PostgreSQL

Разработка приложений

Часть первая.

Зачем нужны СУБД

- Унифицированный доступ к унифицированным данным из разных систем, платформ и языков.
- Средство обеспечения целостности данных.
- Как оно бывает
  - Мусор в данных
  - Потерянные ссылки
  - Неверные суммы/количества
- Как оно продолжается
  - Три налоговых извещения, потерянные платежи и т.п.
- «Segmentation fault – core dumped» это плохо, а когда есть потерянные объекты – это ничего? **Это ровно то же самое.**

Пишем форум - 1

```
public class Message {
    String body;
    Author author;
    . . .
}

public class Messages{
    private final Collection<Message> messages = new ArrayList/ConcurrentLinkedQueue();
    public void addMessage(Message msg){
    . . .

}

public class Author{
    String name;
    private final List<Message> posted;

    private int posted_count;
    . . .
}
```

Пишем форум -1

- Работает!
- Работает, но…
- Многопоточность?
- ```
  public class Message {
        . . .
        synchronized getBody()/setBody…
  }
  ```
- Ну хорошо. А класс Messages? Все методы synchronized? В результате все пойдет последовательно. К чему тогда все это было?
  ```

Решение!

- Синхронизированные коллекции!
- Не работает – задача удалить все сообщения от автора «Василий Пупкин» вот так просто не решается – никто не мешает в процессе удаления добавить новое сообщение от проклятого автора.

Решение-2!

- Блокировки
  - Можно shared/exclusive lock
  - Блокируем отдельные сообщения
  - Блокируем отдельные доски.
  - Memory barrier, правда, никто не отменял – либо методы по-прежнему synchronized, либо volatile и т.п.
  - При удалении сообщений сначала ставим shared-блокировку на автора, после чего удаляем сообщения, после чего снимаем блокировку с автора.

```
class Author{
    . . .
    final ReentrantReadWriteLock rwl = new ReentrantReadWriteLock();
```

  - Заработало!!!

Продолжаем расширять функционал

- Бизнес-задача – при удалении сообщения отправить письмо автору с уведомлением или еще чего-нибудь. Реализуем Observer pattern. Теперь при удалении может выполняться произвольный код.
- ```
  class Message{
     public void delete(){
     . . . – много чего делаем
     try{
       author.writeLock.lock();
       getOwnerBoard().getMessages().delete(this);
       author.writeLock.unlock();
       for(MessageObserver mo: getObservers())
          mo.run(this); // get exception here!
       Logger.getLogger("MBoard").info("Message has been deleted");
     }catch(Exception e){
        author.writeLock.unlock();
     }
  }
  ```
- Как откатить изменения, сделанные до исключения? Вести журнал; создать библиотеку, позволяющую сторонним модулям писать в этот журнал изменения (авторы, наверное, будут чрезвычайно рады и сделают все очень хорошо и совсем без ошибок).
- Катастрофа: удалили сообщение, в observers получили исключение, но другой поток успел увидеть неверное количество сообщений от автора.
- Опять не слава богу: дедлоки. Ну, разобрались и с дедлоками. (ThreadMXBean, отдельная нить)
  ```

Протокол блокировок

- Простой подход с блокировками, как описано ранее, не работает.
- Блокировки снимаются только все сразу по окончанию работы бизнес-метода.
-
```
Interface Lockable{
  void lock();
  void unlock()
}
class Author implements Lockable{
. . .
class LockManager{
   final Collection<Lockable> = new ArrayList();
   void lock(Lockable lockable){ . . .
   void unlockAll(){ . . .
}
```

Опять не слава богу - deadlocks

- Нить X получила блокировку на объект A и ждет блокировку на объект B
- Нить Y получила блокировку на объект B и ждет блокировку на объект A.
- Взаимоблокируются две или более нитей.
- Методы разрешения: поиск циклов в графе блокировок, таймауты, определенный порядок доступа (ManagementFactory.getThreadMXBean().findDeadlockedTRhreads())
- В сторону – вообще говоря, возникают нечасто, но неожиданно.

Секундочку!

- А не слишком ли много возни для форума, на котором будут фото котиков?
- Что, собственно, уже сделано:
  - Большая исследовательская работа.
  - Журнал транзакций и библиотека для работы с ним.
  - Менеджер блокировок.
  - Обнаружение дедлоков.
- А дальше? Стоит ли все это таких усилий? Может, бог бы с ним? Пусть теряются платежи и сообщения? Пусть приходят странные письма?
  - Может быть и да: если все это кем-то как-то просматривается и корректируется. Но это не наш случай.

А к чему это все? Это же Java, а не Postgres

- Эти проблемы возникают вне зависимости от платформы
- В СУБД (и Postgres) они давно решены и существуют удобные методы работы с подобного рода задачами – по факту их даже никто обычно не замечает.
- Выходит, это все не надо?
  1. Надо! Хотя бы для реализации СУБД
  2. Как бы я делал без этого презентацию?
  3. Всякое бывает – может и потребоваться.
  4. А может быть, все-таки проще воспользоваться СУБД? И писать удобнее, и отлаживаться, и мониторить работу. Ну и устойчивость к сбоям тоже.

Итоги

- Только что мы изобрели собственную СУБД. Она очень плохая и чрезвычайно ограниченная, но некоторые ключевые вещи там присутствуют:
  - Блокировки
  - Журнал транзакций
    - Любопытно, что эти вещи оказались взаимосвязанными.
- Стоит ли продолжать изобретение велосипеда полувековой давности?