MySQL数据库设计规范

一、 表设计

1.库名、表名、字段名必须使用小写字母，“_”分割。
2.库名、表名、字段名必须不超过12个字符。
3.库名、表名、字段名见名知意,建议使用名词而不是动词。
4.建议使用InnoDB存储引擎。
5.存储精确浮点数必须使用DECIMAL替代FLOAT和DOUBLE。
6.建议使用UNSIGNED存储非负数值。
7.建议使用INT UNSIGNED存储IPV4。
8.整形定义中不添加长度，比如使用INT，而不是INT(4)。
9.使用短数据类型，比如取值范围为0-80时，使用TINYINT UNSIGNED。
10.不建议使用ENUM类型，使用TINYINT来代替。
11.尽可能不使用TEXT、BLOB类型。
12.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。
13.VARCHAR(N)，N尽可能小，因为MySQL一个表中所有的VARCHAR字段最大长度是65535个字节，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。
14.表字符集选择UTF8。
15.使用VARBINARY存储变长字符串。
16.存储年使用YEAR类型。
17.存储日期使用DATE类型。
18.存储时间（精确到秒）建议使用TIMESTAMP类型，因为TIMESTAMP使用4字节，DATETIME使用8个字节。
19.建议字段定义为NOT NULL。
20.将过大字段拆分到其他表中。
21.禁止在数据库中使用VARBINARY、BLOB存储图片、文件等。
22.表结构变更需要通知DBA审核。
二、 索引

1.非唯一索引必须按照“idx_字段名称_字段名称[_字段名]”进行命名。
2.唯一索引必须按照“uniq_字段名称_字段名称[_字段名]”进行命名。
3.索引名称必须使用小写。
4.索引中的字段数建议不超过5个。
5.单张表的索引数量控制在5个以内。
6.唯一键由3个以下字段组成，并且字段都是整形时，使用唯一键作为主键。

7.没有唯一键或者唯一键不符合5中的条件时，使用自增（或者通过发号器获取）id作为主键。
8.唯一键不和主键重复。
9.索引字段的顺序需要考虑字段值去重之后的个数，个数多的放在前面。
10.ORDER BY，GROUP BY，DISTINCT的字段需要添加在索引的后面。
11.使用EXPLAIN判断SQL语句是否合理使用索引，尽量避免extra列出现：Using File Sort，Using Temporary。
12.UPDATE、DELETE语句需要根据WHERE条件添加索引。
13.不建议使用%前缀模糊查询，例如LIKE “%weibo”。
14.对长度过长的VARCHAR字段建立索引时，添加crc32或者MD5 Hash字段，对Hash字段建立索引。
15.合理创建联合索引（避免冗余），(a,b,c) 相当于 (a) 、(a,b) 、(a,b,c)。
16.合理利用覆盖索引。
17.SQL变更需要确认索引是否需要变更并通知DBA。
三、 SQL语句

1.使用prepared statement，可以提供性能并且避免SQL注入。
2.SQL语句中IN包含的值不应过多。
3.UPDATE、DELETE语句不使用LIMIT。
4.WHERE条件中必须使用合适的类型，避免MySQL进行隐式类型转化。
5.SELECT语句只获取需要的字段。
6.SELECT、INSERT语句必须显式的指明字段名称，不使用SELECT *，不使用INSERT INTO table()。*
7.使用SELECT column_name1, column_name2 FROM table WHERE [condition]而不是SELECT column_name1 FROM table WHERE [condition]和SELECT column_name2 FROM table WHERE [condition]。
8.WHERE条件中的非等值条件（IN、BETWEEN、<、<=、>、>=）会导致后面的条件使用不了索引。
9.避免在SQL语句进行数学运算或者函数运算，容易将业务逻辑和DB耦合在一起。
10.INSERT语句使用batch提交（INSERT INTO table VALUES(),(),()……），values的个数不应过多。
11.避免使用存储过程、触发器、函数等，容易将业务逻辑和DB耦合在一起，并且MySQL的存储过程、触发器、函数中存在一定的bug。
12.避免使用JOIN。
13.使用合理的SQL语句减少与数据库的交互次数。
14.不使用ORDER BY RAND()，使用其他方法替换。
15.建议使用合理的分页方式以提高分页的效率。
16.统计表中记录数时使用COUNT()，而不是COUNT(primary_key)和COUNT(1)。
17.禁止在从库上执行后台管理和统计类型功能的QUERY。
四、 散表

