DOI 10.37882/2223–2966.2023.03.26

РАЗРАБОТКА И ПРИМЕНЕНИЕ В ГРАФИЧЕСКОМ РЕДАКТОРЕ АЛГОРИТМОВ МНОГОСЛОЙНОЙ ОБРАБОТКИ ИЗОБРАЖЕНИЙ

DEVELOPMENT AND APPLICATION OF ALGORITHMS FOR MULTILAYER IMAGE PROCESSING IN A GRAPHICAL EDITOR

V. Monastyrev
S. Molodyakov

Summary. In this paper, we consider an algorithm for multilayer image processing, which allows you to combine computing resources on the local device and the server side. The paper considers existing solutions, identifies their advantages and disadvantages. An algorithm for multilayer image processing is described, and an example of its implementation is given.

Keywords: image processing, multilayer processing algorithm, image editor, machine learning.

Монастырев Виталий Викторович
Аспирант, Санкт-Петербургский политехнический университет Петра Великого
vit34–95@mail.ru
Молодяков Сергей Александрович
Д.т.н., профессор, Санкт-Петербургский политехнический университет Петра Великого
molodyakov_sa@spbstu.ru

Аннотация. В данной работе рассматривается алгоритм многослойной обработки изображений, который позволяет совмещать вычислительные ресурсы на локальном устройстве и серверной части. В работе рассматриваются существующие решения, выявляются их преимущества и недостатки. Описан алгоритм многослойной обработки изображений, а также приводится пример его реализации.

Ключевые слова: обработка изображений, алгоритм многослойной обработки, редактор изображений, машинное обучение.

Введение

Область обработки изображений в последние годы активно развивается в связи с популярностью фотографий и необходимостью их редактирования. Развитие во многом стало возможно благодаря увеличению мощности устройств, которые используются при редактировании. Это смартфоны, планшеты и компьютеры. Отдельным вектором развития стали технологии машинного обучения. Благодаря таким технологиям стали возможны новые способы обработки изображений, например применение GAN-алгоритмов [1], замена и удаление объектов с фото с восстановлением фона и другое. Машинное обучение на сегодняшний день доступно как на мобильных платформах, так и на стационарных, в виде различных библиотек и фреймворков. На платформах компании Apple основой для машинного обучения является библиотека Core ML [2]. На других платформах широкое развитие получили PyTorch [3], TensorFlow [4], Keras [5] и другие. Известны методы модульно-конвейерной обработки данных [6] .

Отдельной задачей является комбинация различных алгоритмов совместно. Здесь есть несколько сложностей. Во-первых, это платформа, на которой будет работать редактор изображений. В зависимости от платформы у разработчиков есть определенный стек технологий, который можно использовать. Вторая проблема — это производительность и технические возможности устройств, на которых будет работать редактор. Некоторые устройства могут поддерживать библиотеки и производить ресурсоемкие вычисления, а другие нет. Таким образом, ставится задача о том, как можно производить параллельную обработку изображений, чтобы часть вычислений производилась на устройстве, а часть на сервере.

Анализ существующих решений

На данный момент представлено достаточно большое количество различных редакторов изображений в магазинах приложений. Наиболее известные примеры — это Photoshop для редактирования изображений на компьютере. Photoshop позволяет использовать многослойное редактирование изображений, создавать коллажи и прочее. Если рассматривать мобильные приложения, то можно выделить Pixelmator. Данное приложение также обладает большим функционалом для редактирования изображений, использует принципы многослойности. Однако стоит отметить, что как правило в данных приложениях обработка идет на устройствах пользователя. В данной статье будет предложен алгоритм, который позволит вести обработку частично на устройстве пользователя, а частично на удаленном сервере, причем обработка для пользователя будет выглядеть как единый процесс. Данное разделение позволит использовать ресур-

