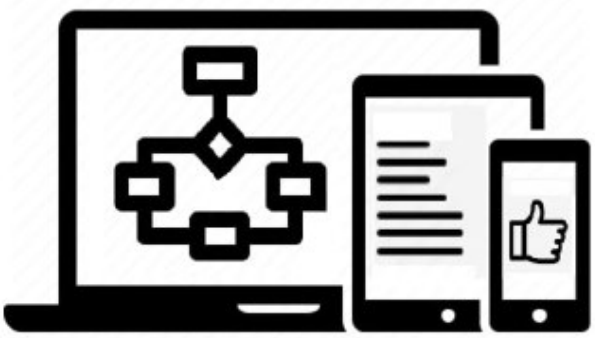
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические рекомендации к самостоятельной работе для студентов специальности 1-28 01 02 «Электронный маркетинг» заочной форм обучения



Могилев 2022

УДК 004.7
ББК 32.973.202
П79

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «04» марта 2022 г., протокол № 9

Составители: канд. техн. наук, доц. С. К. Крутолевич;
канд. техн. наук, доц. Э. И. Ясюкович;
канд. техн. наук, доц. К. В. Захарченков;
ст. преподаватель Ю. В. Вайнилович

Рецензент Ю. С. Романович

Методические рекомендации содержат основные базовые теоретические сведения, некоторые приемы реализации задач, а также практические задания для выполнения самостоятельных работ по всем темам курса.

Учебно-методическое издание

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Ответственный за выпуск	В. В. Кутузов
Корректор	Т. А. Рыжикова
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Целью учебной дисциплины «Проектирование информационных систем» является изучение возможностей в области проектирования информационных систем, рассмотрение теоретических и практических основ управления проектами, принципов и моделей представления проектных и архитектурных решений, методов проектирования и концепции вызова удаленных процедур в области распределённых технологий.

Самостоятельные работы № 1–5 посвящены изучению языка визуального моделирования UML. Данный язык используется для разработки архитектуры информационной системы или технического задания на разработку.

Самостоятельные работы № 6 и 7 позволяют изучить основы создания таблиц для хранения данных и язык SQL-запросов к базе данных.

Самостоятельная работа № 8 посвящена изучению языка html и каскадной таблицы стилей CSS.

Язык html (Hyper Text Markup Language) используется для добавления разметки в обычный текст. Это позволяет создавать статистические и динамические сайты и является языком, описывающим структуру и семантику веб-документа.

Каскадные таблицы стилей (Cascading Style Sheets – CSS) влияют на отображение страниц в окнах браузеров (цвета, шрифты, фоновые изображения, интервалы между строками, отступы, границы, эффекты и даже анимация элементов). Благодаря CSS можно производить изменения, относящиеся ко всем страницам сайта, редактируя при этом лишь один файл таблицы стилей.

Самостоятельная работа № 9 позволяет ознакомиться с языком XML.

Самостоятельная работы № 10 позволяет изучить основы распределенных технологий на примере технологии CORBA.

Технология CORBA (Common Object Request Broker Architecture) – это стандарт написания распределенных приложений, предложенный консорциумом OMG (Open Management Group). Создавая CORBA-объекты, мы можем, например, существенно уменьшить время решения задач, требующих выполнения большого объема вычислений. Это возможно благодаря размещению CORBA-объектов на разных машинах. Каждый удаленный объект решает определенную подзадачу, тем самым разгружает клиента от выполнения лишней работы.

Самостоятельная работа № 11 позволяет студенту закрепить навыки в разработке веб-приложений.

1 Самостоятельная работа № 1. Общая характеристика CASE-средства Enterprise Architect

Цель работы: изучение основных возможностей и режимов работы CASE-средства Enterprise Architect.

Теоретические сведения

При запуске Enterprise Architect первое, что отображается, – это стартовая страница. Эта страница предоставляет следующие возможности.

Search – поиск объектов в Enterprise Architect. Введите имя объекта в текстовое поле и нажмите кнопку.

Getting Started открывает Tasks Pain. Отображает полезные темы и руководства для различных областей деятельности в Enterprise Architect.

Online Resources & Tutorials открывает страницу веб-сайта компании Sparx Systems, который предоставляет доступ к широкому кругу уроков по языку UML, демонстраций, примеров, надстроек и обсуждений.

Configure Options отображает диалоговое окно Options, которое позволяет определить, каким образом Enterprise Architect отображает и обрабатывает информацию.

Open a Project File отображает диалоговое окно Open Project, которое используется для открытия существующего проекта.

Create a New Project позволяет сохранить новый проект.

Copy a Base Project позволяет копировать базовый проект.

Connect to Server служит для выбора имени источника данных для подключения.

Браузер проекта (Project Browser) позволяет перемещаться по пространству проекта Enterprise Architect. В нем отображаются пакеты, диаграммы, элементы и их свойства. Можно перетаскивать элементы из папки в папку или из браузера прямо на диаграмму. Браузер проекта служит для управления и представления всех элементов в модели. Браузер проекта можно разделить на представления, каждое из которых содержит диаграммы, пакеты и другие элементы.

Представление вариантов использования включает модели вариантов использования и бизнес-процессов.

Динамическое представление включает диаграммы состояний, деятельности и последовательности. Логическое представление включает модель классов.

Представление компонентов – это представление системных компонентов (исполняемые, DDL и другие компоненты).

Представление размещения – физическая модель, которая показывает, какие использованы технические средства ЭВМ и какое программное обеспечение установлено.

Панель инструментов Enterprise Architect представляется как панель с иконками, которые используются для создания элементов и связей между ними. К тому же родственные элементы и связи организованы в страницы. Каждая

страница содержит элементы и связи, используемые в конкретном типе диаграммы.

Главное меню Enterprise Architect обеспечивает управляемый мышью доступ ко многим функциям, связанным с жизненным циклом проекта наряду с функциями администрирования.

Меню инструментов (Tools Menu) обеспечивает доступ к различным настройкам, общим для генерации кода, управления EAP, файлами, настройкам правописания, внешних ресурсов, настройкам таких функций, как управление ярлыками.

В меню инструментов наибольший интерес представляет пункт Options. Здесь находятся опции настройки Enterprise Architect для отображения и работы с моделями и элементами модели. Основные режимы представлены в таблице 1.1.

Таблица 1.1 – Назначение вкладок пункта меню Options

Вкладка	Назначение вкладки
General	Общие настройки проекта, например, автор, адрес домашнего веб-сайта, общие настройки браузера проекта
Standard Colors	Позволяет установить цвет ряда объектов и их фон
New Diagram Defaults	Позволяет конфигурировать опции для новых диаграмм и общего поведения диаграммы
Diagram Appearance	Позволяет определить, как диаграммы и их содержимое отображается на экране
Diagram Behavior	Позволяет определить, как диаграмма реагирует на действия, производимые над ней
Diagram Sequence	Позволяет установить настройки шрифта и фокус контрольного индикатора для диаграммы последовательности
Objects	Позволяет установить, как элементы выглядят на диаграмме
Links	Настройки создания, поведения и нотации связей
Communication Colors	Позволяет установить цвета, используемые в диаграммах взаимодействия
Source Code Engineering	Описывает общие настройки, применяемые для всех языков при генерации кода из Enterprise Architect
C++	Установка опций для генерации кода на C++

Задание

Освоить приемы работы в среде Enterprise Architect.

Контрольные вопросы

1 Перечислить основные диаграммы для моделирования АСОИ.
2 Перечислить основные элементы экрана.
3 Дать характеристику браузеру.
4 Дать характеристику окну диаграмм.
5 Дать характеристику четырем представлениям модели.

2 Самостоятельная работа № 2. Разработка диаграммы вариантов использования

Цель работы: изучение технологии формирования диаграммы вариантов использования.

Теоретические сведения

Создание действующих лиц в среде Enterprise Architect.

Чтобы поместить действующее лицо:

– щелкните левой кнопкой мыши на панели инструментов Toolbox по кнопке Actor;

– щелкните по рабочей области диаграммы, в появившемся окне введите имя актера, выберите стереотип.