1.每张表数据量建议控制在5000w以下。
2.可以结合使用hash、range、lookup table进行散表。
3.散表如果使用md5（或者类似的hash算法）进行散表，表名后缀使用16进制，比如user_ff。
4.推荐使用CRC32求余（或者类似的算术算法）进行散表，表名后缀使用数字，数字必须从0开始并等宽，比如散100张表，后缀从00-99。
5.使用时间散表，表名后缀必须使用特定格式，比如按日散表user_20110209、按月散表user_201102。
五、 其他

1.批量导入、导出数据需要DBA进行审查，并在执行过程中观察服务。
2.批量更新数据，如update,delete 操作，需要DBA进行审查，并在执行过程中观察服务。
3.产品出现非数据库平台运维导致的问题和故障时，如前端被抓站，请及时通知DBA，便于维护服务稳定。
4.业务部门程序出现bug等影响数据库服务的问题,请及时通知DBA，便于维护服务稳定。
5.业务部门推广活动，请提前通知DBA进行服务和访问评估。
6.如果出现业务部门人为误操作导致数据丢失，需要恢复数据，请在第一时间通知DBA，并提供准确时间，误操作语句等重要线索。

FAQ

1-1.库名、表名、字段名必须使用小写字母，“_”分割。

a) 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。

b) 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。

c) 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。

d) 为了统一规范， 库名、表名、字段名使用小写字母。

back

1-2.库名、表名、字段名必须不超过12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，必须不超过12字符。

back

1-3.库名、表名、字段名见名知意,建议使用名词而不是动词。

a) 用户评论可用表名usercomment或者comment。

b) 库表是一种客观存在的事物，一种对象，所以建议使用名词。

back

1-4.建议使用InnoDB存储引擎。

a) 5.5以后的默认引擎，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。

b) 具体比较可见附件的官方白皮书。

back

1-5.存储精确浮点数必须使用DECIMAL替代FLOAT和DOUBLE。

a) mysql中的数值类型（不包括整型）：
IEEE754浮点数： float （单精度） ， double 或 real （双精度）
定点数： decimal 或 numeric
单精度浮点数的有效数字二进制是24位，按十进制来说，是8位；双精度浮点数的有效数字二进制是53位，按十进制来说，是16 位
一个实数的有效数字超过8位，用单精度浮点数来表示的话，就会产生误差！同样，如果一个实数的有效数字超过16位，用双精度浮点数来表示，也会产生误差
b) IEEE754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，也会造成误差，原因是不是所有的数都能转换成有限长度的二进制数。
即一个二进制可以准确转换成十进制，但一个带小数的十进制不一定能够准确地用二进制来表示。

实例：
drop table if exists t;

create table t(value float(10,2));

insert into t values(131072.67),(131072.68);

select value from t;

+-----------+

| value |

+-----------+

| 131072.67 |

| 131072.69 |

+-----------+

back

1-6.建议使用UNSIGNED存储非负数值。

同样的字节数，存储的数值范围更大。如tinyint 有符号为 -128-127，无符号为0-255

back

1-7. 如何使用INT UNSIGNED存储ip?

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或者两个bigINT来存储。例如：

SELECT INET_ATON('209.207.224.40');

3520061480

SELECT INET_NTOA(3520061480);

209.207.224.40

back

1-8. INT[M]，M值代表什么含义?

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。

back

1-10.不建议使用ENUM、SET类型，使用TINYINT来代替。

a) ENUM，有三个问题：添加新的值要做DDL，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)，索引值问题（插入数字实际是插入索引对应的值）

实例：

```
drop table if exists t;

create table t(sex enum('0','1'));

insert into t values(1);

insert into t values('3');

select * from t;

+------+

| sex |

+------+

| 0 |

| |

+------+

2 rows in set (0.00 sec)
```

back

1-11.尽可能不使用TEXT、BLOB类型。

a) 索引排序问题，只能使用max_sort_length的长度或者手工指定ORDER BY SUBSTRING(column, length)的长度来排序

b) Memory引擎不支持text,blog类型，会在磁盘上生成临时表

c) 可能浪费更多的空间

d) 可能无法使用adaptive hash index

e) 导致使用where没有索引的语句变慢

back

1-13. VARCHAR中会产生额外存储吗?

