**Воеводин И.А.** «Создание информационной системы с web-интерфейсом на основе СУБД MongoDB» / ПГГПУ, Пермь, 2017. – 60с. + Прил. [Электр.ресурс]. Настоящая работа посвящена определению методов и технологий позволяющих разработать web-интерфейс, который позволит совершать основные операции с базой данных на своем сайте пользователем. Данный проект выполнен на языке разметки HTML с помощью языка программирования PHP. Подготовлены инструктивные материалы и методические рекомендации по разработке web-интерфейса. Данный проект может быть использован непрофессиональными разработчиками, при создании собственного информационного ресурса, где необходим web-интерфейс для работы с базой данных MongoDB.

**Оглавление**

## Введение

Современные информационные технологии являются неотъемлемой частью нашей жизни. Использование смартфонов, компьютеров и, конечно, Интернета стало обыденностью и ежедневной потребностью во многих сферах деятельности человека.

Сеть Интернет открывает перед человечеством доступ к неограниченным знаниям, и используя его в обучении можно достичь больших результатов. Поэтому создание сайтов с образовательным контентом имеет значимую роль в развитии всемирной паутины.

Крупные Интернет-порталы используют такой мощный инструмент, как системы управления базами данных – СУБД. Они позволяют обеспечить удобство хранения контента в базах данных, осуществлять к ней контролируемый доступ и повышают скорость загрузки страницы, которая также немаловажна для пользователей.

Научная новизна работы состоит в изучении и обосновании необходимых методов и технологий для разработки web-интерфейса, позволяющего совершать основные операции с базой данных MongoDB на своем сайте непрофессиональными разработчиками.

Практическая значимость заключается в выработке рекомендаций для создания web-интерфейса, позволяющего работать с базами данных на сайте, и демонстрации web-интерфейса на примере конкретного сайта и базы данных (БД).

*Актуальность* дипломной работы заключается в широком использовании баз данных на web-ресурсах в сети Интернет, с возможностью работы с ними прямо на сайте. Возникает вопрос, как подключить базу данных к сайту и создать web-интерфейс для работы с подключенной БД. Это особенно актуально, потому что СУБД MongoDB является относительно новым программным продуктом и количество литературы по ней явно недостаточно.

*Объект исследования:* web-интерфейс для работы с базой данных.

*Предмет исследования:* определение методов и технологий, позволяющих организовать доступ к базе данных с сайта и разработатьweb-интерфейс для ее редактирования.

*Цель работы:*проектирование и разработка общеобразовательного сайта с web-интерфейсом на основе СУБДMongoDB.

В соответствии с поставленной целью необходимо решить следующие задачи*:*

1. Выполнить обзор современных СУБД.
2. Сравнить их возможности, выявить достоинства и недостатки.
3. Обосновать выбор СУБДMongoDB.
4. Разработать общеобразовательный сайт и создать базу данных, для хранения его контента.
5. Создать web-интерфейс для работы с базой данных на сайте.
6. Подготовить руководство пользователя.
7. Опубликовать сайт на хостинге.

# ГЛАВА 1. ВОЗМОЖНОСТИ СОВРЕМЕННЫХ СУБД ДЛЯ ОРГАНИЗАЦИИ ДОСТУПА К ДАННЫМ ПО СЕТИ

## 1.1. СУБД. Виды СУБД

Под системой управления базами данных (СУБД) понимается совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Обычно современная СУБД содержит следующие компоненты:

- ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;
- процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных;
- подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД;
- сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы [5].

Классификация систем управления базами данных:

- по модели данных;
- по степени распределенности;
- по способу доступа к БД.

По модели данных различают следующие БД.

- *Иерархические:* используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более

низкого уровня. Иерархической базой данных является файловая система, состоящая из корневого каталога, в котором имеется иерархия подкаталогов и файлов.

- *Сетевые*: базы данных подобны иерархическим, за исключением того, что в сетевой модели каждый узел может быть связан с любым другим узлом.
- *Реляционные:* используют организацию данных в виде двумерных таблиц. Каждая такая таблица, называемая реляционной таблицей, или отношением, представляет собой двумерный массив и обладает следующими свойствами:
    - все столбцы в таблице однородные, т.е. все элементы в одном столбце имеют одинаковый тип и максимально допустимый размер;
    - каждый столбец имеет уникальное имя;
    - одинаковые строки в таблице отсутствуют;
    - порядок следования строк и столбцов в таблице не имеет значения.

  Основными структурными элементами реляционной таблицы являются поле и запись. Поле (столбец реляционной таблицы) – элементарная единица логической организации данных, которая соответствует конкретному атрибуту информационного объекта. Запись (строка реляционной таблицы) – совокупность логически связанных полей, соответствующая конкретному экземпляру информационного объекта.
- *Объектно-ориентированные:* управляют базами данных, в которых данные моделируются в виде объектов, их атрибутов, методов и классов. Этот вид СУБД позволяет работать с объектами баз данных так же, как с объектами в объектно-ориентированных языках программирования. ООСУБД расширяет языки программирования, прозрачно вводя долговременные данные, управление параллелизмом,

восстановление данных, ассоциированные запросы и другие возможности.

- *Объектно-реляционные:* этот тип СУБД позволяет через расширенные структуры баз данных и язык запросов использовать возможности объектно-ориентированного подхода: объекты, классы и наследование. Зачастую все те СУБД, которые называются реляционными, являются по факту объектно-реляционными.

По степени распределенности:

- локальные СУБД (все части локальной СУБД размещаются на одном компьютере);
- распределённые СУБД (части СУБД могут размещаться на двух и более компьютерах).

По способу доступа к БД:

- *Файл-серверные:* в файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок. Преимуществом этой архитектуры является низкая нагрузка на процессор файлового сервера.

  Недостатки: потенциально высокая загрузка локальной сети; затруднённость или невозможность централизованного управления; затруднённость или невозможность обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях, которые используют функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД.

  На данный момент файл-серверная технология считается устаревшей, а её использование в крупных информационных системах – недостатком.

- *Клиент-серверные:* СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно. Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу. Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность.
- *Встраиваемые:* СУБД, которые могут поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемые СУБД предназначены для локального хранения данных своего приложения и не рассчитаны на коллективное использование в сети. Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить черезязык структурированных запросов (SQL), либо через специальные программные интерфейсы [Там же].

