МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

База данных для сети книжных магазинов с применением пространственных типов данных.

Выполнил студент Мотолыга Д.И.

(Ф.И.О.)

Руководитель проекта Нистюк О. А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

Пояснительная записка курсового проекта содержит 39 страниц пояснительной записки, 20 иллюстраций, 5 источников литературы, 3 приложения.

MICROSOFT SQL SERVER, MICROSOFT SQL SERVER MANAGEMENT STUDIO.

Цель курсового проекта — разработка базы данных, которая обеспечит простое и удобное взаимодействие, что позволит пользователю регистрироваться и заходить в систему, оформлять заказы и просматривать их, находить ближайшие магазины.

В первой главе проводится аналитический обзор схожих приложений по тематике курсового проекта.

Вторая глава посвящена процессу проектирования системы и описание технологий, использованных во время выполнения проекта.

В третьей главе описано тестирование, проверка работоспособности и анализ данных системы.

В заключении приведены результаты проделанной работы.

Содержание

[1. Обзор технических решений и литературных источников 4](#_Toc58901340)

[2. Проектирование и разработка базы данных 5](#_Toc58901341)

[2.2. Разработка модели базы данных 5](#_Toc58901342)

[2.3. Хранимые процедуры 7](#_Toc58901343)

[2.3.1. Выборка данных из таблиц 8](#_Toc58901344)

[2.3.2. Регистрация 8](#_Toc58901345)

[2.3.3. Удаление данных из таблиц 9](#_Toc58901346)

[2.3.4. Добавление данных в таблицы 10](#_Toc58901347)

[2.3.5. Изменение данных в таблицах 10](#_Toc58901348)

[2.3.6. Экспорт и импорт таблиц в формат xml 10](#_Toc58901349)

[2.3.7. Получение ближайших магазинов рядом с пользователем 11](#_Toc58901350)

[2.3.8. Получение всех заказов одного пользователя 12](#_Toc58901351)

[2.3.9. Изменение статуса заказа 12](#_Toc58901352)

[2.3.10. Поиск ближайших магазинов рядом с юзером с конкретной книгой 13](#_Toc58901353)

[2.4. Реализация технологий 13](#_Toc58901354)

[2.4.1. Реализация технологии пространственного типа данных 13](#_Toc58901355)

[3. Тестирование, проверка работоспособности и анализ данных 15](#_Toc58901356)

[Заключение 18](#_Toc58901357)

[Список литературы 19](#_Toc58901358)

[Приложение А 20](#_Toc58901359)

[Приложение Б 20](#_Toc58901360)

[Приложение В 20](#_Toc58901361)

**Введение**

В данной записке приведено описание реляционной базы данных, разработанной в соответствии с заданием на курсовое проектирование по теме «База данных для сети книжных магазинов с применением пространственных типов данных» по дисциплине «Базы данных».

Задачей данной курсовой работы является разработка базы данных, предназначенной для безотказного последующего использования.

Функционально должны быть выполнены следующие задачи:

* Регистрация и авторизация пользователей.
* Просмотр корзины пользователя.
* Создание, удаление книжных магазинов.
* Поиск ближайшего магазина рядом с пользователем.

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Для разработки и управления базой данных курсового проекта использовалась система управления реляционными базами данных Microsoft SQL Server, в связи с ее простотой, производительностью и надежностью.

1. Обзор технических решений и литературных источников

В качестве примера по используемому функционалу я выбрал сайт oz.by, поскольку он максимально приближен к теме моего проекта и база данных будет проектироваться на основе функций данного приложения.

Основная идея книжного магазина – это предоставление сервиса по поиску необходимой книги и возможности приобретения.

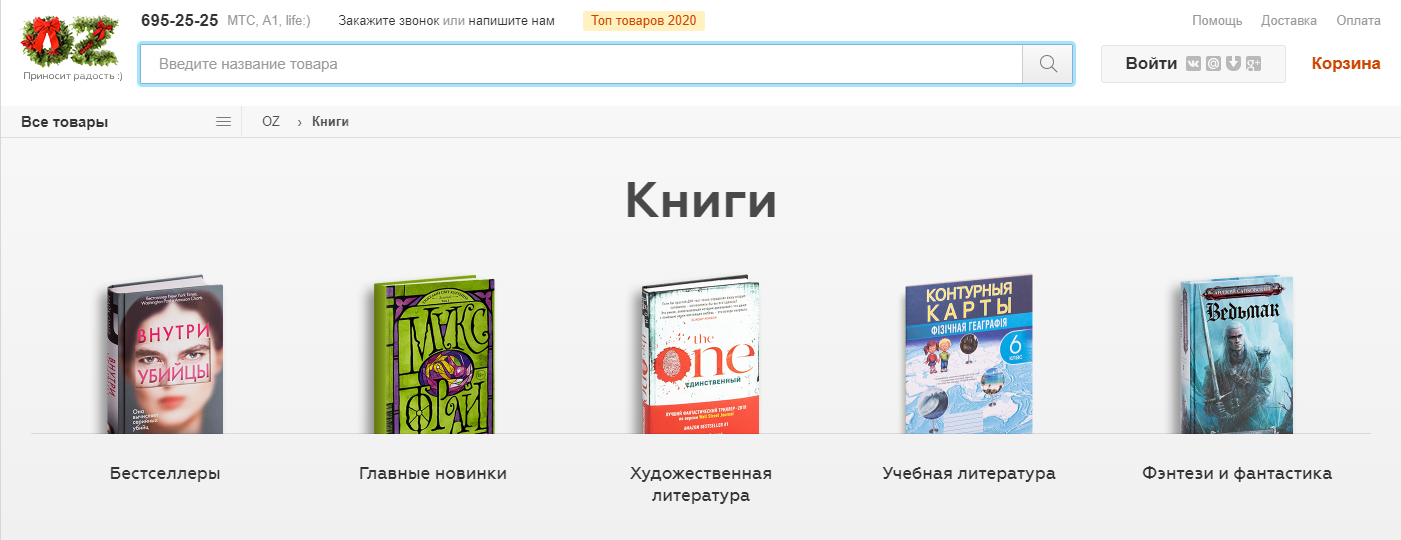


Рисунок 1.1 — Главная страница сайта oz.by

Проанализировав прототипы сайтов различных книжных магазинов, я выделил основные функции предоставляемых услуг клиентам:

* регистрация и авторизация клиентов;
* авторизация администратора;
* возможность поиска книг;
* возможность создавания заказа;
* возможность добавление книг.

Но не была найдена функция поиска ближайший магазинов с пользователем и было решено добавить эту функцию в проект, используя технологию пространственных типов Microsoft SQL Server.

1. Проектирование и разработка базы данных

Для хранения и предоставления данных пользователю была разработана архитектура базы данных, которая содержит в себе 20 таблиц. Диаграмма базы со структурой связей представлена в приложении А.

При разработке курсового проекта понадобились следующие объекты:

1. таблицы;
2. хранимые процедуры;
   1. Разработка модели базы данных

Для реализации базы данных для сети магазинов было разработано 20 таблиц. Диаграмма базы со структурой связей представлена в приложении А, а в приложении Б представлен скрипт создания БД.

Можно вывести 20 основных таблиц: User, Type, BookScore, StatusOrder, City, Location, Publish\_Office, Books\_Orders, Order, Book\_Store, Library, Libaries\_Books, Book, FullName, Envelope, Envelopes\_Books, Format, Publication\_Type, Authors\_Books, Author.

Таблица User представляет перечень театров, состоит из столбцов:

* ID\_User – идентификатор пользователя, тип int, первичный ключ, имеет связь 1-1 к таблице Location;
* ID\_Type – идентификатор типа пользователя, тип int, внешний ключ;
* ID\_FullName – идентификатор полного имени, тип int, внешний ключ;
* Password – пароль пользователя, тип nvarchar;
* mail – почта пользователя, тип nvarchar.

Таблица Type представляет перечень типов пользователя, состоит из столбцов:

* id\_Type – идентификатор типа, тип int, первичный ключ;
* Type – название типа, тип nvarchar.

Таблица BookScore представляет информацию о оценке книги пользователем, состоит из столбцов:

* ID\_Users – идентификатор пользователя, тип int, первичный ключ;
* ID\_Books – идентификатор книги, тип int;
* rating – рейтинг, тип int;
* comment – комментарий книге пользователем.

Таблица StatusOrder представляет перечень статусов заказа, состоит из столбцов:

* ID\_StatusOrder – идентификатор статуса, тип int, первичный ключ;
* name – имя статуса, тип nvarchar;

Таблица City представляет информацию о городах, состоит из столбцов:

* ID\_City – идентификатор города, тип int, первичный ключ;
* name – имя города, тип nvarchar,

Таблица Location представляет информацию о локации, состоит из столбцов:

* ID\_Location – идентификатор локации, тип int, первичный ключ;
* ID\_City– идентификатор города, тип int, внешний ключ;
* Street – название улицы, тип nvarchar;
* Flat – номер квартиры, тип int;
* House – номер дома, тип int;
* Postcode – почтовый индекс, тип int;
* Point – пространственный тип данных, показывающий локацию, тип geography.

Таблица Publish\_Office представляет информацию об издателях, состоит из столбцов:

* ID\_Publish\_Office – идентификатор издателя, тип int, первичный ключ;
* Name – имя издателя, тип nvarchar.

