Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Фурсов Владимир Александрович
Должность: И.о. директора Пятигорского института (филиал) Северо-Кавказского федерального университета
Дата подписания: 08.06.2026 13:56:45
Уникальный программный ключ:
1c378726a41fd0143ae5bccc8ba818[illegible]

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Колледж Пятигорского института (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

МДК.03.02 РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ

Специальности СПО

09.02.011 Разработка и управление программным обеспечением

Пятигорск 2026

Методические указания для лабораторных работ по дисциплине МДК.03.02 Разработка кода информационных систем составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Программа дисциплины Разработка кода информационных систем предусматривает изучение алгоритмизации как базовой составляющей технологического процесса создания программного продукта, а так же изучение языка программирования и формирование навыков разработки ИС, объектно-ориентированного программирования у студентов. При изучении предмета следует соблюдать единство терминологии и обозначения в соответствии с действующими стандартами, Международной системной единицы (СИ).

В результате изучения дисциплины студенты *должны:*

знать:

- основные виды и процедуры обработки информации, модели и методы решения задач обработки информации;
- основные платформы для создания, исполнения и управления информационной системой;
- основные процессы управления проектом разработки;
- основные модели построения информационных систем, их структуру, особенности и области применения;
- методы и средства проектирования, разработки и тестирования информационных систем;
- систему стандартизации, сертификации и систему обеспечения качества продукции.

уметь:

- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств;
- использовать алгоритмы обработки информации для различных приложений;
- решать прикладные вопросы программирования и языка сценариев для создания программ;
- разрабатывать графический интерфейс приложения;
- создавать и управлять проектом по разработке приложения;
- проектировать и разрабатывать систему по заданным требованиям и спецификациям. **иметь практический опыт в**:
- управлении процессом разработки приложений с использованием инструментальных средств;
- обеспечении сбора данных для анализа использования и функционирования информационной системы;
- программировании в соответствии с требованиями технического задания;
- использовании критериев оценки качества и надежности функционирования информационной системы;
- применении методики тестирования разрабатываемых приложений;

-определении состава оборудования и программных средств разработки информационной системы;

- разработке документации по эксплуатации информационной системы;
- проведении оценки качества и экономической эффективности информационной системы в рамках своей компетенции;
- модификации отдельных модулей информационной системы.

В результате освоения учебной дисциплины студент должен овладевать:

Общими компетенциями:

ОК 01. Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях

ОК 04. Эффективно взаимодействовать и работать в коллективе и команде

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных российских духовно-нравственных ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения

ОК 07. Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях

ОК.08. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности

ОК 09. Пользоваться профессиональной документацией на государственном и иностранном языках

Профессиональными компетенциями:

ПК 3.1. Собирать исходные данные для разработки проектной документации на информационную систему

ПК 3.2. Разрабатывать проектную документацию на разработку информационной системы в соответствии с требованиями заказчика

ПК 3.3. Разрабатывать подсистемы безопасности информационной системы в соответствии с техническим заданием

ПК 3.4. Производить разработку модулей информационной системы в соответствии с техническим задание

ПК 3.5. Интегрировать информационную систему с существующими информационными системами заказчика

ПК 3.6. Осуществлять модульное и интеграционное тестирование информационной системы

ПК 3.7. Разрабатывать техническую документацию на эксплуатацию информационной системы

ПК 3.8. Производить оценку информационной системы для выявления возможности ее модернизации.

ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

По дисциплине Разработка кода информационных систем лабораторные работы содержат задачи и теоретические вопросы. Варианты для каждого обучающегося - индивидуальные.

Задачи и ответы на вопросы, выполненные не по своему варианту, не засчитываются.

Лабораторная работа выполняется в отдельной тетради. Условия задачи и формулировки вопросов переписываются полностью. Формулы, расчеты, ответы на вопросы пишутся ручкой, а чертежи, схемы и рисунки выполняются карандашом, на графиках и диаграммах указывается масштаб. Вначале задача решается в общем виде, затем делаются расчёты по условию задания. Решение задач обязательно ведется в Международной системе единиц (СИ).

При выполнении лабораторной работы необходимо следовать методическим указаниям: повторить краткое содержание теории, запомнить основные формулы и законы, проанализировать пример выполнения аналогичного задания, затем преступить непосредственно к решению задачи. К зачету допускаются студенты, получившие положительные оценки по всем лабораторным работам.

Правила выполнения лабораторной работы.

1. Студент должен прийти на лабораторную работу подготовленным к выполнению лабораторной работы.
2. Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.
3. Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и.т.д.) карандашом с соблюдением ЕСКД.
4. Расчет следует проводить с точностью до двух значащих цифр.
5. Исправления проводить на обратной стороне листа. При мелких исправлениях неправильное слово (буква, число и т.п.) аккуратно зачеркивается и над ним пишут правильное пропущенное слово (букву, число и т.п.).
6. Вспомогательные расчеты можно выполнять на отдельных листах, а при необходимости на листах отчета.
7. Если студент не выполнит лабораторную работу или часть работы, то он выполнит ее во внеурочное время, согласованное с преподавателем.
8. Оценку по лабораторной работе студент получает с учетом срока выполнения работы, если;

- расчеты выполнены правильно и в полном объеме;
- сделан анализ проделанной работы и вывод по результатам работы;
- студент может пояснить выполнение любого этапа работы;
- отчет выполнен в соответствии с требованиями к выполнению работы.

Лабораторная работа №1. выборка данных из тестовой БД.

Цель: Освоение методов и способов извлечения данных из реляционной базы данных с использованием языка SQL

Теоретическая часть:

База данных (БД) - это организованное хранилище данных, обеспечивающее быстрый доступ, целостность и сохранность информации. Реляционная модель основана на теории множеств и отношения (таблицы), где каждая таблица представляет собой двумерный массив данных, состоящий из строк и столбцов.

2. Язык SQL (Structured Query Language)

SQL - это декларативный язык запросов, используемый для работы с реляционными базами данных. Основные операции SQL:

- SELECT - извлекает данные из таблиц.
- INSERT - добавляет новые записи.
- UPDATE - изменяет существующее содержимое.
- DELETE - удаляет записи.

Примеры SQL-запросов:

```
-- Выборка всех сотрудников отдела Sales
SELECT *
FROM employees
WHERE department = 'Sales';

-- Суммарный доход сотрудников по отделам
SELECT department, SUM(salary)
FROM employees
GROUP BY department;
```

Задание на работу:

1. Создайте базу данных и таблицу для тестирования. Таблица должна содержать несколько атрибутов (например, employee_id, first_name, last_name, salary, department).
2. Заполните таблицу произвольными данными.
3. Получите все значения из таблицы

Лабораторная работа №2. создание и заполнение тестовой базы данных

Цель: Освоение процессов создания и наполнения тестовой базы данных с использованием языка SQL и популярной СУБД (например, PostgreSQL, MySQL, SQLite).

Теоретическая часть:

Основные составляющие базы данных

- Таблицы: основной строительный блок базы данных, содержащий данные в строках и столбцах.
- Атрибуты (столбцы): определяют тип хранимых данных (текст, число, дата и т.д.).
- Записи (строки): содержат индивидуальные значения данных, относящиеся к одному объекту.
- Индексы: ускоряют доступ к данным за счёт быстрого поиска.

