Algorithm

49. Group Anagrams

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        std::unordered_map<string, std::vector<string>> mp;
        for(auto& str : strs){
            std::string tmp(str);
            std::sort(str.begin(), str.end());
            mp[str].push_back(tmp);
        }

        std::vector<vector<string>> result;
        for(auto &item : mp){
            result.push_back(item.second);
        }
        return result;
    }
};
```

Reiew

OLTP Through the Looking Glass, and What We Found There

这是一篇关于OLTP(online transaction processing)数据库的展望性文章。

在传统数据库中，数据库文件存在于硬盘中，通过B数进行组织；基于锁的并发控制；支持多线程；这些结构是在1970年代开始形成。现代电脑的处理器，内存，网络已经同30年前有很大不同，数据库往往能够整个放在内存中，大多数事务能够在微妙级别完成。但是数据库的架构却没有发生大的变化。

作者通过一系列的实验的出了如下图所示的结论：

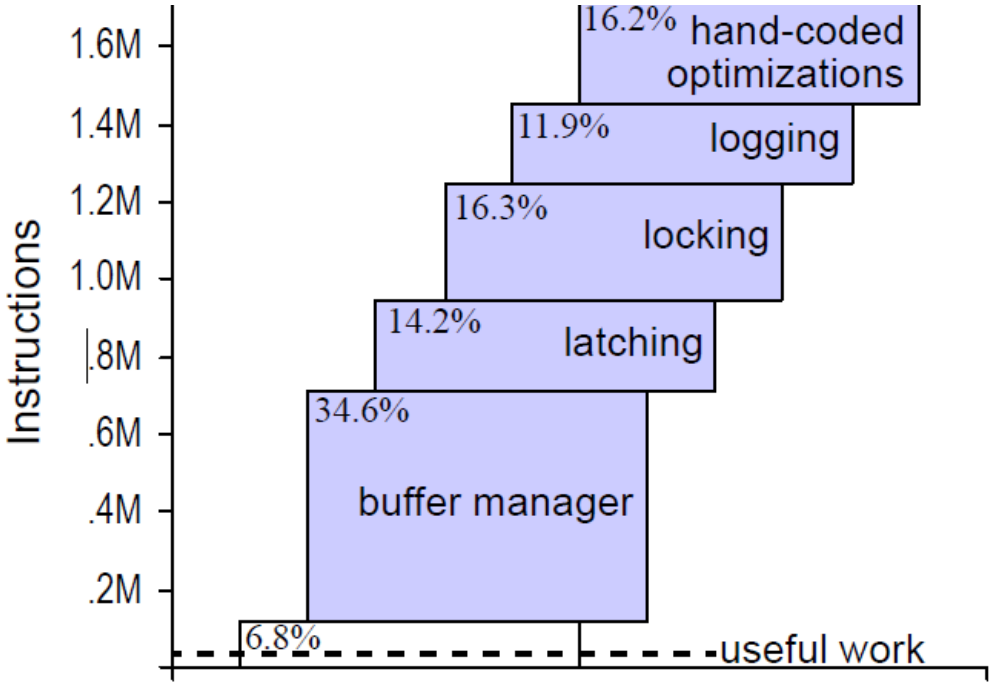

Figure 1. Breakdown of instruction count for various DBMS components for the New Order transaction from TPC-C. The top of the bar-graph is the original Shore performance with a main memory resident database and no thread contention. The bottom dashed line is the useful work, measured by executing the transaction on a no-overhead kernel.

通过统计可以得到，在数据库操作指令中，只有7%左右的指令是用在了真正的操作上。大多数时间都是在缓冲管理，锁，日志等方面。

OLTP的发展趋势

集群计算

现有的数据库架构都是在1970年代设计，典型特诊是基于共享内存，多处理器操作。而在过去20多年里，计算机往往以集群形式对外提供服务，应对大型的计算任务。因此新的数据库系统需要重新设计，以便于在这种集群上运行。

内存驻留数据库

目前计算机的内存显著增长，已经可以将整个数据库系统放入内存中运行。

单线程运行

Tools

数据库的锁

加锁范围：全局锁，表锁和行锁三类。 全局锁

对整个数据库实例加锁。Flush tables with read lock（FTWRL），典型应用场景是逻辑备份。在备份过程中数据库处于只读状态。

在主库上备份，备份期间数据库不能更新，业务停摆 在备库上备份，备份期间从库不能执行主库同步过来的binlog，会导致主从延迟。 官方字段的mysqldump 使用参数-single-transaction时候，导数据之前会启动一个事务，来确保拿到一致性视图。但这个功能需要引擎支持事务。

