## 数据库基础设计规范

1 所有数据库对象名称必须使用小写字母并用下划线分割（mysql对大小写敏感的）

2 所有数据库对象名称禁止使用mysql保留的关键字

3 数据库对象的命名要能做到见名识义，并且最好不要超过32个字符（减少不必要的网络开销）

4 临时库表必须以tmp为前缀并以日期为后缀

5 备份表，备份表必须以bak为前缀并以日期为后缀

6 所有存储相同数据的列名和列类型必须一致（如果作为关联列的数据类型不同的话在查询的时候就会进行数据类型转换，失去关联的作用）

7 5.6之后的所有表都必须用Innodb存储引擎，Innodb支持事务，行级锁，更好的恢复性，高并发下性能更好

8 数据库和表的字符集统一使用UTF-8（兼容性更好），UTF8字符集汉字占3个字节，ASCLL码占用1个字节

9 尽量控制单表数据量的大小，建议控制在500w以内，500w并不是mysql数据库的限制，只是会影响查询，备份，恢复

10 谨慎使用mysql分区表

分区表在物理上表现为多个文件，在逻辑上表现为一个表

谨慎选择分区建，跨分区查询效率可能更低

建议采用物理分表的方式管理大数据

11 尽量做到冷热数据分离，减小表的宽度

mysql限制是4096列主要是减少磁盘的IO，保证热数据的内存缓存命中率

更有效的利用缓存，避免读入无用的冷数据

经常一起使用的列放到一个表中

12 禁止在表中建立预留字段

预留字段的命名很难做到见名识义

预留字段无法确认存储的数据类型，所以无法选择合适的类型

13 禁止在数据库中存储图片，文件等二进制数据

禁止在线上做数据库压力测试

禁止从开发环境，测试环境直接连接生产环境数据库

## 数据库索引设计规范

1 限制每张表上的索引数量，建议单张表索引不超过5个

索引并不是越多越好！索引可以提高效率同样可以降低效率

禁止给表的每一列都建立单独的索引

2 每个innodb表必须有一个主键

不使用更新频繁的列左右主键，不使用多列主键

不使用UUID，MD5,HASH，字符串列作为主键

主键建议选择使用自增ID值

3常见的索引列建议

select update delete 语句的where从句中的列

包含在order by, group by,distinct中的字段

多表join的关联列

4 如何选择索引列的顺序（联合索引 索引列是从左到右的顺序来使用的）

区分度最高的列放在联合索引的最左侧

尽量把字段长度小的列放在联合索引的最左侧

使用最频繁的列放在索引的最左侧

5 避免建立冗余索引和重复索引

重复索引 primary key(id),index(id),unique index(id)

冗余索引 index(a,b,c) index(a,b) index(a)

6 对于频繁的查询优先考虑使用覆盖索引

覆盖索引：就是包含了所有查询字段的索引（指的是查询所用到的字段 where后的字段 ，orderby的字段，groupby的字段，select后面的字段）

避免innodb表进行索引的二次查找

可以把随机IO变为顺序IO加快查询效率

7 尽量避免使用外键约束

不建议使用外键约束，但一定在表与表之间的关联键上建立索引

外键可用于保证数据的参照完整性 但建议在业务端实现

外键会影响父表与字表的写操作从而降低性能

## 数据库字段设计规范

1 优先选择符合存储需要的最小的数据类型

将字符串转化为数字类型存储（减少字节数）

对于非负型的数据来说，要优先使用无符号整形来存储（无符号相对于有符号可以多出 一倍的存储空间）

varchar(n)中的n代表的是字符数，而不是字节数

过大的长度会消耗更多的内存

2 避免使用text，blob的数据类型

建议把blob或是text列分离到单独的扩展表中

text和blob类型只能使用前缀索引

3 避免使用 enum 数据类型

修改enum值需要使用alter语句（存在操作风险）

enum类型的orderby 操作效率低，需要额外的操作

禁止使用数值作为ENUM的枚举值

4 尽可能把所有列定义为NOT null

索引null列需要额外的空间来保存，所以要占用更多的空间

进行比较和计算时要对null值做特别的处理

5 字符串存储日期型的数据（不正确的做法）

缺点: 无法用日期函数进行计算和比较

用字符串存储日期要占用更多的空间

6 使用TIMESTAMP或DATETIME类型存储时间

TIMESTAMP 1970-01-01 00:00:01 ~2038-01-19 03:14:07

TIMESTAMP占用4字节和int相同，但比int可读性更高

超出的需要用DATETIME存储

7 同财务相关的金额类数据，必须使用decimal类型

1非精准浮点：

float，double

2 精准浮点：

decimal

decimal类型为进准浮点数，在计算时不会丢失精度

占的空间由定义的宽度决定

可用于存储比bigint更大的整数数据

## 数据库sql开发规范

1 建议使用预编译语句进行数据库操作

只传参数，比传递sql语句更高效

相同语句可以一次解析，多次使用，提高处理效率

防止sql注入的风险

2 避免数据类型的隐式转换

隐式转换会导致索引失效

3 充分利用表上已经存在的索引

避免使用双%好的查询条件 如a like ‘%123%’

