# 一种高效的 SQL 语句执行线程①

洪承煜, 杨尚琴, 陈 浩

(中国石化石油物探技术研究院, 南京 211103)

**摘 要**: 在采集、处理、解释一体化软件平台中, 不可避免地有大量的 SQL 语句执行, 而在这些 SQL 语句执行中, 不需要返回的 SQL 语句又占主要比例. 通过设计 SQL 语句缓存区, 并把不需要返回的 SQL 语句实时放入此缓存区, 最后通过设计高效的执行线程, 对此缓存区内的 SQL 语句进行实时的处理. 这种方法既分离了业务与数据库操作, 也提高了业务对数据库操作请求的效率.

**关键词**: SQL 语句; 线程; 缓存区

## Efficient SQL Statement Execution Threads

HONG Cheng-Yu, YANG Shang-Qin, CHEN Hao

(SINOPEC Geophysical Research Institute, Nanjing 211103, China)

**Abstract**: During the acquisition, processing and interpretation integrated software platform, there are a large number of SQL statements inevitably executed, while running these SQL statements, SQL statements that do not need to return occupy to a major proportion. This paper designs the SQL statements cache and SQL statements that do not need to return real-time put into this cache, and finally by designing the thread of executing SQL statements, SQL statements in the cache real-time process. This method does not only the separation of business and database operations, but also improves the business efficiency of requests for database operations.

**Key words**: SQL statements; threads; cache

在勘探软件领域, 采集、处理、解释一体化[1]软件平台中, 数据库操作越来越频繁, 特别是存在大量不需要返回数据的数据库操作(SQL 语句), 如 INSERT 语句、UPDATE 语句、DELETE 语句等. 而这些高频率的数据库操作, 严重影响到业务的处理速度. 其中业界对这些不需要返回数据的数据库操作, 一般采取放置后台线程(数据库操作线程)中执行, 以减轻对业务处理产生的影响. 它们有各种不同的实现方式, 但基本的思想都是通过发送消息到数据库操作线程的消息队列[2]中, 而相对于这种方法, 本设计的优点是: ①直接控制 SQL 语句缓存区, 当发生队列溢出时, 能及时发现, 并及时扩展; ②SQL 语句缓存区的大小能够根据现场的实际情况, 动态扩展. ③通过同步原语[3]控制 SQL 语句执行线程的运行, 当 SQL 语句缓存区无可执行的 SQL 语句时, 不触发事件, 线程被阻塞; 当 SQL 缓存区有 SQL 语句需要执行时, 触发事件, 线程运行.

## 1 线程执行程序的设计

本文设计的重点是把不需要返回数据的数据库操作, 放到后台线程执行, 在执行时为了能体现本设计的优点, 以下给出了具体的设计要求:

(1) 把不需要返回数据的数据库操作提交给在后台数据库操作线程, 从而避免业务处理线程中此类数据库操作;

(2) 分配 SQL 语句的缓存区, 由需要进行数据库操作的业务线程把相应的 SQL 语句提交到此缓存区中, 并由数据库操作线程从队列中取出 SQL 语句提交给数据库服务器执行;

(3) SQL 语句的缓存区大小可以通过数据库表进行配置, 程序启动时从该表中加载缓存区大小, 若该

① 收稿时间:2013-06-21;收到修改稿时间:2013-07-22

表中没有配置, 则使用程序中的默认值;

(4) 当 SQL 语句的提交速度大于数据库服务器的执行速度, 而导致 SQL 语句缓存区溢出时, 程序能够按系统的设置数值, 自动地扩展此缓存区大小, 并把新的缓存区大小写入数据库配置表中; 当程序再次启动时, SQL 语句的缓存区大小使用配置表中的值;

(5) 当 SQL 缓存区无 SQL 语句时, 数据库操作线程可以自动阻塞, 避免了线程的忙等待, 节省 CPU 时间.

根据以上设计要求, 下面设计了主要的数据结构和操作功能, 最后实现了线程执行程序及其相关的操作功能.

### 1.1 主要数据结构和操作功能

程序中用到的主要的数据结构如下表 1 所示.

