交付案例分享

数据库

全年技术保障 · 性能调优 · 排障与数据救援

SQL Server / MySQL / Oracle 三大引擎全栈覆盖

DB

CRITICAL SYSTEM

EXTERNAL USE / INTERNAL USE / CONFIDENTIAL | 2026.09

目录 CONTENTS

四类场景，一条从"日常保障"到"极端情况"的能力线

我们交付的
不只是"解决问题"

而是让关键业务系统背后的数据库，长期处在被看见、被管理、被兜底的状态。

SQL Server / Oracle / MySQL

覆盖 6 类行业 8 个真实案例

客户名称已做脱敏处理

01 **全年技术支持**
双引擎核心库的常态化巡检与 7×24 值守，把风险挡在故障之前 P04 – P07

02 **性能调优**
把"时快时慢""报表白屏"这类模糊抱怨，变成可验证的优化结果 P08 – P10

03 **排障与数据救援**
AlwaysOn 同步延迟的根因追查、4 小时抢回的置疑数据库 P11 – P13

04 **疑难攻坚与应急支持**
7 轮方案才定位的官方缺陷 P14 – P16

客户与行业覆盖

不同行业的业务逻辑千差万别，但数据库面临的"要命的事"高度相似

6 INDUSTRIES · 3 ENGINES

行业覆盖

6 类行业 · 8 个真实案例

微电子 / 半导体

核心业务双引擎并行，停摆损失按分钟计

50TB 量级 AlwaysOn 稳定保障

电力能源

近百套 MySQL 实例，高并发、全天候不停机

医疗卫生

HIS 系统依赖 AlwaysOn，数据不同步即为业务风险

金融

服务器突发宕机、数据库置疑，需紧急恢复

汽车

报表白屏、数据导入缓慢，直接影响业务效率

生物医药

OA 高并发场景下"时快时慢"，需长期观测定位

技术栈覆盖

三大引擎均具备一线排障与深度优化能力，不依赖厂商现场支持周期。

SQL Server

AlwaysOn 可用性组调优、置疑库修复、DBCC CHECKDB、日志收缩异常处置

Oracle

核心业务库日常运维、巡检基线与容量评估

MySQL

近百实例批量巡检、主主同步、内存泄漏类疑难攻坚

同一套服务体系可跨行业复用——这是方法论，不是运气。

01

CHAPTER 01

全年技术支持

把数据库从"出事再救"变成"全年有人在管"

