MySQL库表设计规范  
  
表设计  
  
1)     表必须定义主键，默认为ID，整型自增，如果不采用默认设计必须咨询DBA进行设计评估  
2)    ID字段作为自增主键，禁止在非事务内作为上下文作为条件进行数据传递  
3)     禁止使用外键  
4)     多表中的相同列，必须保证列定义一致  
5)    国内表默认使用InnoDB，表字符集默认使用gbk，国际默认使用utf8的表  
6)     表必须包含gmt\_create和gmt\_modified字段，即表必须包含记录创建时间和修改时间的字段  
7)     单表一到两年内数据量超过500w或数据容量超过10G考虑分表，且需要提前考虑历史数据迁移或应用自行删除历史数据  
8)    单条记录大小禁止超过8k（列长度(中文)\*2（gbk）/3(utf8)+列长度(英文)\*1）  
9)    日志类数据不建议存储在MySQL上，优先考虑Hbase或OB，如需要存储请找DBA评估使用压缩表存储  
字段设计  
  
1)     表被索引列必须定义为not null，并设置default值  
2)     禁止使用float、double类型，建议使用decimal或者int替代  
3)     禁止使用blob、text类型保留大文本、文件、图片，建议使用其他方式存储（TFS/SFS），MySQL只保存指针信息  
4)    禁止使用varchar类型作为主键语句设计  
语句设计  
  
    1)    数据更新建议使用二级索引先查询出主键，再根据主键进行数据更新  
    2)    禁止使用非同类型的列进行等值查询！  
其他  
  
1)     禁止使用：存储过程、触发器、函数、视图、事件等MySQL高级功能  
2)     禁止使用跨库查询  
3)     禁止使用子查询，建议将子查询转换成关联查询  
4)     禁止核心业务流程SQL包含：计算操作、多表关联、表遍历case when等复杂查询，建议拆分成单表简单查询  
5)    varchar长度设计需要根据业务实际需要进行长度控制，禁止预留过长空间。例如status使用varchar(128)进行'''''''''

MYSQL数据库设计规范

    1、数据库命名规范

        采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成;

        命名简洁明确(长度不能超过30个字符);

        例如：user, stat, log, 也可以wifi\_user, wifi\_stat, wifi\_log给数据库加个前缀;

        除非是备份数据库可以加0-9的自然数：user\_db\_20151210;

    2、数据库表名命名规范

        采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成;

        命名简洁明确,多个单词用下划线'\_'分隔;

        例如：user\_login, user\_profile, user\_detail, user\_role, user\_role\_relation,

            user\_role\_right, user\_role\_right\_relation

        表前缀'user\_'可以有效的把相同关系的表显示在一起;

    3、数据库表字段名命名规范

        采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成;

        命名简洁明确,多个单词用下划线'\_'分隔;

        例如：user\_login表字段 user\_id, user\_name, pass\_word, eamil, tickit, status, mobile, add\_time;

        每个表中必须有自增主键,add\_time(默认系统时间)

        表与表之间的相关联字段名称要求尽可能的相同;

    4、数据库表字段类型规范

        用尽量少的存储空间来存数一个字段的数据;

        例如：能使用int就不要使用varchar、char,能用varchar(16)就不要使用varchar(256);

        IP地址最好使用int类型;

        固定长度的类型最好使用char,例如：邮编;

        能使用tinyint就不要使用smallint,int;

        最好给每个字段一个默认值,最好不能为null;

    5、数据库表索引规范

        命名简洁明确,例如：user\_login表user\_name字段的索引应为user\_name\_index唯一索引;

        为每个表创建一个主键索引;

        为每个表创建合理的索引;

        建立复合索引请慎重;

    6、简单熟悉数据库范式

        第一范式(1NF)：字段值具有原子性,不能再分(所有关系型数据库系统都满足第一范式);

            例如：姓名字段,其中姓和名是一个整体,如果区分姓和名那么必须设立两个独立字段;

        第二范式(2NF)：一个表必须有主键,即每行数据都能被唯一的区分;

            备注：必须先满足第一范式;

        第三范式(3NF)：一个表中不能包涵其他相关表中非关键字段的信息,即数据表不能有沉余字段;

            备注：必须先满足第二范式;

        备注：往往我们在设计表中不能遵守第三范式,因为合理的沉余字段将会给我们减少join的查询;

              例如：相册表中会添加图片的点击数字段,在相册图片表中也会添加图片的点击数字段;

MYSQL数据库设计原则

    1、核心原则

        不在数据库做运算;

        cpu计算务必移至业务层;

        控制列数量(字段少而精,字段数建议在20以内);

        平衡范式与冗余(效率优先；往往牺牲范式)

        拒绝3B(拒绝大sql语句：big sql、拒绝大事物：big transaction、拒绝大批量：big batch);

    2、字段类原则

        用好数值类型(用合适的字段类型节约空间);

        字符转化为数字(能转化的最好转化,同样节约空间、提高查询性能);

        避免使用NULL字段(NULL字段很难查询优化、NULL字段的索引需要额外空间、NULL字段的复合索引无效);

        少用text类型(尽量使用varchar代替text字段);

    3、索引类原则

        合理使用索引(改善查询,减慢更新,索引一定不是越多越好);

        字符字段必须建前缀索引;

        不在索引做列运算;

        innodb主键推荐使用自增列(主键建立聚簇索引,主键不应该被修改,字符串不应该做主键)(理解Innodb的索引保存结构就知道了);

        不用外键(由程序保证约束);

    4、sql类原则

        sql语句尽可能简单(一条sql只能在一个cpu运算,大语句拆小语句,减少锁时间,一条大sql可以堵死整个库);

        简单的事务;

        避免使用trig/func(触发器、函数不用客户端程序取而代之);

        不用select \*(消耗cpu,io,内存,带宽,这种程序不具有扩展性);

        OR改写为IN(or的效率是n级别);

        OR改写为UNION(mysql的索引合并很弱智);

            select id from t where phone = ’159′ or name = ‘john’;

            =>

            select id from t where phone=’159′

            union

            select id from t where name=’jonh’

        避免负向%;

        慎用count(\*);

        limit高效分页(limit越大，效率越低);

        使用union all替代union(union有去重开销);

        少用连接join;

        使用group by;

        请使用同类型比较;

        打散批量更新;

    5、性能分析工具

        show profile;

        mysqlsla;

        mysqldumpslow;

        explain;

        show slow log;

        show processlist;

1.命名规范

**1.库名、表名、字段名必须使用小写字母，并采用下划线分割**。   
a)[**MySQL**](http://lib.csdn.net/base/mysql)有配置参数lower\_case\_table\_names，不可动态更改，[**Linux**](http://lib.csdn.net/base/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。   
b)如果大小写混合使用，可能存在abc，Abc，ABC等多个表共存，容易导致混乱。   
c)字段名显示区分大小写，但实际使⽤用不区分，即不可以建立两个名字一样但大小写不一样的字段。   
d)为了统一规范， 库名、表名、字段名使用小写字母。

**2.库名、表名、字段名禁止超过32个字符。**   
库名、表名、字段名支持最多64个字符，但为了统一规范、易于辨识以及减少传输量，禁止超过32个字符。

**3.使用INNODB存储引擎。**   
INNODB引擎是MySQL5.5版本以后的默认引擘，支持事务、行级锁，有更好的数据恢复能力、更好的并发性能，同时对多核、大内存、SSD等硬件支持更好，支持数据热备份等，因此INNODB相比MyISAM有明显优势。

**4.库名、表名、字段名禁止使用**[**mysql**](http://lib.csdn.net/base/mysql)**保留字。**   
当库名、表名、字段名等属性含有保留字时，SQL语句必须用反引号引用属性名称，这将使得SQL语句书写、SHELL脚本中变量的转义等变得⾮非常复杂。

**5.禁止使用分区表。**   
分区表对分区键有严格要求；分区表在表变大后，执⾏行DDL、SHARDING、单表恢复等都变得更加困难。因此禁止使用分区表，并建议业务端手动SHARDING。

**6.建议使用UNSIGNED存储非负数值。**   
同样的字节数，非负存储的数值范围更大。如TINYINT有符号为 -128-127，无符号为0-255。

**7.建议使用INT UNSIGNED存储IPV4。**   
用UNSINGED INT存储IP地址占用4字节，CHAR(15)则占用15字节。另外，计算机处理整数类型比字符串类型快。使用INT UNSIGNED而不是CHAR(15)来存储IPV4地址，通过MySQL函数inet\_ntoa和inet\_aton来进行转化。IPv6地址目前没有转化函数，需要使用DECIMAL或两个BIGINT来存储。   
例如:

SELECT INET\_ATON('209.207.224.40'); 3520061480

SELECT INET\_NTOA(3520061480); 209.207.224.40

* 1
* 2
* 1
* 2

**8.强烈建议使用TINYINT来代替ENUM类型。**   
ENUM类型在需要修改或增加枚举值时，需要在线DDL，成本较高；ENUM列值如果含有数字类型，可能会引起默认值混淆。

**9.使用VARBINARY存储大小写敏感的变长字符串或二进制内容。**   
VARBINARY默认区分大小写，没有字符集概念，速度快。

**10.INT类型固定占用4字节存储**   
例如INT(4)仅代表显示字符宽度为4位，不代表存储长度。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**](http://lib.csdn.net/base/python)、[**Java**](http://lib.csdn.net/base/java)客户端等不具备这个功能。

**11.区分使用DATETIME和TIMESTAMP。**   
存储年使用YEAR类型。存储日期使用DATE类型。 存储时间(精确到秒)建议使用TIMESTAMP类型。   
DATETIME和TIMESTAMP都是精确到秒，优先选择TIMESTAMP，因为TIMESTAMP只有4个字节，而DATETIME8个字节。同时TIMESTAMP具有自动赋值以及⾃自动更新的特性。注意：在5.5和之前的版本中，如果一个表中有多个timestamp列，那么最多只能有一列能具有自动更新功能。

如何使用TIMESTAMP的自动赋值属性?   
a)自动初始化，而且自动更新：

column1 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATECURRENT\_TIMESTAMP

* 1
* 1

b)只是自动初始化：

column1 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

* 1
* 1

c)自动更新，初始化的值为0：

column1 TIMESTAMP DEFAULT 0 ON UPDATE CURRENT\_TIMESTAMP

* 1
* 1

d)初始化的值为0：

column1 TIMESTAMP DEFAULT 0

* 1
* 1

**12.所有字段均定义为NOT NULL。**   
a)对表的每一行，每个为NULL的列都需要额外的空间来标识。   
b)B树索引时不会存储NULL值，所以如果索引字段可以为NULL，索引效率会下降。   
c)建议用0、特殊值或空串代替NULL值。

MySQL使用技巧

**1.将大字段、访问频率低的字段拆分到单独的表中存储，分离冷热数据。**   
有利于有效利用缓存，防⽌止读入无用的冷数据，较少磁盘IO，同时保证热数据常驻内存提⾼高缓存命中率。

**2.禁止在**[**数据库**](http://lib.csdn.net/base/mysql)**中存储明文密码。**   
采用加密字符串存储密码，并保证密码不可解密，同时采用随机字符串加盐保证密码安全。

**3.表必须有主键，推荐使用UNSIGNED自增列作为主键。**   
表没有主键，INNODB会默认设置隐藏的主键列；没有主键的表在定位数据行的时候非常困难，也会降低基于行复制的效率。

**4.禁止冗余索引。**   
索引是双刃剑，会增加维护负担，增⼤大IO压力。(a,b,c)、(a,b)，后者为冗余索引。可以利用前缀索引来达到加速目的，减轻维护负担。

**5.禁止重复索引。**   
primary key a;uniq index a;重复索引增加维护负担、占用磁盘空间，同时没有任何益处。

**6.不在低基数列上建立索引，例如“性别”。**   
大部分场景下，低基数列上建立索引的精确查找，相对于不建立索引的全表扫描没有任何优势，而且增大了IO负担。

**7.合理使用覆盖索引减少IO，避免排序。**   
覆盖索引能从索引中获取需要的所有字段，从⽽而避免回表进行二次查找，节省IO。   
INNODB存储引擎中，secondary index(非主键索引，又称为辅助索引、二级索引)没有直接存储行地址，而是存储主键值。   
如果用户需要查询secondary index中所不包含的数据列，则需要先通过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。覆盖索引则可以在⼀一个索引中获取所有需要的数据，因此效率较高。   
例如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。

**8.用IN代替OR。SQL语句中IN包含的值不应过多，应少于1000个。**   
IN是范围查找，MySQL内部会对IN的列表值进行排序后查找，比OR效率更高。

**9.表字符集使用UTF8，必要时可申请使用UTF8MB4字符集。**   
a)UTF8字符集存储汉字占用3个字节，存储英文字符占用一个字节。   
b)UTF8统一而且通用，不会出现转码出现乱码风险。   
c)如果遇到EMOJ等表情符号的存储需求，可申请使用UTF8MB4字符集。

**10.用UNION ALL代替UNION。**   
UNION ALL不需要对结果集再进行排序。

**11.禁止使用order by rand()。**   
order by rand()会为表增加一个伪列，然后用rand()函数为每一行数据计算出rand()值，然后基于该行排序，这通常都会生成磁盘上的临时表，因此效率非常低。建议先使用rand()函数获得随机的主键值，然后通过主键   
获取数据。

**12.建议使用合理的分页方式以提高分页效率。**   
假如有类似下面分页语句:

SELECT \* FROM table ORDER BY TIME DESC LIMIT 10000，10;

* 1
* 1

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)

* 1
* 2
* 3
* 1
* 2
* 3

**13.SELECT只获取必要的字段，禁⽌止使用SELECT \*。**   
减少网络带宽消耗；   
能有效利用覆盖索引；   
表结构变更对程序基本无影响。

**14.SQL中避免出现now()、rand()、sysdate()、current\_user()等不确定结果的函数。**   
语句级复制场景下，引起主从数据不一致；不确定值的函数，产⽣生的SQL语句无法利用QUERY CACHE。

**15.采用合适的分库分表策略。例如千库十表、十库百表等。**   
采用合适的分库分表策略，有利于业务发展后期快速对数据库进行水平拆分，同时分库可以有效利⽤用MySQL   
的多线程复制特性。

**16.减少与数据库交互次数，尽量采用批量SQL语句。**   
使用下面的语句来减少和db的交互次数:

a)INSERT ... ON DUPLICATE KEY UPDATE

b)REPLACE INTO

c)INSERT IGNORE

d)INSERT INTO VALUES()

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

**17.拆分复杂SQL为多个小SQL，避免大事务。**   
简单的SQL容易使⽤用到MySQL的QUERY CACHE；减少锁表时间特别是MyISAM；可以使用多核 CPU。

**18.对同一个表的多次alter操作必须合并为一次操作。**   
mysql对表的修改绝大部分操作都需要锁表并重建表，而锁表则会对线上业务造成影响。为减少这种影响，必须把对表的多次alter操作合并为一次操作。例如，要给表t增加一个字段b，同时给已有的字段aa建立索引，   
通常的做法分为两步：

alter table t add column b varchar(10);

* 1
* 1

然后增加索引：

alter table t add index idx\_aa(aa);

* 1
* 1

正确的做法是：

alter table t add column b varchar(10),add index idx\_aa(aa);

* 1
* 1

**19.避免使用存储过程、触发器、视图、自定义函数等。**   
这些高级特性有性能问题，以及未知BUG较多。业务逻辑放到数据库会造成数据库的DDL、SCALE OUT、   
SHARDING等变得更加困难。

**20.禁止有super权限的应用程序账号存在。**   
安全第一。super权限会导致read only失效，导致较多诡异问题而且很难追踪。

**21.不要在MySQL数据库中存放业务逻辑。**   
数据库是有状态的服务，变更复杂而且速度慢，如果把业务逻辑放到数据库中，将会限制业务的快速发展。建议把业务逻辑提前，放到前端或中间逻辑层，而把数据库作为存储层，实现逻辑与存储的分离。

--------------------

# [MYSQL数据库命名及设计规范](http://www.cnblogs.com/amboyna/archive/2008/02/18/1072007.html)

1.设计原则

1) 标准化和规范化  
数据的标准化有助于消除数据库中的数据冗余。标准化有好几种形式，但Third Normal Form（3NF）通常被认为在性能、扩展性和数据完整性方面达到了最好平衡。简单来说，遵守3NF 标准的数据库的表设计原则是：“One Fact in One Place”即某个表只包括其本身基本的属性，当不是它们本身所具有的属性时需进行分解。表之间的关系通过外键相连接。它具有以下特点：有一组表专门存放通过键连接起来的关联数据。  
举例：某个存放客户及其有关定单的3NF 数据库就可能有两个表：Customer和Order。Order表不包含定单关联客户的任何信息，但表内会存放一个键值，该键指向Customer表里包含该客户信息的那一行。  
事实上，为了效率的缘故，对表不进行标准化有时也是必要的。  
2) 数据驱动  
采用数据驱动而非硬编码的方式，许多策略变更和维护都会方便得多，大大增强系统的灵活性和扩展性。  
举例，假如用户界面要访问外部数据源（文件、XML 文档、其他数据库等），不妨把相应的连接和路径信息存储在用户界面支持表里。还有，如果用户界面执行工作流之类的任务（发送邮件、打印信笺、修改记录状态等），那么产生工作流的数据也可以存放在数据库里。角色权限管理也可以通过数据驱动来完成。事实上，如果过程是数据驱动的，你就可以把相当大的责任推给用户，由用户来维护自己的工作流过程。  
3) 考虑各种变化  
在设计数据库的时候考虑到哪些数据字段将来可能会发生变更。  
举例，姓氏就是如此（注意是西方人的姓氏，比如女性结婚后从夫姓等）。所以，在建立系统存储客户信息时，在单独的一个数据表里存储姓氏字段，而且还附加起始日和终止日等字段，这样就可以跟踪这一数据条目的变化。