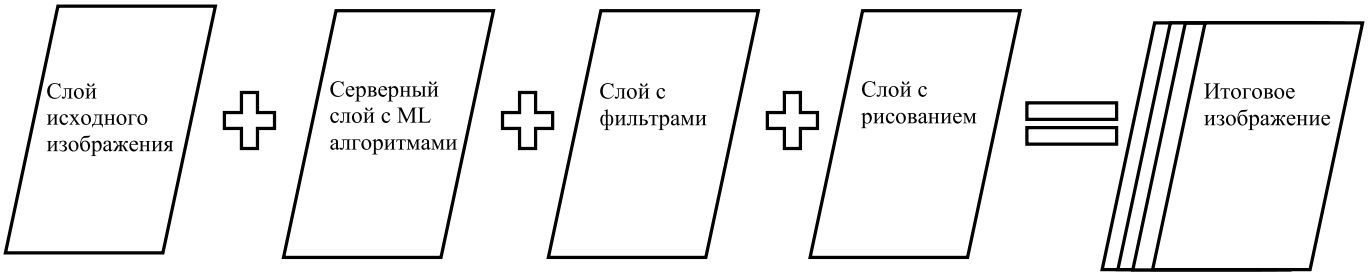

Рис. 1. Многослойный алгоритм обработки

соемкие алгоритмы даже на достаточно старых и мобильных устройствах.

В большинстве известных публикаций рассматриваются отдельные алгоритмы обработки изображений. В одной из статей [7] авторы предлагают архитектуру для удаленной обработки данных, позволяющую уже существующим локальным приложениям получать доступ к удаленным интеллектуальным средам.

В своей архитектуре авторы предлагают использовать 3 уровня для клиента и сервера:

- L1 уровень — представляет приложение, используемое как на стороне клиента, так и на стороне сервера.
- L2 уровень — управляет потоком связи между клиентом и сервером. Он адаптирует данные, сгенерированные приложением на клиенте, в подходящем формате для адекватной обработки прикладным уровнем на стороне сервера.
- L3 уровень — создает каналы связи между клиентом и сервером для передачи данных.

Предлагаемую архитектуру авторы применяли для анализа медицинских изображений. В редакторах изображений же обычно перемешены функции обработки, такие как рисование, фильтрация, изменение размеров и другое. В данной статье будет использоваться предложенная архитектура и предложены улучшения алгоритма в рамках использования в приложении-редакторе.

Алгоритм многослойной обработки изображений

В одной из работ [7] авторы рассматривают полностью удаленную обработку изображений на стороне сервера. Данное решение применялось авторами для обработки медицинских изображений: постановка диагнозов при помощи анализа Machine Learning (ML) алгоритмами и прочее. Если же расширить предложенную архитектуру на пользовательскую обработку изображений, то возникает проблема в том, что нет необходимости запускать все алгоритмы на стороне сервера. На сегодняшний день простые ML-модели можно запускать даже на мобильном устройстве, а на сервере производить обработку только для высокопроизводительных моделей. В качестве высокопроизводительных моделей в данной статье, мы будем рассматривать алгоритмы:

- LaMa [8] — данный алгоритм позволяет выделять на фото объекты, а затем удалять их с восстановлением фона;
- Sky Replacing [9] — данный алгоритм позволяет определять границы неба на фотоизображении, а затем заменять его другим изображением;
- Fast Neural Style [10] — данный алгоритм позволяет применять к фотоизображению стиль с другого изображения.

В качестве простых алгоритмов были выбраны алгоритмы рисования и Look Up Table (LUT) фильтры, так как это одни из самых часто встречающихся инструментов в редакторе изображений. Таким образом, ставится задача по модернизации предложенного алгоритма для возможности многослойной обработки, где часть слоев вычисляется на мобильном устройстве, а часть слоев на сервере.

Для возможности работы описанных выше алгоритмов машинного обучения с фильтрами и инструментами рисования, была предложена многослойная архитектура для клиентского приложения. Каждый слой — это свой собственный набор алгоритмов. Приведем схематичное изображение данной архитектуры на рис. 2. Стоит отдельно отметить серверный слой, который будет вести обработку на удаленном от пользовательского устройства сервере.

Рассмотрим более подробно каждый из слоев предложенного алгоритма.

Слой исходного изображения представляет собой непосредственно изображение, которое было загружено пользователем для обработки. Данный слой должен находиться на стороне клиента и требуется для преобразования изображения в нужный формат.