Таблица Books\_Orders представляет информацию об заказах конкретной книги, состоит из столбцов:

* ID\_Books – идентификатор книги, тип int, первичный ключ;
* ID\_Orders – идентификатор заказа, тип int, первичный ключ;
* Count – количество книг в заказе, тип int.

Таблица Order представляет информацию о заказе, состоит из столбцов:

* ID\_Order – идентификатор заказа, тип int, первичный ключ;
* ID\_StatusOrder – идентификатор статуса заказа, тип int, внешний ключ;
* ID\_User – идентификатор пользователя, тип int, внешний ключ;
* amount – общее количество книг в заказе, тип int.

Таблица Book\_Store представляет информацию о магазинах, состоит из столбцов:

* ID\_Book\_Store – идентификатор магазина, тип int, первичный ключ, связь 1-1 с таблицей Location;
* name – имя магазина, тип nvarchar.

Таблица Library представляет информацию о библиотеках, состоит из столбцов:

* ID\_Library – идентификатор библиотеки, тип int, первичный ключ, связь 1-1 с таблицей Book\_Store;
* amount – общее количество книг в библиотеке, тип int.

Таблица Libraries\_Books представляет информацию о книгах в библиотеках, состоит из столбцов:

* ID\_ Libraries – идентификатор библиотеки, тип int, первичный ключ;
* ID\_ Books – идентификатор книги, тип int, первичный ключ;
* amount – общее количество книги в библиотеке, тип int.

Таблица Book представляет информацию о книгах, состоит из столбцов:

* ID\_ Book – идентификатор книги, тип int, первичный ключ;
* Name – название книги, тип nvarchar;
* Description – описание книги, тип nvarchar;
* ID\_Publication\_Type – идентификатор типа публикации, тип int, внешний ключ;
* ID\_Publishing\_Office – идентификатор издателя, тип int, внешний ключ;
* Size – количество страниц в книге, тип int;
* ID\_Format – идентификатор формата книги, тип int;
* Weight – вес книги, тип int;
* Age\_Limit – возрастное ограничение, тип int.

Таблица FullName представляет информацию о полных именах, состоит из столбцов:

* ID\_FullName – идентификатор полного имени, тип int, первичный ключ;
* Surname – фамилия, тип nvarchar;
* Name – имя, тип nvarchar;
* Patronymic – отчество, тип nvarchar.

Таблица Envelope представляет информацию о обложках, состоит из столбцов:

* ID\_Envelope – идентификатор обложки, тип int, первичный ключ;
* Url – ссылка на обложку, тип nvarchar.

Таблица Envelopes\_Books представляет информацию о обложках книг, состоит из столбцов:

* ID\_Envelope – идентификатор обложки, тип int, первичный ключ;
* ID\_Books – идентификатор книги, тип int, первичный ключ.

Таблица Format представляет информацию о форматах, состоит из столбцов:

* ID\_Format – идентификатор формата, тип int, первичный ключ;
* Format – имя формата, тип nvarchar.

Таблица Publication\_Type представляет информацию о типах публикации, состоит из столбцов:

* ID\_Publication\_Type – идентификатор типа, тип int, первичный ключ;
* Type – имя типа, тип nvarchar.

Таблица Authors\_Books представляет информацию об авторов книг, состоит из столбцов:

* ID\_Author – идентификатор автора, тип int, первичный ключ;
* ID\_Books – идентификатор книги, тип int, первичный ключ.

Таблица Author представляет информацию об авторов книг, состоит из столбцов:

* ID\_Author – идентификатор автора, тип int, первичный ключ;
* ID\_FullName – идентификатор полного имени, тип int, внешний ключ.
  1. Хранимые процедуры

Хранимая процедура — объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

При разработке курсового проекта было создано 40 процедур для следующих целей:

1. Выборка данных из таблиц;
2. Поиск ближайших магазинов рядом с пользователем;
3. Удаление данных из таблиц;
4. Добавление данных в таблицы;
5. Изменение данных в таблицах;
6. Экспорт и импорт таблиц в формат xml;
7. Получение списка заказов.

Весь перечень созданных процедур и их код будет представлен в Приложении В.

* + 1. Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры: GetAllAuthors, GetAllBooks\_Orders, GetAllBook, GetAllBookInStorage, GetAllFullName, GetAllOrders, GetAllPictures, GetAllPublishOffice, GetAllPublishType, GetAllStores, GetAllUsers, GetAllCities, GetAllFormats.

* + 1. Регистрация

Для того, чтобы пользователь мог зарегистрироваться, была реализована процедура CreateUser, которая принимает в себя такие параметры как логин, имя, фамилия, отчество, пароль, номер, почта, идентификатор города, улица, квартира, дом, координаты для пространственной переменной. Процедура представлена на рисунке 2.3.2.1

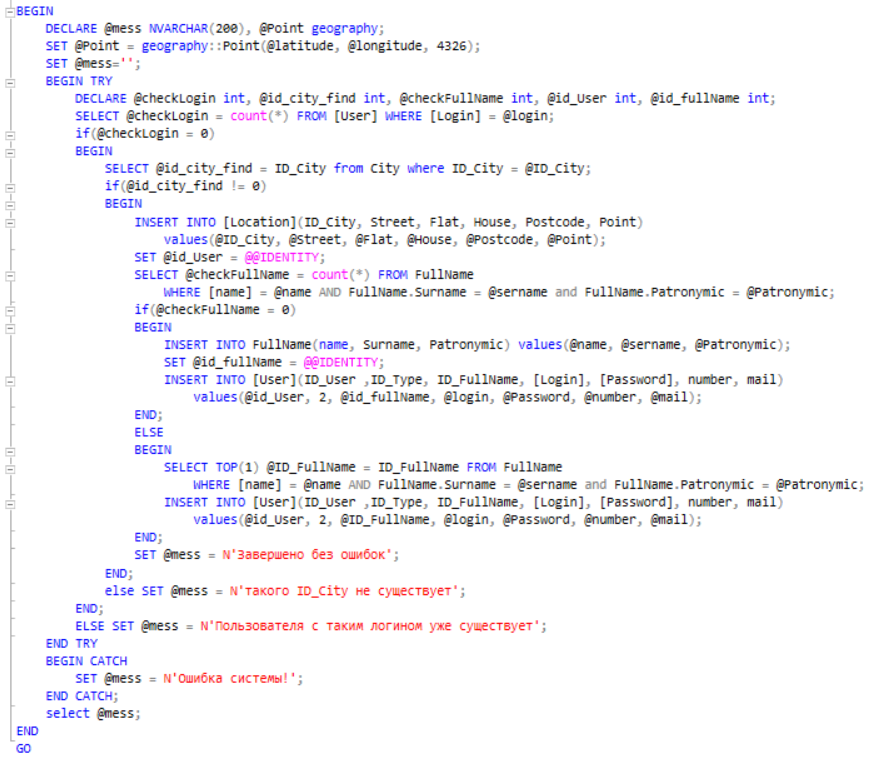
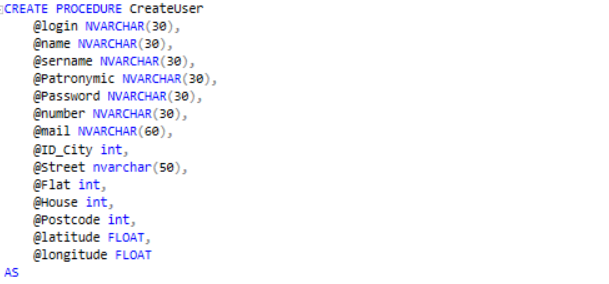


Рисунок 2.3.2.1 — процедура CreateUser

Так же в процедуре есть обработка ошибок при невалидных данных.

* + 1. Удаление данных из таблиц

Для управления базой данных были созданы процедуры удаления DeleteAuthor, DeleteBooksAuthor, DeleteEnvelopsAuthor, DeleteLibrariesBook, DeletePictures, DeleteStore, DeleteUser. Они были разработаны для удаления данных в соответствующих таблицах.

* + 1. Добавление данных в таблицы

Для управления базой данных были созданы процедуры добавления новых строк: AddAuthorToBook, AddBookToLibrary, AddBookToOrder, AddPictureToBook, CreateAuthor, CreateBook, CreateBookScore, CreateOrder, CreatePicture, CreateStore, CreateUser. Они были разработаны для заполнения соответствующих таблиц.

* + 1. Изменение данных в таблицах

Для управления базой данных были созданы процедуры обновления: UpdateMail, UpdateNumber, UpdatePassword. Они были разработаны для обновления данных в соответствующих таблицах.

* + 1. Экспорт и импорт таблиц в формат xml

Для экспорта таблиц в формате xml были разработаны процедуры ExportToFile и ExecExport. Для формирования xml в select запросе используется конструкция FOR XML. Для вывода в файл была использована расширенная хранимая процедура xp\_cmdshell и bcp операторы. Данная процедура была разработана с целью экспорта всех данных о ролях пользователей. Процедура экспорта представлена на рисунке 2.3.6.1.



Рисунок 2.3.6.1 — Пример создания процедур ExportToFile и ExecExpor

Для импорта данных в таблицы, из файла формата xml, была разработана процедура ParseBooks, которая импортирует данные о книгах. Для получения xml файла и последующего разбора со вставкой используется конструкция FROM OPENROWSET совместно с параметром BULK. Процедура импорта представлена на рисунке 2.3.6.2.



