

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

МЭИ

Институт: Информационных и Вычислительных Технологий

Кафедра: Вычислительных Технологий

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА **К КУРСОВОЙ РАБОТЕ ПО КУРСУ:**

Проектирование баз данных

НА ТЕМУ:

Склад строительной фирмы

Выполнил

Студент

Булхак А.Н.

Группа

А-06М-19

Принял

Преподаватель

Бородин Г.А.

Москва 2020

Оглавление

Пояснения о предметной области и назначении автоматизированной информационной системы	3
Техническое задание	4
1. Анализ технического задания	11
2. Описание базы данных и ее объектов	13
3. Разработка приложения-сервер	23
4. Интерфейс приложения на стороне клиента	33
5. Комплект поставки и методика настройки разработанной системы	35
Заключение	37
Список литературы	38

Пояснения о предметной области и назначении автоматизированной информационной системы

Целью курсового проекта является освоение принципов проектирования приложения-клиента, сервера-приложений и сервера СУБД трёхзвенной архитектуры «клиент-сервер». Использование технологии «клиент-сервер» в настоящее время является актуальной задачей при работе с базами данных. К ее достоинствам относят:

- масштабируемость;
- конфигурируемость;
- высокая безопасность;
- высокая надёжность;
- низкие требования к скорости канала (сети) между клиентом и сервером приложений;
- низкие требования к производительности и техническим характеристикам оборудования на стороне клиента, как следствие снижение их стоимости.

В трёхзвенной архитектуре вся бизнес-логика выделяется в отдельное звено, называемое сервером приложений. Клиентское приложение реализует интерфейс пользователя и инициирует обращение к приложению-серверу. Приложение-сервер анализирует требования пользователя и формирует запросы к серверу БД. СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере. СУБД инициирует обращения к данным, находящимся на сервере, результат передается на сервер-приложений. Сервер приложений возвращает результат в клиентское приложение (пользователю). Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Техническое задание

1. Общие сведения

1.1. Полное наименование системы и ее условное обозначение

Система обеспечения доступа к реляционным данным, включающая приложение-клиента, сервер-приложений и сервер СУБД, спроектированная на основе трёхзвенной архитектуры клиент/сервер и работы с ней, создания типовых объектов баз данных.

СДД – система доступа к данным, АС – автоматизированная система.

1.2. Наименование разработчика и заказчика (пользователя) системы и их реквизиты

Разработчик:

Студент Булхак А.Н., А-06М-19.

Заказчик:

Национальный исследовательский университет “МЭИ”, Кафедра
Вычислительной техники в лице д.т.н., профессора Бородина Г. А.
111250, Москва, ул. Красноказарменная, д. 13

1.3. Перечень документов, на основании которых создается система

Рабочая программа учебной дисциплины “Проектирование баз данных”.

Составил: д.т.н., профессор Бородин Г. А.

Утвердил: Зав. Кафедрой Вычислительной техники, д.т.н., профессор
Топорков В. В.

1.4. Плановые сроки начала и окончания работы по созданию системы

Начало работ – 12 февраля 2020 года.

Окончание работ – 2 июня 2020 года.

1.5. Сведения об источниках и порядке финансирования работ

Работы по созданию АС ведутся в рамках учебного плана, предусмотренного документами, перечисленными в пункте 1.3. По результатам приема разработанной системы будет определена стоимость выполненных работ в баллах в виде оценки за курсовой проект.

1.6. Порядок оформления и предъявления заказчику результатов работ по созданию системы (ее частей), по изготовлению и наладке отдельных средств (технических, программных, информационных) и программно-технических (программно-методических) комплексов системы

1. На 3-4 учебных неделях необходимо утвердить техническое задание на выполнение курсового проекта с учетом указанных типовых требований к проекту.
2. За неделю до защиты курсового проекта разработчик может сдать пояснительную записку на проверку консультанту для проверки выполнения требований к содержанию пояснительной записки к типовому проекту и устранения грубых ошибок.
3. На защите курсового проекта необходимо уметь:
 - Установить клиентское приложение, сервер-приложений и сервер-СУБД на различные компьютеры.
 - Произвести регистрацию сервера-приложений.
 - Произвести установку, регистрацию и настройку вспомогательных файлов, утилит, библиотек, сервера СУБД и т. п.
 - Продемонстрировать параллельную работу не менее двух экземпляров приложений-клиентов с разных рабочих мест.
 - Аргументировано отстаивать выбранные решения.
 - По окончании защиты необходимо произвести очистку реестра, удаление клиентских приложений, а также всех объектов из базы данных (таблиц, процедур и функций и т. п.), например, с помощью файл-скрипта.

2. Назначение и цели создания (развития) системы

2.1. Назначение системы

Система предназначена для повышения надежности и простоты хранения и обработки информации.

2.2. Цели создания системы:

Основная цель разработки системы – получение инструмента для постоянного мониторинга и оперативного изменения информации. Основные преимущества, которые должны быть обеспечены внедрением БД:

- создания единой системы контроля информации;
- облегчения работы с информацией;
- обеспечения безопасности данных;
- повышения качества (полноты, точности, достоверности, своевременности, согласованности) информации.

3. Характеристика объектов автоматизации

Объектом автоматизации выбран один из отделов организации - склад. В рамках отдела был произведен анализ и выделение основных бизнес процессов:

- поставка купленных инструментов на склад;
- взятие оборудования со склада;
- возврат на склад;
- контроль физического состояния рабочего оборудования.

4. Требования к системе

4.1. Общие требования к системе

1. Разработать реляционную базу данных, состоящую из 2-3 связанных таблиц (1-To-Many) и 1-2 ссылочных таблиц. Включить поля типа Мемо и OLE-поля (BLOB поля).
2. Создать табличный и графический отчет, последний получаемый, например, с использованием OLE-связи с Excel.

3. Разработать многостраничную экранную форму, позволяющую редактировать данные в таблицах базы данных.
4. Работу с базой данных организовать с помощью триггеров, запросов, хранимых процедур и функций, курсоров, представлений, размещаемых в базе данных.
5. При создании приложений и базы данных предпочтительно использовать надписи, содержание и текст на национальном (русском) языке.

4.2. Требования к функциям, выполняемым системой

1. Реализация подключения двух приложений-клиентов с двух компьютеров.
2. Реализация возможности редактирования одной записи двумя приложениями-клиентами.
3. Реализация каскадного удаления записей из подчинённой таблицы при удалении записи из главной таблицы.
4. Реализация добавления записи в подчинённую таблицу, не имеющей соответствующей записи в главной.
5. Возможность сохранения и визуализации OLE-объектов.
6. Возможность создания табличного и графического (в Excel) отчётов.
7. Реализация создания и работы представления.
8. Реализация создания и работы хранимой процедуры.
9. Возможность очистки компьютеров от установленной программы.

