МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени И.С. ТУРГЕНЕВА»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по направлению подготовки 01.04.02 Прикладная математика и информатика

направленность (профиль) Технологии параллельных и распределённых вычислений

Студента Тютюнова Игоря Андреевича шифр 205017

Факультет (институт) физико-математический

Разработка методов оптимизации   
взаимодействия веб-сайта с базой данных

Студент Тютюнов И.А.

Руководитель к.ф.-м.н., доц. Федяев Ю.С.

Зав. кафедрой Дорофеева В.И.

Орёл 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.С. ТУРГЕНЕВА»

Факультет физико-математический

Кафедра информатики

Направление подготовки 01.04.02 — Прикладная математика и информатика

Направленность (профиль) Технологии параллельных и распределённых вычислений

УТВЕРЖДАЮ:

Зав. кафедрой/РОП

\_\_\_\_\_\_\_\_\_\_\_\_ Дорофеева В.И.

«26» октября 2021г.

**ЗАДАНИЕ**

на выполнение выпускной квалификационной работы

студента Тютюнова Игоря Андреевича 01ПМ-М

1. Тема ВКР «Разработка методов оптимизации взаимодействия веб-сайта с базой данных» утверждена приказом по университету от «26» октября 2021 года № 2-2937

2. Срок сдачи студентом законченной работы «7» июля 2022 года

3. Исходные данные к работе: учебные пособия по аспектам разработки программного обеспечения, электронные источники и ссылки по программированию, алгоритмизации, разработке приложений.

4. Содержание ВКР

ВВЕДЕНИЕ

ГЛАВА 1. ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ ОБ ОПТИМИЗАЦИИ РАБОТЫ С БАЗОЙ ДАННЫХ

1.1. Описание реляционных баз данных

1.2. Описание операций взаимодействия с базой данных

ГЛАВА 2. ОПИСАНИЕ МЕТОДОВ ОПТМИЗАЦИИ ВЗАИМОДЕЙСТВИЯ С БАЗОЙ ДАННЫХ

2.1. Метод кеширования результатов запросов к базе данных

2.2. Метод синхронного выполнения операций с базой данных

ГЛАВА 3. РАЗРАБОТКА ВЕБ-БИБЛИОТЕК

3.1. Описание инструментов использованных при разработке

3.1.1. Описание языка программирования PHP

3.1.2. Инструмент хранения данных Redis

3.1.3. Описание протокола обмена сообщениями AMQP и брокера сообщений RabbitMQ

3.2. Постановка задачи

3.3. Разработка веб-библиотеки для кеширования работы программных алгоритмов

3.4. Разработка веб-библиотеки для синхронного выполнения операций с базой данных

3.5. Результат использования разработанных веб-библиотек

ЗАКЛЮЧЕНИЕ

СПИСОК ЛИТЕРАТУРЫ

ПРИЛОЖЕНИЕ

5. Перечень графического материала

рисунков — 7.

Дата выдачи задания«26» октября 2021 г.

Научный руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Федяев Ю.С.

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тютюнов И.А.

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |
| --- | --- | --- |
| Наименование этапов  ВКР | Срок выполнения  этапов работы | Примечание |
| Подбор и анализ источников и научных изданий в соответствии с темой исследования | ноябрь-декабрь 2021 | Выполнено |
| Написание введения | май 2022 г. | Выполнено |
| Написание главы 1 | январь-февраль 2022 г. | Выполнено |
| Написание главы 2 | март-апрель 2022 г. | Выполнено |
| Написание главы 3 | март-апрель 2022 г. | Выполнено |
| Написание заключения | май 2022 г. | Выполнено |
| Оформление ВКР | май 2022 г. | Выполнено |
| Сдача ВКР | июль 2022 г. | Выполнено |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тютюнов И.А.

Научный руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Федяев Ю.С.

**АННОТАЦИЯ**

Выпускная квалификационная работа на тему «Разработка методов оптимизации взаимодействия веб-сайта с базой данных» содержит 57 страниц текста, рисунков – 8, использованных источников – 38.

В современном мире трудно переоценить значимость баз данных. Базы данных прочно укрепили свои позиции повсеместно в промышленных, образовательных, здравоохранительных, правоохранительных и в других общественно важных структурах, а также в сферах бизнеса. Также, базы данных активно используются в веб-сайтах и от оптимальности взаимодействия веб-сайтов с базой данных зависит скорость работы первых. В свою очередь, скорость работы веб-сайтов влияет на комфорт их использования пользователями.

*Ключевые слова*: оптимизация, кеширвоание, Redis, брокер очередей, RabbitMQ, AMQP

*Предмет исследования*. Методы оптимизации работы веб-сайта с базой данных.

*Объект исследования*. Оптимизация взаимодействия веб-сайта с базой данных.

*Цель работы*. Разработать веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных.

*Метод исследования.* При исследовании применялся теоретический анализ литературы и материалов сети Интернет, методы структурного и объектно-ориентированного программирования.

*Результаты работы*. В работе разработаны веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных путём использования кеширования программных алгоритмов и синхронного выполнения CRUD-операция с базой данных.

*Работа имеет теоретическое и практическое значение*, т.к. базы данных активно используются в веб-сайтах и от оптимальности взаимодействия веб-сайтов с базой данных зависит скорость работы первых. В свою очередь, скорость работы веб-сайтов влияет на комфорт их использования пользователями.

**ANNOTATION**

Final qualifying work on the topic "Development of methods for searching websites with a database" contains 57 pages of text, figures - 7, used sources - 38.

In some cases, it is difficult to overestimate the sensitivity of databases. Databases have firmly established their positions in the field of industry, education, health, health care and other social and structural issues, as well as in the interests of business. In addition, database data is heavily used in websites and is different from other versions of database websites that depend on the speed of the former. In turn, the speed of the websites showed the convenience of their use by users.

*Keywords*: optimization, caching, Redis, queue broker, RabbitMQ, AMQP

*Subject of study*. Website performance optimization.

*Object of study.* Website flavor optimization with database.

*Objective.* Develop web libraries that implement methods of interaction between a website and a database.

*Research method*. Using the materials, a theoretical analysis of the literature and the Internet, methods of structural and object-oriented programming were prepared.

*Work results.* In the work, web libraries have been developed that implement methods for interacting websites with a database, using the use of caching software algorithms and synchronous execution of CRUD operations with the database.

*The work has theoretical and practical significance, because.* Database data is heavily used in websites and is distinguished by the unique effects of websites with a database that depend on the speed of the former. In turn, the speed of the websites showed the convenience of their use by users.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 9](#_Toc107176227)

[ГЛАВА 1. ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ ОБ ОПТИМИЗАЦИИ РАБОТЫ С БАЗОЙ ДАННЫХ 12](#_Toc107176228)

[1.1. ОПИСАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ 12](#_Toc107176229)

[1.2. ОПИСАНИЕ ОПЕРАЦИЙ ВЗАИМОДЕЙСТВИЯ С БАЗОЙ ДАННЫХ 18](#_Toc107176230)

[ГЛАВА 2. ОПИСАНИЕ МЕТОДОВ ОПТМИЗАЦИИ ВЗАИМОДЕЙСТВИЯ С БАЗОЙ ДАННЫХ 22](#_Toc107176231)

[2.1. МЕТОД КЕШИРОВАНИЯ РЕЗУЛЬТАТОВ ЗАПРОСОВ К БАЗЕ ДАННЫХ 22](#_Toc107176232)

[2.2. МЕТОД СИНХРОННОГО ВЫПОЛНЕНИЯ ОПЕРАЦИЙ С БАЗОЙ ДАННЫХ 24](#_Toc107176233)

[ГЛАВА 3. РАЗРАБОТКА ВЕБ-БИБЛИОТЕК 26](#_Toc107176234)

[3.1. ОПИСАНИЕ ИНСТРУМЕНТОВ ИСПОЛЬЗОВАННЫХ ПРИ РАЗРАБОТКЕ 26](#_Toc107176235)

[3.1.1. ОПИСАНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ PHP 26](#_Toc107176236)

[3.1.2. ИНСТРУМЕНТ ХРАНЕНИЯ ДАННЫХ REDIS 27](#_Toc107176237)

[3.1.3. ОПИСАНИЕ ПРОТОКОЛА ОБМЕНА СООБЩЕНИЯМИ AMQP И БРОКЕРА СООБЩЕНИЙ RABBITMQ 29](#_Toc107176238)

[3.2. ПОСТАНОВКА ЗАДАЧИ 41](#_Toc107176239)

[3.3. РАЗРАБОТКА ВЕБ-БИБЛИОТЕКИ ДЛЯ КЕШИРОВАНИЯ РАБОТЫ ПРОГРАММНЫХ АЛГОРИТМОВ 43](#_Toc107176240)

[3.4. РАЗРАБОТКА ВЕБ-БИБЛИОТЕКИ ДЛЯ СИНХРОННОГО ВЫПОЛНЕНИЯ ОПЕРАЦИЙ С БАЗОЙ ДАННЫХ 49](#_Toc107176241)

[3.5. РЕЗУЛЬТАТ ИСПОЛЬЗОВАНИЯ РАЗРАБОТАННЫХ ВЕБ-БИБЛИОТЕК 57](#_Toc107176242)

[ЗАКЛЮЧЕНИЕ 59](#_Toc107176243)

[СПИСОК ЛИТЕРАТУРЫ 60](#_Toc107176244)

[ПРИЛОЖЕНИЕ 65](#_Toc107176245)

# ВВЕДЕНИЕ

**Обоснование выбора темы и ее актуальность**

В современном мире трудно переоценить значимость баз данных. Базы данных прочно укрепили свои позиции повсеместно в промышленных, образовательных, здравоохранительных, правоохранительных и в других общественно важных структурах, а также в сферах бизнеса. Также, базы данных активно используются в веб-сайтах и от оптимальности взаимодействия веб-сайтов с базой данных зависит скорость работы первых. В свою очередь, скорость работы веб-сайтов влияет на комфорт их использования пользователями.

Высокую нагрузку на базу данных вызывают операции создания, чтения, изменения и удаления данных. Эти операции можно объединить одной аббревиатурой — CRUD операции (Create, Read, Update, Delete).

Высокая нагрузка на базу данных, которая может возникать при выполнении CRUD-операции, увеличивает нагрузку на веб-сервер, что отрицательным образом сказывается на скорости работы веб-сайта в целом. Высокая нагрузка на веб-сервер потребует дополнительных затрат на его поддержку. А низкая скорость работы веб-сайта отрицательным образом скажется на комфорте использования сайта пользователями. Пользователей, в свою очередь, не будет устраивать скорость работы веб-сайта, в следствии чего они будут выбирать более производительные сайты конкурентов.

Таким образом, проблема оптимизации взаимодействия веб-сайта с базой данных очень актуальна. Особенно сильно это проблема актуально для веб-сайтов, занимающихся коммерческой деятельностью, например, интернет магазины. Ведь уровень комфорта использования интернет-магазинов напрямую влияет на количество активных пользователей, а соответственно, от этого зависит прибыль. Для решения проблемы оптимизации взаимодействия веб-сайта с базой данных, в рамках выпускной квалификационной работы, будут разработаны веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных.

**Предмет исследования**

Методы оптимизации работы веб-сайта с базой данных.

