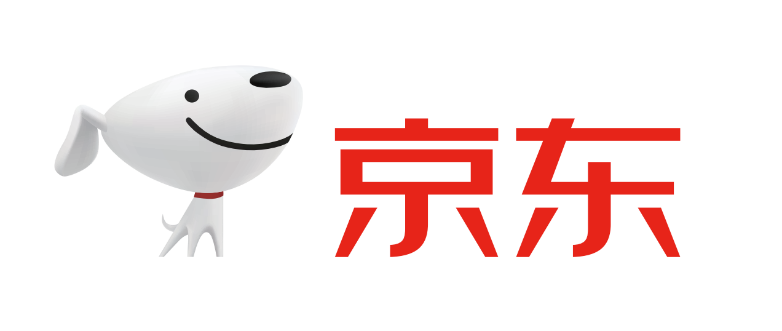
**生旅产品研发部**

**MYSQL数据库规范**



文档更改记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修改日期 | 修改人 | 更改内容 |
| 0.1 | 2018年12月11日 | 伍振华 | 完成初稿 |
| 0.2 | 2018年12月19日 | 伍振华 | 根据初审意见修订内容 |
| 1.0 | 2018年12月26日 | 伍振华 | 合并技术委员会意见，发布版本 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 建表规约

### 【强制】数据库字符集选择UTF8

一般使用普通的utf8，如果需要存储emoji、生僻字等4字节的unicode字符，请使用utf8mb4。万国码，无需转码，无乱码风险，节省空间

### 【强制】使用InnoDB存储引擎

5.5以后的默认引擘，支持事务，行级锁，更好的恢复性，高并发下性能更好，对多核，大内存，ssd等硬件支持更好。

除非是不需要事务，而且是插入不频繁、查询非常频繁的表，才使用MyISAM。

### 【强制】库名、表名、字段名不允许使用大写字母，使用“\_”分割

大小写共用容易引起混乱，为了规范，不允许使用大写：

1. MySQL有配置参数lower\_case\_table\_names，不可动态更改，linux系统默认为0，即库表名以实际情况存储，大小写敏感。如果是1，以小写存储，大小写不敏感。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。

禁止出现数字开头，禁止两个下划线中间只出现数字。

正例：getter\_admin，task\_config，level3\_name 反例：GetterAdmin，taskConfig，level\_3\_name。

### 【强制】禁止存储大文件或者大照片

解读：为何要让数据库做它不擅长的事情？大文件和照片存储在文件系统，数据库里存URI多好

### 【强制】库名、表名、字段名见名知意,建议使用名词而不是动词

1. 便于理解。
2. 库表是一种客观存在的事物，一种对象，所以建议使用名词。
3. 表名不使用复数名词：表名应该仅仅表示表里面的实体内容，不应该表示实体数量，对应于DO类名也是单数形式，符合表达习惯。

### 【强制】命名禁用保留字

如desc、range、match、delayed等，请参考MySQL官方保留字（https://dev.mysql.com/doc/refman/5.7/en/keywords.html）。

### 【强制】表命名详细规则

1. 表名长度不能超过30个字符，表名中含有单词全部采用单数形式。
2. 表名含有的单词用完整的单词,如果导致基本表名长度超过26个字符，则从最后一个单词开始，依次向前采用该单词的缩写。（如果没有约定的缩写，则采用该单词前4个字母来表示）。

### 【强制】使用整数类型（一般是TINYINT UNSIGNED）存储枚举类型

ENUM，有下面三个问题，不建议使用。

1. 添加新的值要做DDL；
2. 默认值问题(将一个非法值插入ENUM【也就是说，允许的值列之外的字符串】，将插入空字符串以作为特殊错误值)；
3. 索引值问题（插入数字实际是插入索引对应的值）。