Создание вариантов использования в среде Enterprise Architect.

Чтобы поместить вариант использования на диаграмму, щелкните левой кнопкой мыши на панели инструментов Toolbox по кнопке Use Case.

Чтобы получить доступ к диаграмме вариантов использования, необходимо сделать следующее.

Рядом с представлением вариантов использования в браузере щелкните на значке « + », это приведет к открытию данного представления.

Дважды щелкните на диаграмме, чтобы открыть её.

Для создания новой диаграммы вариантов использования щелкните правой кнопкой мыши на пакете представления вариантов использования в браузере.

Из всплывающего меню выберите пункт Add > Add Diagram.

Выделив новую диаграмму, введите ее имя.

Дважды щелкните на названии этой диаграммы в браузере, чтобы открыть ее.

Построение диаграммы вариантов использования.

Откройте диаграмму вариантов использования.

Чтобы поместить действующее лицо или вариант использования на диаграмму, перетащите его мышью из браузера на диаграмму вариантов использования.

Прикрепление файла к варианту использования.

Щелкните правой кнопкой мыши на варианте использования.

В открывшемся меню выберите пункт Properties.

Перейдите на вкладку Files.

Укажите путь к ранее созданному файлу (File Path) и нажмите на кнопку Save, чтобы прикрепить файл к варианту использования.

Нажмите кнопку Launch, чтобы открыть прикрепленный файл.

Удаление вариантов использования и действующих лиц.

Существует два способа удалить элемент модели – из одной диаграммы или из всей модели.

Чтобы удалить элемент модели из диаграммы:

– выделите элемент на диаграмме;

– нажмите на клавишу Delete или нажмите сочетание клавиш CTRL+ D.

Обратите внимание, что хотя элемент и удален с диаграммы, он остался в браузере и на других диаграммах системы.

Чтобы удалить элемент из модели:

– выделите элемент на диаграмме;

– выберите пункт меню Delete.

Задание

Разработать диаграммы вариантов использования по индивидуальному заданию.

Контрольные вопросы

1 Цель диаграммы.

2 Дать определение терминам «вариант использования», «действующее лицо».

3 Виды связей между вариантами использования и действующими лицами.

3 Самостоятельная работа № 3. Анализ бизнес-процессов приложения

Цель работы: приобретения навыков описания основных требований к АСОИ.

Теоретические сведения

На каждый вариант использования, представленный на диаграмме вариантов использования, должен быть представлен бизнес-процесс. Сложные бизнес-процессы удобно раскладывать сначала на макрошаги, а потом уже делать детальное описание каждого шага.

Макрошаги – это простая укрупненная последовательность действий в ходе выполнения бизнес-процессов.

При детальном описании конкретного макрошага бизнес-процесса необходимо указать:

– **кто** выполняет бизнес-процесс;

– **что он делает** в рамках этого бизнес-процесса;

– **какие данные являются входом** для этого действия **или что является результатом** этого действия;

– **бизнес-правила**, которые описывают, как он это делает.

Это такая блок-схема, которая разделена на дорожки. Каждая дорожка посвящена одному бизнес-пользователю.

Элементы модели объекта, которые мы размещаем при моделировании на диаграммах, можно собрать в отдельных пакетах модели. В итоге у нас получается полный список справочников и документов, которые участвуют в нашем проекте.

В результате на диаграмме необходимо отразить следующее.

Модель ролей – для того, чтобы посмотреть, что делает конкретный пользователь. При ее описании в диаграмму новые элементы уже не добавляются, а используются те, которые добавили в рамках диаграммы вариантов использования. Также здесь удобно дорожками обозначать те профили информационной системы, благодаря которым пользователь получит права к нужным ему объектам.

Модель метаданных включает сами метаданные и связанные с ними функции. На диаграмме видно, что делают эти метаданные и как они между собой взаимодействуют.

Модель функций. После построения модели метаданных можно смоделировать алгоритмы обработки метаданных.

Задание

Разработать диаграммы бизнес-процессов по индивидуальному заданию.

Контрольные вопросы

1 Цель диаграммы.

2 Модели каких сущностей отображаются на диаграмме.

4 Самостоятельная работа № 4. Разработка диаграммы классов на уровне сущностей

Цель работы: приобретение навыков разработки диаграммы классов.

Теоретические сведения

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов состоит из множества элементов, которые отражают декларативные знания о предметной области.

Графически класс изображается в виде прямоугольника, который разделен горизонтальными линиями на секции. В этих секциях указывается имя класса, атрибуты (переменные) и операции (методы).

Обязательным элементом обозначения класса является его имя. Имя класса должно быть уникальным в пределах проекта.

Атрибут класса служит для представления отдельного свойства или признака. Атрибуты класса записываются во второй сверху секции прямоугольника класса. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости, имени, его кратности, типа значений атрибута и, возможно, его исходного значения.

Общий формат записи отдельного атрибута класса следующий:

<квантор видимости> <имя атрибута> [кратность]:<тип атрибута> = <исходное значение> {строка-свойство}

Квантор видимости может принимать одно из четырех возможных значений и, соответственно, отображается при помощи специальных символов:

1) символ «+» – тип общедоступный;
2) символ «#» – тип защищенный;
3) символ «—» – тип закрытый;
4) символ «~» – тип пакетный.

Кванторы видимости отображают доступность данных атрибута для других классов.

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса.

Тип атрибута определяется типом данных, определенным в пакете «Типы данных» языка UML или разработчиком.

Исходное значение служит для задания некоторого начального значения для соответствующего атрибута.

Строка-свойство служит для указания дополнительных свойств атрибута.

Операция – это некоторый сервис, который предоставляет класс. Операции класса записываются в третьей сверху секции прямоугольника класса, поэтому эту секцию часто называют секцией операций. Каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени, выражения типа возвращаемого значения.

Общий формат записи отдельной операции класса следующий:

<квантор видимости> <имя операции> (список параметров): <выражение типа возвращаемого значения>

Имя операции представляет собой строку текста, которая используется в качестве идентификатора соответствующей операции и поэтому должна быть уникальной в пределах данного класса.

Список параметров является перечнем разделенных запятой формальных параметров.

Выражение типа возвращаемого значения указывает на тип данных значения, которое возвращается объектом после выполнения соответствующей операции.

Задание

Разработать диаграммы классов по индивидуальному заданию.

Контрольные вопросы

1 Цель диаграммы.
2 Стереотипы классов.
3 Связи между классами.

5 Самостоятельная работа № 5. Разработка диаграммы состояний и редактирование свойств ее элементов

Цель работы: изучение технологии создания диаграммы состояний.

Теоретические сведения

Главное предназначение этой диаграммы – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы. Состояния будет ассоциироваться с интерфейсами пользователя.

Состояние на диаграмме изображается прямоугольником с закругленными вершинами. Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния. В противном случае в первой из них записывается имя состояния, а во второй – список некоторых внутренних действий или переходов в данном состоянии.

В качестве имени состояний удобно использовать имя интерфейсов.

Для состояний необходимо указать некоторые действия, которые выполняются в этом состоянии. Для этой цели служит дополнительная секция в обозначении состояния или примечание, содержащая перечень внутренних действий или деятельность.

Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

<метка действия '/ ' выражение действия>

Перечень меток действий в языке UML фиксирован:

entry – указывает на то, что следующее за ней выражение действия должно быть выполнено в момент входа в данное состояние (входное действие);

exit – указывает на то, что следующее за ней выражение действия должно быть выполнено в момент выхода из данного состояния (выходное действие);

do – специфицирует некоторую деятельность (do activity) или так называемую ду-деятельность, которая выполняется в течение всего времени, пока объект находится в данном состоянии;

include – используется для обращения к подсостоянию, при этом следующее за ней выражение действия содержит имя этого подсостояния.

Начальное состояние (initial state) представляет собой частный случай состояния, которое не содержит никаких внутренних действий. Оно служит для

указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний.

Конечное состояние (final state) представляет собой частный случай состояния, которое также не содержит никаких внутренних действий.

