

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет непрерывного и дистанционного обучения

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и системные сети ч.2

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту  
на тему

Импорт данных с удаленного API

БГУИР КП 1-40 01 01 23 ПЗ

Студент: гр. 801021 Михаленя А. Н.

Руководитель: Леванцевич В. А.

Минск 2013



## СОДЕРЖАНИЕ

СОДЕРЖАНИЕ.....	3
1 ПОСТАНОВКА ЗАДАЧИ.....	4
2 API ПОСТАВЩИКА.....	5
2.1 ЧТО ТАКОЕ API.....	5
2.2 ОСОБЕННОСТИ API ПОСТАВЩИКА.....	6
2.3 ПРОБЛЕМЫ ПРИ РАБОТЕ С API.....	7
2.4 ПРОТОКОЛ HTTPS.....	7
3 RATE LIMIT.....	8
3.1 ЧТО ТАКОЕ RATE LIMIT.....	8
3.2 АЛГОРИТМЫ RATE LIMIT.....	8
3.3 МЕТОД ОПРЕДЕЛЕНИЯ СКОРОСТИ.....	9
4 МНОГОПОТОЧНЫЙ ИМПОРТ.....	9
4.1 СИСТЕМА THREAD POOL.....	9
4.2 БЛОКИРОВКА ПОВТОРНОГО ЗАПУСКА.....	10
5 ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА.....	11
5.1 АППАРАТНАЯ СИСТЕМА.....	10
5.2 ОПЕРАЦИОННАЯ СИСТЕМА.....	11
6 ИНСТРУМЕНТЫ РАЗРАБОТКИ.....	12
6.1 JAVA.....	12
6.2 ВСПОМОГАТЕЛЬНЫЕ ПАКЕТЫ.....	13
6.3 MYSQL.....	14
7 РАЗРАБОТКА АЛГОРИТМА И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ.....	14
7.1 ОБЩАЯ СХЕМА РАБОТЫ ПРОГРАММЫ.....	14
7.2 ОПИСАНИЕ АЛГОРИТМА РАБОТЫ ПРОГРАММЫ.....	15
7.3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ.....	16
8 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ.....	18
ЗАКЛЮЧЕНИЕ.....	23
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	24
Приложение А.....	25
Исходный код программных модулей.....	25

# 1 ПОСТАНОВКА ЗАДАЧИ

В рамках данной курсовой работы необходимо разработать CLI(Command Line Interface — интерфейс командной строки) — приложение реализующее многопоточный импорт данных в формате JSON с API поставщика на MySQL сервер. Предполагается, что приложение будет запускаться с помощью демона-планировщика задач — cron. Поэтому необходимо предусмотреть блокировку на повторный запуск программы.

## 2 API ПОСТАВЩИКА

### 2.1 ЧТО ТАКОЕ API

API или интерфейс программирования приложений (англ. application programming interface) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением для использования во внешних программных продуктах. Используется программистами для написания всевозможных приложений. API определяет функциональность, которую предоставляет программа (модуль, библиотека), при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована.

Если программу рассматривать как чёрный ящик, то API — это множество «ручек», которые доступны пользователю данного ящика, и которые он может вертеть и дёргать. Программные компоненты взаимодействуют друг с другом посредством API. При этом обычно компоненты образуют иерархию — высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов. По такому принципу построены протоколы передачи данных по Интернет. Стандартный стек протоколов (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью предыдущего уровня передачи данных и, в свою очередь, предоставляет нужную функциональность следующему уровню.

Важно заметить, что понятие протокола близко по смыслу к понятию API. И то и другое является абстракцией функциональности, только в первом случае речь идёт о передаче данных, а во втором — о взаимодействии приложений. API библиотеки функций и классов включает в себя описание сигнатур и семантики функций.

## 2.2 ОСОБЕННОСТИ API ПОСТАВЩИКА

В данной курсовой работе используется API Яндекс Маркет. Данный API расположен на удалённом сервере, использовать функции API можно с помощью протокола HTTPS. API состоит из 8 основных ресурсов: категории, модели, товарные предложения, отзывы, магазины, регионы, производители, сервисы. При работе с данным API были обнаружены следующие особенности:

a. Запрашивая большой список данных по ресурсу — результат отдаётся постранично. Некоторые данные необходимо хранить у себя в БД, например для работы с древовидными структурами(список категорий). Поэтому придётся реализовать постраничный импорт.

b. Может быть так, что если Rate-limit превышен, API возвращает пустую строку без указания кода ошибки в заголовках. Поэтому, необходимо обязательно реализовать механизм фильтрации запросов.

c. Многие запросы к ресурсам API могут быть регионозависимы - так же необходимо учесть при разработке.

d. Терминология API. В нашем случае, для обозначения некоторых сущностей можно встретить довольно странные названия. Например: гуризованные категории, недогуризованные категории, негуризованные категории, колдунчик (API поиска).

e. Несоответствие данных в разных форматах. Если API умеет возвращать ответ в разных форматах, то может случиться так, что в одном из форматов будет не хватать какой-либо информации. Например в нашем случае некоторая информация есть в формате xml но нет в формате json (характеристика модели — в json отсутствует название группы характеристик). Т.е. необходимо уметь работать со всеми форматами.

f. Иногда для получения некоторого ожидаемого результата необходимо выполнить несколько запросов вместо одного.

## 2.3 ПРОБЛЕМЫ ПРИ РАБОТЕ С API

Вот небольшой список тех проблем, которые необходимо было учитывать при разработке курсового проекта:

а. Невалидные ответы. Например пустая строка, вместо json без указания ошибки в заголовках.

б. Некоторая информация не соответствует действительности. Например количество моделей по категории не совпадает с результатами подбора по параметрам, или количество дочерних категорий не совпадает с их реальным количеством, или количество моделей по производителю не совпадает с реальным и т. д.

в. Размеры картинок прыгали от очень маленьких до очень больших, причём параметр size не всегда работал. Во многих случаях картинки отсутствуют.

И т. д., к сожалению список очень большой.

## 2.4 ПРОТОКОЛ HTTPS

Как уже говорилось выше, API использует протокол HTTPS. HTTPS (HyperText Transfer Protocol Secure) — расширение протокола HTTP, поддерживающее шифрование. Данные, передаваемые по протоколу HTTPS, «упаковываются» в криптографический протокол SSL или TLS, тем самым обеспечивается защита этих данных. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443. Система была разработана компанией Netscape Communications Corporation, чтобы обеспечить аутентификацию и защищённое соединение. HTTPS широко используется в мире Веб для приложений, в которых важна безопасность соединения, например, в платежных системах. HTTPS поддерживается всеми популярными браузерами. HTTPS не является отдельным протоколом. Это обычный HTTP, работающий через шифрованные транспортные механизмы SSL и TLS. Он обеспечивает защиту от атак, основанных на прослушивании сетевого соединения — от снифферских атак и атак типа man-in-the-middle

при условии, что будут использоваться шифрующие средства и сертификат сервера проверен и ему доверяют.

### 3 RATE-LIMIT

#### 3.1 ЧТО ТАКОЕ RATE-LIMIT

В компьютерных сетях термин Rate-Limit используется для обозначения ограничения скорости сетевого трафика. В нашем случае, это означает ограничение количества запросов в секунду к ресурсу.

