МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация Базы данных

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«База данных онлайн-кинотеатра с применением технологии email уведомления о событиях базы данных»

Выполнил студент Сергунов Д. Ю.

(Ф.И.О.)

Руководитель проекта асс. Н.И. Уласевич

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

Консультант: асс. Н.И. Уласевич.

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

Нормоконтролер: асс. Н.И. Уласевич.

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

Курсовой проект защищен с оценкой

Содержание

[Введение 5](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984086)

[1 Постановка задачи 6](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984087)

[2 Проектирование базы данных 7](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984088)

[2.1 Определение вариантов использования 7](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984089)

[2.2 Диаграммы UML, взаимодействие всех компонентов 8](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984090)

[2.3 Вывод 9](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984091)

[3 Разработка объектов базы данных 10](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984092)

[3.1 Разработка таблиц базы данных 10](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984093)

[3.2 Разработка представлений базы данных 13](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984094)

[3.3 Разработка индексов базы данных 14](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984095)

[3.4 Разработка функций базы данных 15](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984096)

[3.5 Разработка процедур базы данных 16](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984097)

[3.6 Разработка триггеров базы данных 16](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984098)

[3.7 Разработка пользователей 17](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984099)

[3.8 Вывод 18](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984100)

[4 Описание процедур импорта и экспорта 19](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984101)

[5 Тестирование производительности 20](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984102)

[6 Описание технологии и ее применение в базе данных 22](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984103)

[7 Руководство пользователя 23](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984104)

[Заключение 24](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984105)

[Список используемых источников 25](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984106)

[Приложение А 26](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984107)

[Приложение Б 28](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984108)

[Приложение В 33](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984109)

[Приложение Г 35](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984110)

[Приложение Д 36](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Toc153984111)

# Введение

С развитием интернет-технологий и стремительным ростом развлекательной индустрии, онлайн-кинотеатры становятся неотъемлемой частью современного образа жизни, поскольку предоставляют пользователю удобный доступ к широкому спектру самых различных фильмов, сериалов, аниме и мультфильмов, непосредственно из уютного кресла своего дома.

Однако онлайн-кинотеатру необходимо хранить огромное количество данных, таких как фильмы, актёры, режиссёры, комментарии и прочую информацию, связанную с кинематографом, без которой он в принципе не может быть реализован. Для всего этого предназначена база данных.

База данных – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. Она необходима для долгосрочного хранения любых данных, управления ими, поиска и извлечения данных, а также для автоматизации процессов, связанных с реакцией базы данных на изменение своего внутреннего состояния, например для отправки email писем с информацией о новинках всем пользователям после добавления очередного фильма.

База данных используется практически во всех областях за счёт необходимости хранения совершенно разной информации, а также отсутствии конкурентного способа хранения и организации данных и следовательно, является важным этапом в разработке любого продукта.

# 1 Постановка задачи

Перед началом разработки необходимо определить цели и задачи проекта, а также должны быть определены функциональные требования.

Задачей проекта является проектирование и создание полноценной базы данных онлайн-кинотеатра, в которой должно осуществляться хранение такой информации как:

* информация о фильмах и сериалах, включая название, описание, жанр длительность, страну, актёров, режиссёра и студию;
* информация о профилях пользователей с персональными данными;
* информация о рейтингах и комментариях к фильмам.

Также необходимо реализовать систему email уведомлений о событиях базы данных, таких как добавление новых фильмов в таблицу базы данных и соответствующее уведомление пользователей о новинках.

Сама база данных должна быть спроектирована в СУБД «Microsoft SQL Server», а доступ к данным должен осуществляться только через соответствующие процедуры. Помимо этого, должен осуществляться импорт и экспорт данных из/в XML файлов.

Необходимо протестировать производительность базы данных и внести изменения в структуру в случае необходимости. Для этого в БД будут загружены не менее 100 000 строк для какой-либо таблицы и будет проверено время выполнения запросов с последующей оптимизацией.

# 2 Проектирование базы данных

## 2.1 Определение вариантов использования

При проектировании базы данных в первую очередь определяются роли пользователей и то, какие действия могут ими выполняться. На данном этапе важно ответить на 2 главных вопроса:

- Кто будет использовать базу данных?

- Как они будут взаимодействовать с ней?

Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны они могут получать, а также как должна быть организована навигация в системе.

Для подробного описания всех вариантов использования сервиса разрабатывается диаграмма вариантов использования, которая позволяет наглядно изобразить взаимодействие пользователя с системой. UML диаграмма вариантов использования представлена на рисунке 2.1

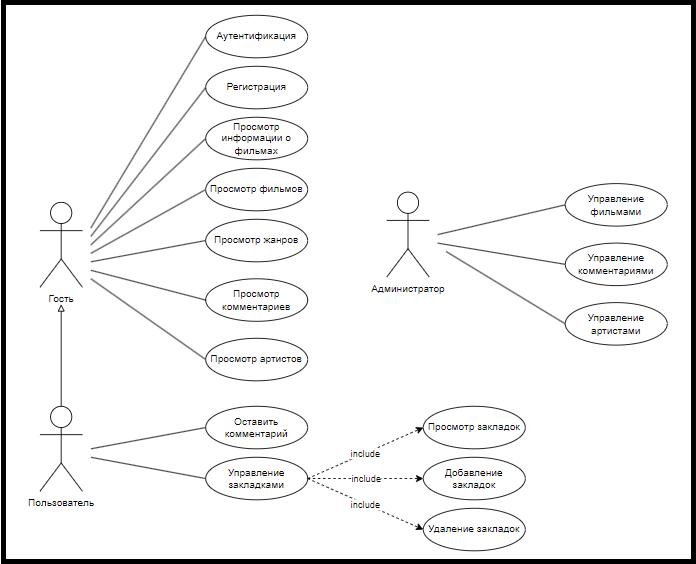


Рисунок 2.1 – UML диаграмма вариантов использования

По умолчанию лицо, которое использует сервис, может взаимодействовать с базой данной, как Guest (гость/неавторизованный пользователь). Данная роль даёт права на все основные функциональные возможности, за исключением оставления комментариев, оценкой фильмов и добавлением фильмов в закладки. Эти функции предоставляются пользователю после регистрации и аутентификации, одновременно с этим изменяя его роль с Guest на User (зарегистрированный пользователь).

Роль администратора (Administrator) может быть получена только путём её выдачи непосредственно системным администратором базы данных. Она предоставляет её владельцу дополнительно права на добавление и изменение информации в таблицы базы данных, такой как изменение информации о фильмах, модерация комментариев и т.д.

## 2.2 Логическая схема базы данных

Логическая схема базы данных является абстрактным описание того, как данные организованы и как они связаны друг с другом в базе данных. Она не привязана к конкретной технической реализации и представляет собой основу для создания физической структуры базы данных и запросов к данным. На этом уровне проектирования описываются таблицы, их столбцы и связи между таблицами. Логическая схема базы данных представлена на рисунке 2.2.

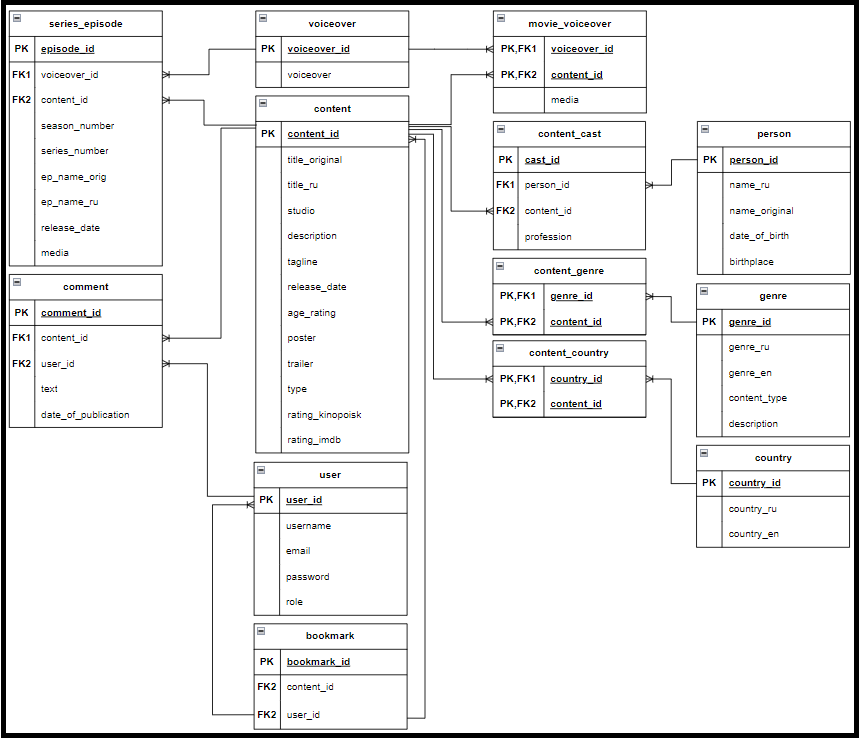


Рисунок 2.2 – Логическая схема базы данных

Данная диаграмма отображает все таблицы базы данных и связи между ними. Только после её разработки можно приступать к реализации базы данных.

## 2.3 Вывод

Проектирование, это неотъемлемая и самая важная часть разработки базы данных поскольку именно на ней определяются все основные элементы, связи между ними, а также их свойства. Важность проектирования базы данных проявляется в нескольких аспектах. Во-первых, хорошо спроектированная база данных обеспечивает целостность данных, предотвращая их потерю или искажение. Это особенно важно в сферах использования базы данных, поскольку здесь как нигде необходима точность информации. Во-вторых, правильное проектирование базы данных способствует эффективному доступу к данным. Оптимизированные структуры данных позволяют быстро извлекать нужную информацию, улучшая производительность системы и удовлетворяя потребностям пользователей

# 3 Разработка объектов базы данных

## 3.1 Таблицы

Таблицы являются основной частью базы данных, поскольку именно в них хранится вся информация. Для базы данных онлайн-кинотеатра было разработано 13 таблиц, назначение которых указано в таблице 3.1. Листинг SQL-кода для создания таблиц находится в приложении A.

Таблица 3.1 – Описание таблиц базы данных

|  |  |
| --- | --- |
| Название | Описание |
| user | Хранит информацию о пользователях |
| content | Хранит информацию о фильмах |
| person | Хранит информацию о артистах |
| genre | Хранит информацию о жанрах |
| country | Хранит информацию о странах |
| voiceover | Хранит информацию об озвучках |
| series\_epsiode | Хранит информацию об эпизодах многосерийных шоу |
| comment | Хранит информацию о комментариях к фильмам |
| bookmark | Хранит информацию о закладках |
| content\_cast | Связующая таблица, хранящая информацию о касте фильмов |
| content\_genre | Связующая таблица, хранящая информацию о жанрах фильмов |
| content\_country | Связующая таблица, хранящая информацию о странах производства фильмов |
| movie\_voiceover | Связующая таблица, хранящая озвучку полнометражных картин |

После описания назначения всех таблиц базы данных, необходимо остановиться на каждой из них подробнее. Описания столбцов всех таблиц представлены в таблицах 3.2 – 3.14:

Таблица 3.2 – Столбцы таблицы «content»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| content\_id | INT | Хранит уникальный идентификатор фильма |
| title\_original | VARCHAR | Хранит название фильма на родном языке |
| title\_ru | VARCHAR | Хранит название фильма на русском языке |
| studio | VARCHAR | Хранит название студии, сделавшей фильм |
| description | VARCHAR | Хранит описание фильма |
| tagline | VARCHAR | Хранит слоган фильма |
| release\_date | DATE | Хранит дату выхода фильма |
| age\_rating | VARCHAR | Хранит возрастной рейтинг фильма |
| poster | VARCHAR | Хранит ссылку на постер фильма |
| trailer | VARCHAR | Хранит ссылку на трейлер фильма |
| type | VARCHAR | Хранит тип контента |

Таблица 3.3 – Столбцы таблицы «country»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| country\_id | INT | Хранит уникальный идентификатор страны |
| country\_ru | VARCHAR | Хранит название страны на русском языке |
| country\_en | VARCHAR | Хранит название страны на английском языке |

Таблица 3.4 – Столбцы таблицы «person»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| person\_id | INT | Хранит уникальный идентификатор артиста |
| name\_original | VARCHAR | Хранит имя артиста на родном языке |
| name\_ru | VARCHAR | Хранит имя артиста на русском языке |
| date\_of\_birth | DATE | Хранит дату рождения артиста |
| birthplace | VARCHAR | Хранит место рождения артиста |

Таблица 3.5 – Столбцы таблицы «genre»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| genre\_id | INT | Хранит уникальный идентификатор жанра |
| genre\_ru | VARCHAR | Хранит названия жанра на русском языке |
| genre\_en | VARCHAR | Хранит названия жанра на английском языке |
| content\_type | VARCHAR | Хранит тип контента жанра |
| description | VARCHAR | Хранит описание жанра |

Таблица 3.6 – Столбцы таблицы «user»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| user\_id | INT | Хранит уникальный идентификатор пользователя |
| username | VARCHAR | Хранит имя пользователя |
| email | VARCHAR | Хранит email пользователя |
| password | VARCHAR | Хранит пароль пользователя |
| role | VARCHAR | Хранит роль пользователя |

Таблица 3.7 – Столбцы таблицы «voiceover»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| voiceover\_id | INT | Хранит уникальный идентификатор озвучки |
| voiceover | VARCHAR | Хранит название озвучки |

Таблица 3.8 – Столбцы таблицы «series\_episode»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| episode\_id | INT | Хранит уникальный идентификатор эпизода |
| voiceover\_id | INT | Хранит идентификатор привязанной к серии озвучки |
| content\_id | INT | Хранит идентификатор привязанного к серии контента |
| ep\_name\_orig | VARCHAR | Хранит название эпизода на родном языке |
| ep\_name\_ru | VARCHAR | Хранит название эпизода на русском языке |
| series\_number | INT | Хранит номер эпизода |
| season\_number | INT | Хранит номер сезона |
| release\_date | DATE | Хранит дату выхода эпизода |
| media | VARCHAR | Хранит ссылку на медиафайл эпизода |

Таблица 3.9 – Столбцы таблицы «bookmark»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| bookmark\_id | INT | Хранит уникальный идентификатор закладки |
| content\_id | INT | Хранит идентификатор привязанного контента |
| user\_id | INT | Хранит идентификатор привязанного пользователя |

Таблица 3.10 – Столбцы связующей таблицы «content\_cast»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| cast\_id | INT | Хранит уникальный идентификатор состава |
| person\_id | INT | Хранит идентификатор привязанного артиста |
| content\_id | INT | Хранит идентификатор привязанного контента |
| proffesion | VARCHAR | Хранит профессию артиста в конкретном фильме |

Таблица 3.11 – Столбцы таблицы «comment»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| comment\_id | INT | Хранит уникальный идентификатор комментария |
| content\_id | INT | Хранит идентификатор привязанного контента |
| user\_id | INT | Хранит идентификатор привязанного пользователя |
| text | VARCHAR | Хранит текст комментария |
| publication\_date | DATE | Хранит дату оставления комментария |

Таблица 3.12 – Столбцы связующей таблицы «content\_genre»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| genre\_id | INT | Хранит идентификатор привязанного жанра |
| content\_id | INT | Хранит идентификатор привязанного контента |

Таблица 3.14 – Столбцы связующей таблицы «movie\_voiceover»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| voiceover\_id | INT | Хранит идентификатор привязанной озвучки |
| content\_id | INT | Хранит идентификатор привязанного контента |
| media | VARCHAR | Хранит ссылку на медиафайл фильма |

После описания всех таблиц базы данных, необходимо понять как они связаны друг с другом. Описание их связи представлено в таблице 3.15

Таблица 3.15 – Связи между таблицами базы данных

|  |  |  |
| --- | --- | --- |
| Таблица PK | Таблица FK | Описание связи |
| content | content\_person | Один ко многим |
| content | content\_genre | Один ко многим |
| content | content\_country | Один ко многим |
| content | movie\_voiceover | Один ко многим |
| content | series\_episode | Один ко многим |
| content | comment | Один ко многим |
| content | bookmark | Многие к одному |
| person | content\_person | Один ко многим |
| genre | content\_genre | Один ко многим |
| country | content\_country | Один ко многим |
| voiceover | movie\_voiceover | Один ко многим |
| voiceover | series\_episode | Один ко многим |
| user | bookmark | Многие к одному |
| user | comment | Один ко многим |

## 

## 3.2 Разработка представлений базы данных

Представление в базе данных – это виртуальная таблица, которая формируется на основе выполнения запроса к одной или нескольким таблицам в базе данных. Представления позволяют объединять данные из нескольких таблиц в одну, не изменяя исходных таблиц и их структуры. Представления используются для обеспечения удобства доступа и управления данными в базе данных, а также для улучшения производительности запросов. Всего было создано 8 представлений:

* get\_films, представление, которое возвращает список всех фильмов и информации о них, необходимой для корректного отображения;
* get\_new\_films, представление, которое возвращает список всех новых фильмов и информации о них, необходимой для корректного отображения;
* get\_series, представление, которое возвращает список всех сериалов и информации о них, необходимой для корректного отображения;
* get\_new\_series, представление, которое возвращает список всех новых сериалов и информации о них, необходимой для корректного отображения;
* get\_anime, представление, которое возвращает список всех аниме и информации о них, необходимой для корректного отображения;
* get\_new\_anime, представление, которое возвращает список всех новых аниме и информации о них, необходимой для корректного отображения.
* get\_cartoons, представление, которое возвращает список всех мультфильмов и информации о них, необходимой для корректного отображения;
* get\_new\_cartoons, представление, которое возвращает список всех новых мультфильмов и информации о них, необходимой для корректного отображения.

Реализация нескольких из представлений представлено в листинге 3.1. Остальные представления будут аналогичны, работая с другими данными. Листинг SQL-кода с созданием представлений находится в приложении Б

|  |
| --- |
| CREATE OR ALTER VIEW get\_films AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where [type] = 'Film';  go  CREATE OR ALTER VIEW get\_new\_films AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where release\_date > DATEADD(MONTH, -6, GETDATE()) and [type] = 'Film';  go |

Листинг 3.1 – Представления get\_films и get\_new\_films

Представление get\_films соединяет в себе данные из главной таблицы фильма content, а также 1 жанр из таблицы genre и 1 страну из таблицы country, для размещения этой информации в карточке фильма.

В свою очередь представление get\_new\_films является расширением базового представления, дополняя его лишь условием, при котором дата выхода фильма не должна быть позднее чем за 6 месяцев.

## 3.3 Разработка индексов базы данных

Индекс в базе данных представляет собой объект, который используется для ускорения поиска данных. Если таблица содержит большое количество строк, то последовательный поиск данных может занимать много времени. Индекс создается на основе значений одного или нескольких столбцов таблицы и указывает на соответствующие строки таблицы. Использование индексов помогает улучшить производительность базы данных, поскольку они имеют оптимизированную структуру для поиска, например, сбалансированное дерево. Один из наиболее распространенных алгоритмов индексации – это B-дерево (B-tree). B-дерево – это сбалансированное дерево поиска, которое обеспечивает эффективный поиск элементов, используя ключи для сортировки данных в индексе. Часть индексов представлена на листинге 3.2.

|  |
| --- |
| CREATE INDEX IDX\_content\_type ON content(type);  CREATE INDEX IDX\_content\_release\_date ON content(release\_date);  CREATE INDEX IDX\_content\_rating\_imdb ON content(rating\_imdb);  CREATE INDEX IDX\_content\_rating\_kinopoisk ON content(rating\_kinopoisk); |

Листинг 3.2 – Индексы таблицы content

Кроме индексов, представленных выше, в базе данных присутствуют индексы во всех оставшихся таблицах.

Применение индексов в базе данных значительно повышает скорость выполнения операций поиска, сортировки и фильтрации данных, особенно в случае больших объемов информации. Однако важно помнить, что создание индексов – это всегда компромисс между производительностью при выполнении запросов и накладными расходами на поддержку этих индексов при вставке, обновлении и удалении данных.

## 3.4 Разработка функций базы данных

Для управления данными через приложение все пользователи использует функции. Функция – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Стоит отметить, что функции не могут содержать группы операторов DML и DDL. Функции в SQL Server могут быть использованы для выполнения различных задач, таких как вычисления, преобразования данных, выполнения операций над строками и датами, а также других распространённых операций. Функции обеспечивают модульность и повторное использование кода, что упрощает разработку и обслуживание базы данных. В данном проекте, функции будут использоваться в качестве способа контроля доступа к определённым данным, а именно заменяя непосредственное обращение к базе данных через select-запрос, на обращение к самим функциям, контролирующими все проходящие через пользователей данные. На листинге 3.3 представлена одна из функций, используемых в проекте.

|  |
| --- |
| CREATE OR ALTER FUNCTION dbo.GetOrderedContent  (  @type VARCHAR(MAX)  )  RETURNS TABLE  AS  RETURN  (  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb,  row\_number() over (ORDER BY content.rating\_imdb desc) as [Место в рейтинге]  FROM content  where [type] = @type  );  go  select \* from GetOrderedContent('Anime')  go |

Листинг 3.3 – Функция GetOrderedContent

Функция GetOrderedContent, представленная выше, принимает один параметр @type типа VARCHAR(MAX), на основе которого выбирается какой тип контента будет возвращён (аниме, фильмы, сериалы, мультфильмы). Данная функция делает select-запрос к базе данных с нужным условием, после чего фильтрует результат по популярности на основе рейтинга imdb и возвращает таблицу. Другие функции, использованные в проекте, представляют из себя следующее

GetContentByGenre – получение необходимого контента в соответствии не только с типом, а уже и с определённым жанром, этого самого контента.

GetContentByYear – возврат таблицы со списком произведений определённого типа, выпущенным в указанный год.

GetContentByGenreAndYear – комбинация двух предыдущих функций, а именно таблица, чей результат основан на фильтрации по определённому типу, жанру, и году одновременно.

GetContentActors – функция позволяющая получить список актёров конкретного произведения, благодаря select-запросу объединяющему несколько таблиц.

GetContentDirector – функция позволяющая получить режиссёоа конкретного произведения, работает по такому же принципу, как и GetContentActors.

GetContentComments – функция, чья задача заключается в возврате списка комментариев, оставленных под произведением.

GetBookMarks – функция позволяющая зарегистрированным пользователям получить список собственных закладок. Непосредственное добавление закладок будет реализовано в следующей главе с помощью процедур.

Код всех разработанных функций представлен в приложении В

## 3.5 Разработка процедур базы данных

Процедура – это сохранённый фрагмент T-SQL кода, который может выполняться из раза в раз для выполнения определённой задачи или серии задач. По большому счёту, процедуры выполняют аналогичные действия с функциями, за тем исключением, что они не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. Это означает что их использование является оптимальным в качестве связующего звена между базой данных и пользователями. Всего было разработано мною 34 процедуры, позволяющие взаимодействовать с базой данных. В листинге 3.4 представлена реализация одной из процедур.

|  |
| --- |
| CREATE PROCEDURE dbo.AddBookmark  @contentId INT,  @userId INT  AS  BEGIN  INSERT INTO bookmark (content\_id, [user\_id])  VALUES (@contentId, @userId);  END;  go |

Листинг 3.4 – Процедура создания закладки

Данная процедура параметрами принимает 2 значения типа INT, а именно уникальный идентификатор фильма, желаемого добавить в закладки, а также уникальный идентификатор пользователя, которому закладка и принадлежит, после чего вставляет новую строку в соответствующую таблицу, хранящую закладки.

## 3.6 Разработка триггеров базы данных

Триггер в базе данных – это объект, который автоматически выполняет определенные действия при возникновении определенных событий в таблице или представлении базы данных. Триггер может быть настроен на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для поддержки целостности данных, контроля доступа к данным и автоматической обработки данных при выполнении определенных операций в таблице.

В листинге 3.5 представлена реализация одного из триггеров.

|  |
| --- |
| CREATE TRIGGER CheckBookmarkCount  ON bookmark  AFTER INSERT, UPDATE  AS  BEGIN  DECLARE @userId INT;  SELECT @userId = [user\_id] FROM inserted;  IF (SELECT COUNT(\*) FROM bookmark WHERE [user\_id] = @userId) > 100  BEGIN  Print('У одного пользователя не может быть больше 100 закладок');  ROLLBACK TRANSACTION;  END  END; |

Листинг 3.5 – Скрипт триггера CheckBookmarkCount

Данный триггер проверяет перед очередной вставкой или обновлением данных, находится ли у пользователя более 100 закладок, и если это так отменяет транзакцию, тем самым не позволяя добавить ещё одну закладку.

## 3.7 Разработка пользователей

Пользователь базы данных – это учётная запись в системе управления базами данных (СУБД), который имеет право доступа к определённым данным и функциям в базе данных. Каждый пользователь имеет права доступа, которые определяют его возможности в рамках базы данных и которые могут включать в себя права на чтение, запись, обновление и удаление данных, а также выполнении определённых операций и функций в БД.

Понятие пользователя базы данных неотъемлимо связано с понятие роли. Роль в базе данных представляет собой именованный набор прав доступа для определённой группы пользователей. Это упрощает администрирование и поддержку системы, поскольку нет необходимости выдавать вручную каждому пользователю определённые права, вместо это можно определить права для роли, а уже роль присваивать пользователям.

При проектировании базы данных было определено 3 пользовательских роли, то есть группы людей, которые имеют права на подключение к ней: администратор, пользователь, гость (незарегистрированный пользователь). После чего было создано 3 пользователя, получивших данные роли с присущими им правами. Создание роли, выдача ей прав, а также создание пользователя с последующей выдачей роли указано в листинге 3.4.

|  |
| --- |
| CREATE ROLE Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON content TO Administrator;  GRANT EXECUTE ON dbo.DeleteContent TO Administrator;  GRANT SELECT ON dbo.GetOrderedContent TO Administrator;  GRANT SELECT ON dbo.GetContentByGenre TO Administrator;  GRANT SELECT ON dbo.GetContentByYear TO Administrator;  GRANT SELECT ON dbo.GetContentActors TO Administrator;  GRANT SELECT ON dbo.GetContentDirector TO Administrator;  GRANT SELECT ON dbo.GetContentComments TO Administrator;  GRANT EXECUTE ON dbo.DeleteContent TO Administrator;  GRANT EXECUTE ON dbo.AddContent TO Administrator;  GRANT SELECT ON dbo.GetContentByGenreAndYear TO Administrator;  CREATE LOGIN AdminUser WITH PASSWORD = 'AdminPassword';  CREATE USER AdminUser FOR LOGIN AdminUser;  ALTER ROLE Administrator ADD MEMBER AdminUser; |

Листинг 3.4 – Создание роли, пользователя и выдача прав

В данном примере создаётся роль администратора, которой даётся разрешение на любые действия с таблицей content, а также разрешение на вызов функции DeleteContent для удаления художественного произведения из базы данных, после чего создаётся пользователь AdminUser, которому выдаётся соответствующая роль, после чего он становится полноправным владельцем таблицы. Полный код создания ролей, выдачи им прав, а также создании пользователей находится в приложении Г.

## 3.8 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для онлайн-кинотеатра. Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе. Благодаря созданию индексов, база данных стала существенно быстрее работать, разработанные функции позволяют взаимодействовать пользователю с системой, являясь посредниками при выполнении команд, обеспечивая тем самым целостность данных и безопасность всей базы данных. Разработанные процедуры позволяют непосредственно уже такие действия как добавление, обновление и удаление данных в базе без необходимости написания лишних запросов. Так же огромную роль сыграли триггеры, которые играют роль ограничений накладываемых на возможности пользователей.

# 4 Описание импорта и экспорта данных

Для таблицы Лоты в базе данных доступна функция экспорта и импорта данных в формате JSON. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в JSON, была разработана функция, результатом которой стало создание JSON файла, с записанными строками из определенной таблицы в JSON формат. Скрипт запроса представлен в листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE FUNCTION EXPORT\_LOTS\_TO\_JSON\_FILE(FILE\_PATH TEXT)  RETURNS VOID AS  $$  DECLARE  JSON\_DATA JSON;  BEGIN  BEGIN  SELECT JSON\_AGG(ROW\_TO\_JSON(Лоты)) INTO JSON\_DATA FROM Лоты;  PERFORM PG\_FILE\_WRITE(FILE\_PATH, JSON\_DATA::TEXT,true);  EXCEPTION WHEN OTHERS THEN  RAISE 'Произошла ошибка: %', SQLERRM;  END;  END;  $$  LANGUAGE PLPGSQL; |

Листинг 4.1 – Функция EXPORT\_LOTS\_TO\_JSON\_FILE

Для обратного экспорта данные в JSON формат из файла, реализован запросом, который обращается к JSON файлу, и выбирает данные в обычные строки, это можно реализовать с помощью функции, либо по мере надобности записать данные во временную таблицу, либо таблицу в другой базе данных. Скрипт запроса представлен в [приложении Г](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Приложение_Г).

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах JSON, что делает управление базой данных более удобным и эффективным.

# 5 Тестирование производительности

Одной из ключевых задач в процессе разработки является тестирование производительности базы данных. Такое тестирование позволяет оценить способность базы данных обрабатывать запросы и возвращать результаты с высокой скоростью. Оценка производительности важна для определения эффективности базы данных и выявления возможных узких мест, которые могут вызывать задержки или проблемы в работе приложения.

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Для данной задачи была разработана соответствующая процедура, которая могла вставила 100000 строк в таблицу content, содержащую в себе художественные произведения

|  |
| --- |
| DECLARE @counter INT = 1;  WHILE @counter <= 100000  BEGIN  INSERT INTO content (title\_original, title\_ru, studio, [description], tagline, release\_date, age\_rating, poster, trailer, [type], rating\_imdb, rating\_kinopoisk)  VALUES (  dbo.GenerateRandomString(50), -- title\_original  dbo.GenerateRandomString(50), -- title\_ru  dbo.GenerateRandomString(50), -- studio  dbo.GenerateRandomString(500), -- [description]  dbo.GenerateRandomString(50), -- tagline  DATEADD(day, -CAST(RAND() \* 365 \* 30 AS INT), GETDATE()), -- release\_date (последние 30 лет)  CASE  WHEN RAND() < 0.2 THEN '0+'  WHEN RAND() < 0.4 THEN '6+'  WHEN RAND() < 0.6 THEN '12+'  WHEN RAND() < 0.8 THEN '16+'  ELSE '18+'  END, -- age\_rating  dbo.GenerateRandomString(255), -- poster  dbo.GenerateRandomString(255), -- trailer  CASE  WHEN RAND() < 0.25 THEN 'Film'  WHEN RAND() < 0.5 THEN 'Cartoon'  WHEN RAND() < 0.75 THEN 'Series'  ELSE 'Anime'  END, -- [type]  RAND() \* 10, -- rating\_imdb  RAND() \* 10 -- rating\_kinopoisk  );  SET @counter = @counter + 1;  END;  go |

Листинг 5.1 – Заполнение таблицы

Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| SET STATISTICS TIME ON;  SELECT title\_ru  FROM content  WHERE title\_ru = 'Фильм44444';  SET STATISTICS TIME OFF; |

Листинг 5.2 – Запрос к таблице content

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составило 41 мс, а время планирования – 0.506 мс. Результаты запроса будут представлены на рисунке 5.1.

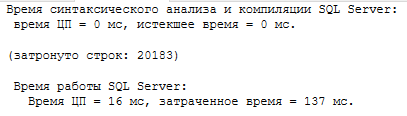


Рисунок 5.1 – Результат выполнения запроса

Для более детальной информации необходимо просмотреть предполагаемый план выполнения, представленный на рисунке 5.2

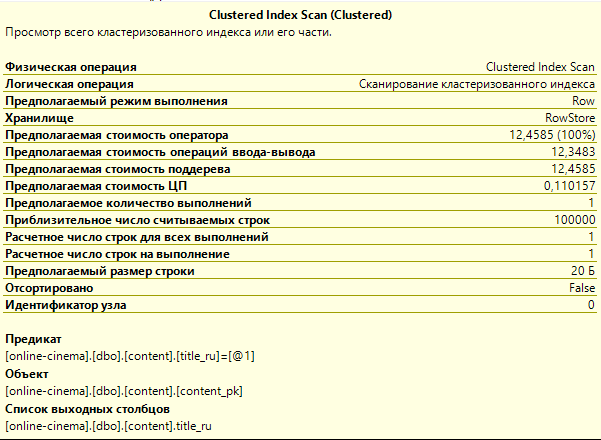


Рисунок 5.2 – Предполагаемый план выполнения

Для ускорения данного процесса был создан индекс на поле rating\_imdb так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 5.3.

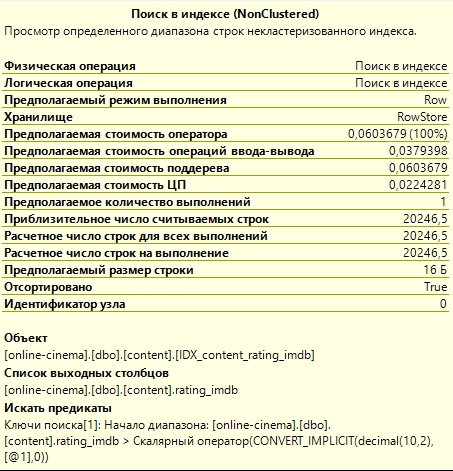


Рисунок 5.3 – Предполагаемый план выполнения после добавления индекса

Результаты тестирования говорят о том, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных. Поэтому было принято решение и для других данных.

1. **Описание технологии и ее применения в базе данных**

Рассылка уведомлений - очень удобный инструмент отслеживания выполнения заданий в SQL Server. Настройка таких рассылок состоит из нескольких шагов. Для этого открываем Management Studio, заходим в «Управление» и нажимаем правой кнопкой мыши по пункту «Компонент Database Mail», затем в появившемся меню щелкаем по пункту «Настроить компонент Database Mail». Для создания учетной записи SMTP указываем e-mail, с которого будут приходить уведомления, и настройки сервера исходящей почты. После чего в списке учетных записей SMTP отобразится только что созданная учетная запись, жмем «Далее». Работу компоненты Database Mail можно проверить, отправив тестовое сообщение.

Database Mail появился в SQL Server 2005, заменив собой компонент SQLMail. Database Mail (как ранее SQLMail) используется для отправки сообщений электронной почты самим SQL Server'ом (точнее, его компонентом Database Engine). Коренным отличием Database Mail от SQLMail является то, что последний, для отправки сообщений, использовал стороний клиент (Microsoft Outlook), который должен был быть установлен на машине с SQL Server'ом, а Database Mail сам общается с почтовым сервером по протоколу SMTP.

Для настройки в командную строку PowerShell вводились следующие данные, которые представлены в листинге 6.1.

|  |
| --- |
| $EmailFrom = "ssjjitt@mail.ru"  $EmailPass = "f7T8Pw81sexdrP4yLHJ9"  $EmailTo = "ssjjitt@mail.ru"  $Port = 587  $Subject = "Test From PowerShell"  $Body = "Did this work?"  $SMTPServer = "smtp.mail.ru"  $SMTPClient = New-Object Net.Mail.SmtpClient($SmtpServer, $Port)  $SMTPClient.EnableSsl = $true  $SMTPClient.Credentials = New-Object System.Net.NetworkCredential($EmailFrom, $EmailPass);  $SMTPClient.Send($EmailFrom, $EmailTo, $Subject, $Body) |

Листинг 6.1 – код для строки PowerShell

Письма на отправляются на почту mail.ru через процедуру SaveTableDataToJsonAndSend, письмо содержит файл JSON с переданной таблицей. Листинг процедуры представлен в листинге 6.2.

|  |
| --- |
| CREATE OR ALTER PROCEDURE SaveTableDataToJsonAndSend  @tableName NVARCHAR(128),  @recipients NVARCHAR(100),  @subject NVARCHAR(255),  @body NVARCHAR(MAX)  AS  BEGIN  DECLARE @fileName NVARCHAR(128);  SET @fileName = REPLACE(@tableName, ' ', '') + '.json';  DECLARE @sql NVARCHAR(1000);  SET @sql = 'bcp "SELECT \* FROM ' + QUOTENAME(@tableName) + ' ' +  'FOR JSON PATH, INCLUDE\_NULL\_VALUES" ' +  'queryout "e:\' + @fileName + '" ' +  '-c -S DESKTOP-F644G2I -d MobileBank -T';  EXEC sys.XP\_CMDSHELL @sql;  DECLARE @attachment NVARCHAR(255);  SET @attachment = 'e:\' + @fileName;  -- Отправка письма с вложением  EXEC msdb.dbo.sp\_send\_dbmail  @profile\_name = 'database profile',  @recipients = @recipients,  @subject = @subject,  @body = @body,  @file\_attachments = @attachment;  END; |

Листинг 6.2 – код для отправки письма на почту

Процедура генерирует имя файла для сохранения данных в формате JSON. Имя файла формируется путем удаления пробелов из имени таблицы и добавления расширения json. Формирует динамический SQL запрос для получения данных из таблицы @tableName в формате JSON с использованием FOR JSON PATH, INCLUDE\_NULL\_VALUES. Результаты запроса сохраняются в файле. Вызывает команду XP\_CMDSHELL сформированным SQL запросом. Это выполняет команду bcp (Bulk Copy Program) для выполнения SQL-запроса и сохранения результатов в файле. Задает путь к сохраненному файлу как вложение для отправки письма. Использует системную процедуру sp\_send\_dbmail для отправки письма с вложением. Параметры письма включают имя профиля отправителя, список получателей, тему, текст письма и путь к вложенному файлу.

# 7 Руководство пользователя

При входе на аукционную площадку, мы будем иметь статус гостя, это значит – ограниченный функционал. Мы сможем лишь просмотреть, какие лоты доступны на аукционе в данный момент (представление num\_lots\_on\_auction), поиск лотов (функция get\_lots\_list) и последняя ставка, которая произведена на лот (представление last\_bid\_for\_lot). Главная функция у гостя – регистрация (процедура Add\_user) и авторизация (функция authenticate\_user).

После регистрации и последующей авторизации, пользователь уже имеет одноименный статус – Пользователь. У него открываются новые возможности, он может создавать лоты для последующих аукционов на них (функция create\_lot), также он имеет возможность участвовать в других аукционах (процедура create\_bid, фунцкия increase\_bid), посмотреть на статистику аукциона (функции get\_active\_auctions, get\_max\_bid\_for\_auction, get\_max\_bid\_for\_auction, get\_bids\_for\_lot, get\_auctions\_for\_user, get\_users\_with\_won\_lots).

Если вход на площадку был произведен с аккаунта менеджера, то мы сможем просматривать и редактировать все лоты и аукционы других пользователей, однако сами создать что-то мы не можем (процедуры update\_auction\_status, update\_user, edit\_auction, update\_lot; функции check\_bid).

Вход с учетной записью администратора дает нам полное управление базой данных аукционной площадки.

Реализация каждой из вышеупомянутой роли приведена в [приложении Д](file:///C:\Users\sergu\Downloads\Telegram%20Desktop\Пояснительная_БД_Трубач.docx#_Приложение_Д).

# Заключение

В данном проекте была поставлена задача разработки базы данных для площадки аукциона с использованием технологии резервного копирования и восстановления в СУБД SQL server, а также применением дополнительной технологии email уведомлений о событиях базы данных.

В ходе выполнения проекта были использованы различные объекты, такие как таблицы, представления, процедуры, индексы триггеры, функции, специальные функции, чтобы обеспечить структурированное хранение данных и удобный доступ к ним. В итоге, поставленная цель была успешно достигнута, и на выходе получена готовая база данных.

В рамках тестирования базы данных был использован поток данных, и результаты тестирования оказались положительными. Также были разработаны процедуры импорта и экспорта данных в формате JSON, что значительно облегчило управление данными и повысило удобство использования базы данных.

Разработанная база данных для площадки аукциона имеет ряд важных особенностей, которые позволяют эффективно управлять ей. Одной из главных особенностей является технология резервного копирования и восстановления, которая обеспечивает сохранность данных в случае любых сбоев или ошибок. Это означает, что все данные аукциона, включая информацию о пользователях, лотах, ставках и аукционах, могут быть быстро восстановлены, минимизируя потерю информации и обеспечивая непрерывность работы площадки.

Кроме того, база данных обладает гибкой системой привилегий, которая позволяет разграничивать доступ к различным функциям и данным в зависимости от роли пользователя. Это обеспечивает безопасность данных и предотвращает несанкционированный доступ.

Делая вывод, можно сказать, что поставленные цели были выполнены. База данных для аукционной площадки выполняет необходимую функцию хранения и управления данными.

# Список используемых источников

1. eBay [Электронный ресурс] – https://www.ebay.com/ – Дата доступа 27.09.2023
2. Amazon Liquidation Auctions [Электронный ресурс] – https://bstock.com/auctions/amazon/ – Дата доступа 28.09.2023
3. Ay.by Аукционы Беларуси [Электронный ресурс] – http://ay.by/ – Дата доступа 28.09.2023
4. PostgreSQL Сайт с информацией [Электронный ресурс] – https://postgrespro.ru/docs/postgresql.com/ – Дата доступа 10.10.2023
5. Технология резервного копирования и восстановления в PostgreSQL [Электронный ресурс] – https://postgrespro.ru/docs/postgresql/9.6/backup-dump/ – Дата доступа 13.11.2023

# Приложение А. Создание таблиц

|  |
| --- |
| CREATE TABLE content (  content\_id INT IDENTITY(1,1),  title\_original VARCHAR(255),  title\_ru VARCHAR(255),  studio VARCHAR(255),  [description] VARCHAR(MAX) NOT NULL,  tagline VARCHAR(255),  release\_date DATE,  age\_rating VARCHAR(255) NOT NULL,  poster VARCHAR(255) NOT NULL,  trailer VARCHAR(255) NOT NULL,  [type] VARCHAR(255) NOT NULL CHECK ([type] in ('Film', 'Cartoon', 'Series', 'Anime')),  rating\_imdb DECIMAL(10,2),  rating\_kinopoisk DECIMAL(10,2),  CONSTRAINT content\_pk PRIMARY KEY(content\_id),  CONSTRAINT content\_age\_rating\_check CHECK (age\_rating in ('0+', '6+', '12+', '16+', '18+')),  CONSTRAINT content\_type\_check CHECK ([type] in ('Film', 'Cartoon', 'Series', 'Anime'))  );  CREATE TABLE voiceover (  voiceover\_id INT IDENTITY(1,1),  voiceover VARCHAR(255) NOT NULL,  CONSTRAINT voiceover\_pk PRIMARY KEY(voiceover\_id),  CONSTRAINT voiceover\_unique UNIQUE (voiceover)  );  CREATE TABLE series\_episode (  episode\_id INT IDENTITY(1,1),  content\_id INT NOT NULL,  voiceover\_id INT NOT NULL,  season\_number INT NOT NULL,  series\_number INT NOT NULL,  ep\_name\_orig VARCHAR(255),  ep\_name\_ru VARCHAR(255),  release\_date DATE,  media VARCHAR(255),  CONSTRAINT series\_episode\_pk PRIMARY KEY(episode\_id),  CONSTRAINT series\_episode\_content\_fk FOREIGN KEY (content\_id) REFERENCES content (content\_id),  CONSTRAINT series\_episode\_voiceover\_fk FOREIGN KEY (voiceover\_id) REFERENCES voiceover (voiceover\_id),  CONSTRAINT series\_episode\_unqiue\_combination UNIQUE (content\_id, voiceover\_id, season\_number, series\_number)  );  CREATE TABLE person (  person\_id INT IDENTITY(1,1),  name\_ru VARCHAR(255) NOT NULL,  name\_original VARCHAR(255) NOT NULL,  date\_of\_birth DATE,  birthplace VARCHAR(255),  CONSTRAINT person\_fk PRIMARY KEY(person\_id),  );  CREATE TABLE genre (  genre\_id INT IDENTITY(1,1),  genre\_ru VARCHAR(255) NOT NULL,  genre\_en VARCHAR(255) NOT NULL,  content\_type VARCHAR(255) NOT NULL,  [description] VARCHAR(255),  CONSTRAINT genre\_pk PRIMARY KEY(genre\_id),  CONSTRAINT genre\_check\_content\_type CHECK (content\_type in ('Film', 'Cartoon', 'Series', 'Anime')),  CONSTRAINT genre\_unique\_index\_ru UNIQUE (genre\_ru, content\_type),  CONSTRAINT genre\_unique\_index\_en UNIQUE (genre\_en, content\_type)  );  CREATE TABLE country (  country\_id INT IDENTITY(1,1),  country\_ru VARCHAR(255) NOT NULL,  country\_en VARCHAR(255) NOT NULL,  CONSTRAINT country\_pk PRIMARY KEY(country\_id),  CONSTRAINT country\_ru\_unique UNIQUE (country\_ru),  CONSTRAINT country\_en\_unique UNIQUE (country\_en)  );  CREATE TABLE [user] (  [user\_id] INT IDENTITY(1,1),  username VARCHAR(255) NOT NULL,  email VARCHAR(255) NOT NULL,  [password] VARCHAR(255) NOT NULL,  [role] VARCHAR(255) NOT NULL,  CONSTRAINT user\_pk PRIMARY KEY([user\_id]),  CONSTRAINT user\_username\_unique UNIQUE (username),  CONSTRAINT user\_email\_unique UNIQUE (email)  );  CREATE TABLE content\_cast (  cast\_id INT IDENTITY(1, 1),  person\_id INT NOT NULL,  content\_id INT NOT NULL,  proffession VARCHAR(255) NOT NULL,  CONSTRAINT content\_cast\_pk PRIMARY KEY (cast\_id),  CONSTRAINT content\_cast\_person\_fk FOREIGN KEY (person\_id) REFERENCES person(person\_id),  CONSTRAINT content\_cast\_content\_fk FOREIGN KEY (content\_id) REFERENCES content(content\_id),  CONSTRAINT content\_cast\_proffession\_check CHECK (proffession in ('Актёр', 'Режиссёр'))  );  CREATE TABLE content\_genre (  genre\_id INT,  content\_id INT,  CONSTRAINT content\_genre\_pk PRIMARY KEY(genre\_id, content\_id),  CONSTRAINT content\_genre\_genre\_fk FOREIGN KEY (genre\_id) REFERENCES genre(genre\_id),  CONSTRAINT content\_genre\_content\_fk FOREIGN KEY (content\_id) REFERENCES content(content\_id),  );  CREATE TABLE content\_country (  country\_id INT,  content\_id INT,  CONSTRAINT content\_country\_pk PRIMARY KEY(country\_id, content\_id),  CONSTRAINT content\_country\_country\_fk FOREIGN KEY (country\_id) REFERENCES country(country\_id),  CONSTRAINT content\_country\_content\_fk FOREIGN KEY (content\_id) REFERENCES content(content\_id),  );  CREATE TABLE movie\_voiceover (  voiceover\_id INT,  content\_id INT,  media VARCHAR(255) NOT NULL,  CONSTRAINT movie\_voiceover\_pk PRIMARY KEY(voiceover\_id, content\_id),  CONSTRAINT movie\_voiceover\_voiceover\_fk FOREIGN KEY (voiceover\_id) REFERENCES voiceover(voiceover\_id),  CONSTRAINT movie\_voiceover\_content\_fk FOREIGN KEY (content\_id) REFERENCES content(content\_id),  CONSTRAINT movie\_voiceover\_media\_unique UNIQUE (media)  );  CREATE TABLE comment (  comment\_id INT IDENTITY(1,1),  content\_id INT NOT NULL,  [user\_id] INT NOT NULL,  [text] VARCHAR(MAX) NOT NULL,  date\_of\_publication DATETIME NOT NULL CONSTRAINT comment\_date\_default DEFAULT GETDATE(),  CONSTRAINT comment\_pk PRIMARY KEY(comment\_id),  CONSTRAINT comment\_content\_fk FOREIGN KEY (content\_id) REFERENCES content(content\_id),  CONSTRAINT comment\_user\_fk FOREIGN KEY ([user\_id]) REFERENCES [user]([user\_id]),  );  CREATE TABLE bookmark (  bookmark\_id INT IDENTITY(1, 1),  content\_id INT NOT NULL,  [user\_id] INT NOT NULL,  CONSTRAINT bookmark\_pk PRIMARY KEY(bookmark\_id),  CONSTRAINT bookmark\_unique\_index UNIQUE (content\_id, [user\_id])  ); |

Листинг 1 – Создание таблиц

# Приложение Б. Функции БД

|  |
| --- |
| CREATE FUNCTION GetPersonInfo (@length INT)  RETURNS TABLE  AS  BEGIN  DECLARE @chars VARCHAR(255) = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';  DECLARE @result VARCHAR(MAX) = '';  DECLARE @i INT = 1;  WHILE @i <= @length  BEGIN  SET @result = @result + SUBSTRING(@chars, CAST((select V from dbo.vRand) \* LEN(@chars) + 1 AS INT), 1);  SET @i = @i + 1;  END  RETURN @result;  END;  go  CREATE OR ALTER FUNCTION dbo.GetOrderedContent  (  @type VARCHAR(MAX)  )  RETURNS TABLE  AS  RETURN  (  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb,  row\_number() over (ORDER BY content.rating\_imdb desc) as [Место в рейтинге]  FROM content  where [type] = @type  );  go  select \* from GetOrderedContent('Anime')  go  CREATE OR ALTER FUNCTION dbo.GetContentByGenre  (  @genreId INT,  @type VARCHAR(MAX)  )  RETURNS TABLE  AS  RETURN  (  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb,  row\_number() over (ORDER BY content.rating\_imdb desc) as [Место в рейтинге]  FROM content join content\_genre on content.content\_id = content\_genre.content\_id  where [type] = @type and genre\_id = @genreId  );  go  select \* from GetContentByGenre(100, 'Film')  go  CREATE OR ALTER FUNCTION dbo.GetContentByYear  (  @year INT,  @type VARCHAR(MAX)  )  RETURNS TABLE  AS  RETURN  (  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb,  row\_number() over (ORDER BY content.rating\_imdb desc) as [Место в рейтинге]  FROM content  where [type] = @type and YEAR(content.release\_date) = @year  );  go  select \* from GetContentByYear(2024, 'Film')  go  CREATE OR ALTER FUNCTION dbo.GetContentByGenreAndYear  (  @year INT,  @genreId INT,  @type VARCHAR(MAX)  )  RETURNS TABLE  AS  RETURN  (  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb,  row\_number() over (ORDER BY content.rating\_imdb desc) as [Место в рейтинге]  FROM content join content\_genre on content.content\_id = content\_genre.content\_id  where [type] = @type and YEAR(content.release\_date) = @year and genre\_id = @genreId  );  go  select \* from GetContentByGenreAndYear(2000, 100, 'Film')  go  CREATE OR ALTER FUNCTION dbo.GetContentActors  (  @contentId INT  )  RETURNS TABLE  AS  RETURN  (  SELECT name\_ru  FROM person join content\_cast on person.person\_id = content\_cast.person\_id  where content\_id = @contentId and name\_ru like 'Актёр%'  );  go  select \* from GetContentActors(1)  go  CREATE OR ALTER FUNCTION dbo.GetContentDirector  (  @contentId INT  )  RETURNS TABLE  AS  RETURN  (  SELECT name\_ru  FROM person join content\_cast on person.person\_id = content\_cast.person\_id  where content\_id = @contentId and name\_ru like 'Режиссёр%'  );  go  select \* from GetContentDirector(1)  go  CREATE OR ALTER FUNCTION dbo.GetContentComments  (  @contentId INT  )  RETURNS TABLE  AS  RETURN  (  SELECT \*  FROM comment  where content\_id = @contentId  ); |

Листинг 1 – Функции, связанные непосредственно с аукционами

# Приложение В. Прдеставления

|  |
| --- |
| use [online-cinema]  go  CREATE OR ALTER VIEW get\_films AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where [type] = 'Film';  go  CREATE OR ALTER VIEW get\_new\_films AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where release\_date > DATEADD(MONTH, -6, GETDATE()) and [type] = 'Film';  go  -- все сериалы  CREATE OR ALTER VIEW get\_series AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where [type] = 'Series';  go  -- сериалы за последние пол года  CREATE OR ALTER VIEW get\_new\_series AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where release\_date > DATEADD(MONTH, -6, GETDATE()) and [type] = 'Series';  go  -- все аниме  CREATE OR ALTER VIEW get\_anime AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where [type] = 'Anime';  go  -- аниме за последние пол года  CREATE OR ALTER VIEW get\_new\_anime AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where release\_date > DATEADD(MONTH, -6, GETDATE()) and [type] = 'Anime';  go  -- все мультфильмы  CREATE OR ALTER VIEW get\_cartoons AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where [type] = 'Cartoon';  go  -- мультфильмы за последние пол года  CREATE OR ALTER VIEW get\_new\_cartoons AS  SELECT content.content\_id [Идентификатор фильма],  title\_ru as [Название фильма],  YEAR(release\_date) as [Год выпуска],  (select top 1 country\_ru from content\_country join country on content\_country.country\_id = country.country\_id where content\_country.content\_id = content.content\_id) as [Страна производства],  (select top 1 genre\_ru from content\_genre join genre on content\_genre.genre\_id = genre.genre\_id where content\_genre.content\_id = content.content\_id) as [Жанр фильма],  rating\_imdb  FROM content  where release\_date > DATEADD(MONTH, -6, GETDATE()) and [type] = 'Cartoon';  go |

Листинг 2 – Представления

# Приложение Г. Триггеры.

|  |
| --- |
| CREATE TRIGGER CheckBookmarkCount  ON bookmark  AFTER INSERT, UPDATE  AS  BEGIN  DECLARE @userId INT;  SELECT @userId = [user\_id] FROM inserted;  IF (SELECT COUNT(\*) FROM bookmark WHERE [user\_id] = @userId) > 100  BEGIN  Print('У одного пользователя не может быть больше 100 закладок');  ROLLBACK TRANSACTION;  END  END;  go  CREATE TRIGGER CheckReleaseDate  ON content  AFTER INSERT, UPDATE  AS  BEGIN  IF EXISTS (SELECT 1 FROM inserted WHERE release\_date > GETDATE())  BEGIN  Print('Дата выхода не может быть позже текущей даты');  ROLLBACK TRANSACTION;  END;  END;  go  CREATE TRIGGER DeleteUserBookmarks  ON [user]  AFTER DELETE  AS  BEGIN  DELETE FROM bookmark WHERE [user\_id] IN (SELECT user\_id FROM deleted);  END;  go  CREATE TRIGGER CheckMovieVoiceoverCount  ON movie\_voiceover  AFTER INSERT, UPDATE  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM content c LEFT JOIN movie\_voiceover mv ON c.content\_id = mv.content\_id WHERE mv.content\_id IS NULL)  BEGIN  Print('Каждый фильм должен иметь хотя бы 1 озвучку');  ROLLBACK TRANSACTION;  END;  END;  go  CREATE TRIGGER CheckCommentLength  ON comment  AFTER INSERT, UPDATE  AS  BEGIN  IF EXISTS (SELECT 1 FROM inserted WHERE LEN([text]) > 255)  BEGIN  Print('Максимальная длинна комментария 255 символов');  ROLLBACK TRANSACTION;  END;  END;  go |

Листинг 1 – Функция IMPORT\_LOTS\_FROM\_JSON\_FILE

# Приложение Д

|  |
| --- |
| CREATE ROLE Administrator;  CREATE ROLE [User];  CREATE ROLE UnauthorizedUser;  GRANT SELECT, INSERT, UPDATE, DELETE ON content TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON bookmark TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON comment TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON content TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON content\_cast TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON content\_country TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON content\_genre TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON genre TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON country TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON movie\_voiceover TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON person TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON series\_episode TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.[user] TO Administrator;  GRANT SELECT, INSERT, UPDATE, DELETE ON voiceover TO Administrator;  GRANT SELECT ON dbo.GetOrderedContent TO Administrator;  GRANT SELECT ON dbo.GetContentByGenre TO Administrator;  GRANT SELECT ON dbo.GetContentByYear TO Administrator;  GRANT SELECT ON dbo.GetContentByGenreAndYear TO Administrator;  GRANT SELECT ON dbo.GetContentActors TO Administrator;  GRANT SELECT ON dbo.GetContentDirector TO Administrator;  GRANT SELECT ON dbo.GetContentComments TO Administrator;  GRANT EXECUTE ON dbo.DeleteContent TO Administrator;  GRANT EXECUTE ON dbo.AddContent TO Administrator;  GRANT EXECUTE ON dbo.UpdateContent TO Administrator;  GRANT EXECUTE ON dbo.DeleteGenre TO Administrator;  GRANT EXECUTE ON dbo.AddGenre TO Administrator;  GRANT EXECUTE ON dbo.UpdateGenre TO Administrator;  GRANT EXECUTE ON dbo.DeleteCountry TO Administrator;  GRANT EXECUTE ON dbo.AddCountry TO Administrator;  GRANT EXECUTE ON dbo.UpdateCountry TO Administrator;  GRANT EXECUTE ON dbo.DeletePerson TO Administrator;  GRANT EXECUTE ON dbo.AddPerson TO Administrator;  GRANT EXECUTE ON dbo.UpdatePerson TO Administrator;  GRANT EXECUTE ON dbo.DeleteVoiceover TO Administrator;  GRANT EXECUTE ON dbo.AddVoiceover TO Administrator;  GRANT EXECUTE ON dbo.UpdateVoiceover TO Administrator;  GRANT EXECUTE ON dbo.DeleteUser TO Administrator;  GRANT EXECUTE ON dbo.AddUser TO Administrator;  GRANT EXECUTE ON dbo.UpdateUser TO Administrator;  GRANT EXECUTE ON dbo.DeleteBookmark TO Administrator;  GRANT EXECUTE ON dbo.AddBookmark TO Administrator;  GRANT EXECUTE ON dbo.UpdateBookmark TO Administrator;  GRANT EXECUTE ON dbo.DeleteComment TO Administrator;  GRANT EXECUTE ON dbo.AddComment TO Administrator;  GRANT EXECUTE ON dbo.UpdateComment TO Administrator;  GRANT EXECUTE ON dbo.DeleteMovieVoiceover TO Administrator;  GRANT EXECUTE ON dbo.AddMovieVoiceover TO Administrator;  GRANT EXECUTE ON dbo.UpdateMovieVoiceover TO Administrator;  DENY SELECT ON content TO [User];  DENY SELECT ON content TO [User];  DENY SELECT ON content TO [User];  DENY SELECT ON bookmark TO [User];  DENY SELECT ON comment TO [User];  DENY SELECT ON content TO [User];  DENY SELECT ON content\_cast TO [User];  DENY SELECT ON content\_country TO [User];  DENY SELECT ON content\_genre TO [User];  DENY SELECT ON genre TO [User];  DENY SELECT ON country TO [User];  DENY SELECT ON movie\_voiceover TO [User];  DENY SELECT ON person TO [User];  DENY SELECT ON series\_episode TO [User];  DENY SELECT ON dbo.[user] TO [User];  DENY SELECT ON voiceover TO [User];  DENY SELECT ON content TO UnauthorizedUser;  DENY SELECT ON content TO UnauthorizedUser;  DENY SELECT ON bookmark TO UnauthorizedUser;  DENY SELECT ON comment TO UnauthorizedUser;  DENY SELECT ON content TO UnauthorizedUser;  DENY SELECT ON content\_cast TO UnauthorizedUser;  DENY SELECT ON content\_country TO UnauthorizedUser;  DENY SELECT ON content\_genre TO UnauthorizedUser;  DENY SELECT ON genre TO UnauthorizedUser;  DENY SELECT ON country TO UnauthorizedUser;  DENY SELECT ON movie\_voiceover TO UnauthorizedUser;  DENY SELECT ON person TO UnauthorizedUser;  DENY SELECT ON series\_episode TO UnauthorizedUser;  DENY SELECT ON dbo.[user] TO UnauthorizedUser;  DENY SELECT ON voiceover TO UnauthorizedUser;  CREATE LOGIN AdminUser WITH PASSWORD = 'AdminPassword';  CREATE USER AdminUser FOR LOGIN AdminUser;  ALTER ROLE Administrator ADD MEMBER AdminUser;  CREATE LOGIN RegularUser WITH PASSWORD = 'UserPassword';  CREATE USER RegularUser FOR LOGIN RegularUser;  ALTER ROLE [User] ADD MEMBER RegularUser;  CREATE LOGIN UnauthorizedUserA WITH PASSWORD = 'GuestPassword';  CREATE USER UnauthorizedUserA FOR LOGIN UnauthorizedUserA;  ALTER ROLE UnauthorizedUser ADD MEMBER UnauthorizedUseA; |

Листинг 1 – Роль гостя