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Над переходом необходимо указать событие, которое и вызвало этот переход.

Задание

Разработать диаграмму состояний по индивидуальному заданию.

Контрольные вопросы

1 Цель диаграммы.
2 Понятие состояний.
3 Деятельность. Входное и выходное действие.
4 Изображение состояний и событий на диаграмме.

6 Самостоятельная работа № 6. Разработка структуры базы данных пользователя

Цель работы: изучение технологии разработки структуры базы данных.

Теоретические сведения

Разработка таблиц базы данных.

Для создание новой таблицы используется кнопка Table на панели инструментов. После добавления таблицы на диаграмме открывается окно ее свойств. В поле Name введите имя таблицы и в поле Database выберите тип базы данных.

Для добавления колонок таблицы и редактирования их свойств используется пункт Attributes. В открывшемся окне введите название колонки. В окне Data Type выберите тип данных. В окне Initial при необходимости введите начальное значение, которое может быть использовано как значение по умолчанию для колонки.

Если колонка представляет первичный ключ для таблицы, установите флажок Primary Key.

Enterprise Architect может генерировать простые DDL-скрипты для создания таблиц в вашей модели.

Связи между классами (ассоциации) определяются на основе диаграмм взаимодействия. Если два объекта взаимодействуют (обмениваются сообщениями), между ними должна существовать связь (путь взаимодействия). Для ассоциаций задаются множественность и, возможно, направление навигации. Могут

использоваться множественные ассоциации, агрегации и классы ассоциаций.

Связи создают непосредственно на диаграмме классов. Панель инструментов диаграммы классов содержит кнопки для создания как одно-, так и двунаправленных связей. Чтобы на диаграмме классов создать связь, необходимо нажать на панель инструментов, выбрать кнопку с соответствующей связью. Затем необходимо провести мышью линию связи от одного класса к другому.

Задание

Разработать структуру базы данных по индивидуальному заданию.

Контрольные вопросы

1 Как создаются таблицы на диаграмме классов?
2 Через какие поля происходит связь между таблицами?

7 Самостоятельная работа № 7. Язык запросов SQL

Цель работы: изучить основы построения запросов к базам данных на языке SQL.

Теоретические сведения

1 Простейшие SELECT-запросы.

Оператор SELECT (выбрать) языка SQL является самым важным и самым часто используемым оператором. Он предназначен для *выборки* информации из таблиц базы данных. Упрощенный синтаксис оператора SELECT выглядит следующим образом:

SELECT [DISTINCT] <список атрибутов>
FROM <список таблиц>
[WHERE <условие выборки>]
[ORDER BY <список атрибутов>]
[GROUP BY <список атрибутов>]
[HAVING <условие>]
[UNION <выражение с оператором SELECT>J;

Ключевое слово SELECT сообщает базе данных, что данное предложение является запросом на ***извлечение*** информации.

После слова SELECT через запятую перечисляются ***наименования полей*** (список атрибутов), содержимое которых запрашивается.

Обязательным ключевым словом в предложении-запросе SELECT является слово FROM (из). За ключевым словом FROM указывается список разделенных запятыми имен таблиц, из которых извлекается информация. Любой SQL-запрос должен заканчиваться символом **≪ ;≫** (*точка с запятой*).

Например,

SELECT NAME,SURNAME FROM STUDENT;

Приведенный запрос осуществляет выборку всех значений полей NAME и SURNAME из таблицы STUDENT.

2 Выборка данных (оператор SELECT).

Например, запрос на вывод записей о предметах, на изучение которых отводится количество часов, находящееся в пределах между 30 и 40, имеет вид:

SELECT * FROM SUBJECT

WHERE HOUR BETWEEN 30 AND 40;

Граничные значения, в данном случае значения 30 и 40, *входят* во множество значений, с которыми производится сравнение. Оператор BETWEEN может использоваться как для числовых, так и для символьных типов полей.

Оператор LIKE применим только к символьным полям типа CHAR или VARCHAR. Этот оператор просматривает строковые значения полей с целью определения, входит ли заданная в операторе LIKE подстрока (образец поиска) в символьную строку-значение проверяемого поля.

Для выборки строковых значений по заданному образцу подстроки можно применять шаблон искомого образца строки, использующий следующие символы:

– символ подчеркивания ≪_≫, указанный в шаблоне, определяет возможность наличия в указанном месте *одного любого* символа;

– символ ≪%≫ допускает присутствие в указанном месте проверяемой строки последовательности любых символов произвольной длины.

Пример – Написать запрос, выбирающий из таблицы STUDENT сведения о студентах, фамилии которых начинаются на букву ≪P≫.

SELECT *

FROM STUDENT

WHERE SURNAME LIKE 'P%';

В случае необходимости включения в образец самих символов ≪_≫ и ≪%≫ применяют так называемые *escape-символы.* Если escape-символ предшествует знаку ≪_≫ и ≪%≫, то эти знаки будут восприниматься буквально. Например, можно задать образец поиска с помощью следующего выражения:

LIKE '__P' ESCAPE 'V.

В этом выражении символ 'V с помощью ключевого слова ESCAPE объявляется escape-символом. Первый символ ≪_≫ в заданном шаблоне поиска '__P' будет соответствовать, как и ранее, любому символу в проверяемой строке. Однако второй символ ≪_≫, следующий после символа 'V, объявленного escape-символом, уже будет интерпретироваться буквально как обычный символ, так же как и символ 'P' в заданном шаблоне.

Обращаем внимание на то, что рассмотренные выше операторы сравнения ≪=, <, >, <=, >=, <>≫ и операторы IN, BETWEEN и LIKE ни в коем случае нельзя использовать для проверки содержимого поля на наличие в нем пустого значения NULL. Для этих целей предназначены специальные операторы is NULL (является пустым) и IS NOT NULL (является непустым).

3 Числовые, символьные и строковые константы.

Несмотря на то, что SQL работает с данными в понятиях строк и столбцов таблиц, имеется возможность применения значений выражений, построенных с использованием встроенных функций, констант, имен столбцов, определяемых как своего рода виртуальные столбцы. Они помещаются в списке столбцов и могут сопровождаться псевдонимами.

Если в запросе вместо спецификации столбца SQL обнаруживает *число*, то оно интерпретируется как *числовая константа.*

Символьные константы должны указываться в одинарных кавычках. Если одинарная кавычка должна выводиться как часть строковой константы, то ее нужно предварить другой одинарной кавычкой.

Например, запрос

```
SELECT 'фамилия', SURNAME, 'Имя', NAME, 100
FROM STUDENT;
```

4 Арифметические операции для преобразования числовых данных.

Унарный (одиночный) оператор ≪–≫ (знак минус) изменяет знак числового значения, перед которым он указан, на противоположный.

Бинарные операторы ≪+≫, ≪–≫, ≪*≫ и ≪/≫ предоставляют возможность выполнения арифметических операций сложения, вычитания, умножения и деления.

Например, запрос

```
SELECT SURNAME, NAME, STIPEND, -(STIPEND*KURS)/2
FROM STUDENT
WHERE KURS = 4 AND STIPEND > 0;
```

5 Операция конкатенации строк.

Операция конкатенации ≪||≫ позволяет соединять (склеивать) значения двух или более столбцов символьного типа или символьных констант в одну строку.

Эта операция имеет синтаксис <значимое символьное выражение > {||} <значимое символьное выражение>.

Например:

```
SELECT SURNAME ||'_'|| NAME, STIPEND
FROM STUDENT
WHERE KURS = 4 AND STIPEND > 0;
```

6 Функции преобразования символов в строке.

LOWER – перевод в строчные символы (нижний регистр) LOWER (<строка>).

UPPER – перевод в прописные символы (верхний регистр) UPPER (<строка>).

INITCAP – перевод первой буквы каждого слова строки в прописную (заглавную) INITCAP (<строка>).

Например:

```
SELECT LOWER (SURNAME), UPPER (NAME)
FROM STUDENT
WHERE KURS = 4 AND STIPEND > 0;
```

7 Строковые функции.

