MySQL数据库设计SQL规范

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；
2、库名、表名、字段名支持最多32个字符，易于辨识以及减少传输量不要超过32；
3、库名、表名、字段名禁止使用MySQL保留关键字；
4、临时库、临时表名必须以tmp为前缀并以日期为后缀；
5、备份库、备份表名必须以bak为前缀并以日期为后缀；
---

2 基本规范

1、使用INNODB存储引擎
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；
2、表字符集使用UTF8
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；统一，不会有转换产生乱码风险；
3、所有表都需要添加注释；
4、不在数据库中存储图片、文件等大数据；
5、禁止在线上做数据库压力测试；
6、禁止从测试、开发环境直连线上数据库；
---

3 库表设计规范

1、尽量避免使用分区表
MySQL的分区表实际性能不是很好。
2、拆分大字段和访问频率低的字段，分离冷热数据
3、采用合理的分库分表策略，推荐使用HASH进行分表，表名后缀使用十进制数，下标从0开始
首次分表尽量多的分，避免二次分表，二次分表的难度和成本较高
4、按日期时间分表需符合YYYY[MM][DD][HH]格式
5、单表字段数控制在20个以内
6、一条完整的建表语句中应包含必要的字段、主键、合理的索引（综合代码中所有的条件语句创建合理的索引，主键必须要有）

---

4 索引设计规范

索引是一把双刃剑，它可以提高查询效率但也会降低插入和更新的速度并占用磁盘空间
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；
对字符串使用前缀索引，前缀索引长度不超过10个字符；如果有一个CHAR(200)列，如果在前10个字符内，多数值是惟一的，那么就不要对整个列进行索引。对前10个字符进行索引能够节省大量索引空间，也可能会使查询更快；
3、表必须有主键，不使用UUID、MD5、HASH作为主键，尽量不选择字符串列作为主键；主键建议选择自增id；
4、创建复合索引时区分度较大的字段放在最前面；不在低区分度的字段上创建索引，例如“性别”；
5、避免冗余或重复索引
合理创建联合索引（避免冗余），index(a、b、c)相当于index(a)、index(a、b)、index(a、、b、c)；
6、索引不是越多越好，按实际需要进行创建
每个额外的索引都要占用额外的磁盘空间，并降低写操作的性能
7、不在索引列进行数学运算和函数运算；
8、尽量不要使用外键
外键用来保护参照完整性，可在业务端实现，对父表和子表的操作会相互影响，降低可用性；
9、不使用%前导的查询，如like“%xxx”，无法使用索引；
10、不使用反向查询，如not in / not like
无法使用索引，导致全表扫描
全表扫描导致buffer pool利用降低
---

5 字段设计规范

1、尽可能不要使用TEXT、BLOB类型
删除这种值会在数据表中留下很大的"空洞"，可以考虑把BLOB或TEXT列分离到单独的表中
2、用DECIMAL代替FLOAT和DOUBLE存储精确浮点数
浮点数相对于定点数的优点是在长度一定的情况下，浮点数能够表示更大的数据范围；浮点数的缺点是会引起精度问题
3、将字符转化为数字
4、使用TINYINT来代替ENUM类型
5、字段长度尽量按实际需要进行分配，不要随意分配一个很大的容量

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；
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；
6、如果可能的话所有字段均定义为not null
7、使用UNSIGNED存储非负整数
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
8、使用TIMESTAMP存储时间. 因为TIMESTAMP使用4字节，DATETIME使用8个字节,同时TIMESTAMP具有自动赋值以及自动更新的特性.
9、使用INT UNSIGNED存储IPV4
10、使用VARBINARY存储大小写敏感的变长字符串
11、禁止在数据库中存储明文密码
---

6 SQL设计规范

1、使用预编译语句prepared statement
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
2、尽量避免相同语句由于书写格式的不同，而导致多次语法分析
3、避免隐式转换
会导致索引失效，如select userid from table where userid=’ 1234’
4、充分利用前缀索引
必须是最左前缀，不可能同时用到两个范围条件
5、避免使用存储过程、触发器、EVENTS等
让数据库做最擅长的事，降低业务耦合度，为sacle out、shading留点余地，避开BUG
6、避免使用大表的join
MySQL最擅长的是单表的主键/二级索引查询
Join消耗较多的内存，产生临时表
7、避免在数据库中进行数学运算
容易将业务逻辑和DB耦合在一起
MySQL不擅长数学运算和逻辑判断
无法使用索引
8、拒绝大SQL，拆分成小SQL
充分利用多核CPU
9、使用in代替or，in的值不超过1000个
10、禁止使用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
11、使用union all而不是union
12、程序应有捕获SQL异常的处理机制
13、禁止单条SQL语句同时更新多个表
14、不使用select * from 消耗cpu和IO、消耗网络带宽，无法使用覆盖索引，减少表结构变更带来的影响
---

7 行为规范

1、批量导入、导出数据必须提前通知DBA协助观察；表结构变更必须通知DBA进行审核
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负载等各种状况；
3、禁止在主库上执行后台管理和统计类的功能查询；
4、禁止有super权限的应用程序账号存在；
5、产品出现非数据库导致的故障时及时通知DBA协助排查；
6、促销活动或上线新功能必须提前通知DBA进行流量评估；
7、数据库数据丢失，及时联系DBA进行恢复；
8、对单表的多次alter操作必须合并为一次操作，相同类型的写操作合并为一条语句；
9、不在MySQL数据库中存放业务逻辑；
10、重大项目的数据库方案选型和设计必须提前通知DBA参与；
11、对特别重要的库表，提前与DBA沟通确定维护和备份优先级；
12、不在业务高峰期批量更新、查询数据库；
13、 提交线上建表需求，必须详细注明所有相关SQL。