2.数据库涉及字符规范

采用26个英文字母(区分大小写)和0-9这十个自然数,加上下划线'\_'组成,共63个字符.不能出现其他字符(注释除外).  
注意事项:  
1) 以上命名都不得超过30个字符的系统限制.变量名的长度限制为29(不包括标识字符@).  
2) 数据对象、变量的命名都采用英文字符,禁止使用中文命名.绝对不要在对象名的字符之间留空格.  
3) 小心保留词,要保证你的字段名没有和保留词、数据库系统或者常用访问方法冲突  
5) 保持字段名和类型的一致性,在命名字段并为其指定数据类型的时候一定要保证一致性.假如数据类型在一个表里是整数,那在另一个表里可就别变成字符型了.

3.数据库命名规范

数据库,数据表一律使用前缀

正式数据库名使用小写英文以及下划线组成,尽量说明是那个应用或者系统在使用的.比如:

web\_19floor\_net  
web\_car

备份数据库名使用正式库名加上备份时间组成,如:

web\_19floor\_net\_20070403  
web\_car\_20070403

4.数据库表命名规范

数据表名使用小写英文以及下划线组成,尽量说明是那个应用或者系统在使用的.  
相关应用的数据表使用同一前缀,如论坛的表使用cdb\_前缀,博客的数据表使用supe\_前缀,前缀名称一般不超过5字  
比如:

web\_user  
web\_group  
supe\_userspace

备份数据表名使用正式表名加上备份时间组成,如:

web\_user\_20070403  
web\_group\_20070403  
supe\_userspace\_20070403

5.字段命名规范

字段名称使用单词组合完成,首字母小写,后面单词的首字母大写,最好是带表名前缀.  
如 web\_user 表的字段:

userId  
userName  
userPassword

表与表之间的相关联字段要用统一名称,  
如 web\_user 表里面的 userId 和 web\_group 表里面的 userId 相对应

6.字段类型规范  
规则:用尽量少的存储空间来存数一个字段的数据.  
比如能用int的就不用char或者varchar  
能用tinyint的就不用int  
能用varchar(20)的就不用varchar(255)  
时间戳字段尽量用int型，如created:表示从'1970-01-01 08:00:00'开始的int秒数，采用英文单词的过去式；gmtCreated:表示datetime类型的时间，即形如'1980-01-01 00:00:00'的时间串，Java中对应的类型为Timestamp

7.数据库设计文档规范

所有数据库设计要写成文档,文档以模块化形式表达.大致格式如下：

'-------------------------------------------  
'    表名：    web\_user  
'    作者：    Aeolus(傻鱼)  
'    日期：    2007-04-11  
'    版本：    1.0  
'    描述：    保存用户资料  
'    具体内容：  
'    UserID  int,自动增量  用户代码  
'    UserName char(12)  用户名字  
'    ......  
'--------------------------------------------

8.索引使用原则：

1) 逻辑主键使用唯一的成组索引,对系统键(作为存储过程)采用唯一的非成组索引,对任何外键列采用非成组索引.考虑数据库的空间有多大,表如何进行访问,还有这些访问是否主要用作读写.  
2) 大多数数据库都索引自动创建的主键字段,但是可别忘了索引外键,它们也是经常使用的键,比如运行查询显示主表和所有关联表的某条记录就用得上.  
3) 不要索引blob/text等字段,不要索引大型字段(有很多字符),这样作会让索引占用太多的存储空间.  
4) 不要索引常用的小型表  
不要为小型数据表设置任何键,假如它们经常有插入和删除操作就更别这样作了.对这些插入和删除操作的索引维护可能比扫描表空间消耗更多的时间.

9.sql语句规范

所有sql关键词全部大写,比如SELECT,UPDATE,FROM,ORDER,BY等,所有的表名和库名都要用``包含  
如:  
SELECT COUNT(\*) FROM `cdb\_members` WHERE `userName` = 'aeolus';

10.其他设计技巧  
1) 避免使用触发器  
触发器的功能通常可以用其他方式实现.在调试程序时触发器可能成为干扰.假如你确实需要采用触发器,你最好集中对它文档化.  
2) 使用常用英语(或者其他任何语言)而不要使用编码或者拼音首字母缩写  
在创建下拉菜单、列表、报表时最好按照英语名排序.假如需要编码或者拼音首字母缩写,可以在旁边附上用户知道的英语.  
3) 保存常用信息  
让一个表专门存放一般数据库信息非常有用.在这个表里存放数据库当前版本、最近检查/修复(对Access)、关联设计文档的名称、客户等信息.这样可以实现一种简单机制跟踪数据库,当客户抱怨他们的数据库没有达到希望的要求而与你联系时,这样做对非客户机/服务器环境特别有用.  
4) 包含版本机制  
在数据库中引入版本控制机制来确定使用中的数据库的版本.时间一长,用户的需求总是会改变的.最终可能会要求修改数据库结构.把版本信息直接存放到数据库中更为方便.   
5) 编制文档  
对所有的快捷方式、命名规范、限制和函数都要编制文档.  
采用给表、列、触发器等加注释的数据库工具.对开发、支持和跟踪修改非常有用.  
对数据库文档化,或者在数据库自身的内部或者单独建立文档.这样,当过了一年多时间后再回过头来做第2 个版本,犯错的机会将大大减少.  
6) 测试、测试、反复测试  
建立或者修订数据库之后,必须用用户新输入的数据测试数据字段.最重要的是,让用户进行测试并且同用户一道保证选择的数据类型满足商业要求.测试需要在把新数据库投入实际服务之前完成.  
7) 检查设计  
在开发期间检查数据库设计的常用技术是通过其所支持的应用程序原型检查数据库.换句话说,针对每一种最终表达数据的原型应用,保证你检查了数据模型并且查看如何取出数据.

**MYSQL数据库命名及设计规范**

http://www.cnblogs.com/amboyna/archive/2008/02/18/1072007.html

**设计原则**

1) 标准化和规范化

数据的标准化有助于消除数据库中的数据冗余。标准化有好几种形式，但Third Normal Form（3NF）通常被认为在性能、扩展性和数据完整性方面达到了最好平衡。简单来说，遵守3NF 标准的数据库的表设计原则是：“One Fact in One Place”即某个表只包括其本身基本的属性，当不是它们本身所具有的属性时需进行分解。表之间的关系通过外键相连接。它具有以下特点：有一组表专门存放通过键连接起来的关联数据。

举例：某个存放客户及其有关定单的3NF 数据库就可能有两个表：Customer和Order。Order表不包含定单关联客户的任何信息，但表内会存放一个键值，该键指向Customer表里包含该客户信息的那一行。

事实上，为了效率的缘故，对表不进行标准化有时也是必要的。

2) 数据驱动

采用数据驱动而非硬编码的方式，许多策略变更和维护都会方便得多，\*\*增强系统的灵活性和扩展性。

举例，假如用户界面要访问外部数据源（文件、XML 文档、其他数据库等），不妨把相应的连接和路径信息存储在用户界面支持表里。还有，如果用户界面执行工作流之类的任务（发送邮件、打印信笺、修改记录状态等），那么产生工作流的数据也可以存放在数据库里。角色权限管理也可以通过数据驱动来完成。事实上，如果过程是数据驱动的，你就可以把相当大的责任推给用户，由用户来维护自己的工作流过程。

3) 考虑各种变化

在设计数据库的时候考虑到哪些数据字段将来可能会发生变更。

举例，姓氏就是如此（注意是西方人的姓氏，比如女性结婚后从夫姓等）。所以，在建立系统存储客户信息时，在单独的一个数据表里存储姓氏字段，而且还附加起始日和终止日等字段，这样就可以跟踪这一数据条目的变化。

**数据库命名规范**

数据库,数据表一律使用前缀

正式数据库名使用小写英文以及下划线组成,尽量说明是那个应用或者系统在使用的.比如:

web\_19floor\_net

web\_car

备份数据库名使用正式库名加上备份时间组成,如:

web\_19floor\_net\_20070403

web\_car\_20070403

**数据库表命名规范**

数据表名使用小写英文以及下划线组成,尽量说明是那个应用或者系统在使用的.

