МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему Сравнительный анализ архитектур с реляционными и объектно-реляционными базами данных

Студент М.Х. Темуршоев _____
(И.О. Фамилия) (личная подпись)

Руководитель А.В. Очеповский _____
(И.О. Фамилия) (личная подпись)

Консультанты А.В. Москалюк _____
(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н, доцент А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____»_____20_____г.

Тольятти 2018

Аннотация

Название бакалаврской работы – «Сравнительный анализ архитектур с реляционными и объектно-реляционными базами данных».

Объект работы: процесс хранения и манипулирования данными в информационных системах.

Предмет исследования: сравнительный анализ эффективности реляционных и объектно-реляционных баз данных.

Цель работы: сравнение архитектур информационных систем с реляционными и объектно-реляционными базами данных.

Для достижения необходимой цели нужно выполнить такие **задачи** как:

- провести обзор литературы по теме и выполнить теоретическое сравнение реляционных и объектно-реляционных баз данных;
- разработать и реализовать архитектуру программного обеспечения для сравнения реляционных и объектно-реляционных баз данных;
- провести вычислительный эксперимент по сравнению эффективности реляционных и объектно-реляционных базы данных.

В первой главе описывается архитектура информационных систем с базами данных, виды СУБД, применение этих архитектур в современных проектах. Рассматривается теоретическое сравнение реляционной и объектно-ориентированной модели данных.

Во второй главе описывается процесс реализация баз данных реляционным и объектно-реляционным методом.

В третьей главе представляются результаты полученных после выполнения запросов и сравниваются.

Бакалаврская работа содержит пояснительную записку объемом 46 страниц, включает в себя 18 рисунков, 7 таблиц и список использованной литературы, состоящий из 21 источника.

ABSTRACT

The title of the graduation work is Comparative Analysis of Architectures with Relational and Object-relational Databases. The basic idea of modern information technology is the concept that all data must be created and organized into databases. Information can be of any kinds, be it information about employees, their salary, customers, goods, subscription fees, addresses, etc. The more heterogeneous information accumulates, the more complex a database is required to store it. We compare the two most popular network management systems - relational databases and object-relational databases. Having made these comparisons, we get an answer to the questions: which method allows us to get access to the data stored in the database faster?

The actuality of the work is to find a solution to the problem of losing compliance. The problem is the loss of consistency between object-oriented programming and the application of the relational database as a permanent storage location. Object-oriented languages make it possible to define complex objects and facilitate the encapsulation of business logic (inheritance, polymorphism).

The aim of the work is to compare information system architectures with relational and object-relational databases. The object of the work is the process of data storage and manipulation in information systems. The subject of the research is the comparative analysis of the effectiveness of relational and object-relational databases.

The graduation work consists of introduction, three chapters and conclusion. The first chapter describes the architecture of information systems with databases, types of DBMS, the application of these architectures in modern projects. The theoretical comparison of the relational and object-oriented data model is considered. The second chapter describes the process of implementing databases using the relational and object-relational methods. The third chapter presents the results obtained after the execution of queries and compared. The work contains an explanatory note on 50 pages, including 18 figures, 6 tables, and a list of references.

Оглавление

Введение

В современном мире на предприятиях не обойтись без информационной системы, которая бы обрабатывала полученную информацию.

Основными идеями современных информационных технологий, является концепция того, что все данные должны быть созданы и организованы в базе данных. Информация может быть любых видов, будь то информация о сотрудниках, их зарплате, клиентах, товарах, абонентской плате, адресах и т.д. Чем больше скапливается разнородной информации, тем более сложная требуется база данных для её хранения. Поэтому для разработки и реализации таких баз данных управляют специальными программными комплексами – системами управления базами данных (СУБД).

Актуальность бакалаврской работы заключается в том, чтобы найти решение проблемы потери соответствия. Проблема потери соответствия состоит в том, что между объектно-ориентированным программированием и применением реляционных базы данных в качестве постоянного места хранения существует конфликт в модели данных. Объектно-ориентированные языки позволяют определять сложные объекты и способствуют инкапсуляции бизнес-логики (наследование, полиморфизм). Потеря соответствия становится очевидной при попытке отображения этих свойств объектно-ориентированного языка на реляционных баз данных, которая может только принимать двумерные структуры (таблицы) скалярных типов (number, char, date).

Объектом исследования бакалаврской работы является процесс хранения и манипулирования данными в ИС.

Предметом исследования является сравнительный анализ эффективности реляционных и объектно-реляционных баз данных.

Целю данной бакалаврской работы является сравнение архитектур информационных систем с реляционными и объектно-реляционными базами данных.

Для достижения необходимой цели нужно выполнить такие **задачи** как:

- провести обзор литературы по теме и выполнить теоретическое сравнение реляционных и объектно-реляционных баз данных;
- разработать и реализовать архитектуру программного обеспечения для сравнения реляционных и объектно-реляционных баз данных;
- провести вычислительный эксперимент по сравнению эффективности реляционных и объектно-реляционных базы данных.

Бакалаврская работа состоит из введения, трех глав и заключения.

В первой главе описывается архитектура информационных систем, отмечается роль баз данных, рассматриваются виды СУБД, применение этих архитектур в современных IT решениях. Рассматривается теоретическое сравнение реляционной и объектно-ориентированной модели данных.

Во второй главе описывается процесс реализация баз данных реляционным и объектно-реляционным методом.

В третьей главе представляются результаты полученных после выполнения запросов и сравниваются.

В заключении сделаны основные выводы и итоги по проделанной бакалаврской работе.

Глава 1 Описание архитектур информационных систем

1.1 Архитектуры информационных систем

1.1.1 Архитектурное построение современных информационных систем

В результате ускоренного темпа продвижения компьютерных и сетевых технологий стало возможным появление огромного количества информационных систем различного назначения как общедоступных в рамках глобальной сети Интернет, так и узкоспециализированных. Данные информационные системы создаются для потребности одной или нескольких корпораций или предприятий. В существующих реализациях рассматриваемых систем, информация, как правило, хранится в базе данных. Параллельный доступ к информационной системе нескольких пользователей создаётся посредством специального программного обеспечения, направленного на работу в сети. Программное обеспечение, которое реализует взаимосвязь должно находиться на обеих сторонах сети - на сервере (программа-сервер, реализующая параллельную обработку запросов к информационной системе нескольких пользователей) и на компьютере пользователя (клиентская часть, образовывающая запросы к серверу и показывающая результаты). Эффективная работа информационной системы зависит от её архитектуры.

Архитектура – это перечень существенных решений, направленных на реализацию программного обеспечения, структурных элементов и структурных интерфейсов, при помощи которых компонуется система, вместе с их поведением, устанавливаемым взаимосвязь между данными элементами, сборка элементов со временем увеличивающиеся подсистемы, так же стиль архитектуры, который наставляет эту организацию – интерфейс элементов и самих элементов, взаимосвязь и компоновку [20].

Архитектура современных информационных систем и их конструкция выделяется огромным спектром технических средств и программных средств. Значительное влияние на утверждаемые технические решения

проявляют такие факторы, как тип ЭВМ, структура базы данных, объем базы данных и т.д. В настоящий момент современные централизованные архитектуры информационных систем разделяют на следующие типы [20]:

- система, основанная на архитектуре файл сервера;
- система, основанная на архитектуре клиент сервера;
- система, основанная на многоуровневой архитектуры.

1.1.2 Архитектура файл-сервера

В архитектуре файл-сервера база данных расположена на сервере, клиент может обращаться к серверу с файловыми командами, а структура регулирования всеми информационными обеспечениями расположена на компьютере клиента.

На рисунке 1.1 представлена структура архитектуры файл-сервера.

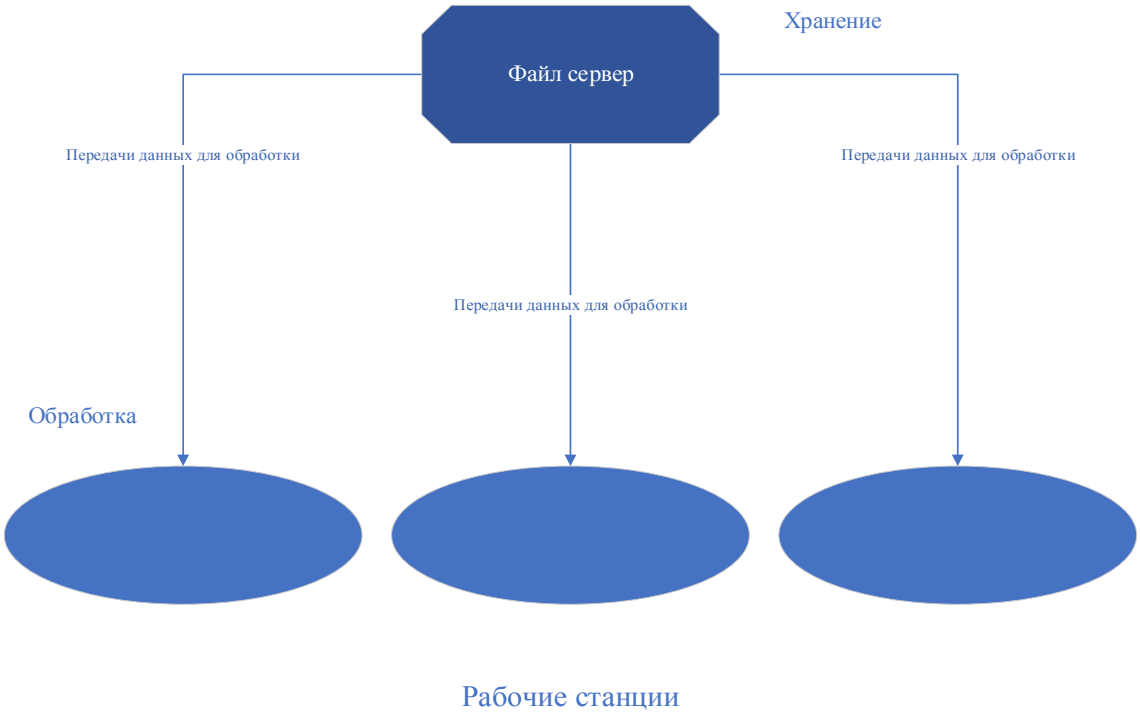

Рисунок 1.1 – Структура информационной системы с файл-сервером

Данная архитектура имеет достаточно большое количество недостатков, сдерживающих диапазон выполняемых ею задач. Простым

примером, является событие, когда данные физически расположены на компьютере, в котором находится само приложение.

Достоинствами рассматриваемой архитектуры являются:

- многопользовательский режим работы с данными;
- удобство централизованного управления доступом;
- невысокая стоимость реализации;
- повышенная скорость реализации;
- низкая стоимость обновления и изменения программного обеспечения.

Недостатками рассматриваемой архитектуры являются:

- отсутствие гарантии целостности
- невысокая производительность (зависит от производительности сети, сервера, клиента)
- некачественная возможность подключения для новых клиентов
- неустойчивость системы
- существенная загрузка локальной сети передаваемыми данными

Задача сервера состоит в хранении данных и коде программы.

Задача клиента состоит в том, чтобы исправлять данные, которые осуществляются только на стороне клиента.

Вся трудность вычислительной работы во время доступа к базе данных ложится на приложение клиента. Это происходит в результате принципа обработки информации в системах "файл-сервер". При предоставлении запроса на извлечение данных из таблицы баз данных, вся таблица копируется на клиентское место, и выборка реализовывается на клиентском месте. Локальные СУБД воспользуются так называемым "навигационным подходом", нацеленный на выполнение отдельных данных.

