大厂MySQL使用规范

军规适用场景：并发量大、数据量大的互联网业务

军规：介绍内容

解读：讲解原因，解读比军规更重要

一、基础规范

（1）必须使用InnoDB存储引擎

解读：支持事务、行级锁、并发性能更好、CPU及内存缓存页优化使得资源利用率更高

（2）必须使用UTF8字符集

解读：万国码，无需转码，无乱码风险，节省空间

（3）数据表、数据字段必须加入中文注释

解读：N年后谁tm知道这个r1,r2,r3字段是干嘛的

（4）禁止使用存储过程、视图、触发器、Event

解读：高并发大数据的互联网业务，架构设计思路是“解放数据库CPU，将计算转移到服务层”，并发量大的情况下，这些功能很可能将数据库拖死，业务逻辑放到服务层具备更好的扩展性，能够轻易实现“增机器就加性能”。数据库擅长存储与索引，CPU计算还是上移吧

（5）禁止存储大文件或者大照片

解读：为何要让数据库做它不擅长的事情？大文件和照片存储在文件系统，数据库里存URI多好

二、命名规范

（6）只允许使用内网域名，而不是ip连接数据库

（7）线上环境、开发环境、测试环境数据库内网域名遵循命名规范

业务名称：xxx

线上环境：dj.xxx.db

开发环境：dj.xxx.rdb

测试环境：dj.xxx.tdb

从库在名称后加-s标识，**备库**在名称后加-ss标识

线上从库：dj.xxx-s.db

线上备库：dj.xxx-sss.db

（8）库名、表名、字段名：小写，下划线风格，不超过32个字符，必须见名知意，禁止拼音英文混用

（9）表名t_xxx，非唯一索引名idx_xxx，唯一索引名uniq_xxx

三、表设计规范

（10）单实例表数目必须小于500

（11）单表列数目必须小于30

（12）表必须有主键，例如自增主键

解读：

a）主键递增，数据行写入可以提高插入性能，可以避免page分裂，减少表碎片提升空间和内存的使用

b）主键要选择较短的数据类型， Innodb引擎普通索引都会保存主键的值，较短的数据类型可以有效的减少索引的磁盘空间，提高索引的缓存效率

c） 无主键的表删除，在row模式的主从架构，会导致备库夯住

（13）禁止使用外键，如果有外键完整性约束，需要应用程序控制

解读：外键会导致表与表之间耦合，update与delete操作都会涉及相关联的表，十分影响sql 的性能，甚至会造成死锁。高并发情况下容易造成数据库性能，大数据高并发业务场景数据库使用以性能优先

四、字段设计规范

（14）必须把字段定义为NOT NULL并且提供默认值

解读：

a）null的列使索引/索引统计/值比较都更加复杂，对MySQL来说更难优化

b）null 这种类型MySQL内部需要进行特殊处理，增加数据库处理记录的复杂性；同等条件下，表中有较多空字段的时候，数据库的处理性能会降低很多

c）null值需要更多的存储空，无论是表还是索引中每行中的null的列都需要额外的空间来标识

d）对null 的处理时候，只能采用is null或is not null，而不能采用=、in、<、<>、!=、not in这些操作符号。如：where name!='shenjian'，如果存在name为null值的记录，查询结果就不会包含name为null值的记录

（15）禁止使用TEXT、BLOB类型

解读：会浪费更多的磁盘和内存空间，非必要的大量的大字段查询会淘汰掉热数据，导致内存命中率急剧降低，影响数据库性能

（16）禁止使用小数存储货币

解读：使用整数吧，小数容易导致钱对不上

（17）必须使用varchar(20)存储手机号

解读：

a）涉及到区号或者国家代号，可能出现+-()

b）手机号会去做数学运算么？

c）varchar可以支持模糊查询，例如：like“138%”

