MySQL 开发规范

知数堂发起

日期	版本	作者	说明
2016.7.26	1.0	知数堂	初始化

1 文档简介

文档是经验总结沉淀，方便传播及阅读。 随着时间的推移，文档里的一些知识也许就变成不合时宜，甚至是错的。因此，我们推崇一个理念：尽信书不如无书，要勇于挑战传统，坚持自己实践之后得出的认知。

本文档由知数堂发起，希望能借此促进行业的 MySQL 开发、使用规范，少走弯路，让大家都能在 MySQL 上做出好的应用。

1.1 联系我们

如果您觉得这份规范还不错，有进一步的想法， 或是有兴趣参与这个规范整理和补充，那就联系我们吧。

文档负责人：**叶金荣**（QQ/Weixin： 4700963），**吴炳锡**（QQ/Weixin： 82565387），也欢迎扫描下方二维码加入 QQ 群进交流。

1.2 适用范围

该文档可以用于各类项目中的 MySQL 使用场景，主要是表结构 DDL 设计建议，SQL 编写优化建议，及进行 SQL Review 等场景。

本文档主要适用于 MySQL 5.0 ~ MySQL 5.6 这几个版本，更早或更新的版本可能有些小差异，请读者自行甄别。

2 写在前面

2.1 几个焦点

最佳实践通常是为了获取最佳性能。

- **性能？一致性？**

  可能在支付业务环境中，每秒只需要能支撑住 3000 个事务就够了，在支付业务中，更看重的是数据一致性。而对于性能，可能就需要具体问题具体分析了，平时的性能需求是多少，是否需要应对突发高峰，能突发多大量，预留多少，等等。

- **读写分离**

  一般而言，不在一个事务中的只读请求是可以向只读服务器请求的，也就是所谓的读写分离，这么做可以有效增加整体处理能力。

  在读写分离方案中，需要考虑清楚怎么实施，是否需要 proxy，如何避免只读服务器数据一致性。

- **分库分表**

  需要先澄清一个误区：并不是数据量一大，就一定要分库分表才是解决问题的唯一选项。一般来说，可以先通过使用更好的硬件提升服务器性能，以应对数据量增长带来的负载。

  分库分表更多的是从为了提升可扩展能力，提到整体服务容量，避免单点风险等角度来考虑。

  当然了，进行分库分表后，也会随之带来架构设计方面的难题，比如怎么对分开离散的数据进行聚合，多表关联 JOIN，全量数据排序等。

  所以说，能不分表就尽量不要分表，未必是好事，有得有失。

2.2 MySQL 特点

- mysql 是单进程多线程，不像 Oracle 那样是多进程的
- 每个 mysql 内部线程同时只能用到一个逻辑 cpu 线程
- 每个 SQL 同时只能用到一个逻辑 CPU 线程
- 无执行计划缓存（类似 ORACLE 的 library cache），不过 MySQL 的执行计划解析比较轻量级，效率还不错，这方面不会是瓶颈
- query cache 的更新需要持有 全局 mutex，数据有任何更新都需要等待该 mutex，效率低，且整个表的 query cache 也会失效，因此，强烈建议关闭 query cache
- 没有 thread pool 时，如果有瞬间大量连接请求，性能会急剧下降

2.3 几点建议

- 使用 MySQL 优秀的地方，避免使用其弱势的特性
- 越是简单的 SQL，通常效率是越高的
- 最常用的引擎有： InnoDB，TokuDB。InnoDB 基本上可以覆盖 90%以上的应用场景
- 计算机在处理整数比较浮点数快 N 倍，因此慎用浮点数（可以 N 倍扩大后转换成整型）
- 尽量不要在 DB 里做运算，这部分工作可以在程序端或中间件层面来完成
- 单实例容量建议：机械盘环境控制在 500G 以内，SSD/Pcie SSD 环境可控制在卡容量的 80%以内（不含 binlog 等日志文件）