Далее представлено две стратегии работы с внешней памятью.

- *СУБД с непосредственной записью* – это СУБД, в которых все измененные блоки данных незамедлительно записываются во внешнюю память при поступлении сигнала подтверждения любой команды. Такая стратегия используется только при высокой эффективности внешней памяти.
- *СУБД с отложенной записью* – это СУБД, в которых изменения аккумулируются в буферах внешней памяти до наступления удобного момента для записи в память [5].

Такая стратегия позволяет избежать частого обмена с внешней памятью и значительно увеличить эффективность работы СУБД.

## 1.2. СУБД MongoDB и возможные аналоги с обоснованием выбора MongoDB

MongoDB(от англ. *humongous –огромный*) – документо-ориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++.

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, ее легче использовать и она обладает лучшей масштабируемостью (поддерживает очень большие базы данных с очень высокой частотой запросов при очень низкой задержке). Масштабируемость в MongoDB достигается за счёт разделения документов из коллекции по узлам наосновании выбранного ключа.

Но, даже учитывая все недостатки традиционных баз данных и достоинства MongoDB, важно понимать, что задачи и методы их решения бывают разные. В какой-то ситуации MongoDB действительно улучшит производительность вашего приложения, например, если надо хранить сложные по структуре данные. В другой же ситуации лучше будет использовать традиционные реляционные базы данных. Кроме того, можно использовать смешенный подход: хранить один тип данных в MongoDB, а другой тип данных - в традиционных БД [10].

Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Функциональность MongoDB позволяет расположить несколько баз данных на нескольких

физических серверах, и эти базы данных смогут легко обмениваться данными и сохранять целостность.

Одним из популярных стандартов обмена данными и их хранения является JSON (Java Script Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который называется BSON (БиСон) или сокращение от binary JSON.

BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью.

MongoDB написана на C++, поэтому ее легко портировать на самые разные платформы. MongoDB может быть развернута на платформах Windows, Linux, MacOS, Solaris. Можно также загрузить исходный код и самому скомпилировать MongoDB, но рекомендуется использовать библиотеки с официального сайта [5].

Если реляционные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк, документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений. Ключ представляет простую метку, с которым ассоциирован определенный кусок данных. В отличие от SQL в MongoDB не нужно хранить пустые поля, которые существуют в таблице, они просто не указываются при заполнении данных (рис. 1).

| SQL | имя:Том | адрес:25 октября | **телефон:" "** |
|---|---|---|---|
| | имя:Майк | **адрес:" "** | телефон:8-952-666-22-22 |

| MongoDB | имя:Том | адрес:25октября |
|---|---|---|
| | имя:Майк | телефон:8-952-666-22-22 |

Рис. 1. Пример хранения данных в SQL и MongoDB

Однако при всех различиях есть одна особенность, которая сближает MongoDB и реляционные базы данных. В реляционных СУБД встречается такое понятие как первичный ключ. Это понятие описывает некий столбец, который имеет уникальные значения. В MongoDB для каждого документа имеется уникальный идентификатор, который называется _id. И если явным образом не указать его значение, то MongoDB автоматически сгенерирует для него значение.

Каждому ключу сопоставляется определенное значение. Но здесь также надо учитывать одну особенность: если в реляционных базах есть четко очерченная структура, где есть поля, и если какое-то поле не имеет значение, ему (в зависимости от настроек конкретной БД) можно присвоить значение NULL. В MongoDB все иначе. Если какому-то ключу не сопоставлено значение, то этот ключ просто опускается в документе и не употребляется.

Если в традиционном мире SQL есть таблицы, то в мире MongoDB есть коллекции. И если в реляционных БД таблицы хранят однотипные жестко структурированные объекты, то в коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств. [21]

Система хранения данных в MongoDB представляет набор реплик. Набор реплик – это группа процессов, которые поддерживают один и тот же набор данных. Наборы реплик обеспечивают избыточность и высокую доступность и являются основой для всех производственных развертываний. В этом наборе есть основной узел, а также может быть набор вторичных узлов. Все вторичные узлы сохраняют целостность и автоматически обновляются вместе с обновлением главного узла. И если основной узел по

каким-то причинам выходит из строя, то один из вторичных узлов становится главным.

Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов[14].

Одной из проблем при работе с любыми системами баз данных является сохранение данных большого размера. Можно сохранять данные в файлах, используя различные языки программирования. Некоторые СУБД предлагают специальные типы данных для хранения бинарных данных в БД (например, BLOB в MySQL).

В отличие от реляционных СУБД MongoDB позволяет сохранять различные документы с различным набором данных, однако при этом размер документа ограничивается 16 мб. Но MongoDB предлагает решение - специальную технологию GridFS, которая позволяет хранить данные по размеру больше, чем 16 мб.

Система GridFS состоит из двух коллекций. В первой коллекции, которая называется files, хранятся имена файлов, а также их метаданные, например, размер. А в другой коллекции, которая называется chunks, в виде небольших сегментов хранятся данные файлов, обычно сегментами по 256 кб[6].

### 1.3. Операции с данными в MongoDB

Все данные хранятся в формате BSON, который близок к JSON, поэтому нужно также вводить данные в этом формате. Даже если в базе данных нет ни одной коллекции, то при добавлении в нее данных она автоматически создается.

Стоит отметить, что имя коллекции – произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания. В то же время имя коллекции не должно начинаться с префикса system., так как он зарезервирован для внутренних коллекций (например, коллекция system.users содержит всех пользователей базы данных). И также имя не должно содержать знака доллара - $.

Для добавления документа в коллекцию могут использоваться три метода:

- insertOne(): добавляет один документ;
- insertMany(): добавляет несколько документов;
- insert(): может добавлять как один, так и несколько документов.

Например, добавим один документ:

>db.users.insertOne({"name": "Tom", "age": 28, languages: ["english", "spanish"]})

Документ представляет набор пар ключ-значение. В данном случае добавляемый документ имеет три ключа: name, age, languages, и каждому из них сопоставляет определенное значение. Отметим, что ключу languages в качестве значения сопоставляется массив. Стоит отметить, что названия ключей могут использоваться в кавычках, а могут и без кавычек.