### 【强制】使用decimal存储小数类型，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。

因为标准的计算机浮点数，在内部是用二进制表示的，但在将一个十进制数转换为二进制浮点数时，可能造成误差，原因是不是所有的数（比如一个带小数的十进制）都能转换成有限长度的二进制数。

另外，对于有固定小数位精度的小数类型（比如金额，精确到分，以元表示时），可以考虑使用整数表示（存储时乘以固定的倍数，在实际使用时再除掉）。

### 【强制】如果存储的字符串长度几乎相等，使用char定长字符串类型

比如UUID列，建议使用CHAR(36)。

### 【强制】表必备字段：create\_time, update\_time

create\_time：表示记录创建时间，TIMESTAMP类型，要求NOT NULL DEFAULT CURRENT\_TIMESTAMP，即自动设置为记录插入时间。

update\_time：表示记录最后更新时间，TIMESTAMP类型，要求NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP，即自动设置为最后更新时间[更新记录时数据库的时间戳]。在某些业务场景下，该字段的时间不能使用更新记录时数据库的时间戳（如：历史数据迁移时该时间需要继续使用历史的时间戳；需要记录业务更新时间，由外部的业务更新数据源给定时间戳），那么需要通过代码来控制更新时间，手动”set update\_time= 实际时间”。

该功能要求MYSQL版本不低于5.6，因为5.6之前一个表不支持多余一个的自动更新TIMESTAMP列。

### 【强制】整型定义中不添加长度，比如使用INT，而不是INT(4)

整数定义的长度不像字符串类型，不影响实际存储的数据长度，比如INT(4)也可以存储12345，为了避免误解，定义时不指定长度。

### 【强制】数据库 - 视图命名

1. 视图名用系统简称\_V\_开头，视图名长度不能超过30个字符。
2. 视图由几个表产生就用下划线（\_）连接几个表的名，如果表过多可以将表名适当简化，但一定要列出所有表名。
3. 一般不建议使用视图，尽量在应用程序内完成视图的工作。

### 【强制】禁止使用VARBINARY、BLOB存储图片、文件

推荐使用oss存储。

### 【强制】禁止使用分区表

MySQL 的分区表实际性能不是很好，且管理维护成本较高。

数据量过大时建议使用分库分表的中间件或者弹性数据库。

### 【推荐】varchar长度尽可能小，且不大于5000

VARCHAR(N)，N尽可能小，因为MySQL在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。

N大于5000时，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。（text类型慎用，可考虑使用对象存储来替代）

### 【推荐】临时库、临时表名必须以 tmp 为前缀并以日期为后缀

### 【推荐】备份库、备份表名必须以 bak 为前缀并以日期为后缀

### 【推荐】字段使用注释说明含义

修改字段含义或对字段表示的状态追加时，需要及时更新字段注释。

### 【推荐】适当使用冗余字段

冗余字段可以提高查询性能，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：用户姓名使用频率高，字段长度短，基本一成不变，可在相关联的表中冗余存储，避免关联查询。

### 【推荐】单表行数超过500万行或者单表容量超过2GB，才推荐进行分库分表

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。

### 【推荐】使用INT UNSIGNED存储IPV4

使用INT UNSIGNED而不是char(15)来存储ipv4地址，可以节省空间，读写时INET\_ATON和INET\_NTOA函数进行转换。

如果需要存储ipv6，可以使用VARBINARY(16)类型。

### 【推荐】存储时间（精确到秒）建议使用TIMESTAMP类型

一般优先使用TIMESTAMP而不是DATETIME，因为：

1. TIMESTAMP使用4字节，而DATETIME需要8个字节。
2. TIMESTAMP存储时间范围是1970-01-01 08:00:01到2038-01-19 11:14:07，一般可以满足需求，否则请使用DATETIME。
3. TIMESTAMP可以自动更新（CURRENT\_TIMESTAMP），不过从5.6版本开始，DATETIME也支持了。

