МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Информационных систем и технологий
Специальность 1-40 05 01 «Информационные системы и технологи»
Специализация Издательско-полиграфический комплекс

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

Разработка базы данных с использованием мультимедийных типов данных для книжного магазина

Выполнил студент Булова Анна Фёдоровна
(Ф.И.О.)

Руководитель проекта асс. Жигаровская С.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Жигаровская С.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Жигаровская С.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2019

Содержание

Введение

Ценность информации в современном мире очень высока. Любая организация нуждается в своевременном доступе к информации. Роль распорядителей информации чаще всего выполняют базы данных.

База данных – совокупность взаимосвязанных, хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений.

Создание базы данных, ее поддержка и обеспечение доступа пользователей к ней осуществляется с помощью специального программного инструментария – системы управления базами данных.

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

Основные функции СУБД:

– Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки;

– Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация);

– Обеспечение логической и физической независимости данных;

– Защита логической целостности базы данных;

– Защита физической целостности;

– Управление полномочиями пользователей на доступ к базе данных;

– Синхронизация работы нескольких пользователей;

– Управление ресурсами среды хранения;

– Поддержка деятельности системного персонала.

– Обычно современная СУБД содержит следующие компоненты:

– Ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;

– Процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;

– Подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД.

СУБД существует огромное множество: Oracle, MS SQL Server, Microsoft Access, MySQL и так далее. В данной работе будет использовано решение MS SQL Server.

Постановка задачи

В наше время почти каждый человек активно использует базы данных в неявном виде.

Таким образом, базы данных ещё многие годы будет актуальным, т.к. оно позволяет пользователям быстро получать информацию из хранилищ без видимой реализации что имеет немаловажное значение.

Просмотр, добавление, удаление и обновление информации в процессе использования базы – ключевой фактор востребованности на рынке услуг за счёт быстрой смены данных и поставки обновлённых сведений пользователям.

Просмотр существующих товаров, таких как аудиозаписи, трейлеры фильмов, электронных книг позволяет пользователям обладать актуальной информацией об ассортименте книжного магазина, что, в свою очередь, выгодно выделяет приложение среди аналогов, так как не все конкуренты работают с всеми мультимедийными видами данных.

Регистрация и авторизация пользователей позволяет им воспользоваться возможностью отложенной покупки – функции корзины в режиме online упрощает выбора нескольких товаров.

Предусмотрена возможность поиска товаров согласно различным критериям.

Технология работы с мультимедийными данными позволяет администратору базы данных легко добавлять любое количества информации, не думая о форматах файлов данных.

1 Разработка модели базы данных

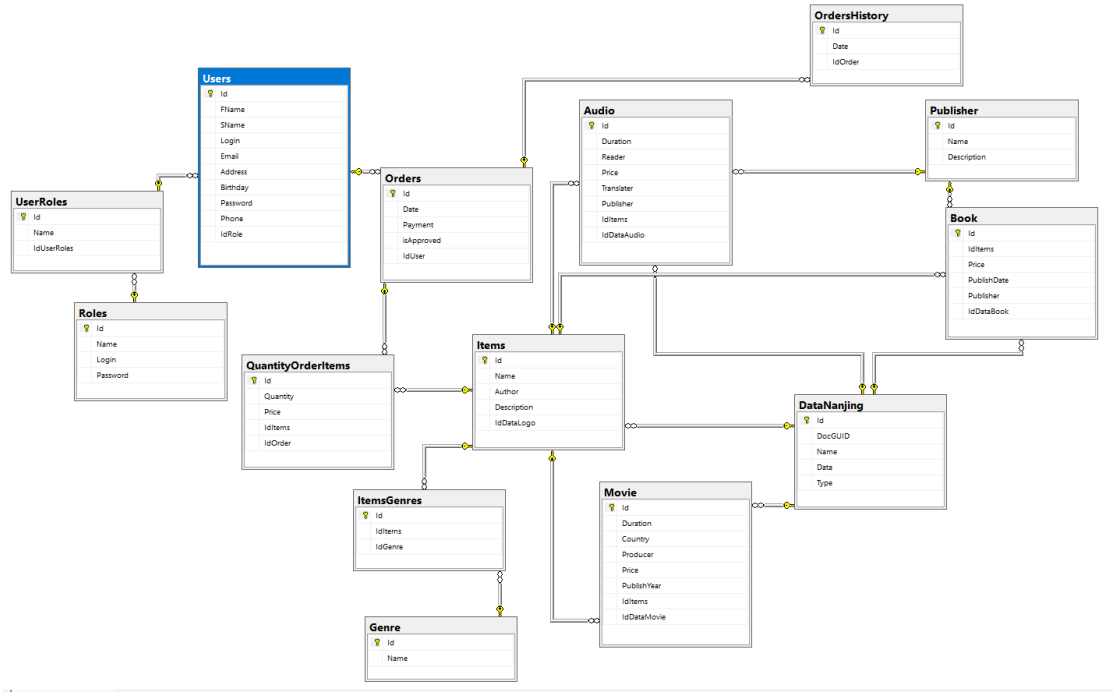


Рисунок 1.1 – Диаграмма базы данных

Для базы данных книжного магазина мной было разработано 14 таблиц. Диаграмма связей таблиц для необходимой базы данных представлена на рисунке 1.1.

Таблица Users, содержащая информацию о всех пользователях приложения:

– Id – уникальный идентификатор;
– FName – имя пользователя;
– SName – фамилия пользователя;
– Email – электронная почта пользователя;
– Birthday – дата рождения пользователя;
– Telephone – мобильный телефон пользователя;
– Login – имя для входа пользователя в приложение;
– Password – пароль пользователя;
– RoleId – роль пользователя в приложении, ограничение на список ролей.

Таблица Publisher, содержащая информацию об издательских домах:

– Id - уникальный идентификатор издателя;
– Name – название издателя;
– Description – описание издателя.

Таблица UserRole, ролей пользователей, позволяющая связать модель пользователя в приложении с моделью пользователя СУБД:

– Id – уникальный идентификатор;
– Name – название роли;
– IdUserRoles – уникальный идентификатор в таблице пользователей СУБД.

Таблица UserRoles, предназначенная для хранения данных пользователей СУБД для доступа к информации:

– Id – уникальный идентификатор роли;

– Name – имя для входа в СУБД;

– Password – пароль для входа в СУБД.

Таблица Item – одна из основных таблиц базы данных, таблица общего описания товаров:

– Id – уникальный идентификатор товара;

– Name – название товара;

– Description – описание товара;

– Author – автор товара.

– IdDataLogo – уникальный идентификатор на таблицу данных для получения Id изображения для товара (может быть пустой).

Таблица Book, также одна из основных таблиц, отвечающая за описание электронных книг:

– Id – уникальный идентификатор книги;

– IdItems – уникальный идентификатор на таблицу товаров (общее описание);

– Price – цена книги;

– PublishDate – дата выпуска;

– Publisher – уникальный идентификатор на таблицу издателей (может быть пустой);

– IdDataBook – уникальный идентификатор на таблицу данных для получения Id файла книги для товара (может быть пустой).

Таблица Audio, также одна из основных таблиц, отвечающая за описание аудиокниги:

– Id – уникальный идентификатор аудиозаписи;

– IdItems – уникальный идентификатор на таблицу товаров (общее описание);

– Duration – продолжительность аудиозаписи;

– Transleter – переводчик книги, если была на иностранном языке (может быть пустой);

– Reader – чтец книги:

– Price – цена аудиозаписи книги;

– Publisher – уникальный идентификатор на таблицу издателей (может быть пустой);

– IdDataAudio – уникальный идентификатор на таблицу данных для получения Id файла аудиозаписи книги для товара (может быть пустой).