Существуют некоторые ограничения при использовании имен ключей, например символ $ не может быть первым символом в имени ключа, а имя ключа не может содержать символ точки. Также имя _id не рекомендуется использовать, так как если его опустить, то он добавляется автоматически.

В случае удачного добавления на консоль будет выведен идентификатор добавленного документа. Для того чтобы убедиться, что документ находится в БД, можно вывести его функцией find.

>db.users.find()

И третий метод - insert() демонстрирует более универсальный способ добавления документов. При его вызове в него также передается добавляемый документ:

>db.users.insertOne({"name": "Tom", "age": 28, languages: ["english", "spanish"]})

После его вызова на консоль выводится количество добавленных записей: WriteResult({"Inserted:1"})

В процессе работы, возможно, потребуется изменить название коллекции. Например, если при первом добавлении данных в ее названии была опечатка. И чтобы не удалять и затем пересоздавать коллекцию, следует использовать функцию renameCollection:

>db.users.renameCollection("новое_название")

И если переименование пройдет удачно, то консоль отобразит:{"ok" : 1}

**Явное создание коллекции**

В предыдущем примере коллекция создавалась неявно автоматически при добавлении в нее первых данных. Но мы также можем создать ее явным образом, применив метод **db.createCollection(name, options)**, где name - название коллекции, а options - необязательный объект с дополнительными настройками инициализации. Например:

>db.createCollection("accounts")

{"ok" : 1}

Таким образом, создается коллекция accounts.

**Ограниченные коллекции**

Когда мы отправляем запрос к БД на выборку, то MongoDB возвращает нам документы в том порядке, как правило, в котором они были добавлены. Однако такой порядок не всегда гарантируется, так как данные могут быть удалены, перемещены, изменены. Поэтому в MongoDB существует понятие ограниченной коллекции (cappedcollection). Подобная коллекция гарантирует, что документы будут располагаться в том же порядке, в котором они добавлялись в коллекцию. Ограниченные коллекции имеют фиксированный размер. И когда в коллекции уже нет места, наиболее старые документы удаляются, и в конец добавляются новые данные.

В отличие от обычных коллекций ограниченные мы можем задать явным образом. Например, создадим ограниченную коллекцию с названиемprofile и зададим для нее размер в 9500 байт:

>db.createCollection("profile", {capped:true, size:9500})

И после удачного создания коллекции консоль выведет:{"ok:1"}

Также можно ограничить количество документов в коллекции, указав его в параметре max:

>db.createCollection("profile", {capped:true, size:9500, max: 150})

Однако при таком способе создания коллекции следует учитывать, что если все место под коллекцию заполнено (например, выделенные нами 9500 байтов), а количество документов еще не достигло максимума, в данном случае 150, то в этом случае при добавлении нового документа самый старый документ будет удаляться, а на его место будет вставляться новый документ. При обновлении документов в таких коллекциях надо помнить, что документы не должны расти в размерах, иначе обновление не удастся произвести.Также нельзя удалять документы из подобных коллекций, можно только удалить всю коллекцию.

**Подколлекции**

Для упрощения организации данных в коллекциях мы можем использовать подколлекции. Например, данные по коллекции users надо разграничить на профили и учетные данные.Для этого необходимо создать коллекции db.users.profiles и db.users.accounts. При этом они не будут никак связаны с коллекцией users. То есть в итоге будут три разные коллекции, однако в плане логической организации хранения данных, возможно, для кого-то так будет проще.

Наиболее простой способом получения содержимого БД представляет использование функции find. Действие этой функции во многом аналогично обычномуSQL запросу SELECT * FROM Table, который извлекает все строки. Например, чтобы извлечь все документы из коллекции users, мы можем использовать команду db.users.find().

Для вывода документов в более удобном наглядном представлении мы можем добавить вызов метода **pretty()**:

```
>db.users.find().pretty()
```

Если необходимо получить не все документы, а только те, которые удовлетворяют определенному требованию. Например, выведем все документы, в которых name=Tom:

```
>db.users.find({name: "Tom"})
```

Такой запрос выведет нам все документы, в которых name=Tom.

**Проекция**

Документ может иметь множество полей, но не все эти поля нам могут быть нужны и важны при запросе. И в этом случае мы можем включить в выборку только нужные поля, использовав проекцию. Например, выведем только порцию информации, например, значения полей "age" у всех документов, в которых name=Tom:

```
>db.users.find({name: "Tom"}, {age: 1})
```

Использование единицы в качестве параметра {age: 1} указывает, что запрос должен вернуть только содержание свойства age.

И обратная ситуация: мы хотим найти все поля документа кроме свойства age. В этом случае в качестве параметра указываем 0:

```
>db.persons.find({name: "Tom"}, {age: 0})
```

При этом надо учитывать, что даже если мы отметим, что мы хотим получить только поле name, поле _id также будет включено в результирующую выборку. Поэтому, если мы не хотим видеть данное поле в выборке, то надо явным образом указать: {"_id":0}

Альтернативно вместо 1 и 0 можно использовать true и false.

**Запрос к вложенным объектам**

Предыдущие запросы применялись к простым объектам. Но документы могут быть очень сложными по структуре. Например, добавим в коллекцию persons следующий документ:

>db.users.insert({"name": "Alex", "age": 28, company: {"name":"microsoft", "country":"USA"}})

Здесь определяется вложенный объект с ключом company. И чтобы найти все документы, у которых в ключе company вложенное свойство name=microsoft, нам надо использовать оператор точку:

>db.users.find({"company.name": "microsoft"})

**Поиск одиночного документа**

Если все документы извлекаются функцией find, то одиночный документ извлекается функцией findOne.

**Курсоры**

Результат выборки, получаемой с помощью функции find, называется **курсором**:

>var cursor = db.users.find(); null;

Чтобы получить курсор и сразу же не выводить все содержащиеся в нем данные, после метода find() добавляет через точку с запятой выражение null;

Курсоры инкапсулируют в себе наборы получаемых из БД объектов. Используя синтаксис языка javascript и методы курсоров, мы можем вывести полученные документы на экран и как-то их обработать. Например:

```
>var cursor = db.users.find();null;
> while(cursor.hasNext()){
... obj = cursor.next();
... print(obj["name"]);
... }
```

Курсор обладает методом **hasNext**, который показывает при переборе, имеется ли еще в наборе документ. А метод **next** извлекает текущий документ и перемещает курсор к следующему документу в наборе. В итоге в переменной obj оказывается документ, к полям которого мы можем получить доступ.