3. Язык SQL

SQL - это специальный язык для работы с реляционными базами данных. Основные команды SQL:

- CREATE TABLE: создает новую таблицу.
- INSERT INTO: вставляет данные в таблицу.
- UPDATE: изменяет данные в таблице.
- DELETE: удаляет данные из таблицы.
- ALTER TABLE: изменяет структуру таблицы.
- DROP TABLE: удаляет таблицу целиком.

Пример создания таблицы:

```
CREATE TABLE employees (
   employee_id SERIAL PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   email VARCHAR(100),
   hire_date DATE,
   job_title VARCHAR(50)
);
```

Пример вставки данных:

```
INSERT INTO employees(first_name, last_name, email, hire_date, job_title)
VALUES ('Иван', 'Иванов', 'ivanov@example.com', '2020-01-01', 'Менеджер');
```

Задание на работу:

1. Создание схемы базы данныхСоздайте базу данных для небольшого онлайн-магазина. База данных должна включать три таблицы:
    - Users: хранение информации о пользователях сайта (идентификатор, имя, email, телефон).
    - Products: описание товаров магазина (идентификатор товара, название, цена, количество на складе).
    - Orders: заказы покупателей (идентификатор заказа, ссылка на покупателя, ссылку на товар, количество заказанного товара, статус оплаты).
2. Заполнение тестовыми даннымиЗаполните каждую таблицу хотя бы тремя разными записями.

Лабораторная работа №3. работа с смежными данными нескольких таблиц тестовой базы данных

Цель: Изучить основы построения сложных запросов SQL, включающих объединение данных из нескольких таблиц. Освоить методы нормализации данных и проектирование связей между таблицами, а также закрепление навыков извлечения данных с использованием объединений JOIN.

Теоретическая часть:

При проектировании информационной системы важно учитывать необходимость хранения и обработки больших объемов разнородных данных. Часто возникает потребность объединять данные из разных таблиц для анализа и предоставления целостной картины информации. Именно здесь ключевую роль играют операторы объединения таблиц в SQL.

Способы объединения таблиц (JOIN):

- INNER JOIN: Возвращает строки, совпадающие по условиям соединения двух таблиц.
- LEFT OUTER JOIN: Включает все строки первой таблицы и соответствующие ей строки второй таблицы, если условие выполняется.
- RIGHT OUTER JOIN: Аналогичен LEFT JOIN, но возвращает все строки второй таблицы.
- FULL OUTER JOIN: Объединяет обе таблицы и показывает все строки обеих таблиц независимо от совпадения условий.
- CROSS JOIN: Дает декартово произведение двух таблиц (каждая строка первой таблицы соединяется с каждой строкой второй таблицы).

Пример использования INNER JOIN:

Допустим, у вас есть таблица Users и таблица Orders:

sql

Копировать

```
SELECT Users.Name, Orders.Product, Orders.Quantity
FROM Users
INNER JOIN Orders ON Users.ID = Orders.UserID;
```

Этот запрос вернет список продуктов, приобретенных каждым пользователем вместе с количеством каждого продукта.

Задание на работу:

Создайте простую базу данных, состоящую из трех таблиц:

- Users (Пользователи): *id*, *name*
- Products (Продукты): *id*, *product_name*, *price*
- Orders (Заказы): *order_id*, *user_id* (ссылается на id из Users), *product_id* (ссылается на id из Products), *quantity*

Шаг 2. Написание запросов

1. Составьте запрос, возвращающий имена пользователей и названия купленных ими товаров, включая количество покупок.

Лабораторная работа №4. первичный анализ и обработка данных

Цель: Научиться эффективно применять агрегирующие функции SQL для выполнения расчетов над наборами данных. Закрепить навыки анализа и группировки данных, построенные на понимании агрегатных операторов SQL.

Теоретическая часть:

Агрегатные функции позволяют обрабатывать большие объемы данных, группируя их по определенным критериям и вычисляя статистические показатели, такие как сумма, среднее арифметическое, минимальное/максимальное значение и число элементов.

Основные агрегатные функции SQL:

- COUNT(): подсчет числа строк в заданном наборе данных.
- SUM(): вычисляет сумму значений в указанном столбце.
- AVG(): находит среднее арифметическое значение указанного столбца.
- MIN(), MAX(): определяют наименьшее и наибольшее значение соответственно.
- GROUP BY: группирует строки по указанному полю перед применением агрегатных функций.

Пример использования агрегирования:

Предположим, у вас есть таблица продаж Sales с полями (*Product*, *Quantity*, *Price*):

```
SELECT Product, COUNT(*) AS Num_Sales, AVG(Price) AS Average_Price
FROM Sales
GROUP BY Product;
```

Этот запрос выведет список товаров с числом проданных единиц и средней ценой продажи для каждого товара.

Задание на работу:

Создайте таблицу Sales следующей структуры:

Поле	Тип данных
Sale_ID	INT
Product	VARCHAR
Quantity	INT
Price	DECIMAL
Date	DATE

Проведите анализ данной таблицы применив каждую из агрегатных функций sql

Лабораторная работа №5.
нормализация схемы на примере
тестовой базы данных.

Цель: Изучение принципов нормализации реляционных схем баз данных и приобретение практических навыков проектирования нормализованных структур данных с целью повышения эффективности хранения и управления информацией.

Теоретическая часть:

Нормализация является процессом преобразования структуры базы данных таким образом, чтобы минимизировать избыточность данных и обеспечить целостность данных путем устранения аномалий обновления, вставки и удаления.

Существует несколько формальных уровней нормализации, называемых нормальными формами (NF). Основные формы нормализации включают:

1. Первая нормальная форма (1NF): Все атрибуты должны иметь атомарные значения (не повторяющиеся наборы значений); каждая запись уникальна.
2. Вторая нормальная форма (2NF): Таблица находится в первой нормальной форме, и каждый неключевой атрибут зависит от всего ключа.
3. Третья нормальная форма (3NF): Таблица находится во второй нормальной форме, и все неключевые атрибуты зависят только от ключей, а не друг от друга.
4. Бойса-Кодда (BCNF): Улучшенная версия третьей нормальной формы, устраняющая любые зависимости, кроме функциональной полной зависимости от ключа.

Процесс нормализации включает разделение таблиц и удаление избыточных зависимостей, приводящих к дублированию данных.

Задание на работу:
Представьте исходную схему базы данных магазина, содержащую следующую таблицу:

```
CREATE TABLE shop_orders (
   order_id INT PRIMARY KEY,
   customer_name VARCHAR(100),
   address VARCHAR(200),
   phone_number VARCHAR(20),
   product_name VARCHAR(100),
   quantity INT,
   total_price DECIMAL(10, 2)
);
```

Эта таблица имеет очевидные недостатки: адрес клиента и телефон повторяются многократно для разных заказов одного и того же покупателя, что вызывает избыточность данных и увеличивает риск ошибок.
Задача:

Определите потенциальные проблемы с производительностью и целостностью данных, вызванные структурой этой таблицы.