**Объект исследования**

Оптимизация взаимодействия веб-сайта с базой данных.

**Цель работы**

Разработать веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных.

**Основные задачи исследования**

1. Изучить операции взаимодействия веб-сайта с базой данных;
2. Проанализировать методы оптимизации взаимодействия веб-сайта с базой данных;
3. Разработать веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных.

**Структура работы**

Работа состоит из введения, трёх глав, заключения и списка источников.

Во введении рассматривается актуальность работы, ставится цель и формулируются задачи, необходимые для достижения поставленной цели.

Первая глава представляет собой описание операций взаимодействия с базой данных.

Во второй главе описываются методы оптимизации взаимодействия с базой данных.

В третьей главе описывается разработка веб-библиотек реализующих методы оптимизации веб-сайта с базой данных и результаты их использования.

В заключении делаются выводы по проделанной работе.

В конце работы приводится список использованных источников

# ГЛАВА 1. ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ О ВЗАИМОДЕЙСТВИИ ВЕБ-САЙТА С БАЗОЙ ДАННЫХ

## Описание реляционных баз данных

Реляционные базы данных — это средство для рационального и эффективного хранения информации. База данных обеспечивают надежную защиту данных от случайной потери или порчи, экономно используют ресурсы (как людские, так и технические) и снабжена механизмом поиска информации удовлетворяющим разумным требованиям к производительности. При разработке баз данных используют средства систем управления базами данных (СУБД).

Реляционные базы данных приобрели свою популярность в связи с реализацией в системах управления реляционных моделей, что является очень удобным при работе с данными. Само понятие «Система управления базами данных» разработал англичанин Эдгара Кодд. Реляционные модели управления характеризуются своей простотой, удобной табличной форме, а также применением формальной математики и реляционных вычислений для обрабатываемых данных [22].

Хранение данных, в реляционных базах, имеют вид таблиц, которые состоят из строк и столбцов. Поля таблицы имеют собственное наименование. Столбцы таблицы могут иметь данные скалярного вида, такие как даты, числа и др.

Отношение таблиц, в реляционной базе данных, представляются как один к одному, многие ко многим или один ко многим.

Строки записей могут представляться в неограниченном количестве, но каждая запись отвечает за свое запись строк записей в таблице неограниченно, и каждая запись соответствует своей сути [17].

Данные, в реляционных моделях, представляют собой двумерный массив и характеризуются следующими особенностями:

1. Любая составляющая таблицы является одной составляющей данных;
2. Любой столбец имеет свое уникальное имя;
3. Отсутствие одинаковых строк в таблице;
4. Все составляющие в столбцах имеют однородный тип;
5. Строки и столбцы имеют произвольный порядок.

Большинство баз данных предназначено для моделирования некоторых предметных областей реального мира.

Модель данных — это концептуальное описание предметной области. Она включает определение сущностей и их атрибутов: например, сущность покупатель имеет атрибут имя и адрес. Для сущности определяем ограничения. Модель данных не содержит ссылок и указаний на физическую модель самой системы. Предметная область — это определенная часть реального мира. Модель данных — это концептуальное описание предметной области. Схема базы данных содержит описание модели данных, используемое базой данных.

Таким образом, база данных представляет собой реализацию схемы базы данных и модели данных.

Механизм базы данных — это специальные средства, предназначенные для физического манипулирования данными: хранением их на диске и извлечением по запросу. SQL Server использует клиент-серверную архитектуру и предназначен для создания систем, от средних до больших. Он прекрасно масштабируется и может поддерживать несколько тысяч пользователей, работающих с важными приложениями.

Реляционная модель основывается на математических принципах, вытекающих непосредственно из теории множеств и логики предикатов. Эти принципы впервые были применены в области моделирования данных в конце 60-х гг. доктором Е.Ф. Коддом, в то время работавшим в IBM, а впервые опубликованы — в 1970 г. Реляционная модель определяет способ представления данных (структуру данных), методы зашиты данных (целостность данных), а также операции, выполняемые с данными (манипулирование данными).

Реляционная модель не единственный метод хранения и манипулирования данными. Существуют альтернативные варианты: иерархическая, сетевая, а также звездообразная модели данных. У каждой из них свои преимущества при решении задач определенного типа.

В общих чертах основные принципы реляционных систем баз данных можно сформулировать так:

• Все данные на концептуальном уровне представляются в виде упорядоченной организации, определенной в виде строк и столбцов и называемой отношением.

• Все значения являются скалярами. Это означает, что для любой строки и столбца любого отношения существует одно и только одно значение.

• Все операции выполняются над целым отношением, и результатом выполнения этих операций также является целое отношение. Этот принцип называется замыканием.

Принцип замыкания заключается в том, что и базовые таблицы, и результаты операций над ними на концептуальном уровне представляются как отношения. Он позволяет непосредственно использовать результаты одной операции в качестве исходных данных для выполнения другой.

Рассмотрим термины, используемые в реляционной теории.

Сущность — это нечто, о чем нужно хранить информацию в разрабатываемой системе.

Атрибуты. Атрибуты сущности — это записи об определенных параметрах каждой из сущностей. (сущность Покупатель, атрибуты: фамилия, имя, род деятельности или сущность «Звонок в службу технической поддержки», атрибутами являются: кто звонил, когда, вид проблемы, удалось ли решить проблему).

Домен. Домен определяет «вид» данных, которые представляет данный атрибут. Если дать более четкое определение, то домен — это набор всех допустимых значений, которые может содержать данный атрибут.

Связи. Кроме атрибутов каждой сущности модель данных должна определять связи между сущностями. На концептуальном уровне связи представляют собой простые ассоциации между сущностями.

Рассмотрим основные принципы нормализации реляционных баз данных. Принципы нормализации, являются инструментом контроля структуры данных. Нормальные формы определяют возрастающую строгость правил, которым подчиняются структуры отношений. Мы рассмотрим шесть таких форм. Каждая последующая форма расширяет предыдущую, устраняя при этом возможность возникновения аномалий обновления определенного типа.

Нормальные формы не являются руководством для создания «правильной» модели данных. Реляционная модель позволяет различным образом соединять отношения, связывая их через атрибуты. Процесс получения полностью нормализованной модели данных включает в себя устранение избыточности. Для этого отношение, содержащее избыточные данные, разбивают на несколько других отношений. Нужно сделать это так, чтобы получившиеся в результате отношения можно было бы вновь соединить и получить точную копию структуры и данных исходного отношения.

Мы знаем, что содержимое отношения — это неупорядоченное множество, состоящее из 0 или более кортежей, и каждый элемент множества кортежей должен быть уникален. В таком случае для любого отношения должна существовать комбинация атрибутов, однозначно определяющая каждый кортеж. Такой набор из одного или более атрибутов называют ключом-кандидатом.

Может существовать более одного ключа-кандидата, но каждый ключ-кандидат должен однозначно определять каждый кортеж, не только для любого специфического множества кортежей, а для всех возможных кортежей в любой момент времени. Обратный принцип тоже верен: если взять два кортежа с одинаковыми значениями ключа-кандидата, то оба этих кортежа должны представлять одну и ту же сущность. Однозначная идентификация любого кортежа является непременным требованием к ключу-кандидату. Необходимо досконально понимать специфику предметной области, чтобы правильно определить ключевой набор атрибутов.

Первая нормальная форма. Отношение находится в первой нормальной форме, если домены, в которых определены его атрибуты, являются скалярными величинами. Понятие скалярной величины — одновременно самое простое и самое сложное в моделировании данных. Принцип таков: каждый атрибут кортежа должен содержать отдельную величину.

Вторая нормальная форма. Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме, и кроме того, все его атрибуты зависят от полного набора атрибутов ключа-кандидата.

Третья нормальная форма. Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме, и кроме того, все не ключевые атрибуты совершенно независимы.

Решение, когда и как вводить в модель третью нормальную форму, может приниматься только с учетом семантики строящейся модели (впрочем, как и любое другое решение в ходе построения модели данных). Как правило, отдельные отношения проектируются только для моделирования наиболее важных сущностей предметной области, или когда предполагается частое изменение каких-то данных, или когда вы получаете в результате создания конкретного отношения серьезные технологические выгоды.

Нормальная форма Бойса-Кодда. Нормальная форма Бойса-Кодда рассматривается как вариант третьей нормальной формы. Она имеет дело со специальной разновидностью отношений, для которых существует несколько ключей-кандидатов. Фактически, чтобы применять нормализацию по Бойсу-Кодду, нужно сочетание нескольких условий:

• отношение должно иметь не менее двух ключей-кандидатов;

• по крайней мере два ключа-кандидата должны быть составными;

• ключи-кандидаты должны иметь перекрывающиеся атрибуты.

Четвертая нормальная форма. Четвертая нормальная форма подводит теоретическую базу под интуитивно очевидный принцип: независимые повторяющиеся группы данных не следует размещать в одном и том же отношении. Упрощенно, нормализация до четвертой нормальной формы состоит в выделении многозначных зависимостей в разные отношения. Формально отношение находится в четвертой нормальной форме, если оно находится в нормальной форме Бойса-Кодда, и кроме того, все многозначные зависимости являются также функциональными зависимостями от ключей-кандидатов. Приведение отношения к четвертой нормальной форме актуально, только если между атрибутами существуют многозначные связи.

Пятая нормальная форма. Пятая нормальная форма имеет дело с чрезвычайно редко встречающимся случаем зависимостей соединения. Зависимости соединения подчиняются следующему принципу: «Если сущность 1 зависит от сущности 2, сущность 2 зависит от сущности 3, а сущность 3 в свою очередь зависит от сущности 1, то все три сущности обязательно должны входить в один и тот же кортеж».

Ранее был подробно рассмотрен процесс нормализации модели данных. Суть этого процесса — в анализе сущностей предметной области для моделирования отношений, охватывающих все относящиеся к ним данные. Но отношения составляют лишь определенную часть модели данных. Ее другая неотъемлемая часть — связи между отношениями и ограничения, налагаемые на эти связи.

Сущности, между которыми существуют связи, называются «участниками», а число участников связи — «размерностью» связи. Большинство связей между сущностями — это двойные связи, то есть такие, в которых участвуют две сущности. Встречаются также унарные связи (в которых сущность связана сама с собой) и тройные связи (в которых участвуют три сущности). Участие каждой сущности в связи бывает «полным» или «частичным», в зависимости от того, может ли эта сущность существовать, если данная связь не определена. Участие каждой сущности в связи бывает «полным» или «частичным», в зависимости оттого, может ли эта сущность существовать, если данная связь не определена. Связи можно классифицировать одним из трех возможных способов: как «полные» или «частичные», «необязательные» или «обязательные», а также в терминах «слабых» и «обычных» сущностей.

## Описание операций взаимодействия с базой данных

CRUD — акроним, который обозначает четыре базовые операции, используемые при взаимодействии с базой данных:

1. Создание (англ. create),
2. Чтение (англ. read),
3. Модификация (англ. update),
4. Удаление (англ. delete).

Акроним CRUD введён Джеймсом Мартином (англ. James Martin) в 1983 году как классификация функций по манипуляции данными [1].

В системах, реализующих доступ к базе данных с помощью архитектурного стиль взаимодействия REST-API, CRUD функции реализуются через типы запросов POST, GET, PUT и DELETE, соответственно [14].