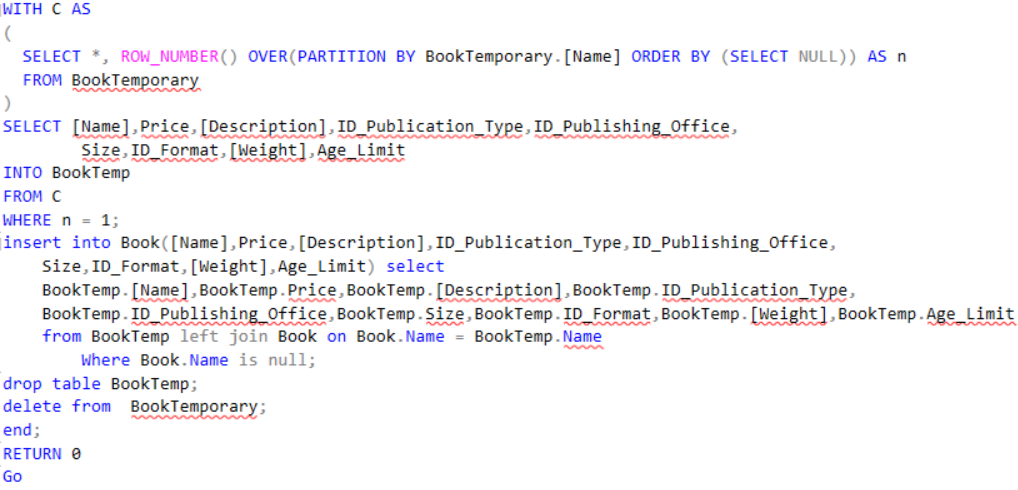


Рисунок 2.3.6.2 — Пример создания процедуры ParseBooks

* + 1. Получение ближайших магазинов рядом с пользователем

Для получения списка ближайших магазинов используется процедура FindNearestStore. Процедура принимает логин пользователя и возвращает три ближайших магазина к нему. Рисунок 2.3.7.1.

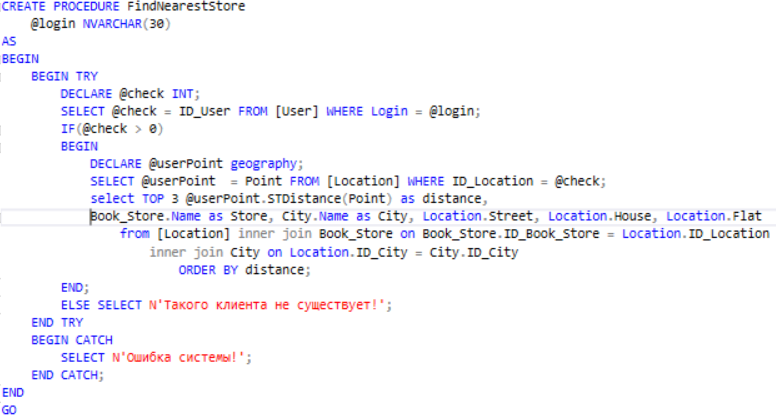


Рисунок 2.3.7.1 — Пример создания процедуры FindNearestStore

* + 1. Получение всех заказов одного пользователя

Для получения всех заказов одного пользователя реализована процедура GetAllUserOrders, которая принимает логин пользователя.

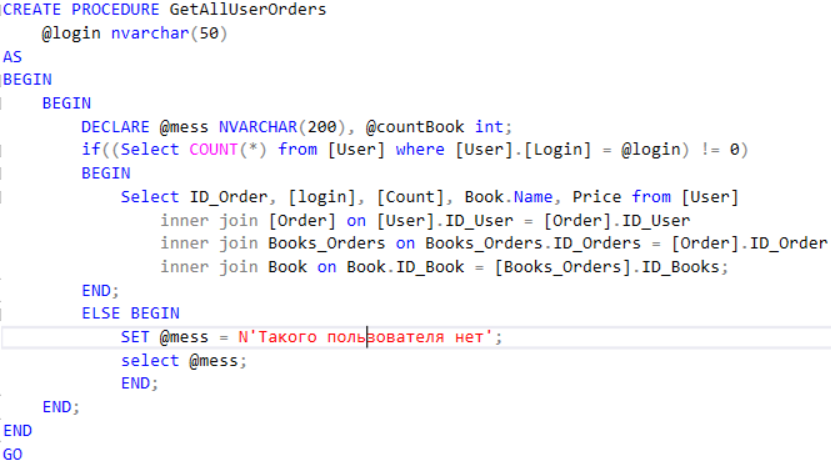
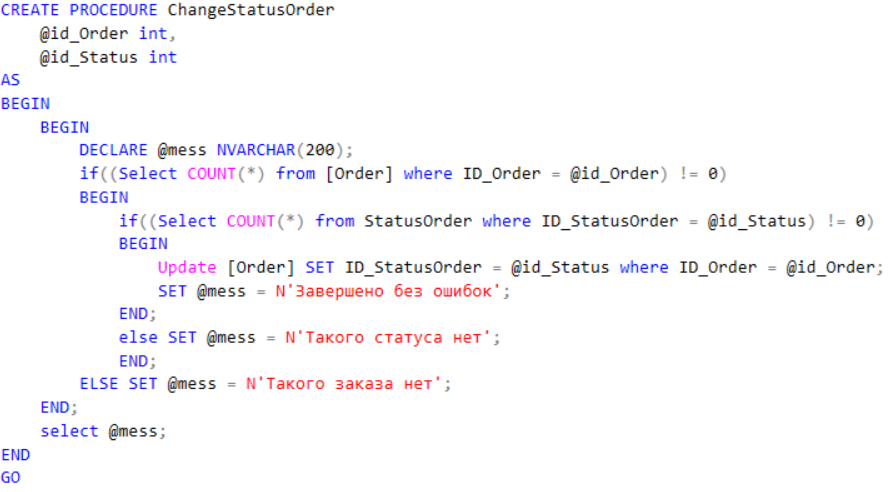


Рисунок 2.3.10.1 — Пример создания процедуры GetAllUserOrders

* + 1. Изменение статуса заказа

Для изменения статуса заказа используется процедура ChangeStatusOrder, которая принимает статус в качестве входного параметра. Рисунок 2.3.11.1.



* + 1. Поиск ближайших магазинов рядом с юзером с конкретной книгой

Для изменения статуса заказа используется процедура FindNearestStoreWithBook, которая принимает логин пользователя и книгу в качестве входных параметров. Рисунок 2.3.11.1.

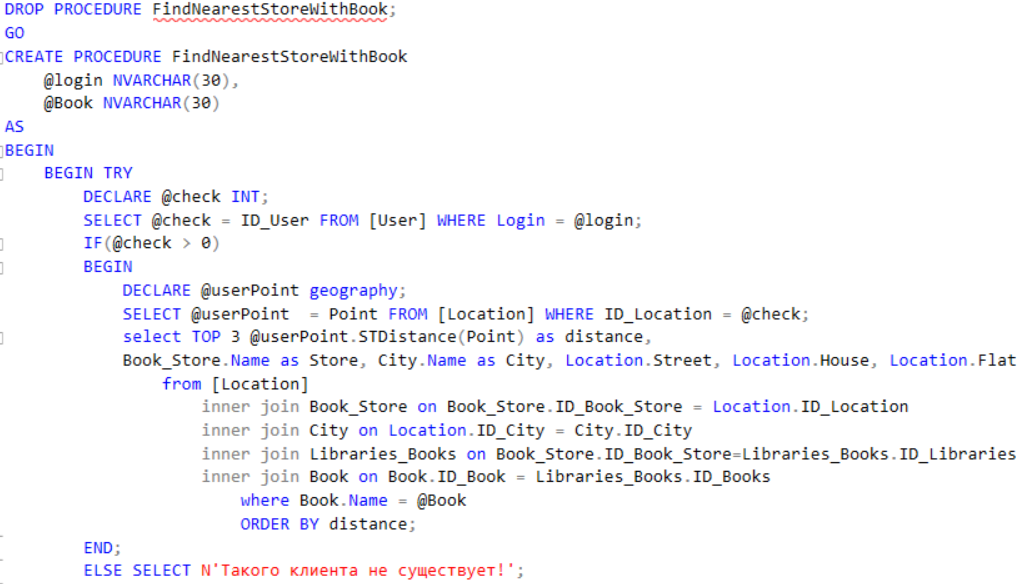


Рисунок 2.3.11.1 — Пример создания процедуры FindNearestStoreWithBook

* 1. Реализация технологий

В процессе разработки проекта были использована технология: пространственный тип данных.

* + 1. Реализация технологии пространственного типа данных

Пространственные данные представляют сведения о физическом расположении и форме геометрических объектов. Этими объектами могут быть точки расположения или более сложные объекты, такие как страны, дороги или озера.

SQL Server поддерживает два пространственных типа данных: geometry и geography .

Пространственный тип данных geometry представляет данные в евклидовой (плоской) системе координат.

Пространственный тип данных geography представляет данные в системе координат для сферической Земли.

В своем проекте я использовал тип geography, а именно подтип Point, для записи координат пользователей и магазинов.

Тип Point для типа данных geography представляет единичное расположение, где Lat представляет широту, а Long — долготу. Значения широты и долготы измеряются в градусах. Значения широты всегда находятся в интервале [-90, 90]. Все значения, находящиеся вне этого диапазона, вызывают исключение. Значения долготы всегда находятся в интервале [-180, 180]. Все значения, находящиеся вне этого диапазона, преобразуются в соответствующие значения в его пределах. Например, если введено значение долготы 190, то оно будет преобразовано в значение -170. SRID представляет идентификатор пространственной ссылки экземпляра geography , который необходимо вернуть.

В курсовом проекте использовались пространственные типы данных в процедурах FindNearestStore, FindNearestStoreWithBook, которые описаны в пунктах 2.3.7, 2.3.10 соответственно. Так же при использовании этого типа данных была использована функция STDistanse, которая находит расстояние между двумя точками в текущих настройках в миллиметрах.

1. Тестирование, проверка работоспособности и анализ данных

Разработанное программное средство полностью соответствует созданному для него техническому заданию, для проверки корректности работы программного средства, продемонстрируем добавление нового пользователя с помощью процедуры CreateAuthor на рисунке 3.1.

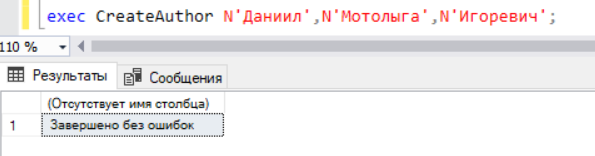


Рисунок 3.1 — Добавление автора

Используя процедуру GetAllFullName можно получить перечень всех полных имен. Рисунок 3.2.

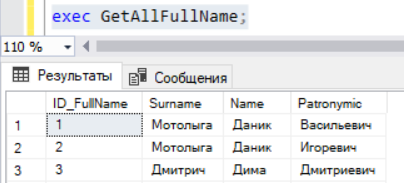


Рисунок 3.2 — Получение всех полных имен

Для удаления авторов используется процедура DeleteAuthor. Она принимает идентификатор автора. Её работа представлена на рисунке 3.3.

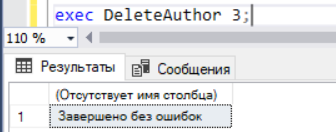


Рисунок 3.3 — Удаление автора

Для поиска ближайших магазинов рядом с пользователем процедура FindNearestStore, которая принимает логин и ищет 3 ближайших магазина. Рисунок 3.4.

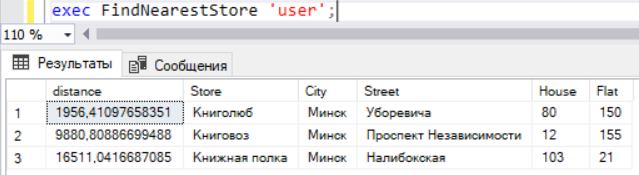


Рисунок 3.4 — Поиск ближайших магазинов (в миллиметрах)

Для получения ближайших магазинов рядом с юзером с наличием конкретной книги использует процедура FindNearestStoreWithBook. Она принимает логин пользователя и название книги. Рисунок 3.5.

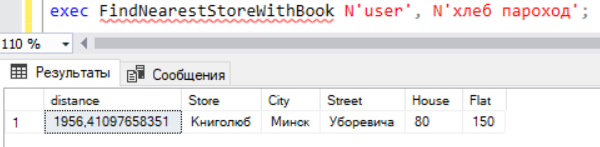


Рисунок 3.5 — Получение ближайших магазинов с книгой

Для экспорта данных из таблицы Book используется процедура ExecExport. Рисунок 3.6.

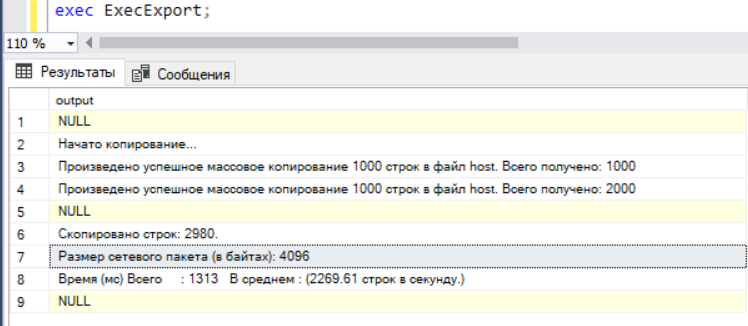


Рисунок 3.6 — Экспорт книг в xml файл

Для импорта книг из файла используется процедура ParseBooks. Рисунок 3.7.

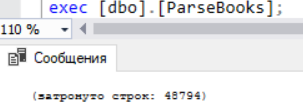


Рисунок 3.6 — Импорт книг из файла

Заключение

В процессе решения поставленной задачи была достигнута поставленная цель по созданию базы данных «сети книжных магазинов», которая предоставляет доступ к получению данных по магазинам, заказам и книгам, возможности изменять эти данные, а также реализованы процедуры поиска ближайших магазинов с использованием пространственного типа данных.

При разработке базы данных было созданы следующие объекты:

* 20 таблиц с внешними связями;
* 42 хранимыe процедур.

При разработке выполнены следующие пункты:

* Регистрация и авторизация пользователей;
* Просмотр корзины пользователя;
* Создание, удаление книжных магазинов;
* Поиск ближайшего магазина рядом с пользователем;
* Поиск ближайшего магазина рядом с пользователем и наличием конкретной книги в этом магазине;
* Добавление книг к существующему заказу;
* Добавление авторов книге;
* импорт и экспорт XML.

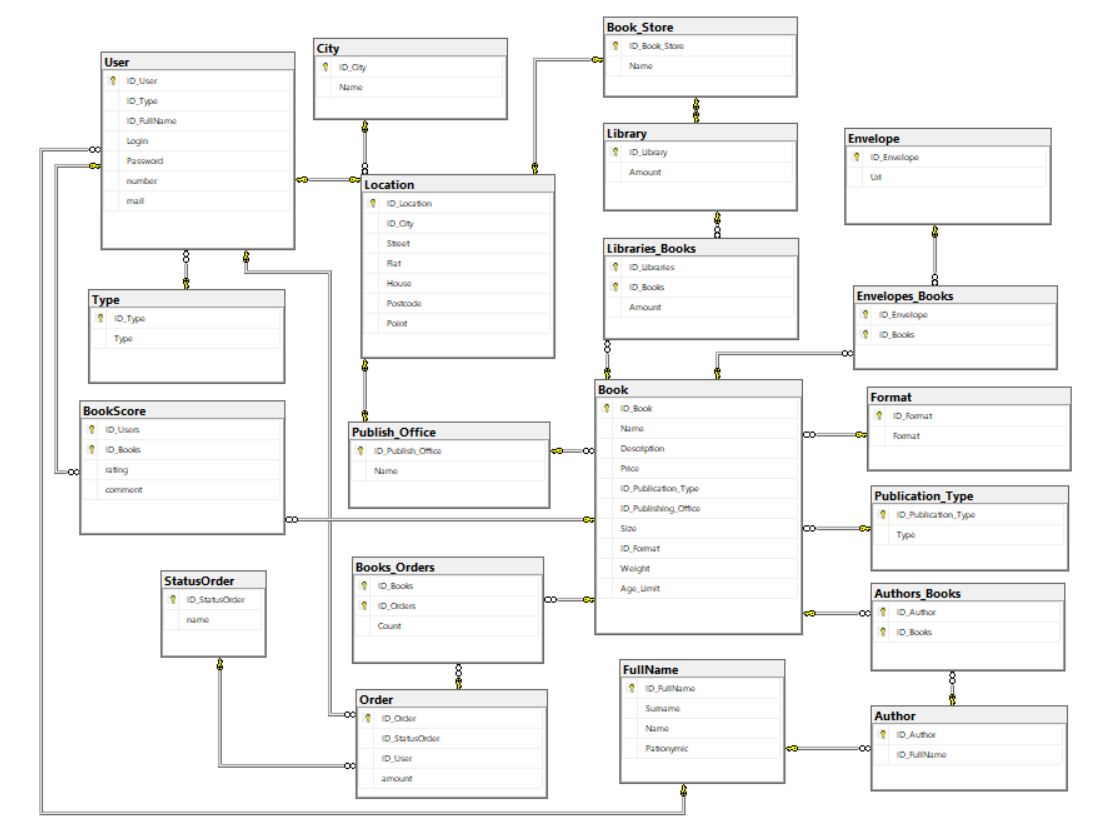
Приложение прошло тестирование при использовании в БД большого количество данных.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объёме.

Список литературы

1. OZ.BY: [Электронный ресурс] —  [https://oz.by/](https://www.teatr.by/) Дата доступа: 12.12.2020
2. Spartial-data: [Электронный ресурс] — https://docs.microsoft.com/ru-ru/sql/relational-databases/spatial/spatial-data-sql-server?view=sql-server-ver15 Дата доступа: 12.12.2020
3. Importing and Processing data from XML files into SQL Server tables: [Электронный ресурс] — https://www.mssqltips.com/sqlservertip/2899/importing-and-processing-data-from-xml-files-into-sql-server-tables/ Дата доступа: 12.12.2020
4. How to export XML using a SQL Server query?: [Электронный ресурс] — https://stackoverflow.com/questions/3952469/how-to-export-xml-using-a-sql-server-query Дата доступа: 12.12.2020
5. Что такое пространственные типы данных?: [Электронный ресурс] — <https://desktop.arcgis.com/ru/arcmap/10.3/manage-data/using-sql-with-gdbs/what-are-spatial-types.htm>l Дата доступа: 12.12.2020

Приложение А



Приложение Б

Create table City

(

ID\_City int identity(1,1) constraint City\_PK primary key,

Name nvarchar(50) UNIQUE not null ,

)

Create table Location

(

ID\_Location int identity(1,1) constraint Location\_PK primary key,

ID\_City int not null,

Street nvarchar(50) not null,

Flat int not null,

House int not null,

Postcode int not null,

Point geography not null,

FOREIGN KEY(ID\_City) REFERENCES [City](ID\_City)

)

Create table Book\_Store

(

ID\_Book\_Store int constraint Book\_Store\_PK primary key,

Name nvarchar(30) not null,

FOREIGN KEY(ID\_Book\_Store) REFERENCES [Location](ID\_Location)

)

Create table Library

(

ID\_Library int constraint Library\_PK primary key,

Amount int default(0)

FOREIGN KEY(ID\_Library) REFERENCES [Book\_Store](ID\_Book\_Store)

)

Create table Publication\_Type

(

ID\_Publication\_Type int identity(1,1) constraint Publication\_Type\_PK primary key,

Type nvarchar(50) UNIQUE not null

)

Create table Publish\_Office

(

ID\_Publish\_Office int constraint Publish\_Office\_PK primary key,

Name nvarchar(50) UNIQUE not null,

FOREIGN KEY(ID\_Publish\_Office) REFERENCES [Location](ID\_Location)

)

Create table [Format]

(

ID\_Format int identity(1,1) constraint Format\_PK primary key,

[Format] nvarchar(50) UNIQUE not null

)

Create table Book

(

ID\_Book int identity(1,1) constraint Book\_PK primary key,

Name nvarchar(50) not null,

Description nvarchar(500) not null,

Price int not null,

ID\_Publication\_Type int not null,

ID\_Publishing\_Office int not null,

Size int not null,

ID\_Format int not null,

Weight int not null,

Age\_Limit int not null,

FOREIGN KEY(ID\_Publication\_Type) REFERENCES [Publication\_Type](ID\_Publication\_Type),

FOREIGN KEY(ID\_Publishing\_Office) REFERENCES [Publish\_Office](ID\_Publish\_Office),

FOREIGN KEY(ID\_Format) REFERENCES [Format](ID\_Format)

)

Create table Libraries\_Books

(

ID\_Libraries int FOREIGN KEY REFERENCES Library(ID\_Library) not null ,

ID\_Books int FOREIGN KEY REFERENCES Book(ID\_Book) not null,

Amount int not null,

constraint Libraries\_Books\_PK primary key(ID\_Libraries,ID\_Books)

);

Create table FullName

(

ID\_FullName int identity(1,1) constraint FullName\_PK primary key,

Surname nvarchar(50) not null,

[Name] nvarchar(50) not null,

Patronymic nvarchar(50) not null

)

Create table Envelope

(

ID\_Envelope int identity(1,1) constraint Envelope\_PK primary key,

[Url] nvarchar(700) UNIQUE not null

)

Create table Author

(

ID\_Author int identity(1,1) constraint Author\_PK primary key,

ID\_FullName int FOREIGN KEY REFERENCES FullName(ID\_FullName) not null

)

Create table Authors\_Books

(

ID\_Author int FOREIGN KEY REFERENCES Author(ID\_Author) not null,

ID\_Books int FOREIGN KEY REFERENCES Book(ID\_Book) not null,

constraint Authors\_Books\_PK primary key(ID\_Author,ID\_Books)

)

Create table Envelopes\_Books

(

ID\_Envelope int FOREIGN KEY REFERENCES Envelope(ID\_Envelope) not null,

ID\_Books int FOREIGN KEY REFERENCES Book(ID\_Book) not null,

constraint Envelopes\_Books\_PK primary key(ID\_Envelope,ID\_Books)

)

Create table StatusOrder

(

ID\_StatusOrder int identity(1,1) constraint StatusOrder\_PK primary key,

[name] nvarchar(50) UNIQUE not null

)

Create table [Type]

(

ID\_Type int identity(1,1) constraint Type\_PK primary key,

[Type] nvarchar(30) UNIQUE not null

)

Create table [User]

(

ID\_User int constraint User\_PK primary key,

ID\_Type int not null,

ID\_FullName int not null,

[Login] nvarchar(30) UNIQUE not null,

[Password] nvarchar(30) not null,

[number] nvarchar(30) not null,

[mail] nvarchar(60) not null,

FOREIGN KEY(ID\_User) REFERENCES [Location](ID\_Location),

FOREIGN KEY(ID\_FullName) REFERENCES [FullName](ID\_FullName),

FOREIGN KEY(ID\_Type) REFERENCES [Type](ID\_Type)

)

Create table [Order]

(

ID\_Order int identity(1,1) constraint Order\_PK primary key,

ID\_StatusOrder int not null,

ID\_User int not null,

amount int not null,

FOREIGN KEY(ID\_StatusOrder) REFERENCES [StatusOrder](ID\_StatusOrder),

FOREIGN KEY(ID\_User) REFERENCES [User](ID\_User),

)

Create table Books\_Orders

(

ID\_Books int FOREIGN KEY REFERENCES Book(ID\_Book) not null,

ID\_Orders int FOREIGN KEY REFERENCES [Order](ID\_Order) not null,

[Count] int not null,

constraint Books\_Orders\_PK primary key(ID\_Orders,ID\_Books)

)

Create table BookScore

(

ID\_Users int FOREIGN KEY REFERENCES [User](ID\_User) not null,

ID\_Books int FOREIGN KEY REFERENCES Book(ID\_Book) not null,

rating int not null,

comment nvarchar(140) not null,

constraint BookScore\_PK primary key(ID\_Users,ID\_Books)

)

Приложение В

---------------------------------------------------------------------------

DROP PROCEDURE ParseBooks;

GO

CREATE PROCEDURE [dbo].[ParseBooks]

AS

Begin

declare @xml xml;

select @xml=(SELECT CONVERT(XML, BulkColumn) AS BulkColumn

FROM OPENROWSET(BULK 'd:\Study\JAVA\XMl\_Parser\other.xml', SINGLE\_BLOB) AS x);

insert into BookTemporary(

[Name],

Price,

[Description],

ID\_Publication\_Type,

ID\_Publishing\_Office,

Size,

ID\_Format,

[Weight],

Age\_Limit

)

select

P.K.value('Name[1]', 'nvarchar(MAX)') as [Name],

P.K.value('Price[1]', 'bigint') as Price,

P.K.value('Description[1]', 'nvarchar(MAX)') as [Description],

P.K.value('ID\_Publication\_Type[1]', 'bigint') as [ID\_Publication\_Type],

P.K.value('ID\_Publishing\_Office[1]', 'bigint') as [ID\_Publishing\_Office],

P.K.value('Size[1]', 'bigint') as [Size],

P.K.value('ID\_Format[1]', 'bigint') as [ID\_Format],

P.K.value('Weight[1]', 'bigint') as [Weight],

P.K.value('Age\_Limit[1]', 'bigint') as [Age\_Limit]

from @xml.nodes('/Root/Book') as P(K);

WITH C AS

(

SELECT \*, ROW\_NUMBER() OVER(PARTITION BY BookTemporary.[Name] ORDER BY (SELECT NULL)) AS n

FROM BookTemporary

)

SELECT [Name],Price,[Description],ID\_Publication\_Type,ID\_Publishing\_Office,

Size,ID\_Format,[Weight],Age\_Limit

INTO BookTemp

FROM C

WHERE n = 1;

insert into Book([Name],Price,[Description],ID\_Publication\_Type,ID\_Publishing\_Office,

Size,ID\_Format,[Weight],Age\_Limit) select

BookTemp.[Name],BookTemp.Price,BookTemp.[Description],BookTemp.ID\_Publication\_Type,

BookTemp.ID\_Publishing\_Office,BookTemp.Size,BookTemp.ID\_Format,BookTemp.[Weight],BookTemp.Age\_Limit

from BookTemp left join Book on Book.Name = BookTemp.Name

Where Book.Name is null;

drop table BookTemp;

delete from BookTemporary;

end;

RETURN 0

Go

---------------------------------------------------------------------------

--экспорт в файл

DROP PROCEDURE ExportToFile;

GO

CREATE PROCEDURE ExportToFile

AS

SELECT [Name],

Price,

[Description],

ID\_Publication\_Type,

ID\_Publishing\_Office,

Size,

ID\_Format,

[Weight],

Age\_Limit

FROM Book FOR XML PATH('Book'), ROOT('Root');

GO

---------------------------------------------------------------------------

--вызов процедуры экспорта в файл

DROP PROCEDURE ExecExport;

GO

CREATE PROCEDURE ExecExport

AS

DECLARE @cmd nvarchar(200);

SET @cmd = 'BCP "EXEC Book\_shops.dbo.ExportToFile" queryout "d:\data.xml" -S (LocalDB)\MSSQLLocalDB -w -C1251 -r -T';

EXEC master.dbo.xp\_cmdshell @cmd;

GO

---------------------------------------------------------------------------

-- ПОИСК БЛИЖАЙШЕГО МАГАЗИНА

DROP PROCEDURE FindNearestStore;

GO

CREATE PROCEDURE FindNearestStore

@login NVARCHAR(30)

AS

BEGIN

BEGIN TRY

DECLARE @check INT;

SELECT @check = ID\_User FROM [User] WHERE Login = @login;

IF(@check > 0)

BEGIN

DECLARE @userPoint geography;

SELECT @userPoint = Point FROM [Location] WHERE ID\_Location = @check;

select TOP 3 @userPoint.STDistance(Point) as distance,

Book\_Store.Name as Store, City.Name as City, Location.Street, Location.House, Location.Flat

from [Location] inner join Book\_Store on Book\_Store.ID\_Book\_Store = Location.ID\_Location

inner join City on Location.ID\_City = City.ID\_City

ORDER BY distance;

END;

ELSE SELECT N'Такого клиента не существует!';

END TRY

BEGIN CATCH

SELECT N'Ошибка системы!';

END CATCH;

END

GO

---------------------------------------------------------------------------

--создание user

DROP PROCEDURE CreateUser;

GO

CREATE PROCEDURE CreateUser

@login NVARCHAR(30),

@name NVARCHAR(30),

@sername NVARCHAR(30),

@Patronymic NVARCHAR(30),

@Password NVARCHAR(30),

@number NVARCHAR(30),

@mail NVARCHAR(60),

@ID\_City int,

@Street nvarchar(50),

@Flat int,

@House int,

@Postcode int,

@latitude FLOAT,

@longitude FLOAT

AS

BEGIN

DECLARE @mess NVARCHAR(200), @Point geography;

SET @Point = geography::Point(@latitude, @longitude, 4326);

SET @mess='';

BEGIN TRY

DECLARE @checkLogin int, @id\_city\_find int, @checkFullName int, @id\_User int, @id\_fullName int;

SELECT @checkLogin = count(\*) FROM [User] WHERE [Login] = @login;

if(@checkLogin = 0)

BEGIN

SELECT @id\_city\_find = ID\_City from City where ID\_City = @ID\_City;

if(@id\_city\_find != 0)

BEGIN

INSERT INTO [Location](ID\_City, Street, Flat, House, Postcode, Point)

values(@ID\_City, @Street, @Flat, @House, @Postcode, @Point);

SET @id\_User = @@IDENTITY;

SELECT @checkFullName = count(\*) FROM FullName

WHERE [name] = @name AND FullName.Surname = @sername and FullName.Patronymic = @Patronymic;

if(@checkFullName = 0)

BEGIN

INSERT INTO FullName(name, Surname, Patronymic) values(@name, @sername, @Patronymic);

SET @id\_fullName = @@IDENTITY;

INSERT INTO [User](ID\_User ,ID\_Type, ID\_FullName, [Login], [Password], number, mail)

values(@id\_User, 2, @id\_fullName, @login, @Password, @number, @mail);

END;

ELSE

BEGIN

SELECT TOP(1) @ID\_FullName = ID\_FullName FROM FullName

WHERE [name] = @name AND FullName.Surname = @sername and FullName.Patronymic = @Patronymic;

INSERT INTO [User](ID\_User ,ID\_Type, ID\_FullName, [Login], [Password], number, mail)

values(@id\_User, 2, @ID\_FullName, @login, @Password, @number, @mail);

END;

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'такого ID\_City не существует';

END;

ELSE SET @mess = N'Пользователя с таким логином уже существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--изменение юзера

DROP PROCEDURE UpdatePassword;

GO

CREATE PROCEDURE UpdatePassword

@login NVARCHAR(30),

@Password NVARCHAR(30)

AS

BEGIN

DECLARE @mess NVARCHAR(200);

SET @mess='';

BEGIN TRY

DECLARE @checkLogin int, @id\_city\_find int, @checkFullName int, @id\_User int, @id\_fullName int;

SELECT @checkLogin = count(\*) FROM [User] WHERE [Login] = @login;

if(@checkLogin != 0)

BEGIN

update [User] set [Password] = @Password where [Login] = @login;

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'Пользователя с таким логином не существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--изменение мыла

DROP PROCEDURE UpdateMail;

GO

CREATE PROCEDURE UpdateMail

@login NVARCHAR(30),

@mail NVARCHAR(100)

AS

BEGIN

DECLARE @mess NVARCHAR(200);

SET @mess='';

BEGIN TRY

DECLARE @checkLogin int, @id\_city\_find int, @checkFullName int, @id\_User int, @id\_fullName int;

SELECT @checkLogin = count(\*) FROM [User] WHERE [Login] = @login;

if(@checkLogin != 0)

BEGIN

update [User] set [mail] = @mail where [Login] = @login;

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'Пользователя с таким логином не существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--изменение номера телефона

DROP PROCEDURE UpdateNumber;

GO

CREATE PROCEDURE UpdateNumber

@login NVARCHAR(30),

@number NVARCHAR(100)

AS

BEGIN

DECLARE @mess NVARCHAR(200);

SET @mess='';

BEGIN TRY

DECLARE @checkLogin int, @id\_city\_find int, @checkFullName int, @id\_User int, @id\_fullName int;

SELECT @checkLogin = count(\*) FROM [User] WHERE [Login] = @login;

if(@checkLogin != 0)

BEGIN

update [User] set [number] = @number where [Login] = @login;

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'Пользователя с таким логином не существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--удаление юзера

DROP PROCEDURE DeleteUser;

GO

CREATE PROCEDURE DeleteUser

@login NVARCHAR(30)

AS

BEGIN

DECLARE @mess NVARCHAR(200);

SET @mess='';

BEGIN TRY

DECLARE @checkLogin int, @checkAdmin int, @Id\_user int, @Id\_users int;

SET @Id\_user = 0; SET @Id\_users = 0;

SELECT @checkLogin = count(\*) FROM [User] WHERE [Login] = @login;

SELECT top(1) @Id\_user = [User].ID\_User from [User] WHERE [Login] = @login;

if(@checkLogin != 0)

BEGIN

SELECT top(1) @checkAdmin = [User].ID\_Type FROM [User] WHERE [Login] = @login;

if(@checkAdmin = 2)

BEGIN

SELECT top(1) @Id\_users = BookScore.ID\_Users FROM BookScore WHERE ID\_Users = @Id\_user;

if(@Id\_users = 0)

BEGIN

SELECT top(1) @Id\_users = [Order].ID\_User FROM [Order] WHERE ID\_User = @Id\_user;

if(@Id\_users = 0)

BEGIN

delete [User] where [Login] = @login;

select @Id\_user;

delete [Location] where ID\_Location = @Id\_user;

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'у пользователя еще остались заказы';

END;

ELSE SET @mess = N'у пользователя есть оценки книг';

END;

else SET @mess = N'you have not permission';

END;

ELSE SET @mess = N'Пользователя с таким логином не существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--создание заказа

DROP PROCEDURE CreateOrder;

GO

CREATE PROCEDURE CreateOrder

@id\_book int,

@login NVARCHAR(30),

@count int

AS

BEGIN

DECLARE @mess NVARCHAR(200);

SET @mess='';

BEGIN TRY

DECLARE @checkLogin int, @checkAdmin int, @id\_Order int, @Id\_user int, @is\_exist\_book int;

SET @Id\_user = 0;

SELECT @checkLogin = count(\*) FROM [User] WHERE [Login] = @login;

SELECT top(1) @Id\_user = [User].ID\_User from [User] WHERE [Login] = @login;

if(@checkLogin != 0)

BEGIN

SELECT top(1) @is\_exist\_book = [Book].ID\_Book from [Book] WHERE [ID\_Book] = @id\_book;

if(@is\_exist\_book != 0)

BEGIN

if((SELECT top(1) [StatusOrder].ID\_StatusOrder from [StatusOrder] WHERE [ID\_StatusOrder] = 1) != 0)

begin

INSERT INTO [Order](ID\_User, ID\_StatusOrder, amount) values(@Id\_user, 1, @count);

SET @id\_Order = @@IDENTITY;

INSERT INTO [Books\_Orders](ID\_Books, ID\_Orders, [Count]) values(@id\_book, @id\_Order, @count);

SET @mess = N'Завершено без ошибок';

end;

else SET @mess = N'таблица типов не корректна';

END;

else SET @mess = N'указанной книги не существует';

END;

ELSE SET @mess = N'Пользователя с таким логином не существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--добавление книги к заказу

DROP PROCEDURE AddBookToOrder;

GO

CREATE PROCEDURE AddBookToOrder

@id\_book int,

@id\_order int

AS

BEGIN

DECLARE @mess NVARCHAR(200);

SET @mess='';

BEGIN TRY

if((SELECT top(1) [Order].ID\_Order from [Order] WHERE [ID\_Order] = @id\_order) != 0)

BEGIN

if((SELECT top(1) COUNT(\*) from [Books\_Orders] WHERE [ID\_Books] = @id\_book AND [ID\_Orders] = @id\_order) != 0)

BEGIN

update [Books\_Orders] set [Count] += 1 where [ID\_Books] = @id\_book AND [ID\_Orders] = @id\_order;

END;

else

BEGIN

INSERT INTO [Books\_Orders](ID\_Books, ID\_Orders, [Count]) values(@id\_book, @id\_Order, 1);

END;

update [Order] set [Order].[amount] += 1 where [Order].ID\_Order = @id\_order;

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'Такого заказа не существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--поставить рейтинг