Архитектура файл-сервера имеет очевидные недостатки. Основной недостаток в том, что данные обрабатываются и хранятся в отдельных местах. Следовательно, их надо посылать по сети, что создаёт очень высокую

нагрузку на сеть и, в итоге резко снижается производительность приложения в случае увеличении числа параллельных работающих клиентов. Вторым значимым недостатком данной архитектуры является децентрализованное решение проблем целостности и слаженности данных и одновременного доступа к данным. Такой подход не является надежным, и поэтому она уменьшает безопасность приложения. В больших и замысловатых случаях архитектура файл-сервера становятся недостаточным, поэтому в сложных случаях ими не пользуются.

Данная архитектура является следующим шагом в прогрессии СУБД. Особенность этой архитектуры в том, что даёт возможность перенести вычислительную нагрузку на сервер базы данных (SQL -сервер) и тем самым максимально разгружает приложение клиента от вычислительной работы. Также нужно отметить значительное укрепление надежности данных - как от внесения вреда, так и от неправильного изменения данных.

1.1.3 Архитектура клиент-сервер

База данных, почтовые службы, службы печати и файловые системы являются видом ресурса для компьютерной сети. Поэтому тип сервера зависит от вида ресурса сервера. Например, сервер называется сервером базы данных при условии, что сама база данных является управляемым ресурсом [20].

Клиент-серверная архитектура рассчитана для нахождения решения проблем файл-серверных приложений посредством распределения и расположения частей приложения там, где они наиболее качественно будут функционировать. Отличительная черта клиент-серверной архитектуры является существование отдельных серверов баз данных, которые принимают запросы на языке SQL, и которые производят сортировку данных, поиск данных и агрегирование данных. Характерное свойство для серверов баз данных — это присутствие хранилища метаданных. В этом хранилище записываются ограничения целостности информации,

конструкция баз данных, серверные и форматы процедур, которые обрабатывают данные по событиям или по вызову в программе. Объектами реализации в таких приложениях, за исключением логики и диалога обработки, являются первоначально модель реляционной баз данных и относящийся с ними набор SQL-операторов для типовых запросов к базе данных.

В клиент-серверной платформе разрабатываются базы данных с огромными количествами пользователей. Используя такую платформу, для получения доступа к базам данных для группы пользователей происходит специальным компьютером — сервером. Клиент даёт поручение серверу сделать различные процессы обновления или поиска баз данных. И мощный сервер, нацеленный на работы с запросами, реализовывает их и уведомляет клиента о полученных результатах. Подобная реализация работы увеличивает качество реализация приложений при помощи использования мощности сервера, освобождает сеть, гарантирует хороший контроль целостности данных.

В клиент-серверных баз данных обнаруживается дополнительное затруднение — нужно приложение реализовывать в таком виде, чтобы это приложение использовало возможности сервера максимально, а нагрузка на сеть была бы минимальной, отправляя через неё минимум данных. Для уменьшения производительности на сеть, опрощения администрирования приложений и увеличения скорости выполнения клиентских приложений с удаленной баз данных вся логика принятия решений оформляется в виде хранимых процедур и реализовывается на сервере баз данных. Хранимые процедуры записываются на специальном алгоритмическом языке. В этих процедурах пишутся часто воспроизводимые последовательности запросов к базе данных. Текст процедуры сохраняется на сервере в откомпилированном виде. Превосходства в применении хранимых процедур очевидны:

- исключает потребность в синтаксической проверке запросов и его компилирование перед реализацией;

• исключает потребность осуществления в клиентской программе запросов, которые определены в теле хранимых процедур.

Хранимые процедуры увеличивают целостность приложений и базу данных, гарантируют актуальность совместных операций и вычислений. Повышается эффективность проведения таких процедур, а также повышает надежность (т.е. нет возможность прямого доступа к данным) [19].

Следует отметить, что перегруженность хранимых процедур прикладной логикой может нагружать сервер, вследствие этого ухудшается производительность. При реализации больших информационных систем, сервер которых параллельно может обращаться сразу к нескольким клиентам, такая проблема особенно актуальна. Из-за этого во многих случаях необходимо принять компромиссные решения: часть логики приложения помещать на стороне клиента и часть на стороне сервера. Клиент-серверные системы с таким компромиссным решениям называются системами с разделенной логикой. Описанная выше архитектура является двухуровневой и называется «толстым клиентом», которая изображена на рис 1.2.

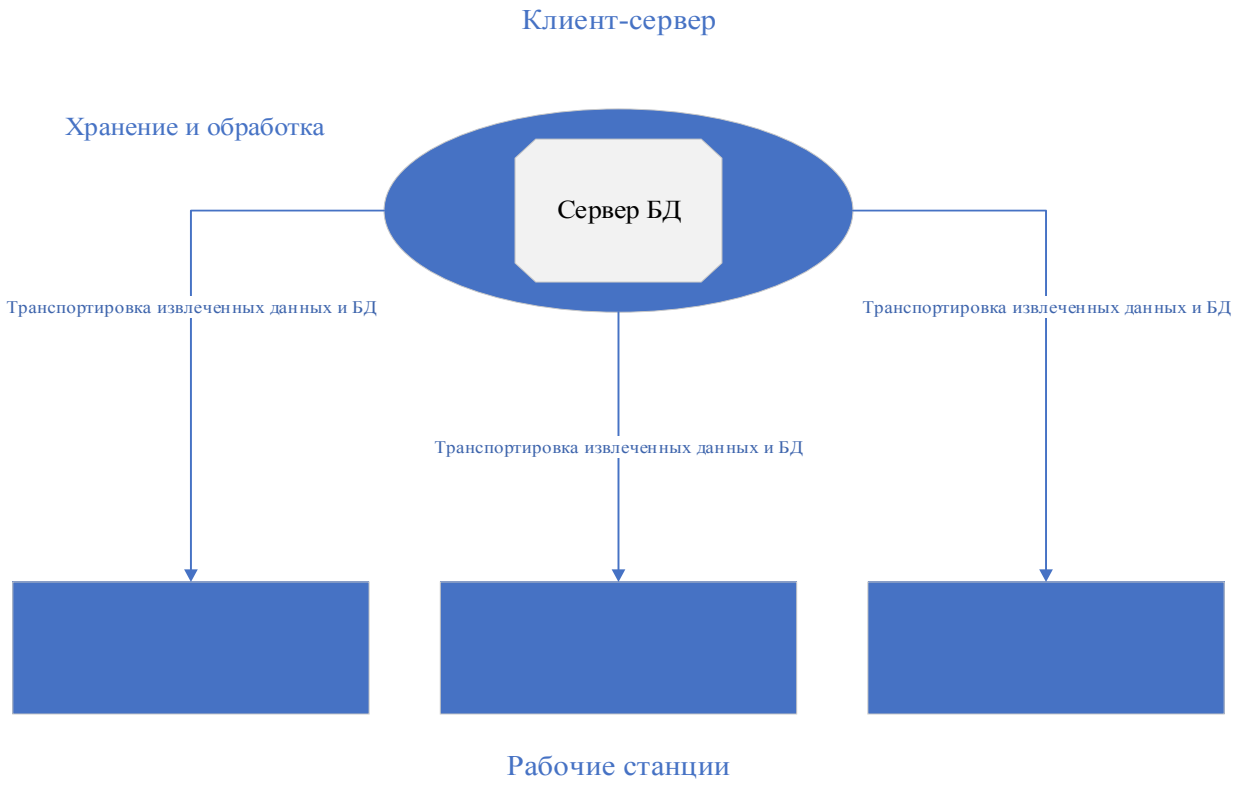

Рисунок 1.2 – Структура информационной системы с клиент-сервером

В настоящее время клиент-серверная архитектура как метод разработки приложений для рабочих групп и информационных систем корпоративного уровня, обрела признание, и стало широко распространяться. Увеличение безопасности данных в рассмотренной архитектуре связано с тем, что выполнения запросов всех клиентов реализуется единой программой, которая размещена на сервер. Сервер обеспечивает общие для всех пользователей правила использования баз данных, может управлять режимами доступа клиентов к информации, не разрешая, в частности, параллельное исправление одной записи несколькими пользователями. Сокращается сложность клиентских приложений, поскольку у них отсутствует код, который связан с контролем баз данных и разграниченным доступа к ней. Для работы со сложными информационными приложениями, в которых множества пользователей и запутанная логика, в большинстве случаев не используются двухуровневые схемы архитектуры клиент-сервера, поскольку они могут привести к проблемам. В таких случаях применяется многоуровневая архитектура.

1.1.4 Многоуровневая архитектура

Многоуровневая архитектура является совершенствованием выше рассмотренных архитектур. На последнем уровне на пользовательских компьютерах размещены приложения клиентов, назначенные для проведения функций и логики представлений, которые гарантируют вызов приложений в программном интерфейсе на среднем уровне. Сервер приложений, находящийся на среднем уровне, которая осуществляет прикладную логику, и которая осуществляет операции логику обработки данных с базой данных, т.е. на среднем уровне обеспечивается обмен данными между распределенной базой данных и пользователем. Таким образом, сервер приложений помещается в узле сети, где имеют доступ все клиенты.

Классическая форма многоуровневой архитектуры состоит из трех уровней как показано на рис 1.3.

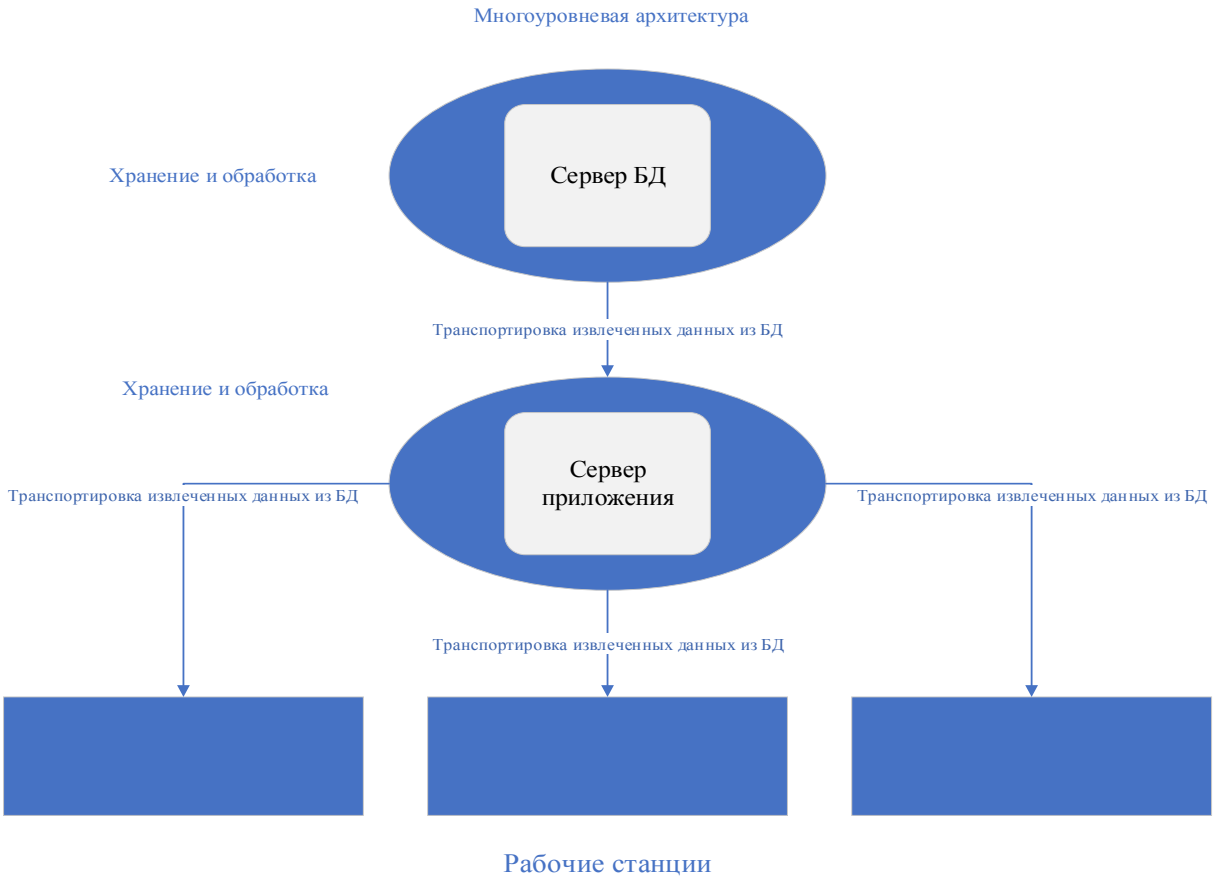

Рисунок 1.3 – Структура многоуровневой архитектуры

На третьем, верхнем уровне расположен удаленный специализированный сервер баз данных, принимающий информацию от сервера приложений. Сервер приложений отправляет информацию на данный сервер. Услуги сервера баз данных является обработка полученных данных и файловая операция.

