# 作者介绍

**贺春旸**，普惠金融MySQL专家，《MySQL管理之道》第一版、第二版作者。曾任职于中国移动飞信、机锋安卓市场，拥有丰富的数据库管理经验。目前致力于MySQL、Linux等开源技术的研究。

# DBA操作规范

1、涉及业务上的修改/删除数据，在得到业务方、CTO的邮件批准后方可执行，执行前提前做好备份，必要时可逆。

2、所有上线需求必须走工单系统，口头通知视为无效。

3、在对大表做表结构变更时，如修改字段属性会造成锁表，并会造成从库延迟，从而影响线上业务，必须在凌晨0:00后业务低峰期执行，另统一用工具pt-online-schema-change避免锁表且降低延迟执行时间。

使用范例：

|  |
| --- |
| #pt-online-schema-change  --alter="add index   IX\_id\_no(id\_no)"  \  --no-check-replication-filters  --recursion-method=none  --user=dba    \  --password=123456  D=test,t=t1 --execute |

对于MongoDB创建索引要在后台创建，避免锁表。

使用范例：

|  |
| --- |
| db.t1.createIndex({idCardNum:1},{background:1}) |

4、所有线上业务库均必须搭建MHA高可用架构，避免单点问题。

5、给业务方开权限时，密码要用MD5加密，至少16位。权限如没有特殊要求，均为select查询权限，并做库表级限制。

6、删除默认空密码账号。

|  |
| --- |
| delete from mysql.user where user='' and password='';  flush privileges; |

7、汇总库开启Audit审计日志功能，出现问题时方可追溯。

操作规范补充一条:

DBA一定要有危险意识，update或delete操作时一定要看清楚有没有加where条件，在update或delete操作前务必先select看看筛选结果是否符合。

举个例子:

update p\_product\_dbinfo set db\_pwd='jishu8.cc' where db\_pwd=’0’;

错误的输成

update p\_product\_dbinfo set db\_pwd='jishu8.cc' where db\_pwd=0;

导致更新出错。

如果在update前先执行select p\_product\_dbinfo set db\_pwd='jishu8.cc' where db\_pwd=0;就可以及时发现筛选条件错误，避免发生事故！

# 行为规范

8、禁止一个MySQL实例存放多个业务数据库，会造成业务耦合性过高，一旦出现问题会殃及池鱼，增加了定位故障问题的难度。通常采用多实例解决，一个实例一个业务库，互不干扰。

解读:多实例在缺乏管理时非常浪费服务器资源，造成性能问题。而多个业务同一个实例，确实容易有耦合问题，一旦出现问题会殃及池鱼，增加了定位故障问题的难度。所以目前我们公司是平台级产品每个业务，用单数据库服务器单实例，主从复制。小型产品/不重要的业务，共用数据库服务器安装多实例。我们所有的产品几乎都是应用与数据库分离，部署于不同服务器。

9、禁止在主库上执行后台管理和统计类的功能查询，这种复杂类的SQL会造成CPU的升高，进而会影响业务。

解读:查询类操作一般都在从库执行，并且从库设置了--super-read-only，严格防写入。

10、批量清洗数据，需要开发和DBA共同进行审查，应避开业务高峰期时段执行，并在执行过程中观察服务状态。

11、促销活动等应提前与DBA当面沟通，进行流量评估，比如提前一周增加机器内存或扩展架构，防止DB出现性能瓶颈。

12、禁止在线上做数据库压力测试。

# 基本规范

13、禁止在数据库中存储明文密码。

14、使用InnoDB存储引擎。

* 支持事务，行级锁，更好的恢复性，高并发下性能更好。
* InnoDB表避免使用COUNT(\*)操作，因内部没有计数器，需要一行一行累加计算，计数统计实时要求较强可以使用memcache或者Redis。

解读:

前几天就有朋友就问过我COUNT(\*)有什么办法优化，我就回复说MyISAM可以秒出，InnoDB没有办法优化，我当时建议新建一个计数表，用触发器来计数。贺总这个用Redis的方法也是一个方法。

15、表字符集统一使用UTF8。

不会产生乱码风险。

解读:

我们公司就是用UTF8，当然移动互联网公司一般用utf8mb4，以兼容表情的存储。

16、所有表和字段都需要添加中文注释。

方便他人、方便自己。

解读:

大神丁奇回复了，请把“中文”两字去掉^^

17、不在数据库中存储图片、文件等大数据。

图片、文件更适合于GFS分布式文件系统，数据库里存放超链接即可。

解读:

我们公司有研发用数据库存储比较大的xml，当时也是建议改为存地址链接。

18、避免使用存储过程、视图、触发器、事件。

MySQL是OLTP应用，最擅长简单的增、删、改、查操作，但对逻辑计算分析类的应用，并不适合，所以这部分的需求最好通过程序上实现。

解读:  
其实我们公司产品大量使用视图、触发器、存储过程、事件，这些东西开发起来方便和迅速，因为产品落地为主，使用用户一般只是局域网，用户数不多，也是可以的

19、避免使用外键，外键用来保护参照完整性，可在业务端实现。

外键会导致父表和子表之间耦合，十分影响SQL性能，出现过多的锁等待，甚至会造成死锁。

解读:

外键在订单等业务是一大利器，看情况吧，只能说尽量不使用。

20、对事务一致性要求不高的业务，如日志表等，优先选择存入MongoDB。

其自身支持的sharding分片功能，增强了横向扩展的能力，开发不用过多调整业务代码。

库表设计规范

21、表必须有主键，例如自增主键。

这样可以保证数据行是按照顺序写入，对于SAS传统机械式硬盘写入性能更好，根据主键做关联查询的性能也会更好，并且还方便了数据仓库抽取数据。从性能的角度来说，使用UUID作为主键是个最不好的方法，它会使插入变得随机。

解读:

主键最好设为一个业务无关的自增id

22、禁止使用分区表。

分区表的好处是对于开发来说，不用修改代码，通过后端DB的设置，比如对于时间字段做拆分，就可以轻松实现表的拆分。但这里面涉及一个问题，查询的字段必须是分区键，否则会遍历所有的分区表，并不会带来性能上的提升。此外，分区表在物理结构上仍旧是一张表，此时我们更改表结构，一样不会带来性能上的提升。所以应采用切表的形式做拆分，如程序上需要对历史数据做查询，可通过union all的方式关联查询。另外随着时间的推移，历史数据表不再需要，只需在从库上dump出来，即便捷地迁移至备份机上。

解读:

看情况，我们zabbix上使用了分区表。按时间分区和清理历史数据。

字段设计规范

23、用DECIMAL代替FLOAT和DOUBLE存储精确浮点数。

浮点数的缺点是会引起精度问题，请看下面一个例子：

|  |
| --- |
| mysql> CREATE TABLE t3 (c1 float(10,2),c2 decimal(10,2));        Query OK, 0 rows affected (0.05 sec) >mysql> insert into t3 values (999998.02, 999998.02);     Query OK, 1 row affected (0.01 sec) >mysql> select \* from t3; +-----------+-----------+ | c1        | c2        | +-----------+-----------+ | 999998.00 | 999998.02 | +-----------+-----------+ 1 row in set (0.00 sec) |

可以看到c1列的值由999998.02变成了999998.00，这就是float浮点数类型的不精确性造成的。因此对货币等对精度敏感的数据，应该用定点数表示或存储。

24、使用TINYINT来代替ENUM类型。

采用enum枚举类型，会存在扩展的问题，例如用户在线状态，如果此时增加了：5表示请勿打扰、6表示开会中、7表示隐身对好友可见，那么增加新的ENUM值要做DDL修改表结构操作了。

解读:

我们这边也是这样的

25、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。

选择字段的一般原则是保小不保大，能用占用字节少的字段就不用大字段。比如主键，强烈建议用int整型，不用uuid，为什么？省空间啊。空间是什么？空间就是效率！按4个字节和按32个字节定位一条记录，谁快谁慢太明显了。涉及几个表做join时，效果就更明显了。更小的字段类型占用的内存就更少，占用的磁盘空间和磁盘I/O也会更少，而且还会占用更少的带宽。  
  
有不少开发人员在设计表字段时，只要是针对数值类型的全部用int，但这不一定合适，就比如用户的年龄，一般来说，年龄大都在1~100岁之间，长度只有3，那么用int就不适合了，可以用tinyint代替。又比如用户在线状态，0表示离线、1表示在线、2表示离开、3表示忙碌、4表示隐身等，其实类似这样的情况，用int都是没有必要的，浪费空间，采用tinyint完全可以满足需要，int占用的是4字节，而tinyint才占用1个字节。  
  
int整型有符号（signed）最大值是2147483647，而无符号（unsigned）最大值是4294967295，如果你的需求没有存储负数，那么建议改成有符号（signed），可以增加int存储范围。

int(10)和int(1)没有什么区别，10和1仅是宽度而已，在设置了zerofill扩展属性的时候有用，例：

|  |
| --- |
| root@localhost(test)10:39>create table test(id int(10) zerofill,id2 int(1)); Query OK, 0 rows affected (0.13 sec) root@localhost(test)10:39>insert into test values(1,1); Query OK, 1 row affected (0.04 sec) root@localhost(test)10:56>insert into test values(1000000000,1000000000); Query OK, 1 row affected (0.05 sec) root@localhost(test)10:56>select \* from test; +------------+------------+ | id         | id2        | +------------+------------+ | 0000000001 |          1 | | 1000000000 | 1000000000 | +------------+------------+ 2 rows in set (0.01 sec) |

