МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

ЧЕРНИГОВСКИЙ НАЦИОНАЛЬНЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

КАФЕДРА ИНФОРМАЦИОННЫХ И КОМПЬЮТЕРНЫХ СИСТЕМ

**БАЗА ДАННЫХ «ПИЛОРАМА»**

Курсовая работа по дисциплине   
«Организация баз данных»

|  |  |  |
| --- | --- | --- |
| Исполнитель  студент гр. КИ-122 |  | П.В. Садовой  С.П. Алябышев  Г.Н. Тур  М.М. Марченок |
| Руководитель  ассистент |  | М.В. Харченко |

**2015**

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

на выполнение курсового проекта

по дисциплине «Организация баз данных»

**Садовой П.В., Алябышев С.П.,** **Тур Г.Н.,** **Марченок М.М. гр. КИ-122**

Тема работы: База данных “Пилорама”

**Предполагаемые технические и эксплуатационные результаты работы:**

Программная система управления заданиями предприятия “Лесопилки“ представляет собой корпоративное приложение, предназначенное для автоматизации процессов выдачи и контроля заданий работникам предприятия, сокращения времени на выдачу задания конкретному исполнителю и упрощения контроля выполнения задания путем отправки почтовых сообщений.

Программная система предоставляет возможность управлять списками сотрудников поставок и изделий, а также древесины.

Данная работа представляет собой разработку полноценной БД, которая размещается на сервере PostgreSQL и используется при разработке информационно-компьютерной системы в архитектуре клиент-сервер.

Разработка БД должна включать:

1. построение концептуальной модели предметной области, содержащей не менее 4-х объектов, связанных между собой взаимосвязями типа 1:М и M:N, в виде диаграммы объектов языка UML и ее описание;
2. выполнение нормализации схемы БД (не менее 5-ти таблиц) с указанием:
   1. первичных и внешних ключей;
   2. типов данных атрибутов;
   3. индексов;
3. реализацию на уровне структуры БД средств обеспечения целостности данных:
   1. уникальность и обязательность ввода первичных ключей;
   2. поддержка целостности для внешних ключей (каскадное удаление и обновление);
   3. значения атрибутов по умолчанию и обязательность ввода значений атрибутов;
4. на уровне бизнес-логики обеспечение ограничений на значения данных атрибутов вида: "интервал", "перечислимое значение" и "сравнение значений двух атрибутов одной таблицы";
5. реализацию запросов к БД на поиск, обновление и удаление данных;
6. реализацию не мене 2-х хранимых процедур;
7. реализацию не мене 3-х триггеров для Insert, Update и Delete;
8. формирование не менее 2-х вариантов представлений;
9. реализацию назначение, проверку и отмену прав доступа.

Реализация пользовательского интерфейса предполагает обязательное наличие форм для просмотра, добавления, редактирования и удаления записей всех таблиц.

**Объем текстовой и графической документации:**

Пояснительная записка к проекту объемом 15-20 страниц печатного текста формата А4 и программная документация на систему объемом 35-40 страниц печатного текста формата А4. Объемы текстовой информации могут быть скорректированы в процессе работы по согласованию с руководителем.

**Предполагаемая трудоёмкость работы:** 45-50 человеко-часов.

**Плановые сроки по этапам:**

1. Анализ предметной области. Разработка схемы БД 22.02.15
2. Прототип пользовательского интерфейса 07.03.15
3. Классы домена (диаграмма классов + реализация на Java) 22.03.15
4. Классы для работы с БД. (диаграмма классов + реализация на Java) 12.04.15
5. WEB-интерфейс (HTML+CSS+JSP) 26.04.15
6. ANT скрипт сборки проекта. Демонстрация проекта 10.05.15
7. Оформление работы 24.05.15

Демонстрация работоспособности программного проекта до 13.05.15

**Плановый срок защиты проекта: 26.05.15**

Исполнители работы:

Садовой П.В., Алябышев С.П., Тур Г.Н., Марченок М.М.

Руководитель работы:

Ассистент Харченко М.В.

Дата выдачи задания

\_\_ февраля 20\_\_ г.

Реферат

Курсовая работа, \_\_ с., \_\_ рис., \_\_ источников, \_ приложения.

Цель разработки курсовой работы - реализовать приложение, которое позволит работать, с БД, посредством тонкого клиента.

В процессе написания приложения были разработаны и созданы классы и фабрики для работы с сущностями. С помощью ServiceFactory была дополнительно реализована бизнес-логика.

Также была использована технология Servlet- и JSP-контейнера.

В качестве сервера был использован сервер Tomcat v8.0. Приложение было разработано с использованием комплекта JDK версии 1.8.

В ходе разработки было получено корпоративное приложение «Лесопилка» доведенное до уровня стабильной версии. Результат разработки оформлен в виде программного проекта, приводимого в приложении к курсовой работе.

Дальнейшее развитие работы возможно в сторону усовершенствования графического интерфейса пользователя и дополнения функциональных возможнотей.

Java, ORM, JSP, JPA, SQL, Servlet, HTML, TAG

Реферат

Курсова робота, \_\_ с., \_\_ Рис., \_\_ Джерел, \_ програми.

Мета розробки курсової роботи - реалізувати додаток, що дозволить працювати, з БД, за допомогою тонкого клієнта.

У процесі написання програми були розроблені і створені класи та фабрики для роботи з сутностями. За допомогою ServiceFactory була додатково реалізована бізнес-логіка.

Також була використана технологія Servlet- і JSP-контейнера.

Як сервер був використаний сервер Tomcat v8.0. Додаток було розроблено з використанням комплекту JDK версії 1.8.

У ході розробки було отримано корпоративне додаток «Тартак» доведене до рівня стабільної версії. Результат розробки оформлений у вигляді програмного проекту, що приводиться в додатку до курсової роботи.

Подальший розвиток роботи можливе у бік удосконалення графічного інтерфейсу користувача і доповнення функціональних возможноть.

JAVA, ORM, JSP, JPA, SQL, SERVLET, HTML, TAG

THE ABSTRACT

Course work, \_\_ p., \_\_ fig., \_\_ Sources \_ application.

The aim of the course work - to implement an application that allows to work with the database through a thin client.

In the process of writing applications have been designed and created classes and factories to work with the entities. With ServiceFactory it was further implement the business logic.

Also, technology was used Servlet- and JSP-container.

As a server, the server was used Tomcat v8.0. The application was developed using a JDK version 1.8.

During the development of enterprise application was received "Saw" brought to the level of the stable version. The result of the development is designed as a software project in the annex to the course work.

Further development work is possible in the direction of improvement of the graphical user interface and additional functionality vozmozhnot.

JAVA, ORM, JSP, JPA, SQL, SERVLET, HTML, TAG

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 8](#_Toc410595277)

[1 АНАЛИЗ РЕШАЕМОЙ ЗАДАЧИ 9](#_Toc410595278)

[1.1 Анализ предметной области 9](#_Toc410595279)

[1.2 Цели и задачи системы 10](#_Toc410595280)

[1.3 Назначение системы 10](#_Toc410595281)

[1.4 Требования к системе 11](#_Toc410595282)

[2 Проектирование системы 12](#_Toc410595283)

[2.1 Выбор инструментальных средств разработки системы 12](#_Toc410595284)

[2.1.1 Сервер баз данных 12](#_Toc410595285)

[2.1.2 Технологии реализации системы 12](#_Toc410595286)

[2.2 Проектирование архитектуры системы 12](#_Toc410595287)

[2.2.1 Проектирование слоя бизнес-логики и бизнес-правил 12](#_Toc410595288)

[2.2.2 Проектирование слоя доступа к данным 13](#_Toc410595289)

[2.2.3 Проектирование слоя отображения 13](#_Toc410595290)

[3 Разработка 14](#_Toc410595291)

[3.1 Разработка базы данных системы 14](#_Toc410595292)

[3.1.1 Разработка схемы базы данных 14](#_Toc410595293)

[3.1.2 Обеспечение целостности данных 15](#_Toc410595294)

[3.1.3 Разработка базовых запросов 16](#_Toc410595295)

[3.1.4 Выбор индексов 21](#_Toc410595296)

[3.1.5 Разработка хранимых процедур и триггеров 22](#_Toc410595297)

[3.1.6 Разработка представлений 23](#_Toc410595298)

[3.1.7 Организация защиты данных 24](#_Toc410595299)

[3.1.8 Объектно-реляционное отображение 24](#_Toc410595300)

[3.2 Разработка модулей системы 25](#_Toc410595301)

[3.2.1 Разработка модулей слоя бизнес логики и бизнес правил 25](#_Toc410595302)

[3.2.2 Разработка модулей слоя доступа к данным 25](#_Toc410595303)

[3.2.3 Разработка модулей слоя отображения 26](#_Toc410595304)

[ВЫВОДЫ 27](#_Toc410595305)

[ПЕРЕЧЕНЬ ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc410595306)

[ПРИЛОЖЕНИЯ 29](#_Toc410595307)

ВВЕДЕНИЕ

Проектирование базы данных (БД) - одна из наиболее сложных и ответственных задач, связанных с созданием корпоративного приложения (enterprise application). В результате её решения должны быть, определены содержание БД, эффективный для всех её будущих пользователей способ организации данных и инструментальные средства управления данными. Основная цель проектирювания БД - это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте.

Корпоративное приложение представляет собой программное приложение, предназначенное для обработки данных большого объема по бизнес правилам позволяющее принести определенные преимущества корпорации (предприятию) при ее внедрении.

Корпоративное приложение обычно подразумевает необходимость долговременного (иногда в течение десятилетий) хранения данных. Данные зачастую способны пережить несколько поколений прикладных программ, предназначенных для их обработки, аппаратных средств, операционных систем и компиляторов. Множество пользователей обращаются к данным параллельно. Как правило, их количество не превышает сотни, но для систем, размещенных в среде Web, этот показатель возрастает на несколько порядков. Корпоративные приложения редко существуют в изоляции. Обычно они требуют интеграции с другими системами, построенными в разное время с применением различных технологий.

