

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Р.В. Ковин, Е.А. Мирошниченко

Проектирование информационных систем

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Томск 2020

Введение

Лабораторный практикум посвящен решению практических задач по проектированию информационных систем, включая эскизное и техническое проектирование, описание решений с помощью диаграмм UML, проектирование моделей баз данных и интерфейсов пользователя.

Практикум содержит 12 лабораторных работ, относящихся к 9 разделам дисциплины. Особенностью данного практикума является то, что лабораторные работы связаны с разработкой некоторой системы. Задание выдается студенту индивидуально, а каждая лабораторная посвящена отдельному этапу разработки системы.

Содержание практикума соответствует рабочей программе дисциплины «Проектирование информационных систем» для студентов, обучающихся по направлению 09.04.03 «Прикладная информатика».

1. ЛАБОРАТОРНАЯ РАБОТА №1. Пользовательские истории.

Варианты использования

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков работы описания вариантов использования и пользовательских историй.

Введение

Существуют различные методики для выявления требований к поведению системы и их фиксации. В данной лабораторной работе рассматриваются пользовательские истории и варианты использования, но в большей степени внимание уделяется вариантам использования.

1.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1.1. Пользовательские истории (User Stories)

Пользовательские истории (англ. *User Stories*) — способ описания требований к разрабатываемой системе, сформулированных как одно или более предложений на повседневном или деловом языке пользователя.

Пользовательские истории — быстрый способ документировать требования, без необходимости разрабатывать обширные формализованные документы и впоследствии тратить ресурсы на их поддержание. Цель пользовательских историй состоит в том, чтобы быть в состоянии оперативно и без накладных затрат реагировать на быстро изменяющиеся требования реального мира. Пользовательские истории традиционно применяются в гибких методологиях разработки.

Для описания пользовательских историй используется следующий формат:

As a {user type}, I can {do something} so that {I receive some benefit}

Как {тип пользователя} я могу {делать что-нибудь} для того, чтобы {получить некоторую выгоду}

Примеры:

- As a **course participant**, I can **submit a question** so that **I get my concerns about the course materials addressed**.

- As a **course instructor**, I can **view all course participant questions** so that **I can respond in a timely manner**.
- Как модератор форума я могу удалять непристойные сообщения участников для того, чтобы поддерживать порядок на форуме.

Преимущества пользовательских историй:

- Истории короткие. Они представляют маленькие кусочки бизнес-ценности, которые можно реализовать в период от нескольких дней до нескольких недель.
- Позволяют разработчикам и клиентам обсуждать требования на протяжении всей жизни проекта.
- Нуждаются в очень небольшом обслуживании.
- Рассматриваются только в момент использования.
- Поддерживают близкий контакт с клиентом.
- Позволяют разбить проект на небольшие этапы.
- Подходят для проектов, где требования изменчивы или плохо поняты.
- Облегчают оценку заданий.

Недостатки пользовательских историй:

- Без определенных приемочных испытаний являются открытыми для различных интерпретаций, что усложняет их использование как основу для соглашения.
- Требуют близкого контакта с клиентом на протяжении всего проекта, что в некоторых случаях может быть сложно либо приводить к накладным затратам.
- Могут плохо масштабироваться на больших проектах.
- Полагаются на компетентность разработчиков.
- Используются для начала дискуссии. К сожалению, они могут не фиксировать окончание дискуссии и таким образом не в состоянии служить надежным методом документации системы.

1.1.2. Варианты использования (Use Cases)

Вариант использования (ВИ), сценарий использования, прецедент использования (англ. *use case*) — это описание поведения **системы**, когда она взаимодействует с кем-то (или чем-то) из внешней среды.

Вариант использования — формальное описание взаимодействия системы и пользователя при решении конкретной задачи. Каждый ВИ нацелен на конкретную задачу и описывает некоторое функциональное требование.

Вариант использования описывает, «кто» и «что» может сделать с рассматриваемой системой, или что система может сделать с «кем» или «чем».

Методика вариантов использования применяется для выявления требований к поведению системы.

Рассмотрим цели ВИ. ВИ рассматривают систему как «черный ящик». Взаимодействия с системой, включая системные ответы, описываются с точки зрения внешнего наблюдателя. При этом внимание сосредотачивается на том, что система должна сделать, а не как это должно быть сделано.

Достоинства модели вариантов использования заключаются в том, что она:

- определяет пользователей и границы системы;
- определяет системный интерфейс;
- удобна для общения пользователей с разработчиками;
- используется для написания тестов;
- является основой для написания пользовательской документации;
- хорошо вписывается в любые методы проектирования (как объектно-ориентированные, так и структурные).

Различают два уровня описания ВИ:

- Абстрактный уровень / Бизнес-сценарий использования — описывает процесс, ценный для бизнес-агента.
- Системный уровень / Системный сценарий использования описывает, что актер может сделать, взаимодействуя с системой. Обычно описывается на уровне функций системы.

К каждому ВИ предъявляются требования. ВИ должен:

- Описывать, что именно система должна сделать, чтобы актер достиг своей цели.
- Не затрагивать деталей реализации.
- Иметь достаточный уровень детализации.
- Не описывать пользовательские интерфейсы и экраны. Это делается во время проектирования пользовательского интерфейса.

Различают три уровня детализации при описании ВИ (рис. 1.1).

1. Краткий. На этом уровне ВИ описывается в несколько предложений.
2. Обычный. На этом уровне ВИ описывается в несколько абзацев.
3. Детализированный. На этом уровне ВИ представляет собой формальный документ, основанный на подробном шаблоне с различными разделами. Именно этот вариант (детализированный) подразумевается в большинстве случаев под понятием варианта использования.

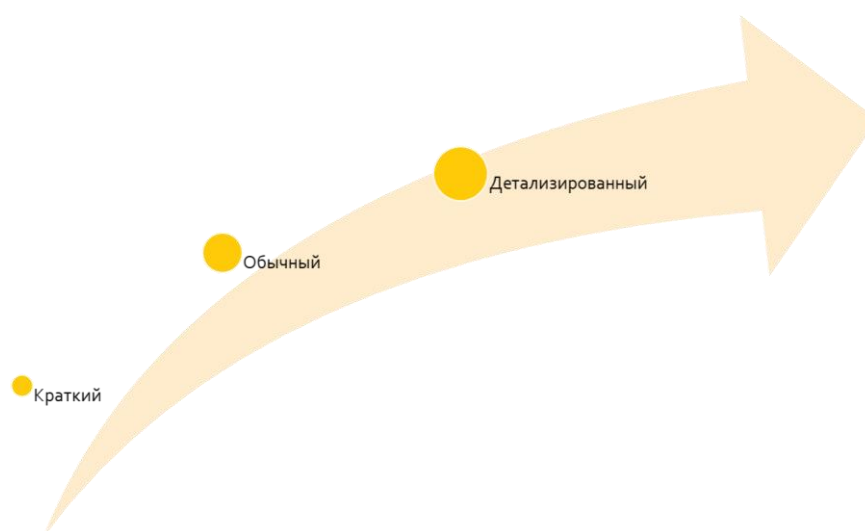


Рис. 1.1. Уровни детализации при описании ВИ

Как правило такой формальный документ состоит из следующих разделов:

Название / Имя

Цель

Акторы

Стейкхолдеры

Предварительные условия / Начальное состояние

Активаторы

Порядок событий / Основной сценарий

Альтернативные сценарии

В этом списке разделы, не являющиеся обязательными, отмечены курсивом. Назначение, рекомендуемая форма записи и примеры этих разделов показаны в таблице:

Раздел	Назначение	Форма записи	Примеры
Название / Имя	Краткое описание достижимой цели	Желательно в формате глагол-существительное	Открыть документ, Купить товар
Цель	Описывает то, чего пользователь намеревается достигнуть с этим ВИ	Кратко, до нескольких предложений	Возможность работы с документом
Актеры	Кто-то или что-то вне системы и влияющий на систему или находящийся под её влиянием	Может быть человеком, устройством, другой системой или подсистемой, или временем. Человек в реальном мире может быть представлен несколькими актерами	Участник форума, Модератор форума, Администратор системы
Стейкхолдеры	Человек или группа, которых затрагивает ВИ	Кратко в виде названия должности, подразделения, организации и ФИО	финансовый отдел
Предварительные условия /	Формулировка условий, при которых данный	Краткое описание	Документ выбран, пользователь в

Раздел	Назначение	Форма записи	Примеры
Начальное состояние	ВИ может быть инициирован		роли «Редактор документов» Выполнен ВИ «Выбрать документ», пользователь авторизован в системе
Активаторы	Активатор — это событие, инициирующее выполнение сценария. Событие может быть: внешним временным внутренним	Краткое описание	Загрузка веб-страницы Срабатывание датчика пожарной системы
Порядок событий / Основной сценарий	Описание типичного хода событий	Обычно ряд пронумерованных шагов	1. Пользователь инициировал открытие документа 2. Система открыла документ для просмотра
Альтернативные сценарии	Описание нетипичного хода событий	Обычно ряд пронумерованных шагов	На шаге 2 основного сценария

Раздел	Назначение	Форма записи	Примеры
			<p>произошла ошибка.</p> <p>2. Система информирует пользователя об ошибке. Конец</p>

В качестве примера опишем ВИ открытия документа для некоторой системы.

ВИ «Открыть документ»

Цель

Возможность работы с документом

Акторы

Пользователь

Стейкхолдеры

Пользователь, Финансовый отдел

Предварительные условия / Начальное состояние

Документ выбран, пользователь в роли «Редактор документов»

Активаторы

Пользователь выполняет команду «Открыть документ»

Основной сценарий

1. Пользователь инициировал открытие документа
2. Система открыла документ для просмотра

Альтернативные сценарии

Предусловие: на шаге 2 Основного сценария доступ к документу закрыт.

3. Система информирует пользователя об отсутствии доступа к документу

Аналогично опишем ВИ «Отправить электронное письмо» для некоторой почтовой программы.

ВИ «Отправить электронное письмо»

Цель

Отправить созданное письмо с проверкой корректности его атрибутов.

Начальное состояние

Выполнен ВИ «Создать новое письмо» или ВИ «Создать ответ на письмо».

Основной сценарий

1. Пользователь выполняет команду отправки письма.
2. Программа проверяет, правильно ли заполнено поле «Адрес».
3. Если нет, программа сообщает об ошибке и отменяет отправку. Конец.
4. Если да, то программа проверяет, заполнено ли поле «Тема».
5. Если нет, программа выдает предупреждение, но не отменяет отправку.
6. Программа помещает письмо в папку «Исходящие» и отсылает его.
7. После отправки программа перемещает письмо в папку «Отправленные».

Сценарий обработки ошибок

Предусловие: на шаге 6 основного сценария происходит ошибка отправки письма (сбой сети и т. п.)

7. Программа сообщает об ошибке и предлагает сохранить текст письма в файл.
8. Если пользователь согласен сохранить текст, выполняется ВИ «Сохранить черновик в файл».

На рис. 1.2 представлено графическое изображение этого альтернативного сценария (белые прямоугольники шагов 7 и 8).

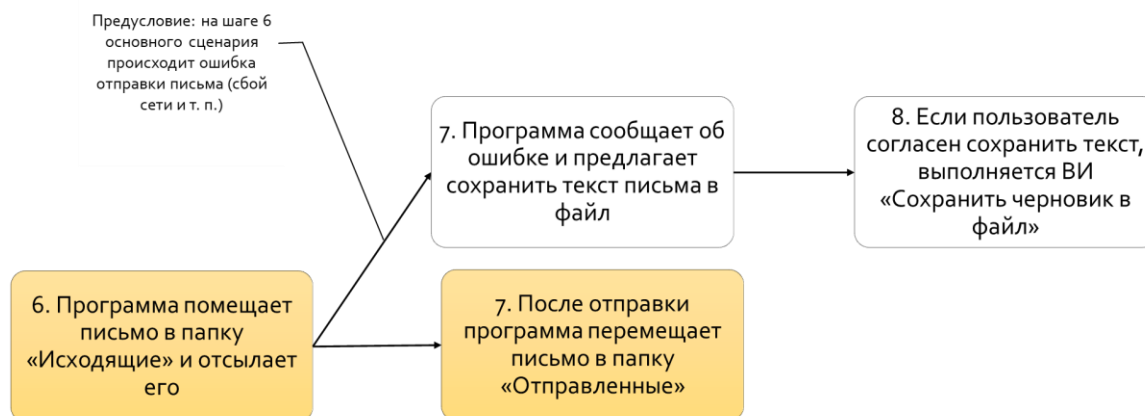


Рис. 1.2. Графическое изображение альтернативного сценария

Если альтернативных сценариев много и часть из них является терминальными (рис. 1.3), то описание ВИ становится громоздким и сложным для восприятия.

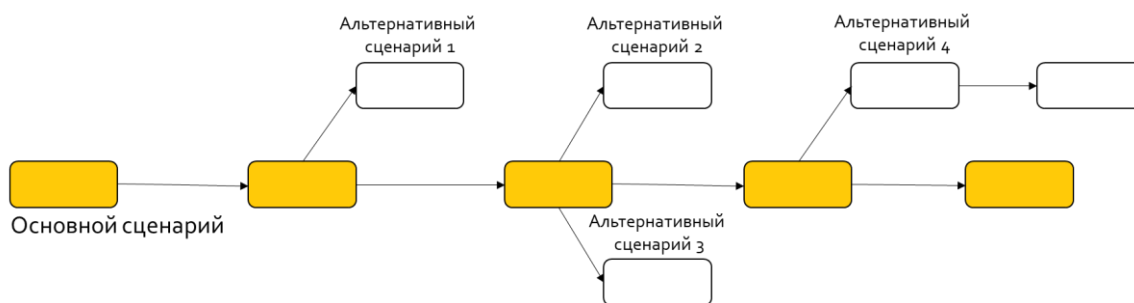


Рис. 1.3. Графическое изображение множества альтернативных сценариев

В этом случае допускается не оформлять их как альтернативные, а включать в текст основного сценария. В нашем примере шаг 6 может быть записан так:

1. Программа помещает письмо в папку «Исходящие» и отправляет его. Если происходит ошибка отправки письма (сбой сети и т. п.), то программа сообщает об ошибке и предлагает сохранить текст письма в файл. Если пользователь согласен сохранить текст, выполняется ВИ «Сохранить черновик в файл». Конец.

Ключевое слово, позволяющее включить альтернативный сценарий в основной, это «ЕСЛИ». На практике могут использоваться различные синонимы: «В СЛУЧАЕ», «КОГДА» и др. Однако не следует увлекаться этим и включать в один шаг основного сценария сразу несколько альтернативных. Это может ухудшить понимание шага сценария.

При написании сценариев рекомендуется применять простое правило — чередование шагов Актор-Система:

1. Пользователь выполнил...
2. Система...
3. Пользователь выполнил...
4. Система...
5. Пользователь выполнил...
6. Система...

Это правило позволяет продемонстрировать действия актора (пользователя) и реакцию на него системы. В дальнейшем именно такие сценарии могут стать основой для тестовых сценариев.

Рассмотрим типовые ошибки описания сценариев. В следующем примере не раскрыта промежуточная реакция системы:

1. Пользователь выполнил...
2. Пользователь выполнил...
3. Пользователь выполнил...
4. Пользователь выполнил...
5. Система...

В следующем примере не раскрыта конечная реакция системы:

1. Пользователь выполнил...
2. Система...
3. Пользователь выполнил...
4. Система...
5. Пользователь выполнил...

В следующих примерах сделано излишнее акцентирование на интерфейсе пользователя

- | | |
|--|---|
| 1. Пользователь нажал на кнопку «Открыть» | 1. Пользователь нажал на кнопку «Открыть» |
| 2. Система показала документ в новом окне. Документ представлен в видах: оглавление и основной текст | 2. Система показала диалоговое окно, содержащее список файлов, список папок, дисков и сетевых подключений |
| | 3. Пользователь, используя мышь выбрал нужный файл и нажал кнопку «Открыть» |
| | 4. Система.... |

1.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

1.2.1. Практическое задание

В качестве практического задания необходимо:

1. Описать пользовательские истории для системы, создаваемой студентом в рамках индивидуального задания на дисциплину.
2. Описать варианты использования для системы, создаваемой студентом в рамках индивидуального задания на дисциплину.

Для описания можно использовать текстовый редактор Microsoft Word или подобный.

1.2.2. Список контрольных вопросов для самопроверки

1. Для чего используются варианты использования и пользовательские истории?
2. Какой уровень детализации традиционно применяется при описании ВИ?
3. В чем отличие отношений ВИ «Включение» и «Расширение»?
4. Какие обстоятельства затрудняют применение пользовательских историй?

1.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

2. ЛАБОРАТОРНАЯ РАБОТА №2. Диаграммы UML.

Диаграмма вариантов использования

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков создания диаграмм вариантов использования.

Введение

Диаграмма вариантов использования является самым общим представлением функциональных требований к системе. Умение создавать такую диаграмму является важным навыком для специалистов, имеющих отношение к разработке программного обеспечения.

2.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1.1. Основные элементы диаграммы

Диаграмма вариантов использования (англ. *use case diagram*) в UML — диаграмма, отражающая отношения между акторами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент (ВИ) соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой.

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

На рис. 2.1 показаны основные элементы диаграммы: актор и ВИ / прецедент.

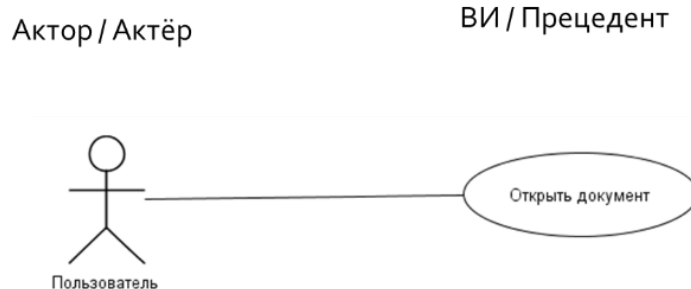


Рис. 2.1. Основные элементы диаграммы ВИ

На диаграмме актор и ВИ соединяются сплошными линиями без стрелок (рис. 2.2).

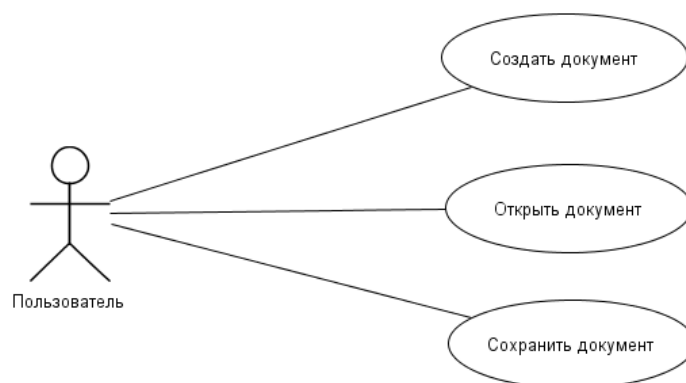


Рис. 2.2. Пример связей актора с ВИ

2.1.2. Отношения между элементами

Между ВИ могут быть разные отношения. *Отношение обобщения* служит для указания того факта, что некоторая сущность А может быть обобщена до сущности В. В этом случае сущность А будет являться специализацией сущности В. На диаграмме данный вид отношения можно отображать только между однотипными сущностями (между двумя вариантами использования или двумя актерами). Отношение показывается в форме стрелки с не закрашенным треугольником. Треугольник ставится у более общего прецедента (рис. 4.3).

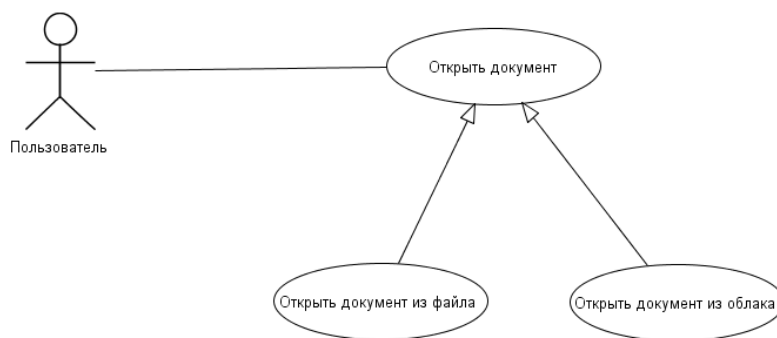


Рис. 2.3. Пример обобщения между ВИ

Обобщение между акторами показано на рис. 4.4.