4.3. Требования к видам обеспечения

1. Тип сервера баз данных: MS SQL Server 2008 R2.
2. Среда для разработки приложений: Java.
3. Создание отчетов: Crystal Reports 9.0 или выше, MS Report Bulder 3.
4. Дополнительные пакеты: Excel 12/14.
5. Технология для реализации 3-х-звенных распределенных приложений: WEB (Servlet / JSP).
6. Доступ к базе данных с использованием: JDBC.

5. Состав и содержание работ по созданию системы

Работы по созданию системы выполняются в несколько этап:

1. Проектирование;
2. Разработка приложений;
3. Создание отчетов;
4. Обеспечение взаимодействия между компонентами системы;
5. Написание пояснительной записки.

Конкретные сроки выполнения стадий и этапов разработки и создания определяются планом выполнения работ, являющимся неотъемлемой частью договора на выполнение работ по настоящему частному техническому заданию.

6. Порядок контроля и приёмки системы

Сдача разработанной системы будет происходить на компьютерах, пользователи которых будут иметь ограниченные права доступа как к локальному компьютеру, так и серверу домена.

При приемке системы необходимо будет продемонстрировать работу:

- поддержания ссылочной целостности посредством триггеров, ограничений, ссылочных спецификаций и т.д.;
- представления;
- хранимой процедуры;
- хранимой функции;
- двух клиентов одновременно.

7. Требования к документированию

Комплект документов, подлежащий разработке:

- Техническое задание
- Пояснительная записка

Выбор тех или иных технологий, принципов, компонентов и т. п. должен сопровождаться ссылками на литературу, подтверждающую правильность и обоснованность суждений разработчика.

Из содержимого должно быть понятно все о созданных частях системы и их использовании без привлечения разработчика для разъяснений.

В общем случае в пояснительной записке следует отразить следующие вопросы:

1. Пояснения о предметной области и назначении автоматизированной информационной системы.
2. Техническое задание и его анализ, включая предположительный размер базы данных, возможно с расчетами.
3. Назначение СУБД, обоснование выбора таблиц, количества и типов полей, доменов и других объектов базы данных. Выбор отношения между таблицами. Обоснование опций ссылочной целостности. Обоснование выбора типов полей для использования в качестве Primary Key, включая обеспечение уникальности. Доказательства нормализованности таблиц базы данных. Обоснование выбора компонентов или объектов для работы с таблицами. Нужны ли индексы, какие, если да?
4. Обоснование выбора уровня изоляции транзакций и его установка (на уровне компонентов или языка SQL). Важно пояснить, когда и кем тот или иной уровень изоляции транзакций устанавливается. Для поддержки транзакций необходим LOG-файл. Каков его размер для разрабатываемой базы данных.
5. Обоснование способа подключения к СУБД (количество соединений, используемые компоненты, их настройка). Необходимо указать достаточный уровень прав для настройки частей распределенного приложения.
6. Настройка средств связи (ODBC, BDE, JDBC и т. п.), определение параметров связи. Настройка сервера приложений на компьютере, включая использование суррогатных псевдопроцессов для функционирования Dll.
7. Документацию, полученную в результате моделирования структуры базы данных (модель физического уровня со слоями, демонстрирующими

ссылочную целостность), включая ограничения, домены, таблицы (например, в виде таблицы, сформированной CASE-средством), триггеры ссылочной целостности, хранимые процедуры и функции, курсоры в виде скрипт-файла, приводимого в приложении (полный текст SQL-скрипта не требуется).

- 8.Отчеты: диаграмма, использующая данные одной таблицы и воспроизводимая на поверхности экранной формы, и табличный, использующий данные главной и подчиненной таблицы и оформленный как таблица (название колонок, графическое оформление), включение вычисляемых полей и итоговых данных.
- 9.Описание методики обмена данными между полями, аналогичными Методу и OLE, с приложениями Windows типа Excel и т.п. Обосновать способ доступа к Excel (вариантная переменная, доступ через интерфейс или диспинтерфейс и др.).
- 10.Комплект поставки разработанного распределенного приложения с разбивкой на три компьютера.
- 11.Список используемой литературы.

8. Источники разработки.

Настоящее Техническое Задание разработано на основе следующих документов и информационных материалов:

- ГОСТ 34.602-89 Информационная технология. Комплекс стандартов на автоматизированные системы.Техническое задание на создание автоматизированной системы.

1. Анализ технического задания

Согласно техническому заданию, требуется разработать систему обеспечения доступа к реляционным данным, включающую приложение-клиента, сервер-приложений и сервер СУБД, спроектированную на основе трёхзвенной архитектуры клиент/сервер.

Так как создание приложения-сервера было решено выполнять на языке программирования Java[1], то и в качестве технологии, реализующей подключение приложения-сервера к серверу БД, была выбрана технология JDBC[2] (Java DataBase Connectivity - интерфейс, при помощи которого Java-приложения взаимодействуют с базами данных и манипулируют с их данными). Это наиболее удобоваримый способ работы с разными видами баз данных, так как JDBC предоставляет возможность использовать стандартные SQL-операторы для запросов к БД.

Для реализации взаимодействия между приложениями клиента и сервера была выбрана связка WEB технологий Servlet[3]/JSP[4](JavaServer Pages), позволяющая взаимодействовать с клиентами посредством принципа запрос-ответ и создавать содержимое веб-страниц, которое имеет как статические, так и динамические компоненты. Для запуска приложения также потребуется контейнер сервлетов Apache Tomcat[5], который реализует спецификацию сервлетов и JSP и содержит программы для самоконфигурирования.

Среда программирования IntelliJ Idea - интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java. Делает создание кода удобной. В ней реализуются приложение-сервер, а также есть возможность взаимодействия с базой данных, тем самым не требуется переходить из одного приложения в другое во время разработки системы.

Архитектура разработанной системы (рис. 1)

- Сервер базы данных – компьютер, с установленным Microsoft SQL Server[6] 2008.

- Приложение-сервер в виде war-файла, который размещается на Apache Tomcat.
- Клиент подключается к серверу в браузере через строку запроса.



Рис. 1. Архитектура системы

2. Описание базы данных и ее объектов

База данных предназначена для взаимодействия с информацией, которая представляет оборудование строительной фирмы. В ней указаны все строительные объекты, в том числе и склад фирмы, вся информация о каждом строительном инструменте и его передвижении со склада на какой-либо объект и обратно.