Рассмотрим подробнее процесс создания новой записи. Функция «Create» предназначена для добавления новых строк в таблицу. Это можно сделать с помощью команды INSERT INTO — ключевого слова, за которым следует название таблицы. Далее указываются имена столбцов и значения, которые нужно вставить (листинг 1). Функция «INSERT INTO» добавит новые строки в таблицу, и каждой созданной записи будет присвоен свой уникальный идентификатор.

Листинг 1 — Примеры запросов на добавление записи

INSERT INTO table\_name

VALUES (value1, value2, value3, ...);

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

Функция «Read», позволяет извлекать определенные записи из базы данных и считывать их значения. Это можно сделать с помощью команды SELECT — ключевого слова, за которым следует название таблицы. Ниже приведен пример такого запроса (листинг 2).

Листинг 2 — Пример запроса вывода данных

SELECT \* FROM table\_name

Этот запрос не внесет никаких изменений в таблицу, а отобразит все существующие записи в этой таблице. Также можно указать критерий поиска записей посредством добавления секции WHERE (листинг 3).

Листинг 3 — Пример запроса вывода данных с условием

SELECT \* FROM table\_name

WHERE NAME LIKE ‘%a’

Обновление «Update» — данная операция позволяет изменять существующих записей в базе данных. Это можно сделать с помощью команды UPDATE. При выполнении операции UPDATE необходимо определить целевую таблицу и столбцы, которые необходимо обновить (листинг 4). Также необходимо указать условие, по которому будут выбираться строки для обновления. В противном случае будут обновлены все указанные столбы таблицы.

Листинг 4 — Примеры запроса на обновление

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE ID = 3;

Операция «Delete» используется для удаления записи из таблицы. Это можно сделать с помощью команды DELETE (листинг 5). В SQL реализована возможность удаления, как одной конкретной записи, посредством указания соответствующего, так и удаление всех записей, содержащихся в таблице.

Листинг 5 — Пример запроса на удаление

DELETE FROM table\_name

WHERE ID = 3;

Благодаря функциям чтения, создания, обновления и удаления удается организовать простое и правильное взаимодействие с хранимыми данными. Также позволяя разграничивать доступ пользователей по группам. Для одной группы разрешать только чтение записей, а другой предоставлять доступ к созданию и обновлению, удалению записей. Операции CRUD являются минимально необходимыми как для взаимодействия пользователей с системой, так и для разработчиков системы.

CRUD-операции требовательны к ресурсам веб-сервера, особенно это касается баз данных с большим количеством таблиц и строк в таблицах.

Операция «Read» способна спровоцировать высокую нагрузку на веб-сервер, в случае большого количества запросов к базе данных, например, по посещении страниц веб-сайта пользователями.

Операции «Create», «Update» и «Delete», кроме выполнения своих непосредственных функций, инициируют пересоздание индексов в таблицах баз данных, которые используются для более быстрого выполнения «Read» операций, что также создаёт дополнительную нагрузку на веб-сервер.

# ГЛАВА 2. Анализ МЕТОДОВ ОПТМИЗАЦИИ ВЗАИМОДЕЙСТВИЯ С БАЗОЙ ДАННЫХ

## 2.1. Метод кеширования результатов запросов к базе данных

В сфере вычислительной обработки данных кэш — это высокоскоростной уровень хранения, на котором требуемый набор данных, как правило, хранится в течении ограниченного периода времени.

На этом уровне доступ к данным осуществляется значительно быстрее, чем к основному месту их хранения. Кэширование позволяет эффективно повторно использовать ранее извлеченные или вычисленные данные.

Данные в кэше обычно хранятся на устройстве с быстрым доступом, таком как ОЗУ (оперативное запоминающее устройство), и могут использоваться совместно с программными компонентами. Основная функция кэша — ускорение процесса извлечения данных. Он избавляет от необходимости обращаться к менее скоростному базовому уровню хранения.

Небольшой объем памяти кэша компенсируется высокой скоростью доступа. В кэше обычно хранится только требуемый набор данных, причем временно, в отличие от баз данных, где данные обычно хранятся полностью и постоянно.

Поскольку ОЗУ и работающие в памяти сервисы обеспечивают высокие показатели скорости обработки запросов, или IOPS (количество операций ввода-вывода в секунду), кэширование повышает скорость извлечения данных и сокращает расходы при работе в больших масштабах. Чтобы обеспечить аналогичный масштаб работы с помощью традиционных баз данных и оборудования на базе жестких дисков, требуются дополнительные ресурсы. Использование этих ресурсов приводит к повышению расходов, но все равно не позволяет достигнуть такой низкой задержки, какую обеспечивает кэш в памяти.

Кэш используется на разных технологических уровнях, включая операционные системы, сетевые уровни, в том числе сети доставки контента (CDN) и систем доменных имён (DNS), интернет-приложения и базы данных.

С помощью кэширования можно значительно сократить задержки и повысить производительность операций ввода-вывода в секунду для многих рабочих нагрузок приложений с большой нагрузкой на чтение, например порталов для вопросов и ответов, игровых ресурсов, порталов для распространения мультимедиа и социальных сетей.

Кэшировать можно результаты запросов к базам данных, вычислений, которые требовательны к ресурсам, запросы к API и ответы на них, а также веб-артефакты, например файлы HTML, JavaScript и изображений.

Рабочие нагрузки, требующие больших вычислительных мощностей для обработки наборов данных, например сервисы рекомендаций и высокопроизводительное вычислительное моделирование, тоже могут эффективно использовать уровень данных в памяти в качестве кэша.

При реализации уровня кэша необходимо принимать во внимание достоверность кэшируемых данных. Эффективный кэш обеспечивает высокую частоту попаданий, то есть наличия в кэше запрашиваемых данных. Для удаления из кэша неактуальных данных применяются такие механизмы, как TTL (время жизни). Следует также понимать, требуется ли для среды кэширования высокая доступность. Если она необходима, можно использовать сервисы в памяти, такие как Redis [38].

Преимущества использования кеширования:

1. Повышение производительности приложений;
2. Сокращение затрат на поддержку базы данных;
3. Снижение нагрузки на серверную часть;
4. Повышение пропускной способности операций чтения (количество операций ввода-вывода в секунду).

Для большинства сайтов кэшировать лучше:

* Логотипы и изображения бренда.
* Не ротируемые изображения в целом (например, значки навигации).
* Стили.
* Общие файлы Javascript.
* Загружаемый контент.
* Файлы мультимедиа.

Эти элементы меняются нечасто, поэтому их можно кэшировать в течение более длительных периодов времени.

А эти элементы нужно кэшировать осторожно:

* HTML-страницы.
* Ротируемые изображения.
* Часто изменяемые Javascript и CSS.
* Контент, запрашиваемый с помощью файлов cookie.

Кэшировать не рекомендуется:

* Активы, связанные с конфиденциальными данными. (банковская информация и т. д.)
* Контент, который зависит от пользователя и часто изменяется.

В дополнение к вышеуказанным общим правилам можно настроить политики, которые позволят вам соответствующим образом кэшировать различные типы контента. Например, если все авторизовавшиеся пользователи видят один и тот же вид сайта, может быть полезно кэшировать его. Если авторизовавшиеся пользователи видят пользовательский вид сайта, который будет действителен в течение некоторого времени, вы можете сохранить кэш в браузере пользователя, но исключить промежуточное кэширование.

Существует много ситуаций, когда кеширование невозможно или не должно реализовываться из-за того, как создается контент (динамически генерируемый контент для каждого пользователя), или из-за характера контента (например, это конфиденциальная банковская информация). Еще одна проблема, с которой сталкиваются многие администраторы при настройке кеширования, – это ситуация, когда более старые версии вашего контента находятся в кэше и еще не устарели, даже если уже появились новые версии.

Эти распространенные проблемы могут серьезно повлиять на производительность кэша и точность контента. Однако их можно смягчить, разработав политику кэширования с их учетом.

Стратегия кэширования всегда зависит от ситуации, но следующие рекомендации могут помочь вам в принятии некоторых решений.

Есть несколько шагов, которые можно предпринять, чтобы увеличить коэффициент попадания в кэш:

* Необходимо устанавливать определенные каталоги для изображений, css и общего контента. Размещение контента в выделенных каталогах позволит легче ссылаться на них с любой страницы сайта.
* Необходимо использовать одинаковые URL-адреса для ссылок на одинаковые элементы. Поскольку кэш отталкивается как от хоста, так и от пути к запрошенному контенту.
* Необходимо использовать CSS-спрайты для таких элементов как значки и навигация. Такой подход уменьшают количество запросов, необходимых для рендеринга вашего сайта, и позволяют сайту кешировать этот единственный спрайт в течение длительного времени.
* Необходимо хранить сценарии и внешние ресурсы локально. Если вы используете скрипты javascript и другие внешние ресурсы, необходимо размещать эти ресурсы сервере, на котором расположен веб-сайт.

Главной целью политики кэширования является достижение баланса, который позволяет применять агрессивное кэширование, когда это возможно, и оставляет возможность аннулировать записи при внесении изменений.

Правильная политика кеширования может оказать значительное влияние на сайт. Кэширование позволяет сократить расходы, связанные с одновременным обслуживанием одного и того же контента.

Сервер также сможет обрабатывать большее количество трафика с помощью того же аппаратного обеспечения. Возможно, самое главное – кэширование может улучшить пользовательский опыт, благодаря чему посетители будут возвращаться на сайт. Эффективное кэширование может значительно повысить производительность.

## 2.2. Метод синхронного выполнения операций с базой данных

Синхронного выполнения операций с базой данных, таких как добавление, изменение, удаление данных, можно добиться использовав брокеры сообщений.

Брокер сообщений — это отдельный сервис, который отвечает за хранение и доставку данных от сервисов-отправителей к сервисам-получателям с помощью модели Producer / Consumer.

Брокер сообщений представляет собой тип построения архитектуры, при котором элементы системы «общаются» друг с другом с помощью посредника. Благодаря его работе происходит снятие нагрузки с веб-сервисов, так как им не приходится заниматься пересылкой сообщений: всю сопутствующую этому процессу работу брокер сообщений берёт на себя.

Можно сказать, что в работе любого брокера сообщений используются две основные сущности: producer (издатель сообщений) и consumer (потребитель/подписчик).

Producer (издатель сообщений) публикуют новую информацию в виде сгруппированных по некоторому атрибуту сообщений;

Consumer (потребитель/подписчик) подписываются на потоки сообщений с определенными атрибутами и обрабатывают их.

Одна сущность занимается созданием сообщений и отправкой их другой сущности-потребителю. В процессе отправки есть ещё серединная точка, которая представляет собой папку файловой системы, где хранятся сообщения, полученные от продюсера.

Возможны несколько вариантов передачи и получения сообщений:

1. Сообщение отправляется напрямую от отправителя к получателю;
2. Схема публикации/подписки.

В первом случае, когда сообщение отправляется напрямую от отправителя к получателю, каждое сообщение используется лишь однократно. Схему данного варианта передачи и получения сообщений можно увидеть на рисунке 1.

