1. **ЛОГИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ**

**1.1 Информационно-логическая модель предметной области.**

**Описание сущностей, атрибутов, ключей, связей**

* + 1. Описание сущностей

При исследовании предметной области были выделены следующие сущности:

– “*Employees*” – хранит информацию о сотрудниках;

– “*Records*” – хранит информацию о документации;

– “*Schedulees*” – хранит информацию о расписании;

– “*Genres*” – хранит информацию о жанрах;

– “*Executors*” – хранит информацию об исполнителях.

***1.1.2*** Описание атрибутов сущностей

Атрибуты сущности “*Employees*”:

– *Number\_employee* – уникальный номер, однозначно идентифицирующий каждого сотрудника;

– *Name* – Имя сотрудника;

– *Surname* – Фамилия сотрудника;

– *Ochestvo* – Отчество сотрудника.

Атрибуты сущности “*Records*”:

– *Number\_composition* – номер композиции;

– *Number\_executor* – номер исполнителя;

– *Album* – альбом;

– *Year* – год;

– *Date\_record* – дата выхода композиции;

– *Length* – продолжительность композиции;

– *Number\_genre* – номер жанра.

Атрибуты сущности “*Schedulees*”:

– *Number\_schedule* – индексация расписания;

– *Date* – дата;

– *Time* – время;

– *Number\_employee* – код сотрудника;

– *Number\_composition* – код композиции;

– *Length* – продолжительность;

– *Rating* – рейтинг.

Атрибуты сущности “*Genres*”:

– *Number\_genre* – код жанра;

– *Description* – описание;

– *Name\_genre* – код жанра.

Атрибуты сущности “*Executors*”:

– *Number\_executor* – код исполнителя;

– *Executore* – исполнитель;

– *Compos* – композиция.

***1.1.3*** Описание ключей

Для того чтобы однозначно идентифицировать каждый экземпляр всех сущностей, необходимо выбрать для каждой сущности ключевой атрибут:

– “*Employees*” – в качестве первичного ключа выбрано поле “*Number\_employee*”, так как он однозначно идентифицирует каждого сотрудника, поскольку не имеет совпадений с кодами других сотрудников. В качестве внешнего ключа выбран атрибут “*Number\_employee*” для связи с сущностью “*Schedulees*”;

– “*Records*” – в качестве первичного ключа выбрано поле “*Number\_composition*”, так как он однозначно идентифицирует каждую запись данных. В качестве внешних ключей выбраны атрибуты “*Number\_executor*” для связи с сущностью “*Executors*” и “*Number\_genre*” для связи с сущностью “*Genres*”.

– “*Schedulees*” – в качестве первичного ключа выбран атрибут “*Number\_schedule*”. В качестве внешних ключей выбраны атрибуты “*Number\_employee*” для связи с сущностью “*Employees*” и “*Number\_composition*” для связи с сущностью “*Records*”;

– “*Genres*” – в качестве первичного ключа выбрано поле “*Number\_genre*”. В качестве внешнего ключа выбран атрибут “*Number\_genre*” для связи с сущностью “*Records*”;

– “*Executors*” – в качестве первичного ключа выбрано поле “*Number\_executor*” В качестве внешнего ключа выбран атрибут “*Number\_executor*” для связи с сущностью “*Records*”.

***1.1.4*** Описание связей

Опишем типы связей между сущностями:

– “*Employees*”→ “*Schedulees*”: тип связи “один ко многим”;

– “*Schedulees*” → “*Employees*”: тип связи “многие к одному”;

– “*Schedulees*” → “*Records*”: тип связи “многие к одному”;

– “*Records*” → “*Schedulees*”: тип связи “один ко многим ”;

– “*Records*” → “*Executors*”: тип связи “многие к одному”;

– “*Records*”→ “*Genres*“: тип связи “многие к одному”;

– “*Executors*”→“*Records*”: тип связи “один ко многим”;

– “*Genres*” → “*Records*”: тип связи “один ко многим”.

Также связи могут быть идентифицирующими или не идентифицирующими. Если задана идентифицирующая связь, то это значит, что каждому экземпляру дочерней сущности должен соответствовать хотя бы один экземпляр родительской [3]. Не идентифицирующая связь означает, что экземпляр дочерней сущности может быть идентифицирован без использования экземпляра родительской сущности [3]. Все первичные ключи в сущностях являются простыми – состоят из одного поля и однозначно идентифицируют запись.

1. **ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ. АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ**

**2.1 Диаграмма базы данных. Описание состава, типов данных и**

**свойств полей таблиц базы данных**

Диаграмма базы данных приведена на рисунке 2.1.