DROP PROCEDURE CreateBookScore;

GO

CREATE PROCEDURE CreateBookScore

@id\_book int,

@id\_user int,

@rating int,

@comment nvarchar(140)

AS

BEGIN

DECLARE @mess NVARCHAR(200);

SET @mess='';

BEGIN TRY

if((SELECT COUNT(\*) from [BookScore] WHERE [ID\_Users] = @id\_user AND [ID\_Books] = @id\_book) = 0)

BEGIN

if((SELECT top(1) COUNT(\*) from [Book] WHERE [ID\_Book] = @id\_book) != 0)

BEGIN

if((SELECT top(1) COUNT(\*) from [User] WHERE [ID\_User] = @id\_user) != 0)

BEGIN

INSERT INTO BookScore(ID\_Books, ID\_Users, rating, comment) values(@id\_book, @id\_user, @rating, @comment);

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'Такого пользователя не существует';

END;

else

SET @mess = N'Такой книги не существует';

END;

ELSE SET @mess = N'вы уже поставили рейтинг этой книге';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--создание магазина

DROP PROCEDURE CreateStore;

GO

CREATE PROCEDURE CreateStore

@name NVARCHAR(30),

@ID\_City int,

@Street nvarchar(50),

@Flat int,

@House int,

@Postcode int,

@latitude FLOAT,

@longitude FLOAT

AS

BEGIN

DECLARE @mess NVARCHAR(200), @Point geography;

SET @Point = geography::Point(@latitude, @longitude, 4326);

SET @mess='';

BEGIN TRY

DECLARE @id\_Store int;

if((SELECT top(1) ID\_City from City where ID\_City = @ID\_City) != 0)

BEGIN

INSERT INTO [Location](ID\_City, Street, Flat, House, Postcode, Point)

values(@ID\_City, @Street, @Flat, @House, @Postcode, @Point);

SET @id\_Store = @@IDENTITY;

INSERT INTO Book\_Store(ID\_Book\_Store, Name) values(@id\_Store, @name);

INSERT INTO Library(ID\_Library, Amount) values(@id\_Store, 0);

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'такого ID\_City не существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--удаление магазина

DROP PROCEDURE DeleteStore;

GO

CREATE PROCEDURE DeleteStore

@ID\_Store int

AS

BEGIN

DECLARE @mess NVARCHAR(200), @Point geography;

SET @mess='';

BEGIN TRY

if((SELECT COUNT(\*) from Libraries\_Books where ID\_Libraries = @ID\_Store) = 0)

BEGIN

delete from Library where ID\_Library = @ID\_Store;

delete from Book\_Store where ID\_Book\_Store = @ID\_Store;

delete from Location where ID\_Location = @ID\_Store;

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'Нельзя удалить не пустом магазин';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--просмотр заказов пользователя

DROP PROCEDURE GetMyOrders;

GO

CREATE PROCEDURE GetMyOrders

@id\_user int

AS

BEGIN

DECLARE @mess NVARCHAR(200);

SET @mess='';

BEGIN TRY

if((SELECT COUNT(\*) from [User] WHERE [ID\_User] = @id\_user) != 0)

BEGIN

select [Book].[ID\_Book], [Book].[Name], [Books\_Orders].[Count] from [Order]

inner join [Books\_Orders] on [Order].ID\_Order = [Books\_Orders].ID\_Orders

inner join [Book] on [Books\_Orders].ID\_Books = [Book].ID\_Book

where ID\_User = @id\_user;

END;

ELSE SET @mess = N'Такого пользователя не существует';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

END

GO

---------------------------------------------------------------------------

--просмотр всех книг

DROP PROCEDURE GetAllBook;

GO

CREATE PROCEDURE GetAllBook

AS

BEGIN

select \* from [Book];

END

GO

---------------------------------------------------------------------------

--просмотр всех магазинов

DROP PROCEDURE GetAllStores;

GO

CREATE PROCEDURE GetAllStores

AS

BEGIN

select Book\_Store.ID\_Book\_Store, Book\_Store.Name, City.Name, Location.Street, Location.House, Location.Flat from [Book\_Store]

inner join [Location] on [Book\_Store].ID\_Book\_Store = [Location].ID\_Location

inner join [City] on [Location].ID\_City = [City].ID\_City;

END

GO

---------------------------------------------------------------------------

--просмотр всех заказов

DROP PROCEDURE GetAllOrders;

GO

CREATE PROCEDURE GetAllOrders

AS

BEGIN

select \* from [Order];

END

GO

---------------------------------------------------------------------------

--просмотр всех издателей

DROP PROCEDURE GetAllPublishOffice;

GO

CREATE PROCEDURE GetAllPublishOffice

AS

BEGIN

select ID\_Publish\_Office, Publish\_Office.Name, City.Name, Location.Street, Location.House, Location.Flat from [Publish\_Office]

inner join [Location] on [Publish\_Office].ID\_Publish\_Office = Location.ID\_Location

inner join [City] on [City].ID\_City = [Location].ID\_City;

END

GO

---------------------------------------------------------------------------

--создание автора

DROP PROCEDURE CreateAuthor;

GO

CREATE PROCEDURE CreateAuthor

@name NVARCHAR(30),

@sername NVARCHAR(30),

@Patronymic NVARCHAR(30)

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200), @checkFullName int;

SELECT @checkFullName = count(\*) FROM FullName

WHERE [name] = @name AND FullName.Surname = @sername and FullName.Patronymic = @Patronymic;

if(@checkFullName = 0)

BEGIN

INSERT INTO FullName(name, Surname, Patronymic) values(@name, @sername, @Patronymic);

INSERT INTO Author(ID\_FullName) values(@@IDENTITY);

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'автор с такими инициаломи уже существует';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--удаление автора

DROP PROCEDURE DeleteAuthor;

GO

CREATE PROCEDURE DeleteAuthor

@id\_author int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((Select COUNT(\*) from Authors\_Books where id\_Author = @id\_author) = 0)

BEGIN

delete from Author where ID\_Author = @id\_author;

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'у автора остались книги';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--получение авторов

DROP PROCEDURE GetAllAuthors;

GO

CREATE PROCEDURE GetAllAuthors

AS

BEGIN

BEGIN

Select Author.ID\_Author, FullName.Surname, FullName.Name, FullName.Patronymic from Author

inner join FullName on Author.ID\_FullName = FullName.ID\_FullName;

END;

END

GO

---------------------------------------------------------------------------

--добавление автора книге

DROP PROCEDURE AddAuthorToBook;

GO

CREATE PROCEDURE AddAuthorToBook

@id\_author int,

@id\_book int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((Select COUNT(\*) from Author where id\_Author = @id\_author) != 0)

BEGIN

if((Select COUNT(\*) from Book where ID\_Book = @id\_book) != 0)

BEGIN

INSERT INTO Authors\_Books(ID\_Author, ID\_Books) values(@id\_author, @id\_book);

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'Такой книги нет';

END;

ELSE SET @mess = N'Такого автора нет';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--удаление автора книге

DROP PROCEDURE DeleteBooksAuthor;

GO

CREATE PROCEDURE DeleteBooksAuthor

@id\_author int,

@id\_book int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((Select COUNT(\*) from Author where id\_Author = @id\_author) != 0)

BEGIN

if((Select COUNT(\*) from Book where ID\_Book = @id\_book) != 0)

BEGIN

delete from [Authors\_Books] where id\_Author = @id\_author AND ID\_Books = @id\_book;

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'Такой книги нет';

END;

ELSE SET @mess = N'Такого автора нет';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--изменение статуса заказа

DROP PROCEDURE ChangeStatusOrder;

GO

CREATE PROCEDURE ChangeStatusOrder

@id\_Order int,

@id\_Status int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((Select COUNT(\*) from [Order] where ID\_Order = @id\_Order) != 0)

BEGIN

if((Select COUNT(\*) from StatusOrder where ID\_StatusOrder = @id\_Status) != 0)

BEGIN

Update [Order] SET ID\_StatusOrder = @id\_Status where ID\_Order = @id\_Order;

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'Такого статуса нет';

END;

ELSE SET @mess = N'Такого заказа нет';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--создание обложки

DROP PROCEDURE CreatePicture;

GO

CREATE PROCEDURE CreatePicture

@url nvarchar(MAX)

AS

DECLARE @mess nvarchar(50);

BEGIN TRY

INSERT INTO Envelope(Url) values(@url);

SET @mess = N'Завершено без ошибок';

END TRY

BEGIN CATCH

SET @mess = N'Ошибка системы!';

END CATCH;

select @mess;

GO

---------------------------------------------------------------------------

--добавление обложки к книге

DROP PROCEDURE AddPictureToBook;

GO

CREATE PROCEDURE AddPictureToBook

@id\_envelope int,

@id\_book int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((Select COUNT(\*) from Envelope where ID\_Envelope = @id\_envelope) != 0)

BEGIN

if((Select COUNT(\*) from Book where ID\_Book = @id\_book) != 0)

BEGIN

INSERT INTO Envelopes\_Books(ID\_Envelope, ID\_Books) values(@id\_envelope, @id\_book);

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'Такой книги нет';

END;

ELSE SET @mess = N'Такой обложки нет';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--удаление обложки у книге

DROP PROCEDURE DeleteEnvelopsAuthor;