表 1 主要数据结构定义

| SQL 语句缓存区的控制结构体 | SQL 语句在缓存区中的位置描述结构体 |
|---|---|
| typedef struct {<br>uint32 _in;<br>uint32 _used;<br>void *_st;<br>void *_end;<br>}BufCtrlBlk; | typedef struct {<br>uint32 _offset;<br>uint32 _len;<br>}SqlOffsetLen; |

(1) SQL 语句缓存区的控制结构体中有 4 个字段, 前两个字段的数据类型是 32 位的整型, 后两个字段的数据类型是 void 型的指针. 其中_in 表示缓存区当前可以空间的起始位置; _used 表示缓存区中有多少字节的空间被使用了; _st 指向数据缓存区的开始位置; _end 指向数据缓存区的结束位置.

(2) SQL 语句在缓存区中的位置描述结构体中有 2 个字段, _offset 表示当前 SQL 语句存放的起始位置相对于缓存区开始的偏移量; _len 表示当前 SQL 语句的长度.

有了程序的主要数据结构, 在此系统中, 需要的主要操作功能有如下 5 部分: ①主程序初始化缓存区、同步原语等资源; ② 数据库操作线程执行函数: void SqlThread (void *param), 它负责缓存区中所有的 SQL 语句的执行; ③ 把 SQL 语句添加到缓存区功能操作, 它的声明是: bool AddSQLCommand( char *sql, uint32 len, void *db), 它实现了把数据库连接句柄是 db, SQL 语句的长度是 len 的 sql 语句添加到缓存区中; ④ 从 SQL 语句缓存区中拷贝一条 SQL 语句给数据库操作线程处理: void HandleSQLCommad(); ⑤在线程执行函数中, 当获得一条待执行的 SQL 语句后, 交给 void Exc SQLCommand(char *sql, uint32 len, void *db)去执行.

### 1.2 操作功能的实现

根据上面的主要数据结构和操作功能, 本文设计了如下图 1 的程序操作流程图.

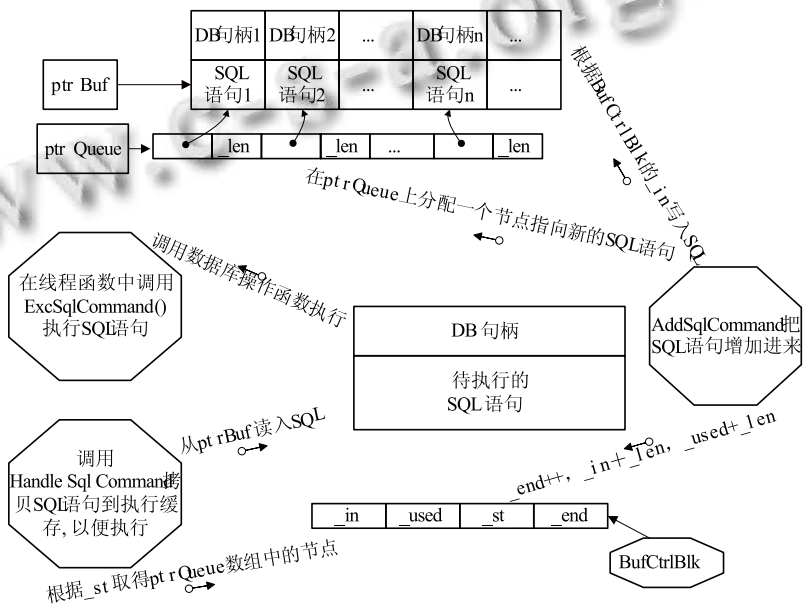


图 1 程序操作流程图

有上图 1 程序操作流程可得, 此系统设计的关键主要有 5 个部分组成, 它们分别实现是: ① 资源分配, 如图 2 所示; ② 执行每一条待执行的 SQL 语句, 如图 3 所示; ③ 添加一条 SQL 语句放入缓存区, 如图 4 所示; ④ 从缓存区中取出一条 SQL 语句待执行, 如图 5 所示; ⑤ 执行一条 SQL 语句, 如图 6 所示.

