MariaDB

对于加锁的处理，可以说就是数据库对于事务处理的精髓所在

优化点

1. 关于对MySQL的SQL_NO_CACHE的理解和用法举例
2. utf8mb4中的 `mb4` 表示**max byte 4**，即最大4个字节；
3. utf8mb4_**_ci中的ci表示**case ignore**，即大小写不敏感； utf8mb4_**_cs中的cs表示**case sensitive**，即大小写敏感；
4. **关于索引 + 复合索引 + LIMIT**

```
-- column_a, column_b为联合索引
-- 以下四种情况，哪些会使用索引，哪些不会？
SELECT * FROM table ORDER BY column_a, column_b;
SELECT * FROM table ORDER BY column_a DESC, column_b;
SELECT * FROM table ORDER BY column_a, column_b DESC;
SELECT * FROM table ORDER BY column_a DESC, column_b DESC;

-- 添加了limit之后呐？
SELECT * FROM table ORDER BY column_a, column_b LIMIT 1;
SELECT * FROM table ORDER BY column_a DESC, column_b LIMIT 1;
SELECT * FROM table ORDER BY column_a, column_b DESC LIMIT 1;
SELECT * FROM table ORDER BY column_a DESC, column_b DESC LIMIT 1;

-- 那“ LIMIT 999999999”呐？
SELECT * FROM table ORDER BY column_a, column_b LIMIT 999999999;
SELECT * FROM table ORDER BY column_a DESC, column_b LIMIT 999999999;
SELECT * FROM table ORDER BY column_a, column_b DESC LIMIT 999999999;
SELECT * FROM table ORDER BY column_a DESC, column_b DESC LIMIT 999999999;

-- 如果是主键排序呐？如下：假设table的主键是id
SELECT * FROM table ORDER BY id DESC LIMIT 1或999999999;
```

知识点

1. InnoDB 的行锁是通过给索引上的索引项加锁来实现的；
2. 只有通过索引条件进行数据检索，InnoDB 才使用行级锁，否则，InnoDB 将使用表锁（锁住索引的所有记录）；
3. InnoDB 的表锁是通过对所有行加行锁实现的。
4. MySQL 会为这张表中所有行加行锁，没错，是所有行。但是呢，在加上行锁后，MySQL 会进行一遍过滤，发现不满足的行就释放锁，最终只留下符合条件的行。虽然最终只为符合条件的行加了锁，但是这一锁一释放的过程对性能也是影响极大的。 MySQL事务隔离级别和实现原理-知乎
5. 意向锁（IX/IS）是一种不与行级锁冲突表级锁。**IX，IS是表级锁，不会和行级的X，S锁发生冲突。只会和表级的X，S发生冲突**
6. Gap（ `间隙锁` ）只在 RR 事务隔离级别存在。因为幻读问题是在 RR 事务通过临键锁和 MVCC 解决的，而临键锁=间隙锁+记录锁，所以间隙锁只在 RR 事务隔离级别存在。； *MySQL InnoDB 锁*
7. innodb不支持FULLTEXT类型的全文索引，但是innodb可以使用sphinx插件支持全文索引，并且效果更好。（ `sphinx` 是一个开源软件，提供多种语言的API接口，可以优化mysql的各种查询）
8. **清空表数据** 参考TRUNCATE_TABLE;
9. 大批量导入到MyISAM表时，请先使用 `ALTER TABLE ... DISABLE KEYS` ，然后再导入数据，最后再恢复（非唯一）索引（ `ALTER TABLE ... DISABLE KEYS` ）。参考
10. MyISAM插入数据时，建议 `insert delayed` （但是容灾性就会有所下降）；
11. 意向锁（IS/IX）针对的是表锁（S/X），当事务需要获取表锁时，必须先获取意向锁（如此避免了逐行检查 `行锁` 来进行“表是否已经被锁定”判断）；
12. 当发生死锁时，InnoDB的方法是，将持有最少行级排他锁的事务回滚；
13. `READ UNCOMMITTED` 隔离级别下, 读不会加任何锁。而写会加排他锁，并到事务结束之后释放。参考
14. 文章最后遗留的问题（“马失前蹄”部分）。
15. undo log不是redo log的逆向过程，其实它们都算是用来恢复的日志：【参考】
    - redo log通常是物理日志，记录的是数据页的物理修改，而不是某一行或某几行修改成怎样怎样，它用来恢复提交后的物理数据页(恢复数据页，且只能恢复到最后一次提交的位置)。
    - undo用来回滚行记录到某个版本。undo log一般是逻辑日志，根据每行记录进行记录。
16. 区分innodb_flush_log_at_trx_commit和sync_binlog
17. 【参考】
    - InnoDB 是自适应哈希索引的（hash 索引的创建由 ==InnoDB 存储引擎自动优化创建==，我们干预不了）。
    - 对于辅助索引，InnoDB 采用的方式是在叶子节点中保存主键值，然后再回表；
    - 多个单列索引在多条件查询时优化器会选择最优索引策略，可能==只用一个索引，也可能将多个索引都用上==。
    - 但是多个单列索引底层会建立多个 B+索引树，比较占用空间，也会浪费搜索效率 所以 多条件联合查询时最好建联合索引。