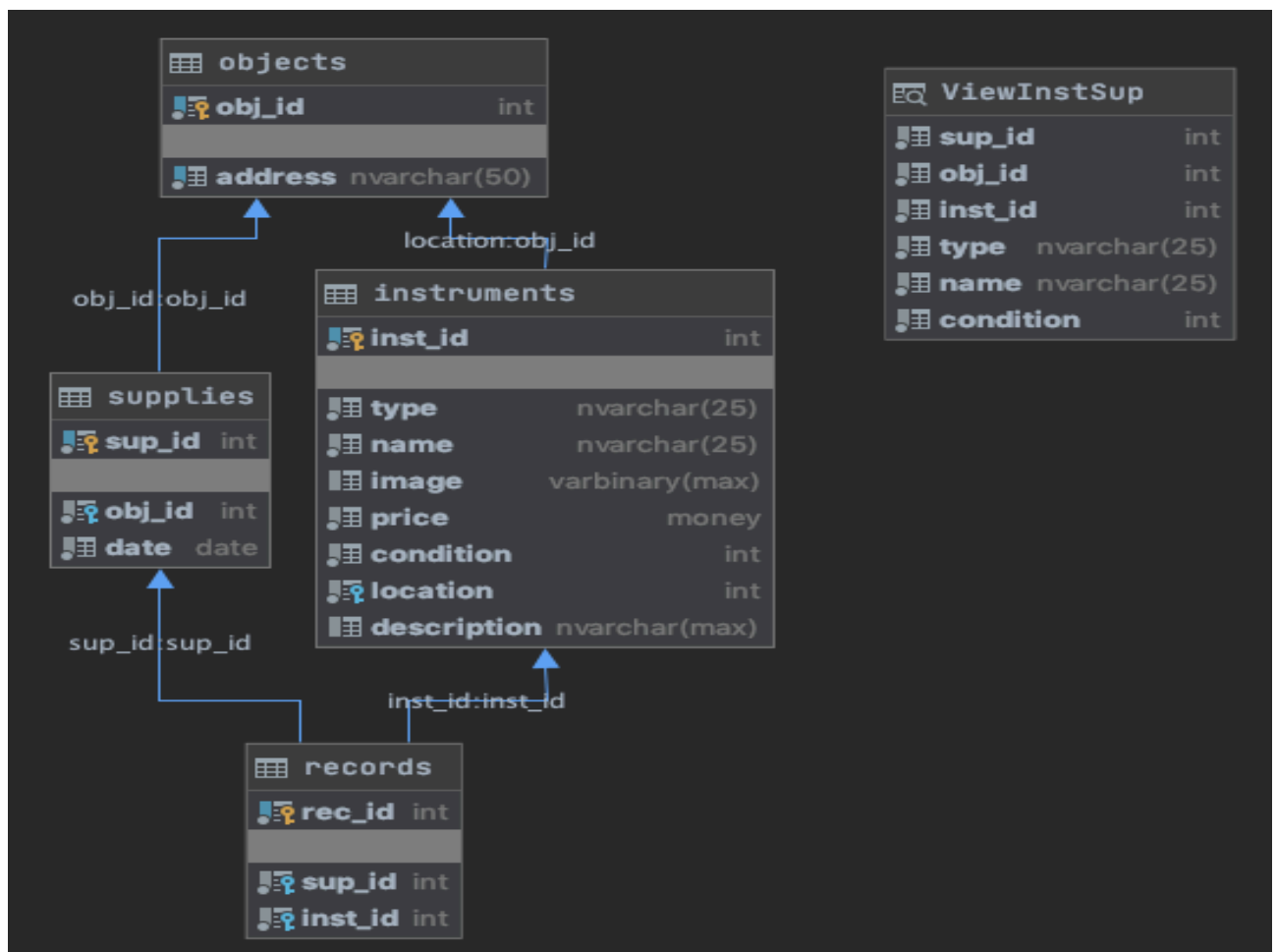


Рис. 2. Реляционная модель базы данных

Структура базы данных (рис. 2):

1. Сущность «Object» содержит в себе поля:
 - obj_id – идентификатор строительного объекта. Первичный ключ, генерируемый системой с помощью опции IDENTITY с шагом 1;
 - address - адрес строительного объекта. Уникальное поле.

2. Сущность «Instrument» содержит в себе поля:

- inst_id – идентификатор строительного инструмента. Первичный ключ, генерируемый системой с помощью опции IDENTITY с шагом 1;
- type - название типа инструмента. Уникальное поле;
- name - название инструмента;
- image - картинка инструмента;
- price - цена инструмента;
- condition - состояние инструмента (в работе / сломан);
- location - место нахождения инструмента. Внешний ключ, ссылающийся на первичный ключ сущности «Object»;
- description - описание и характеристики инструмента.

3. Сущность «Supply» содержит в себе поля:

- sup_id – идентификатор поставки. Первичный ключ, генерируемый системой с помощью опции IDENTITY с шагом 1;
- obj_id - адрес, на который осуществляется поставка оборудования. Внешний ключ, ссылающийся на первичный ключ сущности «Object»;
- date - дата, когда была сделана поставка.

4. Сущность «Record» содержит в себе поля:

- rec_id – идентификатор записи в поставке. Первичный ключ, генерируемый системой с помощью опции IDENTITY с шагом 1;
- sup_id - поставка, в которой имеется данная запись. Внешний ключ, ссылающийся на первичный ключ сущности «Supply»;
- inst_id - инструмент, который содержится в поставке. Внешний ключ, ссылающийся на первичный ключ сущности «Instrument».

Связь сущностей «Instrument» и «Object» означает, что соответствующий инструмент находится на соответствующем объекте. То есть таблица объектов является главной таблицей относительно таблицы инструментов. Связь один ко

многим подразумевает, что на объекте может находиться много инструментов, но инструмент может находиться только на одном объекте.

Связь сущностей «Supply» и «Object» означает, что соответствующая поставка осуществлена на соответствующий строительный объект. То есть таблица объектов является главной таблицей относительно таблицы поставок. Связь один ко многим подразумевает, что на объект может осуществляться много поставок, но поставка может быть сделана только на один объект.

Связь сущностей «Record», «Supply» и «Instrument» означает запись соответствующего инструмента в соответствующую поставку. То есть таблицы инструментов и поставок являются главными таблицами относительно таблицы записей инструментов в поставку. Связь один ко многим подразумевает, что на поставка может содержать в себе много записей инструментов, но запись относится только к одной поставке. Аналогична связь с инструментом.

Ниже приведен текст скрипт файла, содержащего код для создания таблиц в базе данных.