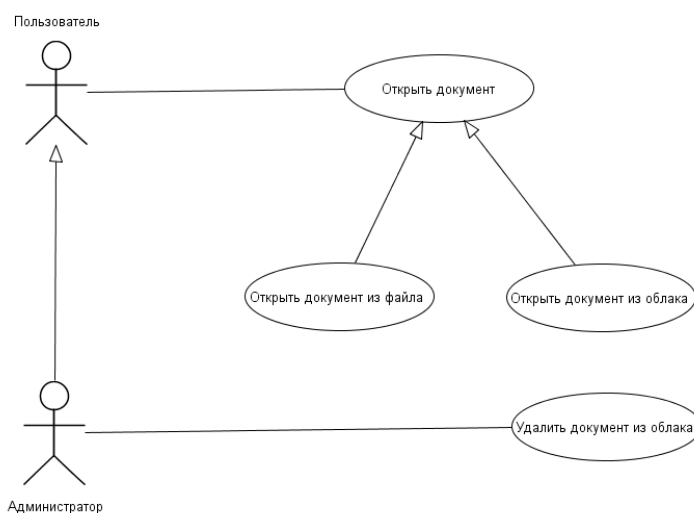


Рис. 2.4. Пример обобщения между акторами

Отношение включения указывает, что некоторое заданное поведение одного варианта использования **обязательно** включается в качестве составного компонента в **последовательность поведения** другого варианта использования. Отношение отображаются штриховой стрелкой и помечена стереотипом «include» (англ. включает). (рис. 2.5). Не следует путать это отношение с зависимостью одного ВИ от другого через начальное состояние.

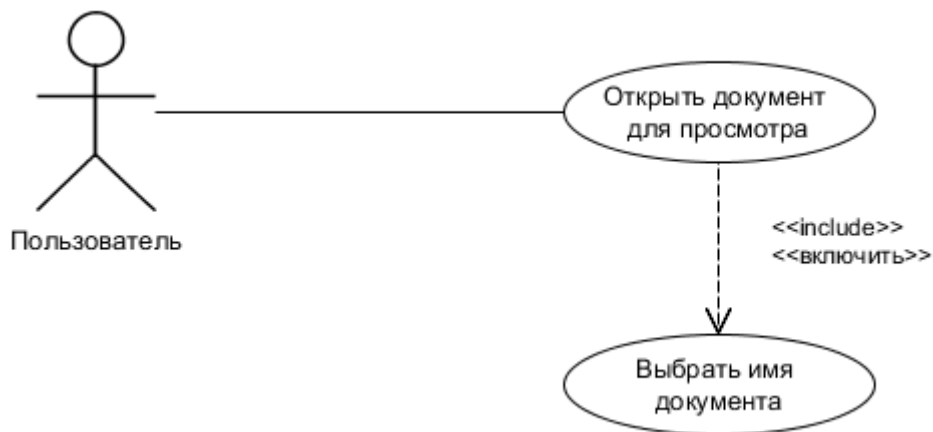


Рис. 2.5. Пример отношения включения

Отношение расширения определяет потенциальную возможность включения поведения одного варианта использования в состав другого. Т. е. дочерний вариант использования может как вызываться, так и не вызываться родительским. Стрелка расширения должна быть направлена от расширяющего варианта к базовому (рис. 2.6) и помечена стереотипом «extend» (англ. расширяет).

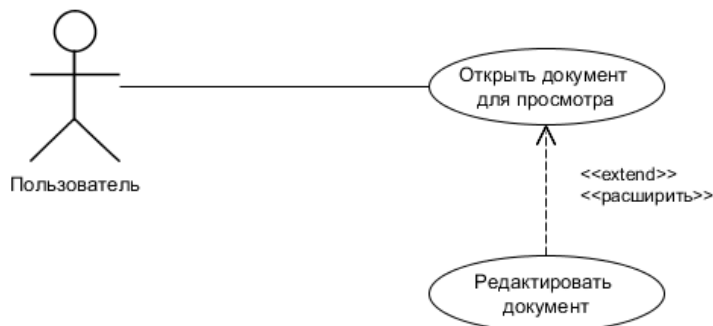


Рис. 2.6. Пример отношения расширения

Данное отношение также может быть описано в расширяющем ВИ в разделе «Начальное состояние». Это может быть указание на то, что базовый ВИ запущен (если расширение возможно во время выполнения базового ВИ) или выполнен (если расширение возможно после окончания ВИ).

Существуют и другие варианты отношений. На рис. 2.7 показаны примеры их отображения.

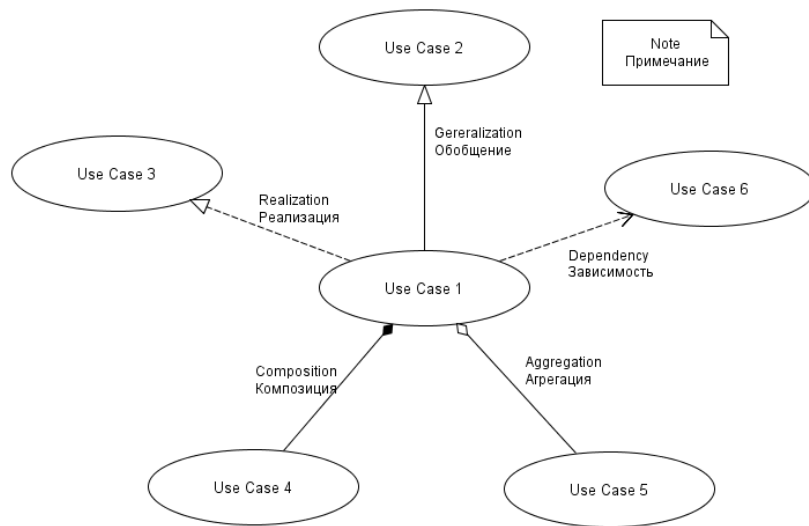


Рис. 2.7. Типы отношений между ВИ

Более подробно почитать о диаграмме ВИ можно по ссылке <https://www.uml-diagrams.org/use-case-diagrams.html>.

2.1.3. Инструментальные средства для создания диаграммы

Для создания диаграммы можно использовать специализированные векторные редакторы. Например, в бесплатном настольном редакторе yEd (<https://www.yworks.com/products/yed>) имеются готовые фигуры для создания диаграммы ВИ (рис. 2.8).

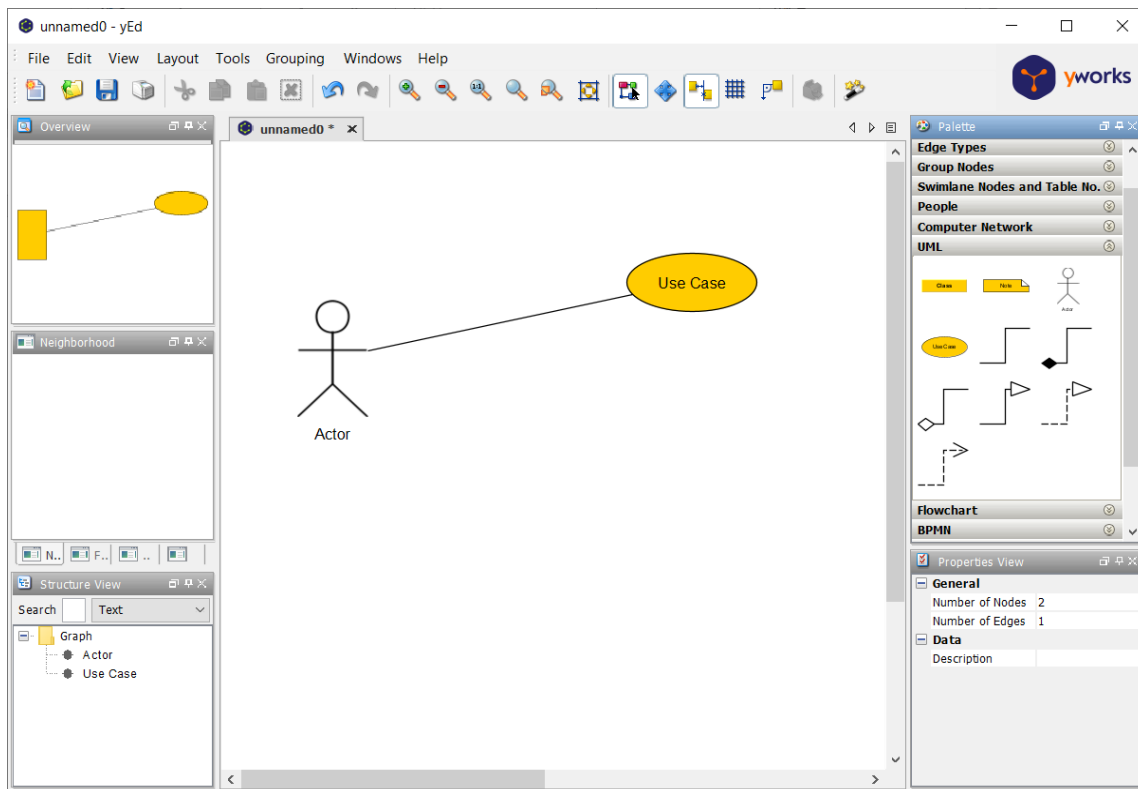


Рис. 2.8. Главное окно yEd

Созданные диаграммы ВИ могут добавляться в различные технические документы, такие как эскизный и технический проекты в виде изображений. При этом важно добавлять диаграммы с высоким качеством. Рекомендуется использовать векторный формат изображений.

2.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.2.1. Практическое задание

В качестве практического задания необходимо создать диаграмму вариантов использования на основе ВИ, описанных в лабораторной работе №1.

2.2.2. Список контрольных вопросов для самопроверки

1. Из каких элементов состоит диаграмма ВИ?
2. Как вы считаете почему для создания диаграмм ВИ желательно использовать специализированные редакторы?

2.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

3. ЛАБОРАТОРНАЯ РАБОТА №3. Техническое задание

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков создания технического задания.

Введение

Техническое задание (ТЗ, техзадание) — основной документ, содержащий требования заказчика к системе, в соответствии с которыми осуществляется создание и разработка конечного продукта.

Техническое задание позволяет:

- исполнителю — понять суть задачи, показать заказчику «технический облик» будущего изделия, программного изделия или автоматизированной системы;
- заказчику — осознать, что именно ему нужно;
- обеим сторонам — представить готовый продукт;
- исполнителю — спланировать выполнение проекта и работать по намеченному плану;
- заказчику — требовать от исполнителя соответствия продукта всем условиям, оговорённым в ТЗ;
- исполнителю — отказаться от выполнения работ, не указанных в ТЗ;
- заказчику и исполнителю — выполнить проверку готового продукта;
- избежать ошибок, связанных с изменением требований.

3.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

3.1.1. Основные работы при разработке требований

При разработке ТЗ необходимо выполнить следующие работы.

1. Исходная постановка задачи.
2. Идентификация и вовлечение стейкхолдеров.
3. Сбор и исследование информации:
 - данные о предметной области в целом;
 - данные о существующих аналогах, конкурирующих продуктах;
 - истории успехов, истории провалов;

- данные о специфике заказчика, например:
 - специфика бизнес-процессов организации;
 - данные об унаследованном ПО (legacy software);
 - используемое аппаратное обеспечение;
 - политика безопасности организации;
 - уровень квалификации персонала.
- 4. Выбор приоритетных критериев качества.
- 5. Формализация требований, их описание.

3.1.2. Понятие требования

Требование — утверждение, которое передаёт или выражает некоторую потребность и связанные с ней ограничения и условия (ISO/IEC/IEEE 29148-2011).

Важно понимать, что выражение некоторой потребности должно соответствовать определенным характеристикам. Характеристики правильного требования:

- **Необходимость.** Транслирует (описывает) реальную потребность стейкхолдеров.
- **Атомарность.** Не является механическим объединением разных требований.
- **Понятность.** Из описания всё понятно, нет необходимости обращаться за толкованием к заказчику.
- **Независимость от реализации.** Не накладывает преждевременные и ненужные технические ограничения.
- **Проверяемость.** Можно объективно проверить, выполнено ли требование.
- **Правдоподобность (реалистичность, выполнимость).** Требование должно быть выполнимо в рамках существующих ограничений, таких как текущий уровень технологий, законодательство, время, деньги и доступные ресурсы.

Не только отдельные требования должны иметь перечисленные характеристики, но и наборы требований должно соответствовать определенным характеристикам. Характеристики хорошего набора требований:

- **Полнота.** Описывает всё, что нужно для определения системы, без значимых упущений.

- **Непротиворечивость.** Одни требования не противоречат другим. Часто требования к функциональности несовместимы с требованиями к производительности; удобство для пользователя несовместимо с информационной безопасностью и т.д.
- **Идентифицируемость.** На каждое требование можно ссылаться, его можно отслеживать, трассировать и т.д.

3.1.3. Идентификация требований

Возможны следующие подходы к идентификации (кодификации) требований:

- никакой;
- нумерация абзацев, как в юридических документах;
- мнемонические идентификаторы (буквенно-цифровые).

Система кодирования требований преследует следующие цели:

- уникальность идентификаторов;
- однозначность ссылок (независимость от словоформ и опечаток);
- компактность ссылок;
- простота поиска;
- исключение потребности в перенумерации требований при изменении нумерации разделов.

В качестве примера рассмотрим вариант мнемонических идентификаторов. В этом примере формат идентификатора имеет вид

[A][BB].[VV].[GG], где:

- A — префикс
- BB, VV, GG — двузначное число от 00 до 99
- BB — код первого уровня
- VV — код второго уровня
- GG — код третьего уровня

Префикс	Тип требования
A	Архитектурное требование
C	Требование к аппаратной или программной совместимости

D	Требование к структуре данных
F	Функциональное требование
R	Требование к надёжности
S	Требование к информационной безопасности
T	Требование к передаче результата (сдача/приёмка, внедрение)
U	Требование к пользовательскому интерфейсу

С требованиями связано понятие стейкхолдера. В техническом задании требования выражают те или иные потребности стейкхолдеров.

3.1.4. Понятие стейкхолдера

Есть разные синонимы/переводы этого термина: «заинтересованная сторона», «причастная сторона», «участник работ», «~~правообладатель~~». Под стейкхолдером понимают:

- Физическое лицо или организация, имеющая права, долю, требования или интересы относительно системы или её свойств, удовлетворяющих их потребностям и ожиданиям (ISO/IEC 15288, ISO/IEC 29148).
- Физическое лицо, команда, организация или их классы, имеющие интерес в системе (ISO/IEC 42010).
- Физическое лицо, группа лиц или организация, которые могут влиять на систему или на которых может повлиять система (OMG Essence).

Важно: стейкхолдеры — это и роли, и люди в этих ролях.

3.1.5. Государственные и международные стандарты для разработки ТЗ

При разработке технического задания следует руководствоваться следующими государственными и международными стандартами:

- ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению.
- ГОСТ 34.602-89 Информационная технология. Техническое задание на создание автоматизированной системы.

Стандарт ISO/IEC/IEEE 29148-2011 определяет:

- Stakeholder requirements specification (StRS);
- System requirements specification (SyRS);
- Software requirements specification (SRS).

3.1.6. Структура ТЗ по ГОСТ 19.201-78

В ГОСТ [19.201-78](#) «Единая система программной документации. Техническое задание. Требования к содержанию и оформлению» в техническом задании предусмотрены следующие разделы:

- Наименование и область применения
- Основание для разработки
- Назначение разработки
- Технические требования
- Требования к функциональным характеристикам
- Требования к надёжности
- Условия эксплуатации
- Требования к составу и параметрам технических средств
- Требования к информационной и программной совместимости
- Требования к маркировке и упаковке
- Требования к транспортированию и хранению
- Специальные требования
- Технико-экономические показатели
- Стадии и этапы разработки
- Порядок контроля и приёмки
- Приложения

3.1.7. Структура ТЗ по ГОСТ 34.602-89

В ГОСТ [34.602-89](#) «Информационная технология. Техническое задание на создание автоматизированной системы» в техническом задании предусмотрены следующие разделы:

- Общие сведения

- Назначение и цели создания (развития) системы
- Характеристика объектов автоматизации
- Требования к системе
 - Требования к системе в целом
 - Требования к функциям (задачам), выполняемым системой
 - Требования к видам обеспечения
- Состав и содержание работ по созданию системы
- Порядок контроля и приемки системы
- Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие
- Требования к документированию
- Источники разработки

3.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

3.2.1. Практическое задание

В качестве практического задания необходимо написать документ «Техническое задание» для системы, создаваемой студентом в рамках индивидуального задания на дисциплину. Особое внимание уделить описанию функциональных требований.

3.2.2. Список контрольных вопросов для самопроверки

1. Почему программисту необходимо уметь составлять техническое задание?
2. Почему требования должны соответствовать определенным характеристикам?
3. Какая польза от идентификации требований?

3.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.

3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

4. ЛАБОРАТОРНАЯ РАБОТА №4. Диаграммы UML.

Диаграмма компонентов

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков создания диаграммы компонентов.

Введение

Диаграмма компонентов (англ. *Component diagram*) — элемент языка моделирования UML, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Диаграмма компонентов разрабатывается для следующих целей:

- визуализация общей структуры исходного кода программной системы.
- спецификация исполнимого варианта программной системы.
- обеспечение многократного использования отдельных фрагментов программного кода.
- представление концептуальной и физической схем баз данных.

Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов.

4.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

4.1.1. Базовые понятия

Интерфейс — это набор операций, которые специфицируют сервис, предоставляемый либо требуемый классом или компонентом.

Компонент — замещаемая часть системы, которая соответствует набору интерфейсов и обеспечивает его реализацию.

Порт — специфическое «окно» в инкапсулированный компонент, принимающее сообщения для компонента и от него в соответствии с заданным интерфейсом.

Внутренняя структура — реализация компонента, представленная набором частей, соединенных друг с другом конкретным способом.

Часть — спецификация роли, составляющей часть реализации компонента. В экземпляре компонента присутствует экземпляр, соответствующий части.

Коннектор — связь коммуникации между двумя частями или портами в контексте компонента.

4.1.2. Компоненты и интерфейсы

Компонент (англ. *component*) — это специальный термин в языке UML, предназначенный для представления физических сущностей. Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели.

На рис. 4.1 показана типичное графическое представление компонента. Внутри прямоугольника записывается имя компонента и при необходимости, дополнительная информация.

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и некоторых знаков препинания. Отдельный компонент может быть представлен

- уровне типа;
- на уровне экземпляра.

Хотя его графическое изображение в обоих случаях схожее, правила записи имени компонента отличаются. Если компонент представляется на уровне типа, то в качестве его имени записывается только имя типа с заглавной буквы. Если же компонент представляется на уровне экземпляра (*instance*), то в качестве его имени записывается <имя компонента ':' имя типа>. При этом вся строка имени подчеркивается.

Интерфейс — набор операций, используемый для спецификации сервиса класса или компонента. Связь между компонентом и интерфейсом имеет важное значение. Все основанные на компонентах средства операционных систем используют интерфейсы в качестве элементов, связывающих компоненты друг с другом. Интерфейс, который реализован компонентом, называется *предоставляемым* (то есть данный компонент предоставляет интерфейс в виде сервиса другим компонентам).

Компонент может декларировать множество предоставляемых интерфейсов. Интерфейс, который он использует, называется *требуемым*: ему соответствует данный компонент, когда запрашивает сервисы от других компонентов. Компонент может соответствовать множеству требуемых интерфейсов. Бывают компоненты, которые одновременно предоставляют и требуют интерфейсы.