GO

CREATE PROCEDURE DeleteEnvelopsAuthor

@id\_envelope int,

@id\_book int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((Select COUNT(\*) from Envelope where id\_envelope = @id\_envelope) != 0)

BEGIN

if((Select COUNT(\*) from Book where ID\_Book = @id\_book) != 0)

BEGIN

delete from [Envelopes\_Books] where ID\_Envelope = @id\_envelope AND ID\_Books = @id\_book;

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'Такой книги нет';

END;

ELSE SET @mess = N'Такой обложки нет';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--удаление обложки

DROP PROCEDURE DeletePictures;

GO

CREATE PROCEDURE DeletePictures

@id\_envelope int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((Select COUNT(\*) from Envelopes\_Books where ID\_Envelope = @id\_envelope) = 0)

BEGIN

delete from Envelope where ID\_Envelope = @id\_envelope;

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'обложка используется в книгах';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--просмотр всех картинок(обложек)

DROP PROCEDURE GetAllPictures;

GO

CREATE PROCEDURE GetAllPictures

AS

BEGIN

select \* from Envelope;

END

GO

---------------------------------------------------------------------------

--создание книги

DROP PROCEDURE CreateBook;

GO

CREATE PROCEDURE CreateBook

@name nvarchar(50),

@Description nvarchar(500),

@Price int,

@ID\_Publication\_Type int,

@ID\_Publishing\_Office int,

@Size int,

@ID\_Format int,

@Weight int,

@Age\_Limit int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((SELECT count(\*) FROM [Format] WHERE ID\_Format = @ID\_Format) != 0)

BEGIN

if((SELECT count(\*) FROM [Publication\_Type] WHERE ID\_Publication\_Type = @ID\_Publication\_Type) != 0)

BEGIN

if((SELECT count(\*) FROM [Publish\_Office] WHERE ID\_Publish\_Office = @ID\_Publishing\_Office) != 0)

BEGIN

INSERT INTO Book(name, Description, Price, ID\_Publication\_Type, ID\_Publishing\_Office, size, ID\_Format, Weight, Age\_Limit)

values(@name, @Description, @price, @ID\_Publication\_Type, @ID\_Publishing\_Office, @Size, @ID\_Format, @Weight, @Age\_Limit);

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'неверный id издателя';

END;

ELSE SET @mess = N'неверный id типа публикации';

END;

ELSE SET @mess = N'неверный id формата';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--добавление книги в библиотеку

DROP PROCEDURE AddBookToLibrary;

GO

CREATE PROCEDURE AddBookToLibrary

@id\_lib int,

@id\_book int,

@amount int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200);

if((Select COUNT(\*) from [Library] where ID\_Library = @id\_lib) != 0)

BEGIN

if((Select COUNT(\*) from Book where ID\_Book = @id\_book) != 0)

BEGIN

UPDATE Library SET Amount+=@amount where ID\_Library = @id\_lib;

if((Select COUNT(\*) from Libraries\_Books where ID\_Books = @id\_book AND ID\_Libraries = @id\_lib) != 0)

BEGIN

UPDATE Libraries\_Books SET Amount+=@amount where ID\_Books = @id\_book AND ID\_Libraries = @id\_lib;

END;

ELSE BEGIN

INSERT INTO Libraries\_Books(ID\_Libraries, ID\_Books, Amount) values(@id\_lib, @id\_book, @amount);

END;

SET @mess = N'Завершено без ошибок';

END;

else SET @mess = N'Такой книги нет';

END;

ELSE SET @mess = N'Такой библиотеки нет';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--удаление книги из библиотеки

DROP PROCEDURE DeleteLibrariesBook;

GO

CREATE PROCEDURE DeleteLibrariesBook

@id\_lib int,

@id\_book int,

@count int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200), @countBook int;

if((Select COUNT(\*) from [Library] where ID\_Library = @id\_lib) != 0)

BEGIN

if((Select COUNT(\*) from Book where ID\_Book = @id\_book) != 0)

BEGIN

if((Select COUNT(\*) from Book where ID\_Book = @id\_book) != 0)

BEGIN

Select @countBook = Amount from Libraries\_Books where ID\_Libraries = @id\_lib AND ID\_Books = @id\_book

if(@countBook >= @count)

BEGIN

UPDATE Library SET Amount-= @count

where ID\_Library = @id\_lib;

if(@countBook = @count)

BEGIN

delete from [Libraries\_Books] where ID\_Libraries = @id\_lib AND ID\_Books = @id\_book;

END;

ELSE BEGIN

UPDATE [Libraries\_Books] SET Amount-= @count

where ID\_Books = @id\_book AND ID\_Libraries = @id\_lib;

END;

SET @mess = N'Завершено без ошибок';

END;

ELSE SET @mess = N'Нельзя удалить книг больше чем находится в магазине';

END;

END;

else SET @mess = N'Такой книги нет';

END;

ELSE SET @mess = N'Такой библиотеки нет';

END;

select @mess;

END

GO

---------------------------------------------------------------------------

--получение всех книг из библиотеки

DROP PROCEDURE GetAllBookInStorage;

GO

CREATE PROCEDURE GetAllBookInStorage

@id\_lib int

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200), @countBook int;

if((Select COUNT(\*) from [Library] where ID\_Library = @id\_lib) != 0)

BEGIN

Select Book\_Store.ID\_Book\_Store, Book\_Store.Name, Amount, Book.\*, City.Name as City, Location.Street, Location.House, Location.Flat from Libraries\_Books

inner join Book\_Store on ID\_Book\_Store = ID\_Libraries

inner join Location on ID\_Location = ID\_Book\_Store

inner join Book on ID\_Book = ID\_Books

inner join City on City.ID\_City = Location.ID\_City where Book\_Store.ID\_Book\_Store = @id\_lib;

END;

ELSE BEGIN

SET @mess = N'Такой библиотеки нет';

select @mess;

END;

END;

END

GO

---------------------------------------------------------------------------

--получение всех книг из библиотеки

DROP PROCEDURE GetAllUsers;

GO

CREATE PROCEDURE GetAllUsers

AS

BEGIN

select \* from [User]

inner join [Location] on [User].ID\_User = [Location].ID\_Location;

END

GO

---------------------------------------------------------------------------

--получение всех типов

DROP PROCEDURE GetAllPublishType;

GO

CREATE PROCEDURE GetAllPublishType

AS

BEGIN

select \* from [Publication\_Type];

END

GO

---------------------------------------------------------------------------

--получение FIO

DROP PROCEDURE GetAllFullName;

GO

CREATE PROCEDURE GetAllFullName

AS

BEGIN

select \* from [FullName];

END

GO

---------------------------------------------------------------------------

--получение Cities

DROP PROCEDURE GetAllCities;

GO

CREATE PROCEDURE GetAllCities

AS

BEGIN

select \* from [City];

END

GO

---------------------------------------------------------------------------

--получение Formats

DROP PROCEDURE GetAllFormats;

GO

CREATE PROCEDURE GetAllFormats

AS

BEGIN

select \* from [City];

END

GO

---------------------------------------------------------------------------

--получение Books\_Orders

DROP PROCEDURE GetAllBooks\_Orders;

GO

CREATE PROCEDURE GetAllBooks\_Orders

AS

BEGIN

select \* from [Books\_Orders];

END

GO

---------------------------------------------------------------------------

--получение всех заказов пользователя

DROP PROCEDURE GetAllUserOrders;

GO

CREATE PROCEDURE GetAllUserOrders

@login nvarchar(50)

AS

BEGIN

BEGIN

DECLARE @mess NVARCHAR(200), @countBook int;

if((Select COUNT(\*) from [User] where [User].[Login] = @login) != 0)

BEGIN

Select ID\_Order, [login], [Count], Book.Name, Price from [User]

inner join [Order] on [User].ID\_User = [Order].ID\_User

inner join Books\_Orders on Books\_Orders.ID\_Orders = [Order].ID\_Order

inner join Book on Book.ID\_Book = [Books\_Orders].ID\_Books;

END;

ELSE BEGIN

SET @mess = N'Такого пользователя нет';

select @mess;

END;

END;

END

GO

---------------------------------------------------------------------------

-- ПОИСК БЛИЖАЙШЕГО МАГАЗИНА

DROP PROCEDURE FindNearestStoreWithBook;

GO

CREATE PROCEDURE FindNearestStoreWithBook

@login NVARCHAR(30),

@Book NVARCHAR(30)

AS

BEGIN

BEGIN TRY

DECLARE @check INT;

SELECT @check = ID\_User FROM [User] WHERE Login = @login;

IF(@check > 0)

BEGIN

DECLARE @userPoint geography;

SELECT @userPoint = Point FROM [Location] WHERE ID\_Location = @check;

select TOP 3 @userPoint.STDistance(Point) as distance,

Book\_Store.Name as Store, City.Name as City, Location.Street, Location.House, Location.Flat

from [Location]

inner join Book\_Store on Book\_Store.ID\_Book\_Store = Location.ID\_Location

inner join City on Location.ID\_City = City.ID\_City

inner join Libraries\_Books on Book\_Store.ID\_Book\_Store=Libraries\_Books.ID\_Libraries

inner join Book on Book.ID\_Book = Libraries\_Books.ID\_Books

where Book.Name = @Book

ORDER BY distance;

END;

ELSE SELECT N'Такого клиента не существует!';

END TRY

BEGIN CATCH

SELECT N'Ошибка системы!';

END CATCH;

END

GO