解读:

贺总讲的非常详细了，认同

26、字段定义为NOT NULL要提供默认值。

从应用层角度来看，可以减少程序判断代码，比如你要查询一条记录，如果没默认值，你是不是得先判断该字段对应变量是否被设置，如果没有，你得通过java把该变量置为''或者0，如果设了默认值，判断条件可直接略过。

NULL值很难进行查询优化，它会使索引统计更加复杂，还需要MySQL内部进行特殊处理。

27、尽可能不使用TEXT、BLOB类型。

增加存储空间的占用，读取速度慢。

# 索引规范

28、索引不是越多越好，按实际需要进行创建。

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。适当的索引对应用的性能至关重要，而且在MySQL中使用索引它的速度是极快的。遗憾的是，索引也有相关的开销。每次向表中写入时（如INSERT、UPDATEH或DELETE），如果带有一个或多个索引，那么MySQL也要更新各个索引，这样索引就增加了对各个表的写入操作的开销。只有当某列被用于WHERE子句时，才能享受到索引的性能提升的好处。如果不使用索引，它就没有价值，而且会带来维护上的开销。

29、查询的字段必须创建索引。

如：1、SELECT、UPDATE、DELETE语句的WHERE条件列；2、多表JOIN的字段。

30、不在索引列进行数学运算和函数运算。

无法使用索引，导致全表扫描。

例：SELECT \* FROM t WHERE YEAR(d) >= 2016;  
由于MySQL不像Oracle那样支持函数索引，即使d字段有索引，也会直接全表扫描。  
应改为----->  
SELECT \* FROM t WHERE d >= '2016-01-01';

31、不在低基数列上建立索引，例如‘性别’。

有时候，进行全表浏览要比必须读取索引和数据表更快，尤其是当索引包含的是平均分布的数据集是更是如此。对此典型的例子是性别，它有两个均匀分布的值（男和女）。通过性别需要读取大概一半的行。在种情况下进行全表扫描浏览要更快。

32、不使用%前导的查询，如like ‘%xxx’。

无法使用索引，导致全表扫描。

|  |
| --- |
| 低效查询   SELECT \* FROM t WHERE name LIKE '%de%';   ----->   高效查询   SELECT \* FROM t WHERE name LIKE 'de%'; |

33、不使用反向查询，如 not in / not like。

无法使用索引，导致全表扫描。

34、避免冗余或重复索引。

联合索引IX\_a\_b\_c(a,b,c) 相当于 (a) 、(a,b) 、(a,b,c)，那么索引 (a) 、(a,b) 就是多余的。

SQL设计规范

35、不使用SELECT \*，只获取必要的字段。

* 消耗CPU和IO、消耗网络带宽；
* 无法使用覆盖索引。

36、用IN来替换OR。

低效查询

SELECT \* FROM t WHERE LOC\_ID = 10 OR LOC\_ID = 20 OR LOC\_ID = 30;

----->

高效查询

SELECT \* FROM t WHERE LOC\_IN IN (10,20,30);

解读:

丁奇大大说5.7是做过优化的，用OR也可以。

37、避免数据类型不一致。

SELECT \* FROM t WHERE id = '19';

----->

SELECT \* FROM t WHERE id = 19;

解读:

类型不一致还会有逻辑错误的！有可能出来的结果不是你想要的结果！

38、减少与数据库的交互次数。

INSERT INTO t (id, name) VALUES(1,'Bea');  
INSERT INTO t (id, name) VALUES(2,'Belle');  
INSERT INTO t (id, name) VALUES(3,'Bernice');  
----->  
INSERT INTO t (id, name) VALUES(1,'Bea'), (2,'Belle'),(3,'Bernice');

Update … where id in (1,2,3,4);

Alter table tbl\_name add column col1, add column col2;

39、拒绝大SQL，拆分成小SQL。

低效查询  
SELECT \* FROM tag  
JOIN tag\_post ON tag\_post.tag\_id = tag.id  
JOIN post ON tag\_post.post\_id = post.id  
WHERE tag.tag = 'mysql';  
可以分解成下面这些查询来代替  
----->  
高效查询  
SELECT \* FROM tag WHERE tag = 'mysql'  
SELECT \* FROM tag\_post WHERE tag\_id = 1234  
SELECT \* FROM post WHERE post\_id in (123, 456, 567, 9098, 8904);

40、禁止使用order by rand()

SELECT \* FROM t1 WHERE 1=1 ORDER BY RAND() LIMIT 4;

---->

SELECT \* FROM t1 WHERE id >= CEIL(RAND()\*1000) LIMIT 4;