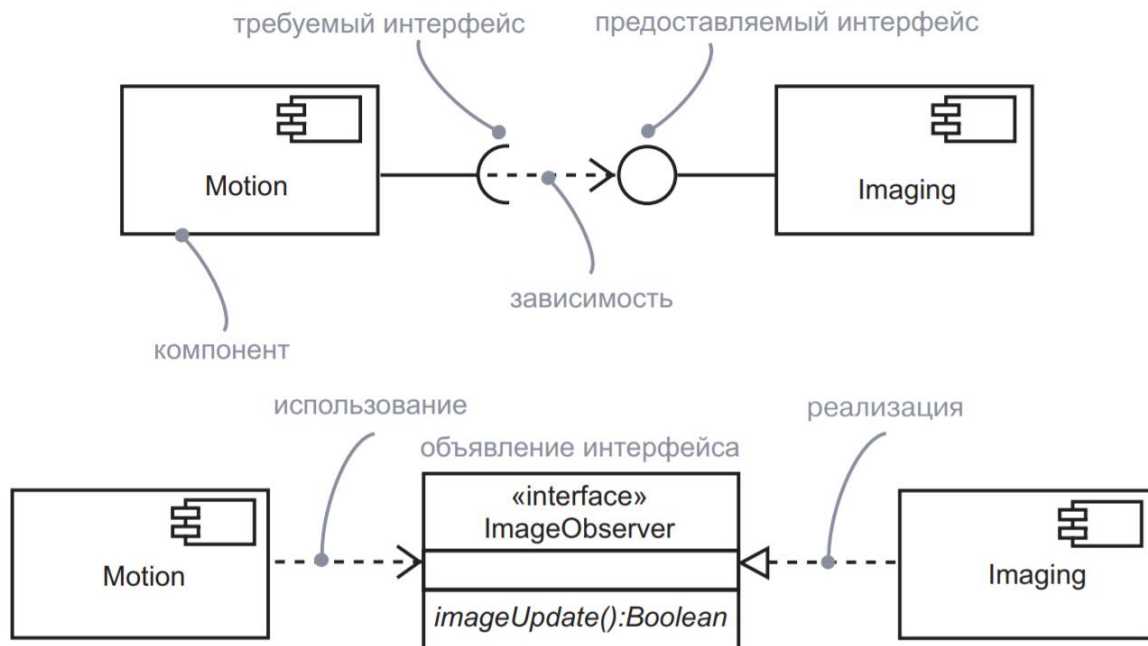


Рис. 4.1. Компоненты и интерфейсы

4.1.3. Порты

Порт (англ. *port*) — это своеобразное «окно» в инкапсулированный компонент. Все взаимодействие с таким компонентом на входе и на выходе происходит через порты.

Один компонент может взаимодействовать с другим через определенный порт. Порт схематически представлен маленьким квадратом на боковой грани компонента — это отверстие в границе инкапсуляции компонента (рис. 4.2). Как предоставляемый, так и требуемый интерфейс может быть соединен с символом порта. Предоставляемый интерфейс изображает сервис, который может быть запрошен извне через данный порт, а требуемый интерфейс – сервис, который порт должен получить от какого-либо другого компонента. У каждого порта есть имя, а следовательно, он может быть идентифицирован по компоненту и имени.

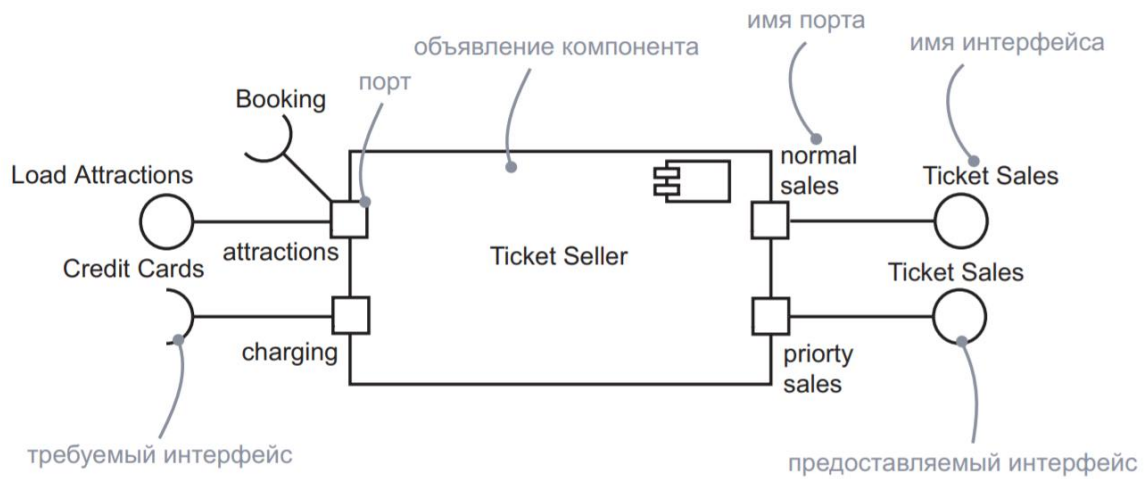


Рис. 4.2. Порты компонента

4.1.4. Части компонента

Компонент может быть реализован как единый фрагмент кода, но в больших системах желательно иметь возможность строить крупные компоненты из малых, которые используются в качестве строительных блоков. Внутренняя структура компонента содержит *части*, которые вкуче с соединениями между ними составляют его реализацию.

Часть — это не то же самое, что класс. Каждая часть идентифицируется по ее имени так же, как в классе различаются атрибуты. Допустимо наличие нескольких частей одного и того же типа, но их можно различать по именам и они предположительно выполняют разные функции внутри компонента.

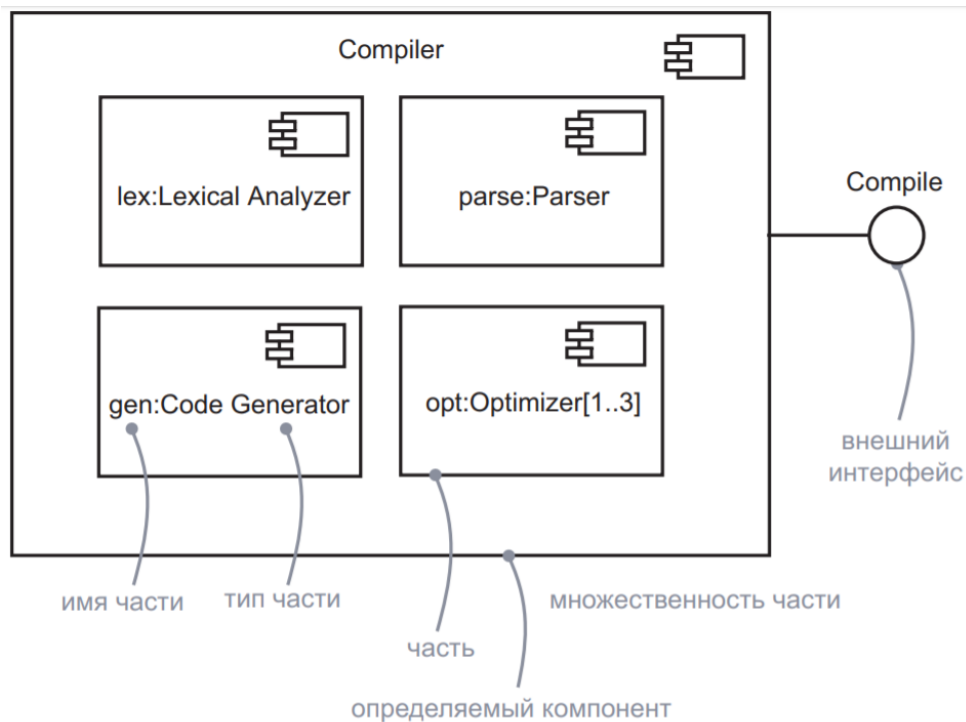


Рис. 4.3. Части компонента

4.1.5. Коннекторы

«Проводок» между двумя портами называется *коннектором*. В экземпляре охватывающего компонента он представляет просто ссылку (*link*) или временную ссылку (*transient link*). Простая ссылка – это экземпляр обычной ассоциации. Временная ссылка представляет связь использования между двумя компонентами. Коннекторы изображаются двумя способами (рис. 4.4). Если два компонента явно связаны друг с другом (либо напрямую, либо через порты), достаточно провести линию между ними или их портами. Если же два компонента подключены друг к другу, потому что имеют совместимые интерфейсы, можно использовать нотацию «шарик–гнездо», чтобы показать, что между этими компонентами не существует постоянной связи, хотя они и соединены внутри объемлющего компонента.

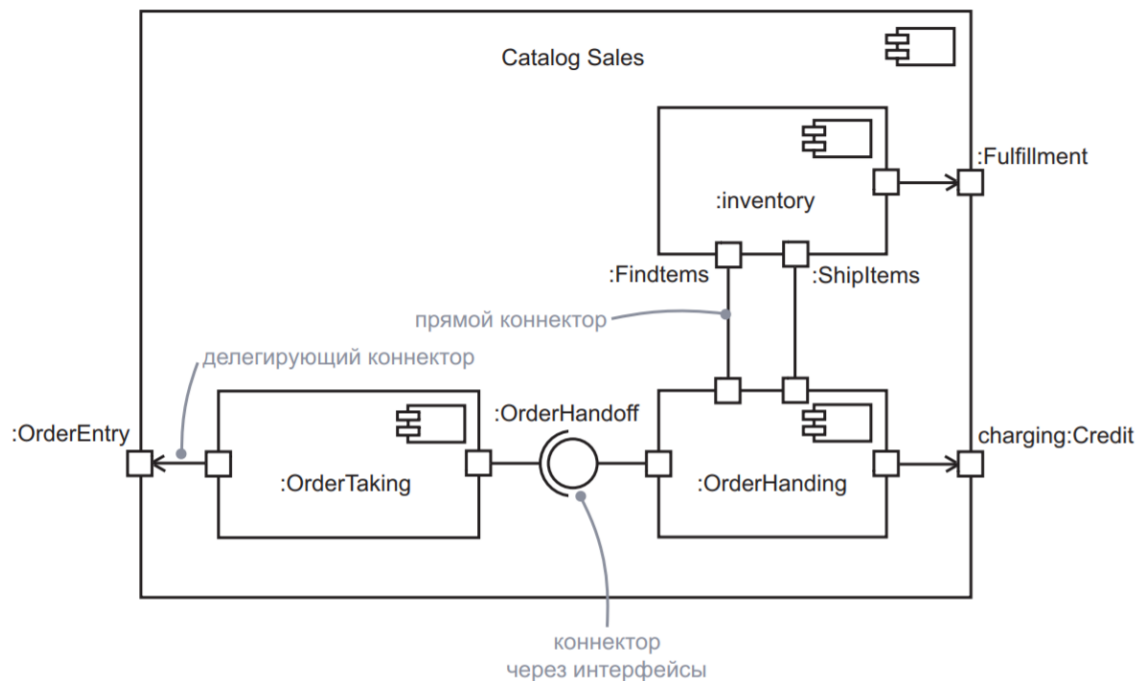


Рис. 4.4. Коннекторы

4.1.6. Рекомендации по построению диаграммы компонентов

Подробнее о построении диаграммы компонентов можно прочитать по ссылке <https://www.omg.org/spec/UML/2.1.2/Superstructure/PDF> и в книге Гради Буч, Джеймс Рамбо, Ивар Якобсон. Язык UML. Руководство пользователя.

4.1.7. Инструментальные средства для создания диаграммы компонентов

Существует большое количество инструментальных средств для создания диаграммы UML¹. По целевой платформе их можно разделить на настольные приложения и онлайн-редакторы. Среди настольных приложений выделим, конечно, Microsoft Visio. Однако эта система является платной. Многие онлайн-редакторы имеют бесплатный тариф, возможностей которого зачастую достаточно для создания диаграмм. Например, в редакторе Visual Paradigm Online Diagrams

¹ См. List of Unified Modeling Language tools // https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

(<https://online.visual-paradigm.com/>) можно создавать разные типы диаграмм UML, включая диаграмму компонентов (рис. 4.5). Однако в бесплатной версии системы при экспорте результатов в файл или буфер обмена добавляются водяные знаки о бесплатном тарифе.

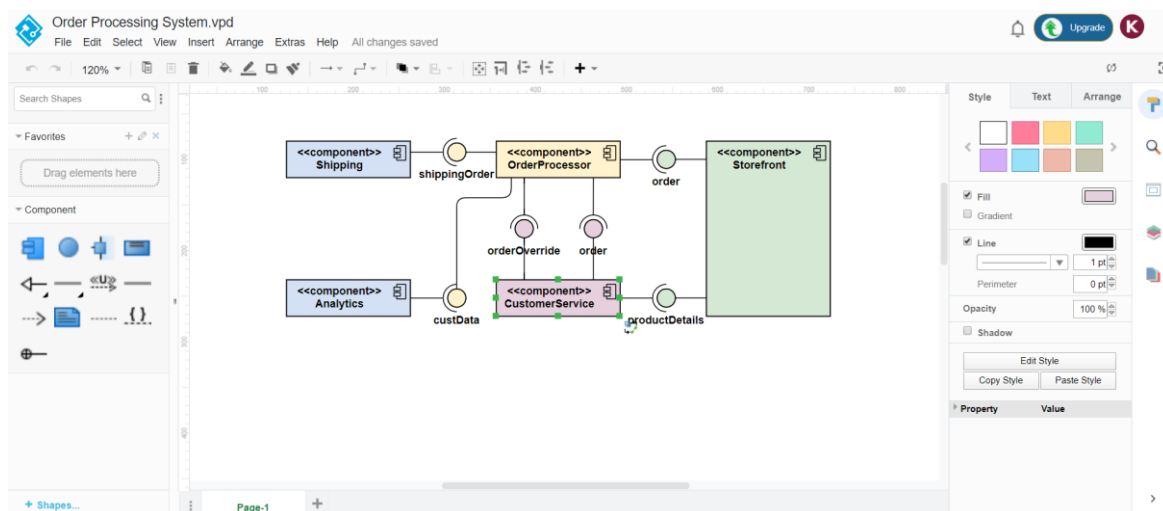


Рис. 4.5. Онлайн-редактор Visual Paradigm Online Diagrams

Созданные диаграммы компонентов могут добавляться в различные технические документы, такие как эскизный и технический проекты в виде изображений. При этом важно добавлять диаграммы с высоким качеством. Рекомендуется использовать векторный формат изображений.

4.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

4.2.1. Практическое задание

В качестве практического задания необходимо создать диаграмму компонентов для системы, создаваемой студентом в рамках индивидуального задания на дисциплину.

4.2.2. Список контрольных вопросов для самопроверки

1. Для чего создается диаграмма компонентов?

2. С помощью каких элементов на диаграмме можно показать классы объектов?
3. Как на диаграмме можно показать экземпляры классов объектов?

4.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

5. ЛАБОРАТОРНАЯ РАБОТА №5. Диаграммы UML.

Диаграмма последовательности

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков создания диаграммы последовательности.

Введение

Диаграмма последовательности (англ. *sequence diagram*) — диаграмма в языке UML, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта и взаимодействие акторов ИС в рамках какого-либо определённого прецедента.

Диаграмма взаимодействия предназначена для моделирования отношений между объектами (ролями, классами, компонентами) системы в рамках одного прецедента (ВИ). Данный вид диаграмм отражает следующие аспекты проектируемой системы:

- обмен сообщениями между объектами (в том числе в рамках обмена сообщениями со сторонними системами);
- ограничения, накладываемые на взаимодействие объектов;
- события, инициирующие взаимодействия объектов.

5.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Диаграмма последовательности является одной из разновидности диаграмм взаимодействия и предназначена для моделирования взаимодействия объектов Системы во времени, а также обмена сообщениями между ними.

Одним из основных принципов ООП является способ информационного обмена между элементами системы, выражающийся в отправке и получении сообщений друг от друга. Таким образом, основные понятия диаграммы последовательности связаны с понятием Объект и Сообщение.

5.1.1. Объекты

На диаграмме последовательности объекты в основном представляю экземпляры класса или сущности, обладающие поведением. В качестве объектов

могут выступать пользователи, инициирующие взаимодействие, классы, обладающие поведением в системе или программные компоненты, а иногда и системы в целом.

Если объект представляется на уровне типа, то в качестве его имени записывается только имя типа с заглавной буквы. Если же компонент представляется на уровне экземпляра (instance), то в качестве его имени записывается <имя компонента ':' имя типа>. При этом вся строка имени подчеркивается.

Объекты располагаются слева направо таким образом, чтобы крайним слева был тот объект, который инициирует взаимодействие. Неотъемлемой частью объекта на диаграмме последовательности является *линия жизни объекта*. Линия жизни показывает время, в течение которого объект существует в системе. Периоды активности объекта в момент взаимодействия показываются с помощью фокуса управления. Временная шкала на диаграмме направлена сверху вниз (рис. 5.1).

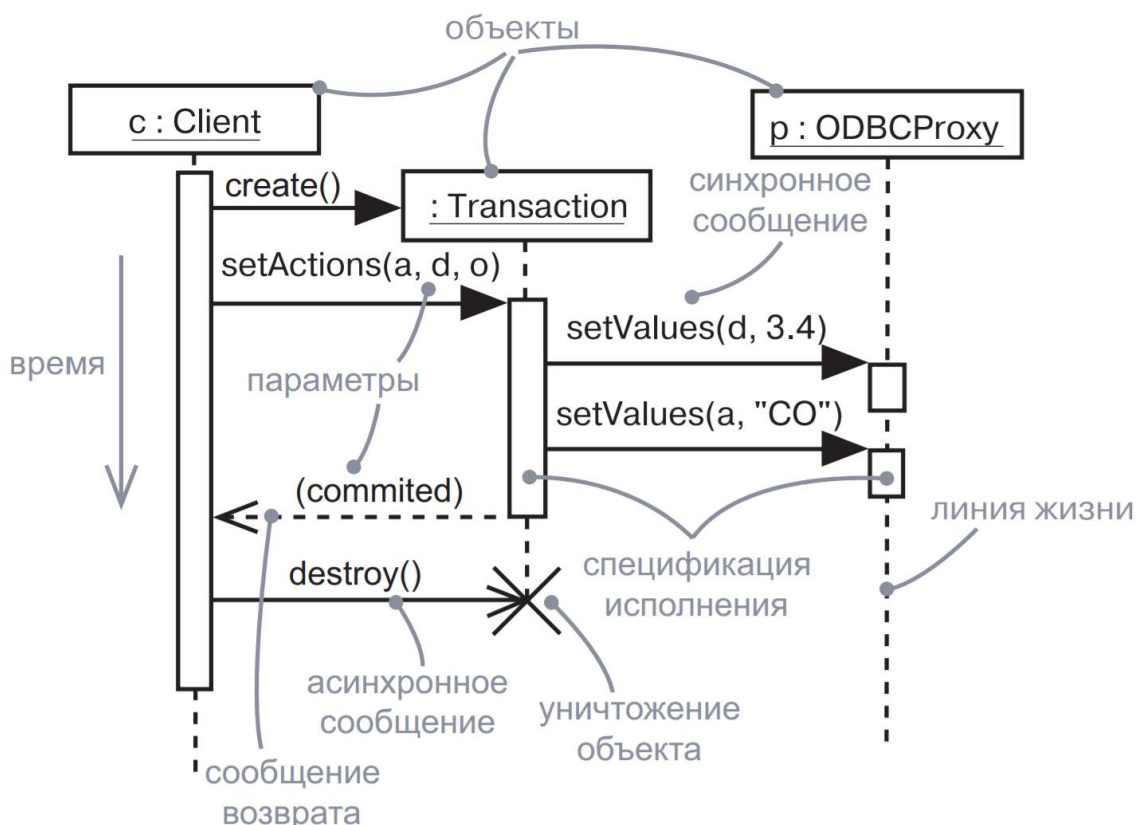


Рис. 5.1. Диаграмма последовательности

Объекты могут быть созданы в процессе взаимодействия. Их время жизни начинается с получения сообщения **create**, направленного к прямоугольнику объекта в начале жизненного пути. Равным образом в процессе взаимодействия объекты

могут уничтожаться. Их линия жизни заканчивается при получении сообщения **destroy**, что графически отмечено большим символом X (рис. 5.1). Если взаимодействие отражает историю конкретных объектов, то символ объекта с подчеркнутым именем размещается в начале линии жизни.