Таблица Movie, также одна из основных таблиц, отвечающая за описание фильмов поставленных по мотивам книг:

– Id – уникальный идентификатор трейлера фильма;

– IdItems – уникальный идентификатор на таблицу товаров (общее описание);

– Duration – продолжительность трейлера фильма;

– Country – страна издания фильма;

– Producer – продюсер фильма:

– Price – цена фильма;

– Publisher – уникальный идентификатор на таблицу издателей (может быть пустой);

– IdDataMovie – уникальный идентификатор на таблицу данных для получения Id файла фильма для товара (может быть пустой).

Таблица DataNanjing, основная таблица базы данных, отвечающая за хранение файлов всех форматов, таких как, аудиозаписи текстовые файлы, видеозаписи, изображения:

– Id – уникальный идентификатор файла;

– DocGUID – глобальный уникальный идентификатор файлов;

– Name – название файла;

– Data – хранит файл базы данных;

– Type – описание формата файла (audio, video, text, image).

Таблица Genre, отвечает за описание жанров произведений:

– Id – уникальный идентификатор жанра;

– Name – название жанра.

Таблица ItemesGenre, дополнительная таблица отвечает за сопоставление жанров произведений и товаров (сами произведения):

– Id – уникальный идентификатор записи;

– IdGenre – уникальный идентификатор на таблицу жанров.

– IdItems – уникальный идентификатор на таблицу товаров (общее описание).

Таблица QuantityOrderItems, содержит описание товара для хранения в корзине:

– Id – уникальный идентификатор записи;

– Quantity – количество товаров (по умолчанию 1);

– Price – цена товара за 1 экземпляр.

– IdItems – уникальный идентификатор на таблицу товаров (общее описание);

– IdOrder – уникальный идентификатор на таблицу заказов.

Таблица Orders, содержит описание заказа товаров (корзина):

– Id – уникальный идентификатор записи;

– Date – дата заказа;

– Payment – тип оплаты (по умолчанию наличными).

– IsApproved – логическое поле (по умолчанию значение 0), которое показывает подтверждён ли товар для заказа, если подтверждён значение меняется на 1;

– IdUser – уникальный идентификатор на таблицу пользователей.

Все скрипты для создания всех таблиц базы данных, а также для создания самой базы данных представлены в Приложении А.

2 Разработка необходимых объектов

2.1 Таблицы

Таблицы являются основой любой базы данных, именно в них хранится вся информация. При проектировании базы данных была создана 14 таблиц, которые подробно описаны ранее в разделе 1, а SQL-скрипты для их создания находятся в Приложении А.

2.2 Пользователи

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей. При проектировании базы данных понадобилось 4 пользователя: по умолчанию (без входа в систему) можно зарегистрироваться и может только просматривать всю информацию, которая есть в базе; зарегистрированный пользователь вдобавок к возможностям неавторизованного пользователя, имеет право на бронирование билета; диспетчер может всё, что может авторизованный пользователь, и может создавать новые элементы базы (станции, поезда, маршруты, цены и т.д.); администратор может всё, что и диспетчер, а так же может создавать резервную копию и восстанавливать базу из неё.

Таким образом, администратор был наделён привилегией на выполнение всех хранимых процедур, разработанных для данной базы данных.

Менеджер может выполнять хранимые процедуры создания и просмотра информации.

Клиенту разрешено выполнять хранимые процедуры, связанные с просмотром информации, а также процедуру регистрации или входа в приложение, процедуру добавления товара в корзину с последующим заказом товара.

Неавторизованному пользователю доступен лишь просмотр информации и регистрация в приложении.

Скрипты для создания пользователей базы данных, ролей и логинов представлены в приложении Б.

2.3 Процедуры

Использование хранимых процедур позволяет ограничить либо вообще исключить непосредственный доступ пользователей к таблицам базы данных, оставив только администраторам на выполнение хранимых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным, а также ограничение на добавление и работу с резервными копиями. Листинги некоторых хранимых процедур представлены в приложении В.

Всего было разработано 50 процедур:

1. AddAudio – процедура для добавления аудиозаписи;
2. AddBook – процедура для добавления книги;
3. AddGenre – процедура для добавления жанра;

4. AddItem – процедура добавления товара;
5. AddItemstoCart – процедура для добавления товара в корзину;
6. AddMovie – процедура добавления фильма;
7. AddData – процедура добавления файлов данных;
8. AddPublisher – процедура добавления издателя;
9. AddQuantityOrderItems – процедура для добавления товара (этап пред корзины);
10. AddToOrdersHistory – процедура добавления заказа в историю заказов;
11. CreateUser – процедура создания пользователя;
12. DeleteAudio – процедура удаления аудио;
13. DeleteBook – процедура удаления книги;
14. DeleteGenre – процедура удаления жанра;
15. DeleteItem – процедура удаления товара;
16. DeleteMovie – процедура удаления фильма;
17. DeletePublisher – процедура удаления издателя;
18. DeleteItemsFromCart – процедура удаления товара из корзины;
19. ExportToXML – экспорт в XML-файл;
20. GetAllAudio – процедура получения всех аудио;
21. GetAllBook – процедура получения всех книг;
22. GetAllGenre – процедура получения всех жанров;
23. GetAllItem– процедура получения всех товаров;
24. GetAllMovie – процедура получения всех фильмов;
25. GetAllPublisher – процедура получения всех издателей;
26. GetAudioById – процедура получения аудио по полю первичного ключа (Id);
27. GetAudioByName – процедура получения аудио по названию;
28. GetBookById – процедура получения аудио по полю первичного ключа (Id);
29. GetBookByName – процедура получения книг по названию;
30. GetMovieById – процедура получения книг по полю первичного ключа (Id);
31. GetMovieByName – процедура получения фильма по названию;
32. GetGenreById – процедура получения фильма по полю первичного ключа (Id);
33. GetItemById – процедура получения товара по полю первичного ключа (Id);
34. GetItemByName – процедура получения товара по названию;
35. GetItemsFromCart – процедура получения товаров из корзины;
36. GetPublisherById – процедура получения издателей по полю первичного ключа (Id);
37. GetPublisherByName – процедура получения издателей по названию;
38. GetRole – процедура получения ролей базы данных;
39. GetUser – процедура получения пользователей приложения;
40. ImportFromXML – процедура импорта из XML-файл;

41. LoginIntoNanjing – процедура авторизации пользователя;
42. UpdateAudio – процедура обновления данных об аудио;
43. UpdateBook – процедура обновления данных о книги;
44. UpdateMovie – процедура обновления данных о фильма;
45. UpdateGenre – процедура обновления данных жанра;
46. UpdatePublisher – процедура обновления данных издателя;
47. UpdateItems – процедура обновления данных товара;
48. Generate_quantity – процедура для заполнения базы на 100000 строк.

Все скрипты хранимых процедур приложены в отдельных файлах в папке Scripts директории прилагаемого диска.

2.4 Триггер

Триггер – хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных.

Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики.

Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

В приложении используется триггер на событие обновления таблицы Items, Audio, Movie, Book для оптимизации заполнения полей идентификаторов данных.

Листинг триггера представлен в приложении Г.

3 Описание процедур импорта и экспорта данных

Можно выполнить массовый импорт XML-документов в базу данных SQL Server или осуществить массовый экспорт XML-документов из базы данных SQL Server. В этом разделе приведены примеры и того, и другого.