LPAD – дополнение строки слева LPAD (<строка>, <длина> [,<подстрока>]).

<строка> дополняется **слева** заданной в <подстроке> последовательностью символов до указанной <длины> (возможно, с повторением последовательности):

– если <подстрока> не указана, то по умолчанию <строка> дополняется пробелами;

– если <длина> меньше длины <строки>, то исходная <строка> усекается слева до заданной <длины>.

RPAD – дополнение строки справа RPAD (<строка>, <длина> [,<подстрока>]).

<строка> дополняется **справа** заданной в <подстроке> последовательностью символов до указанной <длины> (возможно, с повторением последовательности):

– если <подстрока> не указана, то по умолчанию <строка> дополняется пробелами;

– если <длина> меньше длины <строки>, то исходная <строка> усекается справа до заданной <длины >.

8 Преобразование вывода и встроенные функции.

LTRIM – удаление левых граничных символов LTRIM (<строка>[,<подстрока>]):

– из <строки> удаляются слева символы, указанные в <подстроке>;

– если < подстрока > не указана, по умолчанию удаляются пробелы;

– если в <строку> справа добавляется столько пробелов, сколько символов слева было удалено, то длина <строки> остается неизменной.

RTRIM – удаление правых граничных символов RTRIM (<строка> [,<подстрока>]:

– из <строки> удаляются справа символы, указанные в <подстроке>;

– если <подстрока> не указана, по умолчанию удаляются пробелы;

– если в <строку> слева добавляется столько пробелов, сколько символов справа было удалено, то длина <строки> остается неизменной.

Функции LTRIM и RTRIM рекомендуется использовать при написании условных выражений, в которых сравниваются текстовые строки. Дело в том, что наличие начальных или конечных пробелов в сравниваемых операндах может исказить результат сравнения.

Например, константы **' AAA'** и **'AAA '** не равны друг другу.

SUBSTR – выделение подстроки.

SUBSTR (<строка>, <начало> [,<количество>]):

– из <строки> выбирается заданное <количество> символов, начиная с указанной параметром <начало> позиции в строке;

– если <количество> не задано, символы выбираются с <начала> и до конца <строки>;

– возвращается подстрока, содержащая число символов, заданное параметром <количество>, либо число символов от позиции, заданной параметром <начало> до конца *строки;*

– если указанное <начало> превосходит длину <строки>, то возвращается строка, состоящая из пробелов. Длина этой строки будет равна заданному <количеству> или исходной длине <строки> (при незаданном <количестве>).

INSTR – поиск подстроки.

INSTR (<строка>, <подстрока> [,<начало поиска> [,<номер вхождения>]]):

– <начало поиска> задает начальную позицию в строке для поиска <подстроки>. Если не задано, то по умолчанию принимается значение 1;

– <номер вхождения> задает порядковый номер искомой подстроки. Если не задан, то по умолчанию принимается значение 1;

– значимые выражения в <начале поиска> или в <номере вхождения > должны иметь беззнаковый целый тип или приводиться к этому типу;

– тип возвращаемого значения – INT; функция возвращает позицию найденной подстроки.

LENGTH – определение длины строки (<строка>):

– длина <строки>, тип возвращаемого значения;

– функция возвращает NULL, если <строка> имеет NULL-значение.

Примеры запросов, использующих строковые функции:

```
SELECT LPAD (SURNAME, 10, '@'), RPAD (NAME, 10, '$')
FROM STUDENT
WHERE KURS = 3 AND STIPEND > 0;
SELECT SUBSTR(NAME, 1, 1) Ц'.' || SURNAME, CITY, LENGTH (CITY,)
FROM STUDENT
WHERE KURS IN(2, 3, 4)AND STIPEND > 0;
```

9 Функции работы с числами.

ABS – абсолютное значение. ABS (<значимое числовое выражение>).

FLOOR – урезает значение числа с плавающей точкой до наибольшего целого, не превосходящего заданное число. FLOOR (<значимое числовое выражение>).

CEIL – самое малое целое, равное или большее заданного числа. CEIL (<значимое числовое выражение>).

Функция округления – ROUND. ROUND (<значимое числовое выражение>,<точность>) аргумент <точность> задает точность округления.

Функция усечения – TRUNC. TRUNC (<значимое числовое выражение>,<точность>).

Тригонометрические функции – COS, SIN, TAN.

COS (<значимое числовое выражение>).

SIN (<значимое числовое выражение>).
TAN (<значимое числовое выражение>).
Экспоненциальная функция – EXP.
EXP (<значимое числовое выражение>).
Логарифмические функции – LN, LOG.
LN (<значимое числовое выражение>).
LOG (<значимое числовое выражение>).
Функция возведения в степень – POWER.
POWER (<значимое числовое выражение>,<экспонента>).
Определение знака числа – SIGN.
SIGN (<значимое числовое выражение>).
Вычисление квадратного корня – SQRT.
SQRT (<значимое числовое выражение>).

Пример запроса
SELECT UNIVNAME, RATING, ROUND (RATING, -1), TRDNC (RATING, FROM UNIVERSITY;

10 Функции преобразования значений.

Преобразование в символьную строку – TO_CHAR. TO_CHAR (<значимое выражение>[,<символьный формат>]):

– <значимое выражение > – числовое значение или значение типа дата – время;

– для числовых значений <символьный формат> должен иметь синтаксис [S]9[9][,9[9]], где S – представление знака числа (при отсутствии предполагается без отображения знака); 9 – представление цифр-знаков числового значения (для каждого знакоместа). Символьный формат определяет вид отображения чисел. По умолчанию для числовых значений используется формат '999999.99';

– для значений типа дата – время <символьный формат> имеет вид (то вид отображения значений даты и времени):

а) в части даты
'DD-Mon-YY'
'DD-Mon-YYYY'
'MM/DD/YY'
'MM/DD/YYYY1
'DD.MM.YY'
'DD.MM.YYYY'

б) в части времени
'HH24'
ИН24.Ш'
'HH24:MI:SS'
'HH24:MI:SS.FF'
HH24 – часы в диапазоне от 0 до 24;
MI – минуты;

SS – секунды;

FF – сотые доли секунды.

При выводе времени в качестве разделителя по умолчанию используется двоеточие (:), но при желании можно использовать любой другой символ.

Возвращаемое значение – символьное представление <значимого выражения> в соответствии с заданным <символьным форматом> преобразования.

Преобразование из символьного значения в числовое – TO_NUMBER.

TO_NUMBER (<значимое символьное выражение >).

При этом <значимое символьное выражение> должно задавать символьное значение числового типа.

Преобразование символьной строки в дату.

TCMDATE (<значимое символьное выражение > [,<символьный формат>]):

– <значимое символьное выражение> должно задавать символьное значение типа **дата – время**;

– <символьный формат> должен описывать представление значения типа дата – время в <значимом символьном выражении>. Допустимые форматы (в том числе и формат по умолчанию) приведены выше.

Возвращаемое значение – <значимое символьное выражение> во внутреннем представлении. Тип возвращаемого значения.

Над значениями типа DATE разрешены следующие операции:

– бинарная операция сложения;

– бинарная операция вычитания.

В бинарных операциях один из операндов должен иметь значение отдельного элемента даты: только год, или только месяц, или только день.

Например:

– при добавлении к дате '22.05.1998' пяти лет получится дата '22.05.2003';

– при добавлении к этой же дате девяти месяцев получится дата '22.02.1999';

– при добавлении 10 дней получим '01.06.1998'.

При сложении двух полных дат, например, '22.05.1998' и '01.12.2000', результат непредсказуем.

Пример запроса

```
SELECT SURNAME,NAME,BIRTHDAY,
TO_CHAR (BIRTHDAY, 'DD-MON-YYYY'),
TO_CHAR (BIRTHDAY, 'DD.MM.YY')
FROM STUDENT;
```

11 Агрегирование и групповые функции.