5.1.2. Сообщения

Основное содержимое диаграммы последовательности — *сообщения*. Они изображаются стрелками, направленными от одной линии жизни к другой. Стрелка указывает на приемник сообщения. Если сообщение асинхронно, то стрелка рисуется «уголком», а если синхронно (вызов), то закрашенным треугольником. Ответ на синхронное сообщение (возврат из вызова) показывается пунктирной стрелкой «уголком» (рис. 5.1.. Существуют и другие виды сообщений.

5.1.3. Рекомендации по построению диаграммы последовательности

Подробнее о построении диаграммы последовательности можно прочитать по ссылке <https://www.omg.org/spec/UML/2.1.2/Superstructure/PDF> и в книге Гради Буч, Джеймс Рамбо, Ивар Яacobсон. Язык UML. Руководство пользователя.

5.1.4. Инструментальные средства для создания диаграммы последовательности

Существует большое количество инструментальных средств для создания диаграмм UML². По целевой платформе их можно разделить на настольные приложения и онлайн-редакторы. Среди настольных приложений выделим, конечно, Microsoft Visio. Однако эта система является платной. Многие онлайн-редакторы имеют бесплатный тариф, возможностей которого зачастую достаточно для создания диаграмм. Например, в редакторе Visual Paradigm Online Diagrams

² См. List of Unified Modeling Language tools // https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

(<https://online.visual-paradigm.com/>) можно создавать разные типы диаграмм UML, включая диаграмму последовательности (рис. 5.2). Однако в бесплатной версии системы при экспорте результатов в файл или буфер обмена добавляются водяные знаки о бесплатном тарифе.

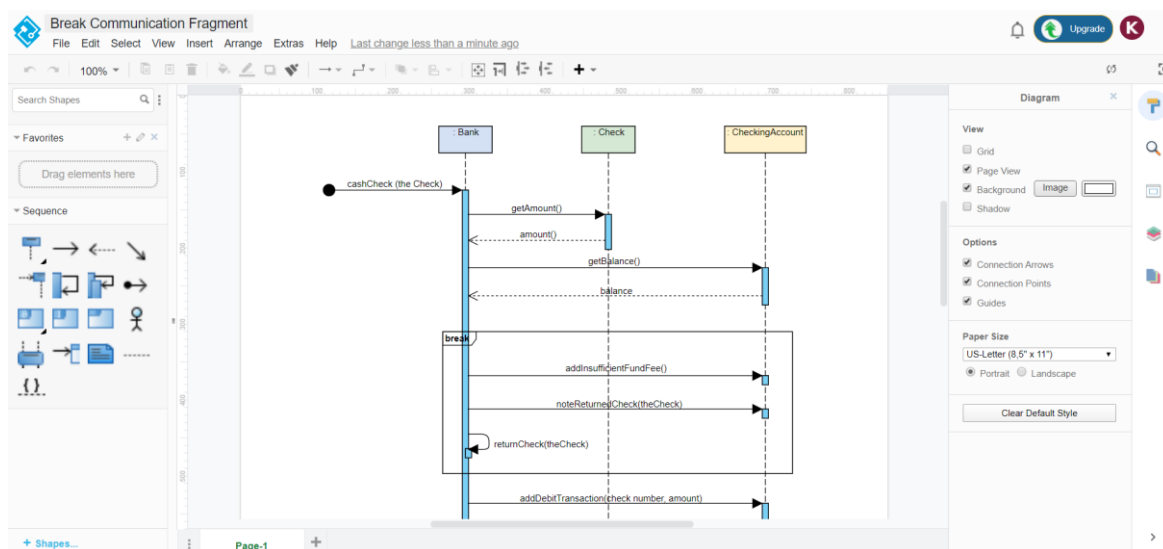


Рис. 5.2. Онлайн-редактор Visual Paradigm Online Diagrams

Созданные диаграммы последовательности могут добавляться в различные технические документы, такие как эскизный и технический проекты в виде изображений. При этом важно добавлять диаграммы с высоким качеством. Рекомендуется использовать векторный формат изображений.

5.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

5.2.1. Практическое задание

В качестве практического задания необходимо создать диаграммы последовательностей для **ключевых** вариантов использования (прецедентов) системы, создаваемой студентом в рамках индивидуального задания на дисциплину. Список ключевых вариантов использования должен быть согласован с преподавателем.

5.2.2. Список контрольных вопросов для самопроверки

1. Для чего создается диаграмма последовательности?
2. Скольким вариантам использования соответствует диаграмма последовательности?
3. Что может выступать в качестве объектов на диаграмме последовательности?

5.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

6. ЛАБОРАТОРНАЯ РАБОТА №6. Эскизный проект.

Технический проект

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков создания эскизного и технического проектов.

Введение

Эскизное проектирование — разработка предварительных проектных решений по системе и её частям. Итоговым документом выполнения работ на данной стадии проектирования является эскизный проект.

Эскизный проект — пакет конструкторской документации, создаваемый на стадии разработки системы. Цель создания этих документов — установить принципиальные, конструктивные решения, представить их для ознакомления с принципами работы и устройством разрабатываемой системы. Также этим проектом может рассматриваться несколько вариантов устройства системы.

Эскизный проект на автоматизированную систему разрабатывают перед техническим проектом или вместе с ним. Эта документация может и не оформляться в случае, если ею не может быть предоставлено никаких новых данных — её необходимость устанавливается техзаданием.

Технический проект — стадия разработки конструкторской документации на изделие или стадия создания автоматизированной системы. В более узком смысле под техническим проектом понимается совокупность технических документов, которые содержат окончательные проектные решения по изделию.

6.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

6.1.1. Эскизный проект

Эскизный проект разрабатывают с целью выявления предварительных технических решений, дающих начальное представление о конструкции системы.

При разработке эскизного проекта составляются:

- Ведомость эскизного проекта (общая информация по проекту).
- Пояснительная записка к эскизному проекту (вводная информация, позволяющая ее потребителю быстро освоить данные по конкретному проекту).

- Схема организационной структуры (описание организационной структуры организации, которая будет использовать создаваемую автоматизированную систему в практической работе).
- Структурная схема комплекса технических средств (техническая составляющая автоматизированной системы, включающая в себя набор серверов, рабочих станций, схему локальной вычислительной сети и структурированной кабельной системы).
- Схема функциональной структуры (описание задач, которые будут использоваться в работе подсистем).
- Схема автоматизации (логический процесс создания автоматизированной системы от начала до конца).

Выполнение эскизного проектирования не является строго обязательным. Если основные проектные решения определены ранее или достаточно очевидны для конкретной ИС и объекта автоматизации, то эта стадия может быть исключена из общей последовательности работ.

6.1.2. Технический проект

Технический проект (ТП) разрабатывают, если это предусмотрено техническим заданием, протоколом рассмотрения технического предложения или эскизного проекта. Технический проект разрабатывают с целью выявления окончательных технических решений, дающих полное представление о конструкции системы, когда это целесообразно сделать до разработки рабочей документации.

При необходимости технический проект может предусматривать разработку вариантов отдельных составных частей системы. В этих случаях выбор оптимального варианта осуществляется на основании результатов испытаний опытных образцов системы.

При разработке технического проекта выполняют работы, необходимые для обеспечения предъявляемых к системе требований и позволяющие получить полное представление о конструкции разрабатываемой системы, оценить его соответствие требованиям технического задания, технологичность, степень сложности изготовления, удобство эксплуатации, целесообразность и т.п.

В соответствии с ГОСТ 34.601-90 определяются следующие этапы формирования ТП:

- разработка проектных решений для системы и ее составляющих;
- разработка документации на систему и ее составляющие;
- формирование документации на поставку изделий для комплектации системы;
- разработка заданий на проектирование в смежных частях проекта объекта автоматизации.

На первом этапе разрабатываются общие решения по системе и ее частям. Формируются функции персонала и решения по организационной структуре, решения по структуре технических средств, по алгоритмам решения задач и применяемым языкам, по организации и ведению информационной базы, решения по системе классификации и кодированию информации, по ПО.

На следующем этапе выполняются разработка, оформление, согласование и утверждение документации. Перечень подлежащих разработке документов определен техническим заданием в разделе «Требования к документированию».

На этапе разработки и оформления документации выполняются подготовка и оформление документации на поставку изделий для комплектования системы и формируются технические требования и ТЗ на разработку изделий, не изготавливаемых серийно.

На последнем этапе формирования ТП осуществляются разработка, оформление, согласование и утверждение заданий на проектирование строительных, электротехнических, санитарно-технических работ и других подготовительных работ, связанных с созданием системы.

Перечень документов, создаваемых на стадии «Технический проект», определяется стандартом [ГОСТ 34.201-89](#).

6.1.3. Пояснительная записка к техническому проекту

Пояснительная записка к техническому проекту — это один из основных документов, входящих в число документации, составляемой на этапе технического проектирования. В пояснительной записке содержатся общие сведения о проектируемой системе, обоснования технических решений, которые были выбраны для ее создания, а также план действий, благодаря которым планируется ввести систему в эксплуатацию.

Этот документ, согласно стандартам и руководящим документам (см. п.п 6.1.4), должен состоять из следующих разделов:

1. Общие положения

- Наименование системы
- Основания для проведения работ
- Наименование организаций – Заказчика и Разработчика
- Цели, назначение и область использования системы
- Нормативные ссылки
- Очередность создания системы

2. Основные технические решения

- Решения по структуре системы, подсистем, средствам и способам связи для информационного обмена между компонентами системы
- Решения по взаимосвязям системы со смежными системами, обеспечению ее совместимости
- Решения по режимам функционирования, диагностированию работы системы
- Решения по персоналу и режимам его работы
- Сведения об обеспечении заданных в техническом задании потребительских характеристик системы, определяющих ее качество
- Состав функций, комплексов задач, реализуемых системой
- Состав и размещение комплексов технических средств
- Решения по составу информации, объему, способам ее организации, видам машинных носителей, входным и выходным документам и сообщениям, последовательности обработки информации и другим компонентам
- Методы и средства разработки

3. Мероприятия по подготовке объекта автоматизации к вводу системы в действие

- Мероприятия по подготовке информационной базы
- Мероприятия по подготовке персонала
- Мероприятия по организации рабочих мест
- Мероприятия по изменению объекта автоматизации
- Прочие мероприятия

Названия разделов и подразделов могут быть изменены на более конкретные.

Подразделы являются необязательными и описываются при необходимости.

Пояснительная записка, служащая для пояснения и перечисления практически всех работ, произведенных во время технического проектирования, составляется на любую программу или автоматизированную систему управления.

6.1.4. Государственные стандарты для разработки эскизного и технического проектов

При разработке эскизного и технических проектов следует руководствоваться следующими государственными стандартами:

- ГОСТ 34.201-89. Виды, комплектность и обозначение документов при создании автоматизированных систем.
- ГОСТ 2.106-96. Единая система конструкторской документации. Текстовые документы.
- ГОСТ 2.119-2013. Единая система конструкторской документации. Эскизный проект.
- ГОСТ 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Термины и определения.
- ГОСТ 34.601-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Стадии создания
- РД 50-34.698-90 Методические указания. ИТ. Комплекс стандартов и руководящих документов на АС. АС. Требования к содержанию документов.

6.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

6.2.1. Практическое задание

В качестве практического задания необходимо на основе результатов, полученных при выполнении лабораторных работ №1 — №5, оформить документ «Пояснительная записка к техническому проекту», сделав основной упор на разделе «2. Основные технические решения». При составлении документа руководствоваться документом РД 50-34.698-90.

6.2.2. Список контрольных вопросов для самопроверки

1. Каковы основные отличия эскизного проекта от технического?
2. В каких случаях эскизный проект может не создаваться?
3. В каких случаях технический проект может не создаваться?
4. Как называется один из основных документов, составляемых на этапе технического проектирования.

6.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

7. ЛАБОРАТОРНАЯ РАБОТА №7. Проектирование БД

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков проектирования баз данных.

Введение

Проектирование баз данных — процесс создания схемы базы данных и определения необходимых ограничений целостности.

В данной лабораторной работе рассматриваются основные этапы проектирования БД, основные нотации, используемые для описания моделей БД.

7.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

7.1.1. Понятие базы данных

База данных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных (ГОСТ Р ИСО МЭК ТО 10032-2007: Эталонная модель управления данными = ISO/IEC TR 10032:2003 Information technology — Reference model of data management).

База данных — совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, причём такое собрание данных, которое поддерживает одну или более областей применения (ISO/IEC 2382:2015 Information technology — Vocabulary).

Постоянные данные в среде базы данных включают в себя *схему* и *базу данных*. Схема включает в себя описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных. База данных включает в себя набор постоянных данных, определённых с помощью схемы. (ISO/IEC TR 10032:2003).

БД классифицируют по различным критериям: по модели данных, по среде постоянного хранения, по содержимому и др.

Система управления базами данных (СУБД, англ. *DBMS*) — совокупность программных и лингвистических средств общего или специального назначения,

обеспечивающих управление созданием и использованием баз данных (ISO/IEC TR 10032:2003).

Важно понимать, что СУБД — это та часть информационной системы, которая скрывает детали физического хранения на носителях, позволяя работать с БД на уровне логических концепций (модели данных).

7.1.2. Этапы проектирования модели БД

При проектировании БД используются следующие уровни абстракции:

- Описание предметной области (текст, спецификации, «на пальцах»).
- Формальная модель предметной области (семантическая/концептуальная схема/модель).
- Конкретная схема в рамках выбранной модели данных (логическая схема/модель).
- Конкретная схема в рамках выбранной СУБД (физическая схема/модель).
- БД как то, к чему адресуются запросы (БД на уровне представления, на логическом уровне).
- БД как файлы, байты (БД на уровне реализации, на физическом уровне).
- БД как материальный объект (уровень носителя: жёсткий диск, флэш-карта).

На начальном этапе проектирования БД производится описание предметной области. Обычно это осуществляется в произвольной текстовой форме, нередко со слов заказчика:

Хочу создать базу данных «кулинарная книга». В неё буду вносить точные рецепты (чего и сколько), и чтобы их можно было распределять по категориям (салаты, напитки и т.п.). Да, и чтобы я могла сделать что-то вроде: «Выдай-ка мне все рецепты, в которых используется сельдерей».

Выделяют следующие этапы проектирования модели БД:

1. Формальная модель предметной области (семантическая/ концептуальная схема/модель).

2. Конкретная схема в рамках выбранной модели данных (логическая схема/модель).

3. Конкретная схема в рамках выбранной СУБД (физическая схема/модель).

Для уточнения концептуальной схемы/модели БД нужно:

- формализовать «язык» моделирования (средство), которым описывать модель БД (результат);
- нужна графическая нотация, чтобы рисовать.

Примеры семантических моделей (средств моделирования):

- Модель «сущность — связь», Entity-Relationship Model, ER-Model, ER-Модель.
- Модель UML (Unified Modeling Language).

7.1.3. ER-модель

ER-модель в 1976 г. предложил Петер Пин-Шен Чен (Peter Pin-Shan Chen). Модель стала наиболее популярным средством формального описания концептуальных схем БД.

Основные элементы:

- сущность (entity);
- атрибут, свойство (property);
- связь (relationship).

Сущность (entity)

Сущность — это некоторый различимый тип объекта. Например: рецепт, категория, ингредиент, количество.

Сущность имеет свойства. Например: название блюда, значение, единица измерения.

Сущность потенциально состоит из множества экземпляров. Например: рецепт салата «Оливье», рецепт торта «Наполеон».

Атрибут, свойство (property)

- Имеет название.
- Имеет тип (data type, domain).
- У экземпляра сущности имеет значение (value).
- Имеет признак обязательности значения (mandatory).

- Может иметь правила (ограничения на значения, ограничения целостности).
- Может иметь значение по умолчанию (default value).
- Может быть идентификатором или входить в составной идентификатор:
 - основной (первичный) идентификатор (primary identifier)
 - неосновной (альтернативный) идентификатор (alternative identifier).

Связь (relationship)

Связь — это нечто, что связывает две или более сущностей: поставщик—товар, работник—отдел, супруг—супруга, студент— предмет—семестр...

В отличие от сущности связь не имеет собственных свойств. Тип (мощность) связи характеризует, сколько может быть экземпляров сущностей — участников связи:

- один к одному (1:1);
- один ко многим (1:N) или многие к одному (N:1);
- многие ко многим (N:M).

Обязательность связи данной сущности характеризует, должен ли в неё обязательно входить хотя бы один экземпляр этой сущности.

Сильные (обычные) и слабые сущности

Сильная (обычная) сущность имеет естественный идентификатор, который однозначно отличает её экземпляры. Слабая (зависимая) сущность не имеет такого естественного идентификатора, и её экземпляры уникальны только вместе со связью с экземпляром сильной сущности (или нескольких) и не могут существовать без связи с экземпляром сильной сущности (или нескольких).

Пример:

Сущность Дом (Город, Улица, Номер, Материал, Количество_этажей)

Сущность Квартира (Номер, Этаж, Количество_комнат, Общая площадь)

Связь Дом—Квартира

Здесь Дом — сильная сущность, Квартира — слабая сущность.

Идентифицирующая связь

Неидентифицирующая (обычная) связь связывает независимые (обычные) сущности.

Идентифицирующая связь связывает слабую сущность с нужной её сильной.

7.1.4. ER-модель: нотации

Существуют разные нотации ER-модели:

- Нотация Чена;
- Нотация Crow's Foot;
- UML;
- и др.

Нотация Чена

В нотации Чена Множества сущность изображается в виде прямоугольника, отношение между сущностями изображается в виде ромба. Если сущности участвует в отношении, они связаны линией. Если отношение не является обязательным, то линия пунктирная. Мощность отношения подписывается у концов линии. Атрибуты изображаются в виде овалов и связываются линией с одним отношением или с одной сущностью (рис. 7.1).

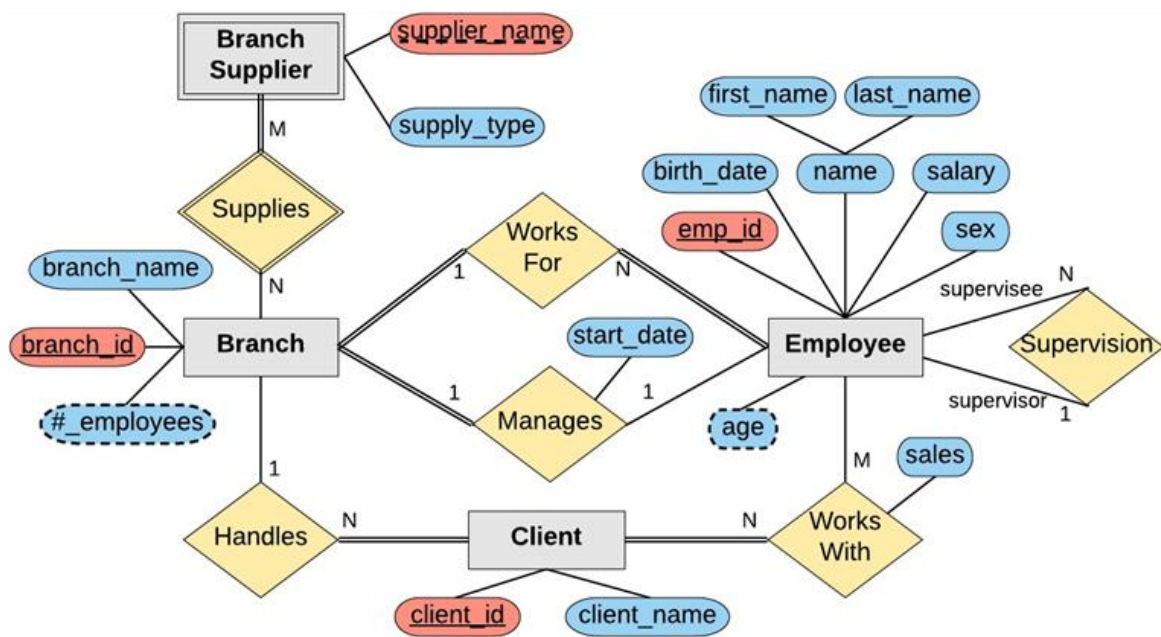


Рис. 7.1. Нотация Чена

Нотация Crow's Foot

В этой нотации сущность изображается в виде прямоугольника, содержащего её имя, выражаемое существительным. Имя сущности должно быть уникальным в

рамках одной модели. При этом имя сущности — это имя типа, а не конкретного экземпляра данного типа.

