МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка базы данных для театра с настройкой системы безопасности СУБД»

Выполнил студент Вашков Никита Андреевич

(Ф.И.О.)

Руководитель проекта ст. пр. 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 ст. пр. Жигаровская С. А.

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

Нормоконтролер ст. пр. Жигаровская С. А.

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

Курсовой проект защищен с оценкой

Минск 2018

**Реферат**

Пояснительная записка курсового проекта содержит 29 страниц, 16 рисунков, 5 источников литературы, 2 приложения и 2 листингов.

НАСТРОЙКОЙ СИСИТЕМЫ БЕЗОПАСНОСТИ СЕРВЕРА СУБД, ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ, SQL, T-SQL, MICROSOFT SQL SERVER

Основными целями курсового проекта являются: проектирование базы данных для театра

Пояснительная записка состоит из введения, шести разделов, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе рассматривается архитектура базы данных.

Во втором разделе представлен процесс и результаты разработки объектов базы данных.

В третьем разделе описывается создание процедур для экспорта и импорта.

В четвертом разделе представлено тестирование производительности.

В пятом разделе описана технология репликации моментальных снимков.

В шестом разделе представлено руководство пользователя.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе проектирования и разработки базы данных.

**Содержание**

[Введение 4](#_Toc531520527)

[Постановка задачи 5](#_Toc531520528)

[1. Разработка модели базы данных 6](#_Toc531520529)

[2. Разработка необходимых объектов 8](#_Toc531520530)

[2.1 Таблицы 8](#_Toc531520531)

[2.2 Хранимые процедуры 8](#_Toc531520533)

[3. Описание процедур импорта и экспорта данных 10](#_Toc531520534)

[3.1. Описание процедуры импорта данных 10](#_Toc531520535)

[3.2. Описание процедуры экспорта данных 11](#_Toc531520536)

[4. Тестирование производительности 12](#_Toc531520537)

[4.1 Тестирование производительности базы данных 12](#_Toc531520538)

[5. Описание технологии настройки безопасности сервера СУБД 14](#_Toc531520539)

[6. Руководство пользователя 18](#_Toc531520543)

[Заключение. 21](#_Toc531520546)

[Список используемой литературы 22](#_Toc531520547)

[ПРИЛОЖЕНИЕ А 23](#_Toc531520548)

[ПРИЛОЖЕНИЕ Б 24](#_Toc531520549)

# **Введение**

В настоящее время успешное функционирование различных фирм, организаций и предприятий просто не возможно без развитой информационной системы, которая позволяет автоматизировать сбор и обработку данных. Обычно для хранения и доступа к данным, содержащим сведения о некоторой предметной области, создается база данных.

База данных (БД) — это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, преимущественно больших объемов.

Для работы с базами данных существуют системы управления базами данных. Система управления базами данных (СУБД) — совокупность языковых и программных средств, предназначенных для создания, наполнения, обновления и удаления баз данных.

К самым популярным СУБД относятся: Oracle, MS SQL Server, Microsoft Access, MySQL, PostgreSQL, MongoDB и так далее. Данный курсовой проект разработан в.

Для разработки приложения использовалась СУБД MS SQL Server с использованием такой технологии технология, как Windows Presentation Foundation(WPF) на языке программирования C#. Для взаимодействия приложения с базой данных использовалась технология ADO.NET.

# **Постановка задачи**

В соответствии с заданием курсового проекта следует создать базу данных (БД) для театра с использованием программного продукта Microsoft SQL Server 2017 в которой доступ к данным должен осуществляться с помощью процедур и должен быть проеден импорт данных из XML файлов, экспорт данных в формат XML. База данных должна быть протестирована на производительность и использовать такую технологию, как настройка системы безопасности сервера СУБД.

1. **Разработка модели базы данных**

Для базы данных театра было разработано 5 таблиц. На рисунке 1.1 предоставлена структура связей внешних ключей и первичных ключей. База данных была разработана в СУБД MS SQL Server.

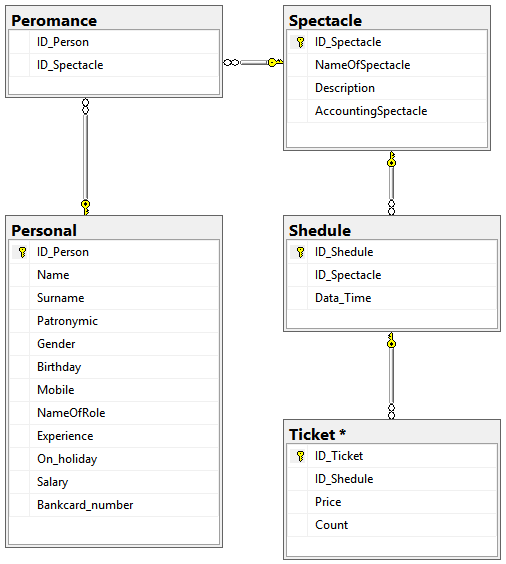


Рисунок 1.1 – Диаграмма базы данных

Таблица Personal предназначена для хранения информации о сотрудниках:

* ID\_Person – уникальный идентификатор;
* Name – имя сотрудника;
* Surname – фамилия сотрудника;
* Patronymic – отчество сотрудника ;
* Gender – пол;
* Birthday – дата рождения;
* Mobile – мобильный телефон;
* NameOfRole – должность сотрудника;
* Experience – опыт работы;
* Salary – зарплата сотрудника;
* Bankcard\_number – номер банковской карты.

Таблица Spectacle предназначена для хранения информации о спектаклях:

* ID\_Spectacle – уникальный идентификатор;
* NameOfSpectacle – название спектакля;
* Description – описание;
* AccountingSpectacle – учет иностранных спектаклей.

Таблица Peromance предназначенная для осуществления связи многиие-ко-многим между таблицами Personal и Spectacle соответсвенно:

* ID\_Person – уникальный идентификатор сотрудника;
* ID\_Spectacle – уникальный идентификатор спектакля;

Таблица Shedule предназначена для хранения информации о дате проведения спектаклей:

* ID\_Shedule – уникальный идентификатор;
* ID\_Spectacle – идентификатор спектакля;
* Data\_Time – дата проведения.

Таблица Ticket хранит поставки товара:

* ID\_Ticket – уникальный идентификатор;
* ID\_Shedule – идентификатор расписания;
* Price – цена билета;
* Count – количество билетов;

Скрипты для создания всех таблиц базы данных предоставлены в корне прилагаемого диска.

1. **Разработка необходимых объектов**

База данных данного курсового проекта содержит следующие объекты: таблицы, хранимые процедуры, индексы. Подробное описание представлено в главе 2.

## **2.1 Таблицы**

База данных театра содержит 5 таблиц, которые описаны в главе 1. SQL-скрипты для создания таблиц находятся на прилагаемом к записке диске.

## **2.2 Хранимые процедуры**

Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Их использование в курсовом проекте предназначено для ограничения доступа пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение некоторых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным.

