MySQL

- 执行计划解读
  - id
    - SELECT 查询序列号，ID 越大越先执行，所以执行计划解析从下往上。
  - partitions
  - select_type
    - simple
      - 简单查询，不包含子查询和union。
    - primary
      - 包含复杂的子查询，最外层查询标记为该值。
    - subquery
      - 在 select 或者 where 里包含的子查询，被标记为该值。
    - derived
      - 在 from 列表中包含子查询被标记为该值，MySQL 会递归执行这些子查询，把结果放在临时表。
    - union
      - 若第二个 select 出现在 union 后，被标记为该值；若 union 包含在 from 的子查询中，则被标记为 derived。
    - union result
      - 从 union 表中获取结果的 select
  - table
  - filtered
    - 返回结果的行数占读取行数的百分比，值越大越好。
  - type
    - system
      - 表只有一行数据，相当于系统表。
    - const
      - 通过索引一次就找到，仅匹配一条数据。
    - eq_ref
      - 唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配，常用于主键或唯一索引骚猫。
    - ref
      - 非唯一性索引扫描，返回匹配某个单独值的所有行，用于 =、< 或 > 操作符带索引的列。
    - range
      - 只检索给定范围的行，使用一个索引来选择行，一般使用between、> 或 < 。
    - index
      - 只遍历索引树
    - all
      - 全表扫描
  - key
    - 显示 MySQL 实际使用的索引。如果为 NULL，则没有使用索引查询。
  - possible_keys
    - 指出 MySQL 能在该表中使用哪些索引有助于查询。如果该值为NULL，说明没有使用索引，可以建立索引提高性能。
  - row
    - 根据表统计信息及选用情况，大致估算出找到所需的记录或所需读取的行数，数值越小越好。
  - key_len
    - 表示索引中使用的字节数，通过该列计算查询中使用的索引的长度。在不损失精确性的情况下，长度越短越好显示的是索引字段的最大长度，并非实际使用长度。
  - ref
    - 显示该表的索引字段关联了哪张表的哪个字段
  - extra
    - 包含不适合在其他列中显示但十分重要的额外信息。常见的值如下。
      - using filesort
        - MySQL 会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取，若出现该值，应该优化 SQL。
      - using index
        - 表示 select 覆盖了索引，效率不错。
      - using join buffer
        - 使用链接缓存
      - using temporaty
        - 使用临时表保存中间结果，比如，MySQL 在对查询结果排序时使用临时表，常见于 order by 和 group by，若出现该值，应该优化 SQL。
      - using where
        - where 语句用于限制哪一行
      - disctinct
        - 发现第一个匹配后，停止为当前的行组合搜索更多的行。
- 查询优化
  - 最左匹配原则，针对组合索引。
  - 字段尽量避免 NULL，除非必须储存 NULL。
  - 对于不支持 hash indexes 的数据库可以自己通过一个 hash 函数来手动建立 hash indexes。比如说一个字段是很长的网址，这个时候就可以用 hash 函数对其生成一个 hash code，手动建立 hash indexes。
  - 在 MySQL 5.0 之后，可以使用触发器，在新纪录插入的时候用 hash 函数对其生成对应的 hash code 存入额外的字段内，作为手动的 hash indexes。
  - 生成的 hash code 最好是 integer 而不是 String。
  - 尽量使用去分度高的列作为索引
  - between、> 或 < 以及 like 用不了索引，但 like 使用时，%在后面，可以使用前缀索引。
  - 索引列不能参与计算，保持列“干净”，比如from_unixtime(create_time) = ’2014-05-29’就不能使用到索引，原因很简单，b+树中存的都是数据表中的字段值，但进行检索时，需要把所有元素都应用函数才能比较，显然成本太大。所以语句应该写成create_time = unix_timestamp(’2014-05-29’)。
  - 尽量的扩展索引，不要新建索引。比如表中已经有a的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可。
- 主从复制/读写分离
  - 分析的 7 个问题
    - 主从复制中，可以有N个 slave,可是这些 slave 又不能进行写操作，要他们干嘛?
    - master 的写操作，slaves 被动的进行一样的操作，保持数据一致性，那么 slave 是否可以主动的进行写操作?
    - 主从复制中有master,slave1,slave2,...等等这么多 MySQL 数据库，那比如一个 JAVA WEB 应用到底应该连接哪个数据库?
    - 如果 MySQL proxy , direct , master 他们中的某些挂了怎么办?
    - 当 master 的二进制日志每产生一个事件，都需要发往 slave，如果我们有 N 个 slave,那是发 N 次，还是只发一次? 如果只发一次，发给了slave-1，那slave-2,slave-3,...它们怎么办?
    - 当一个 select 发往 MySQL proxy，可能这次由 slave-2 响应，下次由slave-3 响应，这样的话，就无法利用查询缓存了。
    - 随着应用的日益增长，读操作很多，我们可以扩展 slave，但是如果master 满足不了写操作了，怎么办呢?
- AUTO_INCREMENT 的原理
- 关键词执行顺序
  - WHERE
  - GROUP BY
  - SELECT
  - HAVING
  - ORDER BY
- Innodb 中索引的实现
  - 表数据文件本身就是按 B+Tree 组织的一个索引结构，这棵树的叶节点 data 域保存了完整的数据记录。这个索引的 key 是数据表的主键，因此 InnoDB 表数据文件本身就是主索引。
  - InnoDB的所有辅助索引都引用主键作为data域。
  - 种类
    - 按物理存储分类
      - 聚簇索引
      - 二级索引
    - 按索引字段个数分类
      - 单列索引
      - 联合索引
    - 按字段特性分类
      - 唯一性索引
      - 主键索引
      - 普通索引
      - 前缀索引
- MVCC (Multi Version Concurrency Control)
- 锁
  - 按粒度分
    - 行级锁（粒度最小）
      - 会死锁
    - 页级锁
      - 会死锁
    - 表级锁（粒度最大）
  - 按级别分
    - 共享锁（读锁）
      - SELECT ... LOCK IN SHARE MODE;
    - 排他锁（写锁）
      - SELECT ... FOR UPDATE;
    - 意向锁
      - InnoDB 自动的，无需干预。
- 四大特性
  - Atomicity 原子性
  - Consistency 一致性
  - Isolation 隔离性
  - Durability 持久性
- 隔离级别
  - Read Uncommitted 读未提交
    - 允许脏读，也就是可能读取到其他会话中未提交事务修改的数据，也就是中间状态的数据。
      - 不可重复读重点在于 update 和 delete
  - Read Committed 读已提交
    - 只能读取到已经提交的数据，Oracle 等多数数据库默认都是该级别 (不重复读)。
  - ★ Repeatable Read 可重复读
    - 可重复读，在同一个事务内的查询都是事务开始时刻一致的，InnoDB 默认级别。在 SQL 标准中，该隔离级别消除了不可重复读，但是还存在幻象读。
      - 幻读的重点在于 insert
  - Serializable 序列化
    - 完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞。