分库分表

垂直拆分

- 两个维度
  - 拆库：将一个数据库，拆分成多个提供不同业务数据处理能力的数据库
  - 拆表：如果单表数据量过大，还可能需要对单表进行拆分。改动太大，尽量少用
- 优点
  - 单库（单表）变小，便于管理和维护
  - 对性能和容量有提升作用
  - 改造后，系统和数据复杂度降低
  - 可以作为微服务改造的基础
- 缺点
  - 库变多，管理变复杂
  - 对业务系统有较强的侵入性
  - 改造过程复杂，容易出故障
  - 拆分到一定程度就无法继续拆分
- 一般做法
  - 梳理清楚拆分范围和影响范围
  - 检查评估和重新影响到的服务
  - 准备新的数据库集群复制数据
  - 修改系统配置并发布新版上线

水平拆分

水平拆分就是直接对数据进行分片，有分库和分表两个具体方式，但是都只是降低单个节点数据量，但不改变数据本身的结构。这样对业务系统本身的代码逻辑来说，就不需要做特别大的改动，甚至可以基于一些中间件做到透明

- 一般原则
  - 一般情况下，如果数据本身的读写压力较大，磁盘 IO 已经成为瓶颈，那么分库比分表要好。分库将数据分散到不同的数据库实例，使用不同的磁盘，从而可以并行提升整个集群的并行数据处理能力。相反的情况下，可以尽量多考虑分表，降低单表的数据量，从而减少单表操作的时间，同时也能在单个数据库上使用并行操作多个表来增加处理能力
- 优点
  - 解决容量问题
  - 比垂直拆分对系统影响小
  - 部分提升性能和稳定性
- 缺点
  - 集群规模大，管理复杂
  - 复杂 SQL 支持问题（业务侵入性、性能）
  - 数据迁移问题
  - 一致性问题

框架和中间件

- 框架
  - TDDL
  - Apache ShardingSphere-JDBC
- 中间件
  - DRDS（商业闭源）
  - Apache ShardingSphere-Proxy
  - MyCat/DBLE
  - Cobar
  - Vitness
  - KingShard

数据迁移

- 全量
  - 全量数据导出和导入
  - 步骤
    - 业务系统停机
    - 业务系统停机数据库迁移，校验一致性
    - 然后业务系统升级，接入新数据库
    - 直接复制的话，可以 dump 后全量导入（如果是）异构数据，需要用程序来处理
- 全量+增量
  - 依赖于数据本身的时间戳
  - 步骤
    - 先同步数据到最近的某个时间戳
    - 然后在发布升级时停机维护
    - 然后在发布升级时停机维护
    - 最后升级业务系统，接入新数据库
- binlog+全量+增量
  - 通过主库或者从库的 binlog 来解析和重新构造数据，实现复制
  - 一般需要中间件等工具的支持
  - 可以实现多线程，断点续传，全量历史和增量数据同步
    - 实现自定义复杂异构数据结构
    - 实现自动扩容和缩容，比如分库分表到单库单表，单库单表到分库分表，分4个库表到分64个库表
- 迁移工具 ShardingSphere-scaling
  - 支持数据全量和增量同步
  - 支持断点续传和多线程数据同步
  - 支持数据库异构复制和动态扩容
  - 具有 UI 界面，可视化配置