Перечень разработанных в ходе выполнения курсового проекта хранимых процедур:

* [dbo].[sp\_InsertPerson] – процедура для добавления сотрудника;
* [dbo].[sp\_InsertShedule] – процедура для добавления расписания;
* [dbo].[sp\_InsertTicket] – процедура для добавления билета;
* [dbo].[sp\_InsertPersonToSpectacle] – процедура для добавления актеров к спектаклю;
* [dbo].[ob\_PersonalSelectAll] – процедура для выборки всех записей сотрудников;
* [dbo].[ob\_SpectacleSelectAll] – процедура для выборки всех записей спектаклей;
* [dbo].[ob\_SheduleSelectAll] – процедура для выборки всех записей распиания;
* [dbo].[ob\_ActorToSpectacleSelectAll] – процедура для выборки всех сотрудников в спектакле;
* [dbo].[ob\_TicketSelectAll] – процедура для выборки всех записей билетов;
* [dbo].[ob\_PersobnalDeleteAll] – процедура для удаления всех записей – процедура;
* [dbo].[ob\_PersonalDeleteItem] – процедура для удаления одного сотрудника – процедура;
* [dbo].[ob\_SpectacleDeleteItem] – процедура для удаления одного спектакль;
* [dbo].[ob\_SheduleDeleteItem] – процедура для удаления одного расписания;
* [dbo].[ob\_TicketDeleteItem] – процедура для удаления одного сотрудника;
* [dbo].[ob\_ActorFromSpectacleDeleteItem] – процедура для удаления одного сотрудника;
* [dbo].[ob\_PersonalSearchItem] – процедура для поиска сотрудника по ФИО;
* [dbo].[ob\_SpectacleSearchItem] – процедура для поиска спектаклей по названию и метки;
* [dbo].[ob\_SheduleSearchItem] – процедура для поиска расписаний по дате;
* [dbo].[ob\_ACTSpecSearchItem] – процедура для поиска актеров задействованных в спектакле;
* [dbo].[ob\_PersonalComboBoxItem] – процедура для вывода сотрудников в combobox;
* [dbo].[ob\_SpectacleComboBoxItem] – процедура для вывода спектаклей в combobox;
* [dbo].[ob\_TicketComboBoxItem] – процедура для вывода билетов в combobox;
* [dbo].[insertDB] – процедура внесение данных в таблицу сотрудников;
* [exportXML] – процедура экспорта данных в XML;
* [importXML] – процедура импорт xml таблица [Personal].

Скрипт хранимых процедур приложен в корне директория прилагаемого диска. Листинги хранимых процедур представлены в приложении Б.

1. **Описание процедур импорта и экспорта данных**

В курсовой работе реализованы процедуры импорта и экспорта данных таблицы Personal.

## **3.1. Описание процедуры импорта данных**

Существует несколько вариантов процедуры импорта данных xml в таблицы MS SQL Server. В данной курсовой работе я остановился на варианте реализации импорта данных с помощью OPENROWSET функции. OPENROWSET содержит все необходимые сведения о соединении, которые требуются для доступа к удаленным данным источника данных OLE DB. Это альтернативный метод для доступа к таблицам на связанном сервере и является однократным нерегламентированным методом соединения и удаленного доступа к данным с помощью OLE DB.

Были выполнены две цели, для выполнения преобразования данных из xml:

* импорт данных xml в MS SQL Server с помощью функции OPENROWSET;
* преобразование формата xml с помощью BULK.

Процедура импорта приведена в листинге 1.

GO

CREATE PROCEDURE [importXML] -- импорт xml таблица [Personal]

@xml XML = NULL

AS

Select @xml =

CONVERT(XML,bulkcolumn,2) FROM OPENROWSET(BULK 'D:\3\_1kurs\Oracle\Курсач\TheaterBD\xmlBD.xml',SINGLE\_BLOB) AS X

SET ARITHABORT ON

INSERT INTO [dbo].[Personal]([ID\_Person], [Name], [Surname] , [Patronymic], [Gender], [Birthday], [Mobile], [NameOfRole], [Experience], [On\_holiday], [Salary], [Bankcard\_number])

SELECT

P.value('ID\_Personal[1]', 'int') AS [ID\_Personal],

P.value('Name[1]','nvarchar(50)') AS [Name],

P.value('Surname[1]','nvarchar(50)') AS [Surname],

P.value('Patronymic[1]','nvarchar(50)') AS [Patronymic],

P.value('Gender[1]','char') AS [Gender],

P.value('Birthday[1]','date') AS [Birthday],

P.value('Mobile[1]','varchar(12)') AS [Mobile],

P.value('NameOfRole[1]','nvarchar(50)') AS [NameOfRole],

P.value('Experience[1]','int') AS [Experience],

P.value('On\_holiday[1]','bit') AS [On\_holiday],

P.value('Salary[1]','money') AS [Salary],

P.value('Bankcard\_number[1]','varchar(16)') AS [Bankcard\_number]

FROM @xml.nodes('//Personals/Person') PropertyFeed(P)

COMMIT

Листинг 1 – процедура импорта из xml в таблицу Personal

## **3.2. Описание процедуры экспорта данных**

Код создания процедуры экспорта данных таблицы в xml формат представлен в листинге 2.

GO

CREATE PROCEDURE [exportXML] -- экспорт

AS

SELECT [ID\_Person], [Name], [Surname] , [Patronymic], [Gender], [Birthday], [Mobile], [NameOfRole], [Experience], [On\_holiday], [Salary], [Bankcard\_number]

FROM [dbo].[Personal]

FOR XML PATH('Person'), ROOT('Personals');

Листинг 2 – процедура создания экспорта данных

1. **Тестирование производительности**

В реляционной СУБД оптимальный план выполнения запроса – это такая последовательность применения операторов реляционной алгебры к исходным и промежуточным отношениям, которое для конкретного текущего состояния БД (её структуры и наполнения) может быть выполнено с минимальным использованием вычислительных ресурсов.

Некластеризованный индекс – это отдельный объект, имеющий указатели на строки таблицы.

В MS SQL индексы могут быть двух типов: кластеризованные и некластеризованные.

Кластеризованный индекс отличается тем, что в листьях этого индекса содержатся не ссылки на записи в таблице, а сами записи.

Скорость доступа к строкам при указании условия where, совпадающего с условием индексирования для кластеризованного индекса, будет выше, чем при поиске записей по простому индексу.

## **4.1 Тестирование производительности базы данных**

Для тестирования базы данных была выбрана таблица Personal, которая наполнена данными больше всех. Начальное количество строк в таблице 100000. Количество сотрудников представлено на рисунке 4.1.

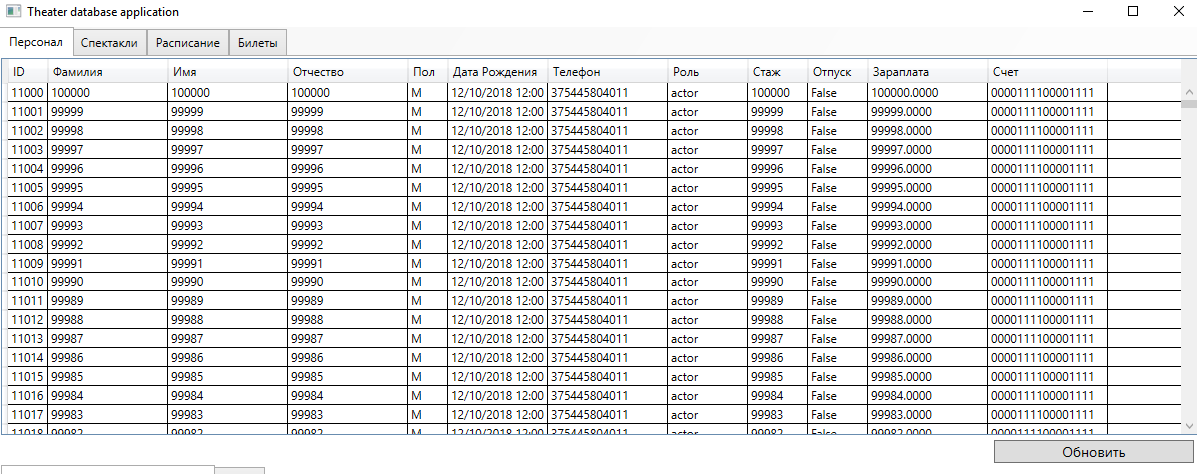


Рис. 4.1 – количество строк в таблице сотрудников

Для сравнения был выполнен запрос к таблице Products без создания индексов. Результат стоимости запроса приведен на рисунке 4.2. Выборка производилась по имени и фамилии.

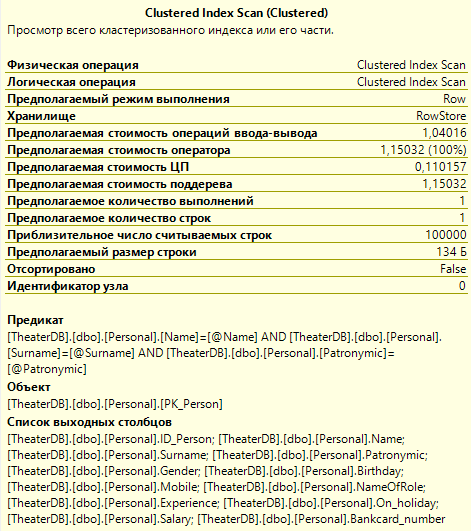


Рис. 4.2 – стоимость запроса к таблице Personal кластеризованного индекса

Теперь для сравнения добавим некластеризованный индекс к этой таблице с помощью запроса CREATE INDEX indPer ON [Personal] ([Name], [Surname]) и выполним поиск снова. Результат представлен на рисунке 4.3.

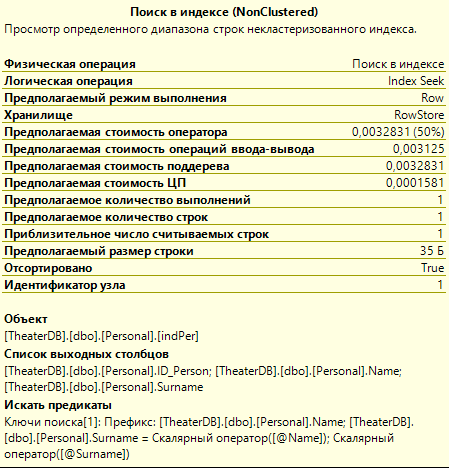
****

Рисунок. 6.2 – стоимость запроса к таблице Personal c использованием некластеризованного индекса

По результатам стоимости запроса можно сказать, что построение индексов в данной таблице максимально необходимо, так как во втором случае общая стоимость запроса уменьшилась более, чем в 2 раза. Также были добавлены еще пару индексов для оптимизации запросов выборки (поиска по заданным значениям). С запросами можно ознакомится в корне диска.

1. **Описание технологии настройки безопасности сервера СУБД**

При использовании баз данных содержащих конфиденциальную информацию о сотрудниках или любую другую важную информацию следует обеспечивать максимальную сохранность и защиту данных от злоумышленников.

Защиту SQL Server можно рассматривать как последовательность шагов, затрагивающих четыре области: платформу, проверку подлинности, объекты (включая данные) и приложения, получающие доступ к системе. В приведенных ниже разделах описано создание и реализация эффективного плана обеспечения безопасности.

* Рекомендуется строго ограничивать доступ к физическим серверам и компонентам оборудования. Например, оборудование сервера базы данных и сетевые устройства должны находиться в закрытых охраняемых помещениях. Доступ к резервным носителям также следует ограничить. Для этого их рекомендуется хранить в отдельных охраняемых помещениях.
* Реализация физической сетевой безопасности начинается с запрета доступа неавторизованных пользователей к сети.
* Уменьшение контактной зоны является мерой безопасности, предполагающей остановку или отключение неиспользуемых компонентов. Уменьшение контактной зоны повышает уровень безопасности за счет уменьшения числа возможных способов атаковать систему. Важную роль в ограничении контактной зоны SQL Server играет запуск необходимых служб по принципу «минимума прав доступа», согласно которому службам и пользователям предоставляются только необходимые для работы права. Следующая таблица содержит дополнительные сведения по службам и доступу к системе.
* Защищаемые объекты — это сервер, база данных и объекты, которые содержит база данных. У каждого из них существует набор разрешений, с помощью которых можно уменьшить контактную зону SQL Server , такие как Пользователи, роли и процессы сервера и базы данных, а также безопасность объектов сервера и базы данных.
* Шифрование повышает безопасность, ограничивая потерю данных даже в тех редких случаях, когда средства управления доступом удается обойти. Например, если главный компьютер, на котором установлена база данных, был настроен неправильно, и злонамеренный пользователь смог получить конфиденциальные данные (например, номера кредитных карточек), то украденная информация будет бесполезна, если она была предварительно зашифрована.

## **5.1 Настройка шифрования данных**

Для шифрования столбца таблицы базы данных требуется в первую очередь запустить специальную службу путем нажатия правой кнопкой мыши по нужному столбцу и выборе надписи “Зашифровать столбец”. Как показано на рисунке 5.1.

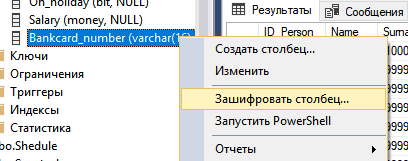


Рисунок 5.1 Запуск службы шифрования

Откроется мастер настройки шифрования Always Encrypted с окном приветствия. Рисунок 5.2.

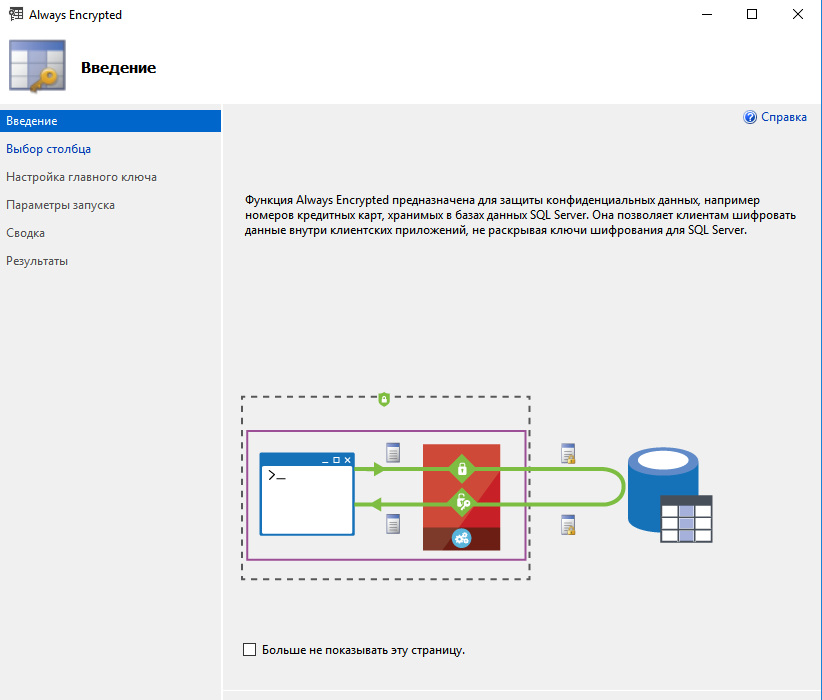


Рисунок 5.2 Окно приветствия Always Encrypted

Далее следует выбрать роле которое необходимое зашифровать и выбрать тип шифрования. Здесь я выбрал детерминированный тип, что означает что процесс шифрования полностью определен алгоритмом, значениями входных переменных, и начальным состоянием системы. Ключ шифрования выбран стандартный CEK\_Auto1, но также можно с помощью определенных SQL-выражений создать свои ключи и сертификаты по которым будет происходить шифрование данных. Рисунок 5.3.

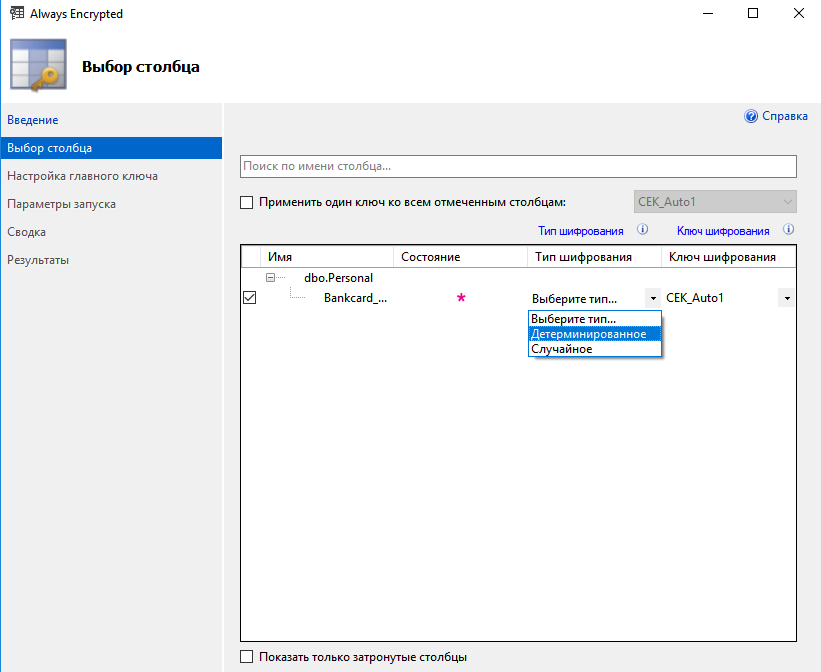


Рисунок 5.3 Выбор детерминированного типа шифрования по ключу CEK\_Auto1

После выполнения вышеуказанных действий мастер настроек укажет о корректности выбранных значений и ключа и предложит завершить настройку шифрования столбца. Рисунок 5.4.

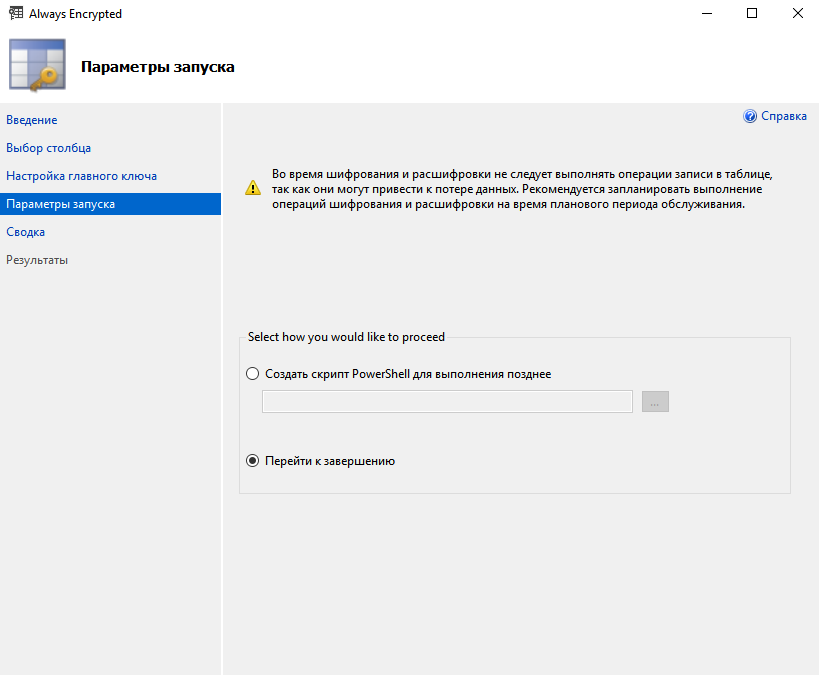


Рисунок 5.4 Окно завершения настроек шифрования

Далее необходимо дождаться процесса шифрования и закрыть мастер настроек. Рисунок 5.5).

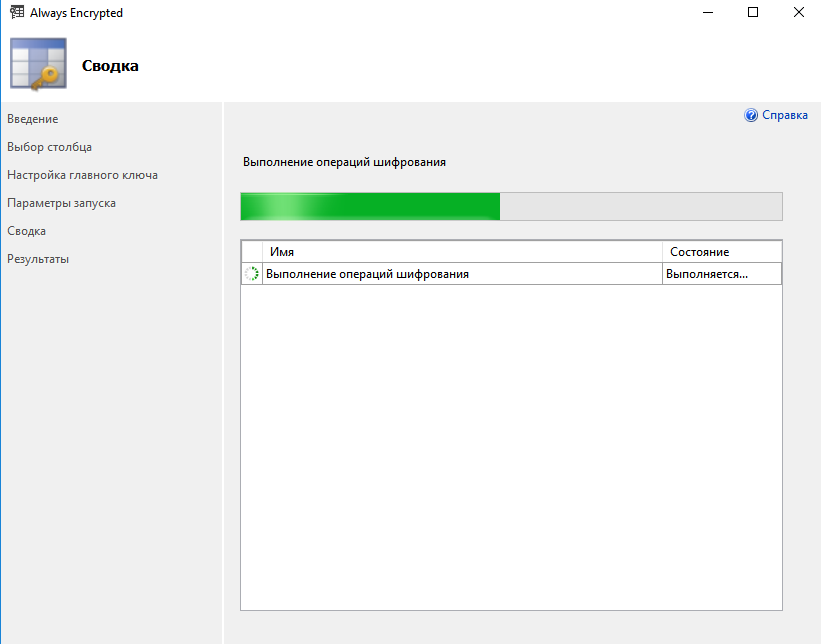


Рисунок 5.5 Завершение процесса шифрования столбца таблицы Personal

Результат проделанной операции для сравнения изменений, которые прошли со столбцом можно увидеть на рисунках 5.6 и 5.7 соответственно.

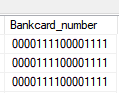


Рисунок 5.6 Данные банковской карты до шифрования

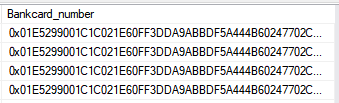


Рисунок 5.7 Данные банковской карты после шифрования

Тем не менее в приложении, разработанном для базы данных театром, необходим вывод полей, которые зашифрованы в базе данных на данный момент. Поэтому для корректного отображения содержимого в строке подключения, которая находится в коде нужно прописать “column encryption setting = true” , что означает подключение ключей и сертификатов шифрования в проект. Рисунок 5.8.



Рисунок 5.8 Включение шифрования/

1. **Руководство пользователя**

Приложение, разработанное в данном курсовом проекте, предназначена для администрирования внутренней организации театра, что подразумевает использование приложения исключительно администраторами. При запуске появляется главное окно с открытой вкладкой “Сотрудники”. Здесь администратор может добавлять, изменять и удалять сотрудников театра, а также производить поиск по ФИО, который находится на вкладке “Поиск”. Начальный экран представлен на рисунке 6.1.

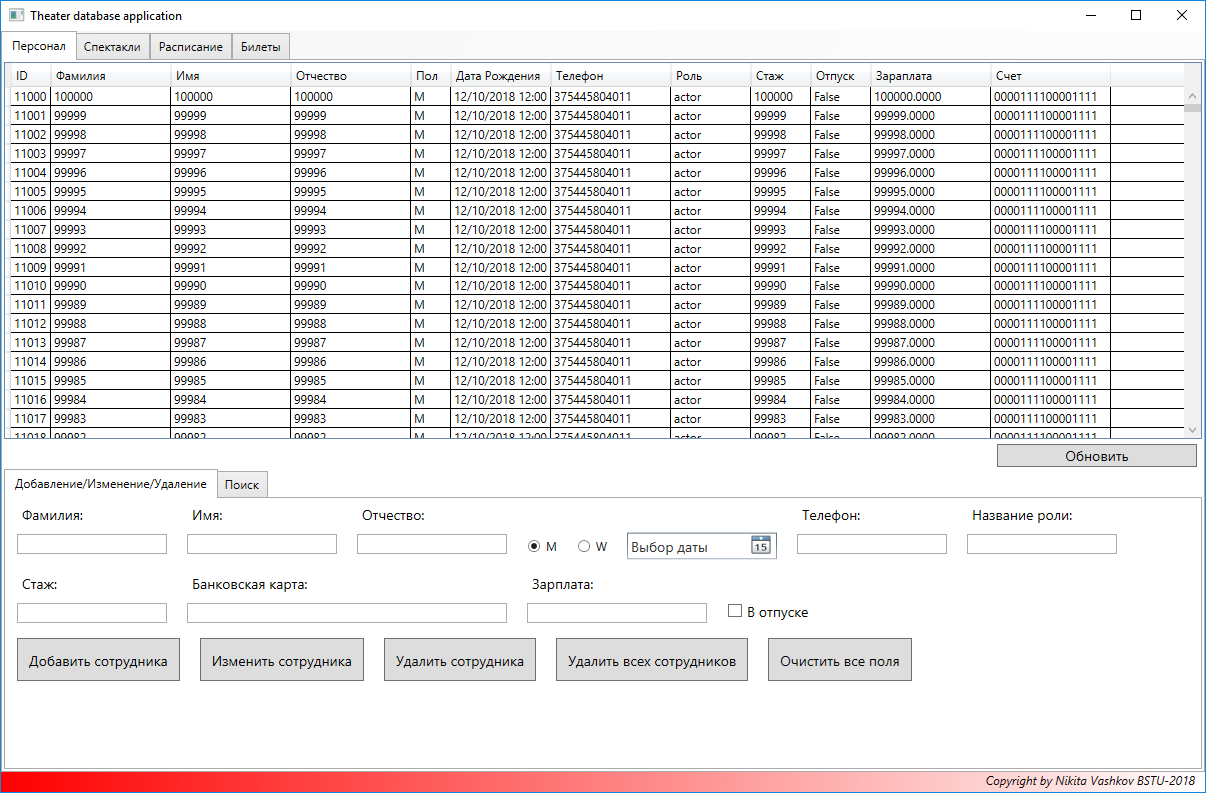


Рисунок 6.1 Вкладка сотрудники театра

Следующая вкладка «Спектакли» предназначена для добавления, изменения и удаления спектаклей. Также администратор может производить поиск спектаклей по названию во вкладке «Поиск». На вкладке «Персонал спектаклей» можно привязывать сотрудников к определенным спектаклям, что позволяет вести учет трупы. Рисунок 6.2.

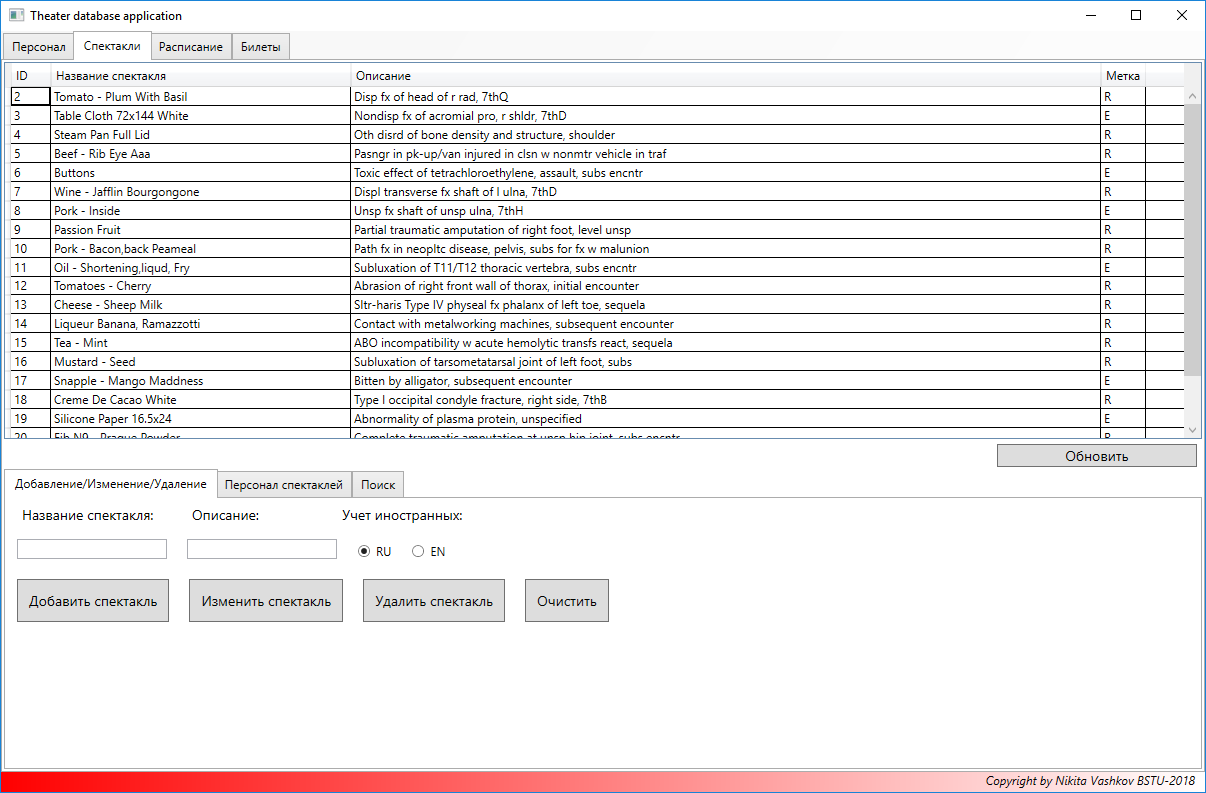


Рисунок 6.2 Вкладка спектаклей

На вкладке «Расписание» можно привязывать дату проведения спектаклей, добавлять, изменять и удалять их. Производить поиск по дате на вкладке «Поиск». Рисунок 6.3.

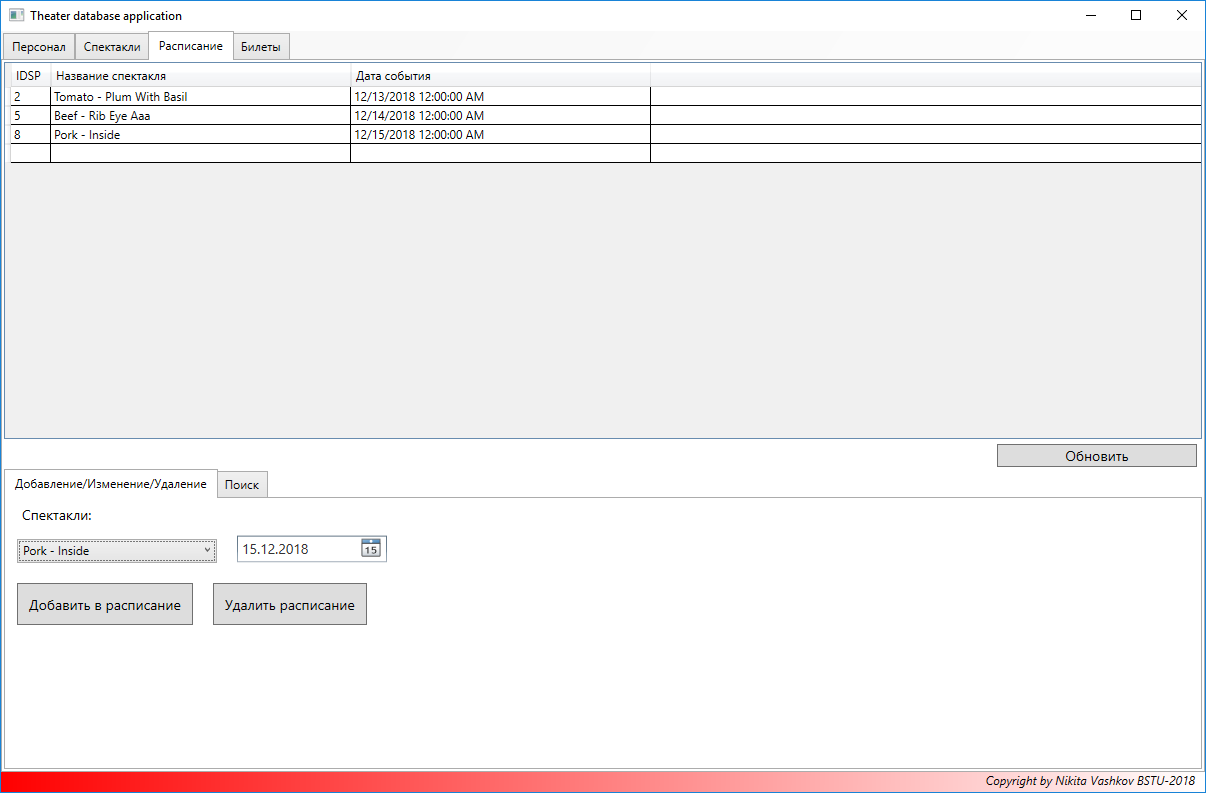


Рисунок 6.3 Вкладка расписание

На вкладке «Билеты» происходит учет билетов на спектакли, где мы можем добавлять их указывая стоимость одного билета и общее доступное их количество. Это представлено на рисунке 6.4.

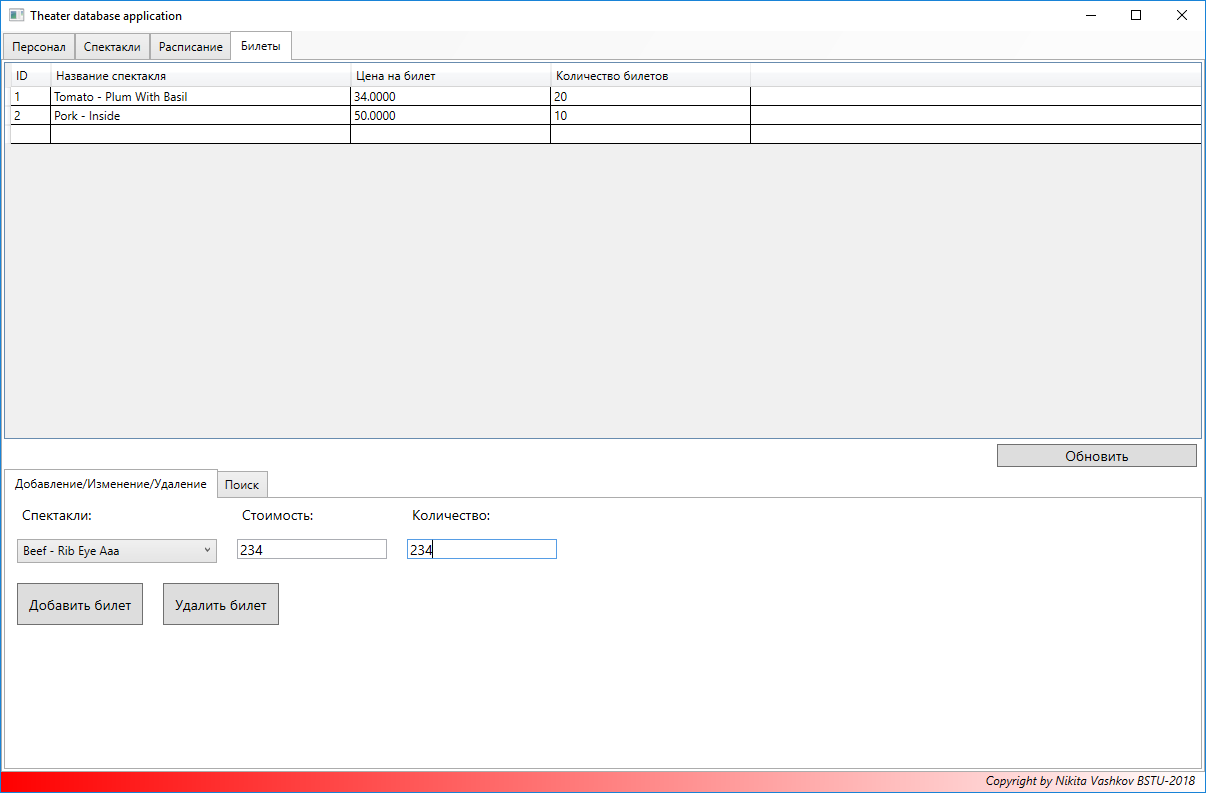


Рисунок 6.4 Вкладка билеты

Каждая вкладка имеет на своем окне datagrid, в котором отображаются данные базы данных и имеется кнопка обновить, отвечающая за обновление datagrid.