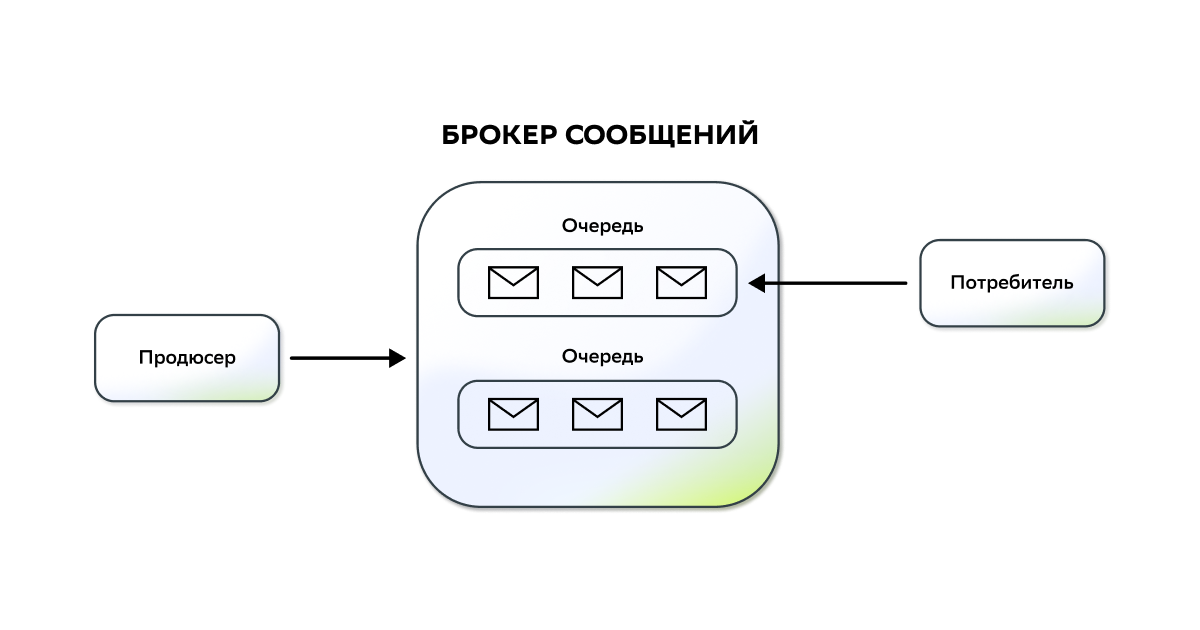


Рисунок 1 — Схема передачи сообщений напрямую от отправителя к получателю

Во втором случае, когда сообщения передаются по схеме публикации/подписки, отправитель не знает своих получателей и просто публикует сообщения в определённую тему. Потребители, которые подписаны на эту тему, получают сообщение. Далее на базе этой системы может быть построена работа с распределением задач между подписчиками. То есть выстроена логика работы, когда в одну и ту же тему публикуются сообщения для разных потребителей. Каждый «видит» уникальный маркер своего сообщения и забирает его для исполнения. Схему данного варианта передачи и получения сообщений можно увидеть на рисунке 2.



Рисунок 2 — Передача сообщений по схеме публикации/подписки

Группирующим сообщения атрибутом выступает очередь, которая нужна, чтобы разделять потоки данных. Таким образом, получатели могут подписываться только на те группы сообщений, которые их интересуют.

Очередь можно представить как канал связи, между писателем и читателем. Писатели кладут сообщения в очередь, после чего они передаются читателям, которые подписаны на эту очередь. Один читатель получает одно сообщение за раз, после чего оно становится недоступно другим читателям.

Под сообщением же подразумевается единица данных, обычно состоящая из тела сообщения и метаданных брокера. В общем случае, тело представляет из себя набор байт определенного формата.

Получатель обязательно должен знать этот формат, чтобы после получения сообщения иметь возможность обработать сообщение.

По такому принципу работает большинство брокеров сообщений, построенных на AMQP (Advanced Message Queuing Protocol) — протоколе, который описывает стандарт отказоустойчивого обмена сообщениями посредством очередей.

Данный подход обеспечивает несколько важных преимуществ:

1. Слабая связанность;
2. Масштабируемость;
3. Эластичность.

Слабая связанность достигается за счет асинхронной передачи сообщений: то есть, отправитель передаёт данные и продолжает работать, не дожидаясь ответа от получателя, а получатель обрабатывает сообщения, когда удобно ему, а не когда они были отправлены.

Масштабируемость. Если сообщения появляются в очереди быстрее, чем читатель успевает их обрабатывать мы можем запустить несколько читателей и подписать их на одну очередь.

Эластичность. Наличие между приложениями такой прослойки, как очередь, помогает справляться с пиковыми нагрузками: в этом случае очередь будет выступать буфером, в котором сообщения будут копиться и по мере возможности считываться читателем [10].

Для чего нужны брокеры сообщений:

1. Для организации связи между отдельными службами, даже если какая-то из них не работает в данный момент. То есть продюсер может отправлять сообщения, несмотря на то, проявляет ли активность потребитель в настоящее время;
2. За счёт асинхронной обработки задач можно увеличить производительность системы в целом;
3. Для обеспечения надёжности доставки сообщений: как правило, брокеры обеспечивают механизмы многократной отправки сообщений в тот же момент или через определённое время. Кроме того, обеспечивается соответствующая маршрутизация сообщений, которые не были доставлены.

Недостатки брокеров сообщений:

1. Усложнение системы в целом как таковой, так как в ней появляется ещё один элемент. Кроме того, возникает зависимость от надёжности распределённой сети, а также потенциальная возможность возникновения проблем из-за потребности в непротиворечивости данных, так как некоторые элементы системы могут обладать неактуальными данными;
2. Из-за асинхронной работы всей системы, а также её распределённого характера могут возникать ошибки, выяснение сути которых может стать непростой задачей;
3. Освоение подобных систем является не самым простым вопросом и может занять существенное время.

Когда брокеры сообщений могут быть полезны:

1. Если в рамках вашей системы есть действия, которые требуют для своего выполнения много времени и потребляют много ресурсов, при этом они не требуют немедленного результата;
2. Микросервисы: если ваша система достаточно сложна и состоит из отдельных сервисов, то для их координации можно использовать брокер сообщений, который в этом случае будет выступать в роли как бы центрального роутера. Каждый сервис подписывается только на свой тип сообщений, выстраивается определённая логика их обработки;
3. Мобильные приложения: здесь возможен вариант с задействованием push-уведомлений, когда множество смартфонов с установленным приложением подписаны на определённую тему. Если в ней публикуется какая-либо новость, то подписанный смартфон выводит уведомление;
4. Транзакционные системы: если какое-то действие системы состоит из множества отдельных этапов, каждый из которых выполняется отдельным элементом системы, то в этом случае брокер сообщений может выступить в роли своеобразной «доски уведомлений». Каждый сервис отписывается после того, как его этап общей задачи был выполнен. После этого в работу вступает следующий сервис, обрабатывающий, соответственно, следующий этап общей задачи.

Брокеров сообщений существует большое множество. Например:

1. Apache ActiveMQ;
2. Apache Kafka;
3. [Apache Qpid](https://ru.wikipedia.org/wiki/Apache_Qpid);
4. [RabbitMQ](https://ru.wikipedia.org/wiki/RabbitMQ).

Каждый из них обладает определёнными возможностями и хорошо решает возложенные на него задачи. Например, если одни предназначены для создания инфраструктуры связи между распределёнными частями приложения, другие предназначены для достаточно специфических задач, например «интернета вещей», и функционируют на основе легковесного протокола MQTT. Подобные брокеры служат для сбора статистики, температуры и других показателей с распределённых датчиков, установленных на определённых машинах, элементах конструкций, географически разных точках территории.

Малое время задержки для прохода сообщения по сети, а также возможность двунаправленной связи в реальном времени позволяют реализовывать достаточно интересные применения. Среди них, может быть, управление робототехническими устройствами в реальном времени. При таком управлении задержка не превышает десятков миллисекунд, что вполне допустимо для низкоскоростных устройств. Например, для управления роботами, движущимися в промышленных цехах и использующимися для перевозки деталей от станков или сырья к производственным станкам.

# ГЛАВА 3. РАЗРАБОТКА ВЕБ-БИБЛИОТЕК

## Инструменты разработки

### Язык программирования PHP

PHP (Hypertext Preprocessor) — это интерпретируемый язык программирования общего назначения с открытым исходным кодом. PHP специально сконструирован для веб-разработки и его код может внедряться непосредственно в HTML [9]. Главная область применения PHP — это написание скриптов, которые выполняются на веб-сервере.

PHP является веб-ориентированным языком программирования с динамической типизацией. Этот язык программирования можно сочетать с HTML кодом. Как правило, программы, написанные на языке программирования РНР, выполняются на веб-сервере, а результат отправляется браузеру в виде HTML разметки. Синтаксис языка программирования PHP схож с синтаксисом языка Си.

Основные возможности языка программирования PHP:

1. Автоматическое извлечение POST- и GET-параметров, а также переменных окружения веб-сервера в предопределённые массивы;
2. Взаимодействие с большим количеством различных систем управления базами данных через дополнительные модули (MySQL, MySQLi, SQLite, PostgreSQL, Oracle Database (OCI8), Microsoft SQL Server, Sybase, ODBC, mSQL, IBM DB2, Cloudscape и Apache Derby, Informix, Ovrimos SQL, Lotus Notes, DB++, DBM, dBase, DBX, FrontBase, FilePro, Ingres II, SESAM, Firebird и InterBase, Paradox File Access, MaxDB, интерфейс PDO, Redis);
3. Автоматизированная отправка HTTP-заголовков;
4. Работа с HTTP-авторизацией;
5. Работа с cookies и сессиями;
6. Работа с локальными и удалёнными файлами, сокетами;
7. Обработка файлов, загружаемых на сервер;
8. Создание и работа с API;
9. Создание приложение с графическим интерфейсом пользователя при использовании фреймворка Qt Designer;
10. Создание консольных приложений;

### Хранилище данных Redis

Redis (Remote DIctionary Server) — это нереляционное хранилище структур данных в памяти с открытым исходным кодом (под лицензией BSD).

Redis предоставляет такие структуры данных как строки, хэши, списки, наборы, отсортированные наборы с запросами диапазона. С этими структурами данных можно выполнять атомарные операции, такие как добавление к строке, увеличение значения в хэше, добавление элемента в список, вычисление пересечения, объединения и разности множеств, или получение элемента с наивысшим рейтингом в отсортированном наборе [31].

Для достижения максимальной производительности Redis работает с набором данных в оперативной памяти. В зависимости от варианта использования, Redis может периодически сохранять данные на постоянное запоминающие устройство.

Redis может быть установлен на такие операционные системы, как: Linux, Windows, macOS.

Отличие Redis от реляционных СУБД:

1. Данные хранятся в оперативной памяти. Благодаря этому Redis выигрывает в производительности у реляционных СУБД;
2. Отсутствует язык SQL;
3. Данные хранятся не в виде таблиц, а в виде строк, списков, хешей, множеств, в том числе отсортированных.

В роли чего можно использовать Redis:

1. Как хранилище пользовательских сессий;
2. Как брокер сообщений;
3. Как СУБД для небольших приложений, блогов;
4. Для кэширования данных из основного хранилища, что значительно снижает нагрузку на реляционную базу данных. В качестве инструмента для кеширования Redis будет использован при разработке программного обеспечения для кеширования программных алгоритмов в рамках текущей работы;
5. Для хранения «быстрых» данных — когда важны скорость и критичны задержки передачи (аналитика и анализ данных, финансовые и торговые сервисы).

Основные команды для управления данными в Redis:

1. HSET — сохраняет значение по ключу;
2. HGET — получение значения по ключу (для определённого поля);
3. HGETALL — получение всех пар «ключ-значение»;
4. HKEYS и HVALS — получение всех ключей и соответствующих им значений.

С помощью подобных команд можно управлять данными непосредственно из командной строки операционной системы, что используется крайне редко. Зачастую, манипуляции с данными происходят с помощью языков программирования, для которых существуют готовые библиотеки, позволяющие взаимодействовать с Redis.

Язык программирования PHP имеет встроенные средства для взаимодействия с Redis и не требует установки сторонних библиотек. В рамках выпускной квалификационной работы Redis использовался при разработке программного обеспечения для кеширования результаты работы программных алгоритмов.

### Протокола обмена сообщениями AMQP и брокера сообщений RabbitMQ

AMQP (Advanced Message Queuing Protocol) — открытый протокол для передачи сообщений между компонентами системы. Основная идея состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет маршрутизацию, возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений [15]. Протокол AMQP вводит три понятия:

1. Exchange (обменник или точка обмена) — в неё отправляются сообщения. Обменник распределяет сообщение в одну или несколько очередей. Он маршрутизирует сообщения в очередь на основе созданных связей (binding) между ним и очередью.
2. Queue (очередь) — структура данных на диске или в оперативной памяти, которая хранит ссылки на сообщения и отдает копии сообщений consumers (потребителям). Одна очередь может использоваться несколькими потребителями.
3. Binding (привязка) — правило, которое сообщает точке обмена в какую из очередей эти сообщения должны попадать. Обменник и очередь могут быть связаны несколькими привязками.

Протокол AMQP работает поверх протокола TCP/IP.

RabbitMQ — это реализация AMQP с открытым исходным кодом. Сервер написан на Erlang и поддерживает несколько клиентов, таких как: Python, Ruby, .NET, Java, JMS, C, PHP, ActionScript, XMPP, STOMP. Он используется для хранения и пересылки сообщений в распределенной системе и хорошо работает с точки зрения простоты использования, масштабируемости и высокой доступности. [RabbitMQ](https://www.rabbitmq.com/) маршрутизирует сообщения по всем базовым принципам протокола [AMQP](https://ru.wikipedia.org/wiki/AMQP). Отправитель передает сообщение брокеру, а тот доставляет его получателю. RabbitMQ реализует и дополняет протокол AMQP.

Основная идея модели обмена сообщениями в RabbitMQ заключается в том, что producer (издатель) не отправляет сообщения непосредственно в очередь. На самом деле и довольно часто издатель даже не знает, будет ли сообщение вообще доставлено в какую-либо очередь. Вместо этого издатель может отправлять сообщения только на обмен. С одной стороны, обмен получает сообщения от издателей, а с другой — отправляет их в очереди. Обмен должен точно знать, что делать с полученным сообщением. Должно ли оно быть добавлено в определенную очередь? Должно ли оно быть добавлено в несколько очередей? Или сообщение нужно игнорировать. [21].

Работу RabbitMQ можно описать следующим образом:

1. Издатель отправляет сообщение определенному обменнику;
2. Обменник, получив сообщение, маршрутизирует его в одну или несколько очередей в соответствии с правилами привязки между ним и очередью;
3. Очередь хранит ссылку на это сообщение. Само сообщение хранится в оперативной памяти или на диске;
4. Как только потребитель готов получить сообщение из очереди, сервер создает копию сообщения по ссылке и отправляет;
5. Потребитель получает сообщение и отправляет брокеру подтверждение;
6. Брокер, получив подтверждение, удаляет копию сообщения из очереди. Затем удаляет из оперативной памяти и с диска.

Для взаимодействия PHP и RabbitMQ использовалась сторонняя PHP-библиотека php-amqplib/php-amqplib. Примеры использования библиотеки php-amqplib/php-amqplib для взаимодействия с брокером сообщений RabbitMQ будут рассмотрены ниже.

RPC. Процесс RPC (remote procedure call) лежит в основе практически всех взаимодействий с ядром RabbitMQ. Например, начальные обсуждения условий клиента с RabbitMQ, демонстрирует определённый процесс RPC. Как только эта последовательность завершится, RabbitMQ будет готов принимать запросы от клиента (рис. 1).



Рисунок 1 — Схема подключения клиента к RabbitMQ

Также в спецификации AMQP и клиент, и сервер могут вызывать команды. Это означает, что клиент ожидает взаимодействие с сервером. Команды — это классы и методы.

Подключение и каналы. Для такого обмена информацией между клиентом и сервером используются каналы. Каналы создаются в рамках определенного подключения. Каждый канал изолирован от других каналов. В синхронном случае невозможно выполнять следующую команду, пока не получен ответ (рис. 2).



Рисунок 2 — Схема взаимодействия клиента к RabbitMQ

Простой пример создания подключения и канала при помощи AMQPStreamConnection можно увидеть в листинге 6.

Листинг 6 — пример создания подключения и канала при помощи AMQPStreamConnection

/\*\*

\* Метод создания подключения с брокером сообщений

\* @return AMQPChannel

\*/

private static function getChannel():AMQPChannel

{

$host = 'localhost';

$user = 'guest';

$password = 'guest';

$vhost = '/';

$port = 5672;

$connection = new AMQPStreamConnection($host, $port, $user, $password, $vhost);

$shutdown = function($connection, $channel)

{

if(!is\_null($connection)){

$connection->close();

}

if(!is\_null($channel)){

$channel->close();

}

};

register\_shutdown\_function($shutdown, $connection, $connection->channel());

return $connection->channel();

}

Открывать новое соединение для каждой операции, настоятельно не рекомендуется, поскольку это приведет к большим затратам. Каналы также должны быть постоянными, но многие ошибки протокола приводят к закрытию канала, поэтому срок службы канала может быть короче, чем у соединения.

Exchange — обменник или точка обмена. В него отправляются сообщения. Exchange распределяет сообщение в одну или несколько очередей. Он маршрутизирует сообщения в очередь на основе созданных связей (bindings) между ним и очередью.

Exchange не является Erlang-процессом. Из соображений масштабируемости exchange — это строка (ссылка на модуль с кодом, где лежит логика маршрутизации) во встроенной базе данных. Рассмотрим все виды exchange.

Direct exchange — используется, когда нужно доставить сообщение в определенные очереди. Сообщение публикуется в обменник с определенным ключом маршрутизации и попадает во все очереди, которые связаны с этим обменником аналогичным ключом маршрутизации. Ключ маршрутизации — это строка. Поиск соответствия происходит при помощи проверки строк на эквивалентность. Графическое представление потока сообщений для direct exchange можно увидеть на рисунке 3.



Рисунок 3 — Графическое представление потока сообщений в direct exchange

В rabbitmq существует понятие обменник по умолчанию. Это direct exchange без имени. Если применяется обменник по умолчанию, то сообщение будет маршрутизироваться в очередь с именем равным ключу маршрутизации сообщения.

Topic exchange – аналогично direct exchange дает возможность осуществления выборочной маршрутизации путем сравнения ключа маршрутизации. Но, в данном случае, ключ задается по шаблону. При создании шаблона используются 0 или более слов (буквы AZ и az и цифры 0-9), разделенных точкой, а также символы \* и #.

* + — может быть заменен на ровно 1 слово;
  + # — может быть заменен на 0 или более слов.

Графическое представление потока сообщений для topic exchange можно увидеть на рисунке 4.

Рисунок 4 — Графическое представление потока сообщений в topic exchange

Особенности:

* + Применение этого обменника может стать хорошим выбором для возможного будущего развития приложения, т.к. шаблоны всегда можно настроить так, чтобы сообщение публиковалось аналогично direct exchange или fanout exchange
  + Шаблоны, которые используют \* намного быстрее, чем шаблоны, которые используют #.
  + Topic exchange медленнее direct exchange

Fanout exchange – все сообщения доставляются во все очереди даже если в сообщении задан ключ маршрутизации.

Особенности:

* + RabbitMQ не работает с ключами маршрутизации и шаблонами что положительно влияет на производительность. Это самый быстрый exchange;
  + Все потребители должны иметь возможность обрабатывать все сообщения.

Графическое представление потока сообщений в Fanout exchange можно увидеть на рисунке 5.



Рисунок 5 — Графическое представление потока сообщений в fanout exchange

Рассмотрим подробнее очереди (Queue) и привязки к очередям (Bindings).

Queue (очередь) — структура данных на диске или в оперативной памяти, которая хранит ссылки на сообщения и отдает их копии потребителям (consumers). Queue представляет собой Erlang-процесс с состоянием (где могут кэшироваться и сами сообщения).

Binding (привязка) — правило, которое сообщает обменнику в какую из очередей должны попадать сообщения.

Если создание очереди происходит с установленным параметром autoDelete, то такая очередь обретает способность автоматически удалять себя. Такие очереди обычно создаются в момент подключения первого клиента и удаляются в момент, когда все клиенты отсоединились.

Если создание очереди происходит с установленным параметром exclusive, то такая очередь разрешает подключаться только одному потребителю и удаляется если закроется канал. До тех пор, пока канал не закроется, клиент может отключаться/подключаться, но только в рамках того же самого соединения. Если параметр exclusive установлен, то параметр autoDelete не имеет никакого эффекта.

Особенности:

* + при кратковременном разрыве связи мы будем терять сообщения, которые ещё не успели дойти до потребителя;
  + можно поймать феномен binding churn. Феномен возникает, когда количество операций по созданию/удалению очередей и привязок достигает очень больших значений. В кластерном режиме такой поток операций будет расползаться по всем узлам и создаст большую нагрузку. Данный процесс можно оптимизировать за счет контроля количества подписок.

Если создание очереди происходит с установленным параметром durable, то такая очередь сохраняет свое состояние и восстанавливается после перезапуска сервера/брокера. Данная очередь будет существовать до тех пор, пока не будет вызвана команду queue\_delete.

Создание очереди. Создание очереди происходит при помощи синхронного RPC запроса к серверу. Запрос осуществляется при помощи метода queue\_declare. Пример создания очереди при помощи queue\_declare можно увидеть в листинге 7.

Листинг 7. Пример создания очереди при помощи queue\_declare

// ...

queue\_declare (

queue: "my\_queue",

durable: false,

exclusive: false,

autoDelete: false,

arguments: null

);

// ...

Параметры метода queue\_declare:

* + queue — название очереди, которую мы хотим создать. Название должно быть уникальным и не может совпадать с системным именем очереди
  + durable — если true, то очередь будет сохранять свое состояние и восстанавливается после перезапуска сервера/брокера
  + exclusive — если true, то очередь будет разрешать подключаться только одному потребителю
  + autoDelete — если true, то очередь обретает способность автоматически удалять себя
  + arguments — необязательные аргументы.