这里不建议使用readonly来取代全局锁：

readonly状态可以当做其他逻辑，比如主备判断 异常处理机制上有差异。FTWRL只有如果客户端发生异常断开，mysql会自动释放这个锁。而设置为readonly之后，会导致长时间不可用。 readonly对超级用户是失效的，slave上的同步线程就是超级用户。 表级锁

1. 表锁，2. 元数据锁

表锁语法：lock tables ... read/write, 可以用unlock主动释放锁。lock table除了限制其他线程读写外，也会限制本线程的后续操作。

MDL（metadata lock).不需要显式使用，在访问一个表时候会自动加上。

在MySQL5.5上，对一个表做增删改查操作的时候，加MDL读锁。 对表结构做变更操作的时候，加MDL写锁。 MDL在事务执行开始时身亲，事务提交后释放。如果在事务阻塞的时候进行修改表操作，则后续的事务都会阻塞住，导致线程占用满，系统崩溃。

如何安全地给小表加字段

解决长事务，事务如果一直不提交，就会一直占用锁。 alter table里设定等待时间，如果这个指定的时间内拿不到锁，不阻塞后续业务。 ONLINE DDL

拿MDL写锁 降级成MDL读锁 真正做DDL 升级成MDL写锁 释放MDL锁 1，2,4,5如果没有锁冲突，执行时间非常短，第三部占用了DDL大部分时间，这期间这个表可以正常读写、因此成为“online”.

Share 事务的隔离

隔离

在RR级别下，事务T启动时候会创建一个视图read_view，之后事务T执行期间，即使有其他事务修改了数据，事务T看到的仍然跟在启动时看到的一样。

begin/start transaction开始的事务时候其实并不是事务的起点，而是第一个操作InnoDB表的语句，事务才真正启动。如果马上想启动一个事务，可以使用start transaction with consistent snapshot这个命令。

视图

在mysql中，有两个视图的概念。

1. view,用查询语句定义的虚拟表。
2. InnoDB在实现mvcc时用到的一致性读视图，即consistent read view，用于支持RC和RR隔离级别。

快照在mvcc是如何工作的

在RR隔离级别下，事务在启动时就拍了个快照，这个快照是基于整个库的。这时候会有一个transaction id，是一个严格递增的数据。每个数据有多个版本，当一个事务开始时，便分配给这个事务数据一个新的版本，为row trx_id，旧的版本要保留，并且在新的数据版本中，能够有信息能够直接拿到它。

即数据表的行记录，可能存在多个版本，每个版本具有不同的row trx_id。 view并不是真实存在的，而是根据当前版本和undo log计算出来的。

事务在启动时候，只会承认事务版本小于等于自身事务id的数据。数据版本的可见性规则，基于数据的row trx_id和这个一致性视图的对比结果得到。 row trx_id分成了几种情况，

1. 已提交事务
2. 未提交事务集合
3. 未开始事务

3. 未开始事务

对于一个事务视图来说，除了自己的更新总是可见以外，有三种情况：

1. 版本未提交，不可见
2. 版本已提交，但是是在视图创建后提交的，不可见
3. 版本已提交，而且是在视图创建前提交的，可见

更新逻辑

更新数据都是先读后写的，只能读当前值，称为“当前读”。（current read） 可重复读的核心就是“一致性读”，而事务更新数据的时候，只能用当前读。如果当前的记录的行锁被其他事务占用的话，就需要进入锁等待。