Примером корпоративного приложения может быть разработанная программа «Пилорама», предназначенная для просмотра сведений о лесозагатовке, лесниках и общей работы лесопилки.

**О актуальности данной программной системы можно говорит очень много.** Во-первых, это удобство в проведении учебного процесса. Во-вторых, это позволяет быстро получать и модифицировать данные. Также в связи с этим упрощается организация учебного процесса.

# АНАЛИЗ РЕШАЕМОЙ ЗАДАЧИ

В данном разделе проводится анализ решения поставленой задачи. Определяется предметная область, которую охватывает разрабатываемая система, ее назначения, цели и задача, которые она покрывает.

## Анализ предметной области

Деревообрабатывающая промышленность является очень важной частью народного хозяйства Украины, предприятия занимающиеся заготовкой леса производят около 7% общего ВВП этой страны.

Типичное деревообрабатывающее предприятие закупает древесину у лесозаготовительных лесных хозяйств. Лесозаготовительные хозяйства зачастую поставляют древесину большими партиями, по несколько различных видов древесины за одну поставку.

Древесина бывает различных разнообразных типов в зависимости от различных древесных растений из которых она была получена. Основными классами древесины является хвойная, лиственная, а также экзотическая древесина, получаемая из деревьев растущих в Африке, Австралии и Южной Америке. Основное отличие хвойной от лиственной древесины это устойчивость к гниению, хвойная древесина зачастую менее устойчива к гниению, поэтому ее рекомендуется хранить отдельно от лиственной.

После получения древесина хранится на складе до начала ее обработки. Древесина обрабатывается в несколько этапов – сперва древесину вымачивают, а также обрабатывают антисептиком, что позволяет существенно увеличить ее прочность, а также увеличить сопротивляемость грибкам, насекомым и влаге. Следующим этапом является распиливание дерева и получение пиломатериалов. После распиливания дерева получают разнообразные пиломатериалы, такие как брус, брусок, доска, шпалы, обаполь и прочие. Также из отходов производства производят такую продукцию как шпон, фанеру, древесно-слоистый пластик (ДСП), древесно волокнистую плиту (ДВП) и прочие композитные материалы.

Следующим этапом деревообработки является хранение полученных пиломатериалов на складе. При правильном хранении древесина может хранится очень долго, для хранения и атмосферной сушки пиломатериалов существует ГОСТ 3808.1-80.

Готовые пиломатериалы очень широко применяются в народном хозяйстве – от создания мебели и построения жилья – до судостроения и авиастроения. Пиломатериалы зачастую покупают крупными партиями, в которые входят различные изделия.

В данной системе принимают участие такие объекты: изделие, лесничий, работник пилорамы, поставка. С данной информационной системой должны участвовать следующие объекты: изделие и древесина.

В системе ведутся учетные записи, поэтому по прохождению некоторого интервала времени создается рейтинг лесников по количеству их поставок. Это дает возможность проследить активность лесников и принять определенные меры.

## Цели и задачи системы

Целью системы управления пилорамой является автоматизация работы этой системы, а именно выдача и контроль заданий, учет лесников, закупаемой древесины и изделий. С помощью автоматизации регулируется вся работа заведения, составляется отчет о активности лесников, о выработке преподавателей, также составляется расписание работы всех участников данной системы. Благодаря ей упрощаеться как администрирование данным сервисом, так и взаимодействие с ним с клиентской стороны благодаря интуитивно понятному интерфейсу.

## Назначение системы

Данной системой будут пользоваться лесники, руководство и рабочие на пилораме. Она позволяет сделать улучшить процесс работы пилорамы, так как упрощается организация закупки древесины и производства изделий.

Система предназначена для автоматизации организации учебного процесса, а также для удобного взаимодействия преподавателей и студентов.

Лесникам предоставлены следующие возможности как:

* Ознакомится с закупаемой древесиной;
* Продать древесину;
* Зарегистрироватся в системе;
* Посмотреть прошлые поставки;

Предполагается, что с данным корпоративным приложением будут

работать сотрудники пилорамы, и они должны иметь возможность решать с его помощью следующие задачи:

* Сделать изделие;
* Посмотреть материалы необходимые для создания изделия;
* Посмотреть наявные изделия;
* Посмотреть наявную древесину;

Также в системе будет администратор, в обязанности которого входит следующее:

* Добавлять производимое изделие;
* Регистрировать пользователей;
* Редактировать информацию о пользователях;
* Добавлять закупаемую древесину;

На рисунке 1. 1 приведена диаграмма вариантов использования.

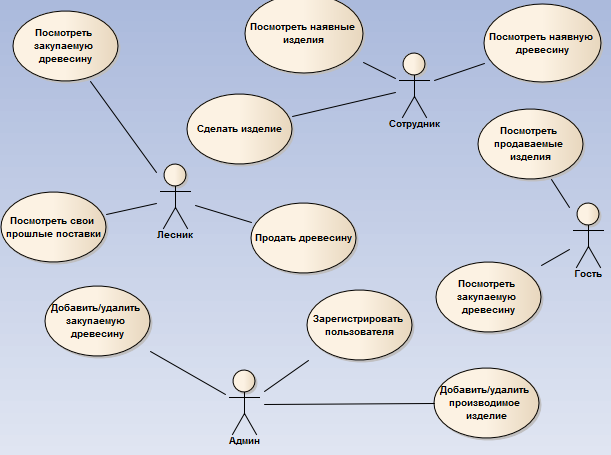


Рисунок 1.1 – Диаграмма вариантов использования

## Требования к системе

Разрабатываемая в рамках данной курсовой работы система «Пилорама» должна оперировать следующими объектами: лесничий, рабочий пилорамы, изделие, поставка, древесина.

Нужно разработать такую систему в которой бы и лесничему и было удобно выполнять свои функции. А именно должно быть удобно просматривать список закупаемых типов древесины, список поставок леснику, а также просматривать и изменять информацию о наявной древесине и производимых изделиях работникам пилорамы.

Система должна: предоставлять возможность руководству пилорамы управлять закупкой древесины и производством новых изделий. Они имеют право изменять, добавлять и удалять информацию в этих таблицах. Также руководство добавляет новых лесников и сотрудников в базу данных.

Лесники и простые рабочие не допускаються к администрированию информации. Лесники могут зарегистрироваться и просматривать необходимую им информацию: список закупаемой древесины и список их прошлых поставок, а также совершать новые поставки. Рабочие пилорамы могут изменять наявное количество древесины на складе, а также изменять количество наявных изделий определенного типа. Также система автоматически учитывает, что лесник, который продал 10 кубометров леса продает 11 кубометр бесплатно.

На объекты и правила взаимодействия между объектами в системе накладываются определенные ограничения, совокупность которых называется бизнес-логикой.

# Проектирование системы

Проектирование это последовательное продумывание и описание того, какой должна быть программа. Проектирование – это процесс разработки [проекта](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82), то есть комплекта документации, предназначенной для создания определённой программы, ее эксплуатации, а также для проверки или воспроизведения промежуточных и конечных решений, на основе которых была разработана данная программа.

## Выбор инструментальных средств разработки системы

В данном пункте будет выбран сервер баз данных, и посредством чего будет происходить связь пользователя с БД, так же будет выбрана технология реализации системы и архитектура.

### Сервер баз данных

При разработке приложения, которое требует осуществления доступа к данным, необходимо упростить разработку такого приложения, увелачив эффективность и скорость работы с получаемыми данными. Поэтому данная проблема остаётся актуальной и сегодняшнее время.

На данный момент существует огромное количество серверов баз данных таких как:: MySQL, PostgreSQL, Microsoft Access и другие.

MySQL это реляционная база данных, работа с данными в которой осуществляется при помощи SQL запросов. Основными преимуществами этого типа БД является скорость и простота в использовании. При помощи MySQL можно производить операции над данными, которые с текстовыми файлами трудно реализуемы. Данный тип баз данных широко используется в порталах, досках объявлений, электронных магазинах. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа [MyISAM](http://ru.wikipedia.org/wiki/MyISAM), поддерживающие полнотекстовый поиск, так и таблицы [InnoDB](http://ru.wikipedia.org/wiki/InnoDB), поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц.

PostgreSQL – [свободная](http://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E) [объектно-реляционная](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) [система управления базами данных](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) (СУБД).

Функции являются блоками кода, исполняемыми на сервере, а не на клиенте БД. Хотя они могут писаться на чистом SQL, реализация дополнительной логики, например, [условных переходов](http://ru.wikipedia.org/wiki/%D0%A3%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4) и [циклов](http://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), выходит за рамки собственно SQL и требует использования некоторых языковых расширений. Функции могут писаться с использованием одного из следующих языков:

* встроенный процедурный язык [PL/pgSQL](http://ru.wikipedia.org/wiki/PL/pgSQL), во многом аналогичный языку [PL/SQL](http://ru.wikipedia.org/wiki/PL/SQL), используемому в СУБД Oracle;
* скриптовые языки — [PL/Lua](http://ru.wikipedia.org/w/index.php?title=PL/Lua&action=edit&redlink=1), [PL/LOLCODE](http://ru.wikipedia.org/w/index.php?title=PL/LOLCODE&action=edit&redlink=1), [PL/Perl](http://ru.wikipedia.org/wiki/PL/Perl), [PL/PHP](http://ru.wikipedia.org/w/index.php?title=PL/PHP&action=edit&redlink=1), [PL/Python](http://ru.wikipedia.org/wiki/PL/Python), [PL/Ruby](http://ru.wikipedia.org/w/index.php?title=PL/Ruby&action=edit&redlink=1), [PL/sh](http://ru.wikipedia.org/w/index.php?title=PL/sh&action=edit&redlink=1), [PL/Tcl](http://ru.wikipedia.org/wiki/PL/Tcl) и [PL/Scheme](http://ru.wikipedia.org/w/index.php?title=PL/Scheme&action=edit&redlink=1);
* классические языки — [C](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [C++](http://ru.wikipedia.org/wiki/C%2B%2B), [Java](http://ru.wikipedia.org/wiki/Java) (через модуль [PL/Java](http://ru.wikipedia.org/w/index.php?title=PL/Java&action=edit&redlink=1));
* статистический язык [R](http://ru.wikipedia.org/wiki/R_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) (через модуль [PL/R](http://ru.wikipedia.org/w/index.php?title=PL/R&action=edit&redlink=1)).