Преимуществом трехуровневой архитектуры являются:

- разгрузка сервера базы данных от реализации части операций, которые перенесены на сервер приложений;
- разгрузка клиентского приложения за счет удаления лишних кодов, что приводит к минимизации размера;
- все клиенты имеют единое поведение;
- упрощение настройки клиентов — т.е. после изменения кода на сервере, поведение клиентских приложений автоматически изменяется.

Трехуровневая модель архитектуры ликвидирует недостатки встречающих в двухуровневой модели. Трехуровневая архитектура даёт возможность еще больше установить равновесие нагрузки на сеть. Это даёт ей преимущество над остальными архитектурами.

Количество системы с клиент-сервером возрастает с каждым годом. Поэтому необходимость в трехуровневой архитектуре становится все более очевидной [19] [20]. Рассмотренная архитектура повышает качество выполнения работы информационных систем и улучшает расположение её ресурсов.

1.1.5 Проблема соответствия при использовании баз данных в архитектурах клиент-сервер

Реляционные базы данных имеют много преимуществ, но также они имеют ряд недостатков. Одна из основных проблем для реализации приложений является проблема потеря соответствия: отличие между реляционной моделью и структурами данных других слоев архитектуры. Т.е. использование реляционной базы данных для хранения объектно-ориентированных данных приводит к данной проблеме. Также при попытке отображения свойств объектно-ориентированного языка на реляционных баз данных, становится очевидным проблема потери соответствия[16].

В результате данная проблема заставляет программиста писать программное обеспечение, которое должно уметь, как обрабатывать данные в объектно-ориентированном виде, так и уметь сохранять эти данные в реляционной форме.

Также со временем появились новые библиотеки и новые технологии программирования для объектно-реляционного отображения, они во многом упростили проблему потери соответствия, один из которых является объектно-реляционное отображение (ORM, Object-Rational Mapping).

ORM (объектно-реляционное отображение) – это технология программирования, которая связывает базы данных с концепциями объектно-

ориентированных языков программирования, при этом создавая виртуальную объектную базу данных.

ORM используется для упрощения процесса сохранения объектов в реляционную базу данных и их извлечения, при этом ORM сама заботится о преобразовании данных между двумя несовместимыми состояниями.

Большинство ORM-инструментов в значительной мере полагаются на метаданные базы данных и объектов, так что объектам ничего не нужно знать о структуре базы данных, а базе данных — ничего о том, как данные организованы в приложении.

На рисунке 1.4 представлена структура работы ORM.

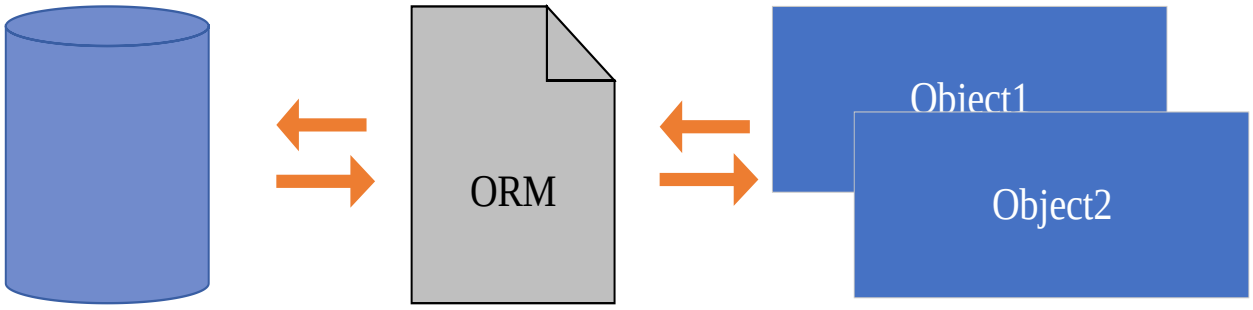

Рисунок 1.4 – Структура ORM

Рассмотрим пример использование ORM. Имеем в базе данных таблицу, изображенная на рис. 1.5.

id	db1	db2
1	Реляционная БД	Объектно-реляционная БД
2	Объектно-реляционная	Объектно-ориентированная БД
3	Объектно-ориентированная БД	Реляционная БД

Рисунок 1.5 – Таблица для использования ORM

Создадим для этой таблицы класс ORM_Database. Далее разработаем функцию для соединения с базой данных и также добавим все поля из таблицы как показано на рис 1.6.

```php
<?php
    class ORM_Database
    {
        private $conn;
        private $id;
        private $db1;
        private $db2;

        public static function SQLConnection() {
            $this->conn = mysqli_connect("localhost", "root", "", "hotel");
            if($this->conn->connect_error) die($this->conn->connect_error);
        }

        private function __construct() {

        }
```

Рисунок 1.6 – Класс ORM_Database

Далее создаём функции get() и set() для каждого поля (геттеры и сеттеры), с помощью которых мы получаем доступ к полям, и также создаём функцию selectRecord(), которая вывод данные из таблицы. Надо отметить, что для поля $id функция set() не устанавливается, потому что она автоматически добавляется при вставке новой строки. Код программы показан на рисунке 1.7.

```php
    public function getConn() {
        return $this->conn;
    }
    public function getId() {
        return $this->id;
    }

    public function setDB1($db) {
        $this->db1 = $db;
    }

    public function getDB1($db) {
        return $this->db1;
    }

    public function setDB2($db) {
        $this->db2 = $db;
    }

    public function getDB2($db) {
        return $this->db2;
    }
```

Рисунок 1.7 – Класс ORM_Database

Таким образом, для решения проблемы потери соответствия рекомендуется использовать ORM.

1.2 Теоретическое сравнение реляционной и объектно-ориентированной модели данных

Чтобы сравнивать объектно-ориентированных и реляционных база данных необходимо анализировать их особенности.

Невзирая на распространённость объектно-ориентированного программирования, в технологии создания баз данных данный метод не особо распространена. И тому есть как субъективные, так и объективные причины:

- популярность. Многие известные и широко использующие продукты реализованы реляционным методом. Эти продукты и в дальнейшем будут развиваться и поддерживаться, потому что заказчики в них вложили большие деньги, и они готовы дальше финансировать и развивать их. Напротив, под объектно-ориентированным СУБД реализовано значительно мало хороших коммерческих продуктов по сравнению с реляционным, т.е. на сегодняшний день реализовано мало мощных объектно-ориентированных СУБД;
- язык запросов и его стандартизация. В 1986 году впервые был принят язык запросов SQL-86. После принятия этого стандарта все разработчики реляционных база данных начали следовать ему. Напротив, для объектно-ориентированных база данных на сегодняшний момент не существует языка запроса. Что значительно усложняет процесс разработки объектно-ориентированных база данных. Между разработчиками даже нет единого мнения о том, что должен язык запросов делать, не говоря уже о том, каким методом он будет работать;
- математический аппарат. Эдгар Кодд для реляционных база данных создал и описал математический аппарат реляционной алгебры. Данный аппарат разъясняет, как будут реализоваться операции над отношениями в базе данных, также доказывая их оптимальность (или же из него видно, где

надо оптимизировать). Несмотря на то, что данный метод был разработан в 80-х годах, но до сих пор для неё не создан такой аппарат. Поэтому, в объектно-ориентированных базах данных не появились такие строгие термины, как отношение, декартовое произведение и т.п.;

• проблема хранения данных и методов. В реляционных базах данных вся информация написана и хранятся в таблицах. Но в объектно-ориентированных базах данных вся информация держится в объектах. В данный момент также нет единого мнения, как в данной модели создания база данных должны реализовываться хранение объектов и вообще, как разработчики должны разработать и спроектировать эти объекты. Впоследствии этого возникают такие проблемы, как хранение иерархии объектов, хранение абстрактных классов и т.п. [3].

Эти две архитектуры информационных систем во многом различаются. Основные признаки, по которым различаются объектно-ориентированная и реляционная база данных, приведены в таблице 1.1.

Таблица 1.1 Сравнение основных признаков база данных реляционными и объектно-ориентированными СУБД

Признак	Объектно-ориентированная СУБД	Реляционная СУБД
Модель данных	Объектная	Табличная
Способ чтения объектов	Данные объекта помещаются в хранилище как единое целое	Объект собирается из отдельных элементов и только потом используется
Характеристика отношений между объектами	Более совершенные средства для отображения реального мира. Естественное представление данных на разных уровнях	Все отношения принадлежат 1 уровню
Новые типы данных	Не требуется модификации ядра при добавлении нового типа данных	Необходима модификация ядра

Признак	Объектно-ориентированная СУБД	Реляционная СУБД
Оптимизации ядра СУБД	Ядро оптимизировано для операции с объектами	Ядро остается реляционным
Язык СУБД и запросы	Разновидность реализации OQL	Различные варианты SQL

Известный идеолог СУБД Мэри Луинс сказал, что объектно-ориентированная СУБД более актуально и более близка к сущностям реального мира [9]. Все данные можно использовать, не помещая их в разных таблицах. Также можно использовать большой набор предопределенных типов.

Несмотря на это можно сделать вывод, что реляционная модель, на сегодняшний день является наиболее укоренившейся и поддерживаемой.

1.3 Разработка технологии проведения эксперимента

Задачей, реализуемой в данной бакалаврской работе, является сравнение реляционной базы данных и объектно-реляционной базы данных, и сделать рекомендации на основе полученных результатов.

Пошаговая инструкция проведения нашего эксперимента:

- создание реляционной базы данных. Для этого надо установить MariaDB Server в связке с PHP 7.1. Для работы с PHP установим кроссплатформенную сборку web-сервера XAMPP. Для работы с базой данных используем phpMyAdmin;
- создание объектно-реляционной базы данных. Для этого установим PostgreSQL и графическое средство работы с ним pgAdmin. Для совместной работы PostgreSQL и PHP используем объектно-реляционное отображение ORM;
- создание таблиц в базах данных. Для создания таблицы в реляционной базе данных используем язык запроса SQL. Для объектно-

реляционной базы данных используем объектное расширение реляционных баз данных;

- реализация запроса на добавление данных в таблицу для каждой базы данных. Как уже выше было сказано, для реализации запроса реляционного метода используем PHP и язык SQL. Для объектно-реляционного метода используем объектно-реляционное отображение ORM. Местами будем использовать библиотеку RedBeanPHP, которая входит в состав ORM;
- проверка времени выполнения запроса. Для этого используем функцию microtime();
- создание таблицы и соединение их в каждой базе данных;
- исследование скорости выполнения запроса. Каждый раз меняем количество данных и количество связанных в запросе таблиц;
- создание запроса на извлечение данных. Аналогично исследуем скорость выполнения программы;
- сравнение полученных результатов. Критерий сравнения в эксперименте - это время выполнения работы и объём программы;
- анализ проведённого эксперимента. Анализируя полученные результаты, делаем выводы и рекомендации по рассмотренным архитектурам.