Часть II. Приведение к первой нормальной форме (1NF):

1. Создайте новую структуру базы данных, преобразовав исходную таблицу в первую нормальную форму.
2. Добавьте необходимые ограничения целостности (первичные ключи, внешние ключи и т.п.).

Результат:

Опишите шаги, предпринятые вами для приведения таблицы к первой нормальной форме.

Часть III. Приведение ко второй нормальной форме (2NF):

1. Преобразуйте вашу базу данных в вторую нормальную форму, разделив одну или несколько таблиц.
2. Убедитесь, что все неключевые столбцы зависят исключительно от полного ключа.

Результат:

Докажите, что ваша новая структура соответствует требованиям второй нормальной формы.

Часть IV. Приведение к третьей нормальной форме (3NF):

1. Проверьте наличие транзитивных зависимостей в ваших таблицах и устраните их, приведя таблицы к третьей нормальной форме.
2. Опишите изменения, внесённые в структуру базы данных.

Лабораторная работа №6. создание схемы с ограничениями на примере тестовой базы данных.

Цель: Приобретение навыков разработки эффективной схемы базы данных с использованием ограничений целостности, обеспечивающих согласованность и качество хранимых данных.

Теоретическая часть:

Ограничения целостности - это правила, устанавливаемые на уровне базы данных для контроля над вводимой информацией и поддержания логической непротиворечивости данных. Использование ограничений позволяет автоматически проверять соблюдение бизнес-правил и предотвращать внесение некорректных данных.

Основные типы ограничений:

1. NOT NULL: Запрещает пустое значение (NULL) для определенного атрибута.
2. UNIQUE: Обеспечивает уникальность значений конкретного атрибута или набора атрибутов.
3. PRIMARY KEY: Устанавливает уникальный идентификатор записи (ключ) и запрещает повторяющиеся и нулевые значения.
4. FOREIGN KEY: Связывает две таблицы, обеспечивая ограничение ссылочной целостности.

5. CHECK: Проверяет условие, которое должно соблюдаться для каждого вносимого значения.
6. DEFAULT: Устанавливает стандартное значение, используемое, если значение не указано явно.

Использование ограничений помогает избежать распространённых ошибок и повышает надёжность базы данных.

Задание на работу:

Создать схему базы данных онлайн-курсов с тремя основными таблицами:

- Курсы (Courses)
- Студенты (Students)
- Оценки (Grades)

Применить ограничения целостности аналогично примеру из лабораторной работы, реализовав поддержку уникальности, ссылочную целостность и проверку введённых данных.

Проверить работоспособность ограничений посредством попыток внести некорректные данные.

Лабораторная работа №7. оптимизация запросов для тестовой базы данных.

Цель: Изучение методов оптимизации SQL-запросов и повышение производительности операций выборки данных в реляционной базе данных

Теоретическая часть:

Оптимизация запросов - важный этап проектирования и эксплуатации баз данных, позволяющий значительно повысить эффективность выборки и обработки данных. Существует ряд техник и инструментов, позволяющих ускорить выполнение запросов и снизить нагрузку на сервер базы данных.

Основные методы оптимизации запросов:

1. Индексация: Индексы ускоряют доступ к записям, позволяя быстро находить нужные строки без сканирования всей таблицы.
2. Нормализация: Устранение избыточности данных уменьшает объем обрабатываемой информации и улучшает производительность.
3. Выбор оптимального порядка соединения таблиц: Порядок соединений влияет на скорость выполнения сложных запросов с JOIN.
4. Анализ плана выполнения запроса: План выполнения показывает порядок действий сервера при выполнении запроса, помогая выявить узкие места.
5. Правильный выбор типов данных: Выбор подходящего типа данных снижает размер хранилища и ускоряет операции чтения-записи.

Задание на работу:

Рассмотрите структуру базы данных учебного проекта интернет-магазина, состоящую из трёх таблиц:

users: Пользователи (id, username, email, registration_date)
products: Товары (id, product_name, price, category)
orders: Заказы (id, user_id, product_id, quantity, total_cost)
Определите наиболее частые сценарии запросов к этой базе данных.

Часть II. Определение неоптимизированных запросов:
Напишите запросы для решения следующих задач:

Получить список пользователей, сделавших заказ в течение последних трех месяцев.
Найти самые дорогие продукты каждой категории.
Посчитать общее количество продаж по каждому продукту.
Запустите профилирование выполнения этих запросов и определите, какой из них выполняется дольше остальных.

Часть III. Оптимизация запросов:
Попробуйте оптимизировать выбранный запрос несколькими способами:

Используйте индексы для ускорения доступа к данным.
Измените порядок соединений таблиц, если возможно.
Попробуйте применить фильтрации данных раньше, чем производить соединения.
Проанализируйте полученный прирост производительности и опишите достигнутый эффект.

Лабораторная работа №8. установка и настройка NoSQL-БД

Цель: Получение практических навыков установки, настройки и первичной работы с базой данных NoSQL-типа MongoDB. Освоение базовых команд администрирования и взаимодействия с базой данных через клиентские интерфейсы.

Теоретическая часть:
NoSQL-базы данных отличаются от традиционных реляционных баз данных отсутствием строгих схем и поддержкой гибких моделей данных. Одной из популярных NoSQL-СУБД является MongoDB, основанная на документах JSON-подобного формата BSON.
MongoDB обладает рядом особенностей:

- Документы хранятся в коллекциях (аналог таблиц в реляционных БД).
- Отсутствие жёсткого отношения между полями документов.
- Высокая масштабируемость и поддержка горизонтального шардинга.
- Гибкая система репликации для повышения доступности данных.

Установка и настройка MongoDB предполагает установку самого ПО, настройку конфигурации и запуск сервиса. После этого можно подключаться к базе данных через консоль или специализированные клиенты.

Задание на работу:

Установите последнюю стабильную версию MongoDB Community Edition на свою рабочую станцию. Для этого используйте инструкции для вашей операционной системы (Windows, Linux, macOS).
Последовательность шагов:

1. Скачать дистрибутив MongoDB с официального сайта (https://www.mongodb.com/download-center/community).
2. Установить пакет с помощью мастера инсталляции или вручную распаковать архив и настроить переменную окружения PATH.
3. Настроить конфигурационный файл mongod.conf для запуска демона mongodb.
4. Запустить сервис MongoDB командой mongod --config /path/to/mongod.conf.

Часть II. Настройка и проверка работоспособности:
Подключитесь к запущенному экземпляру MongoDB через клиент mongo shell и выполните базовые проверки.

1. Просмотр списка существующих баз данных командой show dbs.
2. Создание тестовой базы данных и коллекции командой use testdb, db.createCollection("testcollection").
3. Вставка простых документов в коллекцию.

Лабораторная работа №9. создание базы данных в технологии MongoDB

Цель: Приобретение практических навыков проектирования и создания базы данных в среде MongoDB, освоение основ документоориентированной модели данных и изучение механизмов работы с документами и коллекциями.

Теоретическая часть:

MongoDB - это документно-ориентированная база данных, хранящая данные в виде документов формата BSON (Binary JSON). Каждый документ представляет собой отдельный объект, содержащий произвольное количество пар ключ-значение.

