**Mysql规范**

## 基础规约

1. 【强制】必须使用InnoDB存储引擎，除非有特殊性能或业务需求方可考虑其他存储引擎

说明：支持事务、行级锁、并发性能更好、CPU及内存缓存页优化使得资源利用率更高

2. 【参考】所有的字符存储与表示，均以 utf -8 编码，注意字符统计函数的区别。

说明：

SELECT LENGTH( "轻松工作" )； 返回为 12

SELECT CHARACTER \_ LENGTH( "轻松工作" )； 返回为 4

如果需要存储表情，那么选择 utf8mb4 来进行存储，注意它与utf -8 编码的区别。

3. 【强制】数据表、数据字段必须加入中文注释，枚举值的注释要注明枚举值的意义，业务逻辑上带单位的注明具体单位名称

说明：N年后谁tm知道这个r1,r2,r3字段是干嘛的

4. 【强制】禁止使用存储过程、函数、视图、触发器、Event

说明：

1. 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”
2. 并发量大的情况下，触发器和Event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。
3. 存储过程、函数和视图难以调试和扩展，更没有移植性。

5. 【强制】禁止存储大文件或者大照片

说明：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

6. 【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：以学生和成绩的关系为例，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，即为级联更新。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。

7. 【强制】不在数据库做运算，cpu计算务必移至业务层

8. 【强制】拒绝3B

 说明： 拒绝大sql语句：big sql

   拒绝大事物：big transaction

   拒绝大批量：big batch

9. 【强制】插入更新数据禁止使用now()等时间函数赋值，当前时间由业务层传入

说明：mysql SBR模式主从同步时会造成数据不一致

## 命名规约

1. 【强制】表名、字段名必须使用小写字母或数字 ， 禁止出现数字开头，禁止两个下划线中间只出现数字。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。

说明：MySQL 在 Windows 下不区分大小写，但在 Linux 下默认是区分大小写。因此，数据库名、表名、字段名，都不允许出现任何大写字母，避免节外生枝。

正例： aliyun\_admin ， rdc\_config ， level 3\_name

反例： AliyunAdmin ， rdcConfig ， level\_3\_name

2. 【强制】禁用保留字，如 desc 、 range 、 match 、 delayed 等，请参考 MySQL 官方保留字。

3. 【强制】表达是与否概念的字段，必须使用 is\_xxx 的方式命名，数据类型是 unsigned tinyint

（ 1 表示是，0 表示否 ） 。

说明：任何字段如果为非负数，必须是 unsigned 。

正例：表达逻辑删除的字段名 is\_deleted ，1 表示删除，0 表示未删除。

4. 【强制】表名不使用复数名词。

说明：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。

5. 【强制】主键索引名为 pk\_字段名；唯一索引名为 uk \_字段名 ； 普通索引名则为 idx \_字段名。

说明： pk\_即 primary key；uk\_ 即 unique key；idx\_即 index 的简称。

## 建表规约

1. 【强制】表必须有自增列作为主键，类型为bigInt Unsigned，主键名称也建议统一为id.

说明：mysql不指定主键，会默认创建一个6字节的列为隐藏主键；主键递增，数据行写入可以提高插入性能，可以避免page分裂，减少表碎片提升空间和内存的使用

2. 【强制】严禁使用null列，必须把字段定义为NOT NULL并且提供默认值

说明：

a）null的列使索引/索引统计/值比较都更加复杂，对MySQL来说更难优化

b）null 这种类型MySQL内部需要进行特殊处理，增加数据库处理记录的复杂性；同等条件下，表中有较多空字段的时候，数据库的处理性能会降低很多

c）null值需要更多的存储空，无论是表还是索引中每行中的null的列都需要额外的空间来标识

d）对null 的处理时候，只能采用is null或is not null，而不能采用=、in、<、<>、!=、not in这些操作符号。如：where name!=’shenjian’，如果存在name为null值的记录，查询结果就不会包含name为null值的记录

3. 【强制】禁止使用TEXT、BLOB类型，如果非要使用，请分表，用主表主键关联

说明：会浪费更多的磁盘和内存空间，非必要的大量的大字段查询会淘汰掉热数据，导致内存命中率急剧降低，影响数据库性能

4. 【强制】小数类型为 decimal ，禁止使用 float 和 double 。

说明： 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。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数分开存储。