Также для перебора документов в курсоре в качестве альтернативы мы можем использовать конструкцию итератора javascript - **forEach**:

```
>var cursor = db.users.find()
>cursor.forEach(function(obj){
... print(obj.name);
... })
```

Используя метод sort(), можно отсортировать документы в курсоре:

```
>var cursor = db.users.find();null;
null
>cursor.sort({name:1});null;
null
>cursor.forEach(function(obj){
... print(obj.name);
... })
```

Выражение cursor.sort({name:1}) сортирует документы в курсоре по полю name по возрастанию. Если мы хотим отсортировать по убыванию, то вместо 1 используем -1: cursor.sort({name:-1})

И еще один метод skip() позволяет пропустить при выборке определенное количество документов:

```
>var cursor = db.users.find();null;
null
>cursor.skip(2);null;
null
>cursor.forEach(function(obj){
... print(obj.name);
... })
```

В данном случае пропускаем два документа.

**Число элементов в коллекции**

С помощью функции count() можно получить число элементов в коллекции:

```
>db.users.count()
```

Можно группировать параметры поиска и функцию count, чтобы подсчитать, сколько определенных документов, например, у которых name=Tom:

```
>db.users.find({name: "Tom"}).count()
```

**Группировка и метод group**

Использование метода group аналогично применению выражения GROUPBY в SQL. Метод group принимает три параметра:

- **key**: указывает на ключ, по которому надо проводить группировку
- **initial**: представляет базовое значение для сгруппированного результата
- **reduce**: представляет функцию, возвращающую количество элементов. Эта функция принимает в качестве аргументов два параметра: items и prev
- **keyf**: необязательный параметр. Используется вместо параметра key и представляет функцию, которая возвращает объект key
- **cond**: необязательный параметр. Представляет собой условие, которое должно возвращать true, иначе документ не примет участия в группировке. Если данный параметр неуказан, то в группировке участвуют все документы
- **finalize**: необязательный параметр. Представляет функцию, которая срабатывает перед тем, как будут возвращены результаты группировки.

Например:

```
>db.users.group ({key: {name : true}, initial: {total : 0},
reduce : function(items,prev){prev.total += 1}})
```

Разберем выражение. Параметр key указывает, что группировка будет проводиться по ключу name: key: {name :true}. Значение параметра initial инициализирует начальное значение поля total. Это поле будет представлять количество элементов для группы. И так как элементов может и не быть, то инициализируем нулем. Параметр reduce представляет функцию, где параметр prev ссылается на предыдущий объект в группе. Если найден еще

один объект с определенным значением для поля name, то этот документ добавляется в группу, а у предыдущего документа в группе извлекается и увеличивается на единицу значение total.

**Метод save**

Как и другие СУБДMongoDB предоставляет возможность обновления данных. Наиболее простым для использования является метод save. В качестве параметра этот метод принимает документ.

В этот документ в качестве поля можно передать параметр _id. Если метод находит документ с таким значением _id, то документ обновляется. Если же с подобным _id нет документов, то документ вставляется. Если параметр _id не указан, то документ вставляется, а параметр _id генерируется автоматически как при обычном добавлении через функцию insert:

```
>db.users.save({name: "Eugene", age : 29, languages: ["english", "german",
"spanish"]})
```

В качестве результата функция возвращает объект WriteResult. Например, при успешном сохранении мы получим:

```
WriteResult({"nInserted": 1 })
```

**update**

Более детальную настройку при обновлении предлагает функция **update**. Она принимает три параметра:

- query: принимает запрос на выборку документа, который надо обновить
- objNew: представляет документ с новой информацией, который заместит старый при обновлении
- options: определяет дополнительные параметры при обновлении документов. Может принимать два аргумента: upsert и multi.

  Если параметр upsert имеет значение true, что mongodb будет обновлять документ, если он найден, и создавать новый, если такого документа нет. Если же он имеет значение false, то mongodb не будет создавать

новый документ, если запрос на выборку не найдет ни одного документа.

Параметр multi указывает, должен ли обновляться первый элемент в выборке (используется по умолчанию, если данный параметр не указан) или же должны обновляться все документы в выборке.

Например:

```
>db.users.update({name : "Tom"}, {name: "Tom", age : 25}, {upsert: true})
```

Теперь документ, найденный запросом {name : "Tom"}, будет перезаписан документом {"name": "Tom", "age" :"25"}.

Функция update() также возвращает объект WriteResult. Например:

```
WriteResult({"nMatched": 1, "nUpserted": 0, "nModified": 1})
```

В данном случае результат говорит нам о том, что найден один документ, удовлетворяющий условию, и один документ был обновлен.

**Обновление отдельного поля**

Часто не требуется обновлять весь документ, а только значение одного из его ключей. Для этого применяется оператор **$set**. Если документ не содержит обновляемое поле, то оно создается.

```
>db.users.update({name : "Tom", age: 29}, {$set: {age : 30}})
```

Если обновляемого поля в документе нет, до оно добавляется:

```
>db.users.update({name : "Tom", age: 29}, {$set: {salary : 300}})
```

В данном случае обновлялся только один документ, первый в выборке. Указав значение multi:true, мы можем обновить все документы выборки:

```
>db.users.update({name : "Tom"}, {$set: {name: "Tom", age : 25}}, {multi:true})
```

Для простого увеличения значения числового поля на определенное количество единиц применяется оператор **$inc**. Если документ не содержит обновляемое поле, то оно создается. Данный оператор применим только к числовым значениям.

```
>db.users.update({name :"Tom"}, {$inc: {age:2}})
```

**Удаление поля**

Для удаления отдельного ключа используется оператор **$unset**:

>db.users.update({name : "Tom"}, {$unset: {salary: 1}})

Если вдруг подобного ключа в документе не существует, то оператор не оказывает никакого влияния. Также можно удалять сразу несколько полей:

>db.users.update({name : "Tom"}, {$unset: {salary: 1, age: 1}})