Для выполнения массового импорта данных из файла в таблицу SQL Server или не секционированное представление могут использоваться следующие средства:

– Программа bcp – может выполнять экспорт везде в базе данных SQL Server, где работает инструкция SELECT, включая секционированные представления;

– BULK INSERT – загружает данные из файла данных в таблицу. Эти функциональные возможности аналогичны тем, которые предоставляются параметром in команды bcp, но чтение файла данных выполняется процессом SQL Serve;

– Инструкции INSERT ... SELECT * FROM OPENROWSET (BULK...).

3.1 Процедура импорта данных из XML-файла

Для импорта используется процедура ImportFromXML. Пример данного скрипта представлена в Приложении Д, результат вызова процедуры показан на рисунке 3.1.

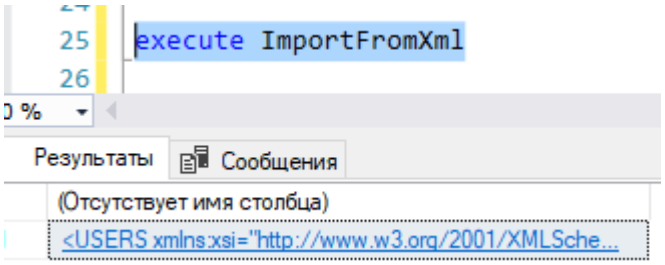


Рисунок 3.1 – Результат вызова процедуры экспорта файлов

При массовом импорте XML-данных из файла, содержащего объявление кодировки, которое необходимо применить, нужно указать параметр SINGLE_BLOB в предложении OPENROWSET (BULK…). Параметр SINGLE_BLOB гарантирует, что средство синтаксического анализа XML в SQL Server произведет импорт данных в соответствии со схемой кодирования, указанной в XML-объявлении.

3.2 Процедура экспорта данных в XML-файл

Для экспорта данных в XML-файл используется процедура ExportToXML.

Пример скрипта процедуры экспорта в XML-файл представлен в Приложении Д. Результат выполнения можно увидеть на следующем рисунке 3.2.

```
38  execute ExportToXML
```

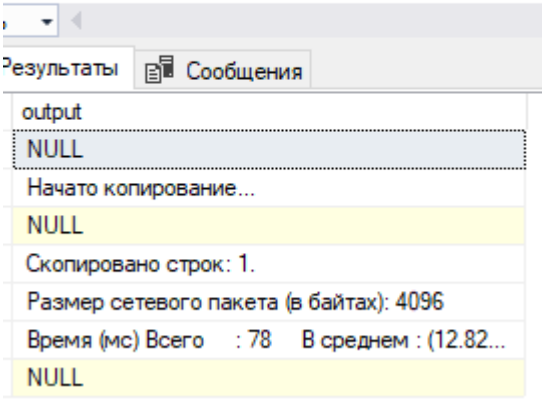


Рисунок 3.2 – Результат вызова процедуры экспорта файлов

SQL Server не сохраняет кодировку XML, если XML-данные постоянно хранятся в базе данных. Поэтому оригинальная кодировка полей XML недоступна при экспорте XML-данных. SQL Server использует для экспорта XML-данных кодировку UTF-16.

4 Технология хранения мультимедийных типов данных

Для хранения мультимедийных типов данных в данной базе данных используется компонент FILESTREAM, который обеспечивает хранение неструктурированные данные, например, документы и изображения, а также осуществлять эффективный доступ к данным больших двоичных объектов.

Содержимое файла помещается в поле data типа varbinary(max). Для снятия ограничения размера используется атрибут FILESTREAM (в действительности размер ограничен базовым томом NTFS), а стандартное ограничение типа varbinary(max), согласно которому размер файла не должен превышать 2 ГБ, не применяется к большим двоичным объектам, сохраняемым в файловой системе

Скрипт создания таблицы FILESTREAM представлен на рисунке 4.1.

```
create table DataNanjing
(
    Id int primary key identity(1,1),
    DocGUID UNIQUEIDENTIFIER NOT NULL ROWGUIDCOL UNIQUE DEFAULT NEWID (),
    Name nvarchar(260) not null,
    Data varbinary(max) FILESTREAM not null,
    [Type] nvarchar(8) check ([type] in ('text','audio','video','picture','file')) default 'file'
);
```

Рисунок 4.1 – Скрипт создания таблицы FILESTREAM.

Таблица, имеющая один или несколько столбцов FILESTREAM, должна также иметь столбец типа uniqueidentifier с атрибутом ROWGUIDCOL. Этот столбец не должен допускать значений NULL и должен иметь относящееся к одному столбцу ограничение UNIQUE или PRIMARY KEY. Значение идентификатора GUID для столбца должно быть предоставлено либо приложением во время вставки данных, либо ограничением DEFAULT, которое использует функцию NEWID ().

5 Тестирование

5.1 Тестирование производительности базы данных

Для тестирования производительности была взята за основу таблица связей маршрутов с расписанием (QuantityOrdersItem), так как выборка данных из неё позволяет переходить по нескольким таблицам.

Изначально таблица QuantityOrdersItem была заполнена на 100000 строк с помощью процедуры, представленной на рисунке 5.1.

```
go
alter procedure generate_quantity
as
begin
    declare @Price float;
    declare @IdOrder nvarchar(max);
    declare @IdItems int;
    declare @Quantity int;
    declare @IdDataItem int;
    declare @TypeItem nvarchar(max);
    declare @counter int = 0;
    begin try
        while @counter <> 100000
            begin
                set @counter = @counter + 1;
                set @Price =  ROUND(1 + RAND() * 99, 2);
                set @Quantity =  ROUND(RAND() * 10, 0);
                set @IdItems = (select top(1) id from Items order by NEWID());
                set @IdOrder = (select top(1) id from Orders order by NEWID());
                set @IdDataItem = ROUND(RAND() * 3, 0);
                set @TypeItem = 'video'
                execute AddQuantityOrderItems @Quantity, @Price, @IdItems, @IdOrder, @IdDataItem, @TypeItem;
            end
    end try
    begin catch
        select ERROR_MESSAGE()
    end catch
end

execute generate_quantity
```

Рисунок 5.1 – Заполнение таблицы QuantityOrderItems

После этого был применён SELECT-запрос к данной таблице и при помощи стандартных средств SQL Management Studio получена предполагаемая стоимость T-SQL выражения, а также получен предполагаемый план выполнения всего запроса (рисунок 5.3). Результат данной оценки запроса приведён на рисунке 5.2.

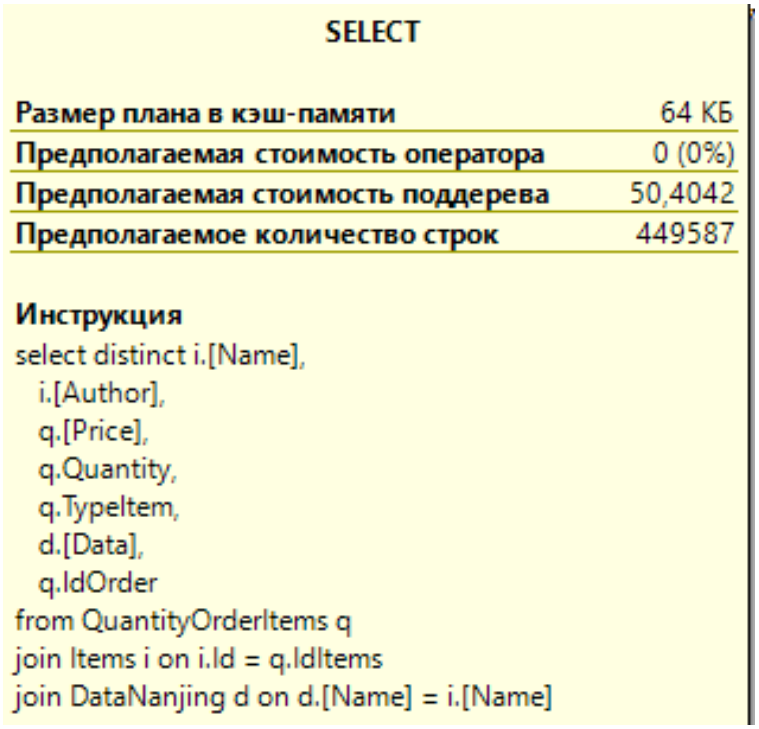


Рисунок 5.2 – Оценка времени выполнения select-запроса к таблице

Фактический план выполнения содержит сведения времени выполнения, такие как фактические метрики использования ресурса и предупреждения времени выполнения (если они есть). Создаваемый план выполнения отображает фактический план выполнения запроса, который используется в Компонент SQL Server Database для выполнения запросов.