3 Schema 设计规范

Schema 设计是数据架构设计中非常重要的一个环节。

进行 Schema 设计时，最好先画 ER 图做初稿，再参考下面的规范建议进行调整后形成 Schema SQL 文档，最终交由团队一起讨论评审。

3.1 字段规范

- **写在前面**

  InnoDB 表是索引聚集组织表（IOT），所有的行数据（row data）都是以主键（严格意义讲，是聚集索引）逻辑顺序存储，而二级索引（或称辅助索引，secondary index）的 value 则同时包含主键。

  InnoDB 的最小 I/O 单位是 data page（默认一个 data page 大小是 16KB），在 buffer pool 中的最小单位是 data page（而不是每行数据哦）。因此也可以这么理解，一个 data page 里的热点数据越多，其在 buffer pool 的命中率就会越高。

  MySQL 复制环境中，如果 binlog format 是 row 的，则从库上的数据更新时是以主键为依据进行 apply 的，如果没有主键则将可能会有灾难性的后果。

- **字段设计参考**
  - 每个表建议不超过 30-50 个字段
  - 如果遇到 BLOB、TEXT 字段，则尽量拆出去，再用主键做关联
  - 在够用的前提下，选择尽可能小的字段，用于节省磁盘和内存空间
  - 涉及精确金额相关用途时，建议扩大 N 倍后，全部转成整型存储（例如把分扩大百倍），避免浮点数加减出现不准确问题

常用数据类型

列类型	表达的范围	存储需求
TINYINT[(M)] [UNSIGNED] [ZEROFILL]	-128 到 127 或 0 到 255	1 个字节
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]	-32768 到 32767 或 0 到 65535	2 个字节
INT[(M)] [UNSIGNED] [ZEROFILL]	-2147483648 到 2147483647 或 0 到 4294967295	4 个字节
BIGINT[(M)] [UNSIGNED] [ZEROFILL]	-9223372036854775808 到 9223372036854775807 或 0 到 18446744073709551615	8 个字节
DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]	整数最大位数（M）为 65，小数位数最大（D）为 30	变长
DATE	YYYY-MM-DD	3 个字节
DATETIME	YYYY-MM-DD HH:MM:SS(1001 年到 9999 年的范围)	5.6.4 版本前：8 个字节 5.6.4 版本后：5 个字节，外加 0-3 个动态字节（后面是否有毫秒、微秒）
TIMESTAMP	YYYY-MM-DD HH:MM:SS（1970 年到 2037 年的范围）	5.6.4 版本前：4 个字节

		5.6.4 版本后：4 个字节，外加 0-3 个动态字节（后面是否有毫秒、微秒）
CHAR(M)	0<M<=255（建议 CHAR(1)外，超过此长度的用 VARCHAR）	M 个字符（所占空间跟字符集等有关系）
VARCHAR(M)	0<M<65532/N	M 个字符（N 大小由字符集，以及是否为中文还是字母数字等有关系）
TEXT	64K 个字符	所占空间跟字符集等有关系

- **主建选择**
  - 强烈建议使用 INT/BIGINT 并且自增做为主键，顺序 insert 效率更高，表空间碎片率更低
  - 主键避免采用字符型，如 VARCHAR/CHAR/UUID，会导致原本可以顺序写入的请求变成随机写入，效率更低
  - 如果需要用 UUID 产生一个较大的随机数，则可用 uuid_short() 来代替，uuid_short()会生成 bigint 类型数据
  - 拆分时如果需要全局唯一主键，可采用发号器服务、 redis 的全局自增，或某个全局 DB 里统一分配等多种方式生成全局唯一值

- **关于 NOT NULL**

  建议每个字段都设置上 NOT NULL 属性，可减小存储开销及避免索引失效的问题。

  同时，为了避免程序写入失败，还可以增加默认值。如：

  a1 varchar(32) not null default ‘’

  或

  c1 int(10) not null default ‘0’

【未完待续。。。】