**Удаление элемента из массива**

Оператор $pop позволяет удалять элемент из массива:

>db.users.update({name : "Tom"}, {$pop: {languages: 1}})

Указывая для ключа languages значение 1, мы удаляем первый элемент с конца. Чтобы удалить первый элемент сначала массива, надо передать отрицательное значение:

>db.users.update({name : "Tom"}, {$pop: {languages: -1}})

Для удаления документов в MongoDB предусмотрен метод **remove**:

>db.users.remove({name : "Tom"})

Метод remove() возвращает объект WriteResult. При успешном удалении одного документа результат будет следующим:

WriteResult({"nRemoved" : 1})

В итоге все найденные документы с name=Tom будут удалены. Причем, как и в случае с find, мы можем задавать условия выборки для удаления различными способами (в виде регулярных выражений, в виде условных конструкций и т.д.):

>db.users.remove({name : /T\w+/i})

>db.users.remove({age: {$lt : 30}})

Метод remove также может принимать второй необязательный параметр булевого типа, который указывает, надо удалять один элемент или все элементы, соответствующие условию. Если этот параметр равен true, то удаляется только один элемент. По умолчанию он равен false:

>db.users.remove({name : "Tom"}, true)

Чтобы удалить разом все документы из коллекции, надо оставить пустым параметр запроса:

>db.users.remove({})

**Удаление коллекций и баз данных**

Мы можем удалять не только документы, но и коллекции и базы данных. Для удаления коллекций используется функция **drop**:

>db.users.drop()

И если удаление коллекции пройдет успешно, то консоль выведет:true. Чтобы удалить всю базу данных, надо воспользоваться функцией **dropDatabase()**:

>db.dropDatabase()

## Возможные аналоги

В последнее время NoSQL стало модным словом в web-разработке. Некоторые проекты используют подобные БД для узких частей архитектуры, некоторые мигрируют на них полностью. Такие БД как правило отличаются от реляционных простой архитектурой и высокой масштабируемостью. На данный момент нас интересует MongoDB как одна из наиболее популярных NoSQL баз данных. Постараемся выделить сильные и слабые стороны, а также рассмотреть некоторые особенности разработки.

MongoDB это что-то среднее между key-value хранилищами (которые обычно быстры и масштабируемы) и традиционными реляционными базами данных (MySQL, PostgreSQL и т.п.), которые предоставляют расширенные запросы и богатый функционал.

MongoDB можно использовать в своих web-приложениях прямо сейчас. Последние версии достаточно стабильны для использования в производстве. Проект разрабатывает команда специалистов на постоянной основе, выходят багфиксы, появляются новые идеи по проекту, используется немалым числом крупных интернет-проектов, написаны драйверы для популярных языков программирования.

Рассмотрим некоторые различия в работе с MongoDB по сравнению с привычным SQL. Все примеры кода написаны на JavaScript и могут быть проверены в интерактивной консоли, о ней ниже.

По аналогии с тем, как в консоли mysql мы можем выполнять SQL-запросы, интерактивная консоль MongoDB позволяет выполнять команды сервера БД используя язык JavaScript. На официальном сайте можно найти online-вариант консоли, который работает прямо в браузере, плюс к тому небольшое введение для начинающих. Начинать рекомендуется именно с этого[14].

Структуру данных в реляционных системах на примере MySQL мы можем представить в виде следующей иерархии:
База данных -> Таблица -> Строка -> Поле + Значение.

Как это все выглядит в MongoDB:
База данных -> Коллекция -> Документ -> ключ + значение.

Таблицы в реляционных БД должны быть жестко структурированы, в то время как в MongoDB можно создавать документы произвольной структуры.

*Ниже представлен пример документа в формате JSON:*

```
{
doc_id: 153,
  title: 'Some title',
  body: '...A lot of text...',
author_id: 73,
  date: 'Sun Oct 31 2010 03:00:00 GMT+0300 (MSK)',
  additional: {
    location: {
      country: 'Russia',
      city: 'Moscow'
    },
    category: 'books'
```

```
  }
  tags: ["tag1", "tag2", "tag3"]
}
```

Как можно заметить у нас полная свобода во вложенности ключей документа, что избавляет нас от необходимости денормализации как в SQL, т.е. нам не нужно разносить одну сущность в разные документы. Как и в SQL в MongoDB есть индексы, причем полная их поддержка. Мы можем строить индекс по любому ключу приведенного выше документа, в том числе по вложенному country или по массиву tags. Можно делать составные индексы по нескольким ключам документа. Правила оптимизации индексов для SQL во многом подходят и для MongoDB и описаны в документации.

Предположим нам надо выбрать все документы из определенной коллекции у которых значение city равно "Moscow".В SQL нам приходится прибегать к инструкции JOIN, т.к. в унифицированной системе мы не всегда можем записать всю информацию в одну таблицу.

*Например:*

```
SELECT docs.title, loc.city FROM documents docs
INNER JOIN doc_locationd_loc ON d_loc.doc_id = docs.doc_id
INNER JOIN location loc ON loc.loc_id = d_loc.loc_id
```

*Как это делается в MongoDB, учитывая что все хранится в одном документе:*

```
db.find({additional.location.city: "Moscow"});
```

По аналогии с JOIN мы также можем создавать ссылки на объекты других коллекций, и нам не придется делать отдельные запросы для получения связанных документов [29].

MongoDB хорошо справляется с большим количеством документов (миллионы), скорость выборки как и в SQL оптимизируется индексами, лимитами на количество получаемых документов за один запрос, как и в привычных реляционных БД индексы отрицательно влияют на скорость записи.

MongoDB поддерживает несколько видов атомарных операций (операции, которые либо выполняются целиком, либо не выполняются вовсе). Мы можем использовать синхронный и асинхронный тип записи, при синхронном типе – обновления должны распространиться по большинству узлов или даже по всем узлам для того, чтобы операция изменения данных считалась завершенной,асинхронный по умолчанию быстрее, так как приложению не приходится ждать ответа от сервера.

MongoDB рекомендуют использовать при большом количестве одновременных запросов (более тысячи в секунду), особенно при большом количестве операций записи. Судя по многочисленным отзывам, именно быстрая запись одно из главных преимуществ этой БД.

## Администрирование БД

Рассмотрим некоторые аспекты администрирования БД.