Связь изображается линией, которая связывает две сущности, участвующие в отношении. Мощност конца связи указывается графически, множественность связи изображается в виде «вилки» на конце связи. Модальность (обязательность) связи также изображается графически: необязательность связи помечается кружком на конце связи, обязательность — вертикальной черточкой.

Атрибуты сущности записываются внутри прямоугольника, изображающего сущность, и выражаются существительным в единственном числе (рис. 7.2).

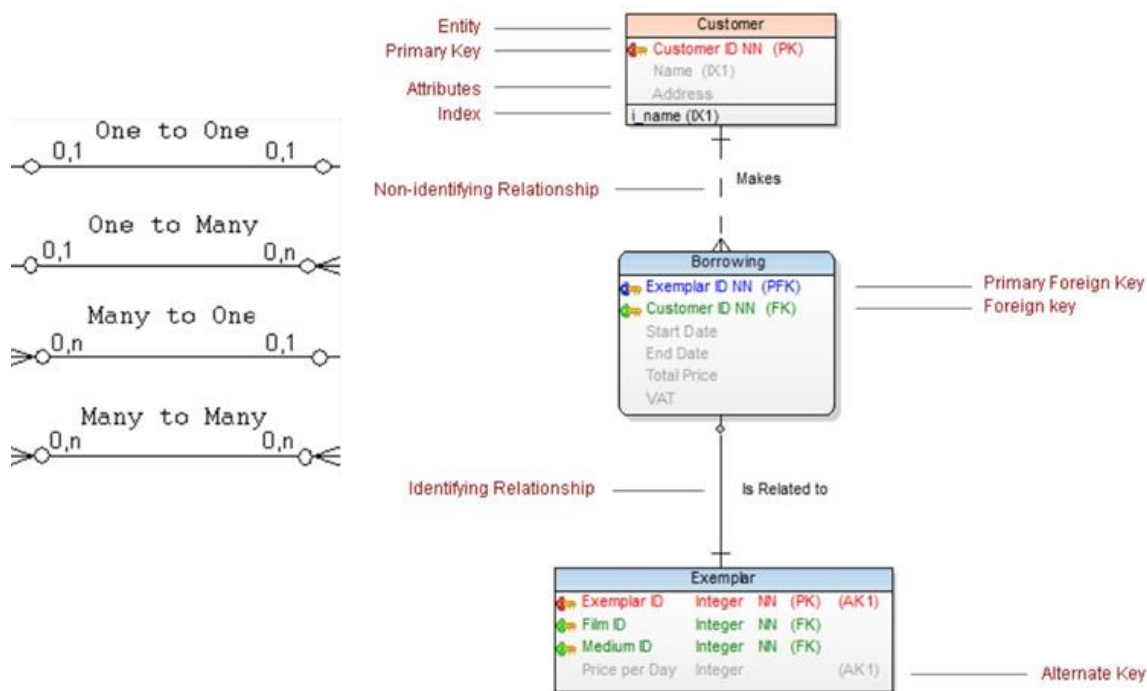


Рис. 7.2. Нотация Crow's Foot

7.1.5. Инструменты проектирования БД

Существует большое количество инструментов проектирования БД. Выделим некоторые часто упоминаемые:

- PowerDesigner (SAP);
- Toad Data Modeler (Quest Software) (<https://www.toadworld.com/download/toad-data-modeler/freeware>);
- ERwin Data Modeler (erwin, Inc.).

Список современных инструментов проектирования БД и сравнение их функциональности можно увидеть в статье «Comparison of data modeling tools» (https://en.wikipedia.org/wiki/Comparison_of_data_modeling_tools).

7.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

7.2.1. Практическое задание

В качестве практического задания необходимо создать ER-диаграмму в нотации Чена, отражающую модель «сущность-связь» предметной области, для которой предназначена разрабатываемая студентом система. Для рисования можно использовать как встроенный векторный редактор в Microsoft Word, так и специализированные редакторы (см. п. 7.1.5). При выполнении задания использовать результаты лабораторных работ № 1—3.

7.2.2. Список контрольных вопросов для самопроверки

1. Какие могут быть источники для описания предметной области?
2. Чем отличаются представления модели на логическом и физическом уровнях?
3. Что характеризует обязательность связи сущности?

7.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

8. ЛАБОРАТОРНАЯ РАБОТА №8. Проектирование БД.

Логическая модель

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков логического проектирования для реляционной модели базы данных.

Введение

Логическое проектирование — это следующий после концептуального этап проектирования базы данных. На этапе логического проектирования учитывается специфика конкретной модели данных (реляционной, иерархической, сетевой и т.д.), но не учитывается специфика конкретной СУБД. В данной лабораторной работе рассматривается процесс создания логической модели в среде Toad Data Modeler.

В свою очередь логическая модель данных является основой для следующего этапа — физического проектирования.

8.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

8.1.1. Toad Data Modeler

Toad Data Modeler — это инструмент проектирования баз данных, позволяющий визуально создавать, поддерживать и документировать новые или существующие системы баз данных.

Основные особенности Toad Data Modeler:

Поддержка нескольких баз данных — подключение несколько баз данных одновременно, включая Oracle, SAP, MySQL, SQL Server, PostgreSQL, DB2, Ingres и Microsoft Access.

Инструмент моделирования данных — создание структур базы данных или автоматическое внесение изменений в существующие модели и предоставление документации на нескольких платформах.

Логическое и физическое моделирование — создание сложных моделей логических и физических сущностей.

Отчетность — создание подробных отчетов о существующих структурах базы данных.

В данной лабораторной работе будут использованы средства и инструменты логического и физического моделирования.

Для создания новой логической модели нужно выполнить команду из основного меню *File*→*New* →*Model*. В появившемся окне *New Model* переключитесь на вкладку *Logical Data Model* и подтвердите создание модели.

Интерфейс пользователя Toad Data Modeler имеет классическую компоновку: основное меню ❶, панели инструментов ❷, информационные панели ❸ и основную рабочую область ❹ (рис. .10.1).

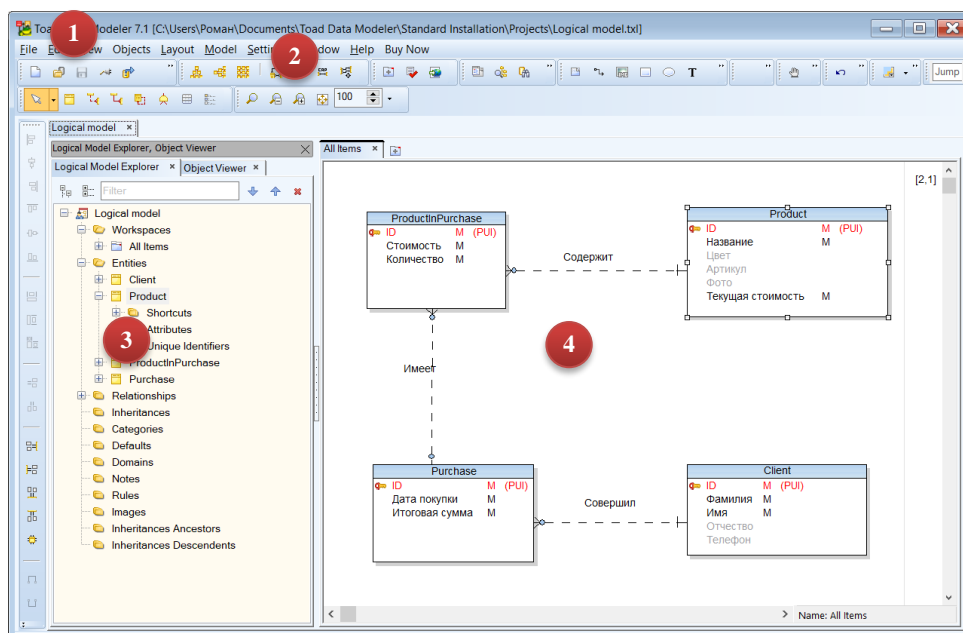




Рис. 8.1. Основное окно Toad Data Modeler

В системе поддерживается иерархическая структура элементов проекта. В одном проекте может находиться произвольное число логических и физических моделей. Каждая модель в проекте располагается на отдельной вкладке. Эта вкладка состоит из Обзорателя модели ❸ и основной рабочей области ❹. Обзоратель модели позволяет просматривать и модифицировать её структуру. В рабочей области в виде отдельных вкладок отображаются диаграммы. Элементы модели показываются в Обзорателе модели и на графически на диаграммах.

8.1.2. Создание и настройка сущностей



Добавление сущности на диаграмму осуществляется из панели *Model Object* с помощью

инструмента *Entity* . Для добавления новой сущности активируйте инструмент и щелкните на диаграмме. Новая сущность создается без атрибутов. Для изменения имени сущности дважды щелкните инструментом выбора  по сущности на диаграмме или используйте команду *Edit* в контекстном меню фигуры или узла сущности в Обзорера (рис. 8.2).

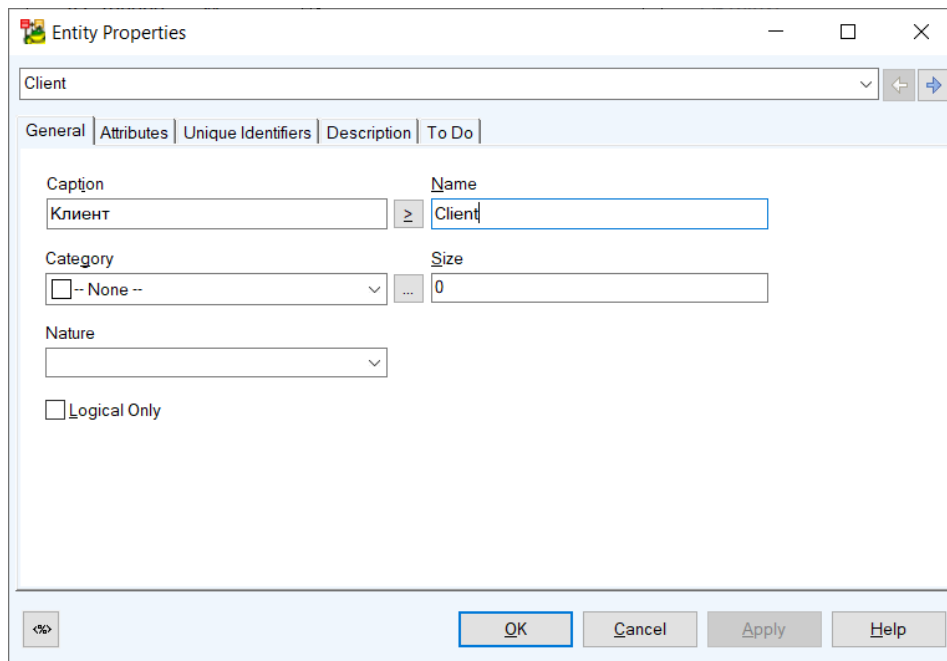


Рис. 8.2. Основные свойства сущности

У сущности кроме имени (*Name*) есть подпись (*Caption*) На панели *Display* можно настроить как отображать сущности (рис. 8.3).

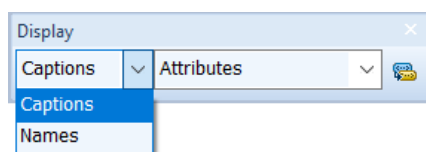


Рис. 8.3. Параметры отображения сущности

Изменение атрибутов сущности реализуется также в окне свойств сущности, во вкладке *Attributes* (рис. 8.4). Как и у самой сущности, у атрибута кроме имени (*Name*) есть подпись (*Caption*). Для каждого атрибута задается тип данных (*Data Type*) и обязательность (*Mandatory*). Изменение свойств атрибута выполняется для каждого атрибута отдельно в окне свойств атрибута (рис. 8.5).

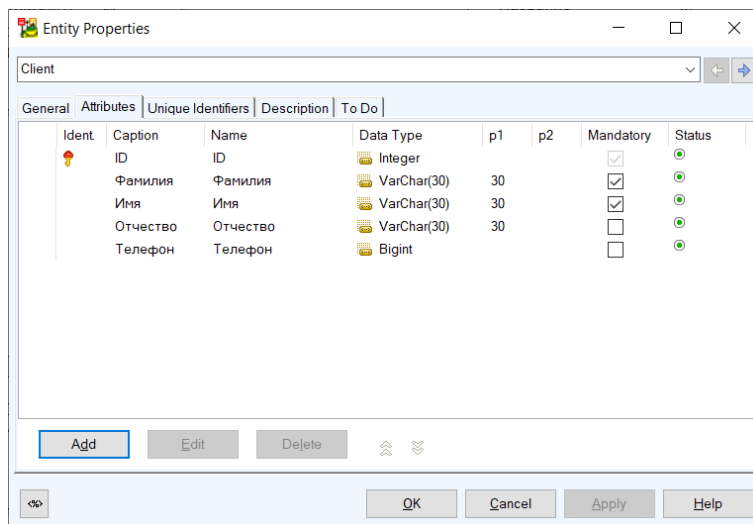


Рис. 8.4. Атрибуты сущности

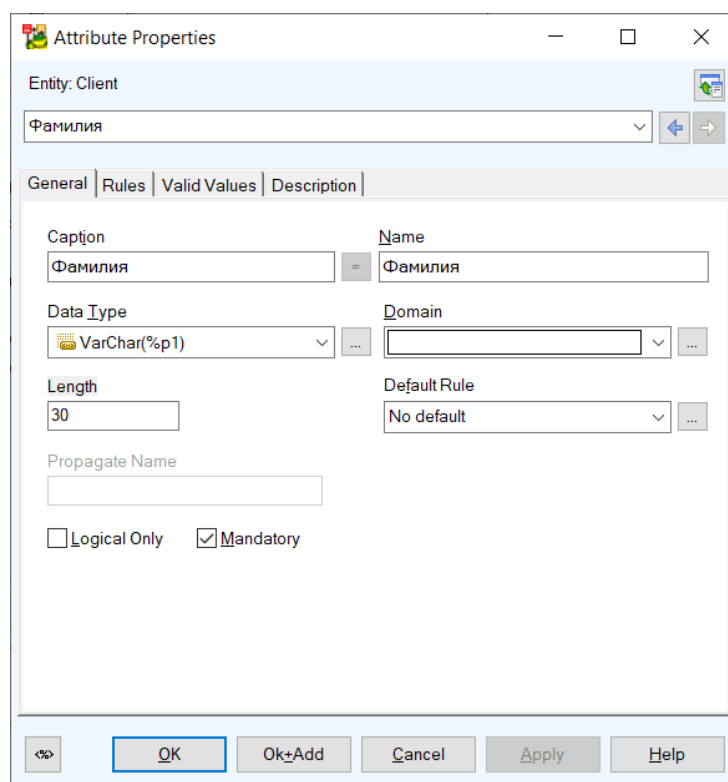


Рис. 8.5. Свойства атрибута




В окне свойств сущности во вкладке *Unique Identifiers* настраиваются уникальные идентификаторы.


8.1.3. Создание и настройка связей

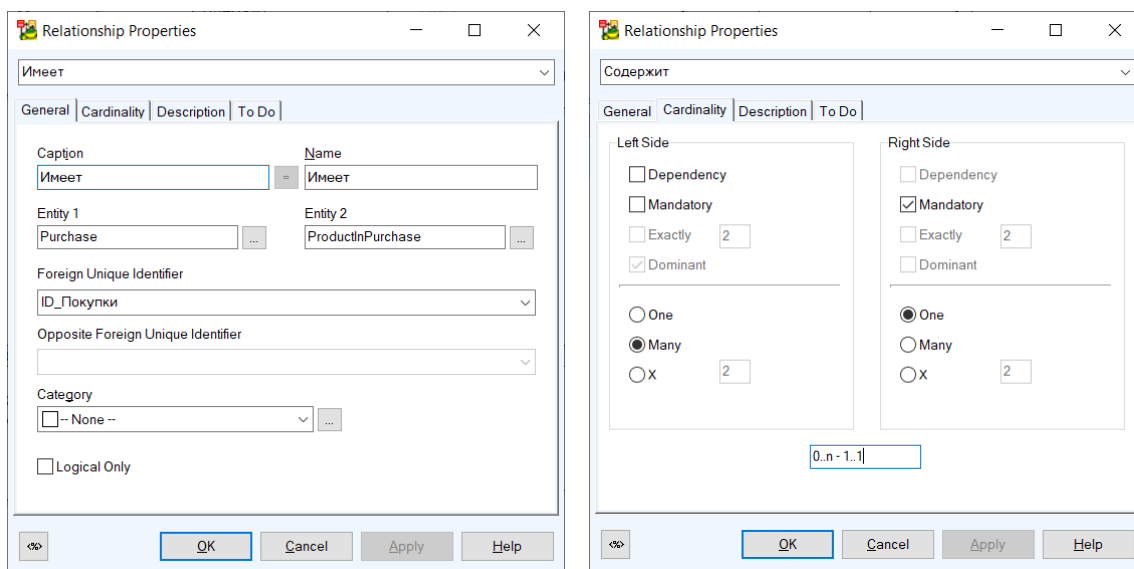


Добавление связей (отношений) между сущностями на диаграмму осуществляется из панели

Model Object с помощью инструментов:

- | | | |
|---|-------------------------------------|------------------------------|
|  | <i>Non-identifying Relationship</i> | Не идентифицирующая связь |
|  | <i>Identifying Relationship</i> | Идентифицирующая связь |
|  | <i>Self Relationship</i> | Связь сущности с самой собой |

Для добавления связи активируйте соответствующий инструмент и щелчками соедините две нужные сущности. После создания связи дважды щелкните инструментом выбора  по связи на диаграмме или используйте команду *Edit* в контекстом меню связи. Настройка свойств связи осуществляется в окне свойств связи (рис. 8.5).



а)

б)

Рис. 8.5. Свойства связи

8.1.4. Удаление элементов диаграммы

Одна и та же сущность или связь может быть представлена на разных диаграммах. Поэтому при удалении элемента из диаграммы нужно указывать: удаляется сам элемент (сущность или связь) из модели или только её графическое отображение (рис. 8.6).

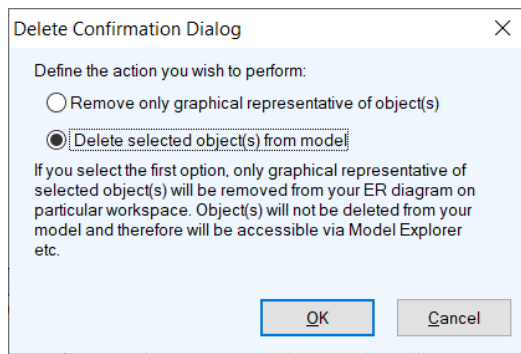


Рис. 8.6. Удаление элемента из диаграммы

В первом случае выбранный элемент удаляется только как графический элемент из диаграммы, но в модели он остается. И его позже можно будет вновь добавить на диаграмму. Во втором случае элемент удаляется из модели и, как следствие из диаграмм(ы).

8.1.5. Проверка модели

Toad Data Modeler имеет встроенные средства проверки корректности модели. Для запуска проверки выполните команду *Model*→*Verify Model*. В появившемся окне *Verification* можно настроить параметры проверки: объекты для проверки и параметры самой проверки. По умолчанию проверке подлежат все сущности и связи модели со всеми типа проверок (ошибки, предупреждения, подсказки), их можно менять.