Основные понятия MongoDB:

- Документ: Эквивалент строки в реляционных системах, хранит данные в парах ключ-значение.
- Коллекция: Группа документов, аналог таблицы в реляционных базах данных.
- Database: Логически изолированное пространство для множества коллекций.

Ключевыми преимуществами MongoDB являются:

- Гибкость модели данных.
- Высокопроизводительная архитектура.
- Поддержка географического распределения данных.

Создание базы данных в MongoDB начинается с выбора подходящей структуры документов и организации их в коллекции. Затем используются встроенные средства MongoDB для ввода, извлечения и модификации данных.

Открываем терминал и запускаем оболочку MongoDB:

```
mongo
```

Это откроет интерактивную сессию MongoDB.

Переходим в нужную базу данных. Если она ещё не существует, MongoDB создаст её автоматически при первом обращении:

```
use eshop
```

Команда use переключается на указанную базу данных, если она уже создана, или создаёт её автоматически.

3

Теперь создадим коллекцию users, где будут храниться данные о пользователях:

```
db.createCollection('users')
```

Можно сразу начать добавлять документы в коллекцию, даже не создавая её предварительно, поскольку MongoDB создает коллекцию автоматически при первой вставке документа.

Например, добавляем пять записей о пользователях:

```
db.users.insertMany([
   { "_id": 1, "username": "ivanov", "email": "ivan@example.com", "registrationDate": new Date("2023-05-01") },
   { "_id": 2, "username": "petrov", "email": "peter@example.com", "registrationDate": new Date("2023-06-15") },
   { "_id": 3, "username": "sidorova", "email": "anna@example.com", "registrationDate": new Date("2023-07-10") },
   { "_id": 4, "username": "kuznetsov", "email": "dmitry@example.com", "registrationDate": new Date("2023-08-20") },
   { "_id": 5, "username": "vasileva", "email": "elena@example.com", "registrationDate": new Date("2023-09-05") }
])
```

Теперь можем проверить наши данные и сделать выборки. Вот несколько примеров:

а) Показать всех пользователей:

```
db.users.find()
```

b) Найти заказы, общая стоимость которых превышает 1000 рублей:

```
db.orders.find({ totalCost: { $gt: 1000 } })
```

c) Показать конкретные поля в выводе:

```
db.orders.find({}, { _id: 1, productName: 1, totalCost: 1})
```

Задание на работу:

Создайте небольшую базу данных библиотеки, включающую следующие объекты:

- Книга (Book): название, автор, жанр, ISBN-код.
- Читатель (Reader): фамилия, имя, дата рождения, номер читательского билета.
- Журнал выдачи книг (Loan): книга, читатель, дата выдачи, срок возврата.

Реализуйте в базе минимум десять книг, пяти читателей и выдачу пяти книг читателям. Постройте два запроса:

1. Отбор всех выданных книг текущего месяца.
2. Отчет по числу выданных книг для каждого жанра.

Лабораторная работа №10. создание первой программы на языке java

Цель: Освоение основ синтаксиса и среды программирования Java, написание своей первой программы, знакомство с компилятором и виртуальной машиной Java (JVM).

Теоретическая часть:

Java - это высокоуровневый объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (ныне Oracle Corporation). Java широко применяется для разработки приложений различного уровня сложности благодаря своей платформе независимости («Write Once, Run Anywhere»).

Основные концепции Java:

- Платформа независимость: код Java компилируется в байт-код, исполняемый на любой платформе с установленной виртуальной машиной Java (JVM).
- Объектно-ориентированность: классы, объекты, наследование, инкапсуляция, полиморфизм и абстракция.
- Безопасность: встроенная защита от большинства уязвимостей и атак.
- Сборщик мусора: автоматическое управление памятью и освобождение ресурсов.

Простая программа на Java состоит из класса, метода main и инструкций внутри блока кода:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Привет, мир!");
    }
}
```

Задание на работу:

1. Загрузите и установите последнюю версию Java Development Kit (JDK) с официального сайта Oracle или OpenJDK.
2. Установите интегрированную среду разработки (IDE), такую как IntelliJ IDEA, Eclipse или NetBeans.
3. Запустив программу создайте новый проект
4. Напишите программу выводящую на экран сумму 2 введённых с клавиатуры чисел

Лабораторная работа №11.
использование условных и циклических
операторов в учебном проекте

Цель: Освоение конструкций ветвления и циклов в языках программирования на практике, разработка небольших учебных проектов с использованием условных операторов (if, switch) и циклов (for, while, do...while).

Теоретическая часть:

Условные операторы:

- Оператор if: позволяет выбрать путь выполнения программы в зависимости от истинности или ложности условия.

```
if (condition) {
   // Действия, если условие верно
} else {
   // Действия, если условие неверно
}
```

- Оператор if-else if-else: позволяет организовать множественный выбор путей выполнения.

```
if (condition1) {
   // Действия, если первое условие выполнено
} else if (condition2) {
   // Действия, если второе условие выполнено
} else {
   // Действия, если ни одно условие не выполнено
}
```

- Оператор switch: упрощает реализацию многофакторного выбора, сравнивая выражение с различными значениями.

```
switch (value) {
   case option1:
      // Действия при первом варианте
      break;
   case option2:
```

```
        // Действия при втором варианте
        break;
    default:
        // Действия, если не совпал ни один вариант
        break;
}
```

Циклы:

- Цикл for: удобен для перебора заранее известного количества итераций.

```
for (инициализация; условие; обновление) {
    // тело цикла
}
```

- Цикл while: выполняется, пока условие остаётся истинным.

```
while (condition) {
    // тело цикла
}
```

- Цикл do...while: гарантирует, что тело цикла выполнится хотя бы один раз.

```
do {
    // тело цикла
} while (condition);
```

Задание на работу:

Напишите программу, которая принимает от пользователя стоимость покупки и выводит итоговую сумму с учетом скидок:

- Покупка менее 1000 руб.: скидка 5%.
- Покупка от 1000 до 5000 руб.: скидка 10%.
- Покупка более 5000 руб.: скидка 15%.

Лабораторная работа №12. добавление методов в учебный проект

Цель: Научиться эффективно организовывать и применять методы в программах на Java, понимая важность модульности и повторного использования кода. Овладеть техникой разделения крупных задач на небольшие управляемые части с помощью методов.

Теоретическая часть:

Метод - это именованный фрагмент кода, предназначенный для выполнения одной или нескольких специфических задач. Методы помогают структурировать программы, уменьшая повторяемость кода и делая его более читабельным и поддерживаемым.
Любой метод имеет следующую структуру:

```
модификаторДоступа типВозвращаемогоЗначения имяМетода([списокПараметров]) {
   // тело метода
   return возвращаемоеЗначение; // если типВозвращаемогоЗначения != void
}
```

Здесь:

- Модификатор доступа контролирует доступность метода:
  - public - метод доступен всему миру.
  - protected - метод доступен в данном пакете и в дочерних классах вне пакета.
  - default (без спецификатора) - метод доступен только в пределах своего пакета.
  - private - метод доступен только внутри своего класса.
- Тип возвращаемого значения обозначает, какое значение метод вернет вызывающему коду.
  - Если метод ничего не возвращает, указывается void.
- Список параметров - это аргументы, которые принимаются методом. Параметры позволяют передать данные внутрь метода.

Важные моменты при создании методов:

1. Имя метода должно быть информативным и легко понимаемым.
2. Комментарии полезны для пояснений назначения метода и его поведения.
3. Исключения, которые метод может выбрасывать, объявляются с ключевым словом throws.

Типы методов:

- Методы без параметров и без возвращаемых значений (void):

```
public void greetUser() {
   System.out.println("Добро пожаловать!");
}
```

- Методы с параметрами и возвращаемым значением:

```
public double circleArea(double radius) {
   return Math.PI * radius * radius;
}
```

- Перегрузка методов (Overloading):Возможность объявления нескольких методов с одним и тем же именем, но разными списками параметров.