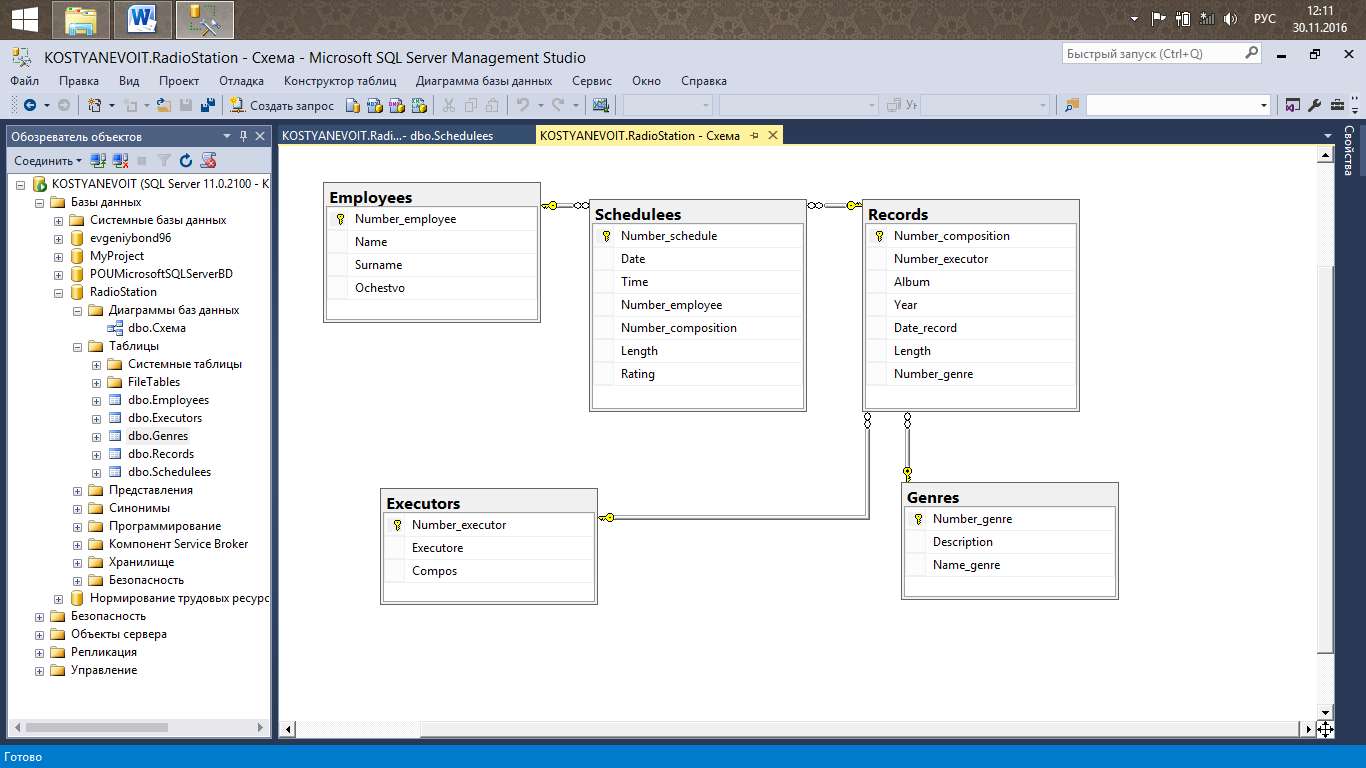


Рисунок 2.1 – Диаграмма базы данных

Структуры таблиц разработанной БД приведены в таблицах 2.1, 2.2, 2.3, 2.4, 2.5.

В таблице 2.1 представлена информация о полях, типах данных, индексации и разрешении пустого поля таблицы “*Employees*”.

Таблица 2.1 – Таблица “*Employees*”

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Индексация | Разрешить NULL |
| *Number\_employee* | *int* | Да | Нет |
| *Name* | *varchar(20)* | Нет | Да |
| *Surname* | *varchar(20)* | Нет | Да |
| *Ochestvo* | *varchar(20)* | Нет | Да |

В таблице 2.2 представлена информация о полях, типах данных, индексации и разрешении пустого поля таблицы “*Records*”.

Таблица 2.2 – Таблица “*Records*”

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Индексация | Разрешить NULL |
| *Number\_composition* | *int* | Да | Нет |
| *Number\_executor* | *int* | Нет | Да |
| *Album* | *varchar(20)* | Нет | Да |
| *Year* | *int* | Нет | Да |
| *Date\_record* | *date* | Нет | Да |
| *Length* | *int* | Нет | Да |
| *Number\_genre* | *int* | Нет | Да |

В таблице 2.3 представлена информация о полях, типах данных, индексации и разрешении пустого поля таблицы “*Schedulees*”.

Таблица 2.3 – Таблица “*Schedulees*”

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Индексация | Разрешить NULL |
| *Number\_schedule* | *int* | Да | Нет |
| *Date* | *date* | Нет | Да |
| *Time* | *datetime* | Нет | Да |
| *Number\_employee* | *int* | Нет | Да |
| *Number\_composition* | *int* | Нет | Да |
| *Length* | *int* | Нет | Да |
| *Rating* | *int* | Нет | Да |

В таблице 2.4 представлена информация о полях, типах данных, индексации и разрешении пустого поля таблицы “*Genres*”.

Таблица 2.4 – Таблица “*Genres*”

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Индексация | Разрешить NULL |
| *Number\_genre* | *int* | Да | Нет |
| *Description* | *varchar(100)* | Нет | Да |
| *Name\_genre* | *varchar(20)* | Нет | Да |

В таблице 2.5 представлена информация о полях, типах данных, индексации и разрешении пустого поля таблицы “*Executors*”.