PostgreSQL допускает использование функций, возвращающих набор записей, который далее можно использовать так же, как и результат выполнения обычного запроса.

Функции могут выполняться как с правами их создателя, так и с правами текущего пользователя.

Иногда функции отождествляются с [хранимыми процедурами](http://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BC%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0), однако между этими понятиями есть различие. С девятой версии возможно написание автономных блоков, которые позволяют выполнять код на процедурных языках без написания функций, непосредственно в клиенте.

Для разработки курсовой работы была использована СУБД PostgreSQL, так как она довольно удобная, проста в работе и была ранее нами изучена.

### Технологии реализации системы

Для реализации системы пользователя (в частности тонкий клиент) используется следующие технологии.

J2EE (Java2 Platform, Enterprise Edition) – набор спецификаций и соответствующей документации для языка [Java](http://ru.wikipedia.org/wiki/Java), описывающей архитектуру серверной платформы для задач средних и крупных предприятий.

Спецификации детализированы настолько, чтобы обеспечить переносимость программ с одной реализации платформы на другую. Основная цель спецификаций — обеспечить [масштабируемость](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C) приложений и целостность данных во время работы системы. J2EE во многом ориентирована на использование её через [веб](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1) как в [интернете](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), так и в [локальных сетях](http://ru.wikipedia.org/wiki/%D0%9B%D0%92%D0%A1). Вся [спецификация](http://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) создаётся и утверждается через [JCP](http://ru.wikipedia.org/wiki/JCP) (Java Community Process).

Java API for XML Web Services (JAX-WS) — это [прикладной программный интерфейс](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) языка [Java](http://ru.wikipedia.org/wiki/Java) для создания [веб-служб](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0), являющийся частью платформы [Java EE](http://ru.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition). JAX-WS является заменой технологии [JAX-RPC](http://ru.wikipedia.org/w/index.php?title=JAX-RPC&action=edit&redlink=1), предоставляя более документо-ориентированную модель сообщений и упрощая разработок веб-служб за счёт использования [аннотаций](http://ru.wikipedia.org/w/index.php?title=%D0%90%D0%BD%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D1%8F_Java&action=edit&redlink=1), впервые появившихся в Java SE 5. Технология JAX-WS является стандартом и описана в [JSR](http://ru.wikipedia.org/wiki/Java_Community_Process) 224.

Сервлеты – это особым образом написанные (согласно спецификации) Java-программы, которые выполняются удаленно на сервере и вызов которых осуществляется удаленно из web-броузера с помощью HTTP протокола через web-сервер.

В технологиях Servlet и JSP введено понятие "контейнера".

Servlet-контейнер – это движок, отвечающий за выполнение Servlet-ов. JSP-контейнер – это движок, отвечающий за трансляцию jsp-страниц в сервлеты и передачу этих сервлетов Servlet-контейнеру. Так как сервлеты и jsp-страницы вызываются через HTTP-протокол, то Servlet-контейнер и JSP-контейнер часто сопровождает еще один компонент – web-сервер, который тоже может быть написан на Java.

Как видно на рисунке 2.1, веб-сервер, написанный на Java, получает запросы от браузера на выполнение того или иного сервлета или jsp-страницы на сервере, передает запрос в контейнер, который выполняет тот или иной сервлет. Результаты выполнения возвращаются контейнером веб-серверу, который в свою очередь пересылает его браузеру.



Рисунок . – Java веб – сервер и контейнеры

Для того, чтобы данное приложение работало, нужно, чтобы на сервере было установлено приложение Tomcat и JVM.

Объектно-реляционное отображение (ORM) – это техника программирования, которая связывает реляционную базу данных с концепциями объектно-ориентированного программирования и создает «виртуальную базу данных объектов».

При разработке приложения, которое требует осуществления доступа к данным, необходимо упростить разработку такого приложения, увеличив эффективность и скорость работы с получаемыми данными. Поэтому данная проблема остаётся актуальной и сегодняшнее время.

На приведенной ниже диаграмме показана взаимосвязь между основными компонентами JPA архитектуры (рисунок 2.2).

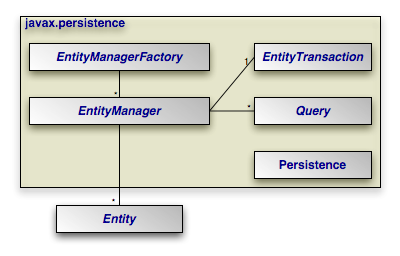


Рисунок . – Архитектура JPA

Persistence – класс содержит вспомогательные статические методы для получения EntityManagerFactory независимым от поставщика способом; EntityManagerFactory – интерфейс, реализация которого является фабрикой для создания объектов EntityManager. EntityManager – является основным JPA интерфейсом используемый в приложениях. Каждый EntityManager управляет набором хранимых объектов и содержит API для вставки новых объектов и удаления существующих. С каждым EntityManager связан свой EntityTransaction и, также, EntityManager выступает фабрикой для объектов Query. Entity – сущность, которая является хранимым объектом. EntityTransaction – объект, который производит управления транзакциями при выполнении операций с хранимыми объектами Entity. Операции группируются и либо выполняются полностью, либо нет, оставляя хранилище данных в неизменном состоянии. Query – интерфейс для выполнения запросов по нахождению хранимых объектов, которые удовлетворяют заданным критериям. JPA поддерживает запросы на объектном языке Java. Получить экземпляры Query можно из объекта EntityManager.

## Проектирование архитектуры системы

Для решения данной задачи идеально подходит многослойная архитектура корпоративного приложения (рисунок 2.3).

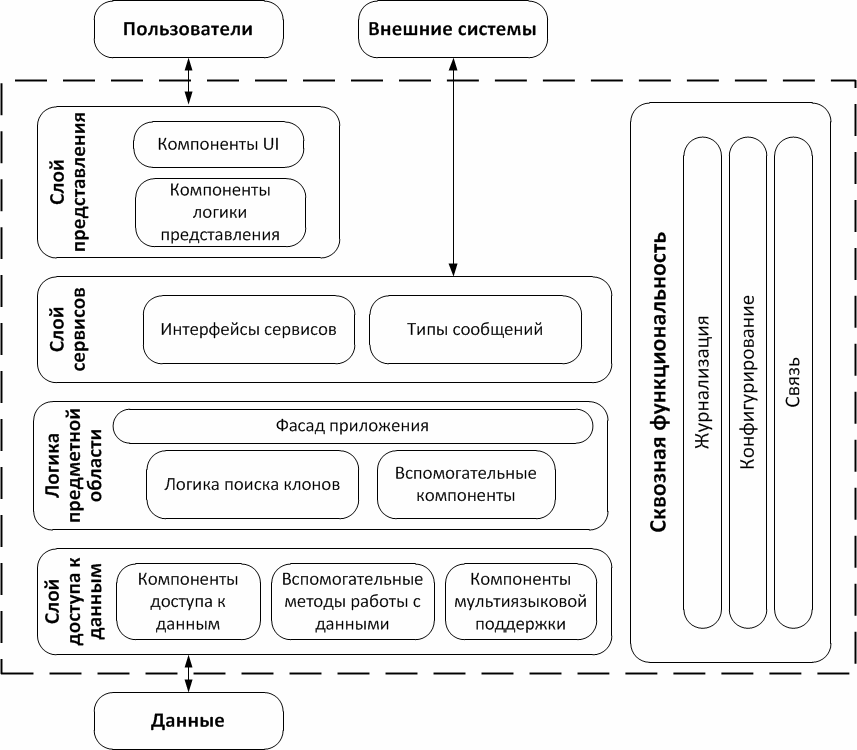


Рисунок . – Многослойная архитектура корпоративного приложения

Роль слоя интеграции (слоя источника данных) состоит в том, чтобы обеспечить возможность взаимодействия приложения с различными компонентами инфраструктуры для выполнения необходимых функций. Главная составляющая подобной проблемы связана с поддержісой диалога с базой данных — в большинстве случаев реляционной. Одной из самых серьезных причин успеха реляционных систем является поддержка ими SQL — наиболее стандартизованного языка коммуникаций с базой данных.

Способ реализации слоя интеграции зависит от взаимодействия бизнес логики с базой данных. Выбор, который делается на этом этапе, имеет далеко идущие последствия и отменить его бывает трудно или даже невозможно.

Поэтому он заслуживает наиболее тщательного осмысления. Нередко подобными решениями как раз и обусловливаются варианты компоновки бизнес - логики.

Разумнее обособить код SQL от бизнес логики, разместив его в специальных классах. Удачный способ организации подобных классов состоит в "копировании" структуры каждой Объекты базы данных в отдельном классе, который формирует шлюз, поддерживающий возможности обращения к таблице. Теперь основному коду приложения нет необходимости что-либо "знать" о SQL, а все SQL-операции сосредоточиваются в компактной группе классов. Более удачный вариант состоит в том, чтобы изолировать модель предметной области от базы данных, возложив на промежуточный слой всю полноту ответственности за отображение объектов домена в Объекты базы данных. Подобный преобразователь данных обслуживает все операции загрузки и сохранения информации, инициируемые бизнес-логикой, и позволяет независимо варьировать как модель предметной области, так и схему базы данных. Это наиболее сложное из архитектурных решений, обеспечивающих соответствие между объектами приложения и реляционными структурами, но его неоспоримое преимущество заключается в полном обособлении двух слоев.