Разберём подробнее параметр arguments:

* + x-message-ttl(x-message-time-to-live) — позволяет установить время истечения срока жизни сообщения в миллисекундах. Если создание очереди происходит с установленным значением аргумента x-message-ttl, то такая очередь будет автоматически исключать сообщения, у которых истек срок действия. Установка значения аргумента x-message-ttl задает максимальный возраст для всех сообщений в данной очереди. Создание такой очереди позволяет предотвратить получение устаревшей информации. Это можно использовать в системах реального времени. Если у очереди, для которой задан обменник для отклоненных сообщений установить значение аргумента x-message-ttl, то отклоненные сообщения в данной очереди начнут обладать сроком жизни;
  + x-expires — задает значение в миллисекундах по истечению которого происходит удаление очереди. Очередь может израсходовать срок своего действия только если она не имеет никаких подписчиков. Если к очереди подключены подписчики, она сможет автоматически удалиться только тогда, когда все подписчики вызовут Basic.Cancel или отсоединятся. Срок жизни очереди может завершиться только в том случае, если к ней не было запроса Basic.Get. Иначе текущее значение установки времени жизни обнуляется, и очередь больше не будет автоматически удаляться. Также нет гарантий того, насколько быстро происходит удаление очереди после истечения её срока жизни;
  + x-max-length — задает максимальное число сообщений в очереди. Если число сообщений в очереди начинает превышать максимальное число, то начинают удаляться самые старые;
  + x-max-lenght-bytes — задает максимально допустимый суммарный размер полезной нагрузки сообщений в очереди. При превышении установленного значения (возникло переполнение очереди при очередной публикации сообщения) самые старые сообщения начнут удаляться;
  + x-overflow — данный аргумент используется для настройки поведения в результате переполнения очереди. Доступны два значения: drop-head (значение по умолчанию) и reject-publish. Если выбрать drop-head, то самые старые сообщения будут удаляться. Если выбрать reject-publish, то прием сообщений будет приостановлен;
  + x-dead-letter-exchange — задает exchange, в который направляются отвергнутые сообщения, которые не поставлены повторно в очередь;
  + x-dead-letter-routing-key — задает не обязательный ключ маршрутизации для отвергнутых сообщений;
  + x-max-priority — разрешает сортировку по приоритетам в очереди с максимальным значением приоритета 255 (RabbitMQ версий 3.5.0 и выше). Число указывает максимальный приоритет, который будет поддерживать очередь. Если аргумент не установлен, очередь не будет поддерживать приоритет сообщений;
  + x-queue-mode — позволяет перевести очередь в ленивый режим. В таком режиме как можно больше сообщений будет храниться на диске. Использование оперативной памяти будет минимально. В случае, если он не установлен, очередь будет хранить сообщения в памяти, чтобы доставлять сообщения максимально быстро;
  + x-queue-master-locator — если у нас кластер, то можно задать мастер очередь;
  + x-ha-policy — используется при создании HA очередей и определяет как сообщение будет распространяться по узлам. Если установлено значение all, то сообщение будет сохраняться на всех узлах. Если установлено значение nodes, то сообщение будет сохраняться на определенных узлах кластера.

Повторный вызов queue\_declare с аналогичными параметрами вернет полезную информацию об этой очереди. Например, общее число сообщений, находящихся в ожидании в данной очереди, и общее число подписанных на неё потребителей.

После того, как очередь простаивает в течении 10 секунд, она впадает в спящий режим, что приводит к значительному сокращению памяти, необходимой для этой очереди.

Создание Binding. Создание привязки происходит при помощи синхронного RPC запроса к серверу. Запрос осуществляется при помощи метода queue\_bind, вызываемого с параметрами:

* название очереди;
* название точки обмена;
* другие параметры.

Пример создания привязки при помощи RabbitMQ.Client можно увидеть в листинге 8.

Листинг 8. Пример создания привязки при помощи queue\_bind

//...

queue\_bind (

queue: queueName,

exchange: "my\_exchange",

routingKey: "my\_key",

arguments: null

);

//...

Параметры метода queue\_bind:

* queue — имя очереди
* exchange — имя обменника
* routingKey — ключ маршрутизации
* arguments — необязательные аргументы

В рамках выпускной квалификационной работы, брокер сообщений RabbiqMQ использовался при разработке программного обеспечения для обмена сообщениями по протоколу AMQP.

## Постановка задачи

Одни из самых ресурсоёмких операций с базой данных — это CRUD операции со строками таблиц. Под аббревиатурой CRUD скрываются такие операции как:

1. Создания записей (Create);
2. Чтение записей (Read);
3. Изменение записей (Update);
4. Удаление записей (Delete).

CRUD операции соответствуют следующие HTTP-методы:

1. Создать запись — метод POST;
2. Прочитать записать — метод GET;
3. Изменить запись — метод PUT;
4. Удалить запись — метод DELETE.

Зачастую, на веб-сайтах, CRUD-операции инициируют пользователи. Например, при посещении страниц веб-сайта, на которых присутствует динамически-изменяемая информация, хранящаяся в базе данных, пользователи инициируют операцию чтения данных. На крупных веб-сайтах, особенно в интернет-магазинах, где количество товаров исчисляется десятками тысяч и тысячами активных пользователей, операция чтения данных способна вызвать серьёзную нагрузку на сервер, в результате чего скорость загрузки страниц веб-сайта будет на неудовлетворительном уровне, а поддержка веб-сервера будет требовать больших финансовых затрат. Пользователей, в свою очередь, не будет устраивать скорость работы веб-сайта, в следствии чего будет уменьшаться количество активных пользователей, а следовательно, прибыль от работы интернет-магазина будет падать.

Для снижения нагрузки на базу данных, которую создают большое количество запросов на чтение записей, можно воспользоваться кешированием запросов к базе данных. Но простых в использовании веб-библиотек, предоставляющих возможность кешировать запросы к базе данных, найдено не было. Поэтому было решено разработать веб-библиотеку, которая позволит кешировать не только запросы к базе данных, а также и кешировать работу программных алгоритмов. А храниться кеш будет в оперативной памяти сервера, что положительно скажется на скорости сохранения кеша и получения кеша.

Что касается операций создания, изменения, удаления записей. Данные операции также инициируют пользователи веб-сайтов. Подобные операции могут вызываться, например, при отправке каких-либо заявок с сайта, оформление заказа, добавление и удаление товаров из корзины интернет-магазина. Когда тысячи пользователей инициируют подобные операции в один момент времени — это отрицательным образом сказывается на нагрузке как на базу данных, так и в целом на веб-сервер. В следствии чего, как и в случае запросов на чтение записей из базы данных, скорость работы сайта падает, количество активных пользователей уменьшается, финансовые затраты на поддержку веб-сервера увеличиваются.

Для снижения нагрузки на базу данных, которую вызывают операции создания, изменения, удаления записей, подойдёт отложенная обработка этих операций путём добавления их в очередь задач и дальнейшего выполнения задач, находящихся в очереди. Готовых, простых в использовании решений, позволяющих реализовать механизм добавления задач в очередь и механизм чтения задач из очереди, не было найдено. Поэтому, было решено разработать веб-библиотеку, которая позволит добавлять задачи в очередь, забирать задачи из очереди и их исполнять. Также разрабатываемая веб-библиотека будет отличаться высокой масштабируемостью. Под масштабируемостью подразумевается лёгкое создание произвольного количества очередей, а также создание произвольного количества обработчиков очередей. Для одной очереди может быть создано несколько обработчиков, что позволит ускорить обработку задач.

В итоге, была поставлена задача по разработке веб-библиотек:

1. Веб-библиотека для кеширования работы программных алгоритмов;
2. Веб-библиотека для добавления задач в очередь и выполнение этих задач.

Веб-библиотеки должны отличаться простотой в использовании, высокой производительностью и масштабируемостью.

## Разработка веб-библиотеки для кеширования работы программных алгоритмов

При разработке веб-библиотеки для кеширования программных алгоритмов использовались следующие инструменты:

1. Язык программирования PHP;
2. Нереляционное хранилище данных Redis;
3. Система контроля версия Git;
4. IDE PhpStorm.

У языка программирования php уже есть встроенные инструменты для взаимодействия с Redis. На основе этих инструментом и будет разработана веб-библиотека.

Веб-библиотека представляет из себя PHP-класс, который предоставляет инструменты взаимодействия с кешем, хранящемся в Redis.

Опишем методы класса. Самый основной метод — это конструктор класса, в который передаются параметры, от которых зависит куда или от куда будет получен кеш. Программный код конструктора класса отражён в листинге 9.

Листинг 9 — Конструктор класса веб-библиотеки для кеширования программных алгоритмов

/\*\*

\* @param array $keyData - массив с данными, из которых нужно сделать ключ кеша

\* @param int $ttl - время жизни кеша

\* @param string $tag - теги кеша, можно использовать для удаления кеша по тегу

\*/

public function \_\_construct(array $keyData, int $ttl = 3600, string $tag = '')

{

$this->redis = new \Redis();

$this->redis->pconnect('localhost');

$keyData[] = $ttl;

if(!empty($tag))

{

$keyData[] = $tag;

}

$this->key = md5(serialize($keyData));

if(!empty($tag))

{

$this->key = "TAG\_" . $tag . ":" . $this->key;

}

$this->ttl = $ttl;

}

В конструктор передаётся три параметра:

1. $keyData;
2. $ ttl;
3. $tag.

Параметр $keyData — это массив с данными, от которых зависит идентификатор сохраняемого кеша. В дальнейшем, по этому идентификатору можно будет получить сохранённые в кеше данные. В качестве этого параметра, например, может выступать условие выборки записей из базы данных. Чтобы преобразовать параметр $keyData в идентификатор кеша, этот параметр сериализуется и хешируется с помощью алгоритма md5.

Параметр $ttl представляет из себя целочисленное значение и необходим для задания времени жизни кеша в секундах. Параметр является необязательным и по умолчанию его значение равно 3600 секунд.

Параметр $tag представляет из себя строку и необходим для поддержки механизма тегированного кеша. Одним тегом можно объединить множество закешированных данных. Затем, по тегу, можно либо получить или удалить данные, объединённые одним тегом.

Для получения закешированных данных реализован метод getCache. Программный код этого метода можно увидеть в листинге 10.

Листинг 10 — Метод для получения закешированных данных

/\*\*

\* Получение закешированных данных. Если данных нет, то вернёт false

\* @return bool|array

\*/

public function getCache()

{

if($this->isExists())

{

return unserialize(self::$redis->get($this->key));

}

return false;

}

В случае, если метод getCache не вернул результат, вызывается метод setCache, который предназначен для сохранения данных в кеш. В данный метод необходимо передать один параметр-массив, в котором должны находится сохраняемые в кеш данный. Программный код методе setCache представлен в листинге 11.

Листинг 11 — Метод для сохранения данных в кеш

/\*\*

\* @param array $data - массив с данными, которые нужно сохранить в кеш

\*/

public function setCache(array $data)

{

self::$redis->set($this->key, serialize($data), $this->ttl);

}

Для удаления кеша, объединённого одним тегом, реализован метод clearCacheByTag. В этот метод необходимо передать один параметр, являющийся тегом кеша. Программный код метода clearCacheByTag представлен в листинге 12.