Глава 2 Исследование информационных систем с реляционной и объектно-реляционной БД

2.1 Разработка аппаратно-программной архитектуры для проведения эксперимента

Разработка аппаратно-программной архитектуры даёт возможность более четко понять работы, выполняемой в эксперименте.

На рисунке 2.1 представлена аппаратно-программная архитектура проведения эксперимента.

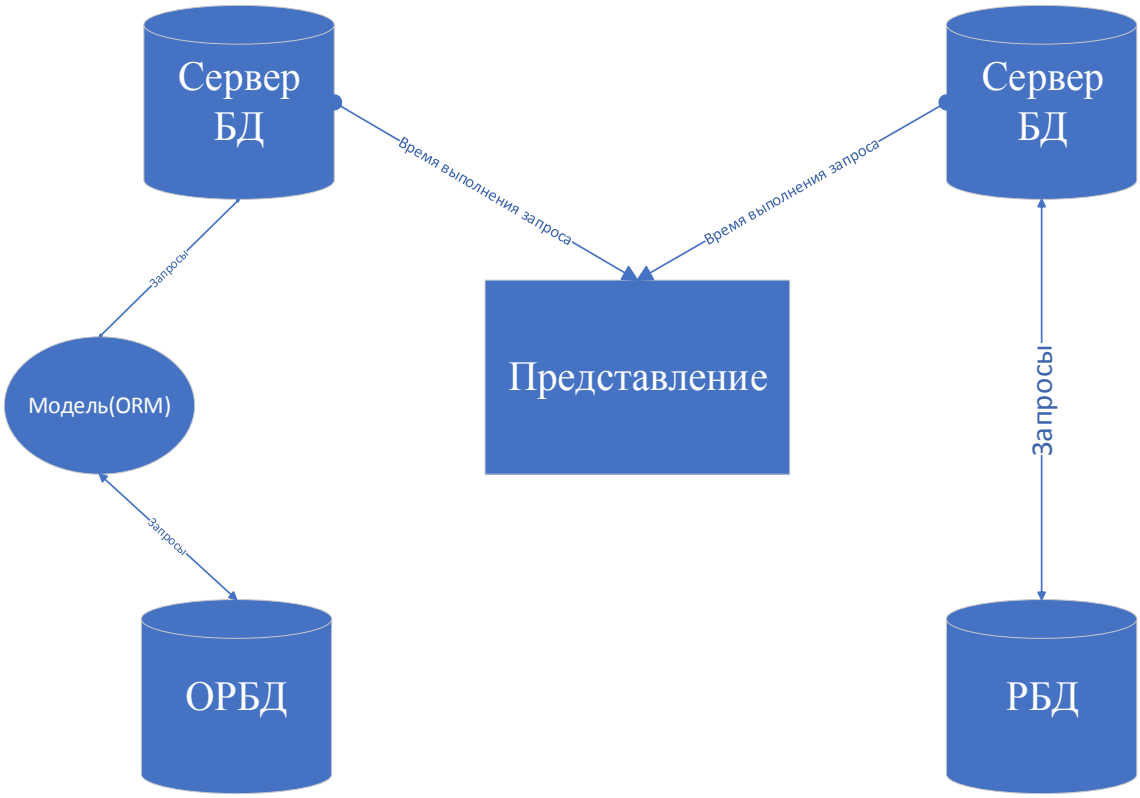

Рисунок 2.1 - Аппаратно-программная архитектура

Представление отправляет запросы на два сервера баз данных, поддерживающих реляционную и объектно-реляционную модели. Реляционная СУБД обрабатывает запрос и отправляет обработанный запрос. Сервер БД замеряет время выполнения каждого запроса.

Для объектно-реляционной модели сервер отправляет запрос на ORM систему. ORM заготавливает сценарий для общения сервера с базой данных. ORM отправляет запрос в базу для обработки. После обработки ORM

получает запрос обратно и отправляет представлению. Сервер БД аналогичным образом устанавливает время выполнения запроса. После получения времени выполнения каждого запроса, полученные результаты сравниваются.

2.2 Выбор программного обеспечения для реляционной БД

Программное обеспечение MySQL позволяет записывать, делать выборки, обновлять, и обрабатывать информацию, хранившаяся в базе данных компьютера, а также даёт возможность управлять самой базой данных. Современные компьютеры дают возможность обработки в огромных количествах данных, и поэтому регулирование базами данных играет очень важную роль в вычислениях. Для разработки данных управления воспользуемся следующими способами: можно реализовать как отдельного утилита, а можно реализовать в виде кода, который имеет непосредственный доступ к другим приложениям или же входит в их состав [1].

MySQL является системой, управляющей реляционными базами данных. В данной модели создания базы данных, вся информация хранится в таблицах, и поэтому скорость доступа к ним очень высокая. Можно сделать взаимосвязь между несколькими таблицами при помощи отношений, которая обеспечивает возможность объединение при реализации запроса данных из нескольких таблиц. SQL - это язык для создания структурированных запросов, и оно является частью системы MySQL. Данный язык является самым популярным языком для работы с базой данных [1].

Сервер MySQL предоставляет поддержку разных вычислительных машин баз данных, и также оно поддерживает многих разных клиентских программ и библиотек, средства администрирования и широкий спектр программных интерфейсов (API). Сервер MySQL также используется в виде многопоточной библиотеки, и эту библиотеку мы можем подключать

к пользовательскому приложению и в результате получаем компактный, более гибкий и быстрый в управлении продукт [13].

MariaDB Server. Майкл "Монти" человек, который является создателем и бессменным лидером проекта СУБД MySQL, в 2008 году покинул компанию под названием «Sun Micrisystems», в которой он разработал и развивал MySQL. В этом же году Монти также перестал участвовать в создании MySQL.

После всего этого он задумывался о создании нового сервера, который был бы максимально совместим с сервером MySQL, но основываясь на новом движке хранилища данных. Использование нового хранилища данных Maria – это устойчивый к сбоям клон MyISAM. По мнению многих специалистов Maria Engine является одним из лучших движков для выборки и обработки данных, и лучшей из всех существующих серверов. В классическом сервере MySQL дефолтный MyISAM является её самым слабым местом.

Основатель MySQL утверждает, что MariaDB Server – это максимально достоверная и совместимая интерфейсная копия SQL, которая используется в оригинальном сервере MySQL 5, тогда как внутри – он в значительной степени превосходит свой прототип, который дал ему жизнь и начальную кодовую основу. Вдобавок Монти заявил, что отныне MariaDB Server – это основная и единственная официальная версия MySQL. Популярность MariaDB Server растет с каждым годом. По статистике в 2017 году MariaDB Server в связке с PHP являлась самой популярной для работы с базами данных.

Таким образом, для работы с реляционными базами данных установим на хостинге MariaDB Server в связке с PHP 7.1. Для работы с базой данных настроем PhpMyAdmin.

2.3 Исследование информационных систем с реляционной базой данных

После выбора программного обеспечения для реляционной модели, приступим к реализации самой базы данных. Как уже было сказано в реляционных базах данных, хранятся только голые данные. Другими словами, в реляционных базах данных данные хранятся в виде таблицы.

Таблица – это главный объект реляционной базы данных. Определяется структура таблицы и связи между таблицами. Под структурами базы данных понимается описание наименований, типов полей, формат, проверки вводимых данных и много других характеристик. Также задаются ключи и индексы для каждой таблицы.

В нашем проекте для реализации реляционной базы данных, была использована схема данных, изображенная на рис. 2.2.

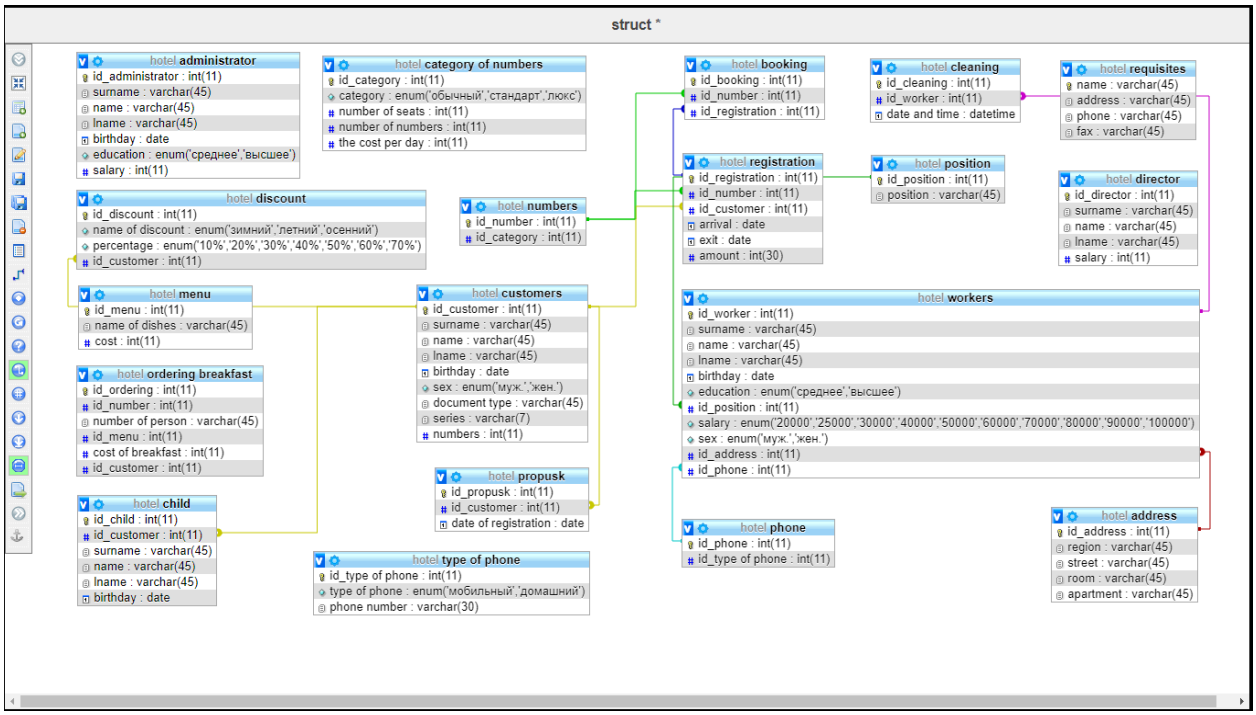

Рисунок 2.2 - Схема данных проекта

Как видно из схемы данных, в нашем проекте задействовано 19 таблиц. Многие таблицы соединены между собой. Для соединения таблиц были созданы внешние и первичные ключи.

В основном была использована связь один-ко-многим. Этот связь означает, что одной записи в таблице, содержащей сведения о сотрудниках, может соответствовать несколько записей в таблице должности или в таблице адреса. Так же мы используем связь один-к-одному. Таблица «Тип телефона» имеет связь один-к-одному с таблицей «Телефон».

Сама база данных представлена на рисунке 2.3.