Таблица 2.5 – Таблица “*Executors*”

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Индексация | Разрешить NULL |
| *Number\_executor* | *int* | Да | Нет |
| *Executore* | *varchar(20)* | Нет | Да |
| *Compos* | *varchar(20)* | Нет | Да |

**2.2 Обоснование соответствия разработанной базы данных требованиям нормализации**

Сущность “*Employees*”. Все поля данной таблицы являются простыми (неделимыми). Выполнено условие 1НФ. Таблица имеет первичный ключ, состоящий из одного столбца. Отсюда следует, что она автоматически находится в 2НФ. Т.к. отсутствует транзитивная зависимость полей (все описательные поля зависят от ключа), то таблица находится в 3НФ.

Сущность “*Records*”. Таблица имеет первичный ключ, состоящий из одного столбца. Значения, содержащиеся в полях таблицы атомарны. Все не ключевые поля функционально связаны с ключом. Между не ключевыми полями нет зависимостей. Значит, таблица соответствует требованиям 3НФ.

Сущность “*Schedulees*”. Таблица имеет первичный ключ, состоящий из одного столбца. Поля атомарны, транзитивная зависимость отсутствует.

Сущность “Диски”. Поля таблицы неделимы (1НФ), функционально зависят от ключа (2НФ) и отсутствует транзитивная зависимость полей (3НФ).

Сущность “*Genres*”. Условие соответствия 3НФ выполнены, т.к. все поля неделимы, первичный ключ состоит из одного столбца (автоматически 2НФ) и нет транзитивной зависимости не ключевых полей.

Сущность “*Executors*”. Условие соответствия 3НФ выполнены, т.к. все поля неделимы, первичный ключ состоит из одного столбца (автоматически 2НФ) и нет транзитивной зависимости не ключевых полей.

**2.3 Создание представлений**

Разработанные представления:

– Отдел кадров (данные обо всех сотрудниках и их должностях, отображение сведений о сотрудниках с определенными: ФИО, образованием). SQL-код представлен на рисунке 2.2.

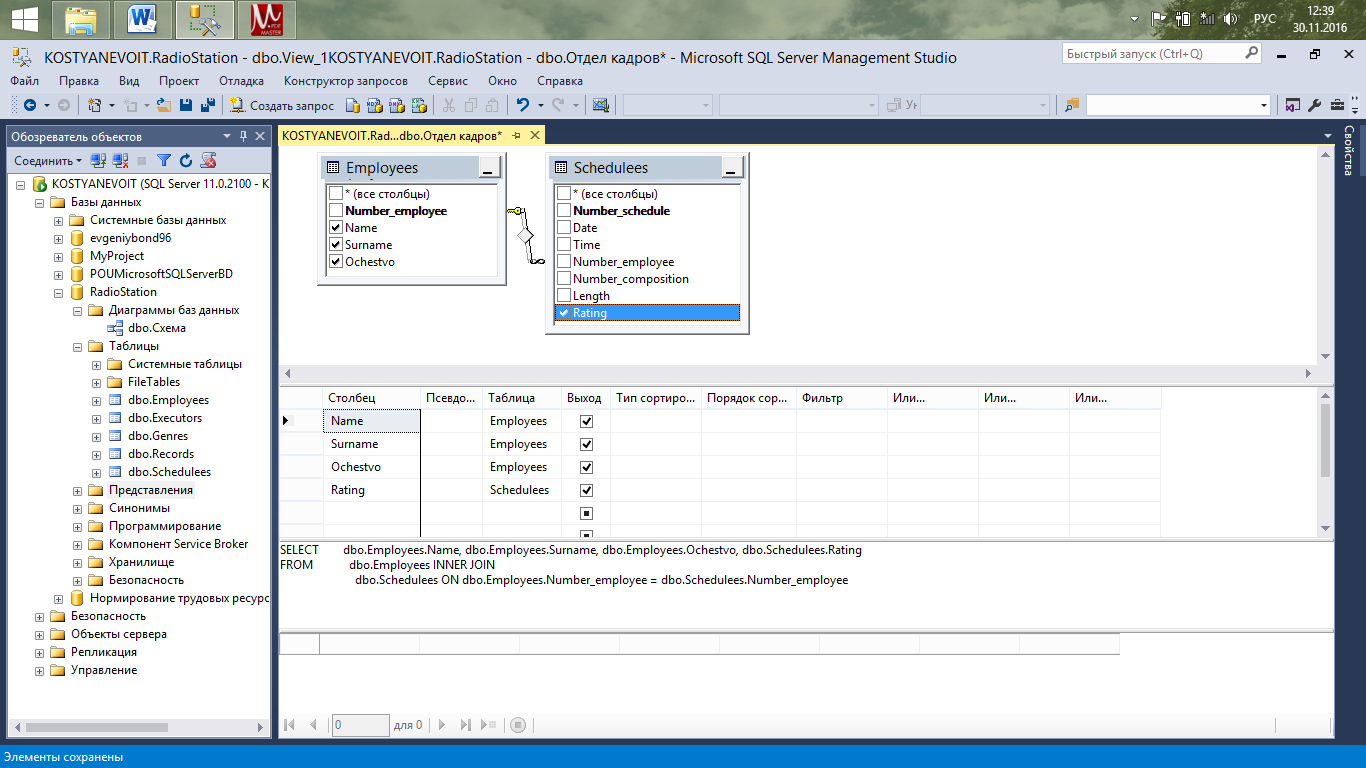


Рисунок 2.2 – SQL-код представления “Отдел кадров”

– Музыкальный архив (полные сведения обо всех записях; сведения о записях заданных: исполнителя, жанра). SQL-код представлен на рисунке 2.3.

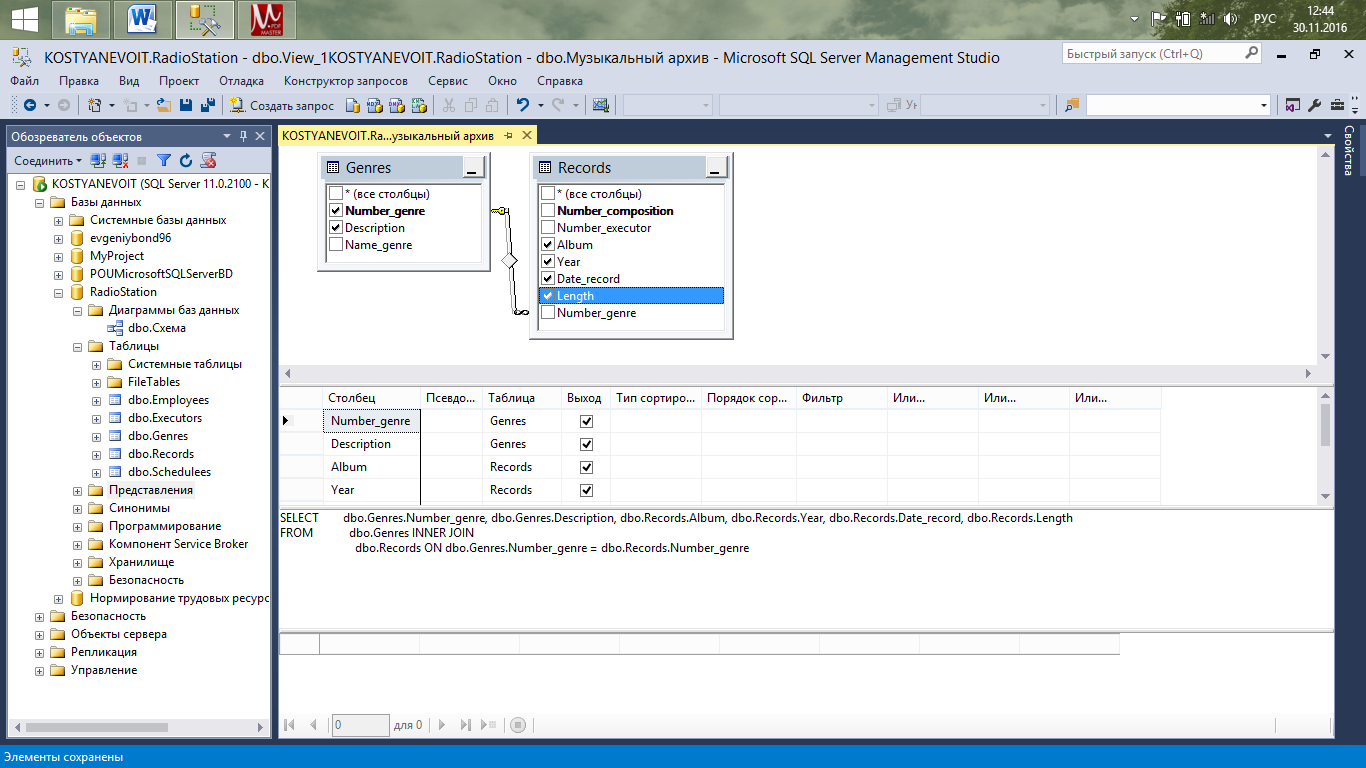


Рисунок 2.3 – SQL-код представления “Музыкальный архив”

– Сетка вещания (полные сведения о рабочем времени всех сотрудников на текущую неделю; сведения о времени работы определенного сотрудника; сведения о датах трансляции определенных композиций; сведения о композициях, которые не транслировались в заданном диапазоне времени). SQL-код представлен на рисунке 2.4.