一个sql只能利用到复合索引中的一列进行范围查询

使用left join或not exists来优化not in操作

4 程序连接不同的数据库使用不同的账号，禁止跨库查询

为数据库迁移和分库分表留出余地

降低业务耦合度

避免权限过大而产生的安全风险

5 禁止使用select \* 必须使用select <字段列表>查询

消耗更多的cpu和IO以及网络带宽资源

无法使用覆盖索引

可减小表结构变更带来的影响

6 禁止使用不含字段列表的insert语句

在insert中指定插入字段的名称 （insert into t(c1,c2,c3) values(‘a’,’b’,’c’)）

可减少对表结构变更带来的影响

7 避免使用子查询，可以把子查询优化为join操作

子查询的结果集无法使用索引

子查询会产生临时表操作，如果子查询数据量大则严重影响效率

子查询产生的临时表会消耗过多的cpu及IO资源，产生大量的慢查询

8 避免使用join 关联太多的表

每join一个表会多占用一部分内存（join\_buffer\_size）

会产生临时表操作，影响查询效率

mysql最多允许关联61个表，建议不超过5个

9 减少同数据库的交互次数

数据库更适合处理批量操作

合并多个相同的操作到一起，可以提高处理效率

10 使用in代替or

in的值不要超过500个

in操作可以更有效的利用索引

11 禁止使用order by rand()进行随机排序

会把表中的所有符合条件的数据装载到内存中进行排序

会消耗大量的cpu和IO以及内存资源

推荐在程序中获取一个随机值，然后从数据库中获取数据的方式

12 where从句中禁止对列进行函数转换和计算

对列进行函数转换或计算会导致无法使用索引

13 在明显不会有重复值时使用union all 而不是union

union会把所有数据放到临时表中后再去进行去重操作（数据量大会消耗大量内存）

union all 不会再对结果集进行去重操作

14 拆分复杂的大sql为多个小sql

mysql一个sql只能使用一个cpu进行计算

sql拆分后可以通过并行执行来提高处理效率

## 数据库操作行为规范

1 超过100万行的批量写操作，要分批多次进行操作

大批量的写操作可能会造成严重的主从延迟

binlog日志为row格式时会产生大量的日志

避免产生大事务操作

2 对大表数据结构的修改一定要谨慎，会造成严重的锁表操作，尤其是生产环境

对于大表使用pt-online-schema-change修改表结构

避免大表修改产生的主从延迟

避免在对表字段进行修改时进行锁表

3 禁止为程序所使用的账号赋予super权限

当达到最大连接数限制时，还允许1个有super权限的用户连接

super权限只能留给DBA处理问题的账号使用

4 对于程序连接数据库账号，遵循权限最小原则

程序使用数据库的账号只能在一个db下使用，不准跨库

程序使用账号原则上不准有drop权限

## 数据库三范式

1NF：字段不可分（原子性 字段不可再分，否则就不是关系数据库）

2NF：有主键，非主键字段依赖主键（唯一性一个表只说明一个事物）

3NF: 非主键字段不能相互依赖（每列都与主键有直接关系，不存在传递依赖）

### 第一范式（1NF）

表的列具有原子性，不可再分解，即列的信息，不能分解，只要数据库是关系型数据库就自动的满足第一范式，数据库表的每一列都是不可分割的原子数据项，而不能是集合，数组，记录等非原子数据项。如果实体中的某个属性有多个值时，必须拆分为不同的属性，通俗理解即一个字段值存储一项信息。

### 第二范式

第二范式是在第一范式的基础上建立起来的，即满足第二范式必须先满足第一范式。第二范式要求数据库表中的每个实例或行必须可以被唯一地区分，为实现区分通常需要我们设计一个主键来实现

即满足第一范式前提，当存在多个主键的时候，才会发生不符合第二范式的情况，比如有两个主键，不能存在这样的属性，它只依赖于其中一个主键，这就是不符合第二范式，通俗理解是任意一个字段都只依赖表中的同一个字段

### 第三范式

满足第三范式必须先满足第二范式。简而言之，第三范式要求一个数据库表中不包含已在其它表中已包含的非主键字段。就是说，表的信息如果能被推导出来，就不应该单独的设计一个字段来存放，很多时候，我们为了满足第三范式往往会把一张表分成多张表

即满足第二范式前提，如果某一属性依赖于其他非主键属性，而其他非主键属性有依赖于主键，那么这个属性就是间接依赖于主键，这被称作传递依赖于主属性。通俗解释就是一张表最多只存两层同类型信息