# **Заключение.**

В данном курсовом проекте была разработана база данных для театра с интерфейсом в виде десктоп приложения, позволяющего ею управлять и изменять данные. Также была проведена настройка шифрования данных таблиц, путем настройки сервера СУБД для предотвращения кражи данных злоумышленниками. Была проведена проверка на производительность путем выборки данных в 100000 строк, и оптимизацией их путем добавления некластерезированных индексов уменьшающих время выполнения запросов путем разбиения данных.

Программная система имеет простой, интуитивно понятный интерфейс пользователя, не занимающий много место на вашем компьютере.

В заключение после проведения определенных тестов, можно сказать, что программа прошла проверку на корректность работы и является готовым рабочим продуктом для администрирования баз данных театрами.

# **Список используемой литературы**

1. Блинова Е.А. Курс лекций по базам данных / Е. А. Блинова.
2. Microsoft SQL Server 2008: Основы T-SQL / В. В. Смелов, Л. С. Мороз.
3. Руководство по ADO.NET и работе с базами данных [Электронный ресурс] <https://metanit.com/sharp/adonet>. Дата доступа 15.11.2018.
4. Обеспечение безопасности баз данных SQL Server [Электронный ресурс] https://docs.microsoft.com/ru-ru/sql/relational-databases/security/securing-sql-server?view=sql-server-2017 Дата доступа 20.11.2018.
5. Database Encryption in SQL Server [Электронный ресурс] https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008/cc278098(v=sql.100). Дата доступа 24.11.2018.

# **ПРИЛОЖЕНИЕ А**

CREATE DATABASE TheaterDB;

Use TheaterDB;

go

CREATE TABLE [Personal] --Таблица персонала

(

[ID\_Person] int constraint PK\_Person primary key,

[Name] nvarchar(50),

[Surname] nvarchar (50),

[Patronymic] nvarchar (50),

[Gender] char default 'M' check([Gender] = 'M' or [Gender] ='W'),

[Birthday] date,

[Mobile] varchar(12),

[NameOfRole] nvarchar (50),

[Experience] int,

[On\_holiday] bit default 'false',

[Salary] money,

[Bankcard\_number] varchar(16)

)

CREATE TABLE [Spectacle] --Таблица спектаклей

(

[ID\_Spectacle] int constraint PK\_Spectacle primary key,

[NameOfSpectacle] nvarchar (50),

[Description] nvarchar (200),

[AccountingSpectacle] char default 'R' check([AccountingSpectacle] = 'R' or [AccountingSpectacle] = 'E')

)

CREATE TABLE [Peromance] --Таблица представлений дял связи многии ко многим

(

[ID\_Person] int constraint FK\_PerfomancePersonSpectacle foreign key references [Personal](ID\_Person),

[ID\_Spectacle] int constraint FK\_PerfomanceSpectaclePerson foreign key references [Spectacle](ID\_Spectacle)

)

CREATE TABLE [Shedule] --Таблица расписание

(

[ID\_Shedule] int constraint PK\_Shedule primary key,

[ID\_Spectacle] int constraint FK\_Spectacle foreign key references [Spectacle](ID\_Spectacle),

[Data\_Time] date

)

CREATE TABLE [Ticket] --Таблица билетов

(

[ID\_Ticket] int constraint PK\_Ticket primary key,

[ID\_Shedule] int constraint FK\_TicketShedule foreign key references [Shedule](ID\_Shedule) NOT NULL,

[Price] money,

[Count] int

)

# **ПРИЛОЖЕНИЕ Б**

USE TheaterDB;

GO

CREATE PROCEDURE [dbo].[sp\_InsertPerson] -- для добавления сотрудника

@Name nvarchar(50),

@Surname nvarchar (50),

@Patronymic nvarchar (50),

@Gender char,

@Birthday date,

@Mobile varchar(12),

@NameOfRole nvarchar (50),

@Experience int,

@On\_holiday bit,

@Salary money,

@Bankcard\_number varchar(16)

AS

INSERT INTO [Personal] ([Name], [Surname], [Patronymic], [Gender], [Birthday], [Mobile], [NameOfRole], [Experience], [On\_holiday], [Salary], [Bankcard\_number])

VALUES (@Name, @Surname, @Patronymic, @Gender, @Birthday, @Mobile, @NameOfRole, @Experience, @On\_holiday, @Salary, @Bankcard\_number)

GO

CREATE PROCEDURE [dbo].[sp\_InsertShedule] -- для добавления расписания

@ID\_Spec int out,

@DateSpec date

AS

INSERT INTO [Shedule] (ID\_Spectacle, Data\_Time)

VALUES (@ID\_Spec, @DateSpec)

GO

CREATE PROCEDURE [dbo].[sp\_InsertTicket] -- для добавления билета

@ID\_Spec int out,

@Price money,

@Count int

AS

INSERT INTO [Ticket] ([ID\_Shedule], [Price], [Count])

VALUES (@ID\_Spec, @Price, @Count)

GO

CREATE PROCEDURE [dbo].[sp\_InsertPersonToSpectacle] -- для добавления актеров к спетаклю

@ID\_Actor int out,

@ID\_Spec int out

AS

INSERT INTO [Peromance] ([ID\_Person], [ID\_Spectacle])

VALUES (@ID\_Actor, @ID\_Spec)

GO

CREATE PROCEDURE [dbo].[ob\_PersonalSelectAll] -- для выборки всех записей сотрудников

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT \* FROM [dbo].[Personal]

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_SpectacleSelectAll] -- для выборки всех записей спектаклей

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT \* FROM [dbo].[Spectacle]

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_SheduleSelectAll] -- для выборки всех записей распиания

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT [Spectacle].[ID\_Spectacle], [Spectacle].[NameOfSpectacle], [Shedule].[Data\_Time] FROM [dbo].[Shedule] INNER JOIN Spectacle ON [Spectacle].ID\_Spectacle = [Shedule].ID\_Spectacle

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_ActorToSpectacleSelectAll] -- для выборки всех сотрудников в спектакле !!!