#### 3.2 АЛГОРИТМЫ RATE-LIMIT

Для того, чтобы API всегда возвращал приемлемый ответ необходимо учитывать условия Rate-limit. Если проект находится на одном сервере, то можно просто подсчитать скорость выполнения запросов. Так же можно использовать алгоритм Token bucket(дырявое ведро). Однако при использовании системы распределения нагрузки, в случае, когда проект расположен в кластере серверов эти варианты не подходят, т. к. запросы будут отфильтрованы только по конкретному узлу. В данном случае можно использовать Проxy сервер. Например Nginx(<http://nginx.org/>). В Nginx можно указать настройки limit\_req на конкретный хост и реализовать кэширование на 1 минуту - проxy\_cache\_min\_uses 1;(см. Рисунок 3.1)

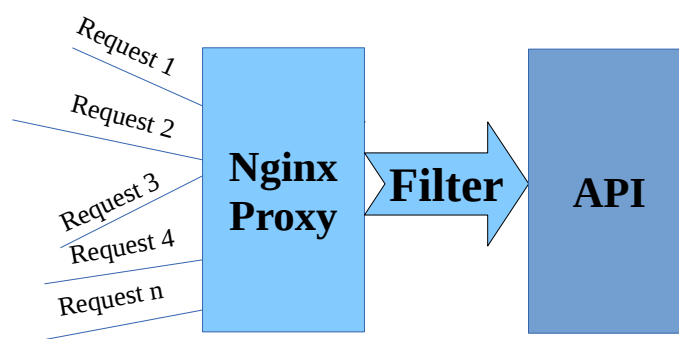


Рисунок 3.1 — Схема использования Proxy



### 3.3 МЕТОД ОПРЕДЕЛЕНИЯ СКОРОСТИ

Поскольку данное приложение будет размещено на 1 сервере - воспользуемся методом подсчёта скорости выполнения запросов в 1 секунду. Реализуем класс, который фиксирует время выполнения в буфере обозначенной длины(зависит от условий Rate-limit), и будет оценивать задержку по времени между нулевым запросом в буфере и вновь прибывшим. К примеру длина буфера равна 8.

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

Таблица 3.1 - Буфер

=>if(v7-v0)>1s → ждём интервал v7-v0

## 4 МНОГОПОТОЧНЫЙ ИМПОРТ

### 4.1 Thread Pool

Итак, мы знаем, что для корректной работы с API необходима реализация алгоритма Rate-limit. Так же мы знаем, что большой список данных API отдаёт постранично. А что, если необходимо анализировать весь этот объём данных на сервере приложения? Посылать и запрашивать каждый раз весь объём данных у API - крайне нерентабельно. Для этого необходимо организовать постраничный импорт данных на сервер БД. Чтобы поддерживать данные в актуальном состоянии — импорт должен выполняться регулярно, обновляя данные на сервере. Для этого необходимо использовать планировщик задач. Но что, если страниц окажется очень много, а данные необходимо обновлять слишком часто? Может случится ситуация, когда импорт ещё не завершил свою работу, а планировщик уже запустил новый импорт - это может продолжаться бесконечно, и привести к нежелательным результатам. Для того, чтобы избежать такой ситуации — импорт должен выполняться многопоточно, чтобы увеличить скорость загрузки данных.

В данной курсовой работе был реализован ThreadPool задач, которые отправляют запрос к API, проверяют валидность ответа, анализируют JSON

и сохраняют данные в БД. В качестве постраничного ресурса используется ресурс **vendor** API поставщика. Ограничение запросов всекунду — 8. Рассмотрим архитектуру реализации ThreadPool более подробно.

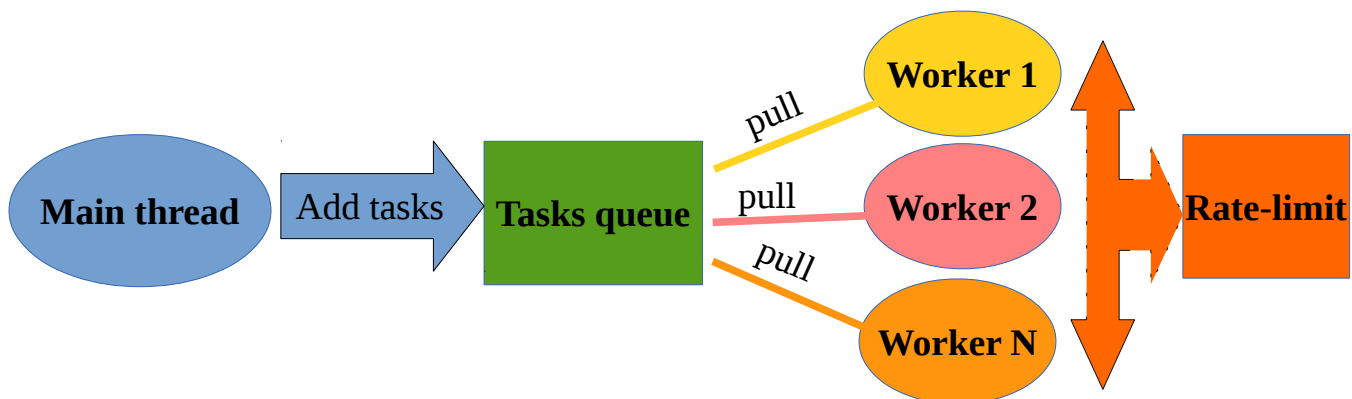


Рисунок 4.1 — Общая архитектура приложения

На рисунке 4.1 изображена общая архитектура многопоточного приложения. Видно, что из главного потока поступают задачи в очередь. Из этой очереди параллельно, так называемые воркеры(исполнители задач), атомарно извлекают и выполняют задачи учитывая алгоритм Rate-limit. После добавления всех задач в очередь, система запускает ежесекундную проверку, на наличие задач в очереди. Как только воркеры извлекли все задачи из очереди, система переходит в состояние ожидания, для того, чтобы дать время на выполнение последних задач. После этого в очередь добавляется Fake — задача, которая завершает работу воркеров и программы в целом.

## 4.2 БЛОКИРОВКА ПОВТОРНОГО ЗАПУСКА

Для того, чтобы планировщик задач не запустил импорт до того, как предыдущий импорт закончится — необходимо предусмотреть блокировку приложения на повторный запуск. Это можно сделать с помощью функции блокировки файла. Используя Java инструменты пробуем открыть fake файл на запись и чтение, если не удалось — создаём и пробуем заблокировать его. Обязательно нужно предусмотреть функцию разблокировки в случае сбоя работы программы. Согласно документации, Java гарантирует выполнения `finally` блока, поэтому необходимости в регистрации `shutdown` функции разблокировки — нет.

## 5 ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА

### 5.1 АППАРАТНАЯ СИСТЕМА

Оптимальные требования к ресурсам для успешной работы программы:  
Для комфортного использования приложения ПК должен соответствовать следующим аппаратным и программным характеристикам:

- Процессоры: Intel Core I5, I7, AMD A6;
- Операционная система: приложение кроссплатформенное;
- ОЗУ минимум 256 Мб, рекомендуется 4 — 6 Гб;
- Жесткий диск - 2 Гб свободного пространства;

Количество ядер в процессоре не обязательно должно быть велико. Т.е. лучше использовать машину с 2,4 ядрами ежели с 8,10. Т.к. важно насколько быстро выполняется операция в 1 потоке, а не насколько задачи распараллелены. Важно также иметь как можно больше оперативной памяти — это намного увеличит производительность(от 6 гб — желательно). Необходимо предусмотреть тот факт, что MySQL хранит данные в `frm` файликах, и поэтому потребуется пространство на жестком диске. Т.к. Данных много, лучше выделить 2 Гб свободного места.

### 5.2 ОПЕРАЦИОННАЯ СИСТЕМА

Данное приложение кроссплатформенное, т. к. и Java и MySQL доступен для разных операционных систем. Однако разработка велась в среде Linux KDE, т. к. она имеет ряд преимуществ:

- ОС бесплатна;
- Софт под данную платформу в основном тоже бесплатный;
- Удобный графический интерфейс;
- Хорошая документация и поддержка;
- Высокий уровень надежности;

- Многоязычная поддержка, что предоставляет возможность для работы во многих странах мира на национальных языках, что достигается применением стандарта ISO Unicode;

- Возможность добавления новых модулей на различные архитектурные уровни операционной системы;

И т. д.

## **6 ИНСТРУМЕНТЫ РАЗРАБОТКИ**

### **6.1 JAVA**

Для разработки данного ПС был выбран язык Java. Java - объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно компилируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине (JVM) вне зависимости от компьютерной архитектуры. Дата официального выпуска — 23 мая 1995 года.

Основные возможности:

- автоматическое управление памятью;
- расширенные возможности обработки исключительных ситуаций;
- богатый набор средств фильтрации ввода/вывода;
- набор стандартных коллекций: массив, список, стек и т. п.;
- наличие простых средств создания сетевых приложений (в том числе с использованием протокола RMI);

- наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы;

- встроенные в язык средства создания многопоточных приложений;

- унифицированный доступ к базам данных:

- на уровне отдельных SQL-запросов — на основе JDBC, SQLJ;

- на уровне концепции объектов, обладающих способностью к хранению в базе данных — на основе Java Data Objects (англ.) и Java Persistence API;

- поддержка обобщений (начиная с версии 1.5);

- параллельное выполнение программ.