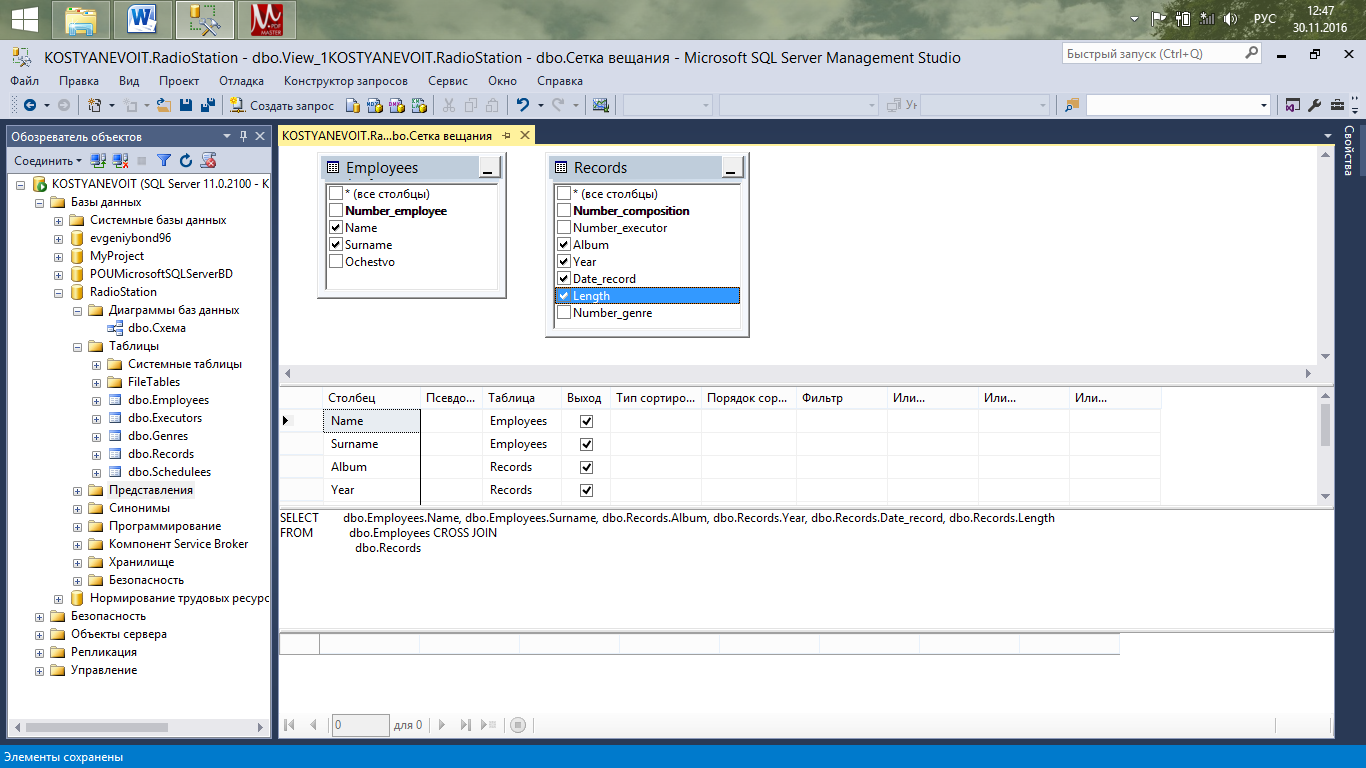


Рисунок 2.4 – SQL-код представления “Сетка вещания”

**2.4Аппаратное и программное обеспечение системы**

Разработанная база данных содержит одну файловую *PRIMARY*, которая создаётся по умолчанию и состоит из двух основных файлов:

– *RadioStation.mdf* – файл самой базы данных. В нём хранятся непосредственно данные и структура базы данных. Размер данного файла 90МБ, максимальный размер файла 104МБ;

– *RadioStation\_log.ldf* – журнал транзакций. Размер журнала 90МБ, максимальный размер файла 104МБ.

Перечислим минимальные программные и аппаратные требования серверной части для работы с ИС [4]:

– процессор: *Intel P4* - 1GHz;

– память (ОЗУ): 512 MB;

– операционная система: *Windows Server* 2003 *Service Pack* 2; *Windows Server* 2008; *Windows Vista*; *Windows Vista Service Pack* 1; *Windows* 7; *Windows* 8.

Перечислим минимальные программные и аппаратные требования клиентской части для работы с ИС:

– процессор: *Pentium III* – совместимый процессор с тактовой частотой 1 GHz;

– оперативная память (*RAM*): 256 MB;

– объём жёсткого диска (*HDD*): 500 MB;

– монитор: видеоадаптер *Super VGA* с разрешением 800х600 или выше;

– операционная система: *Windows Vista*; *Windows* 7; *Windows* 8.

1. **СТРУКТУРА WEB-ПРИЛОЖЕНИЯ**

**3.1 Описание основной структуры ASP.NET документов**

На веб-сайте *ASP.NET URL*-адреса обычно сопоставляются с файлами, которые хранятся на диске (обычно с *ASPX*-файлами). Эти *ASPX*-файлы содержат разметку и код, который обрабатывается для ответа на запрос.

Платформа *ASP.NET MVC* сопоставляет *URL*-адреса с кодом сервера способом, отличным от сопоставления страниц веб-форм *ASP.NET*.Вместо сопоставления *URL*-адресов со страницами или обработчиками *ASP.NET* платформа сопоставляет их с классами контроллеров. Классы контроллера обрабатывают входящие запросы, например: ввод информации пользователем и его действия, а также реализуют соответствующую логику приложений и данных на основании введенной информации. Класс контроллера обычно вызывает отдельный представления, который создает в качестве ответа вывод *HTML*.