*Перенос БД / Резервное копирование:* для создания горячей копии БД удобно использовать утилиту mongodump, которая по умолчанию идет в пакете с сервером. Создание копии всей БД, предназначенной для восстановления данных вслучае их повреждения, сводится к одной команде, например:

mongodump -d DATABASE_NAME

На выходе мы получаем папку с файлами в формате BSON. Эту папку используем для восстановлениярезервной копии, например:

mongorestore BACKUP_FOLDER

*Интерактивная консоль:* описанная выше интерактивная консоль является отличным инструментом администрирования БД. При помощи нее мы можем тестировать запросы, проверять соединение, создавать индексы, просматривать статус выполнения текущей операции и вообще любые другие административные функции.

*Административный интерфейс:* для MongoDB существует несколько визуальных инструментов для администрирования, среди которыхклиенты для OS X и .NET, а также web-интерфейсы на PHP,Python, Ruby[Там же].

В итоге, MongoDB конечно же не претендует на полноценную замену привычного MySQL или PostgreSQL. Его можно использовать для увеличения быстродействия отдельных элементов архитектуры и особенно узких мест. Например, неплохим решением было бы использование этой БД для ведения статистики, системы кэширования, хранения пользовательских сессий, ведения системного журнала, системы управления очередями и т.д.

## 1.4. Web-интерфейсы доступа к БД

Web-интерфейс – это совокупность средств, при помощи которых пользователь взаимодействует с веб-сайтом или любым другим приложением через браузер.

Причина создания web-интерфейсов для доступа к базам данных состоит в том, что доступ к базам данных (даже к тем, которые содержат полностью открытую информацию) ограничен. Чтобы получить интересующую информацию, пользователь должен иметь физический доступ к соответствующей СУБД, быть в курсе модели данных, знать схему базы данных и, наконец, уметь пользоваться соответствующим языком запросов[12].

На сегодняшний день существуют несколько работающих механизмов, позволяющих работать с базами данных с помощью web-интерфейса, и далее мы их обсудим. В целом механизмы делятся на два класса: обеспечивающие доступ к базе данных (по запросу клиента) на стороне Web-сервера и работающие непосредственно на стороне клиента.

**1. Доступ к базе данных на стороне сервера**

Механизм реализуется за счет наличия двух стандартизованных средств: возможности включения форм в документ, составленный с использованием языка гипермедийной разметки HTML (*HyperTextMarkupLanguage* – «Язык гипертекстовой разметки» – стандартизированный язык разметки документов в сети Интернет. Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства), и возможности использования внешних по отношению к серверу web-программ, взаимодействие которых происходит через специфицированный протокол CGI (Common Gateway Interface) или API (Application Program Interface).

CGI – стандарт интерфейса, используемого для связи внешней программы с web-сервером. По сути, позволяет использовать консоль ввода и вывода для взаимодействия с клиентом.

Сам интерфейс разработан таким образом, чтобы можно было использовать любой язык программирования, который может работать со стандартными устройствами ввода-вывода.

API (программный интерфейс приложения) – набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах.

При использовании в основе CGI общая схема реализации доступа к базе данных на стороне web-сервера выглядит следующим образом:

- при просмотре документа клиент встречает ссылку на страницу, содержащую одну или несколько форм, предназначенных для запроса данных из базы данных;
- клиент запрашивает эту страницу; помимо незаполненных форм страница может содержать общую информацию о базе данных и о назначении предлагаемых форм;

- если клиента действительно интересует информация из базы данных, которую можно получить на основе предложенных форм, то он заполняет одну из форм и отправляет ее на сервер;
- получив заполненную форму, сервер запускает соответствующую внешнюю программу, передавая ей параметры и получая результаты на основе протокола CGI;
- внешняя программа преобразует запрос, выраженный с помощью заполненной формы, в запрос на языке, понятном серверу баз данных. Говоря формально, внешняя программа осуществляет компиляцию языка форм в базовый язык сервера баз данных. Это может быть очень простая компиляция с использованием заготовок SQL-операторов. Между прочим, многие люди недооценивают сложность этой проблемы. Язык форм ничем не проще любого другого языка.
- внешняя программа взаимодействует с сервером баз данных. Взаимодействие может быть прямым, если внешняя программа жестко привязана к конкретному SQL-серверу, или при отсутствии жесткой привязки с использованием, протокола и соответствующего драйвера ODBC (*Open Database Connectivity* – это программный интерфейс доступа к базам данных, разработанный на основе спецификаций Call Level Interface (CLI). Стандарт CLI призван унифицировать программное взаимодействие с СУБД, сделать его независимым от поставщика СУБД и программно-аппаратной платформы);
- после получения результатов запроса внешняя программа формирует соответствующую виртуальную или реальную HTML-страницу, передает ее серверу и завершает свое выполнение;
- сервер передает сформированную HTML-страницу клиенту, и на этом процедура доступа к базе данных завершается (сервер разрывает транспортное соединение с клиентом) [Там же].

На сленге Web-мастеров любая внешняя программа, запускаемая Web-сервером в соответствии со спецификациями CGI, называется CGI-скриптом. CGI-скриптможет быть написан на языке программирования (Си, Си++, Паскаль и т.д.) или на командном языке (языки семейства shell, perl и т. д.). CGI-скрипт, выполняющий роль посредника между Web-сервером и другими видами серверов (например, сервером баз данных), называется шлюзом. Наличие CGI-скриптов на стороне Web-сервера позволяет, в частности, перенести часть логики приложения из клиента в сервер.

В спецификациях CGI предусмотрены четыре способа взаимодействия Web-сервера и CGI-скрипта (по крайней мере в среде ОС Unix). Используется создаваемые сервером переменные окружения, через которые передается как общая информация, независящая от функциональных особенностей CGI-скрипта (например, имя и версия Web-сервера), так и специфические данные, определяющие поведение CGI-скрипта (скажем набор значений, введенных в форму на стороне клиента).

При использовании CGI вся интерпретация пользовательского запроса производится CGI-скриптом. Скрипт может быть ориентированным на выполнение запроса к фиксированной таблице фиксированной базы данных, или способным выполнить произвольный запрос к одной или нескольким таблицам базы данных, идентифицируемой в параметрах клиента.