本章涵盖

上海xx微电子 · 双引擎核心库保障

xx半导体 · 50TB 量级 AlwaysOn 稳定保障

xx电力 · 近百套 MySQL 实例巡检

上海xx微电子 · 全年数据库支持

SQL Server + Oracle 双引擎核心库，月度巡检与 7×24 值守并行

案例 01 · 全年技术支持

项目介绍

· 客户核心数据库同时承载 **SQL Server 与 Oracle** 双引擎，支撑生产与研发关键业务系统

· 对稳定性、性能与业务连续性要求极高

· 需要全年持续的技术保障与快速响应机制

· 每月一次数据库巡检，形成巡检记录与风险清单

月度巡检　变更评审　性能基线　风险清单

交付难点

· 产线依赖数据库，任何服务不可用都会造成产线停摆，**损失按分钟计算**

· 需要 7×24 随时响应，工程师须同时具备 SQL Server 与 Oracle 双栈能力

· 关键业务时期需现场保障，无法只靠远程支持

xx半导体 · 50TB 量级 AlwaysOn 稳定保障

数据零丢失与高效稳定并重——50TB 基数下的交付答卷

案例 02 · 全年技术支持

项目介绍

· **客户背景**：集团总部机密文档库，承载多个厂区的文档数据处理及检索服务

· **架构现状**：采用 SQL Server AlwaysOn 架构，数据量高达 50TB

交付难点

· **体量之痛**：50TB 级别实例，常规的备份、甚至索引重建时间都会呈指数级增加，时常业务访问失败

· **同步风险**：AlwaysOn 在高并发写入和海量数据下，极易出现同步延迟，甚至导致主副本阻塞

怎么做到的

01 **定制化巡检**：针对 50TB 实例定制专属巡检基线，重点监控 AlwaysOn 延迟、长事务以及磁盘 I/O 瓶颈

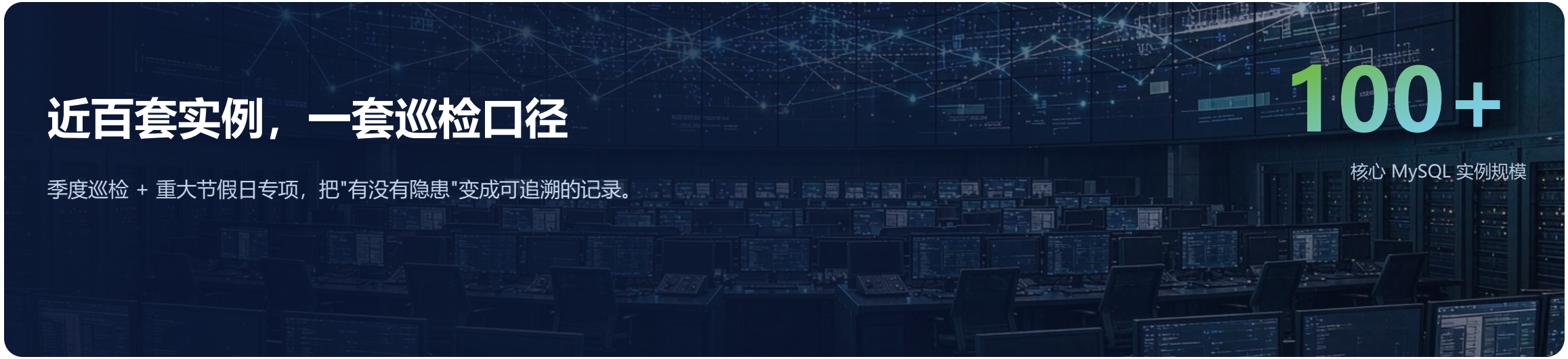
02 **拆分业务台账**：将内容库按业务线拆分，既分散了压力，又让架构更清晰

客户收益 从容应对 50TB 级核心库的稳定运行，AlwaysOn 架构真正做到“敢切换、能兜底”。

xx电力 · 近百套 MySQL 稳定运行保障

规模本身就是难点——靠标准化巡检与隐患台账，而不是人海战术

案例 03 · 全年技术支持



项目介绍

客户有近百套核心 MySQL 实例，需要保障稳定运行；每季度及重大节假日前开展巡检，排查潜在隐患与风险点，形成巡检记录，让"季度体检"成为常态化动作。

交付难点

业务需要高性能、高并发、24 小时不停机，对工程师技术的专业度要求非常高；发生故障时对人员响应速度的要求同样极高，关键业务时期还需要现场保障。

怎么做到的

01 标准化巡检模板，近百实例统一口径

02 隐患台账逐项跟踪，闭环才销项

03 节前专项加固 + 关键期现场保障

02

CHAPTER 02

性能调优

把"时快时慢""报表白屏"这类模糊抱怨，变成可验证的优化结果

本章涵盖

xx生物 · OA系统高并发调优

xxx汽车 · 报表与导入效率优化

xx生物 · OA 系统数据库性能调优

问题非稳定复现——没有捷径，只能把观测周期拉长

案例 04 · 性能调优

CASE · 03

xx生物

员工多、并发高的 OA 系统，页面响应"时快时慢"，是最考验方法论的一类问题。

- 症状：时快时慢，非稳定复现
- 误区：加硬件只能临时缓解

OBSERVE FIRST

先建立基线，再谈优化

非稳定复现的问题，靠一次抓取是找不到根因的。

问题现象

- 客户反馈 OA 系统页面时快时慢，员工多、并发高
- 客户怀疑是数据库存在性能瓶颈
- 问题非稳定复现，无法通过单次抓取定位
- 增加硬件资源只能临时缓解，无法根治