*ASP.NET MV*C – это *framework* для создания *Web*-приложений на платформе *.NET*, который изначально развивался сторонними разработчиками как альтернатива *WebForms* и очень быстро получил популярность в сообществе разработчиков. В настоящее время он официально поддерживается и развивается сотрудниками корпорации *Microsoft*.

Данный *framework* является реализацией шаблона *Model – View – Controller* (модель – представление – контроллер), который хорошо подходит для написания *web*-приложений. Шаблон подразумевает разделение приложения на три основные компоненты, каждая из которых отвечает за свои задачи.

Модель отвечает за бизнес-логику приложения. В ней располагаются классы, отвечающие за работу с данными приложения. Сюда входят классы-сущности данных, методы для получения, изменения, сохранения данных. Данный компонент предоставляет данные и реагирует на запросы, изменяя свое состояние. [3].

Представление отвечает за отображение информации (пользовательский интерфейс). В отличие от *WebForms* код отображения пишется вручную, однако существуют специальные расширяющие методы, генерирующие код в соответствии с переданным набором параметров.

Контроллер является связующим звеном между первыми двумя компонентами. Обрабатывает данные, введённые пользователем, и информирует модель и представление о необходимости соответствующей реакции.

В платформе *ASP.NET MVC* компоненты модели, представления и контроллера разделены. Модель представляет бизнес-логику или логику домена приложения. При этом данные обычно предоставляются из базы данных. Представление выбирается контроллером и создает соответствующий пользовательский интерфейс. Платформа *ASP.NET MVC* по умолчанию использует существующие типы страницы *ASP.NET* (*ASPX*), главной страницы (*MASTER*) и типы пользовательских элементов управления (*ASCX*) для отображения в браузере. Контроллер выполняет в себе поиск соответствующего метода действия, получает значения для использования в качестве аргументов метода, а затем обрабатывает ошибки, возникающие при выполнении метода. После этого выполняется отображение требуемого представления. По умолчанию наборы компонентов хранятся в отдельных папках проекта веб-приложения *MVC*.

Платформа *ASP.NET MVC* использует механизм маршрутизации *ASP.NET*, который обеспечивает гибкость при сопоставлении *URL*-адресов с классами контроллера. Пользователь также может определять правила маршрутизации, которые используются платформой *ASP.NET MVC* для оценки входящих *URL*-адресов и выбора соответствующего контроллера.

Пользователь также может настроить механизм маршрутизации на автоматический анализ переменных, определенных в *URL*-адресе, с последующей передачей значений в качестве аргументов параметра из платформы *ASP.NET MVC* в контроллер.

Платформа *ASP.NET MVC* не использует модель обратной передачи веб-форм *ASP.NET* для взаимодействия с сервером. Все взаимодействия с пользователем передаются в класс контроллера. Это обеспечивает независимость логики пользовательского интерфейса от бизнес-логики и облегчает тестирование. Таким образом события состояния просмотра *ASP.NET* и жизненного цикла страниц веб-форм *ASP.NET* не связаны с представлениями на основе *MVC*.

В состав платформы *ASP.NET MVC* входит шаблон проекта *Visual Studio*, который позволяет создавать веб-приложения, структура которых соответствует шаблону *MVC*. Этот шаблон создает новое веб-приложение *MVC*, конфигурация которого предусматривает все необходимые папки, шаблоны элементов и записи файла конфигурации.

При создании нового веб-приложения *MVC* *Visual Studio* предоставляет возможность создания двух проектов одновременно. Первый проект является веб-проектом, в котором реализуется приложение. Второй проект представляет собой проект модульного теста, в котором возможно создание модульных тестов для компонентов *MVC* первого проекта.

В *ASP.NET MVC* код логики отделен от кода представления, поэтому приложения на *ASP.NET MVC*гораздо проще поддаются модульному тестированию в отличие от приложений, написанных на *WebForms*, где присутствует смешение кода логики и представления.

Необходимость вручную писать код представления страниц в *ASP.NET MVC* приводит к увеличению времени разработки, однако позволяет получить полный контроль над разметкой и создавать более чистый *HTML* код в отличие от *WebForms*.

С точки зрения поисковой оптимизации (*SEO*) следует учитывать тот факт, что в *WebForms* ключевые параметры запроса передаются в *GET* и в *POST*. Это ухудшает индексирование поисковыми системами, а также может привести к тому, что сохраненная ссылка на страницу может содержать информацию отличную от сохраняемой. В *MVC* ключевые параметры указаны в *URL*, поэтому такие страницы хорошо индексируются поисковыми системами, а также являются более наглядными. [4].

Для тестирования приложений *ASP.NET MVC* можно использовать любую платформу модульного тестирования, совместимую с платформой *.NET Framework. Visual Studio* поддерживает тестирование проектов в *MSTest*. [5].

Структура данного *web*-приложения.

Каталог *App\_Data* в него помещаются закрытые данные, такие как XML-файлы или базы данных, если используется SQL Server Express, SQLite или другие хранилища на основе файлов.