Создание таблицы «Objects»:

```
create table objects(  
    obj_id int PRIMARY KEY IDENTITY NOT NULL,  
    address NVARCHAR(50) NOT NULL unique  
);
```

Создание таблицы «Instruments»:

```
create table instruments(  
    inst_id int PRIMARY KEY IDENTITY NOT NULL,  
    type NVARCHAR(50) NOT NULL unique,  
    name NVARCHAR(50) NOT NULL,  
    image VARBINARY(MAX), -- поле BLOB  
    price MONEY NOT NULL CHECK (price >= 0),  
    condition int not null default 1, -- 1-рабочее  
    состояние; 0-инструмент сломан  
    location int not null default 1, -- 1-это id  
    склада стоительной фирмы  
    FOREIGN KEY (location) REFERENCES objects  
    (obj_id) ON DELETE NO ACTION,  
    description nvarchar(max) null -- поле CLOB  
);
```

Создание таблицы «Supplies»:

```
create table supplies(  
    sup_id int PRIMARY KEY IDENTITY NOT NULL,  
    obj_id int not null,  
    FOREIGN KEY (obj_id) REFERENCES objects(obj_id) ON  
DELETE CASCADE ,  
    date DATE NOT NULL DEFAULT getdate()  
);
```

Создание таблицы «Records»:

```
create table records(  
    rec_id int PRIMARY KEY IDENTITY NOT NULL,  
    sup_id int not null,  
    FOREIGN KEY (sup_id) REFERENCES supplies(sup_id) ON  
DELETE CASCADE ,  
    inst_id int not null,  
    FOREIGN KEY (inst_id) REFERENCES instruments(inst_id)  
ON DELETE CASCADE  
);
```

Ссылочная целостность

Ссылочная целостность сохраняет определенные связи между таблицами при добавлении или удалении строк. Она основана на связи первичных и внешних ключей и обеспечивается с помощью ограничений FOREIGN KEY и CHECK. Ссылочная целостность гарантирует согласованность значений ключей во всех таблицах. Этот вид целостности требует отсутствия ссылок на несуществующие значения, а также обеспечивает согласованное изменение ссылок во всей базе данных при изменении значения ключа.

Обеспечение ссылочной целостности между таблицами «Objects» и «Instruments»:

- в таблицу «Instruments» нельзя добавить инструмент, место нахождения которого нет в таблице «Objects». Условие «not null default 1» атрибута Instruments.location подразумевает то, что если новый инструмент куплен не на строительном объекте, атрибуту автоматически присваивается

значение 1, что соответствует id склада фирмы. Поэтому в таблице «Objects» обязательна запись адреса самого склада фирмы первым пунктом;

- обновление адреса объекта никак не повлияет на запись инструмента, а обновлять первичный суррогатный ключ MS SQL Server не позволяет;
- условие «FOREIGN KEY (location) REFERENCES objects (obj_id) ON DELETE NO ACTION» атрибута Instruments.location подразумевает, что если на объект ссылается какой-либо инструмент, то удаление записи не возможно, но если ссылок на этот объект нет, то удаление происходит. Логически это звучит так, когда все строительные работы на объекте завершены и все инструменты отправлены на склад, запись объекта можно удалять. Но если работы еще ведутся, то удаление не возможно.

Обеспечение ссылочной целостности между таблицами «Objects» и «Supplies»:

- в таблицу «Supply» нельзя добавить поставку, место нахождение которой нет в таблице «Objects»;
- обновление адреса объекта никак не повлияет на запись поставки, а обновлять первичный суррогатный ключ MS SQL Server не позволяет;
- условие «FOREIGN KEY (obj_id) REFERENCES objects (obj_id) ON DELETE CASCADE» атрибута Supply.obj_id подразумевает, что если на объект ссылается какая-либо поставка, то удаление записи объекта приводит к удалению всех поставок.

Обеспечение ссылочной целостности между таблицами «Instruments», «Supplies» и «Records»:

- в таблицу «Records» нельзя добавить запись инструмента в поставку, которых нет в таблицах «Instruments» и «Supplies»;
- обновления любых атрибутов поставки и инструмента никак не повлияют на сущность Record, а обновлять первичный суррогатный ключ MS SQL Server не позволяет;

- условие « FOREIGN KEY (sup_id) REFERENCES supplies(sup_id) ON DELETE CASCADE» атрибута Record.sup_id подразумевает, что если на поставку ссылается какая-либо «запись инструмента в поставке», то удаление записи поставки приводит к удалению всех соответствующих записей в таблице «Records»;
- условие «FOREIGN KEY (inst_id) REFERENCES instruments(inst_id) ON DELETE CASCADE» атрибута Record.inst_id подразумевает, что если на инструмент ссылается какая-либо «запись инструмента в поставке», то удаление записи инструмента приводит к удалению всех соответствующих записей в таблице «Records».

Оценка размера базы данных:

Оценки каждой из таблиц в базе данных представлены в таблицах 1 - 4.

Таблица 1. Описание таблицы “Objects”.

Имя столбца	Тип данных	Размер поля
obj_id	INT	4 Б
address	NVARCHAR(50)	125 Б
Максимальный размер одной записи		129 Б

Таблица 2. Описание таблицы “Instruments”.

Имя столбца	Тип данных	Размер поля
inst_id	INT	4 Б
type	NVARCHAR(50)	125 Б
name	NVARCHAR(50)	125 Б
image	VARBINARY(MAX)	2 ГБ
price	MONEY	8 Б
condition	INT	4 Б
location	INT	4 Б
description	NVARCHAR(MAX)	2 ГБ
Максимальный размер одной записи		4 ГБ

Таблица 3. Описание таблицы “Supplies”.

Имя столбца	Тип данных	Размер поля
sup_id	INT	4 Б
obj_id	INT	4 Б
date	DATE	8 Б
Максимальный размер одной записи		16 Б

Таблица 4. Описание таблицы “Records”.

Имя столбца	Тип данных	Размер поля
rec_id	INT	4 Б
sup_id	INT	4 Б
inst_id	INT	4 Б
Максимальный размер одной записи		12 Б

Так как разрабатываемая система носит учебный характер, база данных имеет следующий объем данных: 9 строительных объектов, 16 строительных инструментов, 4 поставки и 8 записей в поставках.

Размер таблиц и отдельных записей в таблице можно просчитать следующими SQL-запросами:

```
EXEC sp_spaceused 'dbo.instruments';

select datalength(inst_id)
      + datalength(type)
      + datalength(name)
      + datalength(price)
      + datalength(image)
      + datalength(condition)
      + datalength(location)
      + datalength(description)
from instruments
where inst_id = 1;
```

Точные размеры таблиц составляют:

- «Objects» – 8 КБ;
- «Instruments» – 1400 КБ;
- «Supplies» – 8 КБ;
- «Records» – 8 КБ.

Хранимые процедура и функция

Так как хранимые процедуры и функции компилируются единожды и в дальнейшем хранятся на сервере, их выполнение никак не будет уменьшать производительность системы.

Хранимая процедура «ProcRunningOutInst» показывает инструменты в рабочем состоянии, которые подходят к концу на складе (меньше 2 экземпляров).

