- sysbench报错 max_prepared_stmt_count
  - max_prepared_stmt_count 该设置多少
    - 和线程数有关
      - oltp_read_only：线程数 * 表数量 * 5 + 线程数
      - oltp_write_only：线程数 * 表数量 * 4 + 线程数
      - oltp_read_write：线程数 * 表数量 * 9 + 线程数
      - oltp_insert：0. (oltp_insert场景没有prepare语句)
      - sysbench在压测开始时，预先建立好线程，而不是用到了再建
    - 和表数量有关 —— 可以看出sysbench会在同一个session内，查询多个表
    - 过大有什么坏处 —— 缓存在线程内，值设置过大的话，占用缓存即内存过多
    - 为什么sysbench需要prepare statement —— 只测试执行消耗时间，不考虑编译和优化消耗的时间
  - prepare
    - 目的：一次编译，多次执行
      - 编译
        - 权限分析
        - 语法解析
        - 语义解析
        - 统计信息分析
      - 优化
        - sql改写
        - 生成执行计划
      - 执行（非prepare） —— 执行结果缓存
        - oracle
          - result_cache_mode —— 考虑redis缓存结果集?
          - result_cache_max_size
        - mysql
          - query_cache_size
            - 8.0后取消
            - 考虑redis缓存结果集?
          - query_cache_type
      - 返回（fetch，非prepare）
    - 商业数据库需要prepare
      - oracle
        - _optim_peek_user_binds
        - prepare查询 —— 共享池
          - data dictionary cache
          - library cache —— 共享游标shard cursor
            - 父游标 —— heap0 —— 里面有记录依赖对象
            - 子游标
              - heap6
              - 不能共享原因 —— V$SQL_SHARED_CURSOR —— mismatch
                - BIND_MISMATCH
                  - 绑定变量的数据类型，有不同长度
                  - 绑定变量的参数值，有不同长度
                  - 以2个变量为例：
变量1，如果4种等级的长度都有。
变量2，如果也是4种等级的长度都有，
那就4*4=16种不同的cursor版本。
                - OPTIMIZER_MODE_MISMATCH
                - AUTH_CHECK_MISMATCH
      - sqlserver
        - adhoc查询
          - 内存使用量：sys.dm_exec_cached_plans where objtype='adhoc' —— cache缓存上限
            - 16万条
            - max plan cache size
          - 优化点：Optimize for Adhoc Workloads=False改成true —— 如果都使用存储过程，没必要修改
          - 参数化，还是adhoc，不是prepare，这个跟oracle不同，也跟其他数据库不同
        - prepare查询 —— 共享池 —— plan cache优化
          - depend object越多，占用内存越多 —— 依赖对象发生改变，需要重新解析
          - 拆成不同的小存储过程
          - adhoc和prepare共同使用共享池，互相挤压，减少adhoc使用
    - 开源数据库也需要prepare
      - mysql
        - prepare泄露
          - ps.events_statements_summary_by_thread_by_event_name
          - general_log —— grep Prepare
        - prepeare的生命周期：session结束 —— 如果在存储过程中使用prepare，在存储过程结束时不会deallocate，直到session结束
        - 没有共享池 —— 思路：交给连接池实现
          - mycat
            - client--prepare协议-->mycate--常规协议--> mysql
            - prepare语句缓存在mycat —— 但生命周期任然是session级
          - 爱可生dble —— 同mycat，也是前端走server-side prepare，但是后端走常规协议
          - jboss —— PreparedStatementCache
      - pg
        - 通用plan
        - 自定义plan —— 5次转通用plan
        - 没用共享池 —— 思路：连接池来实现
          - pgbouncer
            - session pool —— 支持prepare statement
              - 协议层prepare
                - pgbouncer内部的优化机制，可用缓存plan
                - 一般适合频繁+简单sql场景
              - 显式PREPARE/DEALLOCATE
                - pg数据库自身的机制
                - 一般适合频繁+复杂sql场景
            - tranasaction pool
              - 不支持prepare statement —— 每个查询可能分配到不同的后端 —— 阿里云rds pg默认的模式
              - 注意，是指到后端pg。以事务active为准
              - 应该首选该模式，缓解连接数过大问题
            - statemnent pool
            - 引入场景
              - 优点 —— 短连接
                - 连接数>= cpu核数*20
                - pgbouncer到db的连接数=数据库cpu的核数*[2~4]
              - 缺点 —— pgbouncer是单进程模式 —— 解决方式：多个pgbouncer+slb
          - pgpool2
          - pgcat
            - session mode —— 支持prepare statement —— prepared_statements_cache_size
            - transaction mode —— 支持prepare statement —— 解决方式：pgcat重命名prepare的语句，并且做好了client和sever之间mapping关系的记录
    - prepare是否是必要的? —— 要看prepare阶段做的事情的复杂程度 vs 执行阶段的时间长度
      - 大型复杂sql，编译、优化阶段资源消耗高，耗时久 —— 共享池缓存
      - 简单sql，编译、优化阶段消耗低