### 【推荐】建议字段定义为NOT NULL

1. 如果null字段被索引，需要额外的1字节；
2. null使索引，索引统计，值的比较变得更复杂；
3. 可用0，''代替；
4. 如果是索引字段，一定要定义为not null。
5. mybatis更新/插入时，对于内容为null的输入字段，不要直接赋值给not null字段，否则报错，请使用if语句进行判断。

### 【推荐】使用合适的存储长度

不但节约数据库表空间、节约索引存储，更重要的是提升检索速度。

1. 使用UNSIGNED存储非负整数，可以避免误存负数，且扩大了表示范围。
2. 可以预估存储整数的上下限时，使用可以满足存储要求的最短整数类型。tinyint,smallint,int,bigint4种整数类型分别占用1,2,4,8个字节。

比如，比如存储范围是0-200，那么请使用TINYINT UNSIGNED。

### 【推荐】TIMESTAMP类型使用default值并由程序定义值

在MySQL 5.6.6之前，TIMESTAMP列如果没有明确声明NULL属性，默认为NOT NULL。（而其他数据类型，如果没有显示声明为NOT NULL，则允许NULL值。）设置TIMESTAMP的列值为NULL，会自动存储为当前timestamp。建议由程序定义数值。

CREATE TABLE语句中，第1个TIMESTAMP列可以用下面的任何一种方式声明：

1) 如果定义时DEFAULT CURRENT\_TIMESTAMP和ON UPDATE CURRENT\_TIMESTAMP子句都有，列值为默认使用当前的时间戳，并且自动更新。

2) 如果不使用DEFAULT或ON UPDATE子句，那么它等同于DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP。

3) 如果只有DEFAULT CURRENT\_TIMESTAMP子句，而没有ON UPDATE子句，列值默认为当前时间戳但不自动更新。

4) 如果没用DEFAULT子句，但有ON UPDATE CURRENT\_TIMESTAMP子句，列默认为0并自动更新。

5) 如果有一个常量值DEFAULT，该列会有一个默认值，而且不会自动初始化为当前时间戳。如果该列还有一个ON UPDATE CURRENT\_TIMESTAMP子句，这个时间戳会自动更新，否则该列有一个默认的常量但不会自动更新。

第2个TIMESTAMP列，如果没有声明为NULL或者DEFAULT子句，默认自动分配’0000-00-00 00:00:00’。插入行时没有指明改列的值，该列默认分配’0000-00-00 00:00:00’，且没有警告。

要关闭警告，需要加入下面的参数：

explicit\_defaults\_for\_timestamp=1

### 【推荐】非负数字段，必须是 unsigned

### 【推荐】表达是与否概念的字段，使用 is\_xxx 的方式命名

数据类型使用 unsigned tinyint（ 1 表示是，0 表示否）

### 【推荐】必须使用varchar(20)存储手机号

解读：

a）涉及到区号或者国家代号，可能出现+-()

b）手机号会去做数学运算么？

c）varchar可以支持模糊查询，例如：like“138%”

### 【参考】拆分大字段和访问频率低的字段，分离冷热数据

## 索引规约

为什么MySQL的性能依赖于索引？

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度（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或者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，当拼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会进行全表扫描，会消耗大量的IO。

### 【强制】索引命名规则

1. 索引命名使用全大写。
2. 包含多列的索引，每列都要在命名中提现，使用下划线(\_)分隔。
3. 索引名长度不超过30字符。
4. 主键索引pk\_开头；唯一索引uniq\_开头；普通索引idx\_开头