Из приложенного списка возможностей можно заметить, что язык очень подходит для решения такого рода задач. Так же следует отметить, что язык кроссплатформенный, что позволит запускать ПС в разных ОС. Java постоянно развивается и обладает мощными инструментами для абстрагирования данных.

## 6.2 ВСПОМОГАТЕЛЬНЫЕ ПАКЕТЫ

Для работы с MySQL используется драйвер JDBC. JDBC (англ. Java DataBase Connectivity — соединение с базами данных на Java) — платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета `java.sql`, входящего в состав Java SE. JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному URL. Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий протокол, за который драйвер отвечает.

Для парсинга JSON используется бесплатный пакет — `json.org`, скачать пакет можно на официальном сайте - <http://json.org/>

Для работы с SQL используется бесплатный вспомогательный пакет — `commons.lang` компании Apache Commons. В данной курсовой работе этот пакет необходим для безопасного экранирования данных, перед выполнением SQL запросов.

### 6.3 MYSQL

В качестве СУБД был выбран MySQL сервер. MySQL — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей, именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Сообществом разработчиков MySQL созданы различные ответвления кода, такие как Drizzle (англ.), OurDelta, Percona Server, и MariaDB. Все эти ответвления уже существовали на момент поглощения компании Sun корпорацией Oracle.

Максимальный размер таблиц в MySQL 3.22 до 4 ГБ, в последующих версиях максимальный размер ограничивается максимальным размером файла используемой операционной системы.

Размер таблицы ограничен её типом. В общем случае тип MyISAM ограничен предельным размером файла в файловой системе операционной системы. Например в NTFS этот размер теоретически может быть до 32 эксабайт. В случае InnoDB одна таблица может храниться в нескольких файлах, представляющих единое табличное пространство. Размер последнего может достигать 64 терабайт.

В отличие от MyISAM в InnoDB имеется значительное ограничение на количество столбцов, которое можно добавить в одну таблицу. Размер страницы памяти по умолчанию составляет 16 килобайт, из которых под данные отведено 8123 байта. Размер указателя на динамические поля составляет 20 байт. Таким образом, в случае использования динамического формата строки (ROW\_FORMAT=DYNAMIC), одна таблица может вместить максимум 409 столбцов типа blob или text.

## **7 РАЗРАБОТКА АЛГОРИТМА И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

### *7.1 ОБЩАЯ СХЕМА РАБОТЫ ПРОГРАММЫ*

На рисунке 3. отображена общая схема работы программы.

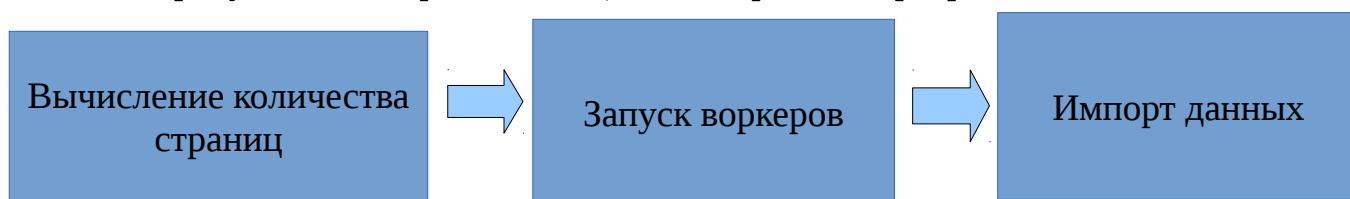


Рисунок 7.1 — Общая схема работы программы

### *7.2 ОПИСАНИЕ АЛГОРИТМА РАБОТЫ ПРОГРАММЫ*

Схема алгоритма представлена в приложении на чертеже Импорт данных с удаленного API схема алгоритма. Опишем пошагово работу алгоритма.

1. Блокировка выполнения повторного импорта
2. Вычисление количества страниц
3. Запуск воркеров
4. Импорт данных
5. Завершение работы

### 7.3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Для работы программного средства были разработаны следующие классы:

- Класс Config (конфигурационные настройки)
- Класс ThreadPool (главный класс, старт выполнения программы, создание воркеров, блокировка, добавление задач в очередь, и реализация импорта)
- Класс TaskQueue (Объявление воркеров и очереди задач)
- Класс Task (выполнение импорта).
- Класс RateLimit (алгоритм Rate-limit).
- Класс JDBC (прослойка между приложением и драйвером JDBC)

Описание некоторых классов приведено в таблицах 7.1-7.4

Таблица 7.1 - Класс Config (конфигурационные настройки)

Свойство	Назначение
yandexKey	Ключ авторизации
yandexURL	Ссылка на API
defaultGeoId	Геопозиция по-умолчанию
dbHost	Хост БД
dbPort	Порт БД
dbUser	Юзер БД
dbPassword	Пароль к БД
dbName	Имя БД
Метод	Назначение
getYandexConfig	Возвращает список настроек для соединения с API
getDbConfig	Возвращает список настроек для соединения с БД



Таблица 7.2 - Класс ThreadPool (главный класс)

Свойство	Назначение
COUNT_ON_PAGE	Количество записей на странице
COUNT_WORKERS	Количество воркеров
PAUSE_BEFORE_FAKE_TASK	Время таймаута до запуска Fake таска
page	Текущая страница
Метод	Назначение
getCountAndSave	Вычислить количество страниц и сохранить первую страницу
logException	Логирование исключений
importData	Запуск импорта
unlock	Разблокировка файла
main	Входная точка

Таблица 7.3 - Класс TaskQueue (Организация очереди задач)

Свойство	Назначение
workers	Воркеры
executors	Потоки в которых исполняются задачи
queue	Очередь
Метод	Назначение
TaskQueue	Конструктор класса
addTask	Добавить задачу
isEmpty	Очередь пустая?

Таблица 7.4 - Task (Выполнение задачи)

Свойство	Назначение
FAKE	Fake таск
id	Id таска
curUrl	Ссылка на текущую страницу
pubDate	Дата публикации
workerId	Id воркера
Метод	Назначение
Task	Конструктор класса
getUrl	Формирует ссылку для выполнения запроса
getContentFromYandexAPI	Посылает запрос к API
saveInfo	Сохранение данных в БД
setWorkerId	Устанавливает id исполнителя
runFakeTask	Запуск Fake таска

## 8 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ

Запуск программы осуществляется командой: `java -jar import.jar`

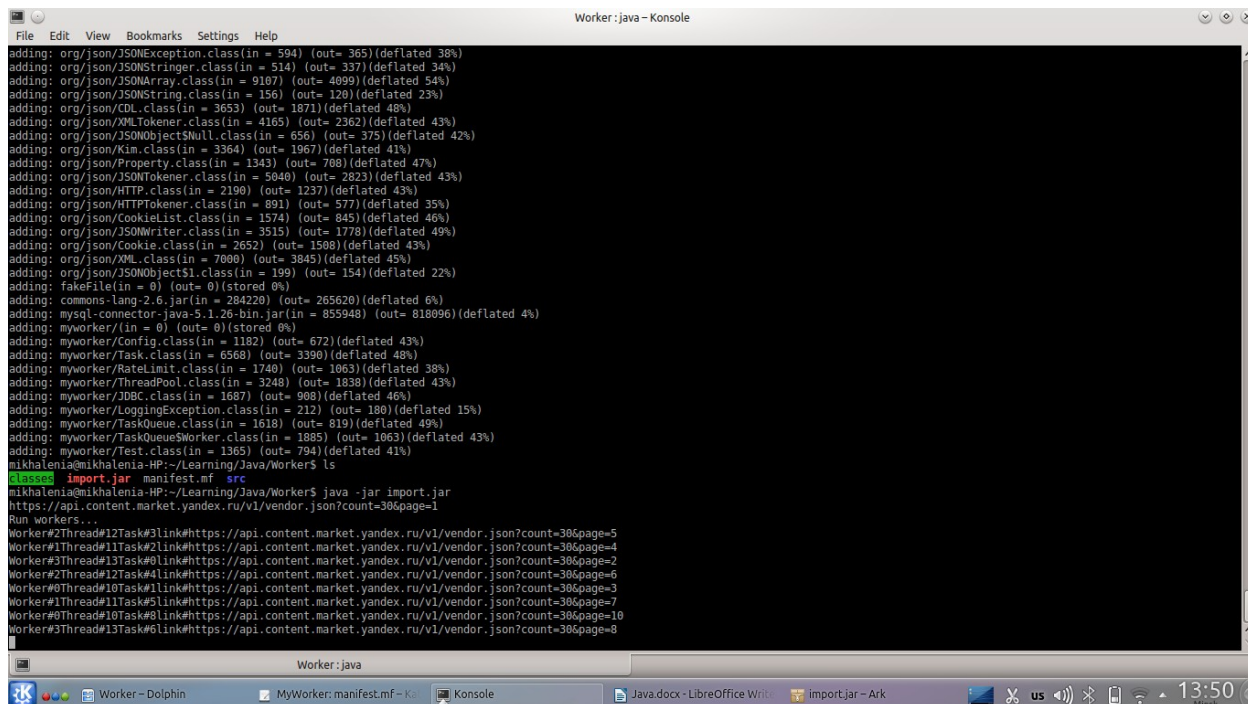


Рисунок 8.1 — Старт программы

На рисунке 8.1 видно, как выполняется первый запрос к API и запускаются воркеры.