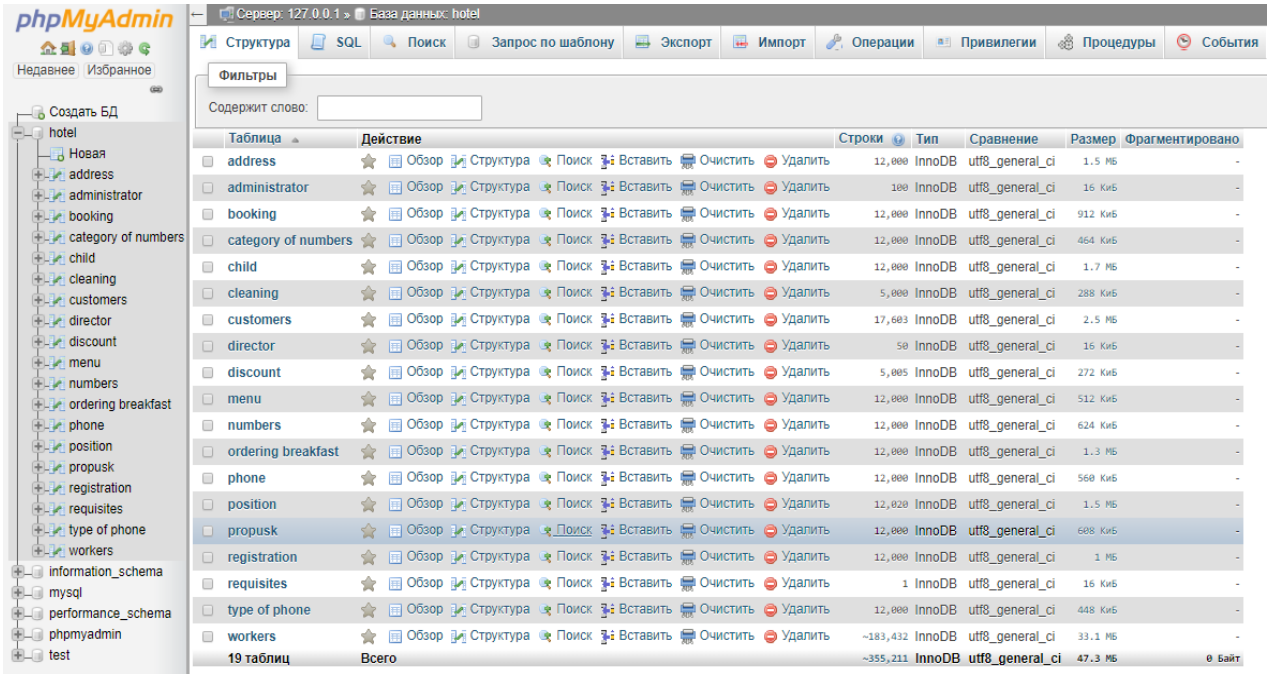

Рисунок 2.3 - База данных в PhpMyAdmin

Ключевое поле таблицы помечается специальным значком — ключик в поле выделения в левой части окна. По-другому этот ключик называется Primary Key, т.е. первичным ключом. В каждом таблице мы создали первичные ключи.

После создания связей между таблицами, заполняем их данными. Для ускорения процесса добавление данных в базу данных, была реализована запрос на добавления данных. Выполняя запрос, добавляется определенное количество данных. Запрос реализуется следующим образом:

- подключается к базе данных;
- выбирается нужная таблица;
- проверяется данные полученные из формы;

- отправляется запрос на добавления данных;
- если ошибок нет, посчитается время выполнения запроса.

Запрос на добавления данных в таблице «Сотрудники» изображён на рис. 2.4.

```php
<?php
    $conn = mysqli_connect("localhost", "root", "", "hotel");
    if($conn->connect_error) die($conn->connect_error);
    mysqli_set_charset($conn, "utf8");
    error_reporting(E_ERROR);

    $start = microtime();

    if("POST" == $_SERVER["REQUEST_METHOD"]){
        if(isset($_POST['surname']) && isset($_POST['name']) && isset($_POST['lname'])
        && isset($_POST['birthday']) && isset($_POST['education']) && isset($_POST['id_position'])
        && isset($_POST['salary']) && isset($_POST['sex']) && isset($_POST['id_address']) && isset($_POST['id_phone']) )
        {
            $query = "INSERT INTO `workers`(`surname`, `name`, `lname`, `birthday`, `education`, `id_position`, `salary`, `sex`, `id_address`, `id_phone`)
            VALUES ('" . $_POST['surname'] . "', '" . $_POST['name'] . "', '" . $_POST['lname'] . "',
            '" . $_POST['birthday'] . "', '" . $_POST['education'] . "', '" . $_POST['id_position'] . "',
            '" . $_POST['salary'] . "', '" . $_POST['sex'] . "', '" . $_POST['id_address'] . "', '" . $_POST['id_phone'] . "')";
            $result = $conn->query($query);
            if(!$result) die($conn->error);
            $end = microtime();
            $res = $end - $start;
            print "Время выполнения $res микросекунды";
        }
    }

print <<< _HTML
<form method = "POST" action = "$_SERVER[PHP_SELF]"><pre>
Фамилия         <input type = "text" name = "surname">
Имя             <input type = "text" name = "name">
Отчество        <input type = "text" name = "lname">
Дата рождения   <input type = "date" name = "birthday">
print <<< _HTML
<form method = "POST" action = "$_SERVER[PHP_SELF]"><pre>
Фамилия          <input type = "text" name = "surname">
Имя              <input type = "text" name = "name">
Отчество         <input type = "text" name = "lname">
Дата рождения    <input type = "date" name = "birthday">
Образование      <select name = "education">
<option value = "среднее">Среднее</option>
<option value = "высшее">Высшее</option>
</select>
Код должности    <input type = "number" name = "id_position" max = "5000">
Оклад            <select name = "salary">
<option value = "20000">20000</option>
<option value = "25000">25000</option>
<option value = "30000">30000</option>
<option value = "40000">40000</option>
<option value = "50000">50000</option>
<option value = "60000">60000</option>
<option value = "70000">70000</option>
<option value = "80000">80000</option>
</select>
Пол              <select name = "sex">
<option value = "муж.">Муж.</option>
<option value = "жен.">Жен.</option>
</select>
Код адреса       <input type = "number" name = "id_address" max = "5000">
Код телефона     <input type = "number" name = "id_phone" max = "5000">
                 <input type = "submit" value = "Добавить">
</pre></form>
_HTML;
?>
```

Рисунок 2.4 - Запрос на добавления данных в таблице «Сотрудники»

Также создаем ограничение для некоторых полей. Например, для поля «Дата рождения» выбираем тип Date(). Тип Date() имеет вид YYYY:mm:dd, что означает год:месяц:день. Или в поле «Пол» можно выбрать только одну данную из двух представленных. Эти ограничения позволяют корректно добавить данные в таблицах, также ускоряют и упрощают процесс добавления данных.

Как уже было сказано функция microtime() считает время выполнения каждого запроса.

После создания запроса на добавление данных, создаем запрос на вывод данных. Запрос на извлечение данных является наиболее часто используемым типом запроса в базе данных. Его результатом является динамическая таблица, которая может быть просмотрена, проанализирована. Запрос на выборку дает возможность:

- включать в результирующую таблицу поля из одной или нескольких таблиц в нужном порядке;
- выбирать записи, удовлетворяющие условиям отбора;
- осуществлять вычисления над полями базы данных.

Процесс запроса на извлечения данных происходит по следующей последовательностью:

- подключается к базе данных;
- создается запрос на извлечение данных из выбранной таблицы;
- проверяется запрос;
- если нет ошибок, происходит вывод данных;
- вместе с выводом данных посчитается время выполнения программы.

Чем больше данных, тем дольше выполняется запрос. Например, для вывода данных из таблицы «Должности», запрос выполняется почти мгновенно, а для вывода данных из таблицы «Клиенты» запрос выполняется очень долго. Это происходит потому, что в таблице «Клиенты» у нас свыше 500 000 данных, а в таблице «Должности» хранятся свыше 12 000 данных.

Результатом запроса на извлечение являются все данные, которые находятся в выбранной таблице. Все данные из таблицы выводятся на экран в виде таблицы.

Проведем эксперимент, сравнивая зависимость времени от количества записей в одной таблице. Это означает, что из одной таблицы будем

извлекать разные количества данных, вычисляя время выполнения каждого запроса. Результат работы показано в таблице 2.1.

Таблица 2.1 Результат сравнения зависимость времени от количества записей в одной таблице

№	Количества записей	Время выполнения запроса (в секундах)
1	2000	0.019154
2	3000	0.021201
3	4000	0.036584
4	5000	0.045120
5	6000	0.052230
6	7000	0.069364
7	12000	0.086483
8	200000	137.838897

Теперь проведём сравнение зависимость времени от количества таблиц при определённом количестве записей в каждой таблице.

Таблица 2.2 Результат сравнения зависимость времени от количества связанных таблиц

Количества связанных таблиц	Количества записей	Время выполнения запроса (в секундах)
2	12000	0.087416
3	12000	0.092342
4	12000	0.105451
5	12000	0.114801
6	12000	0.123413

2.4 Выбор программного обеспечения для объектно-реляционной

БД

Для выполнения эксперимента с объектно-реляционными БД используем PostgreSQL.

PostgreSQL – это свободно передаваемая объектно-реляционная система для управления базами данных (ORDBMS), в наибольшей степени продвинутая полнофункциональная из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных. PostgreSQL разработана в академической среде, и за долгую историю существования собрала вокруг себя широкое сообщество разработчиков.

Объектно-реляционная СУБД, в настоящее время известная как PostgreSQL ведёт свое происхождение от пакета POSTGRES, который был написан в департаменте Беркли, Калифорнийского Университета, под руководством Майкла Стоунбрейкера. Более чем двадцатилетняя разработка PostgreSQL сделала этот продукт одной из наиболее продвинутых СУБД с открытым исходным кодом [15].

Разработчики, используя данный метод, обретают в свое распоряжение богатый инструментарий, который позволяет реализовать приложения всякого типа:

- имеют возможность использовать различные языки для серверного программирования: встроенного PL/pgSQL (удобного тесной интеграцией с SQL), C для критичных по производительности задач, Perl, Python, Tcl, а также JavaScript, Java и других[14];
- имеют возможность, использовав программный интерфейс обращаться к СУБД из любых приложений, реализованных на любом языке, в том числе использовав стандартные интерфейсы ODBC и JDBC;
- перечень объектов в базе данных, который позволяет качественно разработать логику любой уровни сложности на стороне сервера: таблицы и индексы, локализация целостности, предоставления и материализованные предоставления, последовательности, секционирование, подзапросы и with-

запросы (в том числе рекурсивные), агрегатные и оконные функции, хранимые функции, триггеры и т.п.;

- встроенная гибкая система полнотекстового поиска, который поддерживает русский язык и всех европейских языков, дополненная качественным индексным доступом;
- возможность поддержки слабоструктурированных данных, таких как NoSQL: хранилище пар «ключ-значение» xml, hstore, json (как в текстовом, так и в качественном двоичном предоставлении jsonb);
- соединение источников данных, охватывая основные СУБД, в качестве внешних таблиц по стандарту SQL/MED с возможностью их полноценного использования, включая для записи и распределенного осуществления запросов (Foreign Data Wrappers)[14].

PostgreSQL даёт возможность работать по защищенному SSL соединению и предоставляет огромное количество методов аутентификации, в том числе аутентификация по паролю, клиентским сертификатам, а также при помощи внешних сервисов (LDAP, RADIUS, PAM, Kerberos).

pgAdmin — распространённое графическое средство, которое даёт возможность создать базу данных на PostgreSQL. Программа позволяет отображать объекты баз данных, позволяет выполнять запросы SQL, позволяет создать соединение между таблицами, также имеет другие функции, которые упрощает программирования на PostgreSQL. Тем самим, для создания объектно-реляционной базы данных было принято решение, использовать pgAdmin в связке с PHP 7.1.