相关应用的数据表使用同一前缀,如论坛的表使用cdb\_前缀,博客的数据表使用supe\_前缀,前缀名称一般不超过5字

比如:

web\_user

web\_group

supe\_userspace

备份数据表名使用正式表名加上备份时间组成,如:

web\_user\_20070403

web\_group\_20070403

supe\_userspace\_20070403

**字段命名规范**

字段名称使用单词组合完成,首字母小写,后面单词的首字母大写,最好是带表名前缀.

如 web\_user 表的字段:

userId

userName

userPassword

表与表之间的相关联字段要用统一名称,

如 web\_user 表里面的 userId 和 web\_group 表里面的 userId 相对应

**字段类型规范**

规则:用尽量少的存储空间来存数一个字段的数据.

**比如能用int的就不用char或者varchar**

能用tinyint的就不用int

能用varchar(20)的就不用varchar(255)

时间戳字段尽量用int型，如created:表示从'1970-01-01 08:00:00'开始的int秒数，采用英文单词的过去式；gmtCreated:表示datetime类型的时间，即形如'1980-01-01 00:00:00'的时间串，Java中对应的类型为Timestamp

**数据库设计文档规范**

所有数据库设计要写成文档,文档以模块化形式表达.大致格式如下：

*# 表名：    spider\_organization*

*# 作者：    青枫*

*# 日期：    2016-08-10*

*# 版本：    1.0*

*# 描述：    存储考试院,985,211高校网站的domain,编码,正文关键词,标题关键词等*

*# 具体内容：*

*# organizationid(自动增长,主键id)*

**索引使用原则：**

1) 逻辑主键使用唯一的成组索引,对系统键(作为存储过程)采用唯一的非成组索引,对任何外键列采用非成组索引.考虑数据库的空间有多大,表如何进行访问,还有这些访问是否主要用作读写.

2) 大多数数据库都索引自动创建的主键字段,但是可别忘了索引外键,它们也是经常使用的键,比如运行查询显示主表和所有关联表的某条记录就用得上.

3) 不要索引blob/text等字段,不要索引大型字段(有很多字符),这样作会让索引占用太多的存储空间.

4) 不要索引常用的小型表

不要为小型数据表设置任何键,假如它们经常有\*\*和删除操作就更别这样作了.对这些\*\*和删除操作的索引维护可能比扫描表空间消耗更多的时间.

**sql语句规范**

所有sql关键词全部大写,比如SELECT,UPDATE,FROM,ORDER,BY等,所有的表名和库名都要用``包含

如:

SELECT COUNT(\*) FROM `cdb\_members` WHERE `userName` = 'aeolus';

**其他设计技巧**

1) 避免使用触发器

触发器的功能通常可以用其他方式实现.在调试程序时触发器可能成为干扰.假如你确实需要采用触发器,你最好集中对它文档化.

2) 使用常用英语(或者其他任何语言)而不要使用编码或者拼音首字母缩写

在创建下拉菜单、列表、报表时最好按照英语名排序.假如需要编码或者拼音首字母缩写,可以在旁边附上用户知道的英语.

3) 保存常用信息

让一个表专门存放一般数据库信息非常有用.在这个表里存放数据库当前版本、最近检查/修复(对Access)、关联设计文档的名称、客户等信息.这样可以实现一种简单机制跟踪数据库,当客户抱怨他们的数据库没有达到希望的要求而与你联系时,这样做对非客户机/服务器环境特别有用.

4) 包含版本机制

在数据库中引入版本控制机制来确定使用中的数据库的版本.时间一长,用户的需求总是会改变的.最终可能会要求修改数据库结构.把版本信息直接存放到数据库中更为方便.

5) 编制文档

对所有的快捷方式、命名规范、限制和函数都要编制文档.

采用给表、列、触发器等加注释的数据库工具.对开发、支持和跟踪修改非常有用.

对数据库文档化,或者在数据库自身的内部或者单独建立文档.这样,当过了一年多时间后再回过头来做第2 个版本,犯错的机会将\*\*减少.

6) 测试、测试、反复测试

建立或者修订数据库之后,必须用用户新输入的数据测试数据字段.最重要的是,让用户进行测试并且同用户一道保证选择的数据类型满足商业要求.测试需要在把新数据库投入实际服务之前完成.

7) 检查设计

在开发期间检查数据库设计的常用技术是通过其所支持的应用程序原型检查数据库.换句话说,针对每一种最终表达数据的原型应用,保证你检查了数据模型并且查看如何取出数据.

数据库的三个范式

http://www.cnblogs.com/ybwang/archive/2010/06/04/175\*\*\*\*79.html

# 概念：

（1） 实体（entity）：就是实际应用中要用数据描述的事物，一般是名词。

（2） 字段（fields）：就是一项数据，也就是我们平常所说的“列”。

（3） 记录（record）：一个实体的一个实例所特有的相关数据项的集合，也就是我们平常所说的“行”。

（4） 键（key）：可唯一标识一条记录的一个字段或字段集，有时翻译为“码”。

（5） 主键（primary key）：用于唯一标识一个表中的一条记录的键。每个主键应该具有下列特征：

1.  唯一的。

2.最小 的（尽量选择最少键的组合）。

3.非空。

4.不可更新的（不能随时更改）

（6） 外键（foreign keys）：对连接父表和子表的相关记录的主键字段的复制。

（7） 依赖表（dependent table）：也称为弱实体（weak entity）是需要用父表标识的子表。

（8） 关联表（associative table）：是多对多关系中两个父表的子表。

（9） 实体完整性：每个表必须有一个有效的主 键。

（10） 参照完整性：没有不相匹配的外键值。

# 名词解释：

## 函数依赖：

通俗描述：

描述一个学生的关系，可以有学号(SNO),姓名(SNAME),系名(SDEPT)等几个属性。由于一个学号只对应一个学生，一个学生只在一个系学\*\*。因此当学号确定之后，姓名和该学生所在系的值也就唯一被确定了，就像自变量x确定之后，相应的函数值f(x)也就唯一地被确定了一样，称SNO函数决定SNAME和SDEPT，或者说SNAME，SDEPT函数依赖于SNO，记为：SNO -> SNAME， SNO -> SDEPT.

严格定义：

设R(U)是属性集U上的关系模式。X，Y是U的子集。若对于R(U)的任意一个可能的关系r,r中不可能存在两个元组在X上的属性值相等，而在Y上的属性值不相等，则称X函数确定Y或者Y函数依赖于X。记为X->Y。

（如果不知道“关系”、“属性集”等定义，自己看大学教材去。这里的定义摘自萨师煊&王珊《数据库系统概论》第三版）

## 完全函数依赖：

在R(U)中，如果Y函数依赖于X,并且对于X的任何一个真子集X'，都有Y不函数依赖于X'， 则称Y对X完全函数依赖。否则称Y对X部分函数依赖。

举个例子就明白了。假设一个学生有几个属性

SNO 学号

SNAME 姓名

SDEPT 系

SAGE 年龄

CNO 班级号

G 成绩

对于(SNO,SNAME,SDEPT,SAGE,CNO,G)来说，G完全依赖于(SNO, CNO), 因为(SNO,CNO)可以决定G，而SNO和CNO都不能单独决定G。

而SAGE部分函数依赖于(SNO,CNO),因为(SNO,CNO)可以决定SAGE，而单独的SNO也可以决定SAGE。

## 传递函数依赖：

在R(U)中，如果X->Y, Y->Z, 则称Z对X传递函数依赖。

## 候选键

(又称候选码，候选关键字,码 ，candidate key)：

设K是一个R(U)中的属性或属性集合(注意可以是属性集合，也即多个属性的组合)，若K完全函数确定U，则K为R的候选键(Candidate key);

通俗地说就是，能够确定全部属性的某个属性或某组属性，称为候选键。若候选键多于一个，则选定其中一个作为主键。

## 主属性：

包含在任何一个候选键中的属性，叫做主属性(Prime attribute),不包含在任何候选键中的属性称为非主属性或非键属性或非关键字段。

例子：

在(SNO, CNO, G)中，SNO和CNO这俩合起来就是一个候选键，因为每个元组只要确定了SNO和CNO，则其它所有属性都可以根据SNO和CNO来确定。而SNO和CNO就都是“主属性”，G是“非主属性”。由于此例中只有一个候选键，于是只能选择(SNO, CNO)作为主键。

在(SNO,SDEPT, SNAME)中，SNO是一个候选键，因为只要SNO确定了，其它所有属性也都确定了，如果保证没有重名的话，则SNAME也是一个候选键，于是可以选SNO或者SNAME之一作为候选键。如果不能保证没有重名，就不能把SNAME当成候选键，于是就只有SNO能够做主键。

1.第一范式

第一范式：数据库表中的字段都是单一属性的，不可再分。一个列不能有多个值,每一个字段都是不可拆分的.

2.第二范式

第二范式：数据库表中不存在非关键字段对任一候选关键字段的部分函数依赖（部分函数依赖指的是存在组合关键字中的某些字段决定非关键字段的情况），也即所有非关键字段都完全依赖于任意一组候选关键字

第二范式也可以理解为：主键确定一条唯一记录，也就是说关键字在数据库表中唯一出现一次．

3.第三范式

第三范式：在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖则符合第三范式。所谓传递函数依赖，指的是如果存在"A → B → C"的决定关系．

函数依赖：

X—>Y  X确定，则Y确定，X函数确定Y，Y函数依赖于X函数

完全函数依赖：

在R(U)中，如果Y函数依赖于X,并且对于X的任何一个真子集X'，都有Y不函数依赖于X'， 则称Y对X完全函数依赖。否则称Y对X部分函数依赖。

传递函数依赖：

在R(U)中，如果X->Y, Y->Z, 则称Z对X传递函数依赖。

**主键的主要作用是将记录和存放在其他表中的数据进行关联，在这一点上，主键是不同表中各记录间的简单指针，主键约整就是确定表中的每一条记录，主键不能是空值，唯一约束是用于指定一个或多个列的组合值具有唯一性，以防止在列中输入重复的值，所以，主键的值对用户而言是没有什么意义，并且和它赋予的值也没有什么特别联系。**

**将数据装入表中**

创建表后，需要填入内容。通过LOAD DATA和INSERT语句可以完成该任务。

假定你的宠物纪录描述如下。（假定在MySQL中期望的日期格式是YYYY-MM-DD；这可能与你\*\*惯的不同。）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **name** | **owner** | **species** | **sex** | **birth** | **death** |
| Fluffy | Harold | cat | f | 1993-02-04 |  |
| Claws | Gwen | cat | m | 1994-03-17 |  |
| Buffy | Harold | dog | f | 1989-05-13 |  |
| Fang | Benny | dog | m | 1990-08-27 |  |
| Bowser | Diane | dog | m | 1979-08-31 | 1995-07-29 |
| Chirpy | Gwen | bird | f | 1998-09-11 |  |
| Whistler | Gwen | bird |  | 1997-12-09 |  |
| Slim | Benny | snake | m | 1996-04-29 |  |

因为你是从一个空表开始的，填充它的一个简易方法是创建一个文本文件，每个动物各一行，然后用一个语句将文件的内容装载到表中。

你可以创建一个文本文件“pet.txt”，每行包含一个记录，用定位符(tab)把值分开，并且以CREATE TABLE语句中列出的列次序给出。对于丢失的值(例如未知的性别，或仍然活着的动物的死亡日期)，你可以使用NULL值。为了在你的文本文件中表示这些内容，使用\N（反斜线，字母N）。例如，Whistler鸟的记录应为(这里值之间的空白是一个定位符)：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **name** | **owner** | **species** | **sex** | **birth** | **death** |
| Whistler | Gwen | bird | \N | 1997-12-09 | \N |

要想将文本文件“pet.txt”装载到pet表中，使用这个命令：

mysql> **LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;**

请注意如果用Windows中的编辑器（使用\r\n做为行的结束符）创建文件，应使用：

mysql> **LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet**

    -> **LINES TERMINATED BY '\r\n';**

（在运行OS X的Apple机上，应使用行结束符'\r'。）

如果你愿意，你能明确地在LOAD DATA语句中指出列值的分隔符和行尾标记，但是默认标记是定位符和换行符。这对读入文件“pet.txt”的语句已经足够。

-----------

最近一段时间一边在线上抓取SQL来优化，一边在整理这个开发规范，尽量减少新的问题SQL进入生产库。今天也是对公司的开发做了一次培训，PPT就不放上来了，里面有十来个生产SQL的案例。因为规范大部分还是具有通用性，所以也借鉴了像去哪儿和赶集的规范，但实际在撰写本文的过程中，每一条规范的背后无不是在工作中有参照的反面例子的。如果时间可以的话，会抽出一部分或分析其原理，或用案例证明。

## 1. 命名规范

### 库名、表名、字段名必须使用小写字母，并采用下划线分割

* + MySQL有配置参数lower\_case\_table\_names=1，即库表名以小写存储，大小写不敏感。如果是0，则库表名以实际情况存储，大小写敏感；如果是2，以实际情况存储，但以小写比较。
  + 如果大小写混合使用，可能存在abc，Abc，ABC等多个表共存，容易导致混乱。
  + 字段名显示区分大小写，但实际使⽤时不区分，即不可以建立两个名字一样但大小写不一样的字段。
  + 为了统一规范， 库名、表名、字段名使用小写字母。

### 库名以 d 开头，表名以 t 开头，字段名以 f\_ 开头

* + 比如表 t\_crm\_relation，中间的 crm 代表业务模块名
  + 视图以view\_开头，事件以event\_开头，触发器以trig\_开头，存储过程以proc\_开头，函数以func\_开头
  + 普通索引以idx\_col1\_col2命名，唯一索引以uk\_col1\_col2命名（可去掉f\_公共部分）。如 idx\_companyid\_corpid\_contacttime(f\_company\_id,f\_corp\_id,f\_contact\_time)

### 库名、表名、字段名禁止超过32个字符，需见名知意

库名、表名、字段名支持最多64个字符，但为了统一规范、易于辨识以及减少传输量，禁止超过32个字符

### 临时库、表名须以tmp加日期为后缀

如 t\_crm\_relation\_tmp0425。备份表也类似，形如 \_bak20160425 。

### 按日期时间分表须符合\_YYYY[MM][DD]格式

这也是为将来有可能分表做准备的，比如t\_crm\_ec\_record\_201403，但像 t\_crm\_contact\_at201506就打破了这种规范。  
不具有时间特性的，直接以 t\_tbname\_001 这样的方式命名。

## 2. 库表基础规范

### 使用Innodb存储引擎

5.5版本开始mysql默认存储引擎就是InnoDB，5.7版本开始，系统表都放弃MyISAM了。

### 表字符集统一使用UTF8

* + UTF8字符集存储汉字占用3个字节，存储英文字符占用一个字节
  + 校对字符集使用默认的 utf8\_general\_ci
  + 连接的客户端也使用utf8，建立连接时指定charset或SET NAMES UTF8;。（对于已经在项目中长期使用latin1的，救不了了）
  + 如果遇到EMOJ等表情符号的存储需求，可申请使用UTF8MB4字符集

### 所有表都要添加注释

* + 尽量给字段也添加注释
  + 类status型需指明主要值的含义，如”0-离线，1-在线”

### 控制单表字段数量

* + 单表字段数上限30左右，再多的话考虑垂直分表，一是冷热数据分离，二是大字段分离，三是常在一起做条件和返回列的不分离。
  + 表字段控制少而精，可以提高IO效率，内存缓存更多有效数据，从而提高响应速度和并发能力，后续 alter table 也更快。

### 所有表都必须要显式指定主键

* + 主键尽量采用自增方式，InnoDB表实际是一棵索引组织表，顺序存储可以提高存取效率，充分利用磁盘空间。还有对一些复杂查询可能需要自连接来优化时需要用到。
  + 需要全局唯一主键时，使用外部发号器ticket server（建设中）
  + 如果没有主键或唯一索引，update/delete是通过所有字段来定位操作的行，相当于每行就是一次全表扫描
  + 少数情况可以使用联合唯一主键，需与DBA协商