5. 【强制】 禁止使用ENUM，可使用TINYINT代替

说明：

a）增加新的ENUM值要做DDL操作

b）ENUM的内部实际存储就是整数，你以为自己定义的是字符串？

6. 【参考】合适的字符存储长度，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。

正例：如下表，其中无符号值可以避免误存负数，且扩大了表示范围。

对象 年龄区间 类型 字节 表示范围

人 150 岁之内 unsigned tinyint 1 无符号值：0 到 255

龟 数百岁 unsigned smallint 2 无符号值：0 到 65535

恐龙化石 数千万年 unsigned int 4 无符号值：0 到约 42.9 亿

太阳 约 50 亿年 unsigned bigint 8 无符号值：0 到约 10 的 19 次方

7. 【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。

8. 【强制】 varchar 是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长度大于此值，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。

9. 【强制】表必备三字段： id , gmt \_ create , gmt \_ modified 。

说明：其中 id 必为主键，类型为 unsigned bigint 、单表时自增、步长为 1。 gmt\_create,

gmt\_modified 的类型均为 date\_time 类型，前者现在时表示主动创建，后者过去分词表示被动更新。

10. 【推荐】表的命名最好是加上“业务名称\_表的作用”。

正例： alipay \_ task / force \_ project / trade \_ config

11. 【推荐】库名与应用名称尽量一致。

12. 【推荐】如果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。

13. 【推荐】字段允许适当冗余，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：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存储类目名称，避免关联查询。

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

说明：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。

## 索引规约

1. 【强制】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。

3. 【强制】在 varchar 字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，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。

4. 【强制】页面搜索严禁左模糊或者全模糊，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。

5. 【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，并且要同为升序或同为降序，还要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。

6. 【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 。

7. 【推荐】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

8. 【推荐】 SQL 性能优化的目标：至少要达到 range 级别，要求是 ref 级别，如果可以是 consts最好。

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，与全表扫描是小巫见大巫。

9. 【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。

10. 【推荐】防止因字段类型不同造成的隐式转换，导致索引失效。

11. 【参考】创建索引时避免有如下极端误解：

1 ） 宁滥勿缺。认为一个查询就需要建一个索引。

2 ） 宁缺勿滥。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。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 语句规约

1. 【强制】超过三个表禁止 join 。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 也要注意驱动表选取、表索引、SQL性能。

2. 【强制】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。

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

col 1, col 2 ) 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。

4. 【强制】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;

5. 【强制】使用 ISNULL() 来判断是否为 NULL 值。

说明： NULL 与任何值的直接比较都为 NULL。

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 。

6 【强制】 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。

7. 【强制】数据订正时，删除和修改记录时，要先 select ，避免出现误删除，确认无误才能执行更新语句。

8. 【强制】 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。严禁in子查询。

9. 【强制】严禁使用or，or可以改写为in()或union，实在不行可以拆成几条查询

* 10. 【参考】若无需对结果进行去重，则用UNION ALL 代替union

11. 【参考】 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少，但 TRUNCATE无事务且不触发 trigger ，有可能造成事故，故不建议在开发代码中使用此语句。

说明： TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同。

## ORM 映射

1. 【强制】在表查询中，一律不要使用 \* 作为查询的字段列表，需要哪些字段必须明确写明。

说明：1 ） 增加查询分析器解析成本。2 ） 增减字段容易与 resultMap 配置不一致。

2. 【强制】 POJO 类的 布尔 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。

说明：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。

3. 【强制】严禁在mybatis注解里写sql或动态sql，sql一律写在xml文件里

说明：注解里写动态sql无缩进阅读困难，IDE也无法校验xml的正确性；mybatis外代码生成工具可以很方便生成基本的xml文件。

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

说明：

resultClass=”Hashtable” ，会置入字段名和属性值，但是值的类型不可控。

5. 【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与之对应。

说明：配置映射关系，使字段与 DO 类解耦，方便维护。

6. 【强制】sql. xml 配置参数使用：#{}，# param # 不要使用${} 此种方式容易出现 SQL 注入。

7. 【强制】 iBATIS 自带的 queryForList(String statementName , int start , int size) 不推

荐使用。

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。

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

map.put("start", start);

map.put("size", size);

8. 【强制】更新数据表记录时，必须同时更新记录对应的 gmt \_ modified 字段值为当前时间。

9. 【推荐】不要写一个大而全的数据更新接口。传入为 POJO 类，不管是不是自己的目标更新字段，都进行 update table set c1=value1,c2=value2,c3=value3; 这是不对的。执行 SQL

时，不要更新无改动的字段，一是易出错 ； 二是效率低 ； 三是增加 binlog 存储。

10. 【参考】@ Transactional 事务不要滥用。事务会影响数据库的 QPS ，另外使用事务的地方需要考虑各方面的回滚方案，包括缓存回滚、搜索引擎回滚、消息补偿、统计修正等。

11. 【参考】< isEqual >中的 compareValue 是与属性值对比的常量，一般是数字，表示相等时带上此条件 ； < isNotEmpty >表示不为空且不为 null 时执行 ； < isNotNull >表示不为 null 值时执行