# **[MYSQL规约](http://blog.csdn.net/qq_19701185/article/details/55051126)**

<http://blog.csdn.net/qq_19701185/article/details/55051126>

**一) 建表规约**

1. 【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），此规则同样适用于odps建表。

 说明：任何字段如果为非负数，必须是unsigned。

2. 【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。

  正例：getter\_admin，task\_config，level3\_name

  反例：GetterAdmin，taskConfig，level\_3\_name

3. 【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。

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

5. 【强制】唯一索引名为uk\_字段名；普通索引名则为idx\_字段名。

  说明：uk\_即 unique key；idx\_即index的简称。

6. 【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。

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, create\_time, update\_time。

  说明：其中id必为主键，类型为unsigned bigint、单表时自增、步长为1；分表时改为从TDDL Sequence取值，确保分表之间的全局唯一。create\_time,update\_time的类型均为date\_time类型。

10.【推荐】表的命名最好是加上“业务名称\_表的作用”，避免上云梯后，再与其它业务表关联

  时有混淆。

  正例：tiger\_task / tiger\_reader / mpp\_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，以提高性能，但是必须考虑数据同步的情况。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：各业务线经常冗余存储商品名称，避免查询时需要调用IC服务获取。

14.【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。

  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。

  反例：某业务三年总数据量才2万行，却分成1024张表，问：你为什么这么设计？答：分1024

  张表，不是标配吗？

15.【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：人的年龄用unsignedtinyint（表示范围0-255，人的寿命不会超过255岁）；海龟就

  必须是smallint，但如果是太阳的年龄，就必须是int；如果是所有恒星的年龄都加起来，那

  么就必须使用bigint。

**(二) 索引规约**

1. 【强制】业务上具有唯一特性的字段，即使是组合字段，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和控制，只要没有唯一索引，根据墨菲定律，

  必然有脏数据产生。

2. 【强制】超过三个表禁止join。需要join的字段，数据类型保持绝对一致；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。

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。

  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，来避免回表操作。

  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：IDB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1a, (selectid 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wherea=?andb=? ，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：wherea>?

      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.【参考】创建索引时避免有如下极端误解：

  1）误认为一个查询就需要建一个索引。

  2）误认为索引会消耗空间、严重拖慢更新和新增速度。

  3）误认为唯一索引一律需要在应用层通过“先查后插”方式解决。

**(三) SQL规约**

1. 【强制】不要使用count(列名)或count(常量)来替代count(\*)，count(\*)就是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。

2. 【强制】count(distinct col) 计算该列除NULL之外的不重复数量。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。

3. 【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：SELECTIF(ISNULL(SUM(g)),0,SUM(g))FROM

     table;

4. 【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，不是false。

  2） NULL=NULL的返回结果是NULL，不是true。

  3） NULL<>1的返回结果是NULL，而不是true。

5. 【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。

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：（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。

7. 【强制】禁止使用存储过程，存储过程难以调试和扩展，更没有移植性。

8. 【强制】IDB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

  提交执行。

9. 【推荐】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。

10.【参考】因阿里巴巴全球化需要，所有的字符存储与表示，均以utf-8编码，那么字符计数方

  法注意：

  说明：

      SELECT LENGTH("阿里巴巴")； 返回为12

      SELECT CHARACTER\_LENGTH("阿里巴巴")； 返回为4

  如果要使用表情，那么使用utfmb4来进行存储，注意它与utf-8编码。

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。

**(四) ORM规约**

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。

  说明：参见定义POJO类以及数据库字段定义规定，在sql.xml增加映射，是必须的。

3. 【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。

4. 【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。

5. 【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，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);

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

  反例：某同学为避免写一个<resultMap>，直接使用HashTable来接收数据库返回结果，结果

  出现日常是把bigint转成Long值，而线上由于数据库版本不一样，解析成BigInteger，导

  致线上问题。

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

8. 【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，

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

9. 【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。

10.【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。

### **Mysql索引优化注意**

**1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。**

**2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num is null可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：select id from t where num=0**

**3.应尽量避免在 where 子句中使用!=或<>操作符，否则引擎将放弃使用索引而进行全表扫描。**

**4.应尽量避免在 where 子句中使用or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num=10 or num=20可以这样查询：select id from t where num=10 union all select id from t where num=20**

**5.in 和 not in 也要慎用，否则会导致全表扫描，如：select id from t where num in(1,2,3) 对于连续的数值，能用 between 就不要用 in 了：select id from t where num between 1 and 3**

**6.下面的查询也将导致全表扫描：select id from t where name like '李%'若要提高效率，可以考虑全文检索。**

**7. 如果在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然 而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：select id from t where num=@num可以改为强制查询使用索引：select id from t with(index(索引名)) where num=@num**

**8.应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：select id from t where num/2=100应改为:select id from t where num=100\*2**

**9.应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：select id from t where substring(name,1,3)='abc' ，name以abc开头的id**

**应改为:**

**select id from t where name like 'abc%'**

**10.不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。**

**11.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。**

**12.不要写一些没有意义的查询，如需要生成一个空表结构：select col1,col2 into #t from t where 1=0**

**这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：**

**create table #t(...)**

**13.很多时候用 exists 代替 in 是一个好的选择：select num from a where num in(select num from b)**

**用下面的语句替换：**

**select num from a where exists(select 1 from b where num=a.num)**

**14.并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。**

**15. 索引并不是越多越好，索引固然可 以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。**

**16. 应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。**

**17.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。**

**18.尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。**

**19.任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。**

**20.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。**

**21.避免频繁创建和删除临时表，以减少系统表资源的消耗。**

**22.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。**

**23.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。**

**24.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。**

**25.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。**

**26.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。**

**27. 与临时表一样，游标并不是不可使 用。对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时 间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。**

**28.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送DONE\_IN\_PROC 消息。**

**29.尽量避免大事务操作，提高系统并发能力。**

**30.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。**