После подтверждения операции система запускает проверку модели. Результаты проверки показываются в панели *Verification Log* (рис. 8.7).

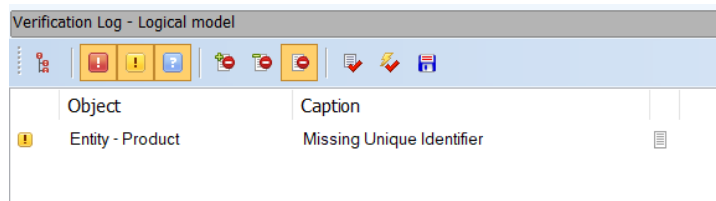


Рис. 8.7. Результаты проверки

После исправления всех ошибок текущий проект необходимо сохранить — он понадобится для выполнения следующей лабораторной работы.

8.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

8.2.1. Практическое задание

В качестве практического задания необходимо создать логическую модели базы данных системы, разрабатываемой студентом. При выполнении задания использовать результаты лабораторной работы №7.

8.2.2. Список контрольных вопросов для самопроверки

1. Какая нотация ER-модели используется в Toad Data Modeler?
2. Если сущность графически представлена на двух диаграммах, то что будет если сущность удалить на одной из диаграмм?
3. Можно ли использовать Toad Data Modeler совместно с какой-либо системой управления версиями?

8.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

9. ЛАБОРАТОРНАЯ РАБОТА №9. Проектирование БД.

Физическая модель

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков физического проектирования БД для реляционной СУБД.

Введение

Физическое проектирование — это следующий после логического этап проектирования базы данных. На этапе физического проектирования, в отличие от логического, учитывается специфика конкретной СУБД, для которой создается модель. В данной лабораторной работе рассматривается процесс создания физической модели в среде Toad Data Modeler.

9.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

9.1.1. Подходы к созданию физической модели

В Toad Data Modeler предлагает два подхода к созданию физической модели:

- создание физической модели с нуля;
- преобразование логической модели в физическую.

В первом случае техника создания физической модели эквивалента созданию логической модели. Второй подход используется при наличии логической модели. В нашем случае логическая модель была создана при выполнении предыдущей лабораторной работы.

Для создания физической модели на основе логической сначала нужно открыть проект, содержащий нужную логическую модель. Напомним, что в Toad Data Modeler проект может содержать произвольное число логических и физических моделей.

9.1.2. Создание физической модели из логической

Для создания физической модели из логической необходимо активировать вкладку с нужной логической моделью и выполнить команду Model→Convert Model. Это приведет к запуску мастера Model Conversion. На первом шаге нужно выбрать целевую СУБД (рис. 9.1).

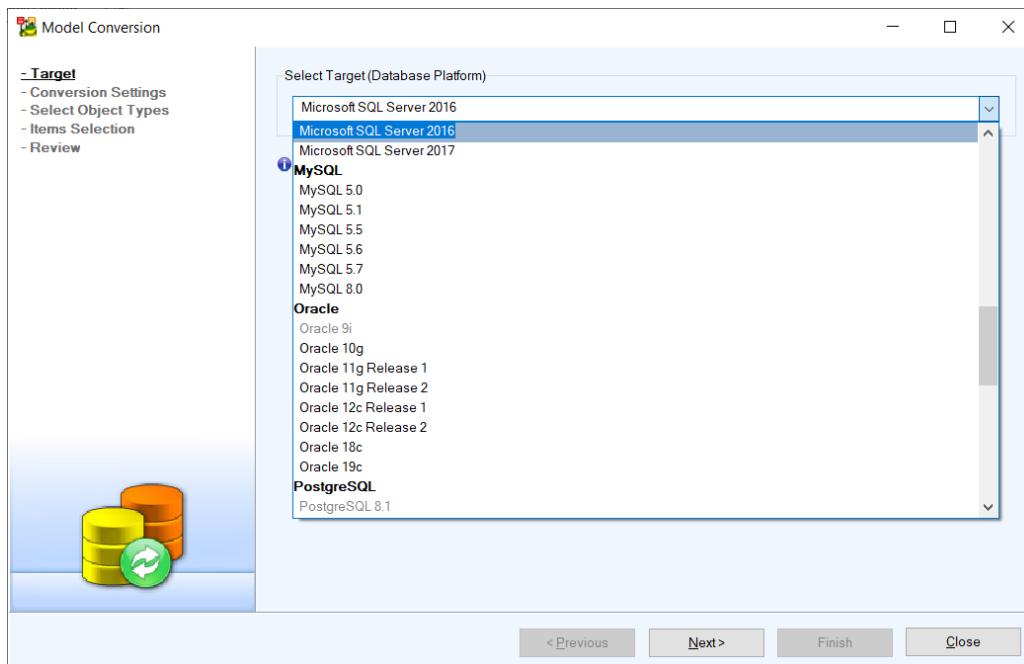


Рис. 9.1. Мастер преобразования модели. Выбор целевой СУБД

На втором шаге задаются настройки преобразования. На третьем – объекты логической модели, которые будут использованы в процессе. Можно настроить какие именно объекты использовать, а какие нет. По умолчанию система предлагает использовать все объекты. В режиме расширенных настроек можно указать какие именно объекты и какие их свойства необходимо переносить в физическую модель (рис. 9.2).

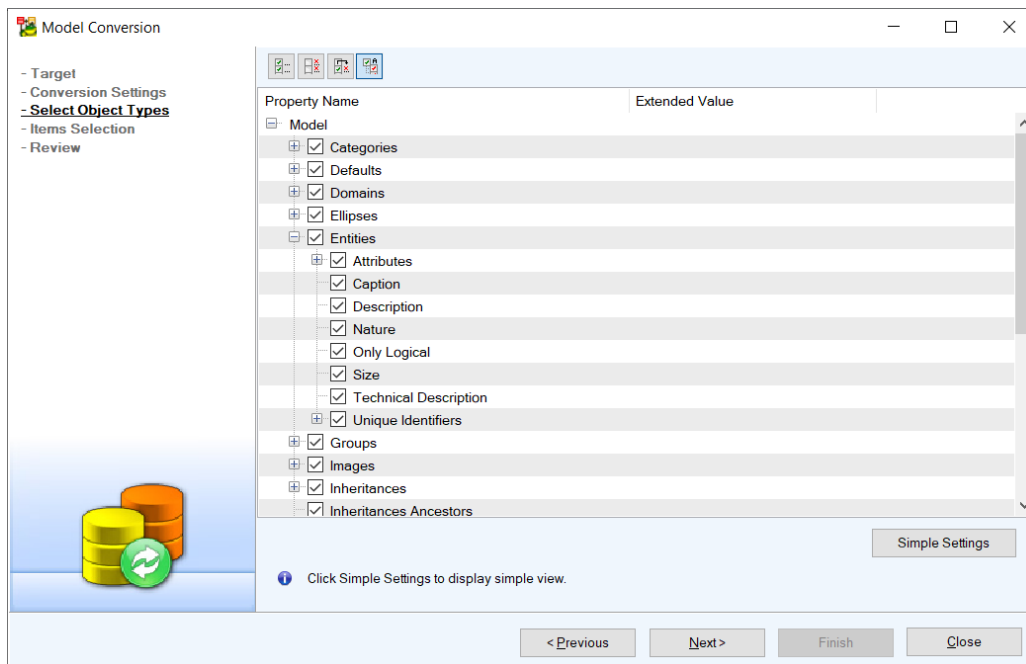


Рис. 9.2. Мастер преобразования модели. Выбор типов объектов

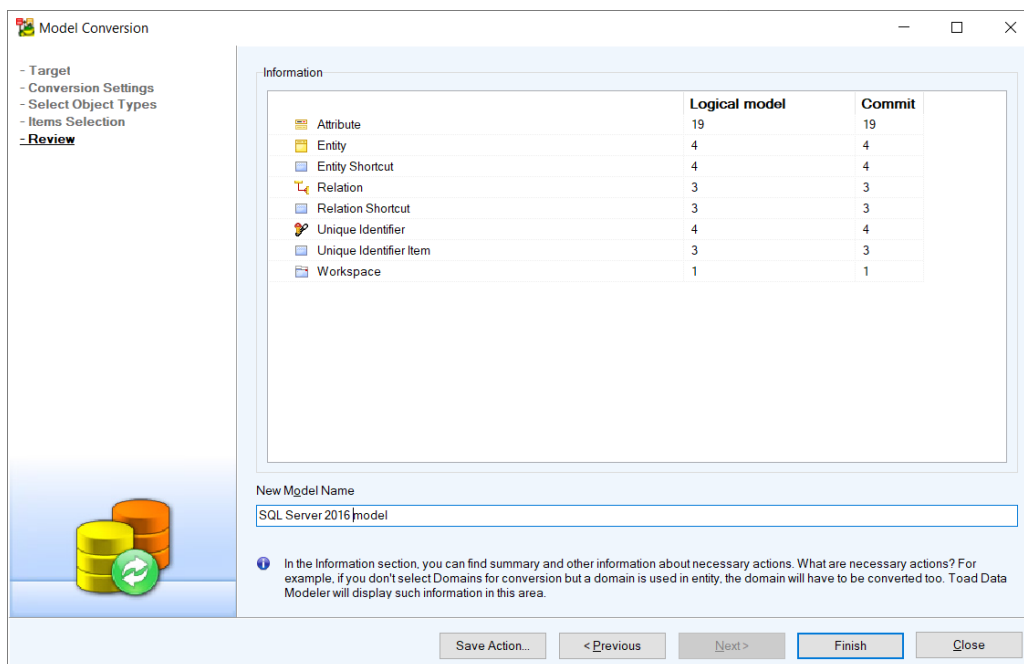


Рис. 9.2. Мастер преобразования модели. Ввод имени физической модели

9.1.3. Коррекция физической модели

После запуска процесса преобразования система генерирует физическую модель и открывает ее в виде отдельной вкладки (рис. 9.3).

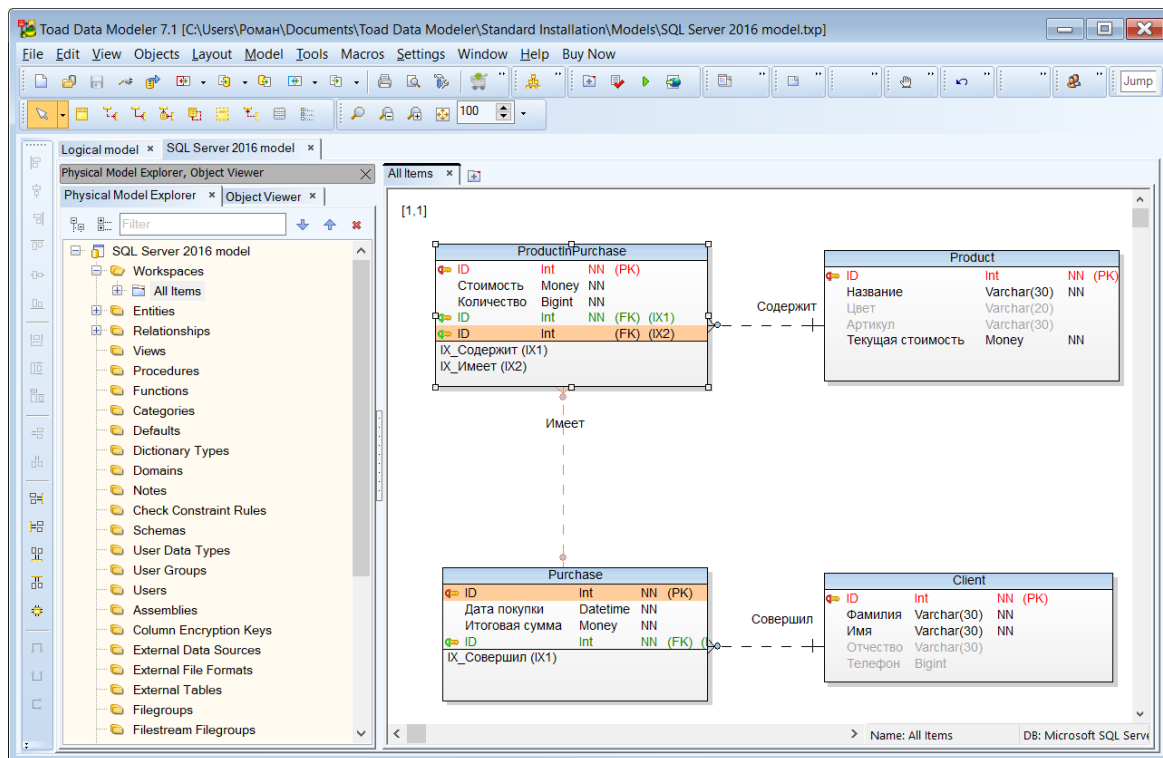


Рис. 9.3. Результат преобразования логической модели в физическую

При преобразовании система анализирует связи между сущностями и автоматически формирует необходимые внешние ключи. При этом в качестве имени внешнего ключа используется имя первичного ключа связанной сущности. Если у двух связанных сущностей идентифицирующие атрибуты имели одинаковые имена, то после преобразования в имя первичного ключа будет совпадать с именами внешних ключей (рис. 9.4)!

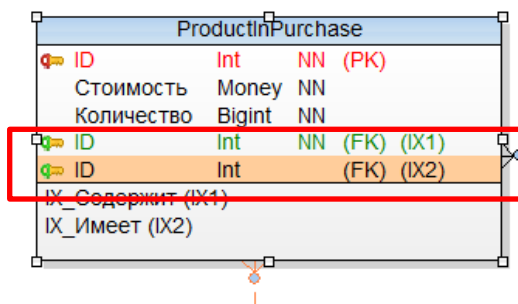


Рис. 9.4. Сгенерированные внешние ключи

Поэтому такую ситуацию нужно исправить вручную через свойства атрибута. Для этого необходимо зайти в свойства таблицы (в терминологии системы *Entity*

Properties), вкладка атрибутов (*Attributes*) и переименовать их, так чтобы названия внешних ключей были уникальными (рис. 11.5).

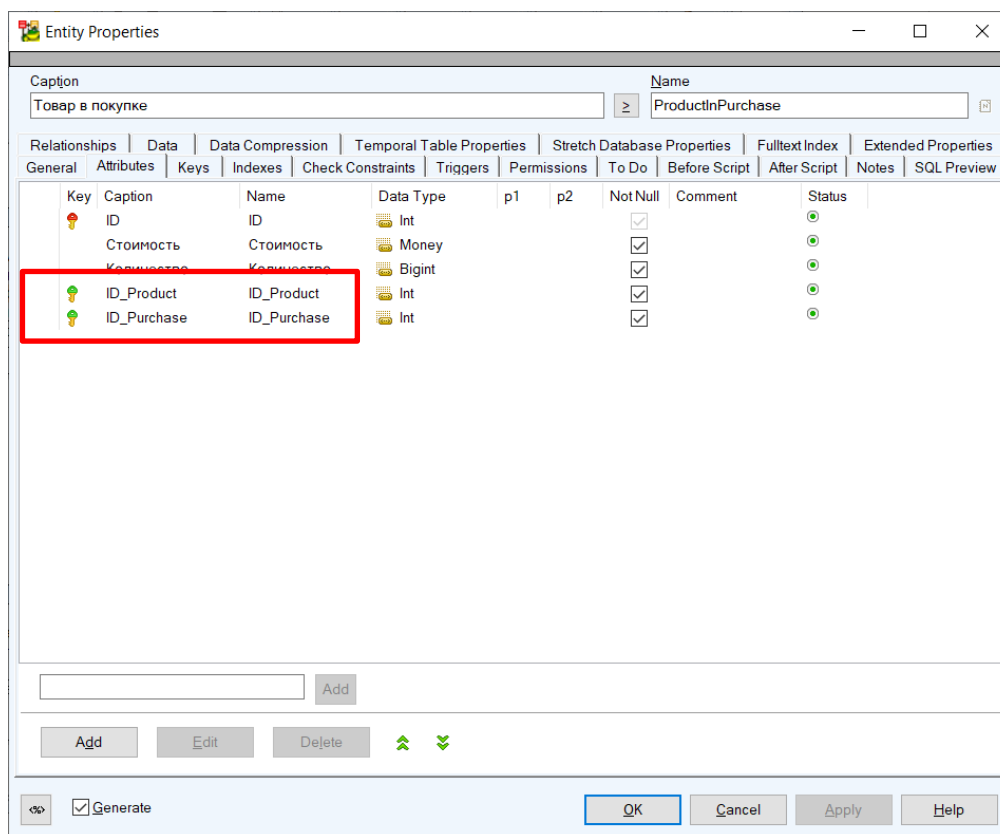


Рис. 9.5. Скорректированные имена внешних ключей

Подобную коррекцию нужно выполнить для всех таблиц, где есть подобная проблема.

Обратите внимание, что система автоматически формирует ключи, индексы и другие элементы. При необходимости их можно скорректировать.

9.1.4. Проверка модели

Проверка физической модели выполняется аналогично проверке логической модели (см. п.п 8.1.5).

9.1.5. Генерация скрипта создания базы данных

Toad Data Modeler позволяет сгенерировать скрипт для создания базы данных в целевой СУБД. Для этого необходимо выполнить команду *Model→Generate DDL Script→Run...* Далее в появившемся окне (рис. 9.6) при необходимости изменить

настройки и нажать кнопку Generate. Результат можно показать, нажав кнопку Show Code (рис. 9.7) или открыв скрипт во внешнем редакторе.