Что касается API, то запуск независимого "тяжеловесного" процесса стоит немало. Загрузка и выполнение еще одной программы в уже существующем адресном пространстве по сравнению с предыдущим вариантом гораздо дешевле. API - это, фактически, дешевый способ выполнить в адресном пространстве сервера WWW программу, которая соответствует спецификациям на языке HTML (внешние программы незнакомы серверу). Такая программа должна быть заранее подготовлена и включена в библиотеку, из которой сервер может производить динамическую загрузку. То есть смысл не меняется, меняется только реализация.

## 2. Доступ к базе данных на стороне клиента

Наиболее мощные средства обеспечения доступа к базам данных на стороне Web-клиента обеспечивает язык Java. Java - это объектно-ориентированный язык программирования, являющийся "безопасным" подмножеством языка Си++. В частности, Java не содержит средств адресной арифметики и не поддерживает механизм множественного наследования.

Различают:

- язык Java, для которого существуют компиляторы в так называемый "мобильный код" (машинно-независимый код, который может интерпретироваться или из которого могут генерироваться машинные коды на разных платформах);
- язык JavaScript, который обычно используется для расширения возможностей языка HTML за счет добавления процедурной составляющей;
- программный продукт HotJava, являющийся, по сути, интерпретатором мобильных кодов Java.

Для обеспечения доступа к базам данных на стороне Web-клиента наиболее существенно наличие языка Java. Технология разработки HTML-документа позволяет написать произвольное количество дополнительных Java-программ, откомпилировать их в мобильные коды и поставить ссылки на соответствующие коды в теле HTML-документа. Такие дополнительные Java-программы называются апплетами (Java-applets). Получив доступ к документу, содержащему ссылки на апплеты, клиентская программа просмотра запрашивает у Web-сервера все мобильные коды. Коды могут начать выполняться сразу после размещения в компьютере клиента или быть активизированы с помощью специальных команд [Там же].

Поскольку апплет представляет собой произвольную Java-программу, то, в частности, он может быть специализирован для работы с внешними базами данных. Более того, система программирования Java включает развитый набор классов, предназначенных для поддержки графического

пользовательского интерфейса. Опираясь на использование этих классов, апплет может получить от пользователя информацию, характеризующую его запрос к базе данных, в том же виде, как если бы использовался стандартный механизм форм языка HTML, а может применять какой-либо другой интерфейс.

Для взаимодействия Java-апплета с внешним сервером баз данных разработан специализированный протокол JDBC, который, фактически, сочетает функции шлюзования между интерпретатором мобильных Java-кодов и ODBC, а также включает ODBC.

В заключение сравним достоинства и недостатки двух рассмотренных подходов. Использование CGI-скриптов на стороне Web-сервера позволяет иметь на стороне клиента только сравнительно простые программы просмотра. Вся хитроумная логика работы с базами данных (возможно, с обработкой полученных данных) переходит на сторону Web-сервера. В последнее время разгрузку клиента от логики приложения называют решением проблемы "толстого" клиента. В данном случае мы можем иметь предельно тонкого клиента при утолщении сервера. Отрицательным моментом является то, что при необходимости подключения нового CGI-скрипта, требуется модификация кода сервера [Там же].

Использование Java-апплетов, обеспечивает более гибкое решение. Апплет - это часть HTML-документа. Для включения нового апплета нужно всего-навсего перекомпоновать документ. Web-сервер трогать не нужно. С другой стороны, клиент должен быть толще. Клиент должен быть достаточно "толстым", чтобы в приемлемое время справиться с интерпретацией всех апплетов. Но, конечно же, сервер по-прежнему должен быть "толще" клиента.

На самом деле и при применении первого подхода, и при использовании второго остается нерешенной одна организационно-производственная проблема: кто должен проектировать, писать, отлаживать и сопровождать процедурный код. Web-мастера, производящие HTML-

документы, обычно считают себя, скорее, дизайнерами нежели программистами. А здесь требуется чисто программистская работа. Иметь же двух мастеров на одном web-сайте– это недопустимая роскошь. Придется искать людей, обладающих достаточными эстетическими способностями и являющимися в то же время квалифицированными и ответственными программистами [12].

Существует множество web-интерфейсов для работы с базами данных, которые облегчают управление разработкой и их администрированием. Также основным преимуществом таких интерфейсов является отсутствие необходимости установки дополнительного программного обеспечения.

У web-мастеров, например, считается стандартом phpMyAdmin. Также используют phpPgAdmin, Chive,Adminer и другие.

Многие web-интерфейсы предоставляются с открытым исходным кодом, и покрывают  почти все нужды разработчиков. В основном они ориентируются на такие возможности как:

- поддержка MySQL, PostgreSQL, SQLite, MS SQL, Oracle, SimpleDB, Elasticsearch, MongoDB;
- поддержка мультиязычности;
- просмотр, создание, изменение данных;
- работа со структурами таблиц и баз;
- работа с индексами;
- дамп и импорт;
- механизм аутентификации с сохранением сессий[12].

Так как существующие web-интерфейсы работают только напрямую с базой данных, то актуальна задача определения методов и технологий которые позволяют пользователю редактировать базу данных прямо на своем сайте.

# ГЛАВА 2. РАЗРАБОТКА ОБРАЗОВАТЕЛЬНОГО САЙТА

**ВЫПИСКА**
из протокола № 11 заседания кафедры мультимедийной дидактики и информационных технологий обучения Пермского государственного педагогического университета
от 14 июня 2017 ГЮ