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT [Spectacle].[NameOfSpectacle], [Personal].[Name], [Personal].[Surname], [Personal].[Patronymic], [Personal].[NameOfRole] FROM [dbo].[Spectacle] INNER JOIN [Peromance] ON [Peromance].[ID\_Spectacle] = [Spectacle].[ID\_Spectacle] INNER JOIN [Personal] ON [Peromance].[ID\_Person] = [Personal].[ID\_Person]

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_TicketSelectAll] -- для выборки всех записей билетов

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

--SELECT [Ticket].[ID\_Ticket], [Spectacle].[NameOfSpectacle], [Ticket].[Price], [Ticket].[Count] FROM [dbo].[Ticket] INNER JOIN Spectacle ON [Ticket].[ID\_Ticket] = [Spectacle].[ID\_Spectacle]

SELECT [Ticket].[ID\_Ticket], [Spectacle].[NameOfSpectacle], [Ticket].[Price], [Ticket].[Count] FROM [dbo].[Ticket] INNER JOIN [Shedule] ON [Ticket].ID\_Shedule = [Shedule].ID\_Shedule INNER JOIN [Spectacle] ON [Shedule].[ID\_Spectacle] = [Spectacle].[ID\_Spectacle]

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_PersobnalDeleteAll] -- для удаления всех записей

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

DELETE FROM [dbo].[Personal]

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_PersonalDeleteItem] -- для удаления одного сотрудника

@id int

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

DELETE FROM [dbo].[Personal] WHERE [ID\_Person] = @id

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_SpectacleDeleteItem] -- для удаления одного спектакль

@id int

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

DELETE FROM [dbo].[Spectacle] WHERE [ID\_Spectacle] = @id

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_SheduleDeleteItem] -- для удаления одного расписания

@id int,

@DateSpec date

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

DELETE FROM [dbo].[Shedule] WHERE [ID\_Spectacle] = @id AND [Data\_Time] = @DateSpec

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_TicketDeleteItem] -- для удаления одного сотрудника

@id int

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

DELETE FROM [dbo].[Ticket] WHERE [ID\_Ticket] = @id

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_ActorFromSpectacleDeleteItem] -- для удаления одного сотрудника

@idA int,

@idSP int

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

DELETE FROM [dbo].[Peromance] WHERE [ID\_Person] = @idA AND [ID\_Spectacle] = @idSP

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_PersonalSearchItem] -- для поиска сотрудника по ФИО

@Name nvarchar(50),

@Surname nvarchar (50),

@Patronymic nvarchar (50)

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT \* FROM [dbo].[Personal] WHERE [Name] = @Name AND [Surname] = @Surname AND [Patronymic] = @Patronymic

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_SpectacleSearchItem] -- для поиска спектаклей по названию и метки

@NameOfSpectacle nvarchar(50),

@AccountingSpectacle char

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT \* FROM [dbo].[Spectacle] WHERE [NameOfSpectacle] = @NameOfSpectacle AND [AccountingSpectacle] = @AccountingSpectacle

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_SheduleSearchItem] -- для поиска расписаний по дате

@DateSpec date

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT [Spectacle].[NameOfSpectacle], [Shedule].[Data\_Time] FROM [dbo].[Shedule] INNER JOIN [Spectacle] ON [Spectacle].ID\_Spectacle = [Shedule].ID\_Spectacle WHERE [Data\_Time] = @DateSpec

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_ACTSpecSearchItem] -- 12345678654324678

@idSP int

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT [Spectacle].[NameOfSpectacle], [Personal].[Name], [Personal].[Surname], [Personal].[Patronymic], [Personal].[NameOfRole] FROM [dbo].[Personal] INNER JOIN [Peromance] ON [Peromance].[ID\_Person] = [Personal].[ID\_Person] INNER JOIN [Spectacle] ON [Peromance].[ID\_Spectacle] = @idSP

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_PersonalComboBoxItem] -- для вывода сотрудников в combobox

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT [dbo].[Personal].[ID\_Person], [dbo].[Personal].[Name], [dbo].[Personal].[Surname], [dbo].[Personal].[Patronymic] FROM [dbo].[Personal]

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_SpectacleComboBoxItem] -- для вывода спектаклей в combobox

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

SELECT [dbo].[Spectacle].[ID\_Spectacle], [dbo].Spectacle.[NameOfSpectacle] FROM [dbo].[Spectacle]

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_TicketComboBoxItem] -- для вывода билетов в combobox

AS

SET NOCOUNT ON

SET XACT\_ABORT ON

BEGIN TRANSACTION

--SELECT [Spectacle].[NameOfSpectacle] FROM [dbo].[Ticket] INNER JOIN [Shedule] ON [Ticket].ID\_Shedule = [Shedule].ID\_Shedule INNER JOIN [Spectacle] ON [Shedule].[ID\_Spectacle] = [Spectacle].[ID\_Spectacle]

SELECT [Shedule].[ID\_Shedule], [Spectacle].[NameOfSpectacle] FROM [dbo].[Shedule] INNER JOIN [Spectacle] ON [Shedule].[ID\_Spectacle] = [Spectacle].[ID\_Spectacle]

COMMIT

GO

CREATE PROCEDURE [dbo].[ob\_SpectacleInsert] -- для внесения спектаклей

@NameOfSpectacle nvarchar(50),

@Description nvarchar (200),

@AccountingSpectacle char

AS

INSERT INTO [Spectacle] ([NameOfSpectacle], [Description], [AccountingSpectacle])

VALUES (@NameOfSpectacle, @Description, @AccountingSpectacle)

COMMIT

GO

CREATE PROCEDURE [exportXML] -- экспорт

AS

SELECT [ID\_Person], [Name], [Surname] , [Patronymic], [Gender], [Birthday], [Mobile], [NameOfRole], [Experience], [On\_holiday], [Salary], [Bankcard\_number]

FROM [dbo].[Personal]

FOR XML PATH('Person'), ROOT('Personals');

GO

--DBCC CHECKIDENT ('[dbo].[Personal]', RESEED, 0);

CREATE PROCEDURE [importXML] -- импорт xml таблица [Personal]

@xml XML = NULL

AS

Select @xml =

CONVERT(XML,bulkcolumn,2) FROM OPENROWSET(BULK 'D:\3\_1kurs\Oracle\Курсач\TheaterBD\xmlBD.xml',SINGLE\_BLOB) AS X

SET ARITHABORT ON

INSERT INTO [dbo].[Personal]([ID\_Person], [Name], [Surname] , [Patronymic], [Gender], [Birthday], [Mobile], [NameOfRole], [Experience], [On\_holiday], [Salary], [Bankcard\_number])

SELECT

P.value('ID\_Personal[1]', 'int') AS [ID\_Personal],

P.value('Name[1]','nvarchar(50)') AS [Name],

P.value('Surname[1]','nvarchar(50)') AS [Surname],

P.value('Patronymic[1]','nvarchar(50)') AS [Patronymic],

P.value('Gender[1]','char') AS [Gender],

P.value('Birthday[1]','date') AS [Birthday],

P.value('Mobile[1]','varchar(12)') AS [Mobile],

P.value('NameOfRole[1]','nvarchar(50)') AS [NameOfRole],

P.value('Experience[1]','int') AS [Experience],

P.value('On\_holiday[1]','bit') AS [On\_holiday],

P.value('Salary[1]','money') AS [Salary],

P.value('Bankcard\_number[1]','varchar(16)') AS [Bankcard\_number]

FROM @xml.nodes('//Personals/Person') PropertyFeed(P)

COMMIT

GO

exec [exportXML] ;

exec [importXML];