（18）禁止使用ENUM，可使用TINYINT代替

解读：

a）增加新的ENUM值要做DDL操作

b）ENUM的内部实际存储就是整数，你以为自己定义的是字符串？

五、索引设计规范

（19）单表索引建议控制在5个以内

（20）单索引字段数不允许超过5个

解读：字段超过5个时，实际已经起不到有效过滤数据的作用了

（21）禁止在更新十分频繁、区分度不高的属性上建立索引

解读：

a）更新会变更B+树，更新频繁的字段建立索引会大大降低数据库性能

b）“性别”这种区分度不大的属性，建立索引是没有什么意义的，不能有效过滤数据，性能与全表扫描类似

（22）建立组合索引，必须把区分度高的字段放在前面

解读：能够更加有效的过滤数据

六、SQL使用规范

（23）禁止使用SELECT *，只获取必要的字段，需要显示说明列属性

解读：

a）读取不需要的列会增加CPU、IO、NET消耗

b）不能有效的利用覆盖索引

c）使用SELECT *容易在增加或者删除字段后出现程序BUG

（24）禁止使用INSERT INTO t_xxx VALUES(xxx)，必须显示指定插入的列属性

解读：容易在增加或者删除字段后出现程序BUG

（25）禁止使用属性隐式转换

解读：SELECT uid FROM t_user WHERE phone=13812345678 会导致全表扫描，而不能命中phone索引，猜猜为什么？（这个线上问题不止出现过一次）

（26）禁止在WHERE条件的属性上使用函数或者表达式

解读：SELECT uid FROM t_user WHERE from_unixtime(day)>='2017-02-15' 会导致全表扫描

正确的写法是：SELECT uid FROM t_user WHERE day>= unix_timestamp('2017-02-15 00:00:00')

（27）禁止负向查询，以及%开头的模糊查询

解读：

a）负向查询条件：NOT、!=、<>、!<、!>、NOT IN、NOT LIKE等，会导致全表扫描

b）%开头的模糊查询，会导致全表扫描

（28）禁止大表使用JOIN查询，禁止大表使用子查询

解读：会产生临时表，消耗较多内存与CPU，极大影响数据库性能

（29）禁止使用OR条件，必须改为IN查询

解读：旧版本Mysql的OR查询是不能命中索引的，即使能命中索引，为何要让数据库耗费更多的CPU帮助实施查询优化呢？

（30）应用程序必须捕获SQL异常，并有相应处理

一，核心军规

- 不在数据库做计算，cpu计算务必移至业务层
- 控制单表数据量，单表记录控制在千万级
- 控制列数量，字段数控制在20以内
- 平衡范式与冗余，为提高效率可以牺牲范式设计，冗余数据
- 拒绝3B(big)，大sql，大事务，大批量

二，字段类军规

- 用好数值类型
  tinyint(1Byte)
  smallint(2Byte)
  mediumint(3Byte)
  int(4Byte)
  bigint(8Byte)
  bad case：int(1)/int(11)

- 有些字符转化为数字
  用int而不是char(15)存储ip

- 优先使用enum或set
  例如：`sex` enum (‘F’, ‘M’)

- 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
  bad case：
  `name` char(32) default null
  `age` int not null
  good case：
  `age` int not null default 0

- 不在数据库里存图片

三，索引类军规

- 谨慎合理使用索引
  改善查询、减慢更新
  索引一定不是越多越好（能不加就不加，要加的一定得加）
  覆盖记录条数过多不适合建索引，例如“性别”

- 字符字段必须建前缀索引

- 不在索引做列运算
  bad case：
  select id where age +1 = 10;

- innodb主键合理使用自增列
  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
  如果不指定主键，innodb会使用唯一且非空值索引代替

- 不用外键，请由程序保证约束

四，sql类军规

- 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
  事务时间尽可能短
  bad case：
  上传图片事务

- 避免使用触发器，用户自定义函数，请由程序取而代之

- 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改写为UNION

画外音：最新的mysql内核已经进行了相关优化

- limit高效分页
  limit越大，效率越低
  select id from t limit 10000, 10;
  应该改为 =>
  select id from t where id > 10000 limit 10;

- 使用union all替代union，union有去重开销

- 尽量不用连接join

- 务必请使用“同类型”进行比较，否则可能全表扫面
- 打散批量更新

- 使用新能分析工具
  show profile;
  mysqlsla;
  mysqldumpslow;
  explain;
  show slow log;
  show processlist;
  show query_response_time(percona)

一、基础规范

- 表存储引擎必须使用InnoDB

- 表字符集默认使用utf8，必要时候使用utf8mb4

解读：
（1）通用，无乱码风险，汉字3字节，英文1字节
（2）utf8mb4是utf8的超集，有存储4字节例如表情符号时，使用它

- 禁止使用存储过程，视图，触发器，Event

解读：

（1）对数据库性能影响较大，互联网业务，能让站点层和服务层干的事情，不要交到数据库层

（2）调试，排错，迁移都比较困难，扩展性较差

- 禁止在数据库中存储大文件，例如照片，可以将大文件存储在对象存储系统，数据库中存储路径
- 禁止在线上环境做数据库压力测试
- 测试，开发，线上数据库环境必须隔离

