下，varchar每一个长度占用3个字节，utf8mb4字符集下，每一个长度占用4个字节。

#### 尽量不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

#### 禁止使用外键约束

外键使得表之间相互耦合，影响update/delete等SQL性能，有可能造成死锁，高并发情况下容易成为数据库瓶颈。建议在业务端实现。

**数据库字段设计规范**

1. 关于数据长度

够用前提下，越短越好，这样能够消耗更少的存储空间；因排序申请的内存大小和字段长度有关，需要进行排序时，长度小的字段消耗更少的内存空间；优先选择符合存储需要的最小的数据类型

1. 禁止使用TEXT/BLOB类型，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

1. 避免使用ENUM(枚举)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

1. 尽可能把所有列定义为NOT NULL

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

NULL只能采用IS NULL或者IS NOT NULL，而在=/!=/in/not in时很容易造成查询结果与设计逻辑不符

1. 使用TIMESTAMP（4个字节）或DATETIME类型（5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

1. 同财务相关的金额类数据{设计使用小数}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。

1. 同一意义的字段定义必须相同

同一意义的字段定义包括字段类型和长度范围必须相同

比如tb\_user表中的id和tb\_user\_item表中的user\_id,需要定义成一样的类型。如果类型不一样，关联的时候会发生强制类型转换，不会用到索引。如果范围不一样，那么一个表正常插入，另一个表就超出范围报错了。

1. 修改浮点型、定点型字段定义，千万谨慎

修改浮点型、定点型字段定义时，如果精度与线上不一致，会导致数据进行四舍五入，从而导致数据（尤其是钱、计算倍率）出现问题。

1. 增加字段时禁止指定after
2. 整形定义中不添加长度，比如使用INT，而不是INT[4]

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，比如INT(3)默认显示3位，空格补齐，超出时正常显示，python、java客户端等不具备这个功能。

1. VARCHAR(N)，N尽可能小

VARCHAR(N)，N表示的是字符数不是字节数，比如VARCHAR(255)，可以最大可存储255个汉字，需要根据实际的宽度来选择N。进行排序和创建临时表一类的内存操作时，会使用N的长度申请内存。VARCHAR(N)，如果N<256时会使用一个字节来存储长度，如果N>=256则使用两个字节来存储长度。

1. 数值型字段，default值建议选用0

**索引设计规范**

1. 创建表一定要有主键（PRIMARY KEY），要求使用整型

MYSQL主键建议使用自增ID值。

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）。

1. 限制每张表上的索引数量

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。

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）。

1. 避免建立冗余索引和重复索引

因为这样会增加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)

1. 优先考虑覆盖索引

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：1.可以把随机IO变成顺序IO加快查询效率；2.能够避免回表查询，提升查询效率

1. 一定要在表与表之间的关联键上建立索引

sql性能问题中，最多遇到的就是关联键索引确实导致sql性能低下，所以在设计中，关联键一定要建立索引

**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。

1. 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';

1. 充分利用表上已经存在的索引
2. 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）。

1. 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。

1. 使用join(left join)来优化子查询

在MYSQL中，子查询很容易导致性能问题，而且在某些情况下，会有先进行关联，再进行子表数据筛选的情况，从而导致两个几十万数据的表，关联时间超过10个小时。另外，建议研发写出的语句，上线前在测试环境关注下执行计划，如果执行计划中出现SUBQUERY , DEPENDENT SUBQUERY ,或者Using temporary,Using join buffer等情况，通常都代表有性能问题。

注：mysql在8.0.6以后的版本才存在hash join，当前表关联是通过nested loop进行，所以表关联对索引的要求很高。

1. 数据库设计时，应该要对以后扩展进行考虑
2. 程序连接不同的数据库使用不同的账号，禁止跨库查询

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

1. 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

1. 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');

1. 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OMM{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。建议不超过3个（对于一些中间件来说，通常最多支持3个表的联查）。

1. 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。

1. 对应同一列进行or判断时，使用in代替or

建议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。

1. 禁止使用order by rand()进行随机排序

此类排序会把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。

1. 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

1. 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和排序操作

UNION ALL不会再对结果集进行去重和排序操作

1. 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：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

1. 每个表增加create\_time、update\_time两个字段

分别表示写入时间以及最后更新时间；业务上可能用不到，但是对日常运维管理则非常有用；可以用来判断哪些是可以归档的老数据，定期进行归档；用来做自定义的差异备份也很方便；另外，对于数据恢复以及影响范围定位能提供极大的帮助

**数据库操作行为规范**

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

大批量操作，容易超过最大packet长度（mysql中默认4M），从而导致报错；

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；

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。

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。

操作中如果出现异常导致回滚，会导致产生大量的IO操作，严重情况可能导致数据库无法响应。

1. 对于大表使用pt-online-schema-change修改表结构或者使用mysql的online ddl

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。

PS.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。

1. 同一张表的结构多次修改，语句尽量整合

因上述原因，大表DDL操作的时候，需要进行整合，避免多次DDL均造成对数据库的影响

整合示例：ALTER TABLE `carr\_vehicle` ADD COLUMN `upload\_status` tinyint(1) DEFAULT '0' COMMENT '上报状态：0未上报 1已上报 2上报失败',ADD COLUMN `upload\_date` datetime DEFAULT NULL COMMENT '上报时间'；

1. DDL操作中，add和modify建议拆分，或者有相应说明

对于DBA进行线上维护时，modify操作通常都会对原结构和数据进行备份，如果提交的脚本在大量add后增加modify，尤其modify的字段为定点型和浮点型字段时，容易导致因备份缺失而无法恢复。

## 数据库权限管理规范

1. 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。

1. 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。

1. 账号授权精确到网段的前三位

账号授权至少精确到前三位ip段，甚至精确到ip地址，防止出现应用程序配置错误连接导致生产数据误操作。

1. 线上数据查询使用查询系统

线上查询必须使用Sindb以只读授权进行查询，具备查询审计、超时管控、数据限制等方面，避免引起线上性能问题以及数据泄露，确保线上数据的安全和稳定。