VARCHAR(M)，如果M<256时会使用一个字节来存储长度，如果M>=256则使用两个字节来存储长度。

back

1-14.表字符集选择UTF8。

a) 使用utf8字符集，如果是汉字，占3个字节，但ASCII码字符还是1个字节。
b) 统一，不会有转换产生乱码风险
c) 其他地区的用户（美国、印度、台湾）无需安装简体中文支持，就能正常看您的文字，并且不会出现乱码
d) ISO-8859-1编码(latin1)使用了单字节内的所有空间，在支持ISO-8859-1的系统中传输和存储其他任何编码的字节流都不会被抛弃。即把其他任何编码的字节流当作ISO-8859-1编码看待都没有问题，保存的是原封不动的字节流。

back

1-15.使用VARBINARY存储变长字符串。

二进制字节流，不存在编码问题

back

1-18. 为什么建议使用TIMESTAMP来存储时间而不是DATETIME?

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动更新的特性。

如何使用TIMESTAMP的自动赋值属性?

a) 将当前时间作为ts的默认值：ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP。

b) 当行更新时，更新ts的值：ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP。

c) 可以将1和2结合起来：ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP。

back

1-19.建议字段定义为NOT NULL。

a）如果null字段被索引，需要额外的1字节

b）使索引，索引统计，值的比较变得更复杂

c）可用0，''代替

d）如果是索引字段，一定要定义为not null

back

1-21.禁止在数据库中使用VARBINARY、BLOB存储图片、文件等。

采用分布式文件系统更高效

back

1. 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。

back

2-5. 为什么一张表中不能存在过多的索引?

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。

back

2-11. EXPLAIN语句

EXPLAIN 语句（在MySQL客户端中执行）可以获得MySQL如何执行SELECT语句的信息。通过对SELECT语句执行EXPLAIN，可以知晓MySQL执 行该SELECT语句时是否使用了索引、全表扫描、临时表、排序等信息。尽量避免MySQL进行全表扫描、使用临时表、排序等。详见官方文档。

back

2-13.不建议使用%前缀模糊查询，例如LIKE “%weibo”。

会导致全表扫描

2-14. 如何对长度大于50的VARCHAR字段建立索引?

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)

)
```

back

2-16. 什么是覆盖索引?

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。

back

3-3.UPDATE、DELETE语句不使用LIMIT。

a) 可能导致主从数据不一致

b) 会记录到错误日志，导致日志占用大量空间

3-4. 为什么需要避免MySQL进行隐式类型转化?

因为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，原因和避免在索引字段中使用函数是类似的。

back

3-6. 为什么不建议使用SELECT *?

增加很多不必要的消耗（cpu、io、内存、网络带宽）；增加了使用覆盖索引的可能性；当表结构发生改变时，前段也需要更新。

back

3-13. 如何减少与数据库的交互次数?

使用下面的语句来减少和db的交互次数：

INSERT ... ON DUPLICATE KEY UPDATE

REPLACE

INSERT IGNORE

INSERT INTO values(),()如何结合使用多个纬度进行散表散库?

例如微博message，先按照crc32(message_id)%16将message散到16个库中，然后针对每个库中的表，一天生成一张新表。

back

3-14. 为什么不能使用ORDER BY rand()?

因为ORDER BY rand()会将数据从磁盘中读取，进行排序，会消耗大量的IO和CPU，可以在程序中获取一个rand值，然后通过在从数据库中获取对应的值。

back

3-15. MySQL中如何进行分页?

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)

back

3-17.为什么避免使用复杂的SQL?

拒绝使用复杂的SQL，将大的SQL拆分成多条简单SQL分步执行。原因：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。

back

1. InnoDB存储引擎为什么避免使用COUNT(*)?

InnoDB表避免使用COUNT(*)操作，计数统计实时要求较强可以使用memcache或者redis，非实时统计可以使用单独统计表，定时更新。