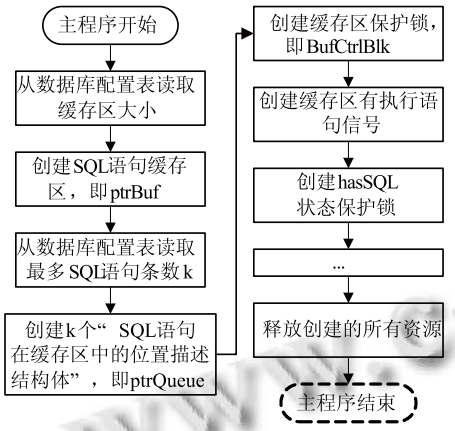


图 2　初始化资源流程图

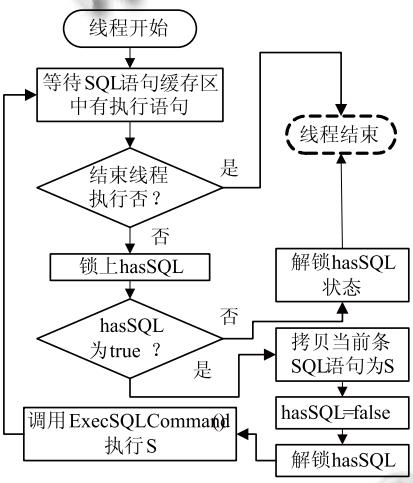


图 3　数据操作线程函数流程图

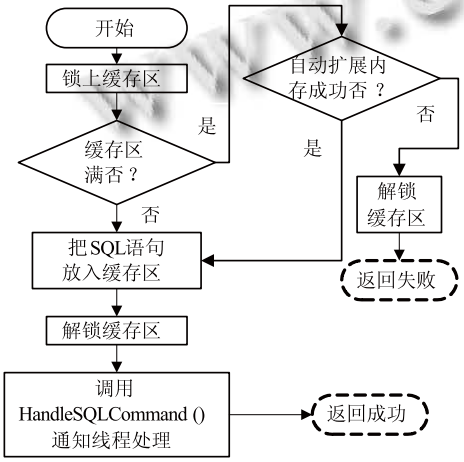


图 4　添加一条 SQL 语句到缓存区流程图

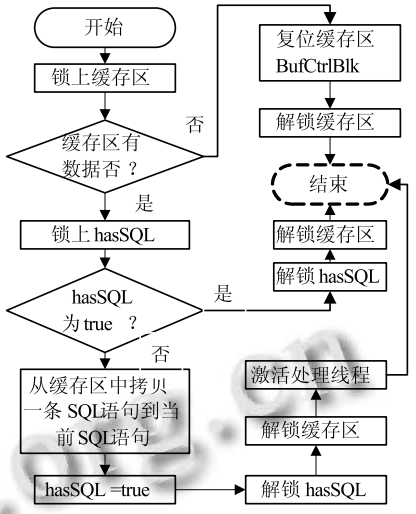


图 5　从缓存区取出一条 SQL 语句流程图


图 6　执行 SQL 语句流程图

下面详细说明这 5 部分的执行过程.

(1) 主程序, 即此进程启动时的主线程中, 对图 1 交互中使用到的资源进行了管理. 如图 2 所示. ①主线程先从数据库配置表中读出了 SQL 语句缓存区的大小, 并创建了 SQL 语句缓存区; ②为了更加快速、便捷的控制 SQL 语句缓存区, 本文设计了另一个标识每一条 SQL 语句在缓存区中的起始位置和长度的缓冲区, 即图 1 中的 ptrQueue 指向的缓存区, 它也根据数据库配置表中的值创建; ③创建缓存区保护锁, 即保护图 1 中 ptrQueue 的操作, 也保护 BufCtrlBlk, 这儿没有分别使用两把锁来分开保护, 是因为这两个数据结构在并发时是同时被操作的; ④创建激活数据库操作线程的信号; ⑤创建 hasSQL 状态保护锁, 当 hasSQL 为 true 时, 表示已经取出了一条 SQL 语句待执行了, 这时执行线程就不需要再去 SQL 缓冲区中取, 可以等当前 SQL 语句执行完再去取下一条 SQL 语句执行; ⑥主线程创建、启动数据库线程, 并可以去做其它事情; ⑦等待数据库操作线程结束; ⑧释放上面创建的所有资源, 结束进程, 主程序结束.