2.5 Исследование информационных систем с объектно-реляционной базой данных

Как уже выше было сказано, для создания объектно-реляционных баз данных используем графическое средство pgAdmin, которая даёт возможность работать с PostgreSQL. Для отправки запроса из PHP используем ORM, а именно библиотеку RedBeanPHP. RedBeanPHP – это ещё

одна ORM-библиотека. Основное отличие от других библиотек, таких как Doctrine или Propel в том, что отсутствует необходимость ручного конфигурирования объектов. Это означает, что для работы на ней не нужно добавлять xml или ini-файлов.

Для работы с библиотекой RedBeanPHP скачиваем и подключаем её в php код. После настроим подключение к серверу. После успешного подключения к серверу, можно создать базу данных. Создаем базу данных и таблицы объектно-ориентированным методом. Для этого используем функции, объекты, методы и другие возможности данного метода.

Аналогично реляционной базе данных, в объектно-реляционной базе данных так же создаем 19 таблиц. Все таблицы имеют те же названия, которые используются в реляционных базах данных. Также все таблицы имеют ту же структуру и те же ограничения. Разница только в том, что в объектно-реляционной базе данных, таблицы были созданы объектно-ориентированным языком.

Как и в реляционной базе данных, создаем соединение между таблицами. В pgAdmin создание связей между таблицами происходит следующим образом:

- нажатием правой кнопки на таблице выбирается поле внешний ключ (Foreign key);
- задается название соединения;
- выбирается таблица и поле в выбранной таблицы, к которой хотим соединиться;
- нажатием на кнопку со специальным значком «плюсик», в правом верхнем углу, добавляется соединение;
- для добавления нового соединения, заново выбирается таблица и нажатием на плюсик, создается новая связь;
- сохраняется созданное соединение.

Также можно создать соединение объектно-ориентированным методом, используя библиотеку RedBeanPHP. Данный метод является очень простым и эффективным. Пример показано в рис. 2.5.

```php
<?php
    require 'rb/rb.php';
    R::setup('pgsql:host=localhost; dbname=diplom', 'postgres', 'unniiekd7');

    if(!R::testConnection())
    {
        exit('Нет подключения к Базе Дныых');
    }
    $hotel = R::dispense('Hotel');
    $hotel->name = "Hyatt";

    $customer = R::dispense('customer');
    $customer->count = 15000;
    $hotel->ownProductList[] = $customer;

    $id = R::store($customer);

    foreach($shop->ownProductList as $product)
    {
        print $product;
    }
?>
```

Рисунок 2.5 - Создание связи между таблицами «Гостиница» и «Клиенты»

Данным способом были соединены все таблицы. Такой способ создания связи между таблицами очень удобный. После создания связей между таблицами, заполняем их данными. Как в реляционной базе данных создаем запросы и считаем время выполнения каждого запроса. Для создания запроса в объектно-реляционной базе данных была использована объектно-ориентированное программирование.

Таким образом, были созданы три типа запроса:

- запрос на добавление данных;
- запрос на удаление данных;
- запрос на вывод данных.

Процесс создания запросов аналогично реляционному методу. Разница в том, что в объектно-реляционном методе используется классы, функции,

методы, объекты и другие возможности объектно-ориентированного программирования.

Запрос на добавление данных в таблицу «Сотрудники» изображен на рис. 2.6.

```php
<?php
    require 'rb/rb.php';
    R::setup('pgsql:host=localhost; dbname=diplom', 'postgres', 'unniiekd7');

    if(!R::testConnection())
    {
        exit('Нет подключения к Базе Дныых');
    }
    $worker = R::dispense('Hotel');
    $worker->name = 'Дмитрий';
    $worker->surname = 'Андрей';
    $worker->lname = 'Иванович';
    $worker->birthday = '1995-12-05';
    $worker->education = "высшее";
    $worker->id_position = 1201;
    $worker->salary = 50000;
    $worker->sex = "муж.";
    $worker->id_address = 201;
    $worker->id_phone = 77;

    $id = R::store($worker);

//  $worker = R::load('Work', $id);

//  R::trash($worker);

?>
```

Рисунок 2.6 - Запрос на добавление данных в объектно-реляционную базу данных

Аналогично проведем эксперимент, сравнивая зависимость времени от количества записей в одной таблице. Также будем извлекать из одной таблицы разные количества данных, и вычислим скорость выполнения запроса. Результат выполнения запроса изображено в таблице 2.3.

Таблица 2.3 Результат сравнения зависимость времени от количества записей в одной таблице

№	Количества записей	Время выполнения запроса (в секундах)
1	2000	0.025601
2	3000	0.037812

№	Количества записей	Время выполнения запроса (в секундах)
3	4000	0.044304
4	5000	0.052320
5	6000	0.065612
6	7000	0.070228
7	12000	0.083151
8	200000	36.069287

Проведём второе сравнение, зависимость времени от количества связанных таблиц при определённом количестве записей в каждой таблице. Результат показано в таблице 2.4.

Таблица 2.4 Результат сравнения зависимости времени от количества связанных таблиц

Количества связанных таблиц	Количества записей	Время выполнения запроса (в секундах)
2	12000	0.085521
3	12000	0.087901
4	12000	0.091334
5	12000	0.094231
6	12000	0.096184

2.6 Исследование информационных систем с объектно-ориентированной базой данных

Рассмотрим ещё одну модель создания базы данных – объектно-ориентированная база данных. Объектно-ориентированные базы данных хранят данные в виде объектов, тем самым, устраняя проблему потери соответствия.

Для создания объектно-ориентированных баз данных и демонстрации сохранения данных в объекте используем встраиваемую систему управления базами данных для объектов db4o (db4objects) и среду NetBeans.

Пошаговое описание создания объектно-ориентированных БД:

• установка среды NetBeans. Скачивание db4o;

• создание нового проекта в среде NetBeans. Подключение библиотеки db4o для языка программирования java. Результат показан на рисунке 2.13;

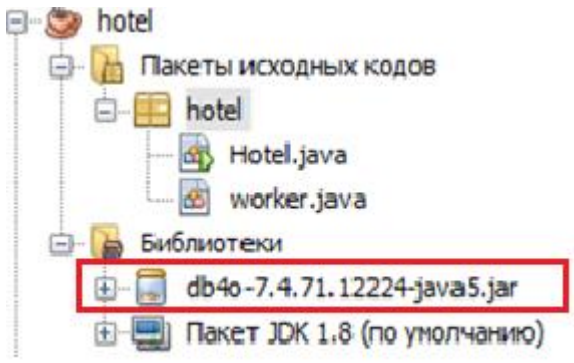

Рисунок 2.8 – Добавление библиотеки db4o

• заполняем класс «worker». Код программы показан на рис 2.14;

```
public class worker {
    private String name;
    private String surname;
    private String lastname;
    private String education;
    private int salary;
    private String sex;

    public worker(String name, String surname, String lastname, String education, int salary, String sex) {
        this.name = name;
        this.surname = surname;
        this.lastname = lastname;
        this.education = education;
        this.salary = salary;
        this.sex = sex;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public String getEducation() {
        return education;
    }

    public void setEducation(String education) {
        this.education = education;
    }
```

Рисунок 2.9 – Создание БД объектно-ориентированным методом

- используя db4o, сохраняем данные в объекте, как показано в рис 2.15

```
import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;

public class Hotel {

    public static void main(String[] args) {
        ObjectContainer db = Db4o.openFile("databaseHotel");
        try{
            addWorker(db);
            addWorker2(db);
            reviere(db);

//            deleteRecord(db);
            ObjectSet result = db.queryByExample(worker.class);
            while(result.hasNext()){
                System.out.println(result.next());
            }
        }
        finally{
            db.close();
        }
    }

    public static void addWorker(ObjectContainer db){
        worker worker1 = new worker("Ремаев", "Д.", "О.", "высшее", 800000, "муж.");
        db.store(worker1);
        System.out.println("Added: " + worker1);
    }

    public static void addWorker2(ObjectContainer db){
        worker worker2 = new worker("Михайлов", "М.", "М.", "высшее", 100000, "муж.");
        db.store(worker2);
        System.out.println("Added: " + worker2);
    }

    public static void listResult(ObjectSet result){
        while(result.hasNext()){
            System.out.println(result.next());
        }
    }

    public static void reviere(ObjectContainer db){
        ObjectSet result = db.queryByExample(worker.class);
    }
```

Рисунок 2.10 – Сохранение данных в объекте

Глава 3 Анализ результатов исследования

3.1 Сравнительный анализ результатов эксперимента

В результате бакалаврской работы были разработаны и реализованы реляционная база данных и объектно-реляционная база данных. Реляционная база данных была реализована на MariaDB Server в связке с PHP 7.1 и phpMyAdmin

Объектно-реляционная база данных была реализована на PostgreSQL в связке с ORM (библиотека RedBeans) и PHP 7.1.

Также была рассмотрена и разработана объектно-ориентированная база данных. База данных была реализована на среде NetBeans в связке с библиотекой db4objects.

В реляционной и объектно-реляционной базе данных были созданы по 19 таблиц. В каждой таблице не менее 20000 данных. В таблице «Сотрудники» и «Клиенты» более 500000 данных. В сумме в каждой базе данных хранятся более миллиона данных.

Сравнение архитектуры информационных систем с реляционными и объектно-реляционными базами данных была проведена на компьютере, имеющим следующие характеристики:

- процессор – Intel Pentium 3558U;
- базовая частота процессора – 1.70 ГГц;
- установленная память – 4 ГБ;
- тип системы – 64 разрядная операционная система.

Для сравнения баз данных в каждой базе были проведены следующие исследования:

- зависимость времени от количества записей в одной таблице;
- зависимость времени от количества таблиц при определённом количестве записей.

Сравниваем полученные результаты в таблице 3.1.

Таблица 3.1. Результат запроса зависимости времени от количества извлеченных данных из одной таблицы

№	Количество	Время выполнения реляционным методом (в секундах)	Время выполнения объектно-реляционным методом (в секундах)
1	2000	0.019154	0.025601
2	3000	0.021201	0.037812
3	4000	0.036584	0.044304
4	5000	0.045120	0.052320
5	6000	0.052230	0.065612
6	7000	0.069364	0.070228
8	12000	0.086483	0.083151
9	200000	137.838897	36.069287

На основании полученных результатов в таблице 3.1 сформируем гистограмму рисунок 3.1.