### 【强制】单张表的索引数量控制在5个以内

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

### 【强制】表必须有主键

1. 有唯一键且由3个以下字段组成，并且字段都是整型时，使用唯一键作为主键。
2. 如果不满足条件1，则使用自增（或者通过发号器获取）id作为主键。

### 【强制】超过三个表禁止join

随着表数量的增加，join的效率会急剧下降。

需要join的字段，数据类型必须绝对一致；多表关联查询时，保证被关联的字段需要有索引。说明：即使双表join也要注意表索引、SQL性能。

对于man端等一些统计型的慢SQL，请务必使用从库。

### 【推荐】业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引

说明：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。

某些场景（如逻辑删除），直接使用唯一业务字段无法实现唯一索引，那么可以有两种方案：

1. 放弃唯一索引，通过应用层控制
2. 联合（唯一业务字段，删除flag）构成唯一索引，然后删除flag取值按照版本号方式生成（如0表示正常，1表示第一次删除，2表示第二次删除等）

### 【推荐】页面搜索不用左模糊或者全模糊

如果需要请走搜索引擎来解决。说明：索引文件具有B-Tree的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。

### 【推荐】在varchar字段上建立索引时，建议指定索引长度

全字段索引可能降低效率，没必要对全字段建立索引，可以根据实际文本区分度决定索引长度即可。说明：索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为20的索引，区分度会高达90%以上，可以使用count(distinct left(列名, 索引长度))/count(\*)的区分度来确定。

### 【推荐】合理创建联合索引（避免冗余）

说明：(a,b,c) 相当于 (a) 、(a,b) 、(a,b,c)。

### 【推荐】利用覆盖索引来进行查询操作，避免回表

说明：如果一本书需要知道第11章是什么标题，会翻开第11章对应的那一页吗？目录浏览一下就好，这个目录就是起到覆盖索引的作用。正例：能够建立索引的种类：主键索引、唯一索引、普通索引，而覆盖索引是一种查询的一种效果，用explain的结果，extra列会出现：using index。

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。如果用户需要查询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。

覆盖索引的概念就是查询可以通过在一个索引中完成，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。

### 【推荐】SQL性能优化的目标：至少要达到 range 级别

使用EXPLAIN判断SQL语句是否合理使用索引，至少要达到 range 级别，要求是ref级别，如果可以是const最好。尽量避免extra列出现：Using File Sort，Using Temporary。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 对索引进行范围检索。

反例：explain表的结果，type=index，索引物理文件全扫描，速度非常慢，这个index级别比较range还低，和全表扫描类似。

### 【推荐】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，避免出现file\_sort的情况，影响查询性能。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。

### 【推荐】建组合索引的时候，区分度最高的在最左边

正例：如果where a=? and b=? ，a列的几乎接近于唯一值，那么只需要单建idx\_a索引即可。

说明：存在非等号和等号混合判断条件时，在建索引时，请把等号条件的列前置。如：where a>? and b=? 那么即使a的区分度更高，也必须把b放在索引的最前列。

### 【参考】面向用户流程的SQL尽量不用join

Join性能比较差，可以考虑用宽表、应用程序内聚合等方式来替代。

### 【参考】过长VARCHAR字段索引优化方法

如果VARCHAR索引长度过长时，效率会明细变差，如果较短索引的区分度不够（见第7条），那么可以添加crc32或者MD5 Hash字段，对该字段建立索引。

如下面的表增加一列url\_crc32，然后对url\_crc32建立索引，减少索引字段的长度，提高效率。

CREATE TABLE url(

……

url VARCHAR(255) NOT NULL DEFAULT 0,

url\_crc32 INT UNSIGNED NOT NULL DEFAULT 0,

……

index idx\_url(url\_crc32)

)

### 【参考】创建索引时应避免的极端误解

1. 宁滥勿缺：误认为一个查询就需要建一个索引。
2. 宁缺勿滥：误认为索引会消耗空间、严重拖慢更新和新增速度。
3. 抵制惟一索引：误认为业务的惟一性一律需要在应用层通过“先查后插”方式解决。

## SQL规约

### 【强制】使用count(\*)

不要使用count(列名)或count(常量)来替代count(\*)，count(\*)是SQL92定义的标准统计行数的语法，跟数据库无关，跟NULL和非NULL无关。