Агрегирующие функции позволяют получать из таблицы сводную (агрегированную) информацию, выполняя операции над группой строк таблицы. Для задания в SELECT-запросе агрегирующих операций используются следующие ключевые слова:

– COUNT – определяет количество строк или значений поля, выбранных посредством запроса и не являющихся NULL-значениями;

– SUM – вычисляет арифметическую сумму всех выбранных значений данного поля;

– AVG – вычисляет среднее значение для всех выбранных значений данного поля;

– MAX – вычисляет наибольшее из всех выбранных значений данного поля;

– MIN – вычисляет наименьшее из всех выбранных значений данного поля.

В SELECT-запросе агрегирующие функции используются аналогично именам полей, при этом последние (имена полей) используются в качестве аргументов этих функций.

Функция AVG предназначена для подсчета среднего значения поля на множестве записей таблицы.

Например, для определения среднего значения поля MARK (оценки) по всем записям таблицы EXAM_MARKS можно использовать запрос с функцией AVG следующего вида:

```
SELECT AVERAGE (MARK)
FROM EXAM_MARKS;
```

Для подсчета общего количества строк в таблице следует использовать функцию COUNT со звездочкой.

```
SELECT COUNT(*)
FROM EXAM_MARKS;
```

Аргументы DISTINCT и ALL позволяют, соответственно, исключать и включать дубликаты обрабатываемых функцией COUNT значений, при этом необходимо учитывать, что при использовании опции ALL значения NULL все равно не войдут в число подсчитываемых значений.

```
SELECT COUNT(DISTINCT SUBJ_ID)
FROM SUBJECT;
```

Предложение GROUP BYGROUP BY (группировать по) позволяет группировать записи в подмножества, определяемые значениями какого-либо поля, и применять агрегирующие функции уже не ко всем записям таблицы, а раздельно к каждой сформированной группе.

Предположим, требуется найти максимальное значение оценки, полученной каждым студентом. Запрос будет выглядеть следующим образом:

```
SELECT STUDENT_ID, MAX(MARK)
FROM EXAM_MARKS
GROUP BY STUDENT_ID;
```

Выбираемые из таблицы EXAM_MARKS записи группируются по значениям поля STUDENT__ID, указанного в предложении GROUP BY, и для каждой группы находится максимальное значение поля MARK. Предложение GROUP BY позволяет применять агрегирующие функции к каждой группе, определяемой общим значением поля (или полей), указанного в этом предложении. В приведенном запросе рассматриваются группы записей, сгруппированные по идентификаторам студентов.

В конструкции GROUP BY для группирования может быть использовано более одного столбца.

Например:

```
SELECT STUDENT_ID, SUBJ_ID, MAX (MARK)
FROM EXAM_MARKS
GROUP BY STUDENT_ID, SUBJ_ID;
```

В этом случае строки вначале группируются по значениям первого столбца, а внутри этих групп – в подгруппы по значениям второго столбца. Таким образом, GROUP BY не только устанавливает столбцы, по которым осуществляется группирование, но и указывает порядок разбиения столбцов на группы.

Следует иметь в виду, что в предложении GROUP BY должны быть указаны все выбираемые столбцы, приведенные после ключевого слова SELECT, кроме столбцов, указанных в качестве аргумента в агрегирующей функции.

При необходимости часть сформированных с помощью GROUP BY групп может быть исключена с помощью предложения HAVING.

Предложение HAVING определяет критерий, по которому группы следует включать в выходные данные, по аналогии с предложением WHERE, которое осуществляет это для отдельных строк.

```
SELECT SUBJ_NAME, MAX(HOUR)
FROM SUBJECT
GROUP BY SUBJ_NAME
HAVING MAX (HOUR) >= 72;
```

В условии, задаваемом предложением HAVING, указывают только поля или выражения, которые на выходе имеют единственное значение для каждой выводимой группы.

Задание

Сформировать не менее пяти запросов к базе данных.

Контрольные вопросы

1 Перечислите структуру запроса SQL.
2 Как создаются условия отбора данных для различных типов данных?
3 Для чего используются агрегатные функции?

8 Самостоятельная работа № 8. Разработка простейшего интернет-приложения

Цель работы: изучение языка HTML при создании интернет-приложения.

Теоретические сведения

Большинство веб-страниц содержат описание разметки гипертекста на языке HTML5, основным элементом которого является тег, представляющий собой правило, согласно которому информация отображается в окне браузера.

Для создания и редактирования веб-страниц используются текстовые редакторы, такие как ***Notepad++***, ***Sublime Text3***, ***Geany***, ***Vim***, ***Emacs*** и др. Простейшая структура ***html***-страницы имеет вид:

```
<!DOCTYPE html>
<html>
   <head>
<meta charset="utf-8"
<title>  </title>
   </head>
   <body>
         содержание html-страницы
   </body>
</html>
```

Первым разделом ***html***-страницы является раздел DOCTYPE, определяющий правила, содержащиеся в файле объявления типа документа.

Парный тег <***html***> </***html***> описывает корневой элемент ***html***-страницы, информирует браузер о том, что внутри него содержится код web-страницы, содержащий два раздела: ***head*** и ***body***. Раздел ***head*** содержит информацию о способе представления данных и стилях, о исполняемых скриптах, а также данные для поисковых систем. Раздел ***body*** – это тело документа, содержащее теги ***html***-страницы. В HTML5 присутствие тегов <***html***>, <***head***> и <***body***> не является обязательным.

Язык HTML5 содержит примерно 100 различных тегов, предоставляющих широкий спектр возможностей для работы с текстом и мультимедиа.

Теги могут содержать дополнительные свойства, называемые атрибутами – параметрами, влияющими на их работу. Параметры тегов перечисляются после имени тега через пробел с определенными значениями, записываемыми в кавычках:

<тег атрибут$_1$="значение" атрибут$_2$="значение" ...>

Атрибуты могут записываться в любом порядке.

У тегов есть собственные атрибуты, но существуют также глобальные атрибуты, доступные всем тегам, например, ***style***, ***id***, ***class***. Так, тег <***body***> обладает рядом атрибутов, задающих глобальные параметры фона, отступов, текста и гиперссылок.

В HTML5 реализовано немало новых тегов, позволяющих повысить эффективность обработки сайта поисковыми системами, например, тегов контейнеров <***header***>, <***footer***>, <***aside***> и <***nav***>.

Теги <***header***> и <***footer***> представляют область верхнего и нижнего информационного блока. В теге <***header***> располагаются логотипы, основные разделы сайта, элементы управления, поиска, регистрации и т. д., а в теге <***footer***> может содержаться контактная информация.

Тег <***aside***> удобно использовать для оформления блока меню, боковой панели или поясняющего раздела, а тег <***nav***> – навигации по сайту.

Часто содержимое сайта можно рассматривать как набор статей похожей структуры и оформления. Для таких целей удобно использовать теги <***main***>, <***article***> и <***section***>.

Для создания и изменения стилей элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, используются каскадные таблицы стилей CSS.

Основы синтаксиса CSS. Если в HTML ключевым элементом являлся тег, то в CSS таким элементом является селектор, представляющий собой некоторое имя стиля, для которого добавляются параметры форматирования. В качестве селектора могут выступать теги, атрибуты тегов, классы и идентификаторы.

Теги-селекторы позволяют определять оформление тегов, а селекторы атрибутов используются для задания их стилей. Классы и идентификаторы повышают гибкость использования стилей и делают стиль независимым от конкретного тега.

Общий способ записи селектора на примере тега <***h1***> имеет вид: селектор (***h1***) – свойство (***background***) – значение свойства ***(#EEE***).

```
h1 {
  background: #EEE;
  …
}
```

Селектор сообщает браузеру, какой элемент необходимо форматировать, а блок объявления (код в фигурных скобках), содержит форматирующие команды, указывающие свойства со значениями.

CSS не чувствителен к регистру, переносу строк, пробелам и символам табуляции, поэтому форма записи зависит от вкуса разработчика. Хорошим считается стиль, когда каждое свойство указывается на отдельной строке, но если в селекторе только один стиль, его часто записывают в одну строчку:

```
body { background: #F5E2D0; }
```

В CSS3 можно выделить следующие виды стилей, определяющие способ их подключения к основному документу:

– встроенные;

– внутренние;

– связанные.