- 联合索引：遇到`范围查询`（>、<、between、like）、`不等于`、`or 连接索引失效`就会停止匹配；

18. 【参考】好文章
    - B-Tree可以对<，<=，=，>，>=，BETWEEN，IN，以及不以通配符开始的LIKE使用索引。
    - 如果查询是连接多个表，仅当ORDER BY中的所有列都是第一个表的列时才会使用索引。
    - ORDER BY子句可以不满足索引的最左前缀要求，那就是前导列为常量时
    - filesort排序的缓存大小配置参数sort_buffer_size/max_length_for_sort_data。
    - 当对连接操作进行排序时，如果ORDER BY仅仅引用第一个表的列，MySQL对该表进行filesort操作，然后进行连接处理，此时，EXPLAIN输出"Using filesort"；**否则**，MySQL必须将查询的结果集生成一个临时表，在连接完成之后进行filesort操作，此时，EXPLAIN输出"Using temporary;Using filesort"。
    - 聚簇索引保证关键字的值相近的元组存储的物理位置也相近（所以字符串类型不宜建立聚簇索引，特别是随机字符串，会使得系统进行大量的移动操作）。
    - MySQL中，只有Memory存储引擎显示支持hash索引，是Memory表的默认索引类型
    - innodb自适应hash索引设置参数innodb_adaptive_hash_index。
    - 索引是在存储引擎中实现的，而不是在服务器层中实现的
    - 引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了
    - 尽量不要让字段的默认值为NULL
    - MySQL无法使用前缀索引做order by和group by
    - 应尽量避免在 where 子句中使用 or 来连接条件【待考究】
    - 查看索引使用情况：Handler_read_key/Handler_read_rnd_next

19. 【参考】
    - 表空间 → 段（leaf/non-leaf）→ 区 → 页
    - 区是由连续的页组成的空间，在任何情况下，每个区的大小都为1MB。
    - `事务ID列` / `undo_log中的trx_id` 大小是6个字节；
    - 如果 InnoDB 表没有定义主键，还会增加一个6字节的 rowid 列。
    - 那么新的问题又来了，多长的 VARCHAR 是保存在单个数据页中的，多长开始会保存在 BLOB 中？InnoDB存储引擎表是索引组织的，即 B+Tree 结构，这样每个页中至少应该有两条记录（不然就退化成链表了）。因此，如果页中只能存下一条记录，那么 InnoDB 引擎就会自动将行数据放到溢出页中。经过测试，这个数字是 8098 ，如果少于这个长度，一个页中就可以放入至少两行数据，VARCHAR 类型的行数据就不会被放到 BLOB 页中去。

20. 对于涉及到`全表扫描`或`大块更新`等操作时，innodb_page_size可以设置16k（默认）或更大（v8.x版本，32k或64k），但OLTP型数据库（此时，innodb_page_size如果太大，容易发生竞争）或SSD硬盘（SSD底层block更加小）时，最好设置更小的值；【参考】

优秀博文

Innodb中的事务隔离级别和锁的关系 | 何为幻读？ | 更多关于ORDER BY MYSQL排序按照自定义的顺序排序 | 图文并茂讲解Mysql事务实现原理

```
select * from A order by IF(ISNULL(a),1,0),a desc
```

行级锁、表级锁、索引锁 | MySQL的锁机制 - 记录锁、间隙锁、临键锁 | 全面了解mysql锁机制（InnoDB）与问题排查 | SSD基本原理 | 聊聊MVCC和Next-key Locks | 数据库MVCC如何解决可重复读问题？ | DATETIME数据类型简介 | MySQL InnoDB 锁 | MYSQL MVCC实现原理 | MySQL到底是怎么解决幻读的？ | mysql myisam vs innodb | 详解MySql的InnoDB中意向锁的作用 | mysql自增长联合主键 | MySQL Online DDL，还是要谨慎 | MySQL 8.0 新特性之函数索引 | MySQL日志系统：redo log、binlog、undo log 区别与作用 | MYSQL MVCC多版本并发控制底层原理及实现机制 | MySQL锁优化 | Innodb中RR隔离级别能否防止幻读？ | MySQL · 引擎特性 · InnoDB 数据文件简述 (←待研究) | MySQL索引为何选择B+树？