说明：count(\*)会统计值为NULL的行，而count(列名)不会统计此列为NULL值的行。

count(\*)和count(1)的实际执行流程是一样的。

### 【强制】查询、插入时使用显式的字段名称

SELECT、INSERT语句必须使用指明字段名称，不使用SELECT \*，不使用INSERT INTO table()。说明：

1. 增加查询分析器解析成本；
2. 增加很多不必要的消耗（cpu、io、内存、网络带宽）；
3. 增加了使用覆盖索引的可能性；
4. 表结构修改时容易引起前后端不一致。

### 【强制】使用prepared statement

可以提供性能并且避免SQL注入。

### 【强制】UPDATE、DELETE语句不使用LIMIT

1. 可能导致主从数据不一致
2. 会记录到错误日志，导致日志占用大量空间

### 【强制】count(distinct col) 计算该列除NULL之外的不重复行数

注意 count(distinct col1, col2) 如果其中一列全为NULL，那么即使另一列有不同的值，也返回为0。

### 【强制】使用ISNULL()来判断是否为NULL值

注意：NULL与任何值的直接比较都为NULL。 说明：

1. NULL<>NULL的返回结果是NULL，而不是false。
2. NULL=NULL的返回结果是NULL，而不是true。
3. NULL<>1的返回结果是NULL，而不是true。

### 【强制】结果数为0的分页逻辑

在代码中写分页查询逻辑时，若count为0应直接返回，避免执行后面的分页语句。

如果使用分页插件，也需要检查该处的实现逻辑（如pagehelper插件满足该逻辑）。

### 【强制】不得使用外键与级联

一切外键概念必须在应用层解决。 说明：（概念解释）学生表中的student\_id是主键，那么成绩表中的student\_id则为外键。如果更新学生表中的student\_id，同时触发成绩表中的student\_id更新，则为级联更新。外键与级联更新适用于单机低并发，不适合分布式、高并发集群；级联更新是强阻塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

### 【强制】sum(col)的空指针问题

当某一列的值全是NULL时，count(col)的返回结果为0，但sum(col)的返回结果为NULL，因此使用sum()时需注意NPE问题。 正例：可以使用如下方式来避免sum的NPE问题：SELECT IF(ISNULL(SUM(g)),0,SUM(g)) FROM table;

### 【强制】WHERE条件中必须使用合适的类型

比如id类型是string，那么不允许where id=1应该使用where id=’1’。说明：避免MySQL进行隐式类型转化，MySQL进行隐式类型转化之后，可能会将索引字段类型转化成=号右边值的类型，导致使用不到索引。

### 【强制】当有主从表时，要先删除从表记录，再删除主表记录

说明：如果从表中有主表外键关联的记录，先删主表会报错。不过规范3.8规定不允许外键，正常不会有主从表。

### 【强制】select、update、delete必须有where条件

容易造成误操作等严重后果。

### 【推荐】Order By排序时，优先使用主键列，索引列

### 【推荐】多表关联查询优化

多表联合查询，优先使用Where条件，再作表关联，并且被关联的字段尽量添加索引。

### 【推荐】减少使用in，控制in后边的集合大小

in操作能避免则避免，若实在避免不了，需要仔细评估in后边的集合元素数量，控制在1000个之内。元素较多时可以使用exist代替，性能一般会更好。

如果集合元素数据十分大，建议使用临时表+索引。

### 【推荐】避免使用存储过程、触发器、自定义函数

1. 容易将业务逻辑和DB耦合在一起，难以调试和扩展，移植性差；
2. MySQL的存储过程、触发器、函数中存在一定的bug。
3. 高并发大数据的互联网业务，架构设计思路是“解放数据库CPU，将计算转移到服务层”，并发量大的情况下，这些功能很可能将数据库拖死，业务逻辑放到服务层具备更好的扩展性，能够轻易实现“增机器就加性能”。数据库擅长存储与索引，CPU计算还是上移吧

### 【推荐】禁止在WHERE条件的属性上使用函数或者表达式

1. 解读：SELECT uid FROM t\_user WHERE from\_unixtime(day)>='2017-01-15' 会导致全表扫描

正确的写法是：SELECT uid FROM t\_user WHERE day>= unix\_timestamp('2017-01-15 00:00:00')

### 【推荐】避免WHERE条件中的非等值条件

非等值条件（IN、BETWEEN、<、<=、>、>=）会导致后面的条件使用不了索引。

### 【推荐】利用延迟关联或者子查询等优化超多分页

利用延迟关联或者子查询优化超多分页场景。 说明：MySQL并不是跳过offset行，而是取offset+N行，然后返回放弃前offset行，返回N行，那当offset特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行SQL改写。 正例：先快速定位需要获取的id段，然后再关联： SELECT a.\* FROM 表1 a, (select id from 表1 where 条件 LIMIT 100000,20 ) b where a.id=b.id

假如有类似下面分页语句：

SELECT \* FROM table ORDER BY TIME 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 table WHERE TIME<last\_TIME ORDER BY TIME DESC LIMIT 10.

SELECT \* FROM table inner JOIN(SELECT id FROM table ORDER BY TIME LIMIT 10000,10) as t USING(id)

### 【推荐】使用合理的SQL语句减少与数据库的交互次数

比如使用下面的语句来减少和db的交互次数：

1. INSERT ... ON DUPLICATE KEY UPDATE
2. REPLACE
3. INSERT IGNORE

### 【参考】避免使用复杂的SQL

避免使用复杂的SQL，将大的SQL拆分成多条简单SQL分步执行。说明：简单的SQL容易使用到MySQL的query cache；减少锁表时间特别是MyISAM；可以使用多核cpu。

## ORM规约

### 【强制】不要用 resultClass 当返回参数

即使所有类属性名与数据库字段一一对应，也需要定义，反过来，每一个表也必然有一个DO类与之对应；配置映射关系，使字段与 DO 类解耦，方便维护。

### 【强制】xml 配置中参数注意使用：#{}、#param#，不要使用${}

${}此种方式容易出现 SQL 注入。

### 【强制】不要使用自带的 queryForList(String statementName,int start,int size)

说明：其实现方式是在数据库取到statementName 对应的SQL 语句的所有记录，再通过subList取 start,size 的子集合，线上因为这个原因曾经出现过 OOM。

正例：在 sqlmap.xml 中引入 #start#, #size#

Map<String, Object> map = new HashMap<String, Object>();

map.put("start", start);

map.put("size", size);

### 【强制】确定不为空的条件中不要用if null判断

比如update table\_name set column\_name=1 where 1=1 and <if test=”id != null”> id=#{id}

如果数据异常造成id为空，则会造成更新全部数据的后果。

### 【强制】不允许直接拿 HashMap 与 Hashtable 作为查询结果集的输出

### 【推荐】POJO 类的 boolean 属性不能加 is，而数据库字段必须加 is\_，要求在 resultMap 中进行字段与属性之间的映射

参见定义POJO类以及数据库字段定义规定，在<resultMap>中增加映射，是必须的。

在MyBatis Generator生成的代码中，需要进行对应的修改。

### 【推荐】不要写一个大而全的数据更新接口

执行 SQL 时，尽量不要更新无改动的字段，而不管是不是自己的目标更新字段，都进行更新，如： update table set c1=value1,c2=value2,c3=value3。因为：

一是易出错；二是效率低；三是 binlog 增加存储。

## 操作规约

### 【强制】禁止从测试、开发环境直连线上数据库

### 【推荐】禁止在线上做数据库压力测试

如果环境、数据等原因必须使用线上数据库时，务必时刻关注影响，避免影响线上业务。

### 【推荐】线上数据库开启慢查询订阅，以及性能监控