Каждый из этих стилей имеет определенные преимущества в плане гибкости использования и эффективности загрузки страницы.

Листинг 1 – html-код страницы

```
<!DOCTYPE html>
<html>
<head>
<title>Меню</title>
<link rel="stylesheet" type="text/css" href="css/style.css" />
</head>
<body>
<div id="mainmenu">
  <ul id="nav">
<li id="settings">
<a href="#"><img src="images/set-tings.png"></a>
</li>
  <li><a href="">Пункт-1</a>
    <ul>
      <li><a href="#">Команда 1-1</a></li>
      <li><a href="#">Команда 1-2</a></li>
    </ul>
  </li>
 </ul>
</div>
</body>
</html>
```

Листинг 2 – CSS-код страницы

```
#mainmenu  {
  float:left;  }
#mainmenu ul  {
  margin: 10;/* отступ ленты меню сверху*/
  padding: 5;  /* отступ ленты меню слева*/
  list-style: none;  }
#mainmenu ul li  {
  position: relative;
  float:left;  }
#mainmenu ul li ul, #mainmenu ul li ul li  {
  width:130px;
  color:red;  /*  */  }
#mainmenu li ul  {
  position: absolute;
  left: 0;
  top: 29px;
  display: none;
  float:left;  }
#mainmenu ul li a  {
  float:left;
  color:blue; /*Цвет текста глав меню  */
  padding: 10px 0 10px 0;
  text-align:center;
  background:  #00f0f0;/*цвет фона гл  меню*/
 }
#mainmenu li ul li a  {
  padding:5px 0 3px 10px;
  text-align:left;  font-size:12px;
  width:80px;
  background: #EEEEEE;/*Цвет поля группы выбр-го подменю*/  }
#mainmenu li ul li a:hover  {
  background:  #0ffff0;  /*Цвет фона подменю */
  color:blue; }
#settings a {  /*Графич-й эл-т */
  padding: 18px;
  height: 19px;  /*Высота графич эл-та */
  font-size:  10px;  line-height: 24px; }
* html #mainmenu ul li {float: left; height: 1%;}
* html #mainmenu ul li a { height: 1%; }
```

```
    width:80px;        /*Длина ленты глав меню */
    font-size:16px;
    #mainmenu li:hover ul,
    #mainmenu li.over ul { display: block; }
```

Задание

Разработать страницу веб-приложения.

Контрольные вопросы

1 Перечислите основные разделы html-страницы.
2 Для чего используются CSS-страницы?

9 Самостоятельная работа № 9. Язык XML. Основные методы, модели и средства передачи и обработки файлов XML

Цель работы: изучить основы разметки страницы XML.

Теоретические сведения

Синтаксис XML.

Для ограничения тегов в разметке XML, так же как и в HTML, используются угловые скобки: тег начинается со знака «меньше» (<) и завершается знаком «больше» (>). Но необходимо помнить, что в отличие от HTML вся разметка XML чувствительна к регистру символов, это касается как имен тегов, так и значений атрибутов.

Имена.

В языке XML все имена должны начинаться с буквы, символа нижнего подчеркивания (_) или двоеточия (:) и продолжаться только допустимыми для имен символами, а именно: они могут содержать только буквы, входящие в секцию букв кодировки Unicode, арабские цифры, дефисы, знаки подчеркивания, точки и двоеточия. Однако имена не могут начинаться со строки xml в любом регистре. Имена, начинающиеся с этих символов, зарезервированы для использования консорциумом W3C.

Структура XML-документа.

Любой XML-документ состоит из следующий частей:

− необязательный пролог;
− тело документа;
− необязательный эпилог, следующий за деревом элементов.

Рассмотрим каждую из частей более подробно.

Пролог.

Пролог состоит из нескольких частей.

Необязательное объявление XML (XML Declaration).

Объявление заключено между символами и содержит:

− пометку xml и номер версии (version) спецификации XML;

− указание на кодировку символов (encoding), в которой написан документ (по умолчанию encoding="UTF-8");

− параметр standalone, который может принимать значения "yes" или "no" (по умолчанию standalone="yes"). Значение "yes" показывает, что в документе содержатся все требуемые декларации элементов, а "no" - что нужны внешние определения DTD.

Все это вместе может выглядеть следующим образом:

<?xml version ="1.0" encoding-"windows-1251" standalone="yes"?>

Важно отметить, что в объявлении XML только атрибут version является обязательным, все остальные атрибуты могут быть опущены и, следовательно, принимать значения по умолчанию. Также нужно помнить, что все эти атрибуты следует указывать только в приведенном выше порядке.

Комментарии.

Инструкции по обработке.

Назначение инструкций по обработке − сообщить информацию, передаваемую XML-процессором приложению. Инструкция по обработке имеет следующую общую форму записи:

<? Кому инструкция ?>

Здесь Кому есть имя приложения, которому адресована инструкция. Допускается любое имя при соблюдении следующих правил:

1) имя должно начинаться с буквы или символа подчеркивания (_), после чего могут следовать или не следовать другие буквы, цифры, точки (.), тире (−) или символы подчеркивания (_);

2) имя «xml», в любом сочетании строчных или прописных букв, зарезервировано («xml» строчными буквами используется в объявлении XML-документа, которое представляет собой разновидность инструкции по обработке).

Инструкция − информация, передаваемая приложению. Она может состоять из любой последовательности символов, за исключением пары ?>, зарезервированной для обозначения окончания инструкции по обработке.

Например, следующая инструкция по обработке предписывает браузеру использовать CSS-таблицу из файла styles.css:

<?xml-stylesheet type="text/css" href=" styles.css"?>

Символы пустых пространств.

Необязательное объявление типа документа, DTD (Document Type Declaration), которое заключено между символами и может занимать несколько строк. В этой части объявляются теги, использованные в документе, или приводится ссылка на файл, в котором записаны такие объявления.

После объявления типа документа также могут следовать комментарии, команды обработки и символы пустых пространств.

Поскольку все эти части необязательны, пролог может быть опущен.

Тело документа.

Тело документа состоит из одного или больше элементов. В правильно оформленном XML-документе элементы формируют простое иерархическое де-

рево, в котором обязательно присутствует корневой элемент (root element), в который вложены все остальные элементы документа. Имена элементов должны быть уникальны в пределах документа. Имя корневого элемента считается именем всего документа и указывается во второй части пролога после слова Doctype.

Элемент начинается открывающим тегом, затем идет необязательное содержимое элемента, после чего записывается закрывающий тег (в отличие от HTML, наличие закрывающего тега обязательно, исключением являются элементы без содержания, так называемые пустые элементы, которые могут быть записаны в сокращенной форме).

Другие элементы.

Символьные данные.

Ссылки на символы. Для того чтобы вставить в текст документа некоторый символ, который, например, не присутствует в раскладке клавиатуры либо может быть неправильно истолкован анализатором, используют ссылки на символы. Ссылка на символ обязательно начинается со знака "&" (амперсанта) и заканчивается точкой с запятой. Ссылки на символы записываются в следующем виде:

&# код_символа_в_Unicode;

Код символа можно записать и в шестнадцатеричном виде. В этом случае перед ним ставится символ "x": &#x

Шестнадцатеричный_код_символа;

Ссылки на сущности. Ссылки на сущности позволяют включать любые строковые константы в содержание элементов или значение атрибутов. Ссылки на сущности, как и ссылки на символы, начинающиеся с амперсанта, после которого идет имя сущности, и заканчивающиеся точкой с запятой:

&имя_сущности;

Ссылки на сущности указывают программе-анализатору подставить вместо них строку символов, заранее заданную в определении типа документа (DTD).

Комментарии.