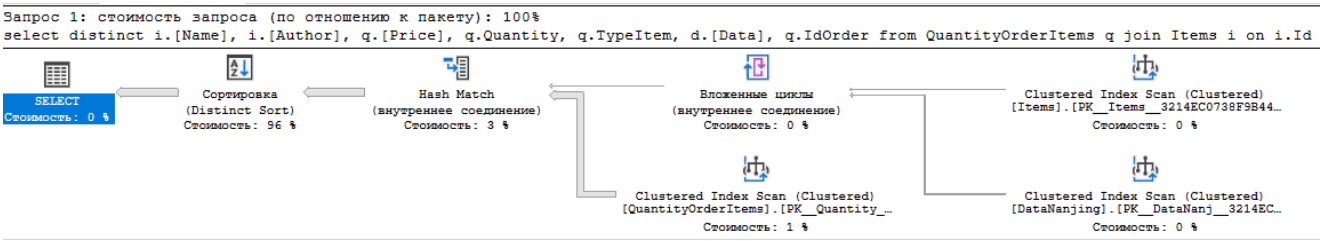


Рисунок 5.3 – Предполагаемый план выполнения запроса

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных.

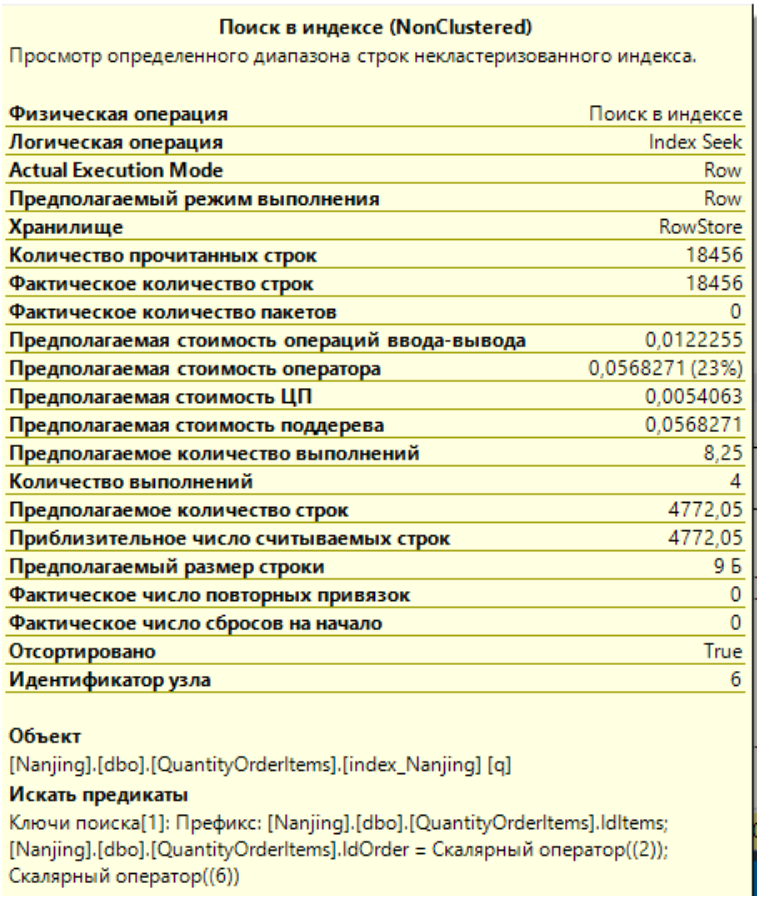
Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени.

Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска.

Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированного дерева.

Из-за того, что в запросе идёт выборка по кластеризованным индексам в таблицах, создание дополнительных индексов не требовалось, что свидетельствует о грамотном выборе структуры базы данных и связей между таблицами.

После проведения первоначальной оценки был построен не кластеризованный составной индекс покрытия к таблице QuantityOrdersItem по столбцам IdItem и id_order, включая столбцы price, quantity и проведена оценка такого же SELECT-запроса к таблице Заказы. Результаты, полученные во время оценки, представлены на рисунке 5.4.



Поиск в индексе (NonClustered)

Просмотр определенного диапазона строк некластеризованного индекса.

Физическая операция	Поиск в индексе
Логическая операция	Index Seek
Actual Execution Mode	Row
Предполагаемый режим выполнения	Row
Хранилище	RowStore
Количество прочитанных строк	18456
Фактическое количество строк	18456
Фактическое количество пакетов	0
Предполагаемая стоимость операций ввода-вывода	0,0122255
Предполагаемая стоимость оператора	0,0568271 (23%)
Предполагаемая стоимость ЦП	0,0054063
Предполагаемая стоимость поддерева	0,0568271
Предполагаемое количество выполнений	8,25
Количество выполнений	4
Предполагаемое количество строк	4772,05
Приблизительное число считываемых строк	4772,05
Предполагаемый размер строки	9 Б
Фактическое число повторных привязок	0
Фактическое число сбросов на начало	0
Отсортировано	True
Идентификатор узла	6

Объект

[Nanjing].[dbo].[QuantityOrderItems].[index_Nanjing] [q]

Искать предикаты

Ключи поиска[1]: Префикс: [Nanjing].[dbo].[QuantityOrderItems].IdItems; [Nanjing].[dbo].[QuantityOrderItems].IdOrder = Скалярный оператор((2)); Скалярный оператор((6))

Рисунок 5.4 – Оценка работы некластеризованого индекса таблицы QuantityOrdersItem

По результатам проведённых оценок до и после построения не кластеризованного индекса покрытия можно сделать вывод, что после создания индекса получили прирост скорости выборки из таблицы при большом наборе.

Таким образом, мы получили прирост производительности и более выгодное распределение ресурсов, в зависимости от параметра, характеризующего запрос.

6 Руководство пользователя

Для взаимодействия пользователя с базой данных используются процедуры, описанные выше. В самой базе данных созданы как пользователи базы, так и для разработки приложения, таких пользователя четыре: администратор, менеджер, пользователь и пользователь базы данных по умолчанию.

Взаимодействие происходит благодаря процедурам. Для каждого пользователя динамически создаётся объект подключения, который и вызывает процедуры.

Каждая процедура возвращает при удачном запросе значение result = 1, иначе в процедуре генерируется ошибка (также работает с процедурой авторизации).

Процесс работы с базой данных может осуществляться и без авторизации пользователя. В этом случае пользователю будет доступна любая выдаваемая информация, но изменять базу данных сможет только в случае регистрации нового пользователя.

Вызов процедур регистрации нового пользователя и авторизации существующего пользователя представлен на рисунке 6.1. Полный листинг процедур регистрации и авторизации находится в приложении В.