### 不强制使用外键参考

即使2个表的字段有明确的外键参考关系，也不使用 FOREIGN KEY ，因为新纪录会去主键表做校验，影响性能。

### 适度使用存储过程、视图，禁止使用触发器、事件

* 存储过程（procedure）虽然可以简化业务端代码，在传统企业写复杂逻辑时可能会用到，而在互联网企业变更是很频繁的，在分库分表的情况下要升级一个存储过程相当麻烦。又因为它是不记录log的，所以也不方便debug性能问题。如果使用过程，一定考虑如果执行失败的情况。
  + 使用视图一定程度上也是为了降低代码里SQL的复杂度，但有时候为了视图的通用性会损失性能（比如返回不必要的字段）。
* 触发器（trigger）也是同样，但也不应该通过它去约束数据的强一致性，mysql只支持“基于行的触发”，也就是说，触发器始终是针对一条记录的，而不是针对整个sql语句的，如果变更的数据集非常大的话，效率会很低。掩盖一条sql背后的工作，一旦出现问题将是灾难性的，但又很难快速分析和定位。再者需要ddl时无法使用pt-osc工具。放在transaction执行。
* 事件（event）也是一种偷懒的表现，目前已经遇到数次由于定时任务执行失败影响业务的情况，而且mysql无法对它做失败预警。建立专门的 job scheduler 平台。

### 单表数据量控制在5000w以内

### 数据库中不允许存储明文密码

## 3. 字段规范

### char、varchar、text等字符串类型定义

* + 对于长度基本固定的列，如果该列恰好更新又特别频繁，适合char
  + varchar虽然存储变长字符串，但不可太小也不可太大。UTF8最多能存21844个汉字，或65532个英文
  + varbinary(M)保存的是二进制字符串，它保存的是字节而不是字符，所以没有字符集的概念，M长度0-255（字节）。只用于排序或比较时大小写敏感的类型，不包括密码存储
  + TEXT类型与VARCHAR都类似，存储可变长度，最大限制也是2^16，但是它20bytes以后的内容是在数据页以外的空间存储（row\_format=dynamic），对它的使用需要多一次寻址，没有默认值。  
    一般用于存放容量平均都很大、操作没有其它字段那样频繁的值。  
    网上部分文章说要避免使用text和blob，要知道如果纯用varchar可能会导致行溢出，效果差不多，但因为每行占用字节数过多，会导致buffer\_pool能缓存的数据行、页下降。另外text和blob上面一般不会去建索引，而是利用sphinx之类的第三方全文搜索引擎，如果确实要创建（前缀）索引，那就会影响性能。凡事看具体场景。  
    另外尽可能把text/blob拆到另一个表中
  + BLOB可以看出varbinary的扩展版本，内容以二进制字符串存储，无字符集，区分大小写，有一种经常提但不用的场景：不要在数据库里存储图片。

### int、tinyint、decimal等数字类型定义

* + 使用tinyint来代替 enum和boolean  
    ENUM类型在需要修改或增加枚举值时，需要在线DDL，成本较高；ENUM列值如果含有数字类型，可能会引起默认值混淆  
    tinyint使用1个字节，一般用于status,type,flag的列
  + 建议使用 UNSIGNED 存储非负数值  
    相比不使用 unsigned，可以扩大一倍使用数值范围
  + int使用固定4个字节存储，int(11)与int(4)只是显示宽度的区别
  + 使用Decimal 代替float/double存储精确浮点数  
    对于货币、金额这样的类型，使用decimal，如 decimal(9,2)。float默认只能能精确到6位有效数字

### timestamp与datetime选择

* + datetime 和 timestamp类型所占的存储空间不同，前者8个字节，后者4个字节，这样造成的后果是两者能表示的时间范围不同。前者范围为1000-01-01 00:00:00 ~ 9999-12-31 23:59:59，后者范围为 1970-01-01 08:00:01 到 2038-01-19 11:14:07 。所以 TIMESTAMP 支持的范围比 DATATIME 要小。
  + timestamp可以在insert/update行时，自动更新时间字段（如 f\_set\_time timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP），但一个表只能有一个这样的定义。
  + timestamp显示与时区有关，内部总是以 UTC 毫秒 来存的。还受到严格模式的限制
  + 优先使用timestamp，datetime也没问题
  + where条件里不要对时间列上使用时间函数

### 建议字段都定义为NOT NULL

* + 如果是索引字段，一定要定义为not null 。因为null值会影响cordinate统计，影响优化器对索引的选择
  + 如果不能保证insert时一定有值过来，定义时使用default ‘’ ，或 0

### 同一意义的字段定义必须相同

比如不同表中都有 f\_user\_id 字段，那么它的类型、字段长度要设计成一样

## 4. 索引规范

### 任何新的select,update,delete上线，都要先explain，看索引使用情况

尽量避免extra列出现：Using File Sort，Using Temporary，rows超过1000的要谨慎上线。  
explain解读

* + type：ALL, index, range, ref, eq\_ref, const, system, NULL（从左到右，性能从差到好）
  + possible\_keys：指出MySQL能使用哪个索引在表中找到记录，查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询使用
  + key：表示MySQL实际决定使用的键（索引）  
    如果没有选择索引，键是NULL。要想强制MySQL使用或忽视possible\_keys列中的索引，在查询中使用FORCE INDEX、USE INDEX或者IGNORE INDEX
  + ref：表示选择 key 列上的索引，哪些列或常量被用于查找索引列上的值
  + rows：根据表统计信息及索引选用情况，估算的找到所需的记录所需要读取的行数
  + Extra
    - Using temporary：表示MySQL需要使用临时表来存储结果集，常见于排序和分组查询
    - Using filesort：MySQL中无法利用索引完成的排序操作称为“文件排序”

### 索引个数限制

* + 索引是双刃剑，会增加维护负担，增大IO压力，索引占用空间是成倍增加的
  + 单张表的索引数量控制在5个以内，或不超过表字段个数的20%。若单张表多个字段在查询需求上都要单独用到索引，需要经过DBA评估。

### 避免冗余索引

* + InnoDB表是一棵索引组织表，主键是和数据放在一起的聚集索引，普通索引最终指向的是主键地址，所以把主键做最后一列是多余的。如f\_crm\_id作为主键，联合索引(f\_user\_id,f\_crm\_id)上的f\_crm\_id就完全多余
  + (a,b,c)、(a,b)，后者为冗余索引。可以利用前缀索引来达到加速目的，减轻维护负担

### 没有特殊要求，使用自增id作为主键

* + 主键是一种聚集索引，顺序写入。组合唯一索引作为主键的话，是随机写入，适合写少读多的表
  + 主键不允许更新

### 索引尽量建在选择性高的列上

* + 不在低基数列上建立索引，例如性别、类型。但有一种情况，idx\_feedbackid\_type (f\_feedback\_id,f\_type)，如果经常用 f\_type=1 比较，而且能过滤掉90%行，那这个组合索引就值得创建。有时候同样的查询语句，由于条件取值不同导致使用不同的索引，也是这个道理。
  + 索引选择性计算方法（基数 ÷ 数据行数）  
    Selectivity = Cardinality / Total Rows = select count(distinct col1)/count(\*) from tbname，越接近1说明col1上使用索引的过滤效果越好
  + 走索引扫描行数超过30%时，改全表扫描

### 最左前缀原则

* + mysql使用联合索引时，从左向右匹配，遇到断开或者范围查询时，无法用到后续的索引列  
    比如索引idx\_c1\_c2\_c3 (c1,c2,c3)，相当于创建了(c1)、(c1,c2)、(c1,c2,c3)三个索引，where条件包含上面三种情况的字段比较则可以用到索引，但像 where c1=a and c3=c 只能用到c1列的索引，像 c2=b and c3=c等情况就完全用不到这个索引
  + 遇到范围查询(>、<、between、like)也会停止索引匹配，比如 c1=a and c2 > 2 and c3=c，只有c1,c2列上的比较能用到索引，(c1,c2,c3)排列的索引才可能会都用上
  + where条件里面字段的顺序与索引顺序无关，mysql优化器会自动调整顺序

### 前缀索引

* + 对超过30个字符长度的列创建索引时，考虑使用前缀索引，如 idx\_cs\_guid2 (f\_cs\_guid(26))表示截取前26个字符做索引，既可以提高查找效率，也可以节省空间
  + 前缀索引也有它的缺点是，如果在该列上 ORDER BY 或 GROUP BY 时无法使用索引，也不能把它们用作覆盖索引(Covering Index)
  + 如果在varbinary或blob这种以二进制存储的列上建立前缀索引，要考虑字符集，括号里表示的是字节数