Если надо вставить в текст документа комментарий либо сделать какой-то фрагмент «невидимым» для программы-анализатора, то его оформляют следующим образом:

Разделы CDATA

Секция CDATA используется для того, чтобы задать область документа, которую при разборе анализатор будет рассматривать как простой текст, игнорируя любые инструкции и специальные символы. Программа-анализатор не разбивает секцию CDATA на элементы, а считает ее просто набором символов. В отличие от комментариев, содержание данной секции не игнорируется, а передается без изменений на выход программы-анализатора, благодаря чему его можно использовать в приложении.

Инструкции по обработке.

Инструкции по обработке содержат указания программе-анализатору документа XML. Инструкции по обработке заключаются между символами <? и ?>. Сразу за начальным вопросительным знаком записывается имя программного модуля, которому предназначена инструкция. Затем через пробел идет сама инструкция, передаваемая программному модулю. Сама инструкция – это обычная

строка. Примером инструкции по обработке может служить строка объявления XML:

< ?xml version="1.0" encoding="windows-1251"?>

Эта инструкция предназначена программе, обрабатывающей документ XML. Инструкция передает ей номер версии и кодировку, в которой записан документ.

Атрибуты.

Открывающие теги либо теги пустых элементов в XML могут содержать атрибуты, представляющие собой пару имя=значение. В одном открывающем теге разрешается использовать только один экземпляр имени атрибута. Атрибуты могут содержать ссылки на объекты, ссылки на символы, текстовые символы. В отличие от языка HTML, в XML значения атрибутов обязательно надо заключать в апострофы ('), либо в кавычки ("). Таким образом, атрибут может быть записан в одном из двух форматов:

имя_атрибута="значение_атрибута"

имя_атрибута='значение_атрибута'

Атрибуты используются для того, чтобы связать некоторую информацию с элементом, а не просто включить ее в содержание последнего. Однозначного ответа на вопрос «Что лучше выбрать – элемент или атрибут?» не существует. Каждый выбирает то, что ему больше нравится. Атрибуты удобно использовать для описания простых значений или для указания типа элемента. Например, мы можем ввести в открывающий тег атрибут type (который может принимать одно из значений: город, поселок, деревня). Тогда данный тег может выглядеть следующим образом: Новосибирск.

<city type="город"> Новосибирск </city>

Эпилог.

В эпилог XML могут входить комментарии, инструкции по обработке и/или пустое пространство.

В языке XML не определен индикатор конца документа, большинство приложений для этой цели используют завершающий тег корневого элемента документа. Таким образом, встретив завершающий тег корневого элемента, скорее всего, приложение закончит обработку документа и эпилог обработан не будет.

Правильно оформленные и валидные документы.

В общем случае XML- документы должны удовлетворять следующим требованиям.

Любой XML-документ должен всегда начинаться с инструкции, внутри которой также можно задавать номер версии языка, номер кодовой страницы и другие параметры, необходимые программе-анализатору в процессе разбора документа.

Теги (метаданные) в документе выделяются символами <> (угловые скобки). Название тега должно начинаться сразу после угловой скобки (пробелы между открывающей угловой скобкой и названием тега недопустимы). Каждый открывающий тег, определяющий некоторую область данных в документе, обязательно должен иметь своего закрывающего «напарника», нельзя опускать закрывающие теги.

В XML учитывается регистр символов.

Все значения атрибутов, используемых в определении тегов, должны быть заключены в кавычки.

Вложенность тегов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тегов.

Вся информация, располагающаяся между начальным и конечными тегами, рассматривается в XML как данные, и поэтому учитываются все символы форматирования (т. е. пробелы, переводы строк, табуляции не игнорируются, как в HTML).

Если XML-документ не нарушает приведенные правила, то он называется формально-правильным и все анализаторы, предназначенные для разбора XML-документов, смогут работать с ним корректно.

Задание

Разработать XML-документ.

Контрольные вопросы

1 Назовите структуру XML-документа.

2 Для чего используются теги?

10 Самостоятельная работа № 10. Разработка интернет-приложения на основе технологии CORBA

Цель работы: освоить приемы разработки интернет-приложений на основе технологии CORBA.

Теоретические сведения

Технология CORBA (Common Object Request Broker Architecture) – это стандарт написания распределенных приложений, предложенный консорциумом OMG (Open Management Group). Создавая CORBA-объекты, мы можем, например, существенно уменьшить время решения задач, требующих выполнения большого объема вычислений. Это возможно благодаря размещению CORBA-объектов на разных машинах. Каждый удаленный объект решает определенную подзадачу, тем самым разгружает клиента от выполнения лишней работы.

Основу CORBA составляет объектный брокер запросов (Object Request Broker). ORB управляет взаимодействием объектов в распределенной сетевой среде. IIOP (Internet Inter-ORB Protocol) – это специальный протокол взаимодействия между ORB.

В адресном пространстве клиента функционирует специальный объект, называемый заглушкой (stub). Получив запрос от клиента, он упаковывает параметры запроса в специальный формат и передает его серверу, а точнее − скелету.

Скелет (skeleton) – объект, работающий в адресном пространстве сервера. Получив запрос от клиента, он распаковывает его и передает серверу. Также скелет преобразует ответы сервера и передает их клиенту (заглушке).

Для того чтобы написать любое приложение CORBA, используя технологию Java, необходимо иметь две вещи – это установленный пакет JDK1.5 и компилятор idlj (…\jdk1.5.0\bin\idlj.exe). JDK предоставляет набор классов для работы с CORBA-объектами, а idlj производит отображение языка IDL в Java.

10.1 Создание простейшего CORBA-приложения

10.1.1 Написание интерфейса.

Создание CORBA-приложения на Java начинают с написания интерфейса для удаленного объекта, используя язык описания интерфейсов (Interface Definition Language, IDL).

Создадим файл hello.idl:

module HelloApp{interface Hello{string sayHello();oneway void shutdown();};};

Данный интерфейс описывает лишь два метода shutdown и **sayHello** . Причем нам не важно, что делают эти методы, главное мы определяем, что они есть, и определяем, какие у них входные и выходные параметры.

Далее следует запустить компилятор IDL-to-Java idlj:

idlj – fall Hello.idl

В текущей директории появилась новая папка Hello App , в которой содержатся шесть java-файлов. Каждый из них имеет свое назначение.

HelloPOA.java java – абстрактный класс, который представляет собой не что иное, как скелет сервера (skeleton) и обеспечивает функциональность сервера.

HelloStub.java – класс, реализующий заглушку (stub) клиента. Обеспечивает функциональность клиента.

HelloHelper.java и HelloHolder.java – классы, предоставляющие вспомогательные функции для CORBA-объектов.

HelloOperations.java – класс, содержащий описание интерфейса hello на языке Java.

Hello.java – класс – наследник HelloOperations, поддерживающий интерфейс org.omg.CORBA. Object.

10.1.2 Создание сервера.

Теперь наша задача – написать класс, реализующий интерфейс hello. В нашем случае это будет HelloImpl . Обратите внимание на то, что он является наследником класса HelloPOA . В HelloImpl реализованы методы, объявленные в Hello.idl.

Для упрощения задачи объявление методов можно взять из файла HelloOperations.java , сгенерированного jdlj .

class HelloImpl extends HelloPOA {private ORB orb; public void setORB (ORB orb_val) {orb = orb_val;} // implement sayHello() methodpublic String sayHello() {return «\nHello world!!\n»;} // implement shutdown() methodpublic void shutdown() {orb.shutdown(false);}}

Следующим шагом будет создание собственно серверной части приложения. Это будет класс HelloServer.

В нем будет всего один метод – стандартная функция main.

Первое, что мы делаем, создаем ORB. Затем создаем экземпляр класса удаленного объекта (HelloImpl) и регистрируем его в ORB. Дальше вызываем специальную службу имен (NameService) и регистрируем в ней имя удаленного объекта, чтобы клиент смог его найти.

Рассмотрим подробнее эти этапы.