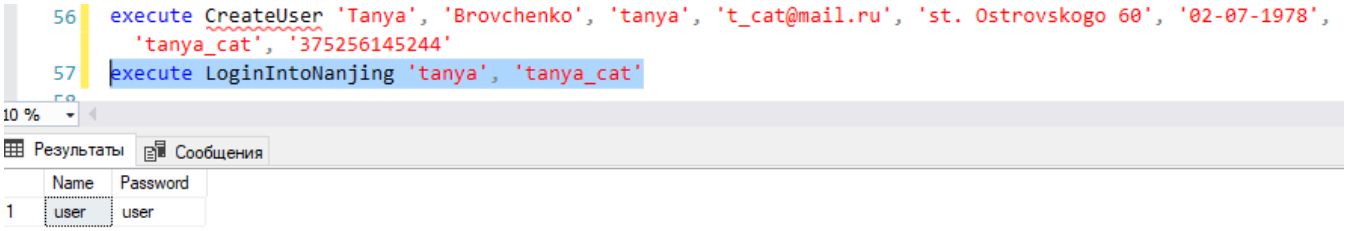


Рисунок 6.1 – Вызов процедур регистрации и авторизации пользователей

При входе в систему, сервер будет вызывать процедуру LoginIntoNanjing, куда передаёт логин и пароль. В случае успеха, он будет записывать полученные данные для создания подключения к базе данных для роли этого пользователя и далее, за счёт хранения информации о текущем сеансе, позволяет свободно перемещаться по всей структуре базы данных с учётом роли и предоставленных разрешений роли.

Далее рассмотрим возможности для каждой роли пользователя.

6.1 Администратор

При входе с учётной записью администратора пользователь получает доступ ко всем без исключения процедурам в приложении, в том числе и технологию хранения мультимедийных типов данных и управление потоком filestream базы данных, а также администратор единственный кто может добавлять менеджеров и удалять пользователей рисунок 6.2.

```
114  execute CreateUser 'Tanya', 'Brovchenko', 'tanya', 't_cat@mail.ru', 'st. Ostrovskogo 60', '02-07-1978',
       'tanya_cat', '375256145244'
115  execute CreateUser 'Вова', 'Бровченко', 'vovan', 'vovan_bear@mail.ru', 'st. Ostrovskogo 60',
       '22-11-1998', 'vovan_bear', '375251132323'
116
117  execute LoginIntoNanjing 'tanya', 'tanya_cat'
118
119  execute GetRole
120  execute GetUser
121  execute DeleteUser 5
122  execute AddManager 'Оля', 'Шинкарёва', 'olya_princec', 'mylove@mail.ru', 'пер. Товарный 13',
       '28-08-1999', 'olychan', '375296657676'
123
```

10 %

Результаты | Сообщения

	Id	FName	SName	Login	Email	Address	Birthday	Password	Phone	IdRole
1	3	Oleg	Lebedev	oleg	oleg564@gmail.com	st. Belarussian 19	1998-12-18	oleg_jojo	375334546371	3
2	4	Tanya	Brovchenko	tanya	t_cat@mail.ru	st. Ostrovskogo 60	1978-07-02	tanya_cat	375256145244	3
3	5	Оля	Шинкарёва	olya_princec	mylove@mail.ru	пер. Товарный 13	1999-08-28	olychan	375296657676	2
4	6	Вова	Бровченко	vovan	vovan_bear@mail.ru	st. Ostrovskogo 60	1998-11-22	vovan_bear	375251132323	3
5	7	Hanna	Bulava	hannika	hannanice94@gmail.com	st. Belarussian 19	1998-08-14	123QWEasd	375292414148	1
6	8	Andrei	Chaeuvskii	andrei	andrei_chaevskii@bk.ru	st. Belarussian 21	1998-12-08	asdQWE123	375336345353	2
7	9	Alena	Savchuk	alena	alena_savchuck@list.ru	st. Belarussian 19	1999-01-29	asdQWE123	375333516762	3

Рисунок 6.2 –Вызова процедур, доступных только администратору

Все разрешения предоставленные роли администратора показаны в приложении Б.

6.2 Зарегистрированный пользователь

При успешной аутентификации как пользователь, клиент получает возможность не только обычной выборке данных (фиолетовый прямоугольник на рисунке 6.3), но и добавление товара в корзину (оформление заказа) рисунке 6.4).

```
go
declare @price float = (select price from Audio where id = 1)
execute AddQuantityOrderItems 4, @price, 1, 6, 1, 'audio'
go
declare @date date = (select GETDATE())
execute AddItemstoCart @date, 'paypal', 7, 0

execute GetItemsFromCart 0
execute DeleteItemsFromCart 5
```

Рисунок 6.3 – Процедуры корзины пользователя

На рисунке выше показан вызов нескольких процедур для добавления товара в корзину, данное разделение было выполнено с той точки зрения, так как пользователи имеют учётные записи, следовательно, в дальнейшем сохранение понравившихся товаров, а не непосредственный заказ сразу после выбора товара, что позволит чаще заказывать интересующие предметы из магазина и в дальнейшем повысит конверсию посещения.

```
execute GetData
execute GetAllItem
execute GetAllBook
execute GetAllMovie
execute GetAllAudio

execute GetItemById 2
execute GetAudioById 1
execute GetMovieById 1
execute GetBookById 2

execute GetAudioByName 'Первому игроку приготовиться'
execute GetMovieByName 'Первому игроку приготовиться'
execute GetItemByName 'Первому игроку приготовиться'
execute GetBookByName 'Первому игроку приготовиться'

execute DeleteAudio 2
execute DeleteMovie 1
execute DeleteBook 2
execute DeleteItem 2

execute UpdateAudio 3, '11:17:09', 'Клюквин Александр', 8.26, 'Росмэн', 6
execute UpdateMovie 2, '2:32:00', 'Великобритания', 'Крис Коламбус', 26, 2001
execute UpdateBook 2, 32.2, '2016', 3
execute UpdateItems 1, 'Первому игроку приготовиться', 'Эрнест Клайн', 'В 2045 году реальный мир – не самое приятное место. По-
  настоящему живым Уэйд Уоттс чувствует себя лишь в OASISe – огромном виртуальном пространстве, где проводит свои дни большая часть
   человечества. Перед смертью эксцентричный создатель OASISa, одержимый по-культурой прошлых лет, составляет на ее основе ряд
  сложнейших головоломок. Тот, кто разгадает их первым, унаследует его огромное состояние – и контроль над самим OASISом.', '6'

execute AddAudio '11:17:09', 'Клюквин Александр', 8.26, NUll, 'Росмэн'
execute AddMovie '2:32:00', 'Великобритания', 'Крис Коламбус', 26, 2001
execute AddBook 32.2, '2016', 3
execute AddItem 'Первому игроку приготовиться', 'Эрнест Клайн', 'В 2045 году реальный мир – не самое приятное место. По-настоящему
  живым Уэйд Уоттс чувствует себя лишь в OASISe – огромном виртуальном пространстве, где проводит свои дни большая часть
  человечества. Перед смертью эксцентричный создатель OASISa, одержимый по-культурой прошлых лет, составляет на ее основе ряд
  сложнейших головоломок. Тот, кто разгадает их первым, унаследует его огромное состояние – и контроль над самим OASISом.'
```

Рисунок 6.4 – Процедуры по выбору доступные пользователю

Пользователь так же может использовать возможности фильтра для поиска всех товаров различной категории, так и по имени отдельного товара, либо просмотреть доступные жанры произведений. Все процедуры фильтра данных описаны в приложении Е.