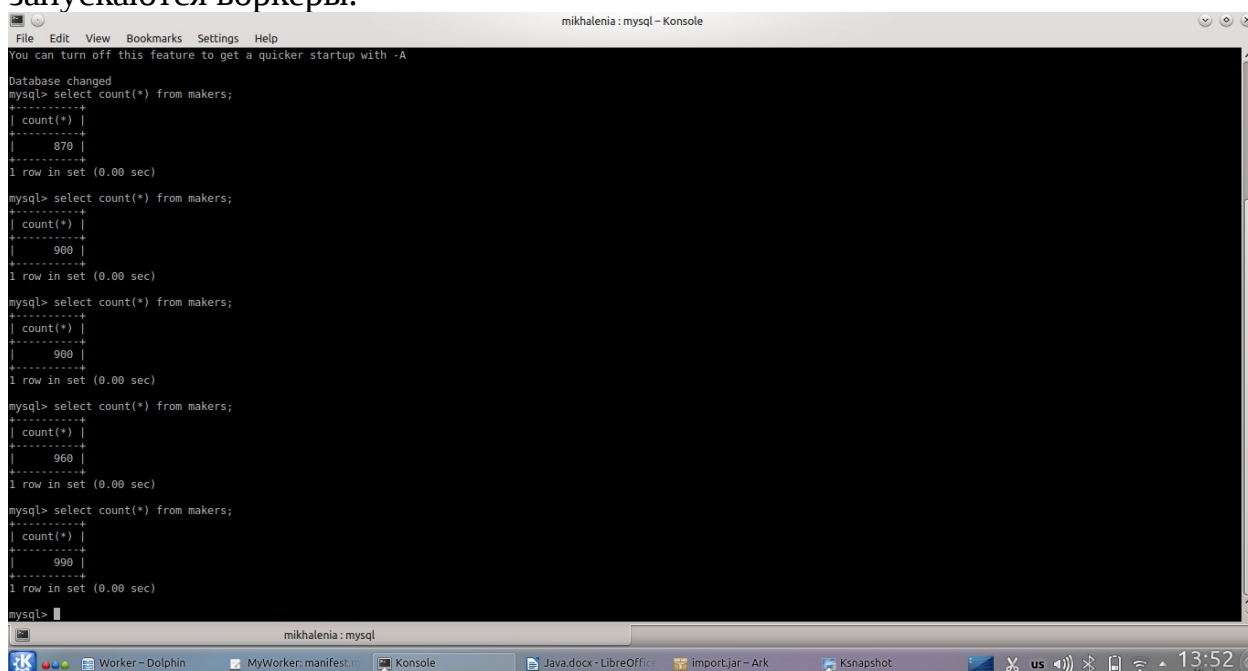
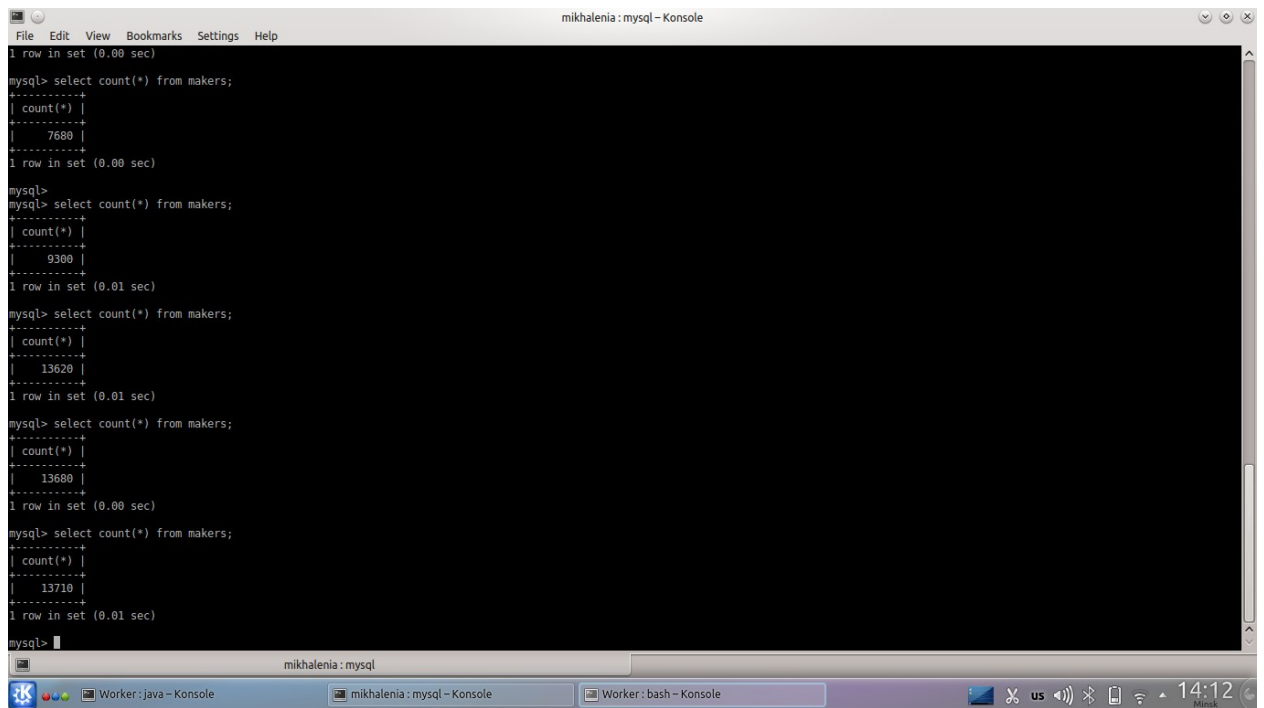