反三范式

没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，提高读性能，就必须降低范式标准，适当保留冗余数据，具体做法是在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑，降低方式就是增加字段减少了查询时的关联，提高查询效率，因为在数据库的操作中查询的比例要远远大于DML的比例，但是反范式化一定要适度，并且在原本已满足三范式的基础撒花姑娘在做调整

# 数据库分区

## HASH分区表

查看数据库是否有分区功能

SHOW PLUGINS;

有partition属性就代表该数据库支持分区

HASH分区的特点

根据MOD（分区键,分区数）的值把数据行存储到表的不同分区中

数据可以平均的分布在各个分区中

HASH分区的键值必须是一个INT类型的值，或是通过函数可以转为INT类型



图中是可以用来建立分区表的函数

sql示例：

CREATE TABLE `customer\_login\_log` (

`customer\_id` INT(10) UNSIGNED NOT NULL COMMENT '登录用户ID',

`login\_time` TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP

ON UPDATE CURRENT\_TIMESTAMP COMMENT '用户登录时间',

`login\_ip` INT(10) UNSIGNED NOT NULL COMMENT '登录IP',

`login\_type` TINYINT(4) NOT NULL COMMENT '登录类型:0未成功 1成功'

) ENGINE=INNODB DEFAULT CHARSET=utf8 COMMENT='用户登录日志表'

PARTITION BY HASH(customer\_id)

PARTITIONS 4;

## 范围分区（RANGE）

RANGE分区特点

根据分区键值的范围把数据行存储到表的不同分区中

多个分区的范围要持续，但是不能重叠

默认情况下使用VALUES LESS THAN属性，即每个分区不包括指定的那个值

sql示例:

CREATE TABLE `customer\_login\_log` (

`customer\_id` INT(10) UNSIGNED NOT NULL,

`login\_time` DATETIME NOT NULL ,

`login\_ip` INT(10) UNSIGNED NOT NULL,

`login\_type` TINYINT(4) NOT NULL

) ENGINE=INNODB

PARTITION BY RANGE ( YEAR(login\_time) ) (

PARTITION p2 VALUES LESS THAN (2020),

PARTITION p2 VALUES LESS THAN (2021),

PARTITION p2 VALUES LESS THAN (2022),

PARTITION p2 VALUES LESS THAN MAXVALUE,

);

## LIST分区

按分区键取值的列表进行分区

各分区的列表值不能重复

每一行数据必须能找到对应的分区列表，否则数据插入失败

sql实例：

CREATE TABLE `list` (

`customer\_id` INT(10) UNSIGNED NOT NULL COMMENT '登录用户ID',

`login\_time` TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP

ON UPDATE CURRENT\_TIMESTAMP COMMENT '用户登录时间',

`login\_ip` INT(10) UNSIGNED NOT NULL COMMENT '登录IP',

`login\_type` TINYINT(4) NOT NULL COMMENT '登录类型:0未成功 1成功'

) ENGINE=INNODB DEFAULT CHARSET=utf8 COMMENT='用户登录日志表'

PARTITION BY LIST(login\_type)(

PARTITION p0 VALUES IN(1,3,5,7,9),

PARTITION p1 VALUES IN(2,4,6,8)

);

-- 查询分区表各分区数据量

SELECT table\_name,partition\_name,partition\_description,table\_rows

FROM information\_schema.`PARTITIONS`

WHERE table\_name='customer\_login\_log'

-- 增加分区

ALTER TABLE customer\_login\_log ADD PARTITION (PARTITION p4 VALUES LESS THAN (2018));

--删除分区

ALTER TABLE customer\_login\_log DROP PARTITION P0;

根据分区删除过期数据比直接删除表中数据要更便捷

---分区数据迁移sql

ALTER TABLE customer\_login\_log EXCHANGE PARTITION `p0`

WITH TABLE customer\_login\_log\_bak;

表分区的条件：

customer\_login\_log\_bak表是非分区表

customer\_login\_log\_bak表是非临时表

customer\_login\_log\_bak和customer\_login\_log的结构相同

customer\_login\_log\_bak表不包含外键约束

customer\_login\_log\_bak表中不包含不满足p0分区边界数据

使用分区表的注意事项

1结合业务场景选择分区键，避免跨分区查询；

2对分区表进行过滤的查询最好在where从句中都包括分区键

3对于具有主键或唯一索引的表，主键或唯一索引必须是分区键的一部分

# Mysql执行计划

执行计划能告诉我们：

sql如何使用索引

连接查询的执行顺序

查询扫描的数据行数

EXPLAIN 查询sql执行计划

列解释

ID列中的数据为一组数字，表示执行select语句的顺序，ID值相同时，执行顺序由上至下，ID值越大优先级越高，越先被执行