Листинг 12 — Метод для очистки кеша, объединённого общим тегом

/\*\*

\* Метод очищает кеш по тегу

\* @param string $tag - тег кеша

\* @return void

\*/

public function clearCacheByTag(string $tag): void

{

self::$redis->del($this->getKeysByTag($tag));

}

Также, можно получить все закешированные данные, объединённые одним тегом. Для этого реализован метод getCacheByTag. Программная реализация этого метода представлена в листинге 13.

Листинг 13 — Метод получения данных, объединённых общим тегом

/\*\*

\* Метод очищает кеш по тегу

\* @param string $tag - тег кеша

\* @return array

\*/

public function getCacheByTag(string $tag): array

{

$arKeys = $this->getKeysByTag($tag);

$arResult = [];

foreach ($arKeys as $key)

{

if($data = self::$redis->get($key));

{

$arResult[] = $data;

}

}

return $arResult;

}

Таким образом, была разработана веб-библиотека для кеширования работы программных алгоритмов.

Данная веб-библиотека обладает следующими возможностями:

1. Сохранение произвольных данных в кеш. Кеш сохраняется в оперативной памяти. Для хранения кеша в оперативной памяти использовался инструмент Redis;
2. Получение данных из кеша по ключу;
3. Задание времени жизни кеша;
4. Задание кешу произвольного тега для получения или удаление кеша по тегу.

## Разработка веб-библиотеки для синхронного выполнения операций с базой данных

При разработке веб-библиотеки использовались следующие инструменты:

1. Язык программирования php;
2. Composer — это менеджер пакетов для php;
3. Система контроля версия Git;
4. IDE PhpStorm.

Язык программирования PHP не имеет инструментов, реализующих стандарт отказоустойчивого обмена сообщениями посредством очередей по протоколу AMQP. Для этого была использована сторонняя веб-библиотека php-amqplib, которая была установлена с помощью менеджера пакетов Composer. На основе php-amqplib и была разработана веб-библиотека для синхронного выполнения операций с базой данных.

Библиотека представляет из себя три класса:

1. Connector — класс для взаимодействием с брокером очередей (установка соединения, отправка и получение сообщений);
2. aMessageProcessing — абстрактный класс для обработки получаемых сообщений. Этот класс обеспечивает высокий уровень масштабируемости, при создании новых классов-обработчиков сообщений, за счёт наследования этого класса.

Рассмотрим основные методы класса Connector. Как было сказано ранее, данных класс обеспечивает соединение с брокером сообщений, а также предназначен для передачи сообщений и получения сообщений, с последующей его передачей в соответствующий класс-обработчик.

Метод getMappingRoutingKeyClass. Данный метод возвращает php-массив соответствий ключа маршрутизации и класса-обработчика сообщений. Ключ маршрутизации передаётся вместе с сообщением и необходим для передачи сообщения в нужный обработчик. Таким образом, при создании нового класса-обработчика, необходимо предусмотреть ключ маршрутизации для этого класса и записать это соответствие в метод getMappingRoutingKeyClass. Пример реализации этого метода можно увидеть в листинге 14.

Листинг 14 — Метод получения соответствий ключа маршрутизации и класса-обработчика сообщений

/\*\*

\* Метод возвращает массив соответствий ключей маршрутизации сообщений и классов для обработки сообщений

\* @return array

\*/

private static function getMappingRoutingKeyClass():array

{

return [

Storage::ROUTING\_KEY => Storage::class,

];

}

Метод getChannel. Данный метод предназначен для установления соединения с брокером сообщений. Внутри этого метода прописываются доступы к брокеру сообщений (хост, логин, пароль и порт). Данный метод возвращает экземпляр класса AMQPChannel, который реализован в веб-библиотеке php-amqplib. Программный код этого метода можно увидеть в листинге 15.

Листинг 15 — Метод инициализации соединения с брокером сообщений

/\*\*

\* Метод создания подключения с брокером сообщений

\* @return AMQPChannel

\*/

private static function getChannel():AMQPChannel

{

$host = 'localhost';

$user = 'guest';

$password = 'guest';

$vhost = '/';

$port = 5672;

$connection = new AMQPStreamConnection($host, $port, $user, $password, $vhost);

$shutdown = function($connection, $channel)

{

if(!is\_null($connection)){

$connection->close();

}

if(!is\_null($channel)){

$channel->close();

}

};

register\_shutdown\_function($shutdown, $connection, $connection->channel());

return $connection->channel();

}

Метод initRabbitConfig. Данный метод предназначен для воссоздания инфраструктуры в брокере очередей, а именно, создаётся обменник(exchange), очереди(queue) и очередям задаются ключи маршрутизации(binding). К одной очереди можно привязать несколько ключей маршрутизации. Данные для создания инфраструктуры берутся из классов, список которых возвращает метод getMappingRoutingKeyClass (листинг 14). Программный код метода initRabbitConfig можно увидеть в листинге 16.

Листинг 16 — Метод создания инфраструктуры в брокере сообщений

/\*\*

\* Метод создания инфраструктуры в брокере сообщений

\*/

private static function initRabbitConfig()

{

$channel = self::getChannel();

$exchange = self::getExchange();

if(!empty($exchange))

{

$channel->exchange\_declare($exchange, AMQPExchangeType::DIRECT, false, true, false);

/\*\*

\* @var aMessageProcessing $CLASS

\*/

$arRk = self::getMappingRoutingKeyClass();

foreach ($arRk as $CLASS)

{

$queue = $CLASS::getQueueName();

$routingKey = $CLASS::getRoutingKey();

if(!empty($queue) && !empty($routingKey))

{

$channel->queue\_declare($queue, false, true, false, false);

$channel->queue\_bind($queue, $exchange, $routingKey);

}

}

$channel->getConnection()->close();

$channel->close();

}

}

Метод processMessage. Данный метод отвечает за передачу сообщения соответствующему классу-обработчику сообщений. Метод требует два обязательных аргумента:

1. $routing\_key — ключ маршрутизации, для сопоставления сообщения и класса, который должен обработать сообщение;
2. $message — непосредственно само полученное сообщение.

Внутри метода определяется необходимый класс-обработчик, которому и передаётся полученное сообщение. Программный код метода processMessage можно увидеть в листинге 17.

Листинг 17 — Метод обработки сообщения

/\*\*

\* Метод для обработки сообщения

\* @param string $routing\_key

\* @param string $message

\* @return array

\*/

private static function processMessage(string $routing\_key, string $message):array

{

$arResult = ['success' => false];

/\*\*

\* @var $obj aMessageProcessing

\*/

$className = self::getClassNameByRoutingKey($routing\_key);

if(!empty($className))

{

$obj = new $className($message);

$obj->runWork();

$arResult = ['success' => $obj->getResult()->isSuccess(), 'errors' => $obj->getResult()->getErrorMessages()];

}

else

{

$arResult['errors'][] = 'Некорректный ключ маршрутизации:' . $routing\_key;

}

return $arResult;

}

Метод runWorkConsumer. Данные метод необходим для запуска обработки сообщений из очереди. В метод необходимо передать один параметр — это идентификатор очереди в брокере сообщений. Программный код метода можно увидеть в листинге 18.

Листинг 18 — Метод прослушивания очереди сообщений

/\*\*

\* Метод запускает Consumer для прослушивания очереди $queue

\* @param string $queue - название очереди, которую нужно прослушивать

\*/

public static function runWorkConsumer(string $queue)

{

self::initRabbitConfig();

/\*\* @var \PhpAmqpLib\Channel\AMQPChannel $channel \*/

$channel = self::getChannel();

$channel->basic\_consume($queue, '', false, false, false, false,

'\Igrik\Vkr\AMQP\Connector::process\_message\_callback');

$timeout = 0;

while ($channel->is\_consuming()) {

$channel->wait(null, false, $timeout);

}

}

Сам метод необходимо вызывать в скрипте, который должен непрерывно работать на веб-сервере и автоматически перезапускаться в случае его остановки.

Таким образом реализован класс Connector, который предназначен для создания соединения с брокером сообщений, а также реализует передачу сообщений в необходимый класс-обработчик.

Рассмотрим абстрактный класс aMessageProcessing. Этот класс должны наследовать все классы-обработчики сообщений. В нём реализована основная логика по обработке сообщений, что позволяет с лёгкостью масштабировать веб-библиотеку, добавляя новые классы-обработчики сообщений.

Классы-обработчики сообщений, которые наследую абстрактный класс aMessageProcessing обязательно должны описать следующие методы:

1. getRoutingKey — метод возвращающие ключ маршрутизации. Результат этого метода будет использован в классе Connector, для сопоставления классов-обработчиков и получаемых из брокера очередей сообщений;
2. getQueueName — метод возвращающий идентификатор очереди в брокере сообщений. Необходим для автоматического создания инфраструктуры в брокере сообщений с помощью метода initRabbitConfig класса Connector (листинг 18).
3. runProcessMessage — основной метод класса-обработчика сообщений, в котором должна быть реализована обработка полученного сообщения.

Работа с классами-обработчиками сообщений начинается с конструктора абстрактного класса aMessageProcessing, в который передаётся сообщение. А сам экземпляр класса-обработчика и обработка полученного сообщения инициируется в методе processMessage класса Connector (листинг 17).

Таким образом реализован абстрактный php-класс aMessageProcessing, который предназначен для создания классов-обработчиков сообщений, которые, в свою очередь, будут заниматься обработкой сообщений.

## Результат использования разработанных веб-библиотек

Разработанные веб-библиотеки были использованы в реальном проекте, который является интернет-магазином. Характеристики интернет-магазина:

1. Более 40 000 товаров;
2. Более 10 000 активных пользователей в сутки;
3. Содержит информацию о наличии и ценах товаров в 63 розничных магазинах по всех России;
4. Есть интеграция с системой учёта 1С.

После применения веб-библиотеки для кеширования программных алгоритмов, скорость загрузки страниц интернет-магазина уменьшилась с 2-3 секунд, до десятых долей секунды. А количество запросов к базе данных, при посещении страниц пользователями, было уменьшено с 2-3 тысяч, до нескольких сотен, что положительным образом сказалось на нагрузке на базу данных и на веб-сервер в целом.

Также, интернет-магазин имеет интеграцию с системой учёта 1С, из которой приходит информация о товарах, ценах, остатках по всем розничным магазинам. Данные из 1С передавались по HTTP протоколу в XML-формате, что отрицательным образом сказывалось на скорости обновления данных на сайте, ведь очередная порция данных могла быть отправлена из 1С только после того, как был получен ответ от сайта на обработку предыдущей порции данных. Для оптимизации процесса передачи данных из системы учёта 1С на сайт была использована разработанная веб-библиотека для синхронного выполнения операция с базой данных, что позволило использовать брокер сообщений в процессе получения данных из 1С сайтом.

После применения разработанных веб-библиотек удалось уменьшить нагрузку на веб-сервер на 30%. Эти данные удалось получить благодаря системы мониторинга веб-сервера Zabbix, которая позволила получить графики нагрузки на веб-сервер до использования разработанных веб-библиотек (рис. 6) и после (рис. 7).