На сегодняшний день Java разработчики могут воспользоваться уже имеющимися средствами: сериализация, средства объектно - реляционного отображения, объектные базы данных.

Сериализация. Встроенный Java-механизм, обеспечивающий преобразование объектов в последовательность байт, которые могут быть сохранены в файл или переданы по сети. Сериализация очень проста в использовании, е:о и весьма ограниченна. Пэи использовании сериализации объект сохраняется как одно целое. Она не поддерживает транзакции, а так же использование одного и того же сериализованного объекта в различных потоках или программах без возникновения конфликтов между ними;

**Объектно-реляционное отображение(JРА).** JPA не является новой технологией, а, скорее, это собрание идей лучших из имеющихся технологий, таких как Hibernate, TopLink и JDO. Как результат JPA является стандартизованной спецификацией входящей в J2EE5, что позволяет строить слой сохранения данных независимо от каких-либо конкретных провайдеров. Т.е. реализаций спецификации JPA может быть много, одной из таких, например, является фреймворк OpenJPA или тот же Hibernate.

Объектные базы данных.Объектные базы данных были специально разработаны для хранения объектов и полностью вписываются в концепцию объектно-ориентированного программирования. Object Database Management Group (ODMG) была создана для разработки единого АРІ для работы с такими базами. Однако, многие поставщики баз данных все еще не решаются перейти с хорошо себя зарекомендовавшей реляционной системы на объектно-ориентированную. Так же меньшее число средств анализа данных доступно для объектных баз и очень большое количество данных уже сохранено в реляционных базах. По этим причинам, а так же по множеству других, объектные базы данных не нашли такого широкого применения, на которое надеялись их создатели;

Enterprise Java Beams (EJBs). EJB представляют собой компоненты, которые; хранят свое состояние в реляционной базе данных и обеспечивают объектно-ориентированное отображение постоянных данных. В отличке от продуктов объектно-реляционного отображения, EJB имеют жесткую спецификацию, что делает возможным использование продуктов от различных поставщиков. К сожалению, EJB стандарт ограничен в объектно-ориентированном отношении. Они не поддерживают наследование, полиморфизм и т.п. Так же EJB компоненты требуют больших затрат для их написания и часто специального программного обеспечения для их работы.

На сегодняшний день существуют различные фреймворки, использующие данную технику программирования. Вот некоторые из них: Hibernate, iBATIS, Java Data Objects (JDO), JPOX, Cayenne, TopLink, JPA.

При организации ORM с использованием различных технологий необходимо создать файлы отображения объектов; создать конфигурационные файлы, в которых заказываются файлы ресурсов, источники данных, поддержка транзакций и т.д.

### Проектирование слоя бизнес-логики и бизнес-правил

Прежде всего, спроектируем объекты, которые будут отображаться в БД и заложим в них дополнительно необходимые поля, для реализации бизнес логики.

**Определение объектов системы**

Программная система должна реализовать выдачу и контроль заданий, учет лесников, закупку древесины и изделий. Исходя из вышесказанного, программная система оперирует такими объектами: лесники; древесина; изделия; поставки.

Диаграмма классов предметнойобласти представлена на рисунке 2.4.

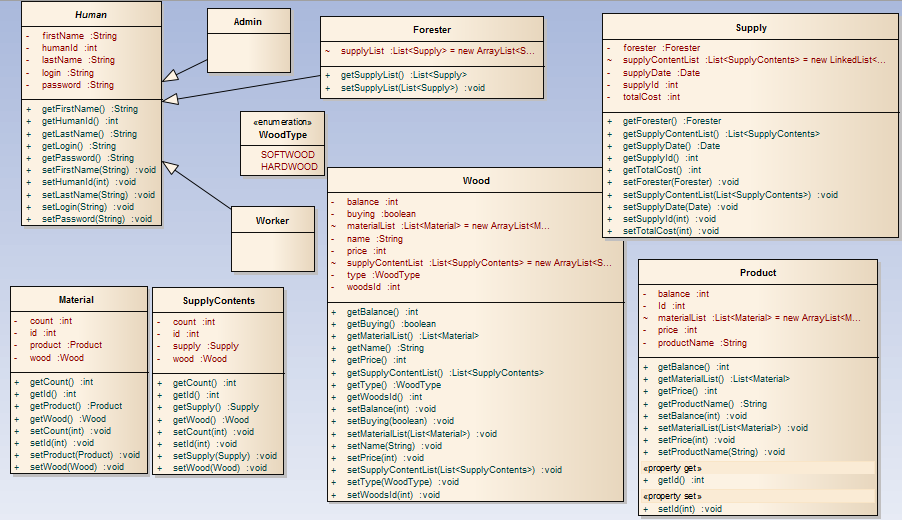


Рисунок 2.4 – Диаграмма классов предметной области

**Проектирование бизнес правил**

Бизнес правила заключаются в выполнении неких действий при наступлении определенных условий.

1. не позволять добавлять изделия стоимость которых меньше их себестоимости;
2. не позволять в рамках одной поставки поставлять хвойную и лиственную породу древесины;
3. каждый десятый кубометр леса лесник продает бесплатно.

Исходя из указанных выше требований можно построить классы бизнес логики. Диаграмма классов представлена на рисунке 2.5

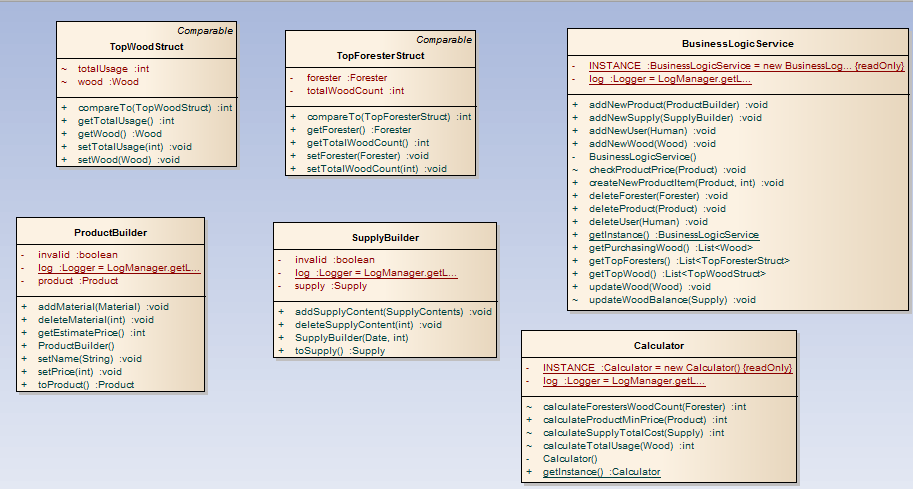


Рисунок 2.5 – Диаграмма классов бизнес логики

### Проектирование слоя доступа к данным

Для доступа к данным, хранящимся во внешнем хранилище, удобнее всего определить отдельные интерфейсы с методами для манипуляции данными. Реализация этих интерфейсов может быть любая, например, использующая JDBC или JPA, JAXB или даже простые Java-коллекции. В качестве упрощения данного курсового проекта был выбран JPA.

Проектирование интерфейсов DAO

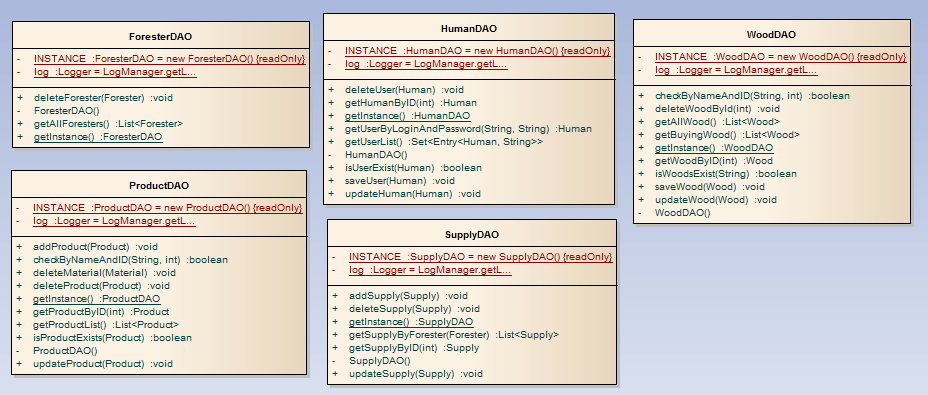
Среди всех операций доступа к данным можно отчетливо выделить базовые CRUD (create, read, update, delete) опреация - создание объекта, удаление объекта, обновление объекта, получение объекта по идентификатору, и получение всех объектов.

Рисунок 2.? – Диаграмма классов DAO

### Проектирование слоя отображения

Данный слой представляет собой тонкий клиент.

Для работы с приложением были созданы несколько страниц, которые обеспечивают вывод, добавление, редактирование и удаление данных. При разработке пользовательского интерфейса для максимального удобства была разработана страница с использованием web-интерфейса. Информация о древе, преподавателях, оценках, предметах, расписаниях и о группах отображается наглядно, в отдельных блоках. Данные для добавления, изменения и удаления из базы данных осуществляется через поля ввода. Видимость информации зависит от того, какой пользователь сейчас работает с приложением, то есть он может видеть и работать только со своей информацией. Незарегистрированные пользователи не могут войти в приложение.

Пользовательские страницы содержат удобные навигационные компоненты, позволяющие быстро добавлять студентов, группы, оценки, предметы, расписания и преподавателей изменять свои данные и выходить с приложение. Каждый графический блок для отдельного пользователя содержит удобные компоненты для обновления (изменения) информации о данном пользователе, для удаления.