Рисунок 8.2 — Проверка количества записей в таблице



```
mikhalenia : mysql - Konsole
File Edit View Bookmarks Settings Help
1 row in set (0.00 sec)

mysql> select count(*) from makers;
+-----+
| count(*) |
+-----+
|      7680 |
+-----+
1 row in set (0.00 sec)

mysql>
mysql> select count(*) from makers;
+-----+
| count(*) |
+-----+
|      9300 |
+-----+
1 row in set (0.01 sec)

mysql> select count(*) from makers;
+-----+
| count(*) |
+-----+
|     13620 |
+-----+
1 row in set (0.01 sec)

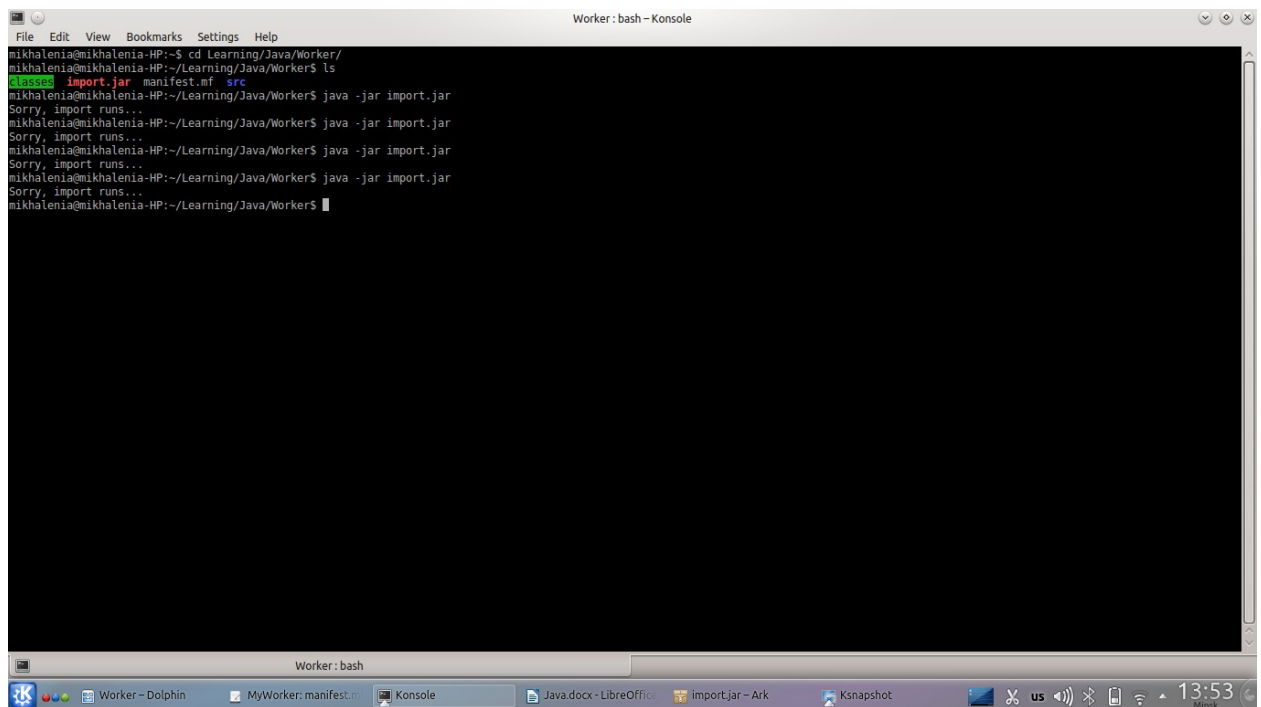
mysql> select count(*) from makers;
+-----+
| count(*) |
+-----+
|     13680 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from makers;
+-----+
| count(*) |
+-----+
|     13710 |
+-----+
1 row in set (0.01 sec)

mysql>
```

Рисунок 8.3 — Проверка количества записей в таблице

На рисунках 8.2 и 8.3 изображено несколько запросов к таблице, следим за она пополняется.



```
Worker: bash - Konsole
File Edit View Bookmarks Settings Help
mikhailenia@mikhailenia-HP:~$ cd Learning/Java/Worker/
mikhailenia@mikhailenia-HP:~/Learning/Java/Worker$ ls
import.jar manifest.mf src
mikhailenia@mikhailenia-HP:~/Learning/Java/Worker$ java -jar import.jar
Sorry, import runs...
mikhailenia@mikhailenia-HP:~/Learning/Java/Worker$ java -jar import.jar
Sorry, import runs...
mikhailenia@mikhailenia-HP:~/Learning/Java/Worker$ java -jar import.jar
Sorry, import runs...
mikhailenia@mikhailenia-HP:~/Learning/Java/Worker$ java -jar import.jar
Sorry, import runs...
mikhailenia@mikhailenia-HP:~/Learning/Java/Worker$
```

Рисунок 8.4 — Попытка запустить повторный импорт

На рисунке 8.4 изображена попытка повторного запуска приложения не дожидаясь завершения первого.

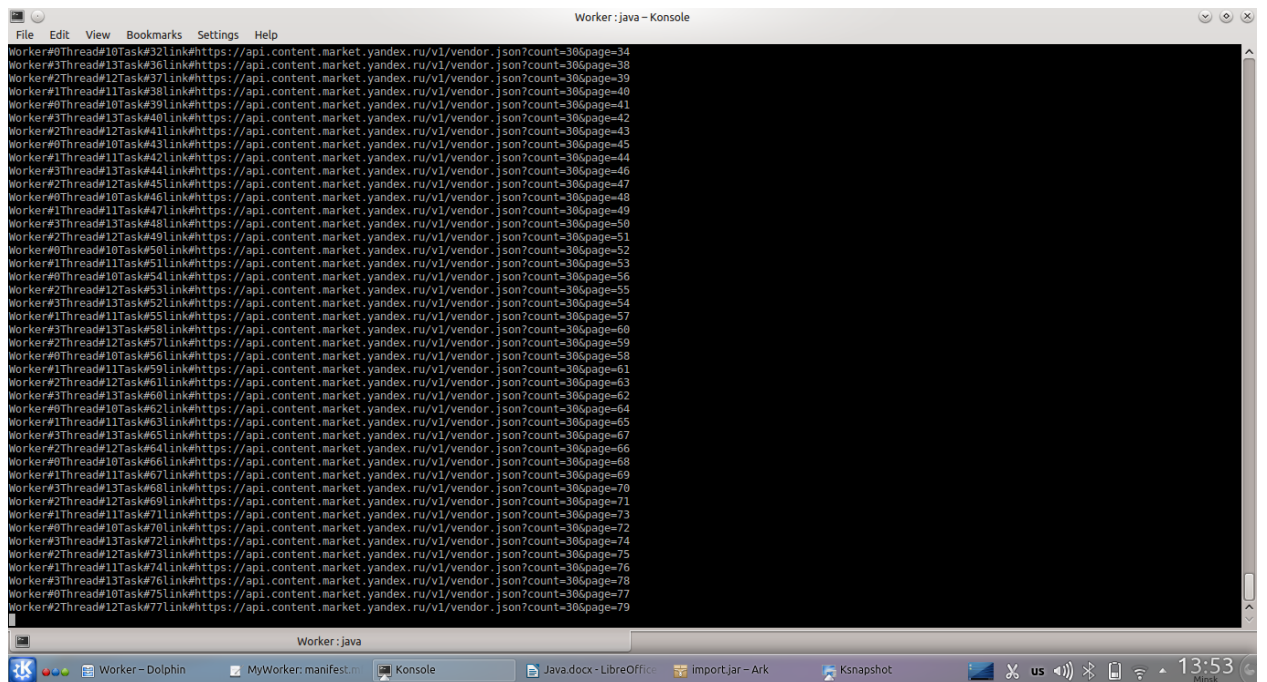


Рисунок 8.5 — Процесс импорта

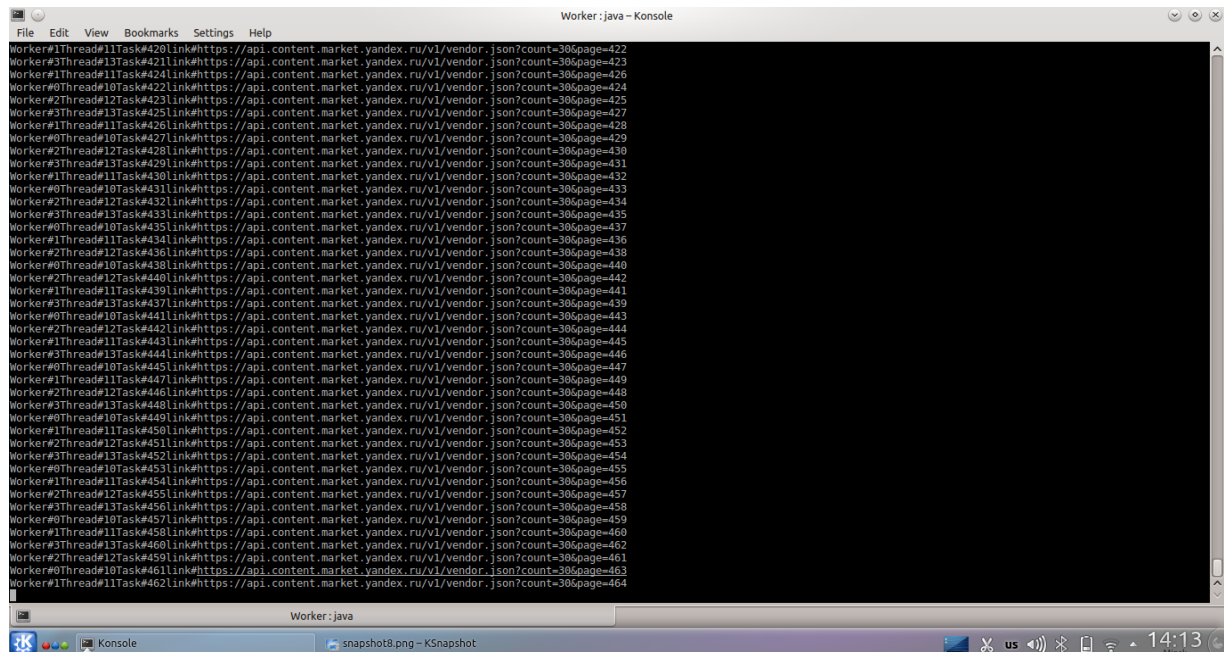


Рисунок 8.6 — Процесс импорта

Нарисунках 8.5 и 8.6 изображены запросы к API с указанием информации о том, какой воркер выполняет задачу, какой это таск и поток.