6.3 Менеджер

Пользователю с ролью «Менеджер» доступно всё, что доступно клиентам с ролью «Пользователь», процедуры редактирования данных такие как: создания, обновления и удаления информации о книгах, фильмах, аудиокнигах, и товаров в целом, кроме функций, доступных только клиенту с ролью «Администратор» которые были описаны выше. Все разрешения на выполнение соответствующих процедур доступных менеджеру описаны в приложении Б.

6.4 Вывод

В итоге, функционал базы данных достаточно обширен для выполнения различных задач. В дальнейшем соответственно может подвергаться изменениям и улучшениям как с точки зрения пользователей так и с точки зрения производительности работы с большим объёмом данных.

Заключение

В данном курсовом проекте была разработана база данных с использованием технологии для хранения мультимедийных данных, а также реализованы различные процедуры удовлетворения нужд пользователя, а также для упрощения взаимодействия с данными.

Помимо этого, добавлена возможность выполнить экспорт/импорт из/в XML-файл на случай некорректного ввода данных, разработан функционал взаимодействия пользователя по ролям базы данных согласно соответствующим разрешениям.

Также требования технического задания выполнены в полном объеме.

Список использованной литературы

1. Блинова Е.А. Курс лекций по базам данных / Е.А. Блинова
2. Пацей, Н.В. Технология разработки программного обеспечения / Н.В. Пацей. – Минск: БГТУ, 2016. – 129 с.
3. Оф. Документация к SQL [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/ru-ru/sql/?view=sql-server-2017 - Дата доступа: 20.03.2019.

Приложение А.

```
USE [master]
GO

EXEC sp_configure filestream_access_level, 2
RECONFIGURE

create database Nanjing;
use Nanjing;

alter database Nanjing add filegroup FileStreamGroup1 contains filestream
alter database Nanjing add file (
	name = FSGroup1File,
	filename = 'F:\Hannika\Univer\Semestr_6\CP_DB\CP_Nanjing\DB\file group') to filegroup
FileStreamGroup1

--роли бд
create table Roles
(
	Id int primary key identity(1,1) not null,
	Name nvarchar(max) not null,
	Login nvarchar(max) not null,
	Password nvarchar(25) not null
);

--роли приложения
create table UserRoles
(
	Id int primary key identity(1,1) not null,
	Name nvarchar(max) not null,
	IdUserRoles int foreign key references Roles(Id)
);

--пользователи
create table Users
(
	Id int primary key identity(1,1) not null,
	FName nvarchar(max) not null,
	SName nvarchar(max) not null,
	[Login] nvarchar(25) not null,
	Email nvarchar(max) not null,
	[Address] nvarchar(max) not null,
	Birthday date not null,
	[Password] nvarchar(25) not null,
	Phone nvarchar(12) not null,
	IdRole int foreign key references UserRoles(Id)
);

create table Genre
(
	Id int primary key identity(1,1) not null,
	Name nvarchar(max) not null
);


create table ItemsGenres
(
	Id int primary key identity(1,1) not null,
	IdItems int foreign key references Items(Id),
	IdGenre int foreign key references Genre(Id)
);
```

```
 --General description items
create table Items
(
	Id int primary key identity(1,1) not null,
	Name nvarchar(max) not null,
	Author nvarchar(max) not null,
	Description nvarchar(max) not null,
	IdDataLogo int foreign key references DataNanjing(Id)
);


create table Publisher
(
	Id int primary key identity(1,1) not null,
	Name nvarchar(max) not null,
	Description nvarchar(max) not null,
);

use Nanjing

--Description of order
create table Orders
(
	Id int primary key identity(1,1) not null,
	Date date not null,
	Payment nvarchar(max) default('cash'),
	isApproved bit not null default 0,
	IdUser int foreign key references Users(Id)
);


--Quantity of items oredered
create table QuantityOrderItems
(
	Id int primary key identity(1,1) not null,
	Quantity int not null default(1),
	Price float not null,
	IdItems int foreign key references Items(Id),
	IdOrder int foreign key references Orders(Id)
);

--History orderds
create table OrdersHistory
(
	Id int primary key identity(1,1) not null,
	Date date not null,
	IdOrder int foreign key references Orders(Id)
);

--Description of book
create table Book
(
	Id int primary key identity(1,1) not null,
	IdItem int foreign key references Items(Id) not null,
	Price float not null,
	PublishDate date not null,
	Publisher int foreign key references Publisher(Id),
	IdDataBook int foreign key references DataNanjing(Id)
);

--Description movie
create table Movie
(
	Id int primary key identity(1,1) not null,
```

```
	Duration time(7),
	Country nvarchar(max),
	Producer nvarchar(max),
	Price float,
	PubliserYear date,
	IdItems int foreign key references Items(Id),
	IdDataMovie int foreign key references DataNanjing(Id)
);

--Description audio
create table Audio
(
	Id int primary key identity(1,1) not null ,
	Duration time(7),
	Reader nvarchar(max),
	Price float,
	Translater nvarchar(max),
	Publisher int foreign key references Publisher(Id),
	IdItems int foreign key references Items(Id),
	IdDataAudio int foreign key references DataNanjing(Id)
);

alter table Movie ALTER COLUMN duration time(7) null
alter table Movie ALTER COLUMN Country nvarchar(max) null
alter table Movie ALTER COLUMN Producer nvarchar(max) null
alter table Movie ALTER COLUMN Price float null

--Files data of Nanjing
create table DataNanjing
(
	Id int primary key identity(1,1),
	DocGUID UNIQUEIDENTIFIER NOT NULL ROWGUIDCOL UNIQUE DEFAULT NEWID (),
	Name nvarchar(260) not null,
	Data varbinary(max) FILESTREAM not null,
	[Type] nvarchar(8) check ([type] in ('text','audio','video','picture','file')) default
'file'
);

--Alternative full variant data table

--create table NangingFilesData as FILETABLE with
--(
--	FILETABLE_DIRECTORY = 'NangingFileTable'
--);


drop table Items;

drop table OrdersHistory;
drop table QuantityOrderItems;
drop table Orders;
drop table Movie;
drop table Audio;
drop table Publisher;
drop table Book;

drop table Roles;
drop table Users;
drop table Genre;
drop table DataNanjing;
drop database Nanjing;
```