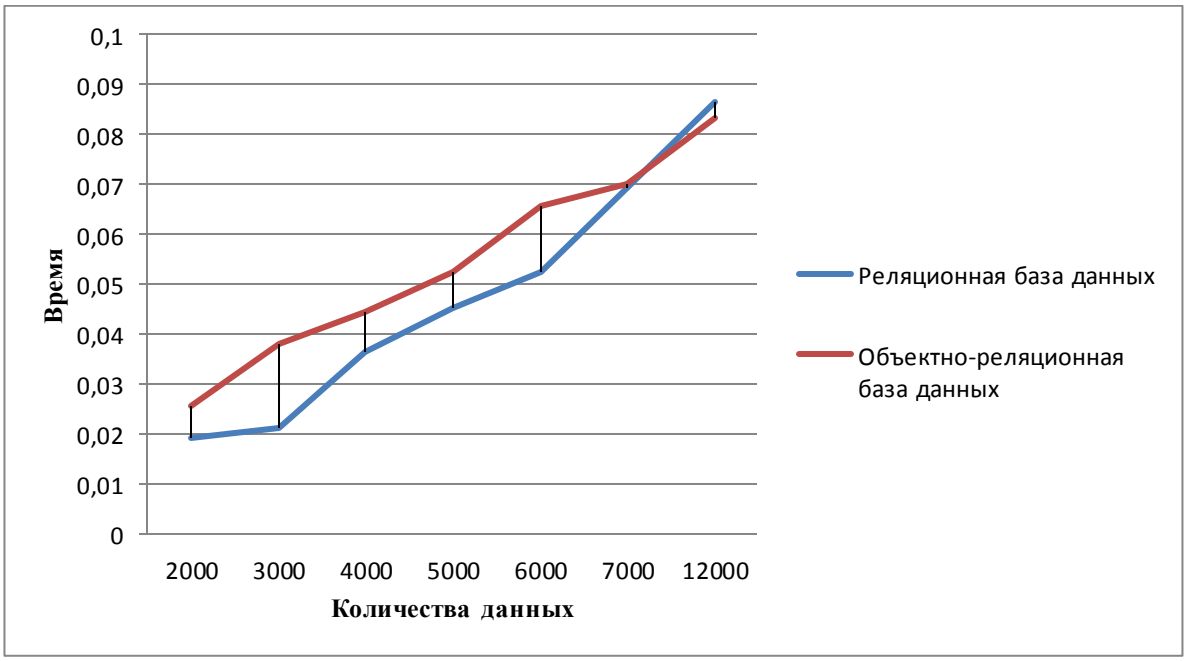

Рисунок 3.1 – Результат запроса зависимости времени от количества извлеченных данных из одной таблицы

Таблица 3.2. Результат запроса зависимости времени от количества таблиц при определённом количестве записей

Количества связанных таблиц	Количества записей	Время выполнения реляционным методом (в секундах)	Время выполнения объектно-реляционным методом (в секундах)
2	12000	0.087416	0.085521
3	12000	0.092342	0.087901
4	12000	0.105451	0.091334
5	12000	0.114801	0.094231
6	12000	0.123413	0.096184

На основании полученных результатов в таблице 3.2 сформируем гистограмму рисунок 3.2.

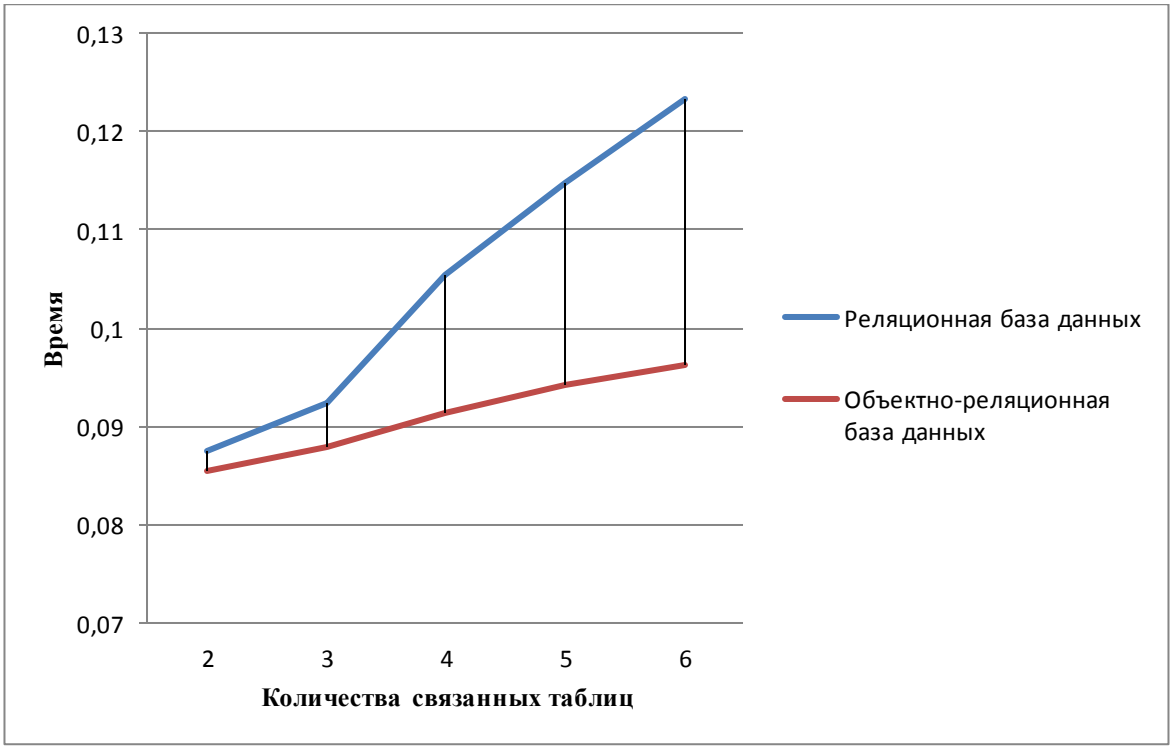

Рисунок 3.2 – Результат запроса зависимости времени от количества таблиц при определённом количестве записей

На рисунке 3.1 и 3.2 синим цветом показано результат запроса реляционной базы данных, соответственно красным цветом показано результат запроса объектно-реляционной базы данных. Слева по вертикали показано время выполнения программы, а по горизонтали в первой гистограмме показано количество данных, использующих в запросе и во второй гистограмме количество связанных таблиц.

Результаты выполнения запроса зависимости времени от количества извлеченных данных из одной таблицы реляционным методом и объектно-реляционным методом показали, что для извлечения до 7000 данных из базы, быстрее выполняется реляционная модель. Но для вывода от 7000 данных скорость выполнения реляционной базы данных, уступает объектно-реляционной базе данных. Чем больше данных, тем быстрее выполняется объектно-реляционная модель. Тем самым рекомендуется для использования запроса на вывод данных до 7000 использовать реляционную модель.

Результаты выполнения запроса зависимости времени от количества таблиц при определённом количестве записей так же показали преимущества объектно-реляционной модели. Чем больше связанных таблиц, тем больше увеличивается разница времени выполнения между рассматриваемыми архитектурами.

3.2 Выработка рекомендаций по использованию модели баз данных

После проведения сравнительных анализов архитектур информационных систем с реляционными и объектно-реляционными базами данных, получили перечень результатов, по которым сделаем выводы и рекомендации.

Начнем с самого начала, а именно с создания базы данных и таблицы. Создание базы данных и работа с ними, реляционным методом, можно сделать исключительно используя язык запроса sql.

В объектно-реляционных базе данных для создания и работы с базами можно использовать как реляционный метод, так и объектно-

ориентированный метод, что даёт большой плюс для PostgreSQL, и многие разработчики именно поэтому выбирают её.

Результаты запроса на добавление данных показали, что в реляционных базах данных для добавления данных в таблицы уходит очень много времени, что не скажешь о объектно-реляционном методе. Разница скорости выполнения запроса на добавления между двумя исследуемыми архитектурами очень большая. Поэтому для добавления данных рекомендуется использовать объектно-реляционную модель создания баз данных.

Сравнивая вышеуказанные запросы, мы получили ответ на вопрос: каким методом быстрее получаем доступ к данным?

В реляционных базах данных процесс получения доступа к небольшим данным происходит очень быстро и эффективно. Но большая проблема реляционной модели в том, что доступ к огромным количествам данных происходит очень долго. Особенно запрос на добавления данных выполняется очень медленно.

В этом объектно-ориентированная база данных, очевидно, превосходит реляционную базу данных и объектно-реляционную базу данных. Процесс получения доступа к огромному количеству данных происходит очень быстро и очень качественно.

В объектно-реляционных базах данных доступ к большим данным наравне с объектно-ориентированным методом происходит очень быстро, и превосходит по этим показателям реляционную базу данных. И так же не сильно уступает реляционному методу по скорости получения доступа к небольшим данным.

Можно сделать вывод по тем результатам, которые мы получили в результате сравнения архитектур информационных систем с реляционными базами данных, объектно-реляционными базами данных и объектно-ориентированными базами данных пришло время использования объектно-ориентированной и объектно-реляционной базы данных. Во многих аспектах

они все ещё уступают реляционному методу, но и во многом имеет преимущество. Мы можем сказать, что реляционный метод имеет математический аппарат, который объясняет, как должны выполняться основные операции над отношениями в базе данных, что нет в объектно-ориентированных и объектно-реляционных моделях. Но также мы можем сказать, что реляционная модель не обрабатывает мультимедийные данные, с которыми работает объектно-ориентированная модель и позволяет создавать структуры данных любой сложности.

Доступ к данным является самым важным аспектом работы СУБД. Анализируя полученные результаты можно смело, сказать, что на рассмотренных архитектурах объектно-реляционная база данных во многом превосходит реляционную модель.

Заключение

В данной бакалаврской работе была поставлена цель – сравнить архитектуру информационных систем с реляционными и объектно-реляционными базами данных.

В ходе выполнения бакалаврской работы была проанализирована необходимая литература, с помощью которой были определены основные требования, предъявляемые к созданию базы данных разными архитектурами.

Для выполнения поставленной цели в бакалаврской работе был проведен анализ предметной области, рассмотрены основные функции и способы реализации базы данных, реляционная модель, объектно-реляционная модель, объектно-ориентированная модель, требования, предъявляемые к разным методам создания базы данных и их реализация. Соответственно, был проведён сравнительный анализ архитектур информационных систем с реляционными и объектно-реляционными базами данных.

Задачи, выполненные в ходе данной работы, позволили в итоге создать реляционную базу данных и объектно-реляционную базу данных, сравнить их и понять, что для работы с небольшими данными (до 7000 записей) лучше и качественнее использовать реляционные базы данных, а с большими данными (от 7000 записей) эффективнее работает объектно-реляционная база данных.

Список используемой литературы

1. Аткинсон Л. MySQL. Библиотека профессионала / Л. Аткинсон; - М.: Вильямс, 2013. – 624 с

2. Ахаян Р. Эффективная работа с СУБД / Р. Ахаян, А. Горев, С. Макашарипов; СПб: Питер, 2013. – 704 с.

3. Голицына О.Л. Базы данных: Учебное пособие / О.Л. Голицына, Н.В. Максимов, И.И. Попов. - М.: Форум, 2012. - 400 с.

4. Карпова, И.П. Базы данных: Учебное пособие / И.П. Карпова. - СПб.: Питер, 2013. - 240 с.

5. Агальцов В.П. Базы данных. В 2-х т. Т. 2. Распределенные и удаленные базы данных: Учебник / В.П. Агальцов. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. - 272 с.

6. Агальцов В.П. Базы данных. В 2-х т. Т. 1. Локальные базы данных: Учебник / В.П. Агальцов. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. - 352 с.

7. Ульман Дж. Основы систем баз данных / Дж. Ульман. - М.: Финансы и статистика, 2017. - 292 с.

8. Емельянов Н.Е. Введение в СУБД ИНЕС / Н.Е. Емельянов. - М.: Наука, 2012. - 256 с.

9. Харрингтон Д. Проектирование объектно-ориентированных баз данных / Д. Харрингтон, Пер. с англ. – М.: ДМК Пресс, 2001. – 272 с.

10. Грэй П. Логика, алгебра и базы данных / П. Грэй. - М.: Машиностроение, 2017. - 368 с.

11. Глушаков С.В. Базы данных / С.В. Глушаков, Д.В. Ломотько. - М.: Харьков: Фолио, 2017. - 504 с.

12. Regina O. PostgreSQL – Up and Running / O. Regina - Москва: Огни, 2012. - 166 с.

13. Яргер Р.Дж. MySQL и mSQL: Базы данных для небольших предприятий и Интернета / Р.Дж. Яргер, Дж. Риз, Т. Кинг. - М.: СПб: Символ-Плюс, 2012. - 560 с.