### 合理使用覆盖索引减少IO

INNODB存储引擎中，secondary index(非主键索引，又称为辅助索引、二级索引)没有直接存储行地址，而是存储主键值。  
如果用户需要查询secondary index中所不包含的数据列，则需要先通过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。覆盖索引则可以在一个索引中获取所有需要的数据列，从而避免回表进行二次查找，节省IO因此效率较高。  
例如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。

### 尽量不要在频繁更新的列上创建索引

如不在定义了 ON UPDATE CURRENT\_STAMP 的列上创建索引，维护成本太高（好在mysql有insert buffer，会合并索引的插入）

## 5. SQL设计

### 杜绝直接 SELECT \* 读取全部字段

即使需要所有字段，减少网络带宽消耗，能有效利用覆盖索引，表结构变更对程序基本无影响

### 能确定返回结果只有一条时，使用 limit 1

**在保证数据不会有误的前提下**，能确定结果集数量时，多使用limit，尽快的返回结果。

### 小心隐式类型转换

* + 转换规则

a. 两个参数至少有一个是 NULL 时，比较的结果也是 NULL，例外是使用 <=> 对两个 NULL 做比较时会返回 1，这两种情况都不需要做类型转换  
b. 两个参数都是字符串，会按照字符串来比较，不做类型转换  
c. 两个参数都是整数，按照整数来比较，不做类型转换  
d. 十六进制的值和非数字做比较时，会被当做二进制串  
e. 有一个参数是 TIMESTAMP 或 DATETIME，并且另外一个参数是常量，常量会被转换为 timestamp  
f. 有一个参数是 decimal 类型，如果另外一个参数是 decimal 或者整数，会将整数转换为 decimal 后进行比较，如果另外一个参数是浮点数，则会把 decimal 转换为浮点数进行比较  
g. 所有其他情况下，两个参数都会被转换为浮点数再进行比较。

* + 如果一个索引建立在string类型上，如果这个字段和一个int类型的值比较，符合第 g 条。如f\_phone定义的类型是varchar，但where使用f\_phone in (098890)，两个参数都会被当成成浮点型。发生这个隐式转换并不是最糟的，最糟的是string转换后的float，mysql无法使用索引，这才导致了性能问题。如果是 f\_user\_id = ‘1234567’ 的情况，符合第 b 条,直接把数字当字符串比较。

### 禁止在where条件列上使用函数

* + 会导致索引失效，如lower(email)，f\_qq % 4。可放到右边的常量上计算
  + 返回小结果集不是很大的情况下，可以对返回列使用函数，简化程序开发

### 使用like模糊匹配，%不要放首位

会导致索引失效，有这种搜索需求是，考虑其它方案，如sphinx全文搜索

### 涉及到复杂sql时，务必先参考已有索引设计，先explain

* + 简单SQL拆分，不以代码处理复杂为由。
  + 比如 OR 条件： f\_phone=’10000’ or f\_mobile=’10000’，两个字段各自有索引，但只能用到其中一个。可以拆分成2个sql，或者union all。
  + 先explain的好处是可以为了利用索引，增加更多查询限制条件

### 使用join时，where条件尽量使用充分利用同一表上的索引

* + 如 select t1.a,t2.b \* from t1,t2 and t1.a=t2.a and t1.b=123 and t2.c= 4 ，如果t1.c与t2.c字段相同，那么t1上的索引(b,c)就只用到b了。此时如果把where条件中的t2.c=4改成t1.c=4，那么可以用到完整的索引
  + 这种情况可能会在字段冗余设计（反范式）时出现
  + 正确选取inner join和left join

### 少用子查询，改用join

小于5.6版本时，子查询效率很低，不像Oracle那样先计算子查询后外层查询。5.6版本开始得到优化

### 考虑使用union all，少使用union，注意考虑去重

* + union all不去重，而少了排序操作，速度相对比union要快，如果没有去重的需求，优先使用union all
  + 如果UNION结果中有使用limit，在2个子SQL可能有许多返回值的情况下，各自加上limit。如果还有order by，请找DBA。

### IN的内容尽量不超过200个

超过500个值使用批量的方式，否则一次执行会影响数据库的并发能力，因为单SQL只能且一直占用单CPU，而且可能导致主从复制延迟

### 拒绝大事务

比如在一个事务里进行多个select，多个update，如果是高频事务，会严重影响MySQL并发能力，因为事务持有的锁等资源只在事务rollback/commit时才能释放。但同时也要权衡数据写入的一致性。

### 避免使用is null, is not null这样的比较

### order by .. limit

这种查询更多的是通过索引去优化，但order by的字段有讲究，比如主键id与f\_time都是顺序递增，那就可以考虑order by id而非 f\_time 。

### c1 < a order by c2

与上面不同的是，order by之前有个范围查询，由前面的内容可知，用不到类似(c1,c2)的索引，但是可以利用(c2,c1)索引。另外还可以改写成join的方式实现。

### 分页优化

建议使用合理的分页方式以提高分页效率，大页情况下不使用跳跃式分页  
假如有类似下面分页语句:  
SELECT FROM table1 ORDER BY ftime 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 table1 WHERE ftime < last\_time ORDER BY ftime DESC LIMIT 10  
即传入上一次分页的界值

SELECT \* FROM table as t1 inner JOIN (SELECT id FROM table ORDER BY time LIMIT 10000，10) as t2 ON t1.id=t2.id

### count计数

* + 首先count()、count(1)、count(col1)是有区别的，count()表示整个结果集有多少条记录，count(1)表示结果集里以primary key统计数量，绝大多数情况下count()与count(1)效果一样的，但count(col1)表示的是结果集里 col1 列 NOT null 的记录数。优先采用count()
  + 大数据量count是消耗资源的操作，甚至会拖慢整个库，查询性能问题无法解决的，应从产品设计上进行重构。例如当频繁需要count的查询，考虑使用汇总表
  + 遇到distinct的情况，group by方式可能效率更高。

### delete,update语句改成select再explain

select最多导致数据库慢，写操作才是锁表的罪魁祸首

### 减少与数据库交互的次数，尽量采用批量SQL语句

* + INSERT ... ON DUPLICATE KEY UPDATE ...，插入行后会导致在一个UNIQUE索引或PRIMARY KEY中出现重复值，则执行旧行UPDATE，如果不重复则直接插入，影响1行。
  + REPLACE INTO类似，但它是冲突时删除旧行。INSERT IGNORE相反，保留旧行，丢弃要插入的新行。
  + INSERT INTO VALUES(),(),()，合并插入。

### 杜绝危险SQL

* + 去掉where 1=1 这样无意义或恒真的条件，如果遇到update/delete或遭到sql注入就恐怖了
  + SQL中不允许出现DDL语句。一般也不给予create/alter这类权限，但阿里云RDS只区分读写用户

## 6. 行为规范

* 不允许在DBA不知情的情况下导现网数据
* 大批量更新，如修复数据，避开高峰期，并通知DBA。直接执行sql的由运维或DBA同事操作
* 及时处理已下线业务的SQL
* 复杂sql上线审核  
  因为目前还没有SQL审查机制，复杂sql如多表join,count,group by，主动上报DBA评估。
* 重要项目的数据库方案选型和设计必须提前通知DBA参与

## 本文参考

* [互联网MySQL开发规范](http://wangwei007.blog.51cto.com/68019/1709769) 这个基本也是《去哪儿MySQL开发规范.pdf》版本
* MySQL数据库开发的三十六条军规石展完整.pdf
* [老叶观点：MySQL开发规范之我见](http://imysql.com/2015/07/23/something-important-about-mysql-design-reference.shtml)
* [MySQL开发规范与使用技巧总结](http://blog.csdn.net/xlgen157387/article/details/48086607)
* <http://highdb.com/mysql%E5%BC%80%E5%8F%91%E8%A7%84%E8%8C%83/>

 写在前面：无规矩不成方圆。对于刚加入互联网的朋友们，肯定会接触到MySQL，MySQL作为互联网最流行的关系型数据库产品，它有它擅长的地方，也有它不足的短板，针对它的特性，结合互联网大多应用的特点，笔者根据自己多年互联网公司的MySQL DBA经验，现总结出互联网MySQL的一些开发规范，仅供参考。