```
// Метод для двух целых чисел
public int addNumbers(int num1, int num2) {
   return num1 + num2;
}

// Метод для трех целых чисел
public int addNumbers(int num1, int num2, int num3) {
   return num1 + num2 + num3;
}
```

- Рекурсивные методы:Рекурсия возникает, когда метод вызывает сам себя, формируя цепочку вызовов. Важно предусмотреть условие завершения рекурсии.

```
public long factorial(long number) {
   if (number <= 1) {
      return 1;
   }
   return number * factorial(number - 1);
}
```

Основные приемы работы с методами:

- Абстрактные методы определяются в абстрактных классах и требуют переопределения в потомках.
- Интерфейсы определяют контракт, обязывающий реализовать методы.
- Методы экземпляра принадлежат объектам класса, тогда как статические методы относятся непосредственно к самому классу.

Задание на работу:

Модифицируйте учебную программу калькулятора, дополнив её новыми методами:

- метод для вычисления площади прямоугольника по двум сторонам;
- метод для перевода метров в километры;
- метод для проверки принадлежности числа к интервалу (меньше, равно или больше).

Протестируйте каждое изменение отдельно и убедитесь в правильности работы вашего решения.

Лабораторная работа №13. Включение класса в учебный проект

Цель: Приобретение навыков проектирования классов и включения их в структуру проекта на Java. Изучение принципа инкапсуляции и практики эффективного использования классов для улучшения качества и читаемости кода.

Теоретическая часть:

Класс в Java - это шаблон для объектов, определяющий свойства (атрибуты) и поведение (методы) объекта. Класс объединяет данные и операции над ними, образуя единую сущность.

Класс имеет следующую структуру:

```
class ClassName {
   // Поля (атрибуты)
   private тип поле1;
   private тип поле2;
```

```
    // Конструктор
    public ClassName(тип arg1, тип arg2) {
        this.поле1 = arg1;
        this.поле2 = arg2;
    }

    // Методы
    public тип метод1() {
        // логика метода
    }

    // Геттеры и сеттеры
    public тип getПоле1() {
        return поле1;
    }

    public void setПоле1(тип новоеЗначение) {
        поле1 = новоеЗначение;
    }
}
```

Ключевые элементы класса:

- Атрибуты (fields/роля): Хранят состояние объекта. Обычно приватные, защищаются принципом инкапсуляции.
- Конструкторы: Специальные методы, инициализирующие объект при его создании.
- Методы: Функциональные единицы, задающие поведение объекта.
- Геттеры и сеттеры: Методы для доступа и изменения значений атрибутов.

Принцип инкапсуляции:

Инкапсуляция - это механизм сокрытия внутренней реализации класса и предоставления внешнего интерфейса для взаимодействия с объектом. Атрибуты скрываются с помощью модификатора доступа private, а доступ осуществляется через публичные геттеры и сеттеры.

Преимущества инкапсуляции:

- Безопасность данных.
- Упрощение сопровождения и расширения программы.
- Возможность изменять внутреннюю реализацию без нарушения внешней совместимости.

Задание на работу:

1. Создайте класс Student, который будет представлять студентов учебного заведения. Класс должен содержать следующие атрибуты:
    - Имя (name)
    - Возраст (age)
    - Средний балл (averageScore)
2. Включите конструкторы для инициализации объектов класса.

3. Предоставьте геттеры и сеттеры для всех атрибутов.
4. Добавьте метод printInfo(), который выводит всю информацию о студенте на экран.

Лабораторная работа №14. Разработка приложения в соответствии с принципами объектно-ориентированного программирования (начальный этап).

Цель: Формирование базовых навыков проектирования и реализации приложений на языке Java с использованием ключевых концепций объектно-ориентированного программирования (ООП): инкапсуляции, наследования, полиморфизма и абстракции.

Теоретическая часть:

Основные принципы объектно-ориентированного программирования:

1. Инкапсуляция:Скрытие внутренних деталей реализации объекта и предоставление внешнего интерфейса для взаимодействия с ним.Атрибуты класса обычно защищены модификаторами доступа (private), а доступ обеспечивается через методы-геттеры и сеттеры.
2. Наследование:Механизм создания новых классов на основе существующих, наследующих свойства и поведение родительских классов. Используется для сокращения дублирования кода и достижения большей гибкости.
3. Полиморфизм:Способность объектов вести себя по-разному в зависимости от контекста. Например, одна и та же операция может быть выполнена по-разному для разных классов.
4. Абстракция:Умение выделить существенные признаки объекта и игнорировать несущественные детали. Абстрактные классы и интерфейсы служат основой для моделирования общей функциональности.

Этапы разработки ООП-приложения:

1. Анализ задачи:Определение потребностей и требований к приложению.
2. Проектирование классов:Разделение приложения на логические модули (классы), определение взаимосвязей между ними.
3. Реализация классов:Создание классов с соответствующими атрибутами и методами.
4. Тестирование и отладка:Проверка работы классов и выявление возможных ошибок.

Задание на работу:

1. Создайте класс Car, который описывает автомобиль.Этот класс должен содержать следующие атрибуты:
    - Марка автомобиля (brand, String)
    - Модель автомобиля (model, String)
    - Год выпуска (year, int)
    - Пробег (mileage, int)

2. Добавьте конструктор, который инициализирует все атрибуты при создании объекта.
3. Реализуйте геттеры и сеттеры для каждого атрибута.

Лабораторная работа №15.
Использование коллекций в учебном проекте

Цель: Изучение основных коллекций Java и приобретение навыков эффективного использования стандартных библиотек коллекций для хранения, обработки и манипулирования наборами данных в реальных проектах.

Теоретическая часть:

Коллекции Java представляют собой обобщённый механизм для работы с группами объектов. Библиотека Collections Framework в Java предоставляет широкий спектр готовых контейнеров для хранения данных, предлагая удобные способы работы с ними. Наиболее распространены следующие типы коллекций:
Основные типы коллекций:

1. List:Хранит упорядоченные коллекции объектов, допускающие повторяющиеся элементы. Основные реализации:
    - ArrayList: динамически растущий массив, быстрый доступ по индексу.
    - LinkedList: двусвязный список, быстрая вставка и удаление элементов.
2. Set:Собирает уникальные элементы (без дубликатов). Основные реализации:
    - HashSet: хранение элементов без учёта порядка.
    - TreeSet: сохраняет элементы в отсортированном порядке.
3. Map:Связывает ключи с соответствующими значениями. Основные реализации:
    - HashMap: быстрое обращение по ключу, хаотичный порядок элементов.
    - TreeMap: сортирует пары ключ-значение по ключам.

Характеристики коллекций:

- Порядок следования: Некоторые коллекции сохраняют порядок вставки элементов (ArrayList, LinkedList), другие - нет (HashSet, HashMap).
- Уникальность элементов: Set-коллекции обеспечивают уникальное содержимое, List-коллекции допускают повторы.
- Производительность: Каждая коллекция имеет свою специализацию (быстрая вставка, быстрый поиск, сохранение порядка и т.д.)

Задание на работу:

1. Создание списка учеников:Создайте список учеников школы, используя класс ArrayList<String> и наполните его элементами.
2. Работа с набором предметов:Создайте HashSet<String>, который хранит предметы школьной программы.

Лабораторная работа №16. Реализация параметризованного интерфейса в учебном проекте.

Цель: Освоение механизма реализации параметризованных интерфейсов в Java и развитие навыков работы с дженериками (generic types). Повышение уровня знаний о применении универсальных шаблонов для повышения гибкости и надежности кода.

Теоретическая часть:

Параметризованные интерфейсы (интерфейсы с дженериками) позволяют создавать универсальные контракты, которые работают с любыми типами данных. Вместо жесткого фиксирования конкретного типа данных интерфейс или класс может оперировать любым нужным типом, сохраняя сильную типизацию и безопасность.

Рассмотрим стандартный синтаксис интерфейса с дженериком:

```
interface InterfaceName<T> {
   void method(T param);
}
```

Где <T> - это тип-параметр, заменяющий реальный тип данных. Интерфейс можно реализовать следующим образом:

```
class ImplementationClass implements InterfaceName<Integer> {
   @Override
   public void method(Integer param) {
      // реализация метода
   }
}
```

Применение дженериков позволяет писать универсальный код, избавляя от необходимости дублировать одни и те же операции для разных типов данных.

Особенности дженериков:

- Универсальность: Генерирует единый интерфейс для любых типов данных.
- Безопасность типов: Во время компиляции контролируется тип передаваемых данных, что предотвращает ошибки несоответствия типов.
- Автоматическое приведение типов: Не требует ручного приведения типов, всё делается автоматически компилятором.

Задание на работу:

Предположим, вам необходимо разработать систему обработки платежей. Платежи могут поступать в различных валютах (рубли, доллары, евро и т.д.) или формах (наличные, банковские карты, электронные деньги и т.д.).

Создайте параметризованный интерфейс PaymentProcessor<T>, который содержит метод processPayment(T payment) для обработки платежа любого типа.

Лабораторная работа №17. Создание простого Spring Boot приложения

Цель: Изучение основ фреймворка Spring Boot и приобретение навыков быстрой разработки веб-приложений на Java. Освоение базовых принципов конфигурирования и развертывания Spring Boot приложений.

Теоретическая часть:

Spring Boot - это проект экосистемы Spring Framework, созданный для упрощения разработки приложений на Java. Он позволяет быстро создавать производительные микросервисы и полноценные серверные решения без излишней конфигурации и сложной настройки.

Основные преимущества Spring Boot:

- Автоматическое конфигурирование компонентов.
- Быстрое начало разработки новых проектов.
- Встроенные средства развертывания и тестирования.
- Широкий спектр готовых интеграций с популярными технологиями (например, базы данных, веб-сервисы).

Для начала работы с Spring Boot потребуется установить две основные среды:

- JDK версии 8 или выше.
- Инструменты сборки проекта: Maven или Gradle.

Создание базового проекта с Spring Initializr - удобной утилитой для быстрого старта. Это позволит автоматически выбрать необходимые зависимости и структуры папок вашего проекта.

Основные этапы разработки:

1. Настройка окружения (установка JDK, Maven/Gradle).
2. Генерация нового проекта через Spring Initializr.
3. Подключение необходимых зависимостей.
4. Реализация контроллера REST API.
5. Запуск и тестирование приложения.

Задание на работу:

Вам предстоит разработать многофункциональное веб-приложение на платформе Spring Boot, включающее следующее:

1. Проект должен быть организован в стиле *Model-View-Controller* (*MVC*) с чётким разделением ответственности между компонентами:
    - Модели данных (Models) отвечают за хранение и управление информацией.
    - Контроллеры (Controllers) обрабатывают запросы клиента и формируют ответ.
    - Сервисы (Services) выполняют основную логику приложения.
    - Репозитории (Repositories) обеспечивают связь с базой данных.

2. Необходимо спроектировать простую базу данных с двумя взаимосвязанными таблицами:
    - Таблица Пользователей (User) включает поля: уникальный идентификатор, имя пользователя, email и пароль.
    - Таблица Продуктов (Product) включает поля: уникальный идентификатор, название продукта, цену и ссылку на владельца продукта (связана с пользователями).
3. Нужно обеспечить базовые операции CRUD (Create, Read, Update, Delete) для продуктов. То есть вы сможете добавлять новые продукты, просматривать список существующих товаров, редактировать информацию о продукте и удалять ненужные записи.
4. Подключив модуль Spring Security, создайте механизм аутентификации и авторизации пользователей. Ваша система должна поддерживать два типа аккаунтов:
    - Обычные пользователи смогут просматривать продукты и покупать товары.
    - Администраторы получат дополнительные права: создание, изменение и удаление записей.
5. Вам необходимо написать автоматизированные тесты для покрытия основных функциональных возможностей вашего приложения. Эти тесты проверят функциональность контроллеров, сервисов и репозиториев.

Лабораторная работа №18. Работа с данными в Spring Boot

Цель: Ознакомление с методами работы с данными в приложениях на платформе Spring Boot. Изучение способов подключения и взаимодействия с различными источниками данных (базы данных, файлы и др.) с использованием фреймворков Spring Data и Hibernate.

Теоретическая часть:

Spring Boot обеспечивает удобные механизмы для работы с данными благодаря поддержке большого количества различных хранилищ данных, среди которых наиболее популярны реляционные базы данных (PostgreSQL, MySQL, SQLite и др.). Основным инструментом работы с этими хранилищами выступает библиотека Spring Data JPA совместно с ORM-платформой Hibernate.
Ключевые понятия:

- JPA (Java Persistence API) - стандартизированный интерфейс для объектно-реляционного отображения в Java.
- Hibernate - одна из популярных реализаций JPA, позволяющая гибко настраивать связь объектов и таблиц базы данных.
- Entity - объекты в приложении, представляющие собой сущности базы данных (таблицы).
- Repository - интерфейс для работы с объектами Entity, позволяющий осуществлять чтение, запись и обновление данных.

Типичные шаги работы с данными в Spring Boot:

1. Определение структуры базы данных:Проще всего начать с проектирования схем таблицы и связей между ними. Обычно используется аннотационный стиль описания сущностей (Entities).
2. Конфигурация доступа к базе данных:Важно правильно настроить источник данных (DataSource) и включить режим автоматической миграции (например, через Flyway или Liquibase).
3. Объявление репозиториев:Через специальные интерфейсы (наследование от стандартных интерфейсов Spring Data) определяются методы для работы с данными (выборка, вставка, обновление, удаление).

Service Layer (Сервисный слой):
Класс, содержащий бизнес-логику и взаимодействие с репозиториями.