14. Стоунз PostgreSQL. Основы / Стоунз, М. Ричард, Нейл - М.: СПб: Символ-Плюс, 2015. - 640 с.

15. Уорсли Дж. PostgreSQL. Для профессионалов / Дж. Уорсли, Дж. Дрейк. - М.: СПб: Питер, 2016. - 496 с.

16. Greenwald R. Professional Oracle Programming / R. Greenwald, R.Stackowiak, G. Dodge, D. Klein, B. Shapiro, C.G. Chelliah. – Indiana: Wiley Publishing, Inc., 2005. – 784 p.

17. Beaulieu A. Learning SQL, Second Edition / A. Beaulieu - California: O'Reilly Media, Inc., 2009. - 321 p.

18. Salahaldin J. Learning PostgreSQL 10, Second Edition / J. Salahaldin, A. Vannahme, A. Volkov. – Birmingham: Packt Publishing Ltd., 2015. – 433 p.

19. Elmasri R. Fundamentals of Database Systems, Sixth Edition / R. Elmasri, Sh.B. Navathe. – Boston: Addison-Wesley, 2011. – 1172 p.

20. Сергеев, С. Л. Архитектуры вычислительных систем / С.Л. Сергеев. - М.: БХВ-Петербург, 2015. - 240 с.

21. Хорошевский В. Г. Архитектура вычислительных систем / В.Г. Хорошевский. – М.: МГТУ им. Н. Э. Баумана, 2015. - 520 с.

ПРИЛОЖЕНИЕ А

Фрагмент кода реляционного метода

```
<?php
	$conn = mysqli_connect("localhost", "root", "", "hotel");
	if($conn->connect_error) die($conn->connect_error);
	mysqli_set_charset($conn, "utf8");
	error_reporting(E_ERROR);

	$start = microtime();

	if("POST" == $_SERVER["REQUEST_METHOD"]){
		if(isset($_POST['surname']) && isset($_POST['name']) && isset($_POST['lname'])
		&& isset($_POST['birthday']) && isset($_POST['sex']) && isset($_POST['document_type'])
		&& isset($_POST['series']) && isset($_POST['numbers']) && isset($_POST['data_vidachi']) )
		{
			$query = "INSERT INTO `customers`(`surname`, `name`, `lname`, `birthday`, `sex`, `document type`, `series`, `numbers`, `data_vidachi`)
			VALUES ('" . $_POST['surname'] . "', '" . $_POST['name'] . "', '" . $_POST['lname'] . "',
			'" . $_POST['birthday'] . "', '" . $_POST['sex'] . "', '" . $_POST['document_type'] . "',
			'" . $_POST['series'] . "', '" . $_POST['numbers'] . "')";
			$result = $conn->query($query);
			if(!$result) die($conn->error);
			$end = microtime();
			$res = $end - $start;
			print "Время выполнения $res микросекунды";
		}
	}

print <<< _HTML
<form method = "POST" action = "$_SERVER[PHP_SELF]"><pre>
Фамилия		<input type = "text" name = "surname">
Имя		<input type = "text" name = "name">
Отчество		<input type = "text" name = "lname">
Дата рождения	<input type = "date" name = "birthday">
Пол		<select name = "sex">
<option value = "муж.">Муж.</option>
<option value = "жен.">Жен.</option>
</select>
```

```
Тип Документа   <input type = "text" name = "document_type">
Серия           <input type = "text" name = "series">
Номер           <input type = "number" name = "numbers">
                <input type = "submit" value = "Добавить">
</pre></form>
_HTML;

?>
<?php
        $conn = mysqli_connect("localhost", "root", "", "hotel");
        if($conn->connect_error) die($conn->connect_error);
        mysqli_set_charset($conn, "utf8");
        error_reporting(E_ERROR);

        $start = microtime();

        if("POST" == $_SERVER["REQUEST_METHOD"]){
                if(isset($_POST['surname']))
                {
                        $surname = $_POST['surname'];
                        $query = "DELETE FROM`customers` WHERE surname = '" .
$surname . "'";
                        $result = $conn->query($query);
                        if(!$result)  die($conn->error);
                        $end = microtime();
                        $res = $end - $start;
                        print "Время выполнения $res микросекунды";
                }
        }

print <<< _HTML
<form method = "POST" action = "$_SERVER[PHP_SELF]">
Фамилия <input type = "text" name = "surname">
<input type = "submit" value = "DELETE RECORD">
</form>
_HTML;
?>
<?php
        $conn = mysqli_connect("localhost", "root", "", "hotel");
        if($conn->connect_error) die($conn->connect_error);
        mysqli_set_charset($conn, "utf8");

        $surname = array('Варичев', 'Зязев', 'Керимов', 'Колесов', 'Комарова',
'Панченко', 'Кузьмина', 'Сазонов', 'Сарвилина', 'Танких', 'Токарев');
```

```
	$name = array('Илья', 'Игорь', 'Максим', 'Иван', 'Александра', 'Василий', 'Даря', 'Роман', 'Дарина', 'Роман', 'Артем');
	$lname = array('Александрович', 'Максимович', 'Керимович', 'Иванович', 'Василевич', 'Сергеевич', 'Владимировна', 'Никитович', 'Василивна', 'Борисович', 'Медведович');

	$numbers = array();
	$n = '';
	for($i = 0; $i < 1000; $i++){
		for($j = 0; $j < 6; $j++){
			$n .= rand(1, 9);
		}
		$numbers[$i] = $n;
		$n = 0;
	}

	for($i = 0; $i < 1000; $i++){
		$birthday = "" . rand(1800, 1996) . "-" . rand(1, 12) . "-" . rand(1, 28);
		$sex = rand(1, 2);
		$document_type = "Паспорт";
		$series = rand(1000, 8000);
		$query = "INSERT INTO `customers`(`surname`, `name`, `lname`, `birthday`, `sex`, `document type`, `series`, `numbers`)
				VALUES ('" . $surname[rand(0, 10)] . "', '" . $name[rand(0, 10)] . "', '" . $lname[rand(0, 10)] . "', '" . $birthday . "', '" . $sex . "',
				'" . $document_type . "', '" . $series . "', '" . $numbers[$i] . "')";
		$result = $conn->query($query);
		if(!$result) die($conn->error);
	}
?>
<?php
	$conn = mysqli_connect("localhost", "root", "", "hotel");
	if($conn->connect_error) die($conn->connect_error);
	mysqli_set_charset($conn, "utf8");
	error_reporting(E_ERROR);
	ini_set('max_execution_time', 900);
	$start = microtime(true);

	$query = "SELECT * FROM `workers2`";
	$result = $conn->query($query);
	if(!$result) die($conn->error);

print <<< _HTML
```

```
<table border = "2"><tr>
<th>Код сотрудника</th>
<th>Фамилия</th>
<th>Имя</th>
<th>Отчество</th>
<th>Дата рождения</th>
<th>Образования</th>
<th>Код должности</th>
<th>Оклад</th>
<th>Пол</th>
<th>Код адреса</th>
<th>Код телефона</th>
<th>Код директора</th>
</tr>
_HTML;

        $rows = $result->num_rows;
        for($i = 0; $i < $rows; $i++)
        {
                $result->data_seek($i);
                $row = $result->fetch_array(MYSQLI_BOTH);
print <<< _END
<tr>
<td>$row[0]</td>
<td>$row[1]</td>
<td>$row[2]</td>
<td>$row[3]</td>
<td>$row[4]</td>
<td>$row[5]</td>
<td>$row[6]</td>
<td>$row[7]</td>
<td>$row[8]</td>
<td>$row[9]</td>
<td>$row[10]</td>
<td>$row[11]</td>
</tr>
_END;
        }
        echo "</table>";
        $end = microtime(true);
        $res = $end - $start;
        print "Время выполнения $res микросекунды<br>";
        print date("H:i:s", $res);
?>
```

ПРИЛОЖЕНИЕ Б

Фрагмент кода объектно-ориентированного метода

```
package hotel;
public class worker {
    private String name;
    private String surname;
    private String lastname;
    private String education;
    private int salary;
    private String sex;


    public worker(String name, String surname, String lastname, String education,
int salary, String sex) {
        this.name = name;
        this.surname = surname;
        this.lastname = lastname;
        this.education = education;
        this.salary = salary;
        this.sex = sex;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getSurname() {
        return surname;
    }
    public void setSurname(String surname) {
```

```
    this.surname = surname;
}
public String getLastname() {
    return lastname;
}
public void setLastname(String lastname) {
    this.lastname = lastname;
}
public String getEducation() {
    return education;
}
public void setEducation(String education) {
    this.education = education;
}
public int getSalary() {
    return salary;
}
public void setSalary(int salary) {
    this.salary = salary;
}
public String getSex() {
    return sex;
}
public void setSex(String sex) {
    this.sex = sex;
}

@Override
public String toString() {
    return "worker{" + "name=" + name + ", surname=" + surname +
```

```
            ", lastname=" + lastname + ", education=" + education + ", salary=" + salary + ", sex=" + sex + '}';
    }
}
package hotel;
import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;
public class Hotel {
    public static void main(String[] args) {
        ObjectContainer db = Db4o.openFile("databaseHotel");
        try{
            addWorker(db);
            addWorker2(db);
            reviere(db);
//            deleteRecord(db);
            ObjectSet result = db.queryByExample(worker.class);
            while(result.hasNext()){
                System.out.println(result.next());
            }
        }
        finally{
            db.close();
        }
    }
   public static void addWorker(ObjectContainer db){
        worker worker1 = new worker("Ремаев", "Д.", "О.", "высшее", 800000, "муж.");
        db.store(worker1);
        System.out.println("Added: " + worker1);
```

```
    }
    public static void addWorker2(ObjectContainer db){
        worker worker2 = new worker("Михайлов", "М.", "М.", "высшее", 100000,
"муж.");
        db.store(worker2);
        System.out.println("Added: " + worker2);
    }
    public static void listResult(ObjectSet result){
        while(result.hasNext()){
            System.out.println(result.next());
        }
    }
    public static void reviere(ObjectContainer  db){
        ObjectSet result = db.queryByExample(worker.class);
    }
        public static void deleteRecord(ObjectContainer db){
        ObjectSet result = db.queryByExample(new  worker("Михайлов", "", "", "",
0, ""));
        worker found = (worker)result.next();
        db.delete(found);
        reviere(db);
    }
}
```

ПРИЛОЖЕНИЕ С

Фрагмент кода объектно-реляционного метода

```
<?php
        require 'rb/rb.php';
        if(!R::testConnection())
        {
                exit('Нет подключения к Базе Днных');
        }
        $worker = R::dispense('Foot');
        $worker->name = 'Дмитрий';
        $worker->surname = 'Андрей';
        $worker->lname = 'Иванович';
        $worker->birthday = '1995-12-05';
        $worker->education = "высшее";
        $worker->id_position = 1201;
        $worker->salary = 50000;
        $worker->sex = "муж.";
        $worker->id_address = 201;
        $worker->id_phone = 77;
        $id = R::store($worker);
        $worker = R::load('Work', $id);
//      R::trash($post);
?>
<?php
   require 'rb/rb.php';
        R::setup('pgsql:host=localhost; dbname=diplom', 'postgres','unniiekd7');

        if(!R::testConnection())
        {
                exit('Нет подключения к Базе Днных');
```

```
        }
        $hotel = R::dispense('Hotel');
        $hotel->name  = "Hyatt";

        $customer = R::dispense('customer');
        $customer->count = 15000;
        $hotel->ownProductList[] = $customer;
        $id  = R::store($customer);
        foreach($shop->ownProductList as $product)
        {
                print $product;
        }
?>
```