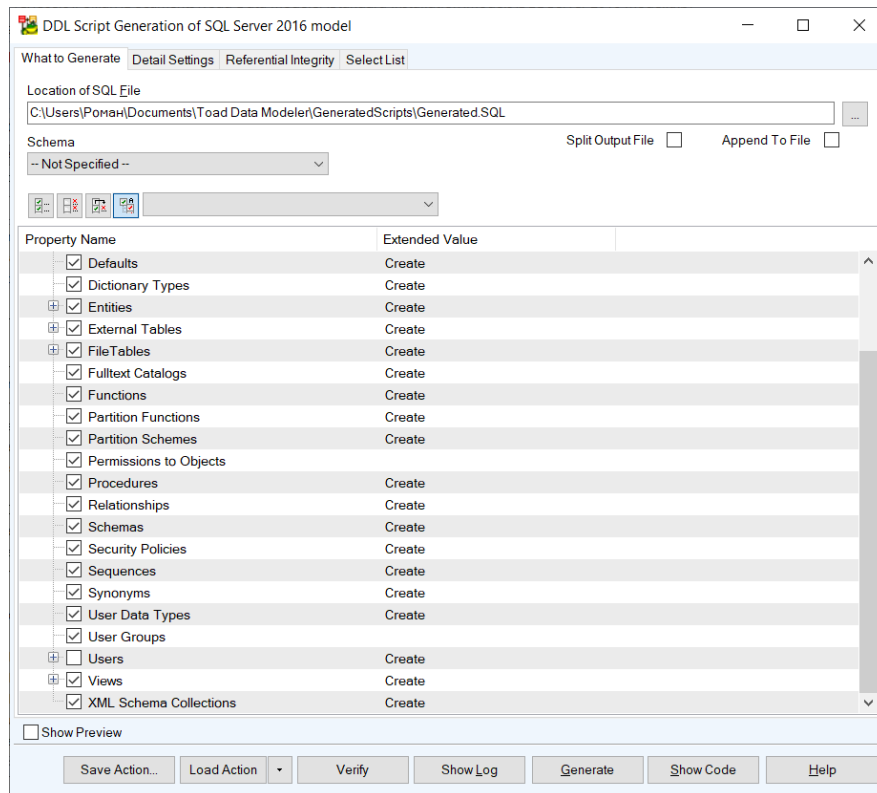
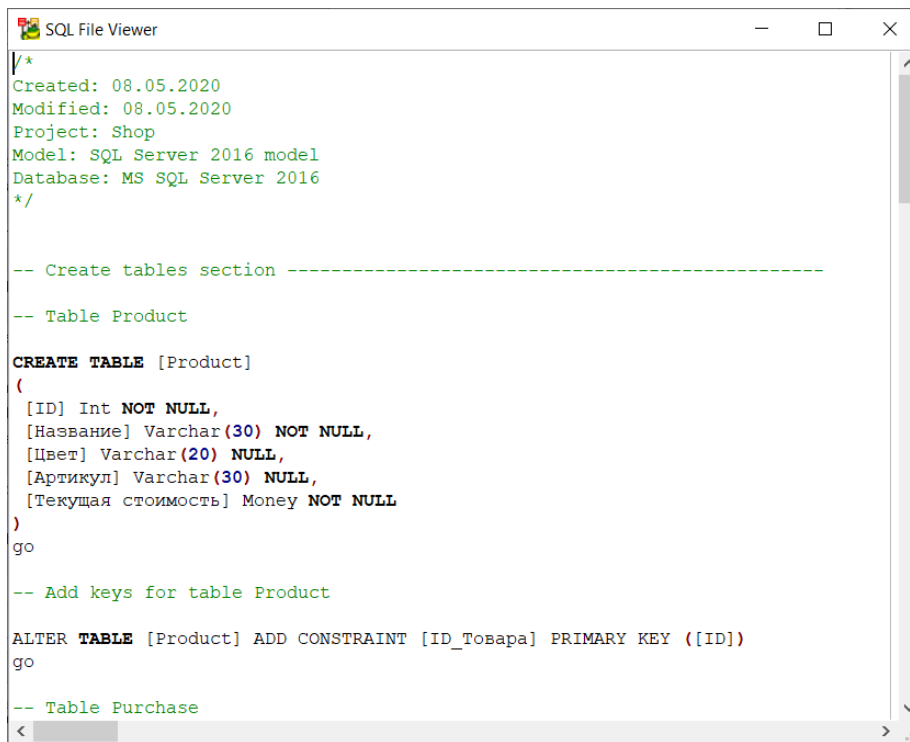


Рис. 9.6. Настройки генерации скрипта



```
SQL File Viewer
/*
Created: 08.05.2020
Modified: 08.05.2020
Project: Shop
Model: SQL Server 2016 model
Database: MS SQL Server 2016
*/

-- Create tables section -----
-- Table Product

CREATE TABLE [Product]
(
  [ID] Int NOT NULL,
  [Название] Varchar(30) NOT NULL,
  [Цвет] Varchar(20) NULL,
  [Артикул] Varchar(30) NULL,
  [Текущая стоимость] Money NOT NULL
)
go

-- Add keys for table Product

ALTER TABLE [Product] ADD CONSTRAINT [ID_Товара] PRIMARY KEY ([ID])
go

-- Table Purchase
```

Рис. 9.7. Сгенерированный скрипт

9.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

9.2.1. Практическое задание

В качестве практического задания необходимо преобразовать логическую модель базы данных, разработанную на лабораторной работе №8, в физическую. Целевая СУБД должна выбираться студентом исходя из проектных решений, зафиксированных в Техническом проекте (лабораторной работе №6). Также нужно проверить модель на корректность и после устранения всех ошибок сгенерировать скрипт создания БД. Выполнить сгенерированный скрипт в целевой СУБД.

9.2.2. Список контрольных вопросов для самопроверки

1. Какие способы создания физической модели БД существуют в Toad Data Modeler?
2. Как в Toad Data Modeler обнаружить ошибки в модели?
3. Как в Toad Data Modeler исправить ошибки в модели?

9.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

10. ЛАБОРАТОРНАЯ РАБОТА №10. Инструменты проектирования интерфейса пользователя

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков проектирования интерфейсов пользователя с без применения цифровых инструментов.

Введение

Проектирование интерфейсов пользователя является важной составной частью процесса разработки программного обеспечения. Для проектирования могут использоваться различные инструменты, начиная с обычной ручки и листа бумаги и заканчивая специальными программными системами, позволяющими быстро создавать не только эскизы интерфейсов пользователя, но и интерактивные прототипы. В данной лабораторной работе упор делается на получение навыков разработки черновых эскизов интерфейсов пользователя без применения цифровых инструментов.

10.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

10.1.1. Общие сведения о проектировании UI

При проектировании пользовательских интерфейсов как части человеко-машинного взаимодействия используются следующие аббревиатуры:

- UI — user interface / пользовательский интерфейс;
- UX — user experience / опыт взаимодействия.

При важно понимать, что эти термины не эквиваленты и их не следует путать: пользовательский интерфейс (UI) не является опытом взаимодействия (UX).

Процесс проектирования UI состоит из следующих этапов (рис. 10.1):

1. Создание эскиза
2. Создание эскиза интерфейса пользователя
3. Создание прототипа
4. Создание макета
5. Программная реализация



Рис. 10.1. Процесс проектирования UI

Назначение каждого этапа и используемые на этапе инструменты показаны в таблице:

Этап	Описание	Используемые инструменты
Эскиз	Базовый концепт того, как будет работать (выглядеть) система в пользовательском интерфейсе. Детали и специфика не важны.	Бумага и карандаш Доска и маркер Цифровые графические инструменты
Схема интерфейса	Детально проработан концепт. Детали и специфика важны.	Цифровые графические инструменты Цифровые инструменты для UI
Прототип	Полностью продуманное поведение. Упрощение графики и контента, полное отсутствие бекэнда. Ключевая особенность – интерактивность.	Цифровые инструменты прототипирования UI
Макет	Финальный дизайн графики и контента	Цифровые инструменты графического дизайна
Код	Готовый продукт	Среды разработки

К проектированию UI имеют отношения разные специалисты и роли. В целом процесс проектирования UI по ролям представлен на рис. 12.2.

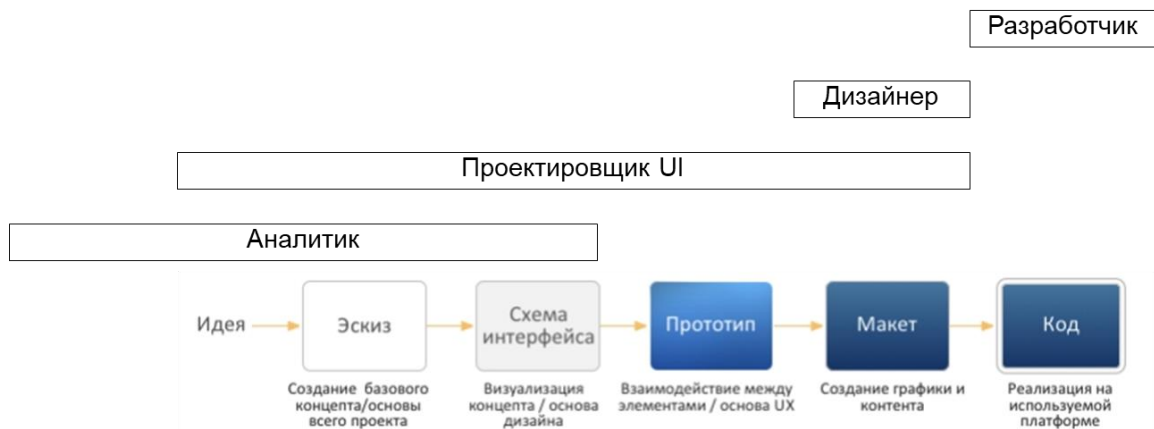


Рис. 10.2. Процесс проектирования UI по ролям

10.1.2. Целевые платформы UI

При проектировании UI важно понимать для какой целевой платформы он разрабатывается (рис. 10.3).







Настольная (Desktop)		Windows
		MacOS
		Linux
Веб (Web)		
Мобильная (Mobile)		Android
		iOS

Рис. 10.3. Типовые платформы

В таблице показаны ключевые показатели типовых платформ:

Платформа		Экран		Средства ввода
		Размер	Ориентация	
Настольная		13" и выше	Альбомная	Клавиатура Мышь
Веб		13" и выше	Альбомная*	Клавиатура Мышь
Мобильная	Смартфоны	4" — 6"	Книжная*	Сенсорный экран
	Планшеты	7" — 10"	Альбомная*	Сенсорный экран

* Типовая

При проектировании UI важно учитывать эти показатели, прежде всего с точки зрения эргономики.

10.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

10.2.1. Практическое задание

В качестве практического задания необходимо *без применения* цифровых инструментов создать эскизы интерфейса пользователя для системы, создаваемой студентом в рамках индивидуального задания на дисциплину. Эскизы должны создаваться для той платформы, которая соответствует индивидуальному заданию:

- настольное приложение для ОС Windows;
- настольное приложение для ОС MacOS;
- настольное приложение для ОС Linux;
- мобильное приложение для ОС Android;
- мобильное приложение для ОС iOS;
- веб-приложение;
- другие (AR/VR, Smart Watch...).

Проектированию подлежат (при наличии):

- главное окно;
- главное меню;
- система навигации;

- все диалоговые окна.

Результаты проектирования должны быть зафиксированы ручкой на листах бумаги или маркером на доске. Все созданные эскизы должны быть сфотографированы и добавлены в отчет по лабораторной работе.

10.2.2. Список контрольных вопросов для самопроверки

1. Какие основные преимущества имеют цифровые инструменты проектирования пользовательских интерфейсов по сравнению с «бумажным» подходом?
2. Существуют ли недостатки цифровых инструментов проектирования пользовательских интерфейсов по сравнению с «бумажным» подходом?

10.2.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

11. ЛАБОРАТОРНАЯ РАБОТА №11. Проектирование интерфейса пользователя

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков проектирования интерфейсов пользователя с помощью специализированного цифрового инструмента.

Введение

Проектирование интерфейсов пользователя является важной составной частью процесса разработки программного обеспечения. Для проектирования могут использоваться различные инструменты, начиная с обычной ручки и листа бумаги и заканчивая специальными программными системами, позволяющими быстро создавать не только эскизы интерфейсов пользователя, но и интерактивные прототипы. В данной лабораторной работе упор делается на получение навыков разработки черновых эскизов интерфейсов пользователя.

11.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

11.1.1. Проектирование UI в среде Evolus Pencil

В качестве базового инструмента студенту предлагается использовать специализированную программу для проектирования интерфейса пользователя [Evolus Pencil](#). Программа является бесплатной и относительно простой. Для выполнения работы студент также может использовать любой другой подобный инструмент по своему усмотрению.

По своей сути программа является специализированным векторным редактором и позволяет создавать дизайн-проект пользовательского интерфейса. Проект представляет собой документ, состоящий из произвольного числа страниц. Основное окно программы показано рис. 11.1. Цифрами обозначены:

- 1 – Меню
- 2 – Панель инструментов
- 3 – Панель фигур
- 4 – Активная страница документ
- 5 – Панель свойств фигуры
- 6 – Панель страниц документа

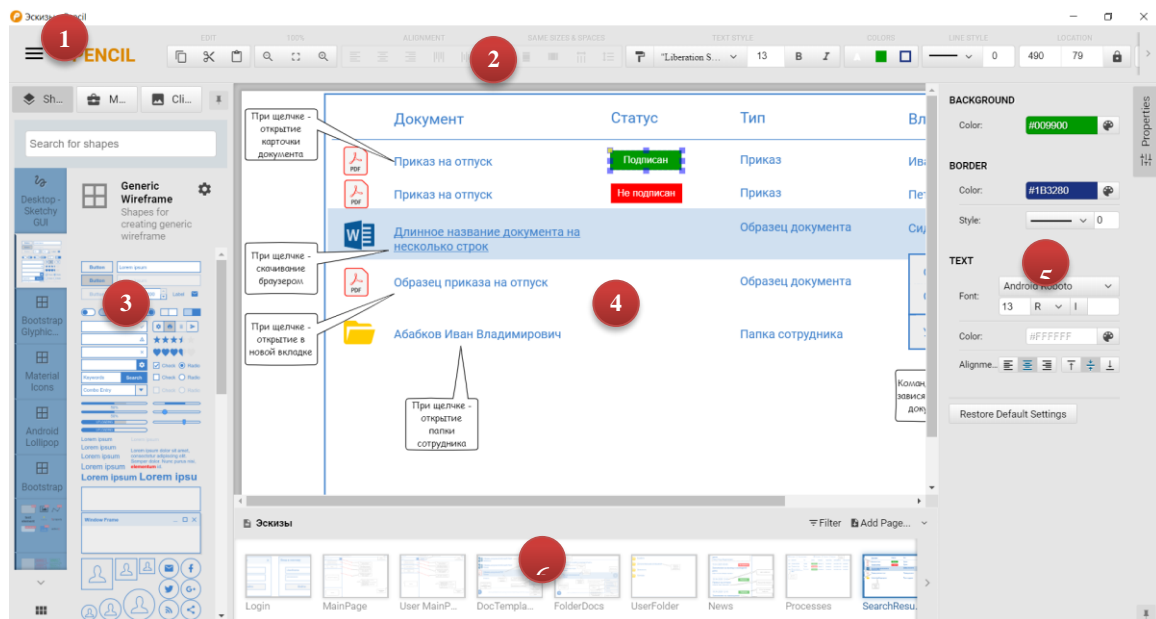


Рис. 11.1. Основное окно Evolus Pencil

Проект не зависит от целевой платформы. Это позволяет в рамках одного проекта создавать эскизы для разных целевых платформ. При этом каждая страница документа может иметь свои размеры. Сделать это можно в свойствах страницы ⑥ (рис. 11.2).

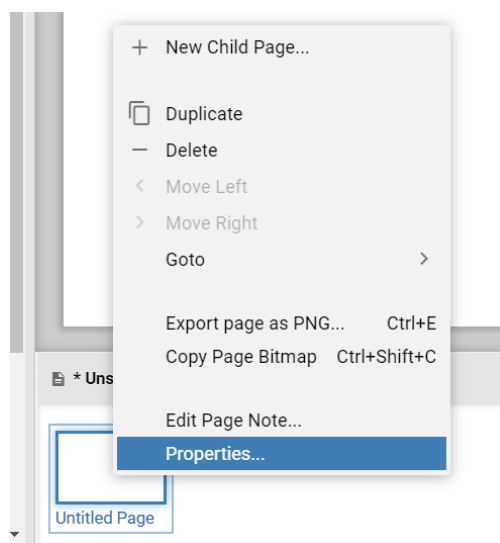


Рис. 11.2. Доступ к свойствам страницы

Общий принцип создания эскиза интерфейса пользователя достаточно прост: из палитры фигур ③ (рис. 11.3) выбирается нужный элемент и перетаскивается на активную страницу ④.

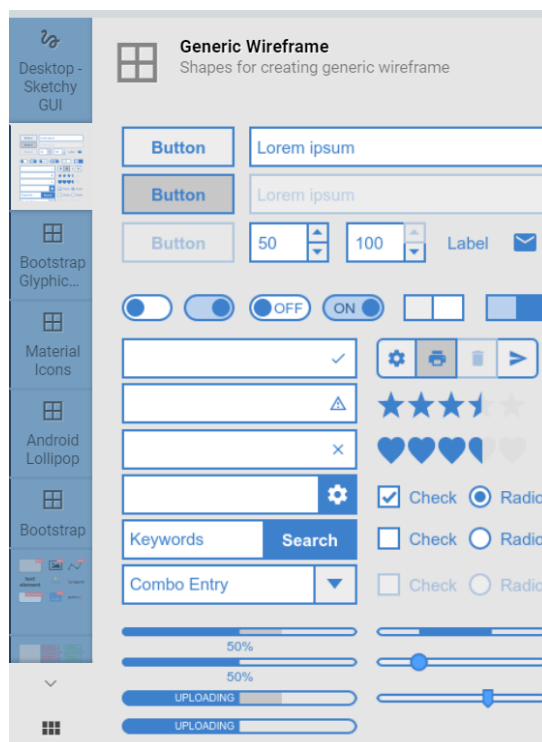


Рис. 11.3. Панель фигур

Все дальнейшие манипуляции с фигурами на канве активной страницы выполняются с помощью классических операций: перемещение, изменение размеров, копирование/вставка и т.д. В процессе перемещения фигур или изменении их размеров программа показывает направляющие, позволяющие точно выровнять фигур относительно друг друга.

Графические свойства фигур можно изменять через панель свойств **5** (рис. 11.4). Для этого фигуру или фигуры необходимо предварительно выбрать.

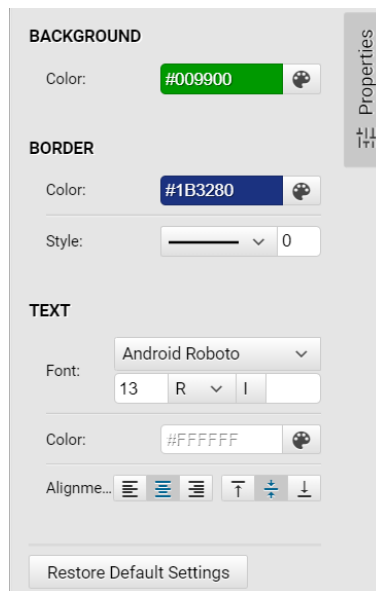




Рис. 11.4. Панель свойств

Панель фигур  разбита на коллекции. Каждая коллекция соответствует определенному набору элементов, подходящих к той или иной платформе. Для управления коллекциями нужно нажать кнопку  в нижней части (рис. 11.3). Можно отобразить/скрыть необходимые коллекции или добавить из файла или репозитория (рис. 11.5). Изучите коллекции, расположенные в репозитории. Часть коллекций специально предназначена для отдельных целевых платформ, особенно это касается мобильных платформ (рис. 11.6).