Создание хранимой процедуры:

```
CREATE PROCEDURE ProcRunningOutOfInst
AS
SELECT type, count(*)
FROM instruments
where location = 1 and condition = 1
group by type
having count(*) < 2;
```

Хранимая функция «FuncInstOnObj» показывает количество определенного типа инструмента на складе. На вход ей подается тип инструмента, на выходе получаем число.

Создание хранимой функции:

```
CREATE FUNCTION FuncInstOnObj(@n1 nvarchar(50))
RETURNS int
AS
BEGIN
Return (SELECT count(*)
FROM instruments
where location = 1 and type = @n1)
END
```

Представление

Представление « ViewInstSup» позволяет увидеть всю информацию о поставке.

```
CREATE VIEW ViewInstSup
AS
SELECT s.sup_id, s.obj_id, i.inst_id, i.type, i.name,
i.condition
FROM supplies s INNER JOIN instruments i ON s.obj_id =
i.location;
```

Триггер

Триггер является хранимой процедурой особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных (INSERT, UPDATE, DELETE).

Представленный ниже триггер «TrigInsertRecord» нужен для того, чтобы при создании поставки и добавлении в нее записей инструментов база автоматически меняла место положение инструмента, то есть изменение атрибута, показывающего на каком строительном объекте находится инструмент.

```
CREATE TRIGGER TrigInsertRecord ON records
AFTER INSERT
AS
UPDATE instruments SET location = s.obj_id
FROM inserted i INNER JOIN supplies s on i.sup_id =
s.sup_id where instruments.inst_id = i.inst_id;
```

Изоляция транзакций

Под «уровнем изоляции транзакций» понимается степень обеспечиваемой внутренними механизмами СУБД, то есть не требующей специального программирования, защиты от всех или некоторых видов несогласованности данных, возникающих при параллельном выполнении транзакций.

На сервере базы данных разрабатываемой системы уровень изоляции по умолчанию задан READ COMMITED. На этом уровне обеспечивается защита

от чернового, «грязного» чтения, тем не менее, в процессе работы одной транзакции другая может быть успешно завершена и сделанные ею изменения зафиксированы. В итоге первая транзакция будет работать с другим набором данных.

Уровни доступа к информации

Для демонстрации различного рода привилегий были созданы три новых пользователя:

- admin1 и admin2 - предоставлены все разрешения на объекты базы данных;
- user1 - имеет ограниченный доступ к информации (возможность только просматривать информацию).

(Пароль у всех один - Password123).

3. Разработка приложения-сервер

В качестве языка реализации был выбран язык программирования Java. Среда программирования, в которой разрабатывалось приложение, называется IntelliJ IDEA. Благодаря ей можно удобно совмещать проектирование и реализацию сервера с работой с базой данных. Также благодаря фреймворку Apache Maven Builder Manager[7] можно без всяких трудностей подключать нужные зависимости и библиотеки в свой проект. Так были подключены такие библиотеки как «`javaxee-web-api`» и «`javax.servlet-api`», включающие в себя классы для работы с HTTP запросами и JSP-страницами, позволяющие создавать содержимое, которое имеет как статические, так и динамические компоненты. Также необходимо было скачать библиотеку «`mssql-jdbc`» и драйвер JDBC для подключения и дальнейшей работы с сервером базы данных, оговоренной в предыдущем разделе. Готовое приложение нужно будет разместить на контейнере сервлетов Apache Tomcat, который занимается системной поддержкой сервлетов и обеспечивает их жизненный цикл в соответствии с правилами, определёнными в спецификациях. Может работать как полноценный самостоятельный веб-сервер. Обеспечивает обмен данными между сервлетом и клиентами, берёт на себя выполнение таких функций, как создание программной среды для функционирующего сервлета, идентификацию и авторизацию клиентов, организацию сессии для каждого из них.

Все перечисленные технологии являются платформенно-независимыми, поэтому разработанную систему можно устанавливать на различные операционные системы.

Если вспомнить структуру трехуровневой архитектуры, то сервер приложений - это средний слой, связующий слой, на нём сосредоточена большая часть бизнес-логики. Вне его остаются только клиентский интерфейс, а также элементы логики, погруженные в базу данных (хранимые процедуры и

триггеры). По сравнению с двухзвенной клиент-серверной архитектурой трёхуровневая архитектура обеспечивает, как правило, большую масштабируемость за счёт горизонтальной масштабируемости сервера приложений, большую конфигурируемость за счёт изолированности уровней друг от друга. При этом обычно разработка трёхзвенных программных комплексов сложнее, чем для двухзвенных, также наличие дополнительного связующего программного обеспечения может налагать дополнительные издержки в администрировании таких комплексов.

Структура приложения выглядит следующим образом:

I. Директория Java-классов

- Классы реализующие объекты базы данных
- Сервлет-классы

II. Веб-директория

- JSP-файлы

Организация подключения к СУБД

По сути JDBC - это Java API для выполнения SQL-запросов к базам данных. Так как Java является объектно-ориентированным языком, то под API (Application Programming Interface) подразумевается набор классов и интерфейсов. Следующие интерфейсы берут на себя основную задачу в работе с базой данных:

- `java.sql.DriverManager` обеспечивает загрузку драйверов и создание новых соединений с базой данных; это стержневой интерфейс JDBC, определяющий корректный выбор и инициализацию драйвера для данной СУБД в данных условиях;
- `java.sql.Connection` определяет характеристики и состояние соединения с БД; кроме того, он предоставляет средства для контроля транзакций и уровня их изолированности;
- `java.sql.Statement` выполняет функции контейнера по отношению к SQL-выражению; при этом под выражением понимается не только сам текст

запроса, но и такие характеристики, как параметры и состояние выражения. Предок `java.sql.PreparedStatement` работает с параметризованными SQL-выражениями;

- `java.sql.ResultSet` предоставляет доступ к набору строк, полученному в результате выполнения данного SQL-выражения.

Для доступа к базе данных необходимо получить объект `java.sql.Connection`. Сделать это можно, обратившись к уровню управления JDBC посредством вызова `java.sql.DriverManager.getConnection`. Основным параметром этого вызова является строка URL (Uniform Resource Locator), описывающая такие характеристики соединения, как субпротокол, порт машины, на которой запущен сервер базы данных и имя базы данных. После получения объекта соединения можно начать работу непосредственно с базой данных, обращаясь к методам `java.sql.Connection` для создания объектов `java.sql.Statement` (`java.sql.PreparedStatement`). JDBC предусматривает и возможность динамического подсоединения к базе данных. В этом случае можно произвести такие операции, как явная загрузка драйвера `Class.forName(«Driver»)`.