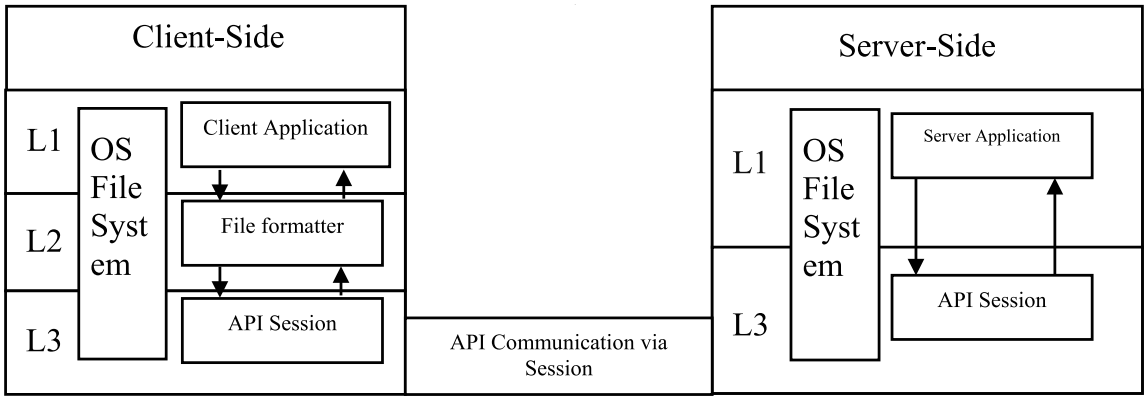

Рис. 2. Клиент-серверная архитектура для удаленной обработки данных

Далее следует слой с ML-алгоритмами. К сожалению, не все ML-алгоритмы можно перенести на сторону клиента. Это связано с различными версиями библиотек и совместимостью устройств. Таким образом, было предложено решение, которое взаимодействует с серверной частью. Стоит отметить, что предложенное решение позволяет подключать неограниченное число алгоритмов машинного обучения, т.е. в дальнейшем возможно расширение функционала через простое добавление нового API.

Далее следует слой с фильтрами. Он позволяет накладывать на исходное изображение различные типы фильтров.

Наконец, последний слой рисования, позволяет пользователям рисовать поверх исходного изображения. Это могут быть как другие изображения, так и различные кривые с эффектами ручки, маркера или неона.

Стоит отметить, что все слои в предложенном алгоритме многослойной обработки являются опциональными. Т.е. они либо могут не использоваться, либо использоваться полностью, либо частично.

Теперь, когда описаны каждый из слоев многослойной обработки изображения, появляется задача в модернизации предложенного выше алгоритма для удаленной обработки изображений.

Для начала обратимся к уровню L3, который устанавливает соединение между сервером и клиентом. Авторы в своей статье предлагают устанавливать соединение на каждый запрос обработки. В рамках редактора изображений это не очень практично, поскольку запросов может быть достаточно много, и чтобы не устанавливать соединение каждый раз, предлагается установить соединение единожды при входе в приложение и далее поддерживать сессию с сервером, пока пользователь не закроет приложение. Такую сессию можно реализовать при помощи временных токенов, которые будут получаться при входе в приложение в дальнейшем производить обращение к серверу через данный токен.

Теперь обратим внимание на уровень L2, который адаптирует данные на стороне клиента и сервера. Если говорить о подобной адаптации в рамках редактора изображений, то здесь основным пунктом адаптации будет изменение разрешения исходного изображения. Например, если исходное разрешение было в формате 3656×2664 пикселей, то размер такого изображения может быть до 20 мб и загрузка фото на сервер, обработка на сервере и выгрузка обратно на устройство пользователя займет большое количество времени и ресурсов, поэтому имеет смысл сжать такое изображение до разрешения 1920×1080 пикселей. Вполне логично, применять уровень L2 только на стороне клиента, чтобы сразу отсылать на сервер нужный формат изображения, тем самым экономя время на пересылку изображения и ресурсы сервера.

На рис. 2 представлено схематичное изображение предложенной архитектуры.

Реализация

Рассмотрим реализацию данного алгоритма и опишем возникшие сложности. Для реализации клиентского приложения использовался язык Swift. Реализованный клиент можно запускать на устройствах с системами iOS, iPadOS и MacOS. Для реализации многослойной обработки изображений, был реализован класс Stack, содержащий вложенный список функций обработки. Каждая из функций обработки реализует один из слоев многослойной обработки. Основная сложность, которая тут возникла, эта разрешение изображения. На этапе серверного применения ML-алго-