Каждой внешней модели доступа к данным соответствует форма ввода/вывода. Формы ввода/вывода были реализованы с помощью стандартных html страниц. Они позволяют получать данные из базы данных и проводить различные манипуляции с этими данными, после чего все изменения сохраняются в БД.

Главная страница (на рисунке 2.4), на которую попадает пользователь, ещё до входа в приложение index.jsp. Реализованный интерфейс позволяет осуществлять регистрацию новых пользователей, а также вход в систему пользователей, которые уже зарегистрированы.

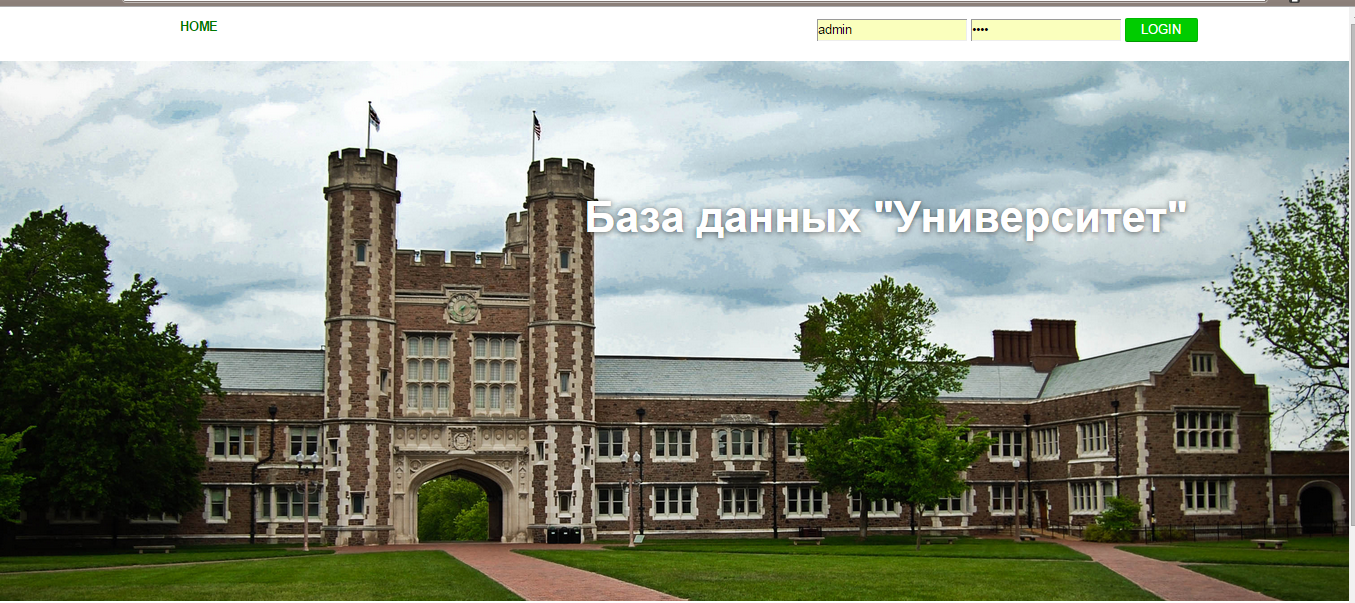


Рисунок 2.4 – Вид главной страницы интерфейса

Под словом навигация, будем подразумевать перемещение между Web-страницами. Навигация по событиям происходит по клику на соответствующем элементе списка (рисунок 2.5). Можно перейти на предыдущую страницу или на страницу Home.



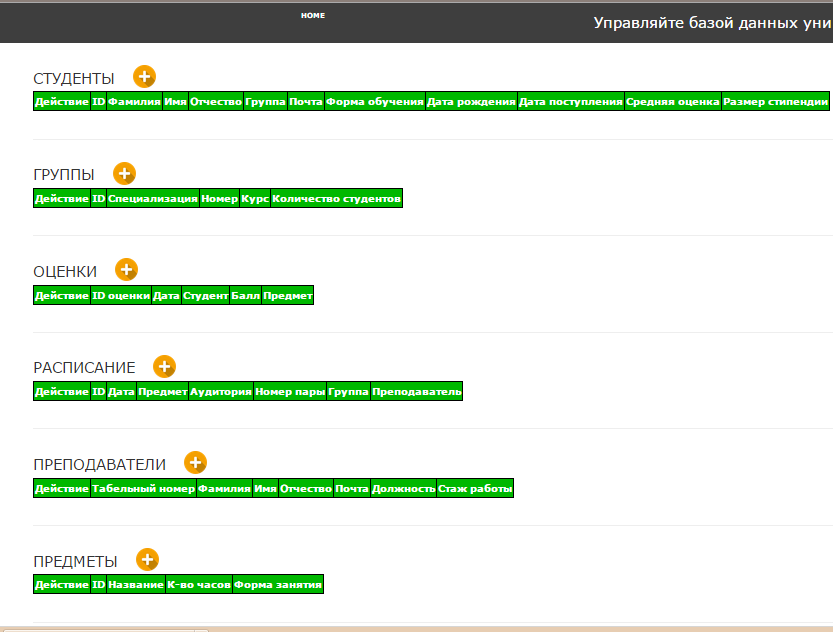


Рисунок 2.5 – Навигация по интерфейсу

# Разработка

Разработка – это центральный раздел, в котором описывается вся основная работа. Происходит выбор модели данных, разрабатывается логическая структура данных и модулей системы.

## Разработка базы данных системы

Разработка баз данных — процесс создания схемы [базы данных](http://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и определения необходимых ограничений [целостности](http://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

### Выбор модели данных

Иерархическая структура представляет совокупность элементов, связанных между собой по определенным правилам. Графическим способом представления иерархической структуры является дерево (рисунок 3.1).

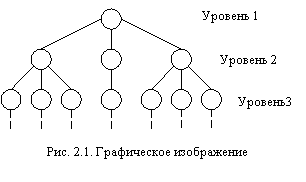


Рисунок 3.1 – Графическое изображение

Дерево представляет собой иерархию элементов, называемых узлами. Под элементами понимается совокупность атрибутов, описывающих объекты. В модели имеется корневой узел (корень дерева), который находится на самом верхнем уровне и не имеет узлов, стоящих выше него. У одного дерева может быть только один корень. Остальные узлы, называемые порожденными, связаны между собой следующим образом: каждый узел имеет только один исходный, находящийся на более высоком уровне, и любое число (один, два или более, либо ни одного) подчиненных узлов на следующем уровне.

К достоинствам иерархической модели данных относятся эффективное использование памяти ЭВМ и неплохие показатели времени выполнения операций над данными.

Недостатком иерархической модели является ее громоздкость для обработки информации с достаточно сложными логическими связями.

Отличие сетевой структуры от иерархической заключается в том, что каждый элемент в сетевой структуре может быть связан с любым другим элементом (рисунок 3.2).

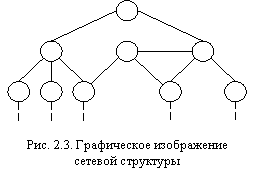


Рисунок 3.2 – Графическое изображение сетевой структуры

Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности.

Недостатком сетевой модели данных являются высокая сложность и жесткость схемы БД, построенной на ее основе.

Реляционная база данных представляет собой хранилище данных, организованных в виде двумерных таблиц. Любая таблица реляционной базы данных состоит из строк (называемых также записями) и столбцов (называемых также полями).

Строки таблицы содержат сведения о представленных в ней фактах (или документах, или людях, одним словом, - об однотипных объектах). На пересечении столбца и строки находятся конкретные значения содержащихся в таблице данных.

Данные в таблицах удовлетворяют следующим принципам:

* каждое значение, содержащееся на пересечении строки и столбца, должно быть атомарным;
* значения данных в одном и том же столбце должны принадлежать к одному и тому же типу, доступному для использования в данной СУБД;
* каждая запись в таблице уникальна, то есть в таблице не существует двух записей с полностью совпадающим набором значений ее полей;
* каждое поле имеет уникальное имя;
* последовательность полей в таблице несущественна;
* последовательность записей в таблице несущественна.

Поле или комбинацию полей, значения которых однозначно идентифицируют каждую запись таблицы, называют **возможным ключом** (или просто **ключом**).

Если таблица имеет более одного возможного ключа, тогда один ключ выделяют в качестве **первичного**. Первичный ключ любой таблицы обязан содержать уникальные непустые значения для каждой строки.

Поле, указывающее на запись в другой таблице, связанную с данной записью, называется **внешним ключом**.

Подобное взаимоотношение между таблицами называется **связью**. Связь между двумя таблицами устанавливается путем присвоения значений внешнего ключа одной таблицы значениям первичного ключа другой.

Группа связанных таблиц называется **схемой базы данных**. Информация о таблицах, их полях, первичных и внешних ключах, а также иных объектах базы данных, называется **метаданными**.

Достоинство реляционной модели данных заключается в простоте, понятности и удобстве физической реализации на ЭВМ.

К основным недостаткам реляционной модели относятся отсутствие стандартных средств идентификации отдельных записей и сложность описания иерархических и сетевых связей.

### Разработка схемы базы данных

Исходя из анализа и проектирования слоя бизнес-логики и правил была построена структура БД. На рисунке 3.2 приведена концептуальная модель базы данных, а на рисунке 3.3 – физическая модель базы данных.