**ПРИСУТСТВОВАЛИ:** зав. кафедрой., д.п.н., профессор Е.В. Оспенникова; канд. тех. наук, профессор О.И. Мухин, кандидат физ.-мат. наук, доцент Е.А. Еремин; кандидат физ.-мат. наук, кандидат физ.-мат. наук, доцент Д.В. Баяндин; доцент, канд. пед.наук И.В. Ильин: доцент, канд. тех. наук О.Ю. Вологжанин; доцент, канд. пед. наук А.А. Оспенников, АСС. Д.А Голубев, асп. В.В. Аспидов; асп. В.В. Васенев; асп. Д.А. Терехин
**СЛУШАЛИ:** Руководителей выпускных квалификационных работ студентов, обучающихся по направлению 09.02.03 «Информационные технологии в образовании», о наличии в тексте ВКР сведений, имеющих действительную и/или потенциальную коммерческую ценность.
В соответствии с п.38 Приказа Министерства образования и науки РФ от 29.06.2015 № 636 «Об утверждении Порядка проведения государственной итоговой аттестации по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры» доступ лиц к текстам выпускных квалификационных работ должен быть обеспечен в соответствии с законодательством Российской Федерации, с учетом изъятия производственных, технических, экономических, организационных и других сведений, в том числе о результатах интеллектуальной деятельности в научно-технической сфере, о способах осуществления профессиональной деятельности, которые имеют действительную или потенциальную коммерческую ценность в силу неизвестности их третьим лицам, в соответствии с решением правообладателя. 9
**ПОСТАНОВИЛИ:** изъять из текста выпускной квалификационной работы студента Воеводина Ильи Андреевича страницы с 36 по 56 при размещении работы в электронно-библиотечной системе ПГГПУ http://vkr.pspu.ru/.

Зав. Кафедрой МД и ИТО, профессор Оспенникова Е.В.

14 июня 2017 года

## *Заключение*

По итогам выполненной работы получены следующие результаты:

1. Рассмотрены основные функции, компоненты и классификации СУБД. Произведено сравнение современных СУБД, выявлены их достоинства и недостатки.
2. Раскрыты преимущества и сильные стороны СУБД MongoDB.
3. Разработан общеобразовательный сайт школы, создана база данных в MongoDB.
4. Создан web-интерфейс для работы с базой данных на сайте.
5. Подготовлено руководство пользователя.
6. Описаны актуальные на данный момент проблемы, возникшие при попытке опубликовать данный проект в сети Интернет, а так же представлены возможные пути решения.

Данный проект предназначен для разработки web-интерфейса, позволяющего совершать основные операции с базой данных на своем сайте непрофессиональными разработчиками. Так же стоит отметить, что для редактирования любого выбранного элемента пользователю не нужно для этого знать структуру и состав базы данных, которую он изменяет, а также язык запросов, который используется при работе с базой данных.

## Библиографический список


1. Доступ к данным через web-интерфейс [электронный ресурс]. – URL: https://www.titorov.ru/index.php/edu/it-tecnology/87-it-1-6(Дата обращения: 12.05.2017).
2. Динамические языки программирования [электронныйресурс]. – URL: http://bourabai.ru/alg/dynamic.htm(Датаобращения: 12.05.2017).
3. Изучение PHP [электронный ресурс]. – URL: http://www.php.su/learnphp/?learnphp(Дата обращения: 12.05.2017).
4. Когаловский М.Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика– 2002. – 800 с.
5. Кузнецов С. Д. Основы баз данных. – 2-е изд. – М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007. – 484 с.
6. Официальный сайт СУБДMongoDB[электронныйресурс]. – URL: https://www.mongodb.comДатаобращения: 12.05.2017).
7. Официальный сайт программы OpenServer[электронныйресурс]. – URL: https://ospanel.io/(Датаобращения: 12.05.2017).
8. Руководство поPHP [электронный ресурс]. – URL: http://php.net/manual/ru/(Дата обращения: 12.05.2017).
9. Реальныеweb-проекты на PHP и MySQL [электронный ресурс]. – URL: https://www.ibm.com/developerworks/ru/library/l-mail_1(Дата обращения: 12.05.2017).
10. Сайт о программировании. Руководство по PHP [электронный ресурс]. – URL: http://metanit.com(Дата обращения: 12.05.2017).
11. Самоучитель html [электронный ресурс]. – URL: http://htmlbook.ru/html(Дата обращения: 12.05.2017).
12. Сервер Учебно-методических комплексов ЦОО ФИСТ УлГТУ[электронныйресурс]. – URL: http://sumk.ulstu.ru/(Датаобращения: 12.05.2017).

13. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системыбазданных. Полныйкурс = Database Systems: The Complete Book. –Вильямс, 2003. – 1088 с.
14. Дэвид Хоус, Питер Мембрей, ЭлкоПлагги, Тим Хоукинс. The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB, Third Edition. –Apress, 2015. – 376 с.
15. Кайл Бэнкер. MongoDB в действии. – ДМК Пресс, 2014. – 394 с.
16. КарлСегуин. TheLittleMongoDBBook [электронныйресурс]. – URL: http://jsman.ru/mongo-book(Дата обращения: 12.05.2017).
17. Коннолли Т., Бегг К. Базы данных. Проектирование, реализацияисопровождение. Теорияипрактика / Database Systems: A Practical Approach to Design, Implementation, and Management. – 3-еизд. – М.:Вильямс, 2003. –1436 с.
18. КристинаЧодороу, МайклДирольф. MongoDB: The Definitive Guide, 2nd Edition. – O'Reilly Media, Inc., 2013. – 432 с.
19. КристоферДейт .Введениевсистемыбазданных / Introduction to Database Systems. – 8-еизд. –М.: Вильямс, 2005. –1328 с.
20. КристоферДейт. Date on Database: Writings 2000–2006. –Apress, 2006. – 566 с.
21. ЛиннБейли, Майкл МоррисонИзучаем PHP и MySQL – 2010. – 768с
22. МитчПиртл. MongoDB for Web Development. – Addison-Wesley Professional – 2011. – 360с.
23. ПитерМембрей, ТимХоукинс. The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing. –Apress, 2010. – 327 с.
24. РубайитИслам. PHP and MongoDB Web Development Beginner's Guide – 2011.– 292с.
25. СтивГоберман. Data Modeling for MongoDB. – Technics Publications, 2014. – 226 с.
26. СтивСейхринг, ЖанетВалайд. Learning PHP, MySQL, & JavaScript 4th Edition – 2013. – 720с.

27. СтивФранцуа. MongoDB and PHP. Document-Oriented Data for Web Developers – 2012. – 80с.
28. MongoDB-PHP [электронныйресурс]. – URL: https://www.tutorialspoint.com/mongodb/mongodb_php.htm(Дата обращения: 12.05.2017).
29. The MongoDB class. Documentation for PHP [электронныйресурс]. – URL: https://secure.php.net/manual/ru/class.mongodb.php(Датаобращения: 12.05.2017).