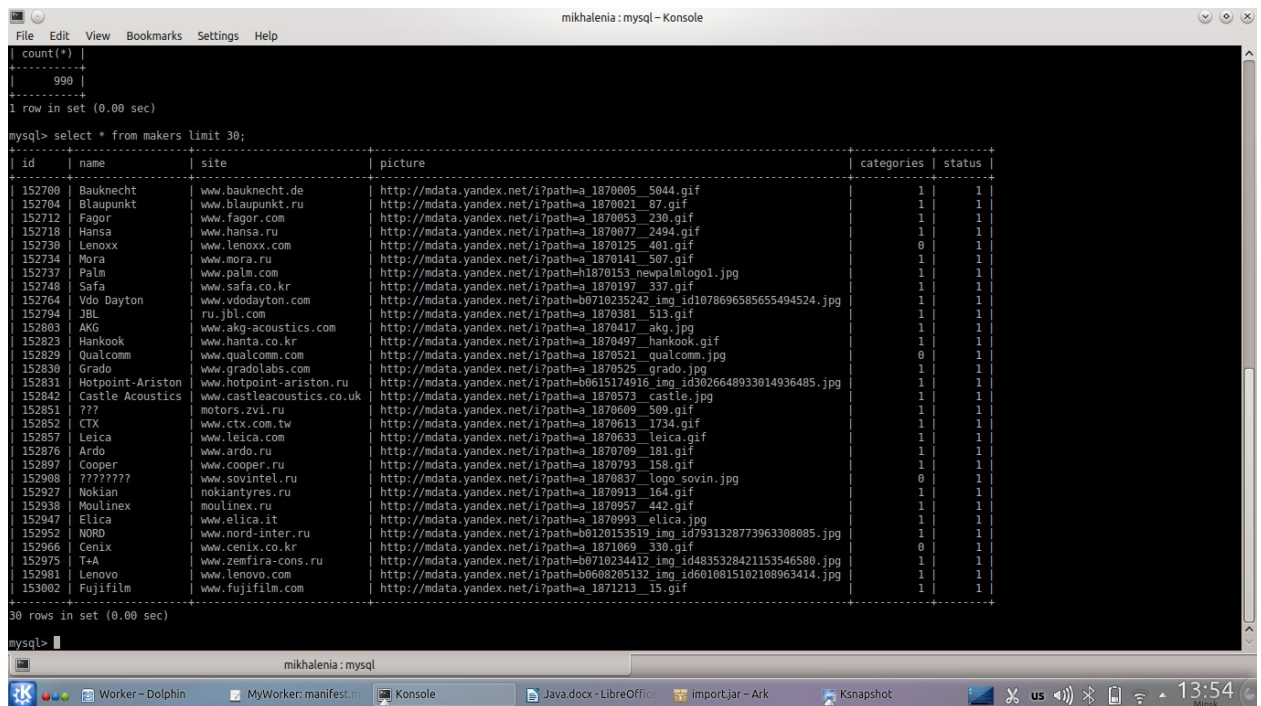


Рисунок 8.7 — Вывод данных на экран

На рисунке 8.7 видно, что записи пишутся в таблицу со статусом 1. После завершения импорта статус сменится и старые данные будут удалены. См. Рисунок 8.8

