



链滴

Mysql 常规面试题

作者: [AshShawn](#)

原文链接: <https://ld246.com/article/1647616074846>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

1. InnoDB 事务的四大特性?
2. InnoDB事务隔离级别有哪些?
3. InnoDB索引
 1. 什么是索引?
 2. 索引的优缺点?
 3. 索引的作用?
 4. 什么情况下需要建索引?
 5. 什么情况下不建索引?
 6. 索引的数据结构
 7. Hash索引和B+树索引的区别?
 8. 为什么B+树比B树更适合实现数据库索引?
 9. 索引有什么分类?
 10. 什么是最左匹配原则?
 11. 什么是聚集索引?
 12. 什么是覆盖索引?
 13. 索引的设计原则?
 14. 索引什么时候会失效?
 15. 什么是前缀索引?
4. 常见的存储引擎有哪些?
5. MyISAM和InnoDB的区别?
6. InnoDBMVCC 实现原理?
7. InnoDB快照读和当前读
8. InnoDB共享锁和排他锁
9. InnoDB大表怎么优化?
10. InnoDBbin log/redo log/undo log
11. bin log和redo log有什么区别?
12. 讲一下MySQL架构?
13. InnoDB分库分表
14. 什么是分区表?
15. 分区表类型
16. 查询语句执行流程?
17. 更新语句执行过程?
18. exist和in的区别?
19. truncate、delete与drop区别?
20. having和where的区别?
21. InnoDB什么是MySQL主从同步?

22. 为什么要做主从同步?
23. 乐观锁和悲观锁是什么?
24. 用过processlist吗?
25. 数据库的三范式是什么?
26. 说一下 MySQL 的行锁和表锁?
27. MySQL 的内连接、左连接、右连接有什么区别?
28. char 和 varchar 的区别是什么?
29. float 和 double 的区别是什么?
30. MySQL 问题排查都有哪些手段?
31. 如何做 MySQL 的性能优化?
32. 为什么要尽量设定一个主键?
33. MySQL中保存钱, 应该使用什么数据类型? 并说明原因
34. 一张自增表里面总共有 7 条数据, 删除了最后 2 条数据, 重启 MySQL 数据库, 又插入了一条数, 此时 id 是几?
35. 死锁排查
36. 分库分表设计,有可能遇到的问题
37. 大数据量查询方案
38. 分布式主键方案,各有什么优缺点
39. explain执行计划如何理解其中的字段
40. 千万数据如何优化?大表查询优化
41. in和exists的区别
42. 数据库中间件,mycat sharding
43. 主从延迟原因,解决方案
44. 数据库连接池
45. sql语句执行过程
46. 日期格式与时区转换
47. 慢查询优化
48. mysql性能分析常用命令
49. blob和text区别
50. 视图的特点/使用场景
51. count(1),count(*),count(id)
52. int(10),char(10),varchar(10)
53. drop,delete,truncate
54. sql执行顺序
55. 如何安全存储用户密码
56. cpu飙升如何排查
57. 读写分离/多数据源

58. heart主从复制原理

59. heart数据库监控工具/常用监控参数/如何查看/如何排查

60. 主从一致性校验/主从数据库不一致怎么处理

61. 一个6亿的表a，一个3亿的表b，通过外键tid关联，你如何最快的查询出满足条件的第50000到第60200中的这200条数据记录。

MySQL

存储引擎

- InnoDB
 - 支持事务
 - 支持行级锁
 - 支持在线热备份
- MyISAM
 - 不支持事务
 - 表级锁
 - 支持空间数据索引
- MyISAM和InnoDB的比较
 - 事务: InnoDB 支持事务, MyISAM 不支持事务
 - 锁: InnoDB 支持行级锁, MyISAM 只支持表级锁
 - 外键: InnoDB 支持外键
 - InnoDB 支持在线热备份
 - MyISAM 支持空间索引