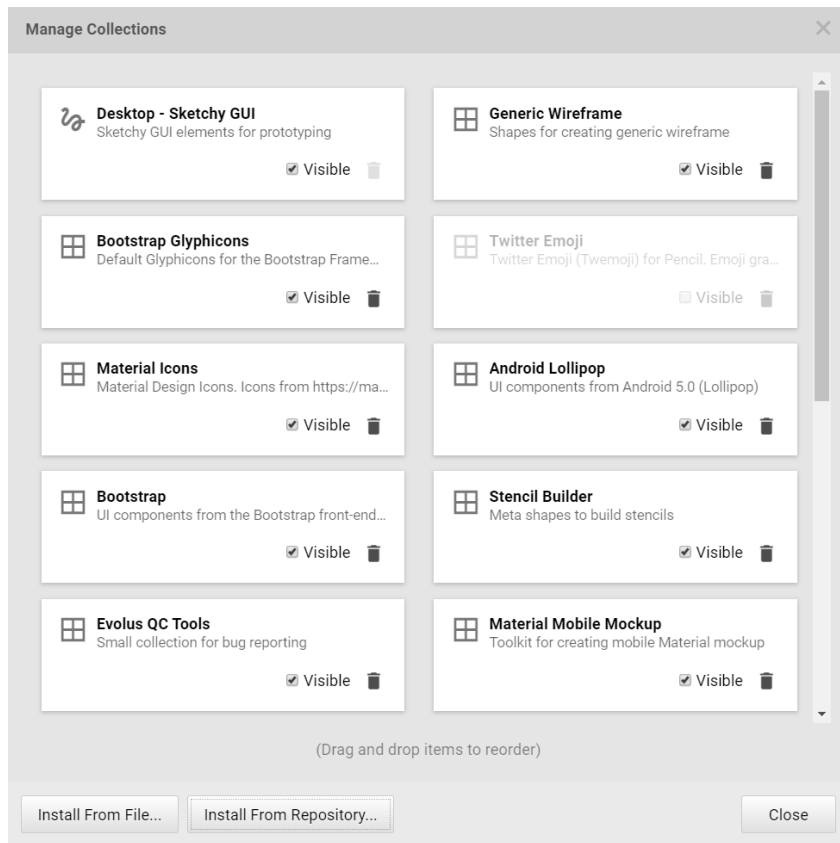


Рис. 11.5. Настройки коллекций фигур

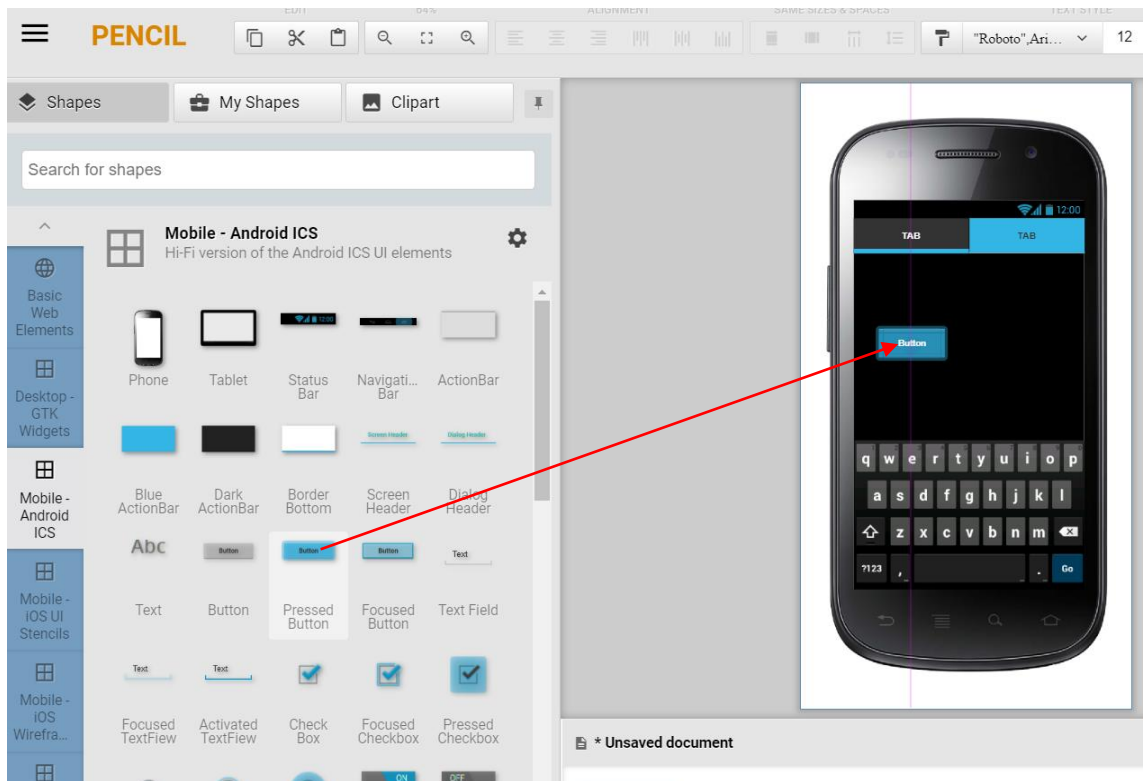


Рис. 11.6. Проектирование UI мобильного приложения

11.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

11.2.1. Практическое задание

В качестве практического задания необходимо в среде Evolus Pencil создать эскизы интерфейса пользователя для системы, создаваемой студентом в рамках индивидуального задания на дисциплину. Эскизы должны создаваться для той платформы, которая соответствует индивидуальному заданию:

- настольное приложение для ОС Windows;
- настольное приложение для ОС MacOS;
- настольное приложение для ОС Linux;
- мобильное приложение для ОС Android;
- мобильное приложение для ОС iOS;
- веб-приложение;
- другие (AR/VR, Smart Watch...).

Проектированию подлежат (при наличии):

- главное окно;
- главное меню;
- система навигации;
- все диалоговые окна.

11.2.2. Список контрольных вопросов для самопроверки

1. Какие основные этапы проектирования UI?
2. Какие ключевые отличия UI для настольных приложений и веб-приложений?
3. В чем отличие эскиза интерфейса от схемы интерфейса?
4. В чем преимущество специализированных программ для проектирования UI от универсальных векторных редакторов?

11.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.

3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

12. ЛАБОРАТОРНАЯ РАБОТА №12. Разработка клиент-серверной системы на основе ORM

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков разработки клиент-серверных систем на основе концепции ORM (Object-Relational Mapping).

Введение

Как правило разработка клиент-серверных приложений предполагает для разработчика различные компетенции для реализации клиентской и серверной частей. Это связано с тем, что серверная часть зачастую должна решать задачи хранения и обработки данных. Для этого применяют реляционные СУБД, требующие особых навыков и умений, например, знаний языка SQL. Клиентская часть, как правило, реализуется на языках высокого уровня с применением объектно-ориентированных принципов. Технология ORM позволяет разработчику применять объектно-ориентированные принципы для работы с базами данных, скрывая от разработчика детали реализации взаимодействия с СУБД. В лабораторной работе рассматривается реализация этой технологии от компании Microsoft — ADO.NET Entity Framework.

12.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

12.1.1. Технология ORM

ORM (англ. *Object-Relational Mapping*, рус. *объектно-реляционное отображение*, или *преобразование*) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Основная задача ORM — обеспечить работу с данными в терминах классов, а не таблиц данных и напротив, преобразовать термины и данные классов в данные, пригодные для хранения в СУБД. Обычно классу объекта соответствует таблица базы данных, а экземпляру этого класса запись в таблице. Связи между экземплярами объектами в реляционной базе данных реализуются с помощью ключей и/или дополнительных таблиц.

В общем случае необходимо обеспечить интерфейс работы с CRUD-операциями над данными, избавиться от необходимости писать SQL-код для взаимодействия с СУБД. Использование реляционной базы данных для хранения объектно-ориентированных данных приводит к семантическому разрыву, когда программное обеспечение должно уметь как обрабатывать данные в объектно-ориентированном виде, так и уметь сохранить эти данные в реляционной форме. Очевидно, что это существенно усложняет разработку и повышает требования к квалификации разработчика.

Ключевой особенностью ORM является отображение (mapping), которое используется для привязки объекта к его данным в БД. ORM как бы создает «виртуальную» схему базы данных в памяти и позволяет манипулировать данными уже на уровне объектов. Отображение показывает, как объект и его свойства связаны с одной или несколькими таблицами и их полями в базе данных. ORM использует информацию этого отображения для управления процессом преобразования данных между базой и формами объектов, а также для создания SQL-запросов для вставки, обновления и удаления данных в ответ на изменения, которые приложение вносит в эти объекты.

ORM позволяет программисту просто создавать объекты и работать с ними как обычно, а они автоматически будут сохраняться в реляционной базе данных. То есть ORM избавляет программиста от написания большого объема кода, часто однообразного и возможно подверженного ошибкам. Применение ORM значительно повышает скорость разработки. Однако часто это приводит к тому, что программы работают медленнее и используют больше памяти, чем программы, написанные «вручную». Для устранения этих проблем большинство современных реализаций ORM позволяют программисту при необходимости самому жёстко задать код SQL-запросов, который будет использоваться при тех или иных операциях.

12.1.2. Entity Framework

Entity Framework — это объектно-ориентированная технология доступа к данным, является ORM-решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Entity SQL представляет собой язык,

подобный языку SQL, который позволяет выполнять запросы к концептуальным моделям в Entity Framework.

Entity Framework поддерживает подходы Database First (База данных как основа) и Code First (Код как основа).

Подход Database First дает возможность реконструировать модель по существующей базе данных. Модель хранится в EDMX-файле (расширение .emdx), и её можно просмотреть и изменить в Entity Framework Design. Классы, с которыми программист взаимодействует в приложении, автоматически создаются из файла EDMX.

Подход Code First, наоборот, предполагает создание модели базы данных с по существующим классам, соответствующим сущностям предметной области.

Для выполнения этой лабораторной работы необходимы следующие среды:

- Microsoft Visual Studio 2019 (C#, Entity Framework)
- Microsoft SQL Server 2019 (Express или выше)

12.2. ПРАКТИЧЕСКАЯ ЧАСТЬ

12.2.1. Практическое задание

В качестве практического задания необходимо разработать клиентскую и серверную части для системы, создаваемой студентом в рамках индивидуального задания на дисциплину. Задание должно быть основано на результатах проектных решений, полученных при выполнении лабораторных работ №6 (технический проект), №8 и №9 (логической и физической моделях базы данных) и лабораторной работы №11 (интерфейса пользователя).

12.2.2. Список контрольных вопросов для самопроверки

1. В чем основное достоинство ORM?
2. Какие есть недостатки у большинства реализаций ORM?
3. Как можно снизить влияние недостатков концепции ORM?

12.3. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели и задачи выполняемой лабораторной работы.
3. Пошаговое описание выполняемых заданий лабораторной работы:
4. Ответы на контрольные вопросы.
5. Заключение.

Заключение

Лабораторный практикум по дисциплине «Проектирование информационных систем» соответствует рабочей программе дисциплины «Проектирование информационных систем» для студентов, обучающихся по направлению 09.04.03 «Прикладная информатика». Практикум позволит студентам получить базовые навыки по проектированию информационных систем, включая эскизное и техническое проектирование, описание решений с помощью диаграмм UML, проектирование моделей баз данных и интерфейсов пользователя.

Поднимаемые в практикуме задачи заставляют студентов применять полученные теоретические знания для проектирования информационных систем, что поможет им квалифицированно решать соответствующие прикладные задачи после окончания учёбы.

СПИСОК ЛИТЕРАТУРЫ

Основная литература:

1. Брукс, Фредерик П. Мифический человеко-месяц или как создаются программные системы : пер. с англ. / Ф. П. Брукс. — 2-е изд.. — Санкт-Петербург; Москва: Символ, 2012. — 298 с. Библиотечный фонд: читальный зал технической литературы; Инвентарный номер: 13-1619; Шифр хранения: 004.4 Б89
2. Макконнелл С. Совершенный код. Мастер-класс. — М.: Русская Редакция, 2014. — 896 с. ISBN 978-5-7502-0064-1 Библиотечный фонд: учебный фонд; Инвентарный номер: 1170125; Шифр хранения: 681.3 М158
3. Блюмин А., Печеная Л., Феоктистов Н. Проектирование систем информационного, консультационного и инновационного обслуживания. — М.: Дашков и Ко, 2011.
4. Емельянова Н., Партыка Т., Попов И. Проектирование информационных систем. — М.: Форум, 2012.

Дополнительная литература:

5. Соммервилл, Иан. Инженерия программного обеспечения : пер. с англ. / И. Соммервилл. — 6-е изд. — Москва: Вильямс, 2002. — 624 с.: ил. — ISBN 5-8459-0330-0 Библиотечный фонд: учебный фонд; Инвентарный номер: 1143311; Шифр хранения: 681.3 С614
6. Демарко, Том. Deadline. Роман об управлении проектами : пер. с англ. / Т. Демарко. — Москва: Манн, Иванов и Фербер, 2015. — 294 с. — Библиотека Сбербанка; Т. 31. — ISBN 978-5-00057-055-5. Библиотечный фонд: читальный зал гуманитарной литературы; Инвентарный номер: 17-195; Шифр хранения: У821 Д30
7. Вигерс, Карл И.. Разработка требований к программному обеспечению : пер. с англ. / К. И. Вигерс, Д. Битти. — 3-е изд., доп.. — Санкт-Петербург; Москва: БХВ-Петербург Русская Редакция, 2015. — 718 с.: ил.. — Словарь терминов: с. 707-716.. — ISBN 978-5-9775-3348-5. — ISBN 978-5-7502-0433-5. Библиотечный фонд: читальный зал технической литературы; Инвентарный номер: 15-1124; Шифр хранения: 004.4 В414
8. Гамма Э., Хелм Р., Джонсон Р., Влссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Пи-тер, 2014. — 368 с.: ил. ISBN 978-5-496-00389-6 Библиотечный фонд: научный фонд; Инвентарный номер: 06-4896

9. Коберн А. Современные методы описания функциональных требований к системам: Пер. с англ. — М: Лори, 2012. — 264 с. ISBN 978-5-85582-326-4
Библиотечный фонд: учебный фонд; Инвентарный номер: 1299367; Шифр хранения: У К552
10. ДеМарко Т., Листер Т. Человеческий фактор. Успешные проекты и команды. — М.: Символ-Плюс. — 2014. — 288 с. Библиотечный фонд: читальный зал гуманитарной литературы; Инвентарный номер: 13-1622; Шифр хранения: У821 Д30
11. Дейт К. Дж., Введение в системы баз данных. 8-е изд. / Пер. с англ. — К.: Изд. дом «Вильямс», 2005. — 1328 с.: ил.
12. Гвоздева Т., Баллод Б. Проектирование информационных систем.- М.: Феникс, 2009.
13. Пирогов В. Информационные системы и базы данных. Организация и проектирование.- Спб:БХВ-Петербург, 2009.
14. Рыбина Г. В. Основы построения систем.- М.: Финансы и статистика, Инфра-М, 2010.
15. Бабенко Л.К. и др. Защита данных информационных систем : учебное пособие для вузов. – М.: Гелиос АРВ, 2010.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 2

1. ЛАБОРАТОРНАЯ РАБОТА №1. ПОЛЬЗОВАТЕЛЬСКИЕ ИСТОРИИ. ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ 3

1.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 3

1.1.1. Пользовательские истории (User Stories) 3

1.1.2. Варианты использования (Use Cases) 4

1.2. ПРАКТИЧЕСКАЯ ЧАСТЬ 12

1.2.1. Практическое задание 12

1.2.2. Список контрольных вопросов для самопроверки 13

1.3. ТРЕБОВАНИЯ К ОТЧЕТУ 13

2. ЛАБОРАТОРНАЯ РАБОТА №2. ДИАГРАММЫ UML. ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ 14

2.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 14

2.1.1. Основные элементы диаграммы 14

2.1.2. Отношения между элементами 15

2.1.3. Инструментальные средства для создания диаграммы 18

2.2. ПРАКТИЧЕСКАЯ ЧАСТЬ 19

2.2.1. Практическое задание 19

2.2.2. Список контрольных вопросов для самопроверки 19

2.3. ТРЕБОВАНИЯ К ОТЧЕТУ 20

3. ЛАБОРАТОРНАЯ РАБОТА №3. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 21

3.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 21

3.1.1. Основные работы при разработке требований 21

3.1.2. Понятие требования 22

3.1.3. Идентификация требований 23

3.1.4. Понятие стейкхолдера 24

3.1.5. Государственные и международные стандарты для разработки ТЗ 24

3.1.6. Структура ТЗ по ГОСТ 19.201-78 25

3.1.7. Структура ТЗ по ГОСТ 34.602-89 25

3.2. ПРАКТИЧЕСКАЯ ЧАСТЬ 26

3.2.1. Практическое задание 26

3.2.2. Список контрольных вопросов для самопроверки 26

3.3. ТРЕБОВАНИЯ К ОТЧЕТУ	26
4. ЛАБОРАТОРНАЯ РАБОТА №4. ДИАГРАММЫ UML. ДИАГРАММА КОМПОНЕНТОВ	28
4.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	28
4.1.1. Базовые понятия	28
4.1.2. Компоненты и интерфейсы	29
4.1.3. Порты	30
4.1.4. Части компонента	31
4.1.5. Коннекторы	32
4.1.6. Рекомендации по построению диаграммы компонентов	33
4.1.7. Инструментальные средства для создания диаграммы компонентов	33
4.2. ПРАКТИЧЕСКАЯ ЧАСТЬ	34
4.2.1. Практическое задание	34
4.2.2. Список контрольных вопросов для самопроверки	34
4.3. ТРЕБОВАНИЯ К ОТЧЕТУ	35
5. ЛАБОРАТОРНАЯ РАБОТА №5. ДИАГРАММЫ UML. ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ	36
5.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	36
5.1.1. Объекты	36
5.1.2. Сообщения	38
5.1.3. Рекомендации по построению диаграммы последовательности	38
5.1.4. Инструментальные средства для создания диаграммы последовательности	38
5.2. ПРАКТИЧЕСКАЯ ЧАСТЬ	39
5.2.1. Практическое задание	39
5.2.2. Список контрольных вопросов для самопроверки	40
5.3. ТРЕБОВАНИЯ К ОТЧЕТУ	40
6. ЛАБОРАТОРНАЯ РАБОТА №6. ЭСКИЗНЫЙ ПРОЕКТ. ТЕХНИЧЕСКИЙ ПРОЕКТ	41
6.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	41
6.1.1. Эскизный проект	41
6.1.2. Технический проект	42

6.1.3.	Пояснительная записка к техническому проекту	43
6.1.4.	Государственные стандарты для разработки эскизного и технического проектов	45
6.2.	ПРАКТИЧЕСКАЯ ЧАСТЬ	45
6.2.1.	Практическое задание	45
6.2.2.	Список контрольных вопросов для самопроверки	46
6.3.	ТРЕБОВАНИЯ К ОТЧЕТУ	46
7.	ЛАБОРАТОРНАЯ РАБОТА №7. ПРОЕКТИРОВАНИЕ БД	47
7.1.	ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	47
7.1.1.	Понятие базы данных	47
7.1.2.	Этапы проектирования модели БД	48
7.1.3.	ER-модель	49
7.1.4.	ER-модель: нотации	51
7.1.5.	Инструменты проектирования БД	52
7.2.	ПРАКТИЧЕСКАЯ ЧАСТЬ	53
7.2.1.	Практическое задание	53
7.2.2.	Список контрольных вопросов для самопроверки	53
7.3.	ТРЕБОВАНИЯ К ОТЧЕТУ	53
8.	ЛАБОРАТОРНАЯ РАБОТА №8. ПРОЕКТИРОВАНИЕ БД. ЛОГИЧЕСКАЯ МОДЕЛЬ	54
8.1.	ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	54
8.1.1.	Toad Data Modeler	54
8.1.2.	Создание и настройка сущностей	55
8.1.3.	Создание и настройка связей	58
8.1.4.	Удаление элементов диаграммы	58
8.1.5.	Проверка модели	59
8.2.	ПРАКТИЧЕСКАЯ ЧАСТЬ	60
8.2.1.	Практическое задание	60
8.2.2.	Список контрольных вопросов для самопроверки	60
8.3.	ТРЕБОВАНИЯ К ОТЧЕТУ	60
9.	ЛАБОРАТОРНАЯ РАБОТА №9. ПРОЕКТИРОВАНИЕ БД. ФИЗИЧЕСКАЯ МОДЕЛЬ	61
9.1.	ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	61

9.1.1.	Подходы к созданию физической модели	61
9.1.2.	Создание физической модели из логической	61
9.1.3.	Коррекция физической модели	63
9.1.4.	Проверка модели	65
9.1.5.	Генерация скрипта создания базы данных	65
9.2.	ПРАКТИЧЕСКАЯ ЧАСТЬ	67
9.2.1.	Практическое задание	67
9.2.2.	Список контрольных вопросов для самопроверки	67
9.3.	ТРЕБОВАНИЯ К ОТЧЕТУ	68
10.	ЛАБОРАТОРНАЯ РАБОТА №10. ИНСТРУМЕНТЫ ПРОЕКТИРОВАНИЯ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ	69
10.1.	ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	69
10.1.1.	Общие сведения о проектировании UI	69
10.1.2.	Целевые платформы UI	71
10.2.	ПРАКТИЧЕСКАЯ ЧАСТЬ	72
10.2.1.	Практическое задание	72
10.2.2.	Список контрольных вопросов для самопроверки	73
10.2.3.	ТРЕБОВАНИЯ К ОТЧЕТУ	73
11.	ЛАБОРАТОРНАЯ РАБОТА №11. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ	74
11.1.	ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	74
11.1.1.	Проектирование UI в среде Evolus Pencil	74
11.2.	ПРАКТИЧЕСКАЯ ЧАСТЬ	79
11.2.1.	Практическое задание	79
11.2.2.	Список контрольных вопросов для самопроверки	79
11.3.	ТРЕБОВАНИЯ К ОТЧЕТУ	79
12.	ЛАБОРАТОРНАЯ РАБОТА №12. РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОЙ СИСТЕМЫ НА ОСНОВЕ ORM	81
12.1.	ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	81
12.1.1.	Технология ORM	81
12.1.2.	Entity Framework	82
12.2.	ПРАКТИЧЕСКАЯ ЧАСТЬ	83
12.2.1.	Практическое задание	83

12.2.2. Список контрольных вопросов для самопроверки	83
12.3. ТРЕБОВАНИЯ К ОТЧЕТУ	84
ЗАКЛЮЧЕНИЕ	85
СПИСОК ЛИТЕРАТУРЫ	86