作者是微信订阅号yunweibang特约技术专家刘秋岐，多年数据库经验，如有问题可以订阅yunweibang并留言。

摘要：

* 基础规范
* 命名规范
* 库、表、字段开发设计规范
* 索引规范
* SQL规范
* 流程规范

一

**基础规范**

(1) 使用INNODB存储引擎

(2) 表字符集使用UTF8

(3) 所有表都需要添加注释

(4) 单表数据量建议控制在5000W以内

(5) 不在数据库中存储图、文件等大数据

(6) 禁止在线上做数据库压力测试

(7) 禁从测试、开发环境直连数据库

二

**命名规范**

(1) 库名表名字段名必须有固定的命名长度，12个字符以内

(2) 库名、表名、字段名禁止超过32个字符。须见名之意

(3) 库名、表名、字段名禁止使用MySQL保留字

(4) 临时库、表名必须以tmp为前缀，并以日期为后缀

(5) 备份库、表必须以bak为前缀，并以日期为后缀

三

**库、表、字段开发设计规范**

(1) 禁使用分区表

(2) 拆分大字段和访问频率低的字段，分离冷热数据

(3) 用HASH进散表，表名后缀使进制数，下标从0开始

(4) 按日期时间分表需符合YYYY[MM][DD][HH]格式

(5) 采用合适的分库分表策略。例如千库十表、十库百表等

(6) 尽可能不使用TEXT、BLOB类型

(7) 用DECIMAL代替FLOAT和DOUBLE存储精确浮点数

(8) 越简单越好：将字符转化为数字、使用TINYINT来代替ENUM类型

(9) 所有字段均定义为NOT NULL

(10) 使用UNSIGNED存储非负整数

(11) INT类型固定占用4字节存储

(12) 使用timestamp存储时间

(13) 使用INT UNSIGNED存储IPV4

(14) 使用VARBINARY存储大小写敏感的变长字符串

(15) 禁止在数据库中存储明文密码，把密码加密后存储

(16) 用好数值类型字段

Tinyint      (1Byte)

smallint     (2Byte)

mediumint   (3Byte)

int         (4Byte)

bigint       (8Byte)

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字节 | 最小值 | 最大值 |
|  |  | (带符号的/无符号的) | (带符号的/无符号的) |
| TINYINT | 1 | -128 | 127 |
| 无符号 |  | 0 | 255 |
| SMALLINT | 2 | -32768 | 32767 |
| 无符号 |  | 0 | 65535 |
| MEDIUMINT | 3 | -8388608 | 8388607 |
| 无符号 |  | 0 | 16777215 |
| INT | 4 | -2147483648 | 2147483647 |
| 无符号 |  | 0 | 4294967295 |
| BIGINT | 8 | -9223372036854775808 | 9223372036854775807 |
| 无符号 |  | 0 | 18446744073709551615 |

如果数值字段没有那么大，就不要用 bigint

(17) 存储ip最好用int存储而非char(15)

(18) 不允许使用ENUM

(19) 避免使用NULL字段

NULL字段很难查询优化，NULL字段的索引需要额外空间，NULL字段的复合索引无效

(20) 少用text/blob，varchar的性能会比text高很多，实在避免不了blob，请拆表

(21) 数据库中不允许存储大文件，或者照片，可以将大对象放到磁盘上，数据库中存储它的路径

四

**索引规范**

1、索引的数量要控制：

(1) 单张表中索引数量不超过5个

(2) 单个索引中的字段数不超过5个

(3) 对字符串使用前缀索引，前缀索引长度不超过8个字符

(4) 建议优先考虑前缀索引，必要时可添加伪列并建立索引

2、主键准则

(1) 表必须有主键

(2) 不使用更新频繁的列作为主键

(3) 尽量不选择字符串列作为主键

(4) 不使用UUID MD5 HASH这些作为主键(数值太离散了)

(5) 默认使非空的唯一键作为主键

(6) 建议选择自增或发号器

3、重要的SQL必须被索引，比如：

(1) UPDATE、DELETE语句的WHERE条件列

(2) ORDER BY、GROUP BY、DISTINCT的字段

4、多表JOIN的字段注意以下：

(1) 区分度最大的字段放在前面

(2) 核SQL优先考虑覆盖索引

(3) 避免冗余和重复索引

(4) 索引要综合评估数据密度和分布以及考虑查询和更新比例

5、索引禁忌

(1) 不在低基数列上建立索引，例如“性别”

(2) 不在索引列进行数学运算和函数运算

6、尽量不使用外键

(1) 外键用来保护参照完整性，可在业务端实现

(2) 对父表和子表的操作会相互影响，降低可用性

7、索引命名：非唯一索引必须以 idx\_字段1\_字段2命名，唯一所以必须以uniq\_字段1\_字段2命名，索引名称必须全部小写

8、新建的唯一索引必须不能和主键重复

9、索引字段的默认值不能为NULL，要改为其他的default或者空。NULL非常影响索引的查询效率

10、反复查看与表相关的SQL，符合最左前缀的特点建立索引。多条字段重复的语句，要修改语句条件字段的顺序，为其建立一条联合索引，减少索引数量

11、能使用唯一索引就要使用唯一索引，提高查询效率

12、研发要经常使用explain，如果发现索引选择性差，必须让他们学会使用hint

五

**SQL规范**

(1) sql语句尽可能简单

大的sql想办法拆成小的sql语句(充分利用QUERY CACHE和充分利用多核CPU)

(2) 事务要简单，整个事务的时间长度不要太长

(3) 避免使用触发器、函数、存储过程

(4) 降低业务耦合度，为sacle out、sharding留有余地

(5) 避免在数据库中进数学运算(MySQL不擅长数学运算和逻辑判断)

(4) 不要用select \*，查询哪几个字段就select 这几个字段

(5) sql中使用到OR的改写为用 IN()   (or的效率没有in的效率高)

(6) in里面数字的个数建议控制在1000以内

(7) limit分页注意效率。Limit越大，效率越低。可以改写limit，比如例子改写：

select id from tlimit 10000, 10;  =>  select id from t where id > 10000 limit10;

(9) 使用union all替代union

(10) 避免使大表的JOIN

(11) 使用group by 分组、自动排序

(12) 对数据的更新要打散后批量更新，不要一次更新太多数据

(13) 减少与数据库的交互次数

(13) 注意使用性能分析工具

Sql explain  /  showprofile   /    mysqlsla

(14) SQL语句要求所有研发，SQL关键字全部是大写，每个词只允许有一个空格

(15) SQL语句不可以出现隐式转换，比如 select id from 表 where id='1'

(16) IN条件里面的数据数量要少，我记得应该是500个以内，要学会使用exist代替in，exist在一些场景查询会比in快

(17) 能不用NOT IN就不用NOTIN，坑太多了。。会把空和NULL给查出来

(18) 在SQL语句中，禁止使用前缀是%的like

(19) 不使用负向查询，如not in/like

(19) 关于分页查询：程序里建议合理使用分页来提高效率limit，offset较大要配合子查询使用

(20) 禁止在数据库中跑大查询

(21) 使预编译语句，只传参数，比传递SQL语句更高效；一次解析，多次使用；降低SQL注入概率

(22) 禁止使order by rand()

(23) 禁单条SQL语句同时更新多个表

六

**流程规范**

(1) 所有的建表操作需要提前告知该表涉及的查询sql；

(2) 所有的建表需要确定建立哪些索引后才可以建表上线；

(3) 所有的改表结构、加索引操作都需要将涉及到所改表的查询sql发出来告知DBA等相关人员；

(4) 在建新表加字段之前，要求研发至少要提前3天邮件出来，给dba们评估、优化和审核的时间

(5)批量导入、导出数据必须提前通知DBA协助观察

(6) 禁在线上从库执行后台管理和统计类查询

(7) 禁有super权限的应用程序账号存在

(8) 推广活动或上线新功能必须提前通知DBA进行流量评估

(9) 不在业务高峰期批量更新、查询数据库

作者是微信订阅号yunweibang特约技术专家刘秋岐，多年数据库经验，如有问题可以订阅yunweibang并留言。

**转载本文请务必带有本订阅号二维码及作者信息**

**出自：http://mp.weixin.qq.com/s?plg\_nld=1&plg\_uin=1&mid=207132223&idx=1&plg\_nld=1&scene=22&plg\_auth=1&\_\_biz=MzA3MzYwNjQ3NA%3D%3D&plg\_dev=1&srcid=11031H8kGQS39BOTBXDbsowL&plg\_usr=1&plg\_vkey=1&sn=f5d98146f28235d91fe3e675cead4ce5#rd**