Рисунок 3.1 – Концептуальная модель базы данных

Рисунок 3.2 – Физическая модель базы данных

В базе данных программной системы содержится вся информация относительно её объектов, а именно о :

- леснике;

- поставке;

- древесине;

- содержимом поставки;

- материале изделия;

- изделии;

Между объектами установлены такие связи:

* ЛЕСНИЧИЙ(поставляет) ПОСТАВКА(степень связи N:1);
* ПОСТАВКА(состоит из) ДРЕВЕСИНА(степень связи M:N);
* ИЗДЕЛИЕ(состоит из) ДРЕВЕСИНА(степень связи M:N);

В каждой таблице уникальный первичный ключ является суррогатным. Это дополнительное служебное поле, добавленное к уже имеющимся информационным полям таблицы, единственное предназначение которого — служить первичным ключом. Значения этого поля не образуется на основе каких-либо других данных из БД, а генерируются искусственно. Главное достоинство суррогатного ключа состоит в том, что он никогда не изменяется, поскольку не является информативным полем таблицы (не несёт никакой информации об описываемом записью объекте). Использовать суррогатный ключ имеет смысл в случае, когда возможны изменения полей, составляющих (естественный) первичный ключ (в особенности если этот ключ — составной). В этом случае возникает проблема так называемых "каскадных изменений". При использовании же суррогатного ключа в качестве первичного, изменять его не придётся. Также, при выполнении запросов, использующих суррогатные ключи, сравнение полей будет происходить быстрее, в особенности, если естественный первичный ключ представляет собой строку.

### Обеспечение целостности данных

Обеспечение целостности представляют собой утверждения о допустимых значениях отдельных информационных единиц и связях между ними.

Обеспечение целостности могут относиться к разным информационным объектам: атрибутам, кортежам, отношениям, связям между ними и тому подобное.

Для полей (атрибутов) используются следующие виды ограничений:

* тип и формат поля;
* задание диапазона значений;
* недопустимость пустого поля;
* задание домена;
* проверка на уникальность значения какого-либо поля. Ограничение

позволяет избежать записей-дубликатов.

Ограничения целостности нашей базе данных описаны в таблицах 3.1-3.6.

Таблица 3.1 – Описание таблицы «Человек»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| forester\_id | Идентификационный код | SERIAL | Первичный ключ |
| FirstName | Имя | CHAR(25) | Обязательное для ввода |
| LastName | Фамилия | CHAR(25) | Обязательное для ввода |
| login | Логин | CHAR(25) | Обязательное для ввода, уникальное |

Таблица 3.2 – Описание таблицы «Поставки»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| suppl\_id | Идентификатор поставки | SERIAL | Первичный ключ |
| forester\_id | Идентификатор лесника | SERIAL | внешний ключ, запрет обновления, запрет удаления;  обязательное для ввода. |
| date | Дата поставки | DATE | Default: CURRENT\_DATE |

Таблица 3.3 – Описание таблицы «Содержимое поставки»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| id | Идентификатор | SERIAL | Первичный ключ |
| suppl\_id | Идентификатор поставки | SERIAL | внешний ключ, запрет обновления, запрет удаления;  обязательное для ввода. |
| woods\_id | Идентификатор древесины | SERIAL | внешний ключ, запрет обновления, запрет удаления;  обязательное для ввода. |
| Count | Количество кубометров данной древесины в поставке | SMALLINT | Ограничение:  count > 0 and  count < 1000  NOT NULL |

Таблица 3.4 – Описание таблицы «Древесина»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| woods\_id | Идентификатор | SERIAL | Первичный ключ |
| name | Название вида древесины | CHAR(25) | Обязательно ко вводу, уникальное |
| Type | Тип древесины | CHAR(25) | Обязательно ко вводу, type = 'хвойная' OR type = 'лиственная' |
| Price | Cтоимость одного кубометра данного типа древесины | SMALLINT | Обязательно ко вводу, price > 0  NOT NULL |
| balance | Количество кубометров данной древесины на складе | INT | Ограничение:  balance > 0 and  balance < 1000  NOT NULL |

Таблица 3.5 – Описание таблицы «Материалы»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| id | Идентификатор | SERIAL | Первичный ключ |
| product\_id | Идентификатор изделия | SERIAL | внешний ключ, запрет обновления, запрет удаления;  обязательное для ввода. |
| woods\_id | Идентификатор древесины | SERIAL | внешний ключ, запрет обновления, запрет удаления;  обязательное для ввода. |
| count | Количество кубометров данной древесины необходимой для производства изделия | SMALLINT | Ограничение:  count > 0 and  count < 1000  NOT NULL |

Таблица 3.6 – Описание таблицы «Изделия»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| product\_id | Идентификатор | SERIAL | Первичный ключ |
| name | Название вида изделия | CHAR(25) | Обязательно ко вводу, уникальное |
| Price | Cтоимость одного изделия | SMALLINT | Обязательно ко вводу, price > 0  NOT NULL |
| balance | Количество изделий на складе | SMALLINT | Ограничение:  balance > 0 and  balance < 1000  NOT NULL |

Таблица 3.7 – Описание таблицы «Рабочий»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| HumanId | Внешний ключ | SERIAL | Внешний ключ |

Таблица 3.7 – Описание таблицы «Лесник»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| HumanId | Внешний ключ | SERIAL | Внешний ключ |

Таблица 3.7 – Описание таблицы «Админ»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Описание | Тип данных | Ограничения |
| HumanId | Внешний ключ | SERIAL | Внешний ключ |

### Физическое проектирование базы данных

**DROP** **INDEX** IX\_Woods\_1;

**DROP** **INDEX** IX\_Product\_1;

**DROP** **INDEX** IX\_Foresters\_1;

**DROP** **INDEX** IX\_Foresters\_2;

**DROP** **TABLE** Materials;

**DROP** **TABLE** SupplyContents;

**DROP** **TABLE** Supply;

**DROP** **TABLE** Foresters;

**DROP** **TABLE** Products;

**DROP** **TABLE** Woods;

**CREATE** **TABLE** Woods (

woods\_id SERIAL **NOT** **NULL**

, name **CHAR**(25) **NOT** **NULL** **CONSTRAINT** UQ\_Woods\_1 **UNIQUE**

, price **SMALLINT** **DEFAULT** '0' **NOT** **NULL**

, **type** **CHAR**(25) **NOT** **NULL**

, balance **INT** **DEFAULT** 0

, **PRIMARY** **KEY** (woods\_id)

);

**CREATE** **TABLE** Products (

product\_id SERIAL **NOT** **NULL**

, productName **CHAR**(30) **NOT** **NULL** **CONSTRAINT** UQ\_Product\_1 **UNIQUE**

, balance **SMALLINT** **DEFAULT** '0'

, price **SMALLINT** **DEFAULT** '0' **NOT** **NULL**

, **PRIMARY** **KEY** (product\_id)

);

**CREATE** **TABLE** Foresters (

forester\_id SERIAL **NOT** **NULL**

, firstName **CHAR**(25) **NOT** **NULL**

, lastName **CHAR**(25) **NOT** **NULL**

, middleName **CHAR**(25) **NOT** **NULL**

, phoneNumber **CHAR**(25) **DEFAULT** '111-111-11'

, **PRIMARY** **KEY** (forester\_id)

);

**CREATE** **INDEX** IX\_Foresters\_1 **ON** Foresters (firstName, lastName, middleName);

**CREATE** **INDEX** IX\_Foresters\_2 **ON** Foresters (lastName);

**CREATE** **TABLE** Supply (

suppl\_id SERIAL **NOT** **NULL**

, forester\_id SERIAL **NOT** **NULL**

, **date** **DATE** **DEFAULT** CURRENT\_DATE **NOT** **NULL**

, **PRIMARY** **KEY** (suppl\_id)

, **CONSTRAINT** FK\_Supply\_1 **FOREIGN** **KEY** (forester\_id)

**REFERENCES** Foresters (forester\_id) **ON** **DELETE** **RESTRICT** **ON** **UPDATE** **RESTRICT**

);

**CREATE** **TABLE** SupplyContents (

id SERIAL **NOT** **NULL**

, suppl\_id SERIAL **NOT** **NULL**

, woods\_id SERIAL **NOT** **NULL**

, count **SMALLINT** **DEFAULT** '0' **NOT** **NULL**

, **PRIMARY** **KEY** (id)

, **CONSTRAINT** "FK\_Содержимое\_поставки\_1" **FOREIGN** **KEY** (suppl\_id)

**REFERENCES** Supply (suppl\_id) **ON** **DELETE** **RESTRICT** **ON** **UPDATE** **RESTRICT**

, **CONSTRAINT** "FK\_Содержимое\_поставки\_2" **FOREIGN** **KEY** (woods\_id)

**REFERENCES** Woods (woods\_id) **ON** **DELETE** **RESTRICT** **ON** **UPDATE** **RESTRICT**

);

**CREATE** **TABLE** Materials (

id SERIAL **NOT** **NULL**

, product\_id SERIAL **NOT** **NULL**

, woods\_id SERIAL **NOT** **NULL**

, count **SMALLINT** **DEFAULT** '0' **NOT** **NULL**

, **PRIMARY** **KEY** (id)

, **CONSTRAINT** FK\_Materials\_1 **FOREIGN** **KEY** (product\_id)

**REFERENCES** Products (product\_id) **ON** **DELETE** **RESTRICT** **ON** **UPDATE** **RESTRICT**

, **CONSTRAINT** FK\_Materials\_2 **FOREIGN** **KEY** (woods\_id)

**REFERENCES** Woods (woods\_id) **ON** **DELETE** **RESTRICT** **ON** **UPDATE** **RESTRICT**

);

Триггер для добавления:

При добавлении записей в таблицу «Содержимое поставки» должна выполнятся проверка есть ли в данной поставке древесина иного класса, если есть то отображается сообщение об ошибке ввода.

Триггер для изменения:

При увеличении значения в столбце «количество» в таблице «Изделия» должно происходить уменьшения количества древесины, указанной в таблице «Материалы», в таблице «Древесина» пропорционально изменению количества изделий.

Триггер для удаления:

При удалении из таблицы «Поставки» необходимо также удалить все записи в таблице «Содержимое поставки» которые ссылаются на данную таблицу.