Каталог *App\_Start* содержит ряд основных настроек конфигурации для проекта, в том числе определение маршрутов и фильтров, а также пакетов содержимого.

Каталог *bin* сюда помещается скомпилированная сборка приложения *MVC* вместе со всеми ссылаемыми сборками.

Каталог *Content* сюда помещается статическое содержимое, такое как *CSS*-файлы и изображения сотрудников.

Каталог *Controllers* сюда помещаются классы контроллеров.

Каталог *Models* сюда помещаются классы моделей представлений и моделей предметной области.

Каталог *Scripts* предназначен для хранения библиотек *JavaScript*, используемых в приложении. По умолчанию *Visual Studio* добавляет библиотеки *jQuery* и несколько других популярных *JavaScript*-библиотек.

Каталог *Views* хранит представления и частичные представления, сгруппированные вместе в папках с именами контроллеров, с которыми они связаны.

Каталог *Views/Shared* хранит компоновки и представления, не являющиеся специфичными для какого-либо контроллера.

Файл *Web.config* содержит конфигурационную информацию, которая обеспечивает обработку представлений с помощью *ASP.NET* и предотвращает их обслуживание веб-сервером *IIS*, а также пространства имен, по умолчанию импортируемые в представления.

**3.2 Объектные модели**

Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

Для генерации объектной модели был выбран подход *DatabaseFirst*. Результат представлен на рисунке 3.4.

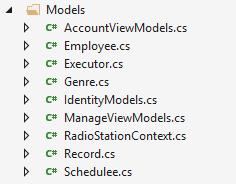


Рисунок 3.4 – Структура модели

Так же на рисунке 3.5 представлена схема модель.

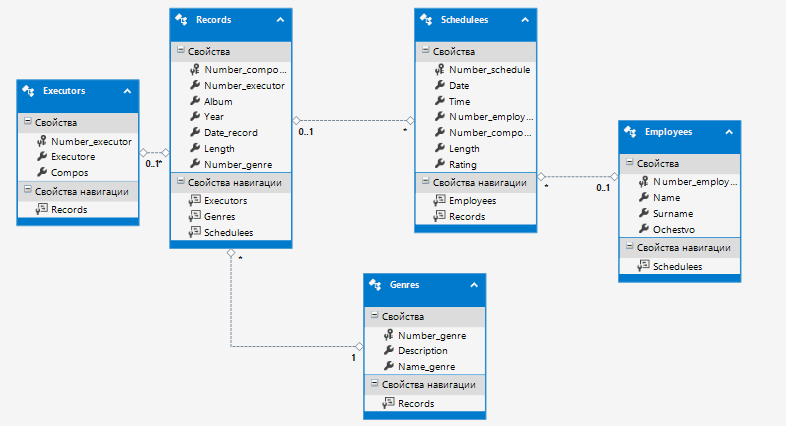


Рисунок 3.5 – Схема модели

**3.3 Контроллеры**

*Контроллер* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Controller*). Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Пример создание контроллера представлен на рисунке 3.6.

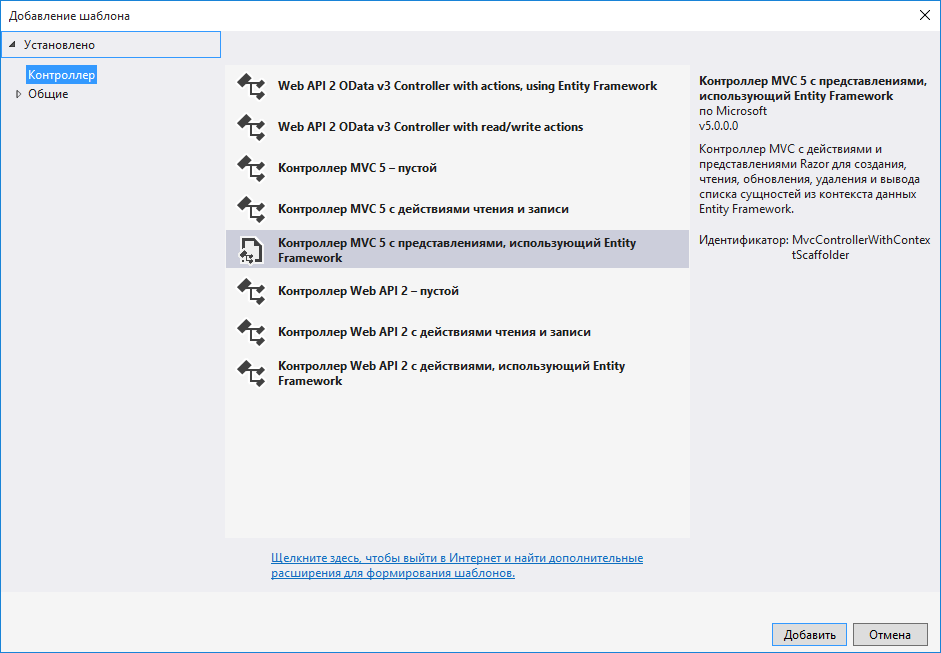


Рисунок 3.6 – Создание нового контроллера

Так же на рисунке 3.7 представлена структура всех созданных контроллеров приложения.