За непосредственное взаимодействие с базой данных отвечает статический Java класс «Database», который хранит в себе всю необходимую информацию для подключения к серверу базы данных:

- Driver - `com.microsoft.sqlserver.jdbc.SQLServerDriver`
- URL - `jdbc:sqlserver://[нужный порт];database=[название бд]`
- Username - [имя пользователя]
- Password - [пароль]

Ниже демонстрируется пример, использующий все описанные выше базовые интерфейсы, подключения к базе данных, отправление запроса с последующим принятием ответа, реализованный в коде программы.

```

try{

Class.forName(driver).getDeclaredConstructor().newInstance();
    try (Connection conn = DriverManager.getConnection(url,
user, password)){

        String sql = "Select dbo.FuncInstOnObj(?) as
[answer]";
        PreparedStatement preparedStatement =
conn.prepareStatement(sql);
        preparedStatement.setString(1, instType);
        ResultSet resultSet =
preparedStatement.executeQuery();
        if(resultSet.next()){
            answer = resultSet.getInt("answer");
        }

        resultSet.close();
        preparedStatement.close();
    }
}
catch(Exception ex){
    System.out.println(ex);
}

```

Технология Servlet / JSP

Сервлет является интерфейсом Java, реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ за частую это запросы GET, POST по протоколу HTTP/HTTPS. Принципиальное отличие между GET и POST в том, что GET-запросы предназначены только для получения данных с сервера, а POST-запросы несут с собой какую-то информацию, и данные на сервере могут измениться.

Работа сервера выглядит следующим образом :

Apache Tomcat сервер находится в режиме ожидания сообщений от клиента. Как только сообщение поступает, сервер анализирует команду и ищет подходящий сервлет, который бы обрабатывал запросы по такому адресу. В случае нахождения Tomcat создает объекта этого сервлета и запускает его в новом потоке, а сервер продолжает работать дальше в своем, принимать и

отправлять запросы. Кроме того, Tomcat создает ещё два объекта: `HttpServletRequest`, `HttpServletResponse`. В первом объекте содержатся все данные, что ему пришли в запросе от клиента. Ну и после всего этого передает два эти объекта в подходящий метод того сервлета, который запущен в отдельном потоке. Как только сервлет закончит работу и у него будет готов ответ, который надо отправить клиенту, Tomcat принимает этот ответ и отправляет его клиенту. Это позволяет Tomcat не отвлекаясь принимать запросы и отправлять ответы. А всю работу делают сервлеты, которые крутятся в отдельных потоках.

За отображение информации на экране клиента отвечает технология JSP. Страница JSP содержит текст двух типов: статические исходные данные и JSP-элементы, которые конструируют динамическое содержимое. Основным поставщиком информации от сервлета к jsp-странице и наоборот является упомянутый выше объект `HttpServletRequest`. На рис. 3 показано взаимодействие основных компонентов Web-приложения.

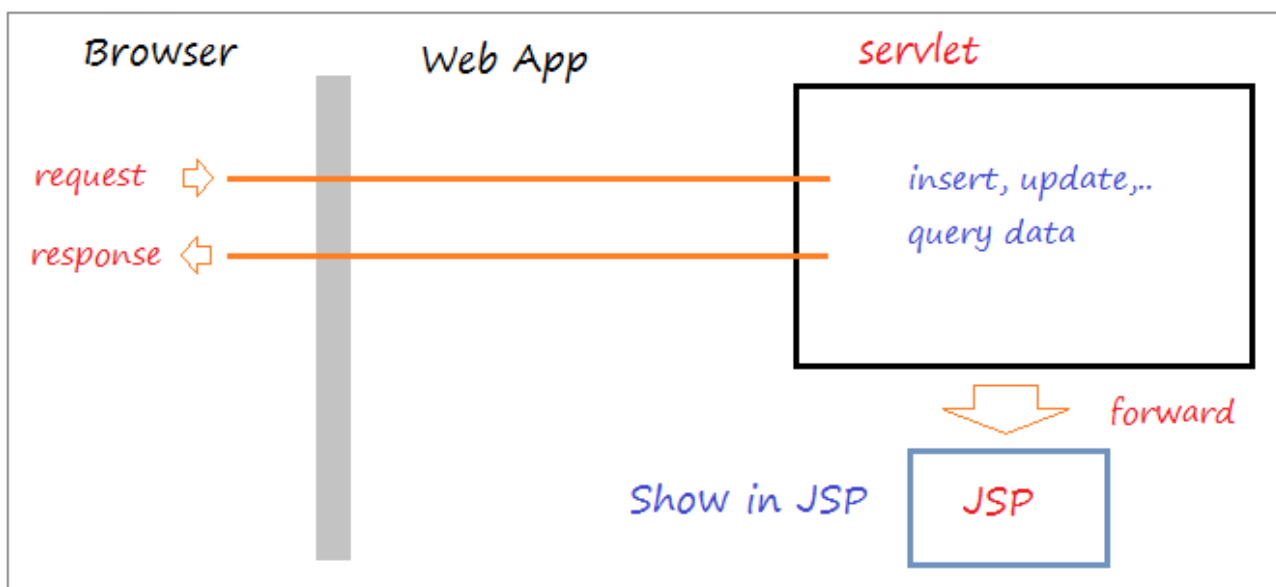


Рис. 3. Взаимодействие компонентов Web-приложения

В качестве примера ниже приведены кодовые реализации разных частей приложения-сервер отвечающих за создание нового строительного объекта.

Аннотация «@WebServlet(«/createObject»»)» указывает контейнеру сервлетов, что именно этот класс отвечает за запроса клиента на создание нового объекта. Метод doGet вызывает jsp-страницу «createObject.jsp», которая отображает на экране пользователя форму для заполнения необходимой информации об объекте. После заполнения формы клиентом и нажатия кнопки «Создать» на сервер отправляется POST запрос, который опять соответствует этому же сервлету, но уже методу doPost, который в свою очередь вынимает из объекта HttpServletRequest клиентские данные и передает их статическому классу Database в метод «insertObject». Класс в свою очередь подключается к базе данных, отправляет insert запрос и закрывает соединение. По той же идее реализовано все приложение-сервер.

```
@WebServlet("/createObject")
public class CreateObjectServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        getServletContext().getRequestDispatcher("/
createObject.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        try {
            request.setCharacterEncoding("UTF-8");
            String address = request.getParameter("address");
            if (!address.equals("")) {
                Database.insertObject(address);
                response.sendRedirect(request.getContextPath() + "/"
createObject.jsp");
            }
        }
        catch(Exception ex) {
            getServletContext().getRequestDispatcher("/
createObject.jsp").forward(request, response);
        }
    }
}
```

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
</head>
<body>
<br>
<h1>Новый Объект</h1>
<br>
<p><a href="objects">Назад</a></p>
<br>
<form action="createObject" method="post">
    <label>Адрес</label><br>
    <input name="address"/><br><br>
    <input type="submit" value="Добавить" />
</form>
</body>
</html>

```

```

Class.forName(driver).getDeclaredConstructor().newInstance();
try (Connection conn = DriverManager.getConnection(url, user, password)) {