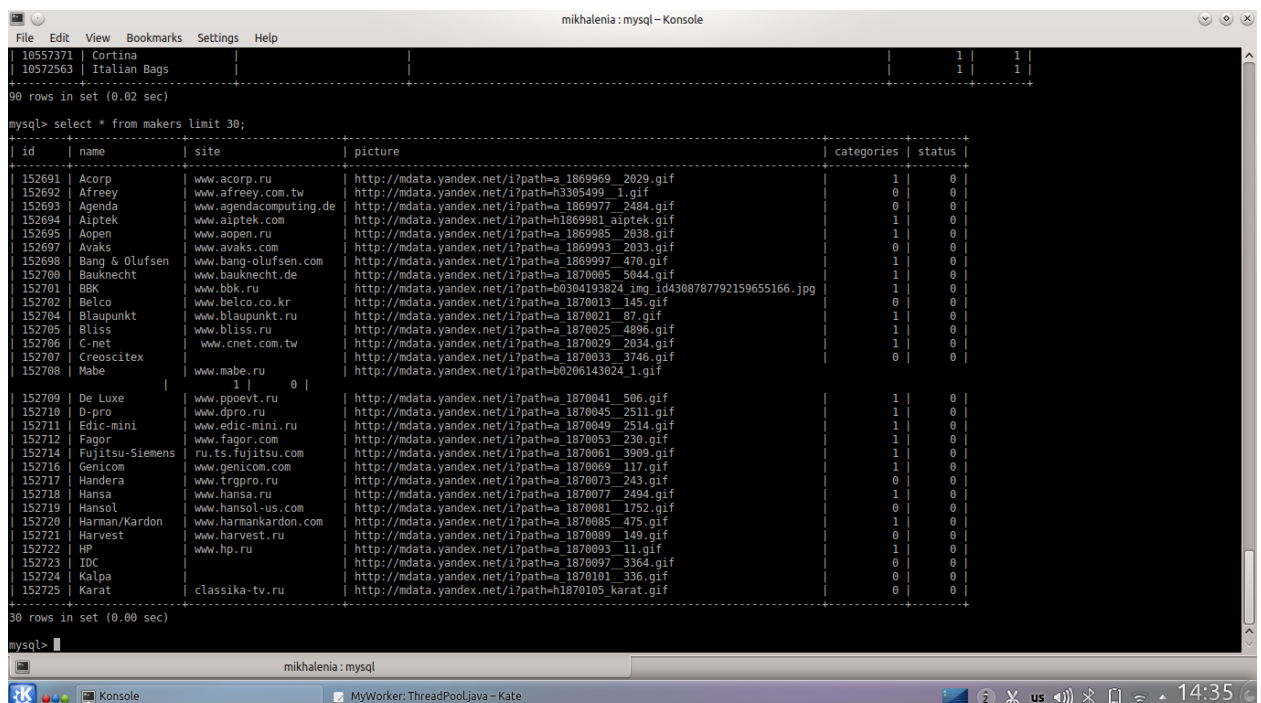


Рисунок 8.8 — Изменение статуса

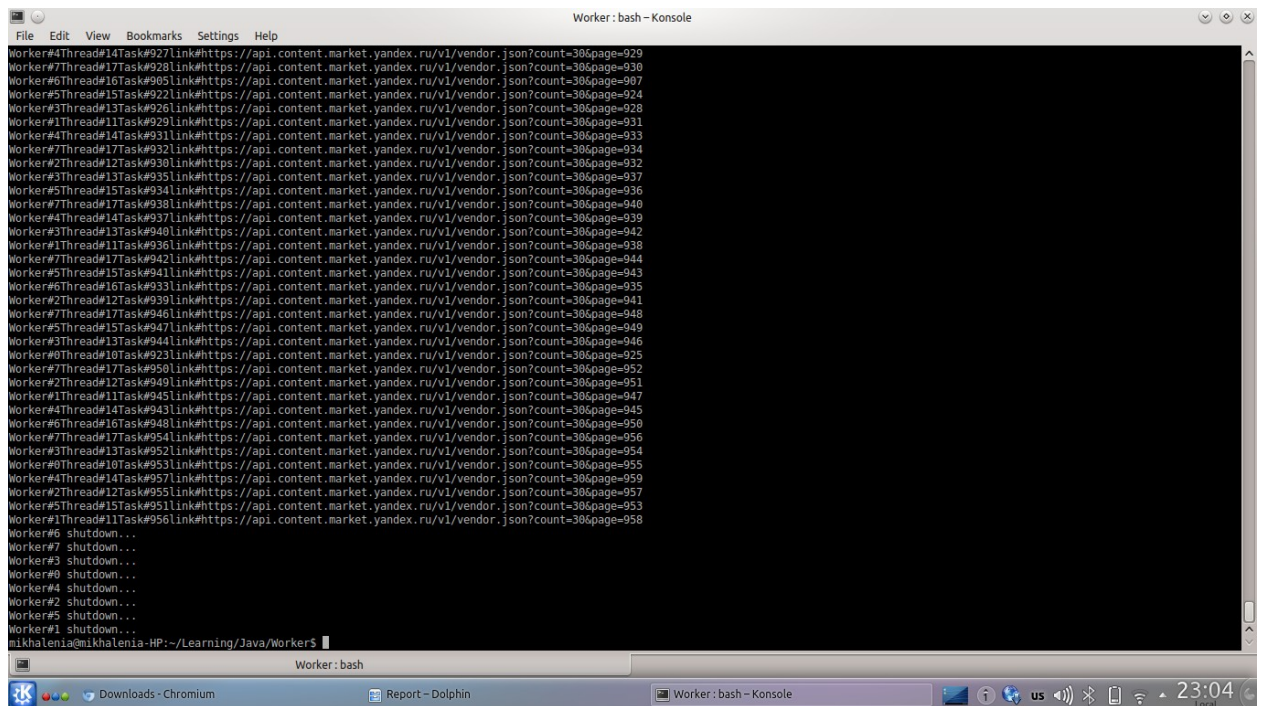


Рисунок 8.9 — Завершение работы

На рисунке 8.9 отображено завершение работы воркеров и программы.

## ЗАКЛЮЧЕНИЕ

В результате написания курсовой работы была разработана программное средство многопоточно импорта данных на MySQL сервер. Были рассмотрены алгоритмы Rate-limie, архитектура многопоточного приложения, и множество вспомогательных инструментов. Данное приложение может послужить началом для реализации крупного веб-прприложения.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. <http://ru.wikipedia.org>
2. <http://docs.oracle.com>
3. Философия Java. Брюс Эккель - Библиотека программиста
4. <http://www.mysql.com/>



## Приложение А

### *Исходный код программных модулей*

#### ThreadPool.java

```
package myworker;
import java.util.*;
import java.util.logging.*;
import java.io.*;
import java.nio.channels.*;
import java.util.concurrent.*;
import org.json.JSONArray;
import org.json.JSONObject;
import static myworker.Task.*;
import static java.util.concurrent.TimeUnit.*;

public class ThreadPool
{
    private static final int COUNT_ON_PAGE=30;
    private static final int COUNT_WORKERS=8;
    private static final int PAUSE_BEFORE_FAKE_TASK=RateLimit.ONESECOND*20;
    private static int page=0;
    private static int getCountAndSave()
    {
        HashMap<String,String> pathParams=new HashMap<String,String>();
        HashMap<String,String> getParams=new HashMap<String,String>();
        pathParams.put("path","vendor");
        pathParams.put("format","json");
        getParams.put("page",Integer.toString(++page));
        getParams.put("count",Integer.toString(COUNT_ON_PAGE));
        String link=getUrl(pathParams,getParams);
        System.out.println(link);
        String content=getContentFromYandexAPI(link);
        saveInfo(content);
        JSONObject jsonObject=new JSONObject(content);
        return (int)
(jsonObject.getJSONObject("vendorList").getInt("total")/COUNT_ON_PAGE);
    }
    private static void logException(Exception e)
    {
        Logger logger=Logger.getLogger("ThreadPool");
        StringWriter trace=new StringWriter();
        e.printStackTrace(new PrintWriter(trace));
        logger.severe(trace.toString());
        System.exit(0);
    }
    private static void importData()
    {
        int countPages=0;
        try{countPages=getCountAndSave();}catch(Exception e){logException(e);}
        TaskQueue tq=new TaskQueue(COUNT_WORKERS);
```

```

        for(int i=page+1;i<countPages;i++)
            try{tq.addTask(new Task(i));}catch(Exception e){logException(e);}
        while(!tq.isEmpty())
            try{TimeUnit.MILLISECONDS.sleep(RateLimit.ONESECOND);}catch(
Exception e){logException(e);}
            try{TimeUnit.MILLISECONDS.sleep(PAUSE_BEFORE_FAKE_TASK);}catch(
Exception e){logException(e);}
            tq.addTask(new Task(Task.FAKE));
        }
    private static void unlock(FileLock lock)
    {
        try
        {
            if(lock!=null)
                lock.release();
        }
        catch(Exception e)
        {
            logException(e);
        }
    }
    public static void main(String[] args)
    {
        FileLock lock=null;
        try
        {
            FileChannel channel=new RandomAccessFile(new
File("./classes/fakeFile"),"rw").getChannel();
            lock=channel.tryLock();
            if(lock==null)
            {
                System.out.println("Sorry, import runs...");
                System.exit(0);
            }
            importData();
        }
        catch(Exception e)
        {
            logException(e);
        }
        finally
        {
            unlock(lock);
        }
    }
}

```

#### TaskQueue.java

```

package myworker;
import java.util.*;
import java.util.concurrent.*;

```

```

public class TaskQueue
{
    private Worker[] workers;
    private ExecutorService executors;
    private ConcurrentLinkedQueue<Task> queue;
    public TaskQueue(int countWorkers)
    {
        queue=new ConcurrentLinkedQueue<Task>();
        workers=new Worker[countWorkers];
        executors=Executors.newFixedThreadPool(countWorkers);
        for(int i=0;i<countWorkers;i++)
        {
            workers[i]=new Worker(i);
            executors.execute(workers[i]);
        }
        System.out.println("Run workers...");
    }
    public void addTask(Task job)
    {
        if(job!=null)
            queue.add(job);
    }
    public class Worker implements Runnable
    {
        private int id;
        private Long timeLastTask;
        public Worker(int id)
        {
            this.id=id;
        }
        private boolean isShutDown()
        {
            long tm=System.currentTimeMillis();
            if(timeLastTask==null)
                timeLastTask=tm;
            long delta=tm-timeLastTask;
            if(delta>=RateLimit.ONESECOND*5)
                return true;
            return false;
        }
        public void run()
        {
            while(!Thread.interrupted())
            {
                if(queue.peek()==null)
                {
                    if(isShutDown())
                        Thread.currentThread().interrupt();
                    continue;
                }
                Task oTask=queue.poll();
            }
        }
    }
}

```

```

        oTask.setWorkerId(id);
        oTask.run();
        oTask=null;
        timeLastTask=System.currentTimeMillis();
    }
    System.out.println("Worker#" + id + " shutdown...");
    executors.shutdown();
}
}
public boolean isEmpty()
{
    return queue.isEmpty();
}
}
}

```

### Task.java

```

package myworker;
import java.util.concurrent.*;
import java.util.*;
import java.net.*;
import java.io.*;
import java.sql.*;
import org.json.JSONArray;
import org.json.JSONObject;
import org.apache.commons.lang.StringEscapeUtils;
import static java.util.concurrent.TimeUnit.*;
import static myworker.Config.*;
import static myworker.JDBC.*;

public class Task implements Runnable
{
    public static final boolean FAKE=true;
    private static int count=0;
    private final int id=count++;
    private boolean isFakeTask=false;
    private String curUrl;
    private int workerId;
    private int page=0;
    private static final int COUNT_ON_PAGE=30;
    private static HashMap<String,String> yandexConfig=getYandexConfig();
    public Task(int page)
    {
        this.page=page;
    }
    public Task(boolean isFakeTask)
    {
        this.isFakeTask=isFakeTask;
    }
    public static String getUrl(HashMap<String,String>
pathParams,HashMap<String,String> getParams)
    {