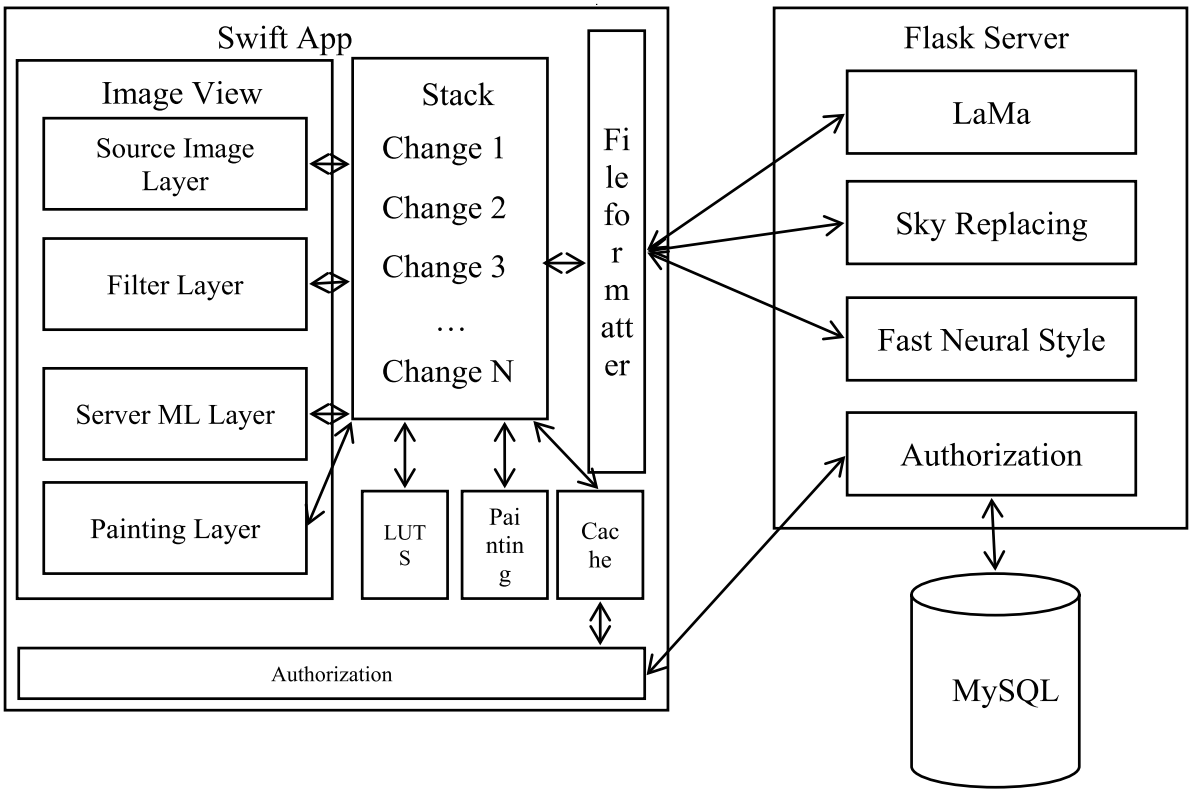

Рис. 3. Подробная архитектура редактора

ритмов, исходное изображение может быть уменьшено. Это связано с ресурсоемкостью GAN-алгоритмов и сделано для возможности экономии ресурсов сервера и скорости передачи данных. Для решения данной проблемы, было предложено решение в виде сенсоров на входе и выходе функций ML-обработки, которые следят за изменившимся разрешением изображения. Если разрешения поменялось, то дальнейшие алгоритмы после этапа ML, будут работать с корректным разрешением, поскольку получат эту информацию от сенсоров. Данная доработка алгоритма многослойной обработки изображения позволила решить проблему ресурсоемких ML-алгоритмов и адаптировать стек изменений к нужному разрешению.

Для возможности вызова ML-алгоритма с серверной части был реализован Flask-сервер на Python. Вызов на сервер приходит через https-запрос согласно реализованному API. В зависимости от запроса, запускается тот, либо иной алгоритм обработки изображения. Данное решение позволяет подключать неограниченное количество алгоритмов обработки изображений с обработкой на серверной части.

Фильтры были реализованы на основе LUT-изображений. LUT-изображения содержат в заданном порядке определенные цвета, которые применяются к исходному изображению. LUT-изображение применяется к изображению, которое получено после шага обработки ML.

Для возможности наложения стикеров и рисования поверх исходного изображения, был добавлен новый слой поверх исходного изображения. Данный слой позволяет пользователю рисовать и добавлять новые объекты поверх исходного изображения.

Стоит отметить, что слой Stack содержит в себе информацию о производимых изменениях на каждом слое. Так, если пользователь сменил фильтр, то будут пропущены шаги ML-обработки и рисования. Аналогично, если пользователь добавил стикер на изображение, то будут пропущены шаги ML-обработки и применения стикера. Данная доработка позволяет избавиться от излишних вычислений. Подробная схема реализации представлена на рис. 3.

Как видно, пользователь может взаимодействовать с 4 слоями: исходным изображением, фильтрами, серверными моделями и слоем рисования. Все изменения от пользователя переносятся в класс Stack и сохраняются. Далее, в зависимости от внесенного изменения вызывается тот, либо иной модуль: применение LUT-фильтра, применение рисования, либо обращение к серверу

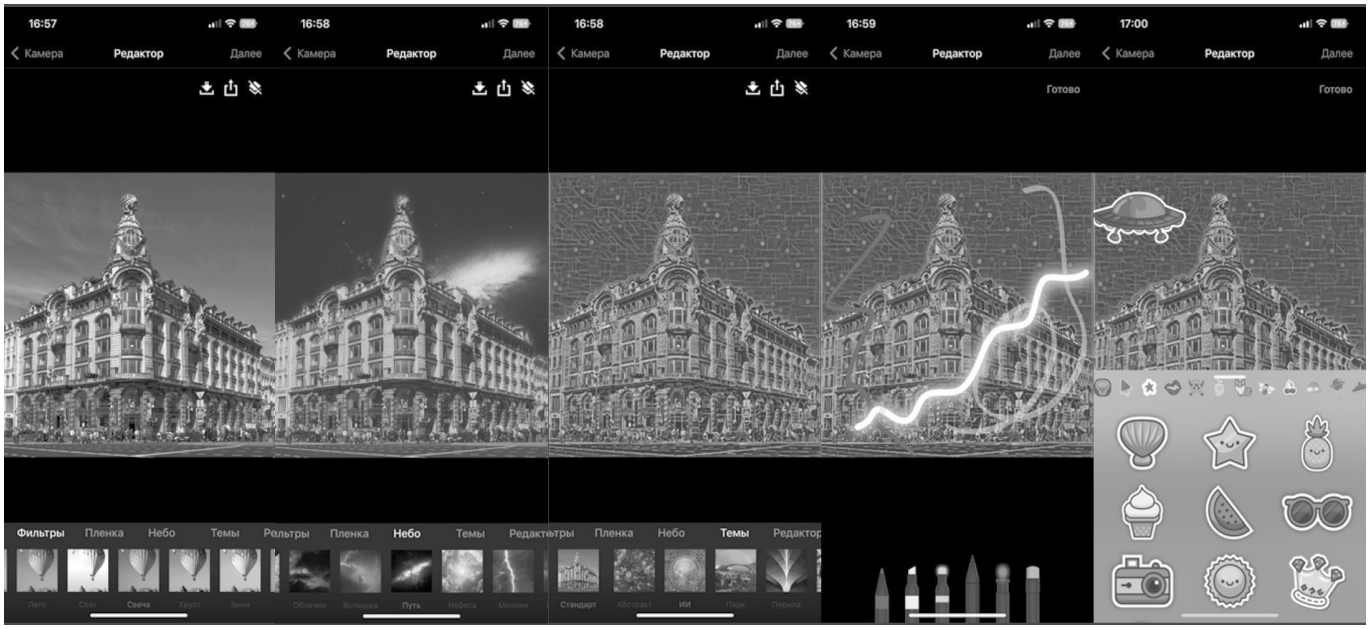

Рис. 4. Пример работы алгоритма многослойной обработки изображения по этапам: базовое изображение; использование алгоритма замены неба; наложение фильтра; наложение изображений и рисунков

Рис. 5. Скорость выполнения алгоритмов

и вызов необходимой модели. После применения изменения, оно отображается на необходимом слое в Image View. Также было реализовано кэширование, чтобы при повторных вызовах изменений с ML, не вызывать обращение к серверу повторно. Авторизация вынесена в отдельный модуль Authorization. Она происходит единожды при входе в приложение. При авторизации пользователь получает токен, который сохраняется в кэш и используется для последующей отсылки запросов на сервер. В качестве базы данных для хранения списка пользователей была использована MySQL [11] .