    String sql = "INSERT INTO instruments (type, name, image, price,
description) Values (?, ?, ?, ?, ?)";
    try(PreparedStatement preparedStatement = conn.prepareStatement(sql)){
        preparedStatement.setString(1, instrument.getInstType());
        preparedStatement.setString(2, instrument.getInstName());
        preparedStatement.setBinaryStream(3, inputStream);
        preparedStatement.setInt(4, instrument.getInstPrice());
        preparedStatement.setString(5, instrument.getInstDescription());

        preparedStatement.executeUpdate();

    }
}

```

Сервер приложений содержит функции, позволяющие принимать следующие запросы от клиента:

1. Добавление записи:

```

public static void insertObject(String address) {
    try{
        Class.forName(driver).getDeclaredConstructor().newInstance();
        try (Connection conn = DriverManager.getConnection(url, user,
password)){

            String sql = "INSERT INTO objects (address) Values (?)";
            try(PreparedStatement preparedStatement =
conn.prepareStatement(sql)){
                preparedStatement.setString(1, address);
                preparedStatement.executeUpdate();

            }

        }
    } catch (Exception ex) {
        System.out.println(ex);
    }
}

```

2. Обновление записи:

```
Class.forName(driver).getDeclaredConstructor().newInstance();
try (Connection conn = DriverManager.getConnection(url, user, password)){

    String sql = "UPDATE objects SET address = ? WHERE obj_id = ?";
    try(PreparedStatement preparedStatement = conn.prepareStatement(sql)){
        preparedStatement.setString(1, object.getObjAddress());
        preparedStatement.setInt(2, object.getObjId());

        preparedStatement.executeUpdate();

    }

}
```

3. Удаление записи:

```
Class.forName(driver).getDeclaredConstructor().newInstance();
try (Connection conn = DriverManager.getConnection(url, user, password)){

    String sql = "DELETE FROM objects WHERE obj_id = ?";
    try(PreparedStatement preparedStatement = conn.prepareStatement(sql)){
        preparedStatement.setInt(1, id);
        preparedStatement.executeUpdate();

    }

}
```

4. Выборка данных:

```
ArrayList<Object> objectList = new ArrayList<>();
try{
    Class.forName(driver).getDeclaredConstructor().newInstance();
    try (Connection conn = DriverManager.getConnection(url, user,
password)){

        Statement statement = conn.createStatement();
        ResultSet resultSet = statement.executeQuery("SELECT * FROM
objects");
        while(resultSet.next()){

            int id = resultSet.getInt(1);
            String address = resultSet.getString(2);
            Object object = new Object(id, address);
            objectList.add(object);

        }

        statement.close();
        resultSet.close();

    }

}
```

5. Вызов представления:

```
ArrayList<Instrument> instrumentList = new ArrayList<>();
try{
    Class.forName(driver).getDeclaredConstructor().newInstance();
    try (Connection conn = DriverManager.getConnection(url, user,
password)){

        String sql = "SELECT * FROM ViewInstSup where sup_id = ?";
        try(PreparedStatement preparedStatement =
conn.prepareStatement(sql)){
            preparedStatement.setInt(1, id);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()){
                int instId = resultSet.getInt(3);
                String instType = resultSet.getString(4);
                String instName = resultSet.getString(5);
                int instCondition = resultSet.getInt(6);

                Instrument instrument = new Instrument(instId,
instType, instName,
                    0, instCondition, 0, "");
                instrumentList.add(instrument);
            }

            resultSet.close();
        }
    }
}
```

6. Вызов хранимой процедуры:

```
ArrayList<ArrayList> res = new ArrayList<>();
ArrayList<String> instType = new ArrayList<>();
ArrayList<Integer> instAmount = new ArrayList<>();
try{
    Class.forName(driver).getDeclaredConstructor().newInstance();
    try (Connection conn = DriverManager.getConnection(url, user,
password)){

        Statement statement = conn.createStatement();
        ResultSet resultSet = statement.executeQuery("EXECUTE
ProcRunningOutOfInst");
        while(resultSet.next()){
            instType.add(resultSet.getString(1));
            instAmount.add(resultSet.getInt(2));
        }

        statement.close();
        resultSet.close();
    }
    res.add(instType);
    res.add(instAmount);
}
```

7. Вызов хранимой функции:

```
int answer = 0;
try{
    Class.forName(driver).getDeclaredConstructor().newInstance();
    try (Connection conn = DriverManager.getConnection(url, user,
password)){
        String sql = "Select dbo.FuncInstOnObj(?) as [answer]";
        PreparedStatement preparedStatement =
conn.prepareStatement(sql);
        preparedStatement.setString(1, instType);
        ResultSet resultSet = preparedStatement.executeQuery();
        if(resultSet.next()){
            answer = resultSet.getInt("answer");
        }

        resultSet.close();
        preparedStatement.close();
    }
}
```

8. Загрузка и выборка картинки:

```
Class.forName(driver).getDeclaredConstructor().newInstance();
try (Connection conn = DriverManager.getConnection(url, user,
password)){
    String sql = "update instruments set instruments.image = ? where
inst_id = ?";
    try (PreparedStatement preparedStatement =
conn.prepareStatement(sql)) {
        preparedStatement.setBinaryStream(1, inputStream);
        preparedStatement.setInt(2, instrument.getInstId());

        preparedStatement.executeUpdate();
    }
}

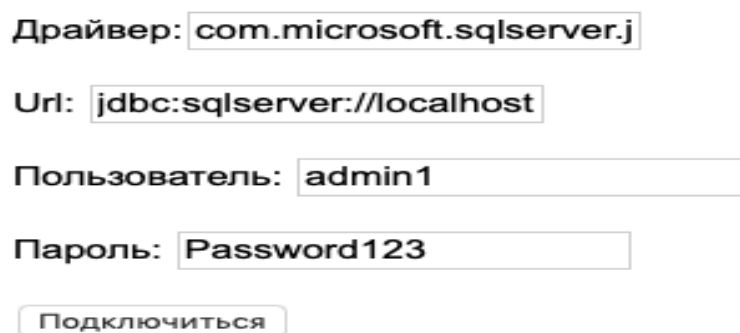
byte[] instId = new byte[0];
try{
    Class.forName(driver).getDeclaredConstructor().newInstance();
    try (Connection conn = DriverManager.getConnection(url, user,
password)){
        String sql = "SELECT image FROM instruments WHERE inst_id=?";
        try(PreparedStatement preparedStatement =
conn.prepareStatement(sql)){
            preparedStatement.setInt(1, id);
            ResultSet resultSet = preparedStatement.executeQuery();
            while(resultSet.next()){
                instId = resultSet.getBytes(1);
            }