8.DBA规范

MySQL开发规范V1.2

9.FAQ

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)字段名显式区分大小写,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。

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版本以后的默认引擎,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。

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

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,成本较大;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、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,并自动更新: column1 TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
b)只是自动初始化: column1 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
c)自动更新,初始化的值为0: column1 TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
d)初始化的值为0: column1 TIMESTAMP DEFAULT 0

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。

13.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缓存命中率。

14.禁止在数据库中存储明文密码。

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。防止数据库数据被公司内部人员或黑客获取后，采用字典攻击等方式暴力破解用户密码。

15.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。

16.禁止冗余索引。

索引是双刃剑,会增加维护负担,增大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。

17.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。

18.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。

19.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个索引中获取所有需要的数据,因此效率较高。主键查询是天然的覆盖索引。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。

20.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。

21.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。

22.用UNION ALL代替UNION。

UNION ALL不需要对结果集再进行排序。

23.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。

24.建议使用合理的分页方式以提高分页效率。

第一种分页写法：

```
select *
  from t
 where thread_id = 771025
   and deleted = 0
 order by gmt_create asc limit 0, 15;
```

```
select * from t
 where thread_id = 771025
   and deleted = 0
 order by gmt_create asc limit 0, 15;
```

原理：一次性根据过滤条件取出所有字段进行排序返回。

数据访问开销=索引IO+索引全部记录结果对应的表数据IO

缺点：该种写法越翻到后面执行效率越差，时间越长，尤其表数据量很大的时候。

适用场景：当中间结果集很小（10000行以下）或者查询条件复杂（指涉及多个不同查询字段或者多表连接）时适用。

第二种分页写法：

```
select t.* from (
   select id from t
    where thread_id = 771025 and deleted = 0 order by gmt_create asc limit 0, 15) a, t
 where a.id = t.id;
```

前提：假设t表主键是id列，且有覆盖索引secondary key:(thread_id, deleted, gmt_create)

原理：先根据过滤条件利用覆盖索引取出主键id进行排序，再进行join操作取出其他字段。

数据访问开销=索引IO+索引分页后结果（例子中是15行）对应的表数据IO。

优点：每次翻页消耗的资源和时间都基本相同，就像翻第一页一样。

适用场景：当查询和排序字段（即where子句和order by子句涉及的字段）有对应覆盖索引时，且中间结果集很大的情况时适用。

25.SELECT只获取必要的字段,禁止使用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。

26.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,产生的SQL语句无法利用QUERY CACHE。

27.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MySQL的多线程复制特性。

28.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()

29.拆分复杂SQL为多个小SQL,避免大事务。

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。

30.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);
然后增加索引:
alter table t add index idx_aa(aa);
正确的做法是:
alter table t add column b varchar(10),add index idx_aa(aa);

31.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。

32.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。

33.提交线上建表改表需求,必须详细注明涉及到的所有SQL语句(包括INSERT、DELETE、UPDATE),便于DBA进行审核和优化。

并不只是SELECT语句需要用到索引。UPDATE、DELETE都需要先定位到数据才能执行变更。因此需要业务提供所有的SQL语句便于DBA审核。

34.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。

35.建表语句示例

```
CREATE TABLE `order_info` (
`id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT '订单表id',
`order_no` varchar(18) NOT NULL DEFAULT '' COMMENT '订单号',
`order_source` tinyint(2) NOT NULL DEFAULT '0' COMMENT '订单来源 0 定向发布 1 律师社区 2 非定向发布',
`order_status` tinyint(2) NOT NULL DEFAULT '0' COMMENT '0 已支付 1服务中 2未评价3 已完成 98 已取消 99 待支付',
`direction_id` int(11) NOT NULL DEFAULT '0' COMMENT '分类方向Id',
`is_refund` tinyint(2) NOT NULL DEFAULT '0' COMMENT '0 未退款 1已退款',
`order_content` varchar(1000) NOT NULL DEFAULT '',
`start_time` datetime NOT NULL DEFAULT '1900-01-01 00:00:00' COMMENT '服务开始时间',
`end_time` datetime NOT NULL DEFAULT '1900-01-01 00:00:00' COMMENT '服务结束时间',
`price` decimal(8,2) NOT NULL DEFAULT '0.00' COMMENT '订单金额',
`tel_no` varchar(20) NOT NULL DEFAULT '' COMMENT '委托人电话',
`lawyer_id` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '律师id',
`customer_id` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '用户id',
`lawyer_comment` varchar(200) NOT NULL DEFAULT '' COMMENT '律师评价',
`dt_create` datetime NOT NULL DEFAULT '2015-01-01 00:00:00' COMMENT '创建时间',
`dt_update` datetime NOT NULL DEFAULT '2015-01-01 00:00:00' COMMENT '更新时间',
PRIMARY KEY (`id`),
UNIQUE KEY `uniq_order_no` (`order_no`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COMMENT='订单表';
```

36.产品线对应名称说明(供参考)