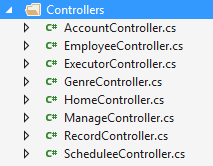


Рисунок 3.7 – Структура контроллеров

С подробным текстом кода контроллеров можно ознакомиться в приложении А.

**3.4 Представления**

*Представление*, *вид* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *View*). Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает [форма (окно)](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D0%BD%D0%BE_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) с графическими элементами.

Пример создания представления представлен на рисунке 3.8.

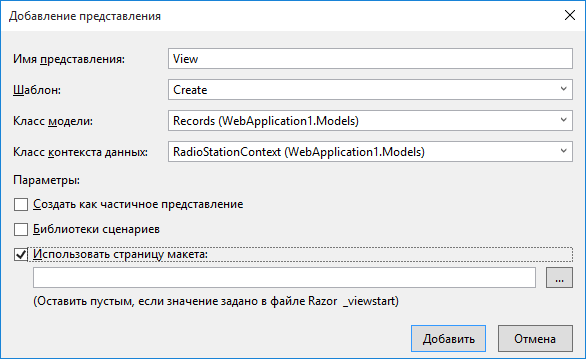


Рисунок 3.8 – Создание нового представления

На рисунке 3.9 представлена структура всех созданных представлений приложения.

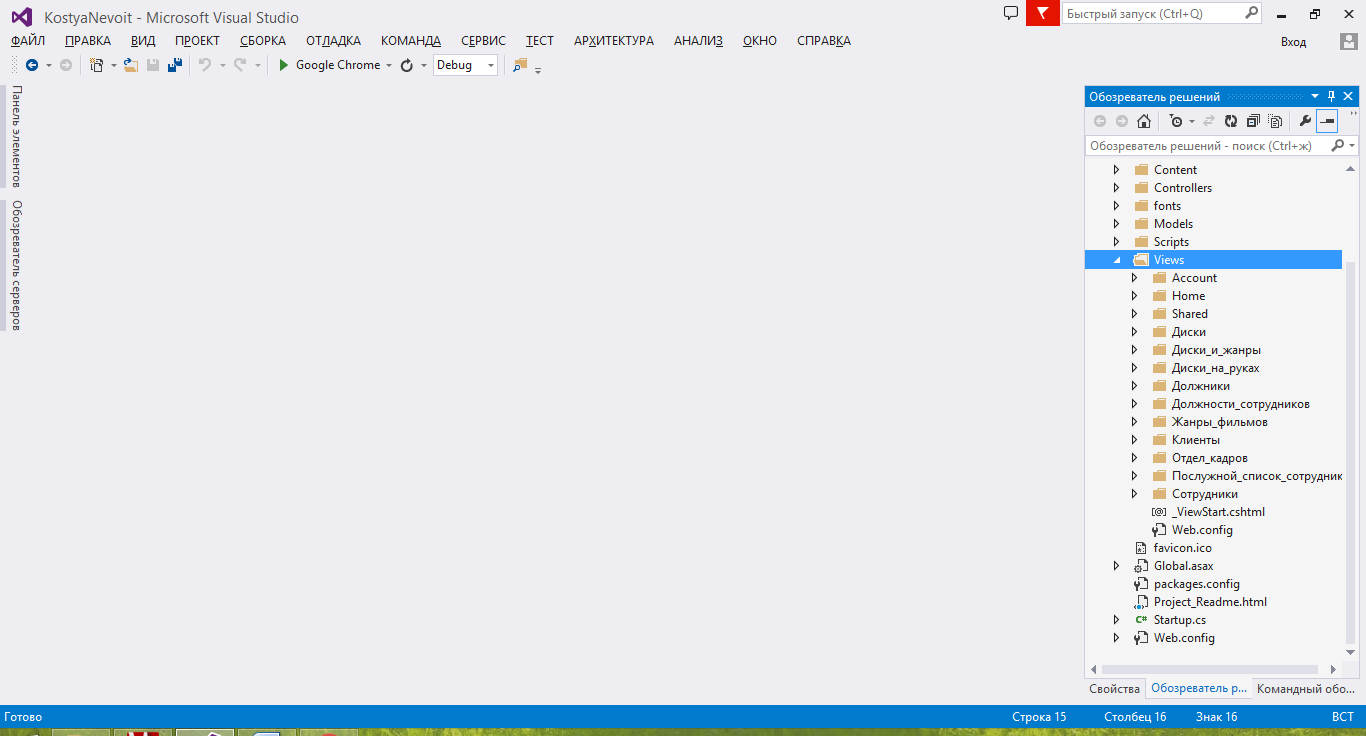


Рисунок 3.9 – Структура созданных представлений

С подробным текстом кода всех созданных представлений можно ознакомиться в приложении А.

1. **РУКОВОДСТВОПОЛЬЗОВАТЕЛЯ**

Для работы с базой данных пользователю не нужно иметь специальных навыков, достаточно лишь ознакомиться с данной пояснительной запиской