```

```

        if(pathParams.isEmpty())
            return null;
        String link=yandexConfig.get("url")+"/"+yandexConfig.get("version")+"/";
        if(pathParams.containsKey("path"))
            link+=pathParams.get("path");
        if(pathParams.containsKey("format"))
            link+="."+pathParams.get("format");
        if(getParams.isEmpty())
            return link;
        link+="?";
        String[] get=new String[getParams.size()];
        int i=0;
        for(String key:getParams.keySet())
            get[i++]=key+"="+getParams.get(key);
        link+=implodeArray(get,"&");
        return link;
    }
    public static String getUrl(HashMap<String,String> pathParams)
    {
        return getUrl(pathParams,null);
    }
    public static String getContentFromYandexAPI(String link)
    {
        if(link=="")
            return null;
        try
        {
            URL url= new URL(link);
            URLConnection connect=url.openConnection();
            connect.setRequestProperty("Authorization",yandexConfig.get("key"));
            BufferedReader in=new BufferedReader(new
InputStreamReader(connect.getInputStream()));
            String inputLine,result="";
            while((inputLine=in.readLine())!=null)
                result+=inputLine;
            in.close();
            return result;
        }
        catch(IOException e)
        {
            System.out.println("Sorry, cant load content...");
            e.printStackTrace(System.out);
            return null;
        }
    }
    public static String implodeArray(String[] inputArray,String glueString)
    {
        String output="";
        if(inputArray.length==0)
            return null;
        StringBuilder sb = new StringBuilder();
        sb.append(inputArray[0]);

```

```

        for (int i=1; i<inputArray.length; i++)
        {
            sb.append(glueString);
            sb.append(inputArray[i]);
        }
        output=sb.toString();
        return output;
    }
    public static void saveInfo(String content)
    {
        JSONObject jsonObject=new JSONObject(content);
        JSONObject innerObject=jsonObject.getJSONObject("vendorList");
        JSONArray jsonArray=innerObject.getJSONArray("vendor");
        int size=jsonArray.length();
        String[] makerValues=new String[size];
        for(int i=0;i<size;i++)
        {
            JSONObject objectInArray=jsonArray.getJSONObject(i);
            String
categories=objectInArray.getJSONArray("categories").length()==0?"0":"1";
            String site=objectInArray.has("site"?objectInArray.getString("site"):"";
            String picture=objectInArray.has("picture"?
objectInArray.getString("picture"):"";
            makerValues[i]="("+
                Integer.toString(objectInArray.getInt("id"))+","+
                """+StringEscapeUtils.escapeSql(objectInArray.getString("name"))
+"""+
                """+StringEscapeUtils.escapeSql(site)+"""+
                """+StringEscapeUtils.escapeSql(picture)+"""+
                """+StringEscapeUtils.escapeSql(categories)+"""+
                "1"+
            ")";
        }
        try
        {
            Connection dbConnect=getDBConnection();
            dbConnect.prepareStatement(
                "INSERT DELAYED INTO
makers(id,name,site,picture,categories,status)" +
                "VALUES "+implodeArray(makerValues,",")+
                "ON DUPLICATE KEY UPDATE "+
                "picture=VALUES(picture)," +
                "categories=VALUES(categories)," +
                "status=VALUES(status)"
            ).execute();
        }
        catch(SQLException e)
        {
            System.out.println("Sorry, can't save the content...");
            e.printStackTrace(System.out);
        }
    }
}

```

```

public void setWorkerId(int workerId)
{
    this.workerId=workerId;
}
private void runFakeTask()
{
    try
    {
        Connection dbConnect=getDBConnection();
        dbConnect.prepareStatement("UPDATE makers SET
status=1-status").execute();
    }
    catch(SQLException e)
    {
        System.out.println("Sorry, can't change status...");
        e.printStackTrace(System.out);
    }
    Thread.currentThread().interrupt();
}
public void run()
{
    if(isFakeTask)
    {
        runFakeTask();
        return;
    }
    RateLimit.check();
    HashMap<String,String> pathParams=new HashMap<String,String>();
    HashMap<String,String> getParams=new HashMap<String,String>();
    pathParams.put("path","vendor");
    pathParams.put("format","json");
    getParams.put("page",Integer.toString(page));
    getParams.put("count",Integer.toString(COUNT_ON_PAGE));
    this.curUrl=getUrl(pathParams,getParams);
    String content=getContentFromYandexAPI(this.curUrl);
    saveInfo(content);
    System.out.println(this);
}
public String toString()
{
    return "Worker#" +this.workerId+"Thread#" +Thread.currentThread().getId()
+"Task#" +id+"link#" +this.curUrl;
}
}

```

### RateLimit.java

```

package myworker;
import java.util.concurrent.*;
import java.util.*;
import static java.util.concurrent.TimeUnit.*;

```

```

public class RateLimit
{
    private static final int RATELIMIT=8;
    public static final int ONESECOND=1000;
    private static Long[] timeBuffer=new Long[RATELIMIT];
    private static Boolean isInit=false;
    private static synchronized long getDelay()
    {
        if(!isInit)
        {
            isInit=initBuffer();
            return 0;
        }
        long tm=System.currentTimeMillis();
        long delta=tm-timeBuffer[0];
        offSet(tm);
        if(delta>=ONESECOND)
            return 0;
        return ONESECOND-delta;
    }
    private static Boolean initBuffer()
    {
        for(int i=0;i<timeBuffer.length;i++)
        {
            if(timeBuffer[i]!=null)
                continue;
            timeBuffer[i]=System.currentTimeMillis();
            if(i==timeBuffer.length-1)
                return true;
            return false;
        }
        return true;
    }
    private static void offSet(long tm)
    {
        for(int i=0;i<timeBuffer.length;i++)
            if(i==timeBuffer.length-1)
                timeBuffer[i]=tm;
            else
                timeBuffer[i]=timeBuffer[i+1];
    }
    private static void pause(long delay)
    {
        try
        {
            TimeUnit.MILLISECONDS.sleep(delay);
        }
        catch(InterruptedException e)
        {
            e.printStackTrace(System.out);
        }
    }
}

```



```

        public static void check()
        {
            long delay=getDelay();
            if(delay!=0)
                pause(delay);
        }
    }
}

```

#### JDBC.java

```

package myworker;
import java.util.*;
import java.sql.*;
import static myworker.Config.*;

public class JDBC
{
    private static HashMap<String,String> db=getDBConfig();
    public static Connection getDBConnection()
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        }
        catch(Exception e)
        {
            e.printStackTrace(System.out);
            System.exit(0);
        }
        try
        {
            return DriverManager.getConnection("jdbc:mysql://"+db.get("host")+
+":"+db.get("port")+ "/" +db.get("name")+"?user="+db.get("user")+
+"&password="+db.get("password"));
        }
        catch(SQLException e)
        {
            e.printStackTrace(System.out);
            System.exit(0);
        }
        System.exit(0);
        return null;
    }
}

```

#### Config.java

```

package myworker;
import java.util.*;

public class Config
{

```

```

private static final String yandexKey="yandexKey";
private static final String yandexURL="https://api.content.market.yandex.ru";
private static final String yandexVersion="v1";
private static final String defaultGeoId="157";
private static final String dbHost="localhost";
private static final String dbPort="3306";
private static final String dbUser="root";
private static final String dbPassword="root";
private static final String dbName="myworker";
public static HashMap<String,String> getYandexConfig()
{
    HashMap<String,String> yandexConfig=new HashMap<String,String>();
    yandexConfig.put("key",yandexKey);
    yandexConfig.put("url",yandexURL);
    yandexConfig.put("version",yandexVersion);
    yandexConfig.put("defaultGeoId",defaultGeoId);
    return yandexConfig;
}
public static HashMap<String,String> getDBConfig()
{
    HashMap<String,String> dbConfig=new HashMap<String,String>();
    dbConfig.put("host",dbHost);
    dbConfig.put("port",dbPort);
    dbConfig.put("user",dbUser);
    dbConfig.put("password",dbPassword);
    dbConfig.put("name",dbName);
    return dbConfig;
}
}

```

#### makers.sql

```

CREATE TABLE IF NOT EXISTS makers(
  id int(11) unsigned NOT NULL AUTO_INCREMENT,
  name varchar(255) NOT NULL DEFAULT "",
  site varchar(255) NOT NULL DEFAULT "",
  picture varchar(255) NOT NULL DEFAULT "",
  categories int(1) unsigned NOT NULL DEFAULT '0',
  status int(2) unsigned NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  KEY cs(categories,status)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=0;

```