Приложение Б.

```
USE [master]
GO

create login [admin_Nanjing] with password = 'admin',
DEFAULT_DATABASE=[master], DEFAULT_LANGUAGE=[русский],
CHECK_EXPIRATION=OFF, CHECK_POLICY=ON;
GO

alter server role [dbcreator] add member [admin_Nanjing];
GO

ALTER LOGIN [adminR_Nanjing] DISABLE
GO

create login [manager_Nanjing] with password = 'manager',
DEFAULT_DATABASE=[Nanjing], DEFAULT_LANGUAGE=[русский],
CHECK_EXPIRATION=OFF, CHECK_POLICY=ON;
GO

ALTER LOGIN [manager_Nanjing] DISABLE
GO

create login [default_Nanjing] with password = 'default',
DEFAULT_DATABASE=[Nanjing], DEFAULT_LANGUAGE=[русский],
CHECK_EXPIRATION=OFF, CHECK_POLICY=ON;
GO

ALTER LOGIN [default_Nanjing] DISABLE
GO

create login [user_Nanjing] with password = 'user',
DEFAULT_DATABASE=[Nanjing], DEFAULT_LANGUAGE=[русский],
CHECK_EXPIRATION=OFF, CHECK_POLICY=ON;
GO

ALTER LOGIN [user_Nanjing] DISABLE
GO


create user admin_Nanjing for login [admin];
create user manager_Nanjing for login [manager_Nanjing];
create user default_Nanjing for login [default_Nanjing];
create user user_Nanjing for login [user_Nanjing];


use Nanjing

--не выполнено ещё

grant execute on object::AddItem
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::AddAudio
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::AddBook
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::AddMovie
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::AddGenre
```

```
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::AddPublisher
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::DeleteItem
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::DeleteAudio
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::DeleteBook
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::DeleteMovie
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::DeleteGenre
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::DeletePublisher
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::CreateUser
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetRole
       to [admin_Nanjing];

grant execute on object::GetUser
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::GetAllItem
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetItemById
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetItemByName
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetAllAudio
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetAudioById
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetAudioByName
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetAllBook
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetBookById
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetBookByName
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetAllMovie
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetMovieById
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];
```

```
grant execute on object::GetMovieByName
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetAllGenre
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetGenreById
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetAllPublisher
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetPublisherById
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::GetPublisherByName
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::LoginIntoNanjing
       to [admin_Nanjing], [manager_Nanjing], [user_Nanjing], [default_Nanjing];

grant execute on object::UpdateItems
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::UpdateAudio
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::UpdateBook
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::UpdateMovie
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::UpdateGenre
       to [admin_Nanjing], [manager_Nanjing];

grant execute on object::UpdatePublisher
       to [admin_Nanjing], [manager_Nanjing]


insert into Roles([Name], [Login], [Password])
       values('admin', 'admin_Nanjing', 'admin'),
              ('manager', 'manager_Nanjing', 'manager'),
              ('default', 'default_Nanjing', 'default'),
              ('user', 'user_Nanjing', 'user');

insert into UserRoles
             values ('Администратор', 1),
                    ('Менеджер',      2),
                    ('Пользователь',  4);
```

Приложение В

```
use Nanjing;
select * from Users
drop procedure CreateUser

--регистрация
go
create procedure CreateUser
@FName nvarchar(max),
@SName nvarchar(max) ,
@Login nvarchar(25),
@Email nvarchar(max),
@Address nvarchar(max),
@Birthday date,
@Password nvarchar(25),
@Phone nvarchar(12),
@IdRole int
as
begin
	declare @countBefore int,
			@countAfter int,
			@count int,
			@result int
	set @countBefore = (select count(*) from [Users])
	insert into [Users] (FName, SName, [Login], Email, [Address], Birthday, [Password],
Phone,  IdRole)
		values (@FName, @SName, @Login, @Email, @Address, @Birthday, @Password, @Phone,
@IdRole)
		set @countAfter = (select count(*) from [Users])
	set @result = @countAfter - @countBefore
	select @result as result
end

--авторизация
go
create procedure LoginIntoNanjing
@login nvarchar(25),
@password nvarchar(25)
as
begin
	if(select COUNT(*) from Users
	   where [login] = @login and [password] = @password) = 1
	   begin
			select [Name], [Password] from Roles
			where Id = (select IdUserRoles from UserRoles
					where Id = (select [IdRole] from [Users]
							where [Login] = @login and
[Password] = @password))
	   end
	else
		begin
			select -1 as result
		end
end
go

go
create procedure GetRoles
as
begin
	select * from UserRoles
end
```

```
go
create procedure GetUser
as
begin
	select * from Users
end

go
create procedure AddQuantityOrderItems
	@quantity int,
	@price float,
	@iditems int,
	@idorders int
as
begin
	begin try
		declare @countBefore int,
			@countAfter int,
			@result int
		set @countBefore = (select count(*) from QuantityOrderItems)
		insert into QuantityOrderItems ([Quantity], [price], [IdItems], IdOrder)
			values ( @quantity, @price, @iditems, @idorders)
		set @countAfter = (select count(*) from QuantityOrderItems)
		set @result = @countAfter - @countBefore
		select @result as result
	end try
	begin catch
		select -1 as result
	end catch
end

go
create procedure AddItemstoCart
	@Date date,
	@Payment nvarchar(max),
	@IdUser int,
	@isApproved bit
as
begin
	begin try
		declare @countBefore int,
			@countAfter int,
			@result int
		set @countBefore = (select count(*) from QuantityOrderItems)
		insert into Orders ([Date],[Payment], [IdUser], [isApproved])
			values ( @Date, @Payment, @IdUser, @isApproved)
		set @countAfter = (select count(*) from QuantityOrderItems)
		set @result = @countAfter - @countBefore
		select @result as result
	end try
	begin catch
		select -1 as result
	end catch
end

go
create procedure GetItemsFromCart
	@isApproved bit
as
begin
	if exists (select id from Orders where @isApproved = 0)
	begin
		select ord.isApproved,
			i.[Name],
```

```
                        i.Author,
                        d.[Data],
                        o.Price,
                        o.Quantity
                from QuantityOrderItems o
                join Orders ord on o.IdOrder = ord.Id
                join Items i on i.Id = o.IdItems
                join DataNanjing d on d.Id = i.IdDataLogo
                where d.[type] = 'image' and i.[Name] = d.[Name]
        end
        else
        begin
                print 'Товаров в корзине нету'
        end
end

go
create procedure DeleteItemsFromCart
        @id int
as
begin
        if exists (select count(1) from Orders where Id = @Id)
        begin
                delete from Orders where Orders.Id = @Id
        end
        else
                ROLLBACK TRANSACTION
                print 'Ошибка удаления в таблице Orders (ItemsFromCart)'
end



go
create procedure AddToOrdersHistory
        @ok bit,
        @Date date,
        @IdOrder int
as
begin
        begin try
                declare @countBefore int,
                        @countAfter int,
                        @result int
                set @countBefore = (select count(*) from OrdersHistory)
                if (@ok = 1)
                begin
                        insert into OrdersHistory ([Date],[IdOrder])
                        values ( @Date, @IdOrder)
                end
                else
                begin
                        rollback transaction
                end
                set @countAfter = (select count(*) from OrdersHistory)
                set @result = @countAfter - @countBefore
                select @result as result
        end try
        begin catch
                select -1 as result
        end catch
end
```