### Выбор индексов

Для удобного использования системы стоит использовать поля login и id из таблицы Human (Человек). Так будет удобно осуществлять поиск человека, для проведения нужных модификаций в таблице.

Также для быстрого и точного поиска студента был создан индекс indWood. В этом индексе объеденены такие поля как woodName и woodId таблицы Woods.

Также для ускорения поиска необходимо добавить индекс в таблицу Products (изделия), состоящий из полей productId и productName.

### Разработка хранимых процедур и триггеров

**3.1.6.1 Описание хранимых процедур**

Хранимые процедуры:

1) Процедура findMostUsedWoods

Входные параметры:

Выходные параметры: таблица со столбцами woods\_id, Woods.name, count.

CREATE OR REPLACE FUNCTION findMostUsedWoods()

RETURNS TABLE(name character(25), count bigint) AS $$

SELECT name, count

FROM woods, (

SELECT woods\_id AS id, COUNT(woods\_id)

FROM (SELECT woods.woods\_id

FROM woods INNER JOIN materials USING(woods\_id)) AS tmp

GROUP BY woods\_id) AS tmp2

WHERE woods.woods\_id = tmp2.id

ORDER BY count DESC

LIMIT 1

$$

LANGUAGE sql;

2) Процедура findMostActiveForester

Входные параметры: –

Выходные параметры: таблица со столбцами forester\_id, Forester.lastName, count.

Процедура находит всех лесников и количество их поставок в порядке убывания.

CREATE OR REPLACE FUNCTION findMostActiveForesters()

RETURNS TABLE (lastName character(25),firstName character(25), count bigint) AS $$

SELECT firstName, lastName, count

FROM foresters, (

SELECT forester\_id AS id, COUNT(forester\_id)

FROM (SELECT foresters.forester\_id

FROM foresters INNER JOIN supply USING(forester\_id)) AS tmp

GROUP BY forester\_id) AS tmp2

WHERE foresters.forester\_id = tmp2.id

ORDER BY count DESC

LIMIT 1

$$

LANGUAGE sql;

#### Описание триггеров

1. AFTER INSERT ON Student: при добавлении новой группы, курсу присвоить значение 1 . Реализация триггера на SQL:

CREATE OR REPLACE FUNCTION trig\_kurs()

RETURNS trigger AS

$BODY$BEGIN

NEW.course=’1’;

RETURN NEW;

END; $BODY$

LANGUAGE 'plpgsql' VOLATILE

COST 100;

ALTER FUNCTION trig\_kurse () OWNER TO postgres;

2) INSERT/ UPDATE INTO student

Если средний балл в таблице student >=4.5, то обновить поле stip, а именно увеличить значение на 20%.

CREATE OR REPLACE FUNCTION () stud\_trig RETURNS trigger AS

$$BEGIN

IF NEW.sr\_ball >=4.5 Then

NEW.stip= stip\*0.2;

END IF;

RETURN NEW;

LANGUAGE 'plpgsql' VOLATILE

COST 100;

ALTER FUNCTION stud\_trig () OWNER TO postgres;

1. AFTER INSERT ON Student: при добавлении нового студента группы, увеличить количество студентов на 1 . Реализация триггера на SQL:

CREATE OR REPLACE FUNCTION trig\_cnt()

RETURNS trigger AS

$BODY$BEGIN

NEW.cnt\_stud=cnt\_stud+1;

RETURN NEW;

END; $BODY$

LANGUAGE 'plpgsql' VOLATILE

COST 100;

ALTER FUNCTION trig\_cnt() OWNER TO postgres;

### Организация защиты данных

Пользователями БД в приведенном примере могут быть: администратор БД, сотрудник пилорамы, лесник и гость. Опишем для каждой группы пользователей права доступа к каждой таблице и к каждому полю (атрибуту). Администратор БД имеет доступ ко всем данным (по записи), может изменять структуру базы данных и связи между отношениями, устанавливает права доступа для всех остальных групп.

Лесник имеет возможность добавлять и просматривать информацию о поставках, увеличивать количество древесины, а также смотреть закупаемую древесину и продаваемые материалы

Сотрудник может просматривать наявную древесину, а также уменьшать ее количество, также он может просматривать изготовляемые изделия и увеличивать их количество.

Гость на веб-сайте может просматривать закупаемую древесину, а также просматривать изготовляемые изделия.

Перечисленные права доступа удобно представить в виде таблицы 3.2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Роль | Сотрудник | Лесник | Поставка | Содержимое поставки | Древесина | Изделия | Материалы |
| Администраор | S,U,D,I | S,U,D,I | S,U,D,I | S,U,D,I | S,U,D,I | S,U,D,I | S,U,D,I |
| Лесник | - | - | S,I | S,I | S,U | S | - |
| Сотрудник | - | - | - | - | S,U | S,U | S |
| Гость | - | - | - | - | S | S | - |

В таблице S –select, U –update, D –delete, I –insert.

### Объектно-реляционное отображение

ОБЪЕКТНО-РЕЛЯЦИОННОЕ отображение (ORM) - это техника программирования, которая связывает реляционнкю базу данных с концепциями объектно-ориентированного программирования и создаёт «виртуальную базу данных объектов». Причинами появления и развития этой техники программирования являются следующие:

1. работа переносится с уровня работы с записями БД на другой уровень - уровень работы с объектами;
2. большая гибкость организации доступа к данным (объект может состоять из полей различных таблиц);
3. поддержка доступа нескольких пользователей к одним и тем же объектам (работа с persistent объектами);
4. однообразный доступ к различным БД - разработчик не принимает во внимание сервер БД, с каким он работает (PostgreSQL, MySQL, Oracle);
5. упрощение организации связи между объектами - ссылки на объекты;
6. уменьшение времени доступа к данным за счёт использования механизма кэширования.

На сегодняшний день существуют различные Фреймворки, использующие данную технику программирования. Вот некоторые из них: Hibernate, iBATIS, Java Data Objects (JDO), JPOX, Cayenne, TopLink.

При разработке программы осуществляется доступ к данным, поэтому для упрощения разработки такой программы и для увеличения эффективности и скорости работы с полученными данными использовалось объектно-реляционное отображение и его реализация на OpenJPA.

Системы управления реляционными базами данных показывают хорошую производительность на глобальных запросах, которые затрагивают большой участок базы данных, но объектно-ориентированный доступ более эффективен при работе с малыми объёмами данных, так как это позволяет сократить семантический провал между объектной и реляционной формами данных.

Все системы ORM обычно проявляют себя в том или ином виде, уменьшая в некотором роде возможность игнорирования базы данных. Более того, слой транзакций может быть медленным и неэффективным (особенно в терминах сгенерированного SQL). Но ORM избавляет программиста от написания большого количества кода, часто однообразного и подверженного ошибкам, тем самым значительно повышая скорость разработки. Кроме того, большинство современных реализаций ORM позволяют программисту при необходимости самому жёстко задать код SQL-запросов, который будет использоваться при тех или иных действиях (сохранение в базу данных, загрузка, поиск и.т.д.) с постоянным объектом.

## Разработка модулей системы

Понятие модуль (от лат. modulus — «маленькая мера») в общем означает составную часть, отделимую или хотя бы мысленно выделяемую из общего. Модульной обычно называют вещь, состоящую из чётко выраженных частей, которые нередко можно убирать или добавлять, не разрушая вещь в целом.

Разработка модулей системы включает в себя несколько слоев: бизнес – логику и отображение.

### Разработка модулей слоя бизнес логики и бизнес правил

Данный модуль представляет собой описания всех сущностей БД.

Он включает в себя 4 класа, а именно:

**Admin.java -** класс, который описывает администратора.

package domain;

import javax.persistence.Entity;

@Entity

public class Admin extends Human{

}

**Forester.java -** класс, который описывает лесника. Он содержит методы получения: и записи полей.

@Entity

public class Forester extends Human {

@OneToMany(cascade = CascadeType.ALL)

@JoinColumn(name = "forester\_humanid")

List<Supply> supplyList = new ArrayList<Supply>();

public List<Supply> getSupplyList() {

return supplyList;

}

public void setSupplyList(List<Supply> supplyList) {

this.supplyList = supplyList;

}

}

**Human.java -** абстрактныйкласс, который описывает человека. Он содержит методы получения: и записи полей.

@Entity

@Inheritance(strategy=InheritanceType.JOINED)

public abstract class Human {

@Id

@GeneratedValue(strategy = GenerationType.TABLE)

private int humanId;

private String firstName;

private String lastName;

@Column(unique=true)

private String login;

private String password;

public int getHumanId() {

return humanId;

}

public void setHumanId(int id) {

humanId = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) throws Exception {

if (firstName.length() > 25){

throw new Exception("firstName has more then 25 characters");

}

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) throws Exception {

if (lastName.length() > 25){

throw new Exception("lastName has more then 25 characters");

}

this.lastName = lastName;

}

public String getLogin() {

return login;

}

public void setLogin(String login) throws Exception {

if (login.length() > 25){

throw new Exception("middleName has more then 25 characters");

}

this.login = login;

}

public String getPassword() {

return password;

}

public void setPassword(String password) throws Exception {

if (password.length() > 25){

throw new Exception("password has more then 25 characters");

}

this.password = password;

}

}

**Material.java** - класс, который описывает материал для создания изделия. Он содержит методы получения: и записи полей.