            resultSet.close();
        }
    }
}
```


4. Интерфейс приложения на стороне клиента

Все что нужно сделать клиенту это перейти на установленный серверу приложений порт и прописать название приложения (например: `http://localhost:8080/ConstructionWarehouse`). Его сразу встретит страница с формой подключения к базе данных (рис. 4).

Данные сервера БД



Драйвер:

Url:

Пользователь:

Пароль:

Рис. 4. Страница Web-приложения для подключения к БД

В главном меню предлагается перейти на страницы посвященные выборке строительных объектов, инструментов, поставок и выполнению хранимых процедуры и функции. На каждой странице имеются кнопки для добавления, обновления и удаления записей. В таблице поставок по кнопке «Информация» вызывается представление и можно увидеть все инструменты в соответствующей поставке.

На рис. 5 приведена страница с просмотром всех строительных инструментов. Если кликнуть по пункту «Подробнее» откроется полная характеристика инструмента. Кликнув по кнопке «Картинка» можно увидеть иллюстрацию соответствующего инструмента.

На рис. 6 продемонстрирована работа хранимой функции. Из выпадающего списка нужно выбрать тип инструмента. В ответе получаем количество инструментов выбранного типа, имеющих на складе.

Список Инструментов

[Назад](#)

[Добавить](#)

ID	Тип	Название	Цена	Состояние	Описание	Показ картинки	Изменение	Удаление
1	Шуруповерт	Makita DF333DWYE	5500руб	В работе	► Подробнее			
2	Уровень	Level 2.0	500руб	В работе	► Подробнее			
3	Отбойный молоток	CTABP МОЭ-1800M	14000руб	В работе	► Подробнее			
4	Болгарка(УШМ)	Ryobi ONE+ R18AG-0	6000руб	В работе	► Подробнее			
5	Шуруповерт	Вихрь ПАБ540	5700руб	В работе	► Подробнее			
6	Рулетка	STANLEY 0-30-687	450руб	В работе	► Подробнее			
7	Циркулярная пила	HAMMER Flex CRP1300D	4550руб	В работе	► Подробнее			
8	Болгарка(УШМ)	Ryobi ONE+ R18AG-0	6000руб	В работе	► Подробнее			
9	Отбойный молоток	CTABP МОЭ-1800M	14000руб	В работе	► Подробнее			
10	Рулетка	STANLEY 0-30-687	450руб	В работе	► Подробнее			

Рис. 5. Страница Web-приложения для просмотра всех строительных инструментов фирмы

Функция

[Назад](#)

[Назад](#)

Выберите инструмент

- ✓ Шуруповерт
- Уровень
- Отбойный молоток
- Болгарка(УШМ)
- Шуруповерт
- Рулетка
- Циркулярная пила
- Болгарка(УШМ)
- Отбойный молоток

Инструмент	Ответ
Шуруповерт	2

Рис. 6. Страница Web-приложения хранимой функции

5. Комплект поставки и методика настройки разработанной системы

Требования к серверу СУБД

На сервере БД должен быть установлен Microsoft SQL Server 2008 R2 или более поздней версии. Для первоначальной загрузки данных пользователю должны быть предоставлены соответствующие права.

Требования к приложению-серверу

Для настройки подключения к базе данных должен быть установлен драйвер JDBC. Также должен быть установлен сервер приложений Apache Tomcat.

Требования к приложению-клиенту

Клиентский компьютер должен иметь доступ по сети к компьютеру, на котором установлено приложение-сервер, так как разработанная система является распределенной и работает по протоколу TCP.

Комплект поставки

В комплект поставки входят:

- war-файл ConstructionWarehouse.war с Web-приложением, который нужно развернуть на Apache Tomcat;
- файл-скрипт «initDB.sql» для создания элементов БД на сервере;

Методика настройки системы

Для установки Tomcat нужно:

- 1) Зайти на официальный сайт данного контейнера на страницу загрузок - <https://tomcat.apache.org/download-90.cgi> и скачать один из двух вариантов для Windows (32-bit Windows zip / 64-bit Windows zip).
- 2) После распаковки архива нужно установить конфигурацию Java_Home. Открыть файл startup.bat (с помощью текстового редактора) находящийся

в папке bin в Tomcat и добавить конфигурацию (свой путь к папке Java jdk).

```
# Vói Windows  
set JAVA_HOME=«C:\DevPrograms\Java\jdk1.8.0_144»
```

- 3) Открыть файл tomcat-users.xml (Находится в папке ApacheTomcat/conf) и в самом низу перед тегом </tomcat-users> прописать следующее.

```
<role rolename="manager-gui"/>  
<role rolename="manager-script"/>  
<role rolename="manager-jmx"/>  
<role rolename="manager-status"/>  
  
<user username="tomcat"  
      password="tomcat"  
      roles=«manager-gui,manager-script,manager-  
jmx,manager-status»/>
```

- 4) Чтобы запустить Tomcat, нажмите два раза на startup.bat в ApacheTomcat/bin. Сервер Tomcat запущен.
- 5) В браузере, перейдите по адресу: <http://localhost:8080/>. Далее нужно перейти на вкладку «Manager App», войти, используя установленный логин и пароль, спуститься в пункт «WAR файл для развёртывания», нажать кнопку «Выбрать файл», выбрать файл ConstructionWarehouse.war и нажать кнопку «Развернуть». Веб-приложение запущено на сервере Tomcat.

Для работы клиента требуется только браузер.

Заключение

В ходе данной курсовой работы была реализована система обеспечения доступа к реляционным данным, включающая приложение-клиента, сервер-приложений и сервер СУБД на основе трёхзвенной архитектуры клиент/сервер. Она позволяет облегчить работу с информацией, обеспечить безопасности данных, повысить качество (полноты, точности, достоверности, своевременности, согласованности) информации.

Исследованы технологии Servlet API и JSP.

Список литературы

- [1] Документация JavaSE, JavaEE [Электронный ресурс] // URL: <https://docs.oracle.com/en/java/>;
- [2] Документация JDBC API [Электронный ресурс] // URL: https://download.oracle.com/otn_hosted_doc/jdeveloper/904preview/jdk14doc/docs/guide/jdbc/index.html;
- [3] Документация Servlet API [Электронный ресурс] // URL: <http://tomcat.apache.org/tomcat-5.5-doc/servletapi/>;
- [4] Документация JSP API [Электронный ресурс] // URL: <http://tomcat.apache.org/tomcat-5.5-doc/jspapi/>;
- [5] Документация Apache Tomcat [Электронный ресурс] // URL: <https://tomcat.apache.org/tomcat-8.0-doc/>;
- [6] Документация Microsoft SQL Server [Электронный ресурс] // URL: <https://docs.microsoft.com/ru-ru/sql/?view=sql-server-ver15>;
- [7] Документация JDBC API [Электронный ресурс] // URL: <https://maven.apache.org/guides/>.