Приложение Г

```
use Nanjing;

drop trigger DataNanjing_Data

go
alter trigger DataNanjing_Data
	on DataNanjing
	after insert, update
as
begin
	declare
		@audio_id int,
		@book_id int,
		@video_id int,
		@audio_data_id int,
		@book_data_id int,
		@video_data_id int;

	if exists (select distinct Items.[Name]
			from Items, inserted
			where Items.[Name] = inserted.[Name]
			    and inserted.[type] = 'picture')
		begin
			update Items set Items.IdDataLogo = inserted.Id
			from inserted
			where inserted.[type] = 'picture'
				and Items.[Name] = inserted.[Name]
		end

	else if exists(select Items.[Name]
				from Items join inserted on Items.[Name] = inserted.[Name]
				where inserted.[type] = 'audio')
		begin
			set @audio_id = (select Items.Id
					from Items join inserted on Items.[Name] = inserted.[Name]
						where inserted.[type] = 'audio')
			set @audio_data_id = (select inserted.Id
							from items, inserted
							where inserted.[type] = 'audio'
								and Items.[Name] =
inserted.[Name]
								and Items.Id = @audio_id)
			if exists(select distinct Audio.IdItems
					from Items join inserted on Items.[Name] = inserted.[Name]
						join Audio on Items.Id = Audio.IdItems)
				begin
					update Audio set Audio.IdDataAudio = inserted.Id
					from items, inserted
					where inserted.[type] = 'audio'
						and Items.[Name] = inserted.[Name]
						and Items.Id = Audio.IdItems
				end
			else
				begin
					insert into Audio(IdItems, IdDataAudio) values (@audio_id,
@audio_data_id)
					print 'Не забудьте добавить данные в таблицу Audio'
				end
		end

	else if exists(select Items.[Name]
			from Items join inserted on Items.[Name] = inserted.[Name]
```

```
                              where inserted.[type] = 'video')
                begin
                        set @video_id = (select Items.Id
                                         from Items join inserted on Items.[Name] = inserted.[Name]
                                         where inserted.[type] = 'video')
                        set @video_data_id = (select inserted.Id
                                              from items, inserted
                                              where inserted.[type] = 'video'
                                                    and Items.[Name] = inserted.[Name]
                                                    and Items.Id = @video_id)
                        if exists(select distinct Movie.IdItems
                                  from Items join inserted on Items.[Name] = inserted.[Name]
                                               join Movie on Items.Id = Movie.IdItems)
                                begin
                                        update Movie set Movie.IdDataMovie = inserted.Id
                                        from items, inserted
                                        where inserted.[type] = 'video'
                                               and Items.[Name] = inserted.[Name]
                                               and Items.Id = Movie.IdItems
                                end
                        else
                                begin
                                        insert into Movie(IdItems, IdDataMovie) values (@video_id,
@video_data_id)
                                        print 'Не забудьте добавить данные в таблицу Movie'
                                end
                end

        else if exists(select Items.[Name]
                                  from Items join inserted on Items.[Name] = inserted.[Name]
                                  where inserted.[type] = 'text')
                begin
                        set @book_id = (select Items.Id
                                         from Items join inserted on Items.[Name] = inserted.[Name]
                                                    where inserted.[type] = 'text')
                        set @book_data_id = (select inserted.Id
                                             from items, inserted
                                             where inserted.[type] = 'text'
                                                    and Items.[Name] = inserted.[Name]
                                                    and Items.Id = @book_id)
                        if exists(select distinct Book.IdItems
                                        from Items join inserted on Items.[Name] = inserted.[Name]
                                                    join Book on Items.Id = Book.IdItems)
                                begin
                                        update Book set Book.IdDataBook = inserted.Id
                                        from items, inserted
                                        where inserted.[type] = 'text'
                                               and Items.[Name] = inserted.[Name]
                                               and Items.Id = Book.IdItems
                                end
                        else
                                begin
                                        insert into Book(IdItems, IdDataBook) values (@book_id,
@book_data_id)
                                        print 'Не забудьте добавить данные в таблицу Book'
                                end
                end
        else
                begin
                        ROLLBACK TRANSACTION
                        print 'Ошибка, нет данных в таблице Items'
                end
end
```

Приложение Д

```
use Nanjing

go
create procedure ImportFromXml
as
begin
	declare @xml xml
	select @xml = convert(xml, bulkcolumn, 2)
	from openrowset(bulk 'E:\Users.xml', single_blob)
	as X
	select @xml
		insert into Users
		select
			p.value('FName[1]', 'nvarchar(max)') as FName,
			p.value('SName[1]', 'nvarchar(max)') as SName,
			p.value('Login[1]', 'nvarchar(25)')  as [Login],
			p.value('Email[1]', 'nvarchar(max)') as [Email],
			p.value('Address[1]', 'nvarchar(max)') as [Address],
			p.value('Birthday[1]', 'date') as [Birthday],
			p.value('Password[1]', 'nvarchar(25)') as [Password],
			p.value('Phone[1]', 'nvarchar(12)') as [Phone],
			p.value('IdRole[1]', 'int') as [IdRole]
			FROM @xml.nodes('/Users/Users') PropertyFeed(P);
end


go
alter procedure ExportToXML
as
begin
	EXEC [Nanjing].dbo.sp_configure 'show advanced options', 1
	RECONFIGURE
	EXEC [Nanjing].dbo.sp_configure 'xp_cmdshell', 1
	RECONFIGURE

	EXEC xp_cmdshell 'bcp "use Nanjing SELECT
FName,SName,Login,Email,Address,Birthday,Password,Phone,IdRole FROM Users FOR XML PATH
(''User''), ROOT(''Users'')" queryout "E:\Users.xml" -U admin_Nanjing -P admin -S
HANNIKA\SQLEXPRESS -w -T'
end

execute ExportToXML
```

Приложение Е

```
use Nanjing

go
create procedure AddGenre
	@NameGanre nvarchar(max)
as
begin
	begin try
		declare @countBefore int,
			@countAfter int,
			@result int
		set @countBefore = (select count(*) from Book)
		insert into Genre([Name]) values (@NameGanre)
		set @countAfter = (select count(*) from Book)
		set @result = @countAfter - @countBefore
		select @result as result
	end try
	begin catch
		select -1 as result
		print 'Ошибка вставки данных в таблицу Genre. Такой жанр уже есть'
	end catch
end


go
create procedure AddPublisher
	@Name nvarchar(max),
	@Desc nvarchar(max)
as
begin
	begin try
		declare @countBefore int,
			@countAfter int,
			@result int
		set @countBefore = (select count(*) from Book)
		insert into Publisher(Name, Description)
			values (@Name, @Desc)
		set @countAfter = (select count(*) from Book)
		set @result = @countAfter - @countBefore
		select @result as result
	end try
	begin catch
		select -1 as result
		print 'Ошибка вставки данных в таблицу Publisher Такой издатель уже есть'
	end catch
end

go
create procedure GetAllGenre
as
begin
	select g.[Name] from Genre g
	order by Name desc
end

go
create procedure GetGenreById
	@Id int,
	@NameGanre nvarchar(max)
as
begin
	select g.[Name] from Genre g where g.Id = @Id
```

```
end


go
create procedure GetAllPublisher
as
begin
	select p.[Name], p.[Description] from Publisher p
	order by Name desc
end

go
create procedure GetPublisherById
	@Id int
as
begin
	select p.[Name], p.[Description] from Publisher p where p.Id = @Id
end

go
create procedure GetPublisherByName
	@Name nvarchar(max)
as
begin
	select p.[Name], p.[Description] from Publisher p where p.[Name] = @Name
end


go
create procedure UpdateGenre
	@Id int,
	@NameGanre nvarchar(max)
as
begin
	if exists (select count(1) from Genre where Genre.Id = @Id)
		begin
			update Genre set Genre.[Name] = @NameGanre
						where Genre.Id = @Id
		end
	else
	ROLLBACK TRANSACTION
	print 'Ошибка обновления данных в таблице Genre.'
end

go
create procedure UpdatePublisher
	@Id int,
	@Name nvarchar(max),
	@Desc nvarchar(max)
as
begin
	if exists (select count(1) from Publisher where Publisher.Id = @Id)
		begin
			update Publisher set Publisher.[Name] = @Name,
					Publisher.[Description] = @Desc
				where Publisher.Id = @Id
		end
	else
	ROLLBACK TRANSACTION
	print 'Ошибка обновления данных в таблице Publisher.'
end

go
create procedure DeleteGenre
```

```
	@Id int
as
begin
	if exists (select count(1) from Genre where Genre.Id = @Id)
	begin
		delete from ItemsGenres where ItemsGenres.IdGenre = @Id
		delete from Genre where Genre.Id = @Id
	end
	else
		ROLLBACK TRANSACTION
		print 'Ошибка удаления в таблицы Movie (proc DeleteMovie)'
end

go
create procedure DeletePublisher
	@Id int
as
begin
	if exists (select count(1) from Publisher where Publisher.Id = @Id)
	begin
		delete from Publisher where Publisher.Id = @Id
	end
	else
		ROLLBACK TRANSACTION
		print 'Ошибка удаления в таблицы Movie (proc DeleteMovie)'
end
```