二、命名规范

- 库名，表名，列名必须用小写，采用下划线分隔

解读：abc，Abc，ABC都是给自己埋坑

- 库名，表名，列名必须见名知义，长度不要超过32字符

解读：tmp，wushan谁TM知道这些库是干嘛的

- 库备份必须以bak为前缀，以日期为后缀
- 从库必须以-s为后缀
- 备库必须以-ss为后缀

三、表设计规范

- 单实例表个数必须控制在2000个以内
- 单表分表个数必须控制在1024个以内
- 表必须有主键，推荐使用UNSIGNED整数为主键

潜在坑：删除无主键的表，如果是row模式的主从架构，从库会挂住

- 禁止使用外键，如果要保证完整性，应由应用程式实现

解读：外键使得表之间相互耦合，影响update/delete等SQL性能，有可能造成死锁，高并发情况下容易成为数据库瓶颈

- 建议将大字段，访问频度低的字段拆分到单独的表中存储，分离冷热数据

解读：具体参加《如何实施数据库垂直拆分》

四、列设计规范

- 根据业务区分使用tinyint/int/bigint，分别会占用1/4/8字节
- 根据业务区分使用char/varchar

解读：

（1）字段长度固定，或者长度近似的业务场景，适合使用char，能够减少碎片，查询性能高

（2）字段长度相差较大，或者更新较少的业务场景，适合使用varchar，能够减少空间

- 根据业务区分使用datetime/timestamp

解读：前者占用5个字节，后者占用4个字节，存储年使用YEAR，存储日期使用DATE，存储时间使用datetime

- 必须把字段定义为NOT NULL并设默认值

解读：

（1）NULL的列使用索引，索引统计，值都更加复杂，MySQL更难优化

（2）NULL需要更多的存储空间

（3）NULL只能采用IS NULL或者IS NOT NULL，而在=/!=/in/not in时有大坑

- 使用INT UNSIGNED存储IPv4，不要用char(15)

- 使用varchar(20)存储手机号，不要使用整数

解读：

（1）牵扯到国家代号，可能出现+/-/()等字符，例如+86

（2）手机号不会用来做数学运算

（3）varchar可以模糊查询，例如like‘138%’

- 使用TINYINT来代替ENUM

解读：ENUM增加新值要进行DDL操作

五、索引规范

- 唯一索引使用uniq_[字段名]来命名
- 非唯一索引使用idx_[字段名]来命名
- 单张表索引数量建议控制在5个以内

解读：

（1）互联网高并发业务，太多索引会影响写性能

（2）生成执行计划时，如果索引太多，会降低性能，并可能导致MySQL选择不到最优索引

（3）异常复杂的查询需求，可以选择ES等更为适合的方式存储

- 组合索引字段数不建议超过5个

解读：如果5个字段还不能极大缩小row范围，八成是设计有问题

- 不建议在频繁更新的字段上建立索引
- 非必要不要进行JOIN查询，如果要进行JOIN查询，被JOIN的字段必须类型相同，并建立索引

解读：踩过因为JOIN字段类型不一致，而导致全表扫描的坑么？

- 理解组合索引最左前缀原则，避免重复建设索引，如果建立了(a,b,c)，相当于建立了(a), (a,b), (a,b,c)

六、SQL规范

- 禁止使用select *，只获取必要字段

解读：

*（1）select *会增加cpu/io/内存/带宽的消耗*

（2）指定字段能有效利用索引覆盖

（3）指定字段查询，在表结构变更时，能保证对应用程序无影响

- insert必须指定字段，禁止使用insert into T values()

解读：指定字段插入，在表结构变更时，能保证对应用程序无影响

- 隐式类型转换会使索引失效，导致全表扫描

- 禁止在where条件列使用函数或者表达式

解读：导致不能命中索引，全表扫描

- 禁止负向查询以及%开头的模糊查询

解读：导致不能命中索引，全表扫描

- 禁止大表JOIN和子查询
- 同一个字段上的OR必须改写问IN，IN的值必须少于50个
- 应用程序必须捕获SQL异常

解读：方便定位线上问题