索引

- B+ 树原理
 - 数据结构
 - B+ Tree 是 B 树的一种变形, 它是基于 B Tree 和叶子节点顺序访问指针进行实现, 通常用于数据库和操作系统的文件系统中
 - B+树有两种类型的节点: 内部节点 (也称索引节点) 和叶子节点, 内部节点就是非叶子节点, 内部节点不存储数据, 只存储索引, 数据都存储在叶子节点
 - 内部节点中的key都按照从小到大的顺序排列, 对于内部节点中的一个key, 左子树中的所有key都小于它, 右子树中的key都大于等于它, 叶子节点中的记录也按照key的大小排列
 - 每个叶子节点都存有相邻叶子节点的指针
 - 操作
 - 查找
 - 插入
 - 删除
 - 常见树的特性
 - AVL 树 平衡二叉树, 高度平衡, 相比于红黑树查找更快, 删除较慢, rebalance 概率更高
 - 红黑树 近似平衡的, rebalance 的概率更低
 - B/B+ 树 多路查找树, 出度高, 磁盘IO低, 一般用于数据库系统中
 - B+ 树与红黑树的比较
 - 磁盘IO次数低 B+ 树一个节点可以存储多个元素, 相对于完全平衡二叉树整体的树高度降低了, 磁盘IO效率提高了
 - 磁盘预读特性
 - B+ 树与 B 树的比较
 - B+ 树的磁盘IO更低
 - 遍历效率高 B+ 树相对 B 树冗余了一下非叶子节点, 提高了范围查询效率
- MySQL 索引
 - B+ 树索引
 - 二分查找, 查找速度快
 - B+ 树有序, 排序、分组速度快
 - 聚簇索引 叶子节点的数据域记录着完整的数据记录
 - 辅助索引 叶子节点的数据域记录着主键的值, 如果查询的不是索引构成列或者主键, 则需要回表
 - 哈希索引
 - 快速精确查询, 但是不支持范围查询
 - 适合场景: 等值查询场景, 例如 Redis, Memcached 等这些 NoSQL 中间件
 - 哈希表
 - 全文索引
 - 空间数据索引
- 索引优化
 - 独立的列 索引列不能是表达式的一部分, 也不能是函数的参数
 - 多列索引 在需要使用多个列作为查询条件时, 使用多列联合索引比使用多个单列索引性能更好
 - 索引列的顺序
 - 将选择行最强的索引列放在最前面
 - 索引的选择性: 不重复的索引值和记录总数的比值
 - 前缀索引
 - 对于 BLOB、TEXT 和 VARCHAR 类型的列, 必须使用前缀索引, 只索引开始的部分字符
 - 前缀索引的选取需要根据索引选择性来确定
 - 覆盖索引
 - 定义: 索引包含所有需要查询的字段
 - 索引通常远小于数据行的大小, 只读取索引能大大减少数据访问量
 - 优点
 - 无需回表
- 索引的优点
- 索引的使用条件

查询性能优化

- 使用 Explain 分析 SELECT 查询语句
 - select_type
 - SIMPLE 简单查询
 - UNION 联合查询
 - SUBQUERY 子查询
 - table 查询的表
 - system the table has only one row.this is a special case of the const join type.
 - const 只有一条查询结果&主键/唯一索引
 - eq_ref 联接查询&主键/唯一索引&只有一条查询结果
 - ref 非唯一索引
 - type
 - range 使用索引进行范围查询时
 - index 查询的字段是索引的一部分, 覆盖索引
 - 使用主键排序
 - all 全表扫描
 - possible_keys 可选择的索引
 - key 实际使用的索引
 - rows 扫描的行数
- 优化数据访问
 - 减少请求的数据量
 - 只查询必要的列, 例如将 * 替换为要查询的列
 - 只返回必要的行, 使用 limit 限制返回行数
 - 减少数据库扫描的行数
 - 缓存重复查询的数据, 例如使用 redis 缓存用户登陆数据
 - 使用索引来覆盖查询
- 重构查询方式
 - 做分页, 分批查询

ACID

- Atomicity 事务被视为不可分割的最小单元, 所有操作要么全部成功, 要么全部失败
- Consistency 数据库在事务执行前后都保持一致性状态, 在一致性状态下, 所有事务对同一数据的读取结果都是相同的
- Isolation 一个事务所做的修改在提交以前, 对其他事务是不可见的
- Durability 一旦事务提交, 其所做的修改将会永久保存到数据库中

ACID 之间的关系

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

共享锁 (S Lock): 允许事务读一行数据

原文链接: [Mysql 常规面试题](#)