```
@Service
public class UserService {
   private final UserRepository userRepository;

   public UserService(UserRepository userRepository) {
      this.userRepository = userRepository;
   }

   public List<User> findAll() {
      return Lists.newArrayList(userRepository.findAll());
   }
}
```

Controller (Контроллер):
Обрабатывает запросы HTTP и возвращает JSON-данные.

```
@RestController
@RequestMapping("/api/users")
public class UserController {
   private final UserService userService;

   public UserController(UserService userService) {
      this.userService = userService;
   }

   @GetMapping
   public ResponseEntity<List<User>> getUsers() {
      return new ResponseEntity<>(userService.findAll(), HttpStatus.OK);
   }
}
```

Подключение базы данных:

Настройка подключения осуществляется в файле application.properties или application.yml.
Пример для MySQL:

spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
spring.

Задание на работу:

1. Создать проект на Spring Boot с настройкой подключения к СУБД PostgreSQL или MySQL.
2. Реализовать простую сущность (Product) с полями:
    - ID продукта (автоинкремент),
    - Название,
    - Цена,
    - Количество товара.
3. Написать репозиторий для сущности Product, реализовав методы сохранения, получения списка товаров и поиска по названию.
4. Разработать REST-контроллеры для обработки GET, POST, PUT и DELETE запросов для продуктов.
5. Продемонстрировать выполнение основных операций CRUD (Create, Read, Update, Delete) над продуктом.
6. Проверить работоспособность приложения локально и убедиться, что операции выполняются корректно.

Лабораторная работа №19. разработка полного стека приложения, включая интеграцию с базами данных MySQL

Цель: Изучение принципов разработки многоуровневого веб-приложения на Java с использованием серверной части (backend) и клиентской части (frontend). Освоение способов интеграции приложения с реляционной базой данных MySQL с применением ORM-технологий и архитектуры MVC (Model-View-Controller).

Теоретическая часть:

Full stack development подразумевает разработку как серверной (back-end), так и клиентской (front-end) частей приложения. Такой подход требует знания широкого спектра технологий и подходов, начиная от языков программирования (Java, JavaScript) и заканчивая работой с базами данных (MySQL, PostgreSQL).
Основными этапами разработки full stack приложения являются:

- проектирование архитектуры приложения;
- выбор технологий и инструментария;
- реализация back-end и front-end;
- интеграция компонентов друг с другом;
- обеспечение надежности и безопасности.

2. Архитектурные шаблоны (Design Patterns)

Наиболее распространенным архитектурным шаблоном в современных проектах является MVC (Model-View-Controller). Это разделение ответственности помогает организовать код таким образом, чтобы упростить дальнейшее развитие и сопровождение приложения.

- Model (Модель) - хранит и управляет данными приложения.
- View (Вид) - отображает данные пользователю.
- Controller (Контроллер) - обрабатывает запросы пользователя и передает их соответствующим методам модели.

Такой подход минимизирует связь между уровнями и повышает гибкость приложения.

3. Язык программирования Java и фреймворк Spring Boot

Java - один из самых популярных языков программирования, широко применяемый в корпоративной среде благодаря своей многоплатформенности и стабильности. Одним из наиболее удобных фреймворков для разработки backend-приложений на Java является Spring Boot.

Основные характеристики Spring Boot:

- Автоконфигурация (autoconfiguration) - минимизирует необходимость настройки вручную.
- Легкий старт проектов - всего несколько шагов для запуска первого приложения.
- Возможность быстрого развертывания на серверах Tomcat, Jetty и других.
- Хорошее покрытие возможностей ORM (Object-Relational Mapping) через JPA и Hibernate.

4. Организация взаимодействия с базой данных MySQL

Одной из главных задач любого веб-приложения является хранение и обработка данных. В нашем проекте используется база данных MySQL, поддерживающая ACID-свойства транзакций и обладающая высоким уровнем производительности.

Интеграция с MySQL осуществляется следующим образом:

- Подключение драйвера MySQL (например, через Maven-зависимости).
- Настройка соединения с базой данных через свойства spring.datasource.*.
- Определение entity-классов для работы с моделями данных.
- Создание репозиториев (реализации интерфейса JpaRepository) для работы с данными.

5. Применение ORM (Object-Relational Mapping)

ORM (Object-Relational Mapping) - это способ преобразования объектов Java в реляционные структуры базы данных и наоборот. Наиболее популярным решением для Java является Hibernate, поддерживающий стандарт JPA (Java Persistence API).

Процесс взаимодействия выглядит примерно так:

- Создается Java-класс, соответствующий таблице в базе данных.
- Используются аннотации для описания полей объекта (например, @Entity, @Column, @Table).
- Через репозитории осуществляются операции CRUD (create, read, update, delete).

6. Методология разработки REST API

Современные веб-приложения активно используют REST (Representational State Transfer) архитектуру для организации взаимодействия между клиентами и сервером. Основное преимущество REST заключается в простоте и универсальности.

Особенности REST API:

- Интерфейс строится вокруг ресурсов, представленных уникальными URL-адресами.
- Данные передаются в JSON или XML форматы.
- Методы HTTP соответствуют операциям CRUD (GET, POST, PUT, DELETE).

7. Frontend-реализация (клиентская сторона)

Клиентская часть приложения включает разработку UI (интерфейса пользователя), который позволяет удобно управлять данными. Можно выбрать одну из популярных библиотек или фреймворков:

- Bootstrap: готовая система стилей и готовых элементов.
- Vue.js, Angular, React: современные фронтенд-фреймворки, позволяющие быстро строить динамические веб-приложения.

Простое решение может выглядеть как простая форма с кнопками для создания, редактирования и удаления записей, а также таблица для вывода имеющихся данных.

Задание на работу:

Для начала спроектируем схему нашей базы данных. Нам понадобится таблица для хранения информации о книгах. Таблица будет иметь следующую структуру:

Поле	Тип
id	INT PRIMARY KEY AUTO_INCREMENT
title	VARCHAR(255)
author	VARCHAR(255)
year	INT

Создаем скрипт для создания таблицы в MySQL:

```
CREATE TABLE books (
   id INT AUTO_INCREMENT PRIMARY KEY,
   title VARCHAR(255) NOT NULL,
   author VARCHAR(255) NOT NULL,
   year INT NOT NULL
);
```

Шаг 2. Настройка проекта на Spring Boot

Создаем новый проект на Spring Initializr (https://start.spring.io) с зависимостями Web, JPA и MySQL Driver. После скачивания архива импортируем проект в IDE (IntelliJ Idea или Eclipse).

Открываем файл application.properties (или application.yml) и настраиваем подключение к базе данных:

```
spring.datasource.url=jdbc:mysql://localhost:3306/your_database_name?useSSL=false&serve
rTimezone=UTC
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.jpa.hibernate.ddl-auto=update
```

spring.jpa.show-sql=true

Здесь мы указали адрес базы данных, учетные данные и настроили автоматическое обновление схемы базы данных при запуске приложения.

Шаг 3. Реализация Model

Создаем модель Book.java, которая описывает нашу книгу:

```
package com.example.bookstore.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Book {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long id;
   private String title;
   private String author;
   private int year;

   // Конструкторы, геттеры и сеттеры...
}
```

Шаг 4. Репозиторий и Сервис

Теперь реализуем интерфейс репозитория, расширяя стандартный интерфейс JpaRepository:

```
package com.example.bookstore.repository;

import com.example.bookstore.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> { }
```

И создадим сервис для работы с книгами:

java

Копировать

```
package com.example.bookstore.service;

import com.example.bookstore.model.Book;
import com.example.bookstore.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
```

```
@Service
public class BookService {
    private final BookRepository bookRepository;

    @Autowired
    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public List<Book> findAllBooks() {
        return bookRepository.findAll();
    }

    public Book saveBook(Book book) {
        return bookRepository.save(book);
    }

    public Book updateBook(long id, Book updatedBook) {
        if (!bookRepository.existsById(id)) throw new RuntimeException("Книга не найдена");
        updatedBook.setId(id); // Сохраняем тот же id
        return bookRepository.save(updatedBook);
    }

    public void deleteBook(long id) {
        bookRepository.deleteById(id);
    }
}
```

Шаг 5. Контроллер для REST API

Создаем контроллер, который будет обрабатывать запросы к нашим данным:

```
package com.example.bookstore.controller;

import com.example.bookstore.model.Book;
import com.example.bookstore.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/books")
public class BookController {
    private final BookService bookService;
```

```
    @Autowired
    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @GetMapping
    public List<Book> getAllBooks() {
        return bookService.findAllBooks();
    }

    @PostMapping
    public Book createBook(@RequestBody Book book) {
        return bookService.saveBook(book);
    }

    @PutMapping("/{id}")
    public Book updateBook(@PathVariable long id, @RequestBody Book updatedBook) {
        return bookService.updateBook(id, updatedBook);
    }

    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable long id) {
        bookService.deleteBook(id);
    }
}
```

Шаг 6. Frontend Часть (Пример на HTML+CSS+Javascript)

Можно создать простой HTML интерфейс для взаимодействия с приложением. Пример формы для добавления книги:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Add a New Book</title>
</head>
<body>
<h1>Добавьте новую книгу:</h1>
<form action="/add-book" method="post">
    Название:<br><input type="text" name="title"><br>
    Автор:<br><input type="text" name="author"><br>
    Год выпуска:<br><input type="number" name="year"><br>
    <button type="submit">Добавить книгу</button>
</form>
</body>
</html>
```

Также потребуется обработчик на стороне JavaScript для отправки AJAX-запросов и получение данных от REST API.

Лабораторная работа №20. связывание базы данных с java приложением

Цель: Ознакомиться с основными способами интеграции базы данных с Java-приложением, изучить технологию JDBC (Java Database Connectivity) и научиться осуществлять операции CRUD (Create, Read, Update, Delete) с использованием простых SQL-запросов и подготовленных выражений.

Теоретическая часть:

Подключение к базе данных из Java-приложения осуществляется с помощью технологии JDBC (Java Database Connectivity). JDBC - это стандартный API для работы с реляционными базами данных (RDBMS), позволяющий разработчикам писать универсальный код для работы с разными системами баз данных (например, MySQL, Oracle, PostgreSQL).

JDBC (Java Database Connectivity) -
стандартный API в Java для взаимодействия с реляционными базами данных. Позволяет Java-приложению подключаться к СУБД, отправлять SQL-запросы и обрабатывать полученные результаты.

Основные компоненты JDBC:

1. DriverManager - управляет драйверами JDBC и устанавливает соединение с БД.
2. Connection - представляет открытое соединение с базой данных.
3. Statement / PreparedStatement - используется для выполнения SQL-запросов.
4. ResultSet - хранит результаты SELECT-запроса.
5. SQLException - исключение, возникающее при ошибках работы с БД.

Пошаговый процесс подключения и работы:

1. Добавление драйвера JDBC. Для работы с конкретной СУБД (MySQL, PostgreSQL и т. д.) требуется соответствующий JDBC-драйвер -
   JAR-файл, который нужно добавить в classpath проекта. Например, для MySQL это mysql-connector-java.
2. Регистрация драйвера. В современных версиях JDBC драйвер регистрируется автоматически. Ранее требовалось явно вызывать Class.forName("com.mysql.cj.jdbc.Driver").
3. Установление соединения. Используется метод DriverManager.getConnection(url, username, password).
   - URL подключения имеет формат: jdbc:<subprotocol>://<host>:<port>/<database>.
   - Пример для MySQL: jdbc:mysql://localhost:3306/mydb.
4. Создание запроса. Через объект Connection создаётся Statement или PreparedStatement.
5. Выполнение запроса:
   - executeQuery() - для SELECT (возвращает ResultSet).

- executeUpdate() -
  для INSERT, UPDATE, DELETE (возвращает количество изменённых строк)
  .

6. Обработка результатов. Для ResultSet используется цикл while (rs.next()), внутри ко торого данные извлекаются методами getInt(), getString() и т. д.
7. Закрытие ресурсов. Важно закрыть ResultSet, Statement и Connection в блоке finally или с использованием конструкции try-with-resources
   .
   Безопасность и лучшие практики:

- Используйте PreparedStatement вместо Statement для предотвращения SQL-инъекци й.
- Всегда закрывайте соединения и ресурсы.
- Обрабатывайте исключения типа SQLException.

Задание на работу:

1. Создать простую базу данных MySQL с одной таблицей (например, таблица пользователей).
2. Написать Java-консольное приложение, выполняющее следующие операции:
   - Подключается к базе данных MySQL.
   - Выполняет запросы типа SELECT для вывода всех записей таблицы.
   - Добавляет новую запись в таблицу (используя INSERT).
   - Обновляет одну из существующих записей (используя UPDATE).
   - Удаляет одну из записей (используя DELETE).

Основная литература:

1. Алдан, А. Введение в генерацию программного кода / А. Алдан. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2026. - 189 с. : схем. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428825

2. Соколов В.П. Кодирование в системах защиты информации [Электронный ресурс]:
учебное пособие/ Соколов В.П., Тарасова Н.П.- Электрон. текстовые данные.- М.: Московский технический университет связи и информатики, 2026.- 94 с.- Режим доступа: http://www.iprbookshop.ru/61485.html.- ЭБС «IPRbooks»

3. Антимиров, В. М. Проектирование аппаратуры систем автоматического управления. В 2 ч. Ч. 1 : учебное пособие для СПО / В. М. Антимиров. - 2-е изд. - Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2022. - 92 с. - ISBN 978-5-4488-04014, 978-5-7996-2834-5. - Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. - URL: http://www.iprbookshop.ru/87852.html. - Режим доступа: для авторизир.
пользователей

Дополнительная литература:

1. Федорова Г.И. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности. Учебное пособие. Изд.: КУРС, Инфра-М. Среднее профессиональное образование. 2026 г. 336 стр.