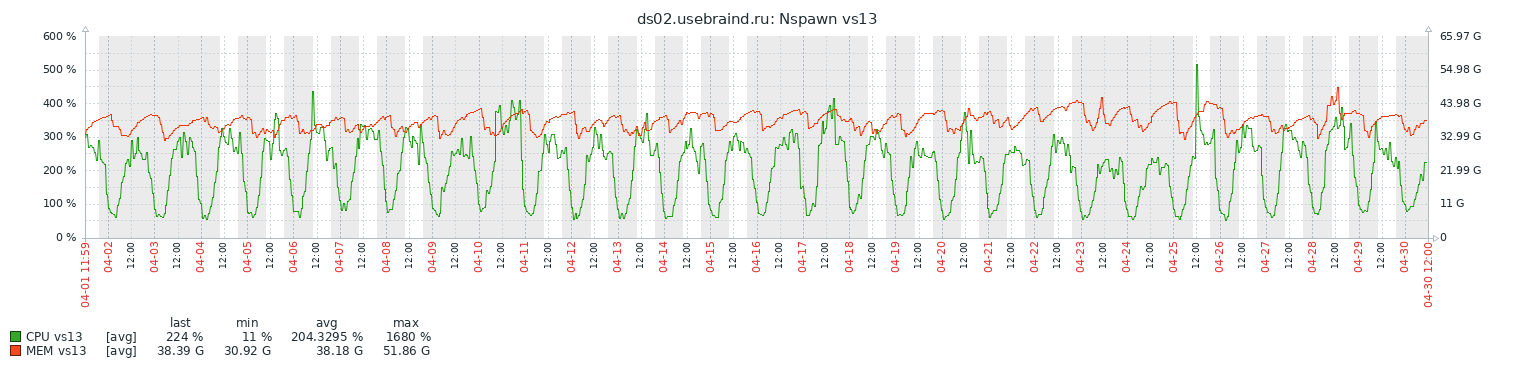


Рисунок 6 — График нагрузки на веб-сервер до использования веб-библиотек

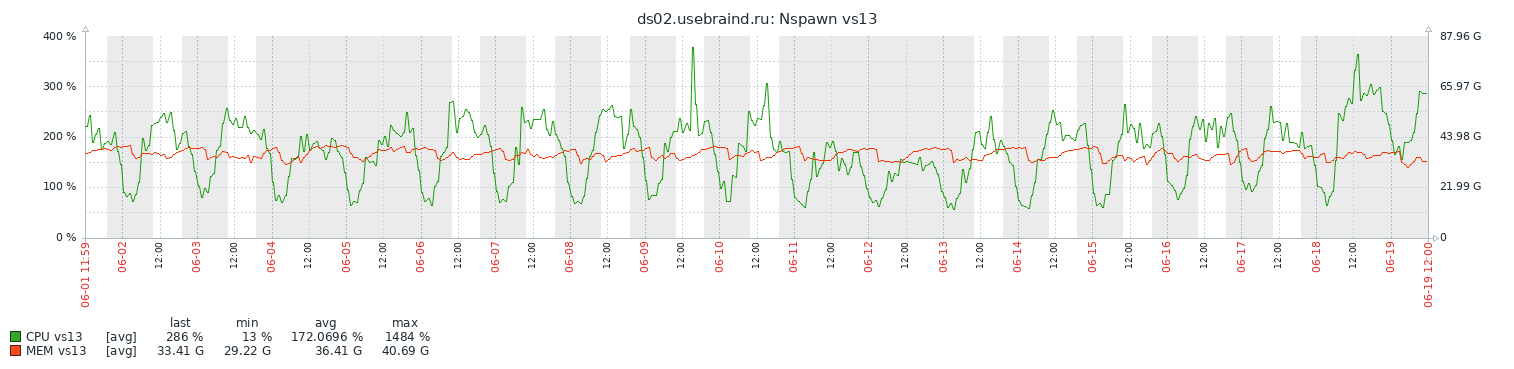


Рисунок 7 — График нагрузки на веб-сервер после использования веб-библиотек

# 

# ЗАКЛЮЧЕНИЕ

В данной работе рассматривалась проблема оптимизации взаимодействия веб-сайта с базой данных. Были изучены CRUD-операции взаимодействия с базой данных. Проанализированы методы оптимизации взаимодействия с базой данных средствами кешировани, а также синхронного выполнения операция CRUD-операций с базой данных. Результатом работы стала разработка веб-библиотек, реализующих метода оптимизации взаимодействия веб-сайта с базой данных. На языке PHP написаны:

* веб-библиотека для кеширования работы программных алгоритмов;
* веб-библиотека для синхронного выполнения CRUD-операций с базой данных.

Разработанные веб-библиотеки были использованы на реальном проекте, являющимся интернет-магазином, и позволили существенно уменьшить нагрузку на веб-сервер при использовании веб-сайта пользователями, а также удалось оптимизировать взаимодействие веб-сайта с системой учёта 1С.

# СПИСОК ЛИТЕРАТУРЫ

1. HTTP Methods [Электронный ресурс]. – Режим доступа: https://restfulapi.net/http-methods/. – Дата доступа:25.01.2022.
2. RabbitMQ tutorial - "Hello World!" — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-one-php.html. - Дата доступа:25.01.2022.
3. RabbitMQ tutorial - Publish/Subscribe — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-three-php.html. - Дата доступа:25.01.2022.
4. RabbitMQ tutorial - Reliable Publishing with Publisher Confirms — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-seven-php.html. - Дата доступа:25.01.2022.
5. RabbitMQ tutorial - Remote procedure call (RPC) — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-six-php.html. - Дата доступа:25.01.2022.
6. RabbitMQ tutorial - Routing — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-four-php.html. - Дата доступа:25.01.2022.
7. RabbitMQ tutorial - Topics — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-five-php.html. - Дата доступа:25.01.2022.
8. RabbitMQ tutorial - Work Queues — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-two-php.html. - Дата доступа:25.01.2022.
9. RabbitMQ. Часть 1. Introduction. Erlang, AMQP [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/488654/– Дата доступа: 25.01.2022.
10. Redis [Электронный ресурс]. – Режим доступа: https://evilinside.ru/redis/– Дата доступа: 25.01.2022.
11. Strong typing for event-driven microservice architecture / Н.С. Герасимов, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Компьютерные инструменты в образовании. – 2019. – № 1. – С. 43-53.
12. Анализ производительности redis в mysql для веб-кэширования / А.К. Зарипов, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Вестник магистратуры. – 2021. – № 6. – С. 52-53.
13. Архитектурные приемы при разработке программного обеспечения, зависимого от интерфейса пользователя / О.В. Игнатьева, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Инженерный вестник Дона. – 2022. – № 2. – С. 10-19.
14. Асинхронное взаимодействие. Брокеры сообщений [Электронный ресурс]. – Режим доступа: https://itnan.ru/post.php?c=1&p=534858. – Дата доступа: 25.01.2022.
15. Басов А.С. Сравнение современных СУБД / А.С. Басов // Вестник науки. – 2020. – № 7. – С. 50-54.
16. Бумажная промышленность: kafka против rabbitmq. сравнительное исследование двух отраслевых эталонных реализаций publish/subscribe / К.С. Москвичева, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Форум молодёжной науки. – 2020. – № 4. – С. 3-17.
17. Васильева К.Н. Реляционные базы данных / К.Н. Васильева, К.Н. Васильева // Colloquium-journal. – 2020. – № 2. – С. 22-23.
18. Глотов И.Н. Защищённая СУБД с сохранением порядка / И.Н. Глотов, С.В. Овсянников, В.Н. Тренькаев // Прикладная дискретная математика. – 2014. – № 7. – С. 81-82.
19. Использование СУБД Redis в качестве промежуточного хранилища данных для PostgreSQL / О.И. Рубин, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // StudNet. – 2020. – № 9. – С. 1646-1650.
20. Исследование производительности субд при работе с кластерными базами данных на основе эргономического анализа / Е.А. Елисеева, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // StudNet. – 2022. – № 4. – С. 2888-2909.
21. Ковега Д.Н. Распределенная отказоустойчивая СУБД / Д.Н. Ковега, В. А. Крищенко // Машиностроение и компьютерные технологии. – 2012. – № 3. – С. 1-7.
22. Мейер М. Теория реляционных баз данных / М. Мейер. – Москва: Мир, 1987. – 608 с.
23. Методы повышения производительности современных веб-приложений / В.Н. Гридин, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Известия Южного федерального университета. Технические науки. – 2020. – № 2. – С. 193-200.
24. Модель затрат для оптимизации аналитических запросов в гетерогенных системах / П.А. Курапов, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // International Journal of Open Information Technologies. – 2022. – № 4. – С. 61-70.
25. Обобщенная схема скрытого компактного хранения данных различных пользователей в общей открытой базе / В.А. Романьков, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Известия Иркутского государственного университета. Серия: Математика. – 2022. – № 40. – С. 63-77.
26. Особенности добавления механизма рабочего процесса в приложениях на базе микросервисной архитектуры, управляемой событиями / Р. Дандан, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Международный журнал гуманитарных и естественных наук. – 2022. – № 2. – С. 26-31.
27. Парсинг телеграм-каналов как элемент системы автоматизированного анализа информации, полученной из сети интернет / И.И. Карабак, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Прикаспийский журнал: управление и высокие технологии. – 2022. – № 1. – С. 9-17.
28. Парсинг телеграм-каналов как элемент системы автоматизированного анализа информации, полученной из сети интернет / И.И. Карабак, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Прикаспийский журнал: управление и высокие технологии. – 2022. – № 9. – С. 9-17.
29. Подсистема распределенного решения оптимизационных задач / Д.В. Заруба, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Известия Южного федерального университета. Технические науки. – 2019. – № 19. – С. 1-12.
30. Применение асинхронного обмена информацией в веб-приложениях / А.В. Скрыпников, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Международный журнал гуманитарных и естественных наук. – 2021. – № 12. – С. 105-108.
31. Работа с rabbitmq php [Электронный ресурс]. – Режим доступа: https://biznessrussia.ru/dom-i-dacha/rabota-s-rabbitmq-php.html– Дата доступа: 25.01.2022.
32. Разработка программно-аппаратного комплекса сбора и хранения данных термометрии / Ш.А. Оцоков, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Инженерный вестник Дона. – 2022. – № 2. – С. 1-9.
33. Разработка распределенной системы обмена уведомлениями на основе микросервисной архитектуры / К.Н. Цебренко, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Международный журнал гуманитарных и естественных наук. – 2021. – № 1. – С. 119-122.
34. Реализация протокола обмена данными между программными агентами в облачной инфраструктуре в географически распределенных центрах обработки данных / Н.Ю. Самохин, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Научно-технический вестник информационных технологий, механики и оптики. – 2019. – № 6. – С. 1086-1092.
35. Система мониторинга Zabbix / В.А. Чистяков, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Science Time. – 2015. – № 1. – С. 836-839.
36. Сравнение протоколов передачи данных в интернете вещей / Т.И. Курмаев, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Международный научно-исследовательский журнал. – 2022. – № 1. – С. 45-47.
37. Тенденции развития, риски и перспективы баз больших данных / В.Л. Плескач, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Colloquium-journal. – 2022. – № 1. – С. 39-42.
38. Что такое PHP? [Электронный ресурс]. – Режим доступа: https://www.php.net/manual/ru/intro-whatis.php. – Дата доступа: 25.01.2022.

# ПРИЛОЖЕНИЕ