@Entity

public class Material {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@ManyToOne

private Wood wood;

@ManyToOne

private Product product;

private int count;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getCount() {

return count;

}

public Wood getWood() {

return wood;

}

public void setWood(Wood wood) {

this.wood = wood;

}

public Product getProduct() {

return product;

}

public void setProduct(Product product) {

this.product = product;

}

public void setCount(int count) {

this.count = count;

}

}

**Product.java** - класс, который описывает изделие. Он содержит методы получения: и записи полей.

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int Id;

@Column(unique = true)

private String productName;

private int balance;

private int price;

@OneToMany(cascade = CascadeType.ALL)

@JoinColumn(name = "product\_productid")

List<Material> materialList = new ArrayList<Material>();

public int getId() {

return Id;

}

public void setId(int id) {

Id = id;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) throws DBConstraintsViolationException {

if (productName.length() > 25){

throw new DBConstraintsViolationException("слишком длинное название изделия");

}

this.productName = productName;

}

public int getBalance() {

return balance;

}

public void setBalance(int balance) {

this.balance = balance;

}

public int getPrice() {

return price;

}

public void setPrice(int price) throws BusinessLogicViolationException {

if (price < 0){

throw new BusinessLogicViolationException("стоимость меньше ноля");

}

this.price = price;

}

public List<Material> getMaterialList() {

return materialList;

}

public void setMaterialList(List<Material> materialList) {

this.materialList = materialList;

}

}

**Supply.java** - класс, который описывает поставку. Он содержит методы получения: и записи полей.

@Entity

public class Supply {

@Id

@GeneratedValue(strategy = GenerationType.TABLE)

private int supplyId;

@OneToMany(cascade = CascadeType.ALL)

@JoinColumn(name = "supply\_supplyid")

List<SupplyContents> supplyContentList = new LinkedList<SupplyContents>();

@Column(name = "date")

private Date supplyDate;

@ManyToOne

private Forester forester;

public Forester getForester() {

return forester;

}

public void setForester(Forester forester) {

this.forester = forester;

}

@Column(name = "total\_cost")

private int totalCost;

public int getSupplyId() {

return supplyId;

}

public void setSupplyId(int id) {

supplyId = id;

}

public Date getSupplyDate() {

return supplyDate;

}

public void setSupplyDate(Date supplyDate) {

this.supplyDate = supplyDate;

}

public int getTotalCost() {

return totalCost;

}

public void setTotalCost(int totalCost) {

this.totalCost = totalCost;

}

public List<SupplyContents> getSupplyContentList() {

return supplyContentList;

}

public void setSupplyContentList(List<SupplyContents> supplyContentList) {

this.supplyContentList = supplyContentList;

}

}

**SupplyContents.java** - класс, который описывает содержимое поставки. Он содержит методы получения: и записи полей.

@Entity

@Table(name = "supplycontents")

public class SupplyContents {

@ManyToOne

private Supply supply;

public Wood getWood() {

return wood;

}

public void setWood(Wood wood) {

this.wood = wood;

}

@ManyToOne

private Wood wood;

public Supply getSupply() {

return supply;

}

public void setSupply(Supply supply) {

this.supply = supply;

}

@Id

@GeneratedValue(strategy = GenerationType.TABLE)

private int id;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getCount() {

return count;

}

public void setCount(int count) throws Exception {

if (count < 0) {

throw new Exception("count < 0");

}

this.count = count;

}

private int count;

}

**Wood.java** - класс, который описывает древесину. Он содержит методы получения: и записи полей.

@Entity

public class Wood {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int woodsId;

@OneToMany(cascade = CascadeType.ALL)

@JoinColumn(name = "wood\_woodsid")

List<SupplyContents> supplyContentList = new ArrayList<SupplyContents>();

@OneToMany(cascade = CascadeType.ALL)

@JoinColumn(name = "wood\_woodsid")

List<Material> materialList = new ArrayList<Material>();

private int price;

@Enumerated(EnumType.STRING)

private WoodType type;

private int balance;

@Column(unique = true)

private String name;

private boolean buying;

public boolean getBuying() {

return buying;

}

public void setBuying(boolean isBuying) {

this.buying = isBuying;

}

public String getName() {

return name;

}

public void setName(String name) throws Exception {

if (name.length() > 25){

throw new Exception("name has more then 25 characters");

}

this.name = name;

}

public int getWoodsId() {

return woodsId;

}

public void setWoodsId(int woodsId) {

this.woodsId = woodsId;

}

public List<SupplyContents> getSupplyContentList() {

return supplyContentList;

}

public void setSupplyContentList(List<SupplyContents> supplyContentList) {

this.supplyContentList = supplyContentList;

}

public int getPrice() {

return price;

}

public void setPrice(int price) throws Exception {

if (price <= 0){

throw new Exception("Стоимость меньше нуля: " + price);

}

this.price = price;

}

public WoodType getType() {

return type;

}

public void setType(WoodType type) {

this.type = type;

}

public int getBalance() {

return balance;

}

public void setBalance(int balance) {

this.balance = balance;

}

public List<Material> getMaterialList() {

return materialList;

}

public void setMaterialList(List<Material> materialList) {

this.materialList = materialList;

}

}

**Worker.java** - класс, который описывает рабочего.

@Entity

public class Worker extends Human{

}

### Разработка модулей слоя доступа к данным

Доступ к данным осуществляется с помощью DAO. Этот модуль представлен пятью классами:

ForesterDAO.java - содержит описания методов, для работы c лесниками.

HumanDAO.java – содержик методы для работы с абстрактными пользователями.

ProductDAO.java - содержит описания методов, для работы со списком изделий.

WoodDAO.java - содержит описания методов, для работы с древесиной.

SupplyDAO.java – содержит описания методов, для работы с gjcnfdrfvb..

### Разработка модулей слоя отображения

Данное приложение имеет в себе набор jsp страниц предоставляющих возможность работы с разнородными сущностями. Перечень jsp страниц:

1. index.jsp – на той странице пользователь должен авторизироваться.
2. craft.jsp – страница на которой рабочий может создавать новые изделия.
3. add\_product.jsp - страница для добавления новых изделий.
4. аdd\_user.jsp — стралица добавления новых пользователей.
5. Add\_woods.jsp - страница добавления новой древесины.
6. Craft\_list.jsp – список материалов для создания изделия для рабочего.
7. Edit\_product.jsp – администратор может редактировать и удалять производимые изделия.
8. Edit\_user.jsp – администратор может редактировать и удалять пользователей.
9. Edit\_woods.jsp – администратор может редактировать древесину.
10. products.jsp – страница просмотра производимых изделий.
11. sell.jsp – страница для совершения поставки лесником.
12. Top\_foresters.jsp – страница просмотра наиболее активных лесников.
13. Woods.jsp – страница просмотра закупаемой древесины.
14. Top\_woods.jsp – страница просмотра наиболее используемой древесины.

Также было применен CSS для оформления страниц.

### Разработка модулей слоя бизнес логики и бизнес правил

Данный модуль представляет собой описания всех сущностей нашей БД. Он включает в себя семь классов, а именно:

* SupplyBuilder.java –класс, в который предназначен для создания новой поставки, в нем выполняются проверки смешения различных типов древесины.
* ProductBuilder.java – класс, для построения изделия из множества объектов класса Material.
* Calculator.java – класс, который проводит разнообразные финансовые вычисления.
* BussinessLogicService.java – класс, в котором содержатся вспомогательные методы бизнес логики.

ВЫВОДЫ

В результате выполнения данной курсовой работы были реализованы все требования, изложенные в техническом задании, в результате чего появился работоспособное корпоративное web приложение, не требующее больших ресурсов.

Получены практические навыки по работе с ORM. В частности с фреймворками, использующими данную технику программирования (Hibernate). Также была освоена технология Java Server Page, технология веб сервисов. Были приобретены навыки в проектировании корпоративных приложений и использовании шаблонов проектирования, таких как: Singleton и Builder.

Был создано веб приложение для электронного песенника со стандартным набором функций такими как:

1. авторизация;
2. добавление, удаление, изменение данных о древесине;
3. добавление, удаление, изменение данных о пользователе;
4. совершение поставок лесниками;
5. создание изделий рабочими;
6. просмотр списка наиболее активных лесников;
7. просмотр списка наиболее часто используемой древесины.

Путем усовершенствования проекта может стать улучшение работы с сессиями, упрощение выполнения основных действий, расширение функций программы (возможность восстановления пароля, усовершенствование внешнего вида). Так же для улучшения безопасности web приложения можно использовать шифрование паролей и т.д.

При внедрении всех возможных улучшений проект вполне может стать конкурентоспособным продуктом: Однако на данном этапе проект является учебным примером.

ПЕРЕЧЕНЬ ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Проектування баз даних для корпоративних застосувань. Методичні вказівки до курсової роботи з дисципліни «Організація баз даних» для студентів напряму підготовки 6.050201 – „Комп'ютерна інженерія”. [Електронний ресурс] /Укл.: А.Л. Зінченко, М.В.Харченко − Рос. Мовою.
2. „Школа SQL”./ Бойко В.В., Савинков В.М – Москва: Мир, 2002.
3. Структурированный язык запросов (SQL)./ Кириллов В.В. – СПб.: ИТМО, 1994.
4. Проектирование реляционных баз данных для использования с микроЭВМ. /Джексон Г. – М.: Мир, 1998.
5. Применение шаблонов Java. Библиотека профессионала. : Пер. с англ. — М.: Издательский дом "Вильямс", 2002. — 576 с.: ил. — Парал. тит. англ.
6. Амриш Кэри «Разработка корпоративных Java-приложений с помощью J2EE и UML»: Пер. с англ. — М.: Издательский дом "Вильямс",2002. – 727 с.: ил. — Парал. тит. англ.
7. Шпильман С. JSTL. Практическое руководство для JSP-программистов. Москва:КУДИЦ-Образ, 2004. – 272 с.
8. Брюс У. Перри. Java сервлеты и JSP. Сборник рецептов. Москва: КУДИЦ-Образ, 2006. – 768 с.
9. Мартин Р. Читый код: создание, анализ и рефакторинг. Библиотека программиста. – СПб.: Питер, 2010. – 446с.
10. Bloch J. Effective Java. Second Edition Addison-Wesley. – NJ, 2008 – 369с.
11. Хейк Б. JDBC. Java и базы данных. – Москва.: Лори, 2000. – 324 с

ПРИЛОЖЕНИЯ