(2) 数据库操作执行函数, 如图 3 所示. ①等待 SQL 语句缓存区中有 SQL 语句, 即等待业务线程添加 SQL 语句到 SQL 缓存区中, 并激活此线程; ②判断是否要线程结束, 是, 则跳去⑧执行, 否, 则执行下一步;

③锁上 hasSQL，判断是否为 true，否，则解锁 hasSQL，跳到⑧执行，是，则执行下一步；④把当前待执行的 SQL 语句拷贝到一个临时空间；⑤把 hasSQL 设置为 false，并解锁 hasSQL；⑥使用临时空间中的 SQL 语句调用 ExecSQLCommand()去执行；⑦跳到(1)执行；⑧结束线程，即退出数据库操作线程. 其中在上面④⑤⑥步中，使用了临时空间来保存待执行的 SQL 语句是为了减少上锁的时间，以便提高并发执行效率. 因为不使用临时空间，锁必须在 ExecSQLCommand()后解锁，而 ExecSQLCommand()函数的执行既需要较长的时间，也不能保证执行的时间是常数，这样上锁的时间加长，影响并发效率.

(3) 业务线程添加一条 SQL 语句去缓冲区，如图 4 所示. ①锁上缓存区；②判断缓存区是否已满，是，则执行下一步，否，则跳④执行；③自动扩展缓存区，并判断是否成功，是则执行下一步，否，则跳⑧执行；④把 SQL 语句添加到缓存区中；⑤解锁缓存区；⑥调用 HandleSQLCommand()，激活数据库操作线程；⑦返回添加成功标志；⑧解锁缓冲区，并返回添加失败标志.

(4) 从 SQL 语句缓存区中取出一条待执行指令给数据库操作线程执行，如图 5 所示. ①锁上缓存区；②判断缓存区是否有 SQL 语句，是，则跳⑤执行，否，则执行下一步；③复位缓存区；④解锁缓存区，并结束执行；⑤锁上 hasSQL 判断是否为 true，是，则解锁 hasSQL，并跳④执行，否，则执行下一步；⑥从缓冲区拷贝一条 SQL 语句到当前待处理处；⑦把 hasSQL 设置为 true，即告诉操作线程当前待处理处有 SQL 语句了；⑧解锁 hasSQL；⑨解锁缓冲区；⑩激活数据库操作线程；⑪结束执行.

上面复位缓存区中，为了快速、简便，使用环形缓冲区，即可通过简单的指针指向移动来达到复位，而不用重新对缓存区的资源进行释放再分配过程.

(5) 把要执行的 SQL 语句提交给数据库执行，如图 6. ①利用 SQL 语句和与它相对应的数据库句柄如图 1 中所示，提交数据库服务器执行；②关闭处理结束的数据库连接；③调用 HandleSQLCommand()，从缓存区中获取下一条 SQL 语句，激活数据库操作线程继续执行；④结束执行.

## 2 结语

按照上述的设计方法，首先实现：①SQL 语句执行线程；②执行线程所需的空间和同步等资源的申请和初始化；③执行线程所需的空间和同步等资源的释放. 然后在采集、处理、解释平台中的数据库部分作以下修改：(1)在数据库应用的启动时，调用上②实现的功能；(2)接着创建和启动上①实现的执行线程；(3)改写相应的 SQL 语句执行处为向 SQL 语句缓冲区添加语句；(4)在数据库应用的结束处，调用上③实现的功能，以便释放系统资源.

在使用本方法前，采集、处理、解释平台中的数据库部分，发现当有大量同时写数据库时，数据库服务器的处理时间有很大的延时，如写数据库量为 4000 条 update 语句，传输的数据值为 500KB，写数据库耗时 50 秒到 134 秒之间，导致数据库服务器长时间的失去响应，严重影响业务的交互体验.

通过使用上述方法实现的数据库操作线程来执行，由于写数据库操作被安排在后台，并且 SQL 缓存能够自动扩充，因此写数据库操作对主程序的运行影响可以忽略不计，大大提高了数据库服务器的业务处理能力，也大大提高了程序的性能，对采集、处理、解释平台中数据库业务的处理和分离起到了重要的作用.

本文实现的方法，主要应用于运行过程中有大量不需要返回数据的数据库操作(UPDATE 语句、INSERT 语句、DELETE 语句等)的程序中，在有类似的数据库设计中具有较好的推广意义.

## 参考文献

1 赵贤正等.高精度三维地震采集处理解释一体化勘探技术与管理.中国石油勘探,2010,(2) .

2 Gilmore WJ. Beginning PHP and MySql(3rd Edtion),2009.

3 Stevens WR, et al. UNIX 网络编程.第 1 卷:套接口 API.北京:清华大学出版社,2006.

4 Silberschatz A, Korth HF, Sudarshan S. Database System Conncepts(6th Edition), 2009.

5 Stephens RK, Plew RR. Database Design, 2001.