1 Создание и инициализация ORB. Производится вызовом статического метода init класса ORB:

ORB orb = ORB.init (args, null);

2 Создание экземпляра класса удаленного объекта и регистрация его в ORB:

HelloImpl helloImpl = new HelloImpl();

helloImpl.setORB(orb);

3 Получение контекста имен (NamingContext):

org.omg.CORBA. Object objRef = orb.resolve_initial_references («NameService»);

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

В первой строчке мы получаем объектную ссылку на службу имен (NameService). Но фактически это обыкновенный CORBA-объект и для того, чтобы использовать его как контекст имен (NamingContext), необходимо вызвать метод narrow класса NamingContextHelper , который как бы конкретизирует данный CORBA-объект.

4 Регистрация имени удаленного объекта (HelloImpl):

String name = «Hello»;

NameComponent path[] = ncRef.to_name(name);

ncRef.rebind (path, href);

Регистрация имени производится для того, чтобы клиент смог найти удаленный объект. Этой цели служит функция rebind (NameComponent[] nc, Object obj) интерфейса NamingContext.

5 Ожидание запросов от клиента:

orb.run();

Теперь сервер готов к работе.

// HelloServer.javaimport HelloApp.*;import org.omg. CosNaming.*;import org.omg. CosNaming. NamingContextPackage.*;import org.omg.CORBA.*;import org.omg. PortableServer.*;import org.omg. PortableServer.POA;import java.util. Properties;class HelloImpl extends HelloPOA {private ORB orb;public void setORB (ORB orb_val) {orb = orb_val;} // implement sayHello() methodpublic String sayHello() {return «\nHello world!!\n»;} // implement shutdown() methodpublic void shutdown() {orb.shutdown(false);}}

public class HelloServer {

```
public static void main (String args[]) {
try {
// create and initialize the ORB
ORB orb = ORB.init (args, null);
// get reference to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow (orb.resolve_initial_references («Root-POA»));
rootpoa.the_POAManager().activate();
// create servant and register it with the ORB
HelloImpl helloImpl = new HelloImpl();
helloImpl.setORB(orb);
// get object reference from the servant
org.omg.CORBA. Object ref = rootpoa.servant_to_reference(helloImpl);
Hello href = HelloHelper.narrow(ref);
// get the root naming context
// NameService invokes the name service
org.omg.CORBA. Object objRef =
orb.resolve_initial_references («NameService»);
// Use NamingContextExt which is part of the Interoperable
// Naming Service (INS) specification.
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
// bind the Object Reference in Naming
String name = «Hello»;
NameComponent path[] = ncRef.to_name(name);
ncRef.rebind (path, href);
System.out.println («HelloServer ready and waiting…»);
// wait for invocations from clients
orb.run();
}
catch (Exception e) {
System.err.println («ERROR:» + e);
e.printStackTrace (System.out);
}
System.out.println («HelloServer Exiting…»);
}
}
```

10.1.3 Создание клиента.

Перейдем к написанию кода для клиента.

Основные шаги написания клиентского приложения.

1 Создание и инициализация ORB.

2 Получение контекста службы имен (NamingContext).

3 Нахождение удаленного объекта.

4 Вызов метода sayHello.

5 Вызов метода shutdown.

Как видно, первые два пункта совпадают с этапами создания серверного приложения, поэтому рассматривать их не будем.

Третий пункт реализуется тоже достаточно просто. Создается объект NameComponent. Вызывается метод resolve (NameComponent[] path), который отыскивает по имени удаленный объект (стандартный CORBA-объект). При помощи метода narrow (org.omg.CORBA. Object obj) класса helloHelper (сгенерированного idlj-компилятором) получаем объектную ссылку на интерфейс hello.

```
String name = «Hello»;
helloImpl = HelloHelper.narrow (ncRef.resolve_str(name));
```

Теперь можно вызывать метод sayHello:

```
System.out.println (helloImpl.sayHello());
```

Метод shutdown завершает работы сервера.

```
helloImpl.shutdown();
//testClient.java
import HelloApp.*;
import org.omg. CosNaming.*;
import org.omg. CosNaming. NamingContextPackage.*;
import org.omg.CORBA.*;
public class HelloClient
{
static Hello helloImpl;
public static void main (String args[])
{
try {
// create and initialize the ORB
ORB orb = ORB.init (args, null);
// get the root naming context
org.omg.CORBA. Object objRef =
orb.resolve_initial_references («NameService»);
// Use NamingContextExt instead of NamingContext. This is
// part of the Interoperable naming Service.
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
// resolve the Object Reference in Naming
String name = «Hello»;
helloImpl = HelloHelper.narrow (ncRef.resolve_str(name));
System.out.println («Obtained a handle on server object:» + helloImpl);
System.out.println (helloImpl.sayHello());
helloImpl.shutdown();
} catch (Exception e) {
System.out.println («ERROR:» + e);
```

```
e.printStackTrace (System.out);
}
}
}
```

10.1.4 Компиляция и запуск приложения.

Файлы HelloServer.java and HelloClient.java, Hello.idl и папка HelloApp, созданная idkj.exe, должны храниться в одной папке.

Для компиляции клиента и сервера надо в командной строке набрать

javac *.java HelloApp/*.java

javac.exe находится в …\jdk1.5.0\bin.

Среда Eclipse не позволяет запускать CORBA-приложения. Для запуска необходимо следующее.

1 Запустить службу orbd – Object Request Broker Daemon (…\jdk1.5.0\bin\orbd.exe). Это делается, чтобы мы смогли получить ссылку на службу имен.

start orbd – ORBInitialPort 1050

Параметр – ORBInitialPort – номер порта, на котором будет работать сервер имен.

2 Запуск сервера

start java HelloServer – ORBInitialPort 1050 – ORBInitialHost localhost

Указывается порт, на котором работает сервер имен. Параметр ORBInitialHost указывает хост, на котором работает сервер имен.

3 Запуск клиента

java HelloClient – ORBInitialPort 1050 – ORBInitialHost localhost

Указывается порт, на котором работает сервер имен. Параметр ORBInitialHost указывает хост, на котором работает сервер имен.

Для удобства компиляции и запуска можно создать bat-файл:

idlj – fall Hello.idl

javac *.java HelloApp/*.java

start java HelloServer – ORBInitialPort 1050 – ORBInitialHost localhost

java HelloClient – ORBInitialPort 1050 – ORBInitialHost localhost.

Задание

Разработать почтового клиента и сервер. Клиент – оконное приложение, которое будет позволять отсылать и получать с сервера сообщения.

Контрольные вопросы

1 Что такое CORBA?

2 Как осуществляется взаимодействие клиента и сервера в CORBA?

3 Как передаются данные между ними?

4 Для чего нужен сервер имен?

5 Как запускается CORBA-сервер?

11 Самостоятельная работа № 11. Разработка веб-приложения по индивидуальным заданиям

Цель работы: закрепить навыки разработки веб-приложений.

Теоретические сведения

Используются теоретические сведения предыдущих самостоятельных работ.

Задание

Разработать веб-приложения по индивидуальному заданию.

Контрольные вопросы

1 Перечислите основные приемы разработки веб-приложения.

2 Как разместить веб-приложение в сети Интернет?

Список литературы

1 **Хутагуров, Я. А.** Проектирование автоматизированных систем обработки информации и управления (АСОИУ) / Я. А. Хутагуров. – Москва: Лаборатория знаний, 2020. – 243 с.

2 **Шустова, Л. И.** Базы данных : учебник / Л. И. Шустова, О. В. Тараканов. – Москва: ИНФРА-М, 2021. – 304 с.

3 **Олейник, П. П.** Корпоративные информационные системы: учебник / П. П. Олейник. – Санкт-Петербург: Питер, 2014. – 176 с.

4 **Ройс, У.** Управление проектами по созданию программного обеспечения: унифицированный подход / У. Ройс. – Москва: Лори, 2014. – 424 с.

5 **Рыбальченко, М. В.** Архитектура информационных систем : учебное пособие / М. В. Рыбальченко. – Москва: Юрайт, 2016. – 91 с.

6 **Орлов, С. А.** Программная инженерия. Технологии разработки программного обеспечения : учебник / С. А. Орлов. – 5-е изд., обновл. и доп. – Санкт-Петербург: Питер, 2016. – 640 с.