Примеры с реализованным интерфейсом пользователя приведены на рис. 4.

При использовании серверной части в редакторе изображений важное значение имеет скорость обработки на стороне сервера, так как на время обработки приходится отключать возможность внесения других изменений, пока не будут получены изменения со стороны сервера. Были проведены соответствующие тесты для используемых в редакторе моделей и получены следующие результаты. В качестве сервера, использо-

вался сервер с конфигурацией 2x2.2ГГц, 12Гб RAM, 50Гб HDD. Запросы отправлялись с клиента, установленного на iPhone. Расстояние между сервером и клиентом около 700 км. Так как перед отправкой изображения, происходит его сжатие до 1920x1080 пикселей, то использовалась изображение с изначально таким разрешением. Результаты тестирования представлены на рис. 5.

Как видно из графика, самым медленным оказался алгоритм LaMa, он выполнялся в районе 45 секунд. Стоит отметить, что время работы можно уменьшить путем использования сервера с GPU, так как все 3 алгоритма используют в качестве основы PyTorch. В любом случае, результат меньше минуты позволяет использовать редактор изображений на практике.

Заключение

В данной работе был рассмотрен алгоритм многослойной обработки изображений, который позволяет совмещать вычислительные ресурсы на локальном устройстве и серверной части, а также описаны произведенные доработки алгоритма обработки изображений на стороне сервера, которые позволили оптимизировать вычисления. Также был описан процесс реализации данного алгоритма на примере связки приложения на Swift в качестве клиента и сервера на Python в качестве серверной части. В качестве будущих направлений исследования, планируется внедрение дополнительных слоев в многослойную архитектуру — слой для добавления звука и слой для анимирования исходного изображения.

ЛИТЕРАТУРА

1. Foster, D., Generative deep learning. Sebastopol, CA: O'Reilly Media, 2019, p. 11–38.
2. Newnham J., Machine Learning with Core ML: An iOS developer's guide to implementing machine learning in mobile apps. Birmingham, England: Packt Publishing, 2018, p. 20–45.
3. Stevens, E., & Antiga, L., Deep learning with PyTorch. New York, NY: Manning Publications, 2020, p. 15–35.
4. Hope, T., Resheff, Y.S., & Lieder, I., Learning TensorFlow. Sebastopol, CA: O'Reilly Media, 2017, p.23–49.
5. Gulli, A., & Pal, S., Deep learning with keras. Birmingham, England: Packt Publishing, 2017, p. 18–42.
6. Монастырев В.В., Молодяков С.А. Методика модульно-конвейерной обработки данных на основе Spark SQL и Spark MLlib с интеграцией языков программирования // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и Технические Науки. —2022. —№ 06/2. — С. 119–124 DOI 10.37882/2223–2966.2022.06–2.26
7. Piantadosi, G., Marrone, S., Sansone, M., & Sansone, C., A secure, scalable and versatile multi-layer client–server architecture for remote intelligent data processing. In Journal of Reliable Intelligent Environments (Vol. 1). Springer Science and Business Media LLC, 2015, p. 173–187, DOI 10.1007/s40860–015–0007–1.
8. Suvorov, R., Logacheva, E., Mashikhin, A., Remizova, A., Ashukha, A., Silvestrov, A., Kong, N., Goka, H., Park, K., & Lempitsky, V., Resolution-robust Large Mask Inpainting with Fourier Convolutions. IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2022, p. 1–11, DOI 10.1109/wacv51458.2022.00323/
9. Zou, Z., Castle in the Sky: Dynamic Sky Replacement and Harmonization in Videos (Version 1), 2020, p. 1–11, DOI 10.48550/ARXIV.2010.11800.
10. Johnson, J., Alahi, A., & Fei-Fei, L., Perceptual Losses for Real-Time Style Transfer and Super-Resolution (Version 1), 2016, p. 1–18, DOI 10.48550/ARXIV.1603.08155.
11. Murach, J., Murach's MySQL, 3rd Edition (3rd ed.). Fresno, CA: Mike Murach & Associates, 2019, p. 3–149.

© Монастырев Виталий Викторович (vit34-95@mail.ru), Молодяков Сергей Александрович (molodyakov_sa@spbstu.ru).

Журнал «Современная наука: актуальные проблемы теории и практики»