解决路径

01 持续采集慢日志，建立性能基线

02 分期优化：索引、SQL 改写、参数调整

03 以慢查询 Top N 前后对比验证效果

04 优化后纳入巡检项，防止回退

xxx汽车 · 报表与数据导入效率优化

用户能感知的卡顿，必须先用数据固定成技术问题

案例 05 · 性能调优

问题现象

用户可感知

- 报表打开长时间无响应，出现白屏
- 数据导入耗时较长，效率偏低

影响用户体验与业务处理效率——但"慢"本身不可定位，必须先量化。

解决路径

trace 慢查询监控 → Top 100 慢查询 → 索引 / 归档

01 部署 trace 慢查询监控

让"偶发卡顿"变成可追踪、可对比的监控数据

02 结合 Top 100 慢查询定位瓶颈

从一百条最慢语句里找出真正压垮响应时间的那几条

03 索引优化 + 数据归档

从根上降低单次查询成本，而非靠扩容掩盖问题

03

CHAPTER 03

排障与数据救援

最容易被忽视的同步延迟与日志异常，往往是更大故障的前兆

本章涵盖

xx医院 · 同步延迟排障

xx银行 · 数据紧急救援

xx医院 · AlwaysOn 同步延迟排障

真正常见的不是技术不行，而是没人知道它已经不行了

案例 06 · 排障

SQL SERVER · HADR

两节点同步链路

主副本 Primary

同步延迟 · 日志无法收缩

辅助副本 Secondary

⚠ 无监控：从库宕机也无人知晓

HIS 系统的可用性组一旦不同步，读到的就是过期数据——这类问题不会报错，只会悄悄发生。

HIS · MISSION CRITICAL

两个症状，两条根因

数据不同步与日志无法收缩，看似两件事，其实同源。

1 数据不同步

现象｜客户 HIS 系统经常遇到 SQL Server AlwaysOn 可用性组数据不同步。

根因｜因数据变更过多造成延迟，从库同步跟不上主库写入节奏。

处置｜在业务低峰期清理从库内存缓冲池，加快同步效率并消除延迟。

关键点：必须选低峰期执行，优化动作本身不能变成二次故障。

2 日志始终无法收缩

现象｜事务日志持续增长，反复收缩仍然无效，磁盘水位不断告警。

根因｜并非日志本身的问题，而是延迟过高或从库宕机（无监控，宕机了也不知道）。

建议｜建立可用性组健康监控与阈值告警，让异常在第一时间被发现。

启示：日志收缩失败往往是"下游出了事"的报警灯，而不是日志自己的病。

xx银行 · 数据库置疑紧急恢复

处置的第一原则不是"快"，而是"别把事情变得更糟"

案例07 · 数据救援

事件经过

服务器突发异常宕机 › **数据库进入置疑状态** › 数据无法访问

客户的某数据库服务器突发异常宕机，业务受到直接影响，需要紧急恢复。

处置时序

数据完整性优先于恢复速度

01 保护现场：禁止盲目重启、附加或重建数据库

02 分析置疑根因，判断是日志损坏还是非正常关机所致

03 利用 DBCC CHECKDB 对置疑数据库进行检测与修复

04 修复后对数据进行逐项对比校验，确认一致性

05 针对暴露的风险提出优化建议，避免同类事件复发

04

CHAPTER 04

疑难攻坚与应急支持

前面讲的是"我们做过什么"，这一章讲"难的时候我们靠不靠得住"

本章涵盖

复杂案例 · 核心 MySQL 内存泄漏攻坚

应急响应机制

复杂案例 · 核心 MySQL 内存泄漏攻坚

7 轮方案迭代之后才发现：答案不在参数里，而在版本里

案例08 · 疑难攻坚

7 轮方案迭代

每条合理路径都被完整走过并排除

30G 内存瞬时释放

停掉节点 A 同步进程后立刻归还

#85371 MySQL 官方缺陷编号

严重级别 S2（Serious）· 影响开启 binlog_rows_query_log_events 的多源复制实例

7 轮尝试，逐条排除

症状：内存持续增长 → OOM Kill

01 连接池改造（4000 → 1000）	02 内存升级（64G → 128G）
03 buffer_pool 从 90G 降到 60G	04 配置 jemalloc 内存管理器
05 FEDERATED 引擎表改造	06 优化 session 级参数
07 Top 100 慢查询优化	× 问题仍未解决

转折点：在 5.7.17 → 5.7.44 的升级验证中，停掉节点 A 同步进程后内存释放 30G，方向由此反转。

根因锁定与结论

现象层 | 内存占用持续增长，OOM Kill 频发，只能定期重启缓解。

根因层 | 确认为 MySQL 官方 bug #85371：开启 binlog_rows_query_log_events 后，多源复制的 slave SQL 线程持续申请内存却不释放（memory/sql/Log_event 长期 COUNT_FREE = 0）。

处置层 | 升级至修复版本或关闭该参数规避，并建立内存水位监控防止复发。

疑难问题的价值不在答案本身，而在敢于把每条看似合理的路走完再排除。

应急响应机制

让"靠个人英雄"变成"靠流程兜底"

5 STEPS · CLOSED LOOP

01 快速接入，确认影响面

7×24 应急通道第一时间响应，先判断"影响了谁、影响到什么程度"

02 先止损，保全数据

务必做好备份；禁止盲目重启或重建；无法确认根因前，不做不可逆操作

03 取证分析，定位根因

基于日志、监控与现场痕迹还原过程，区分"表象"与"真实原因"

04 修复并逐项校验

修复动作完成后必须做数据对比校验，确认一致性再交付业务

05 复盘加固，闭环收尾

输出根因报告与优化建议，把一次性救援转化为长期防护

响应分级

P1 业务中断 / 数据不可访问
立即介入，最高优先级响应

P2 性能劣化 / 同步异常
当日响应，限时定位并给出方案

P3 咨询与优化建议
纳入排期，结合巡检一并推进

三条铁律

· 先止损，再定位——不让故障范围扩大

· 不冒险操作——数据完整性高于恢复速度

· 必须复盘——每一单都要留下加固动作

THANK YOU

Thanks for your attention

从单次故障救援到全年技术保障，从本地库到近百套实例集群—— 数据库这件事，交给一支成体系的团队，比交给一个人更稳。

如果您对任何方案或服务感兴趣，可随时与我们取得联络，将有专人为您提供一对一的专业咨询服务。

www.sqlfans.cn | 数据库交付与运维服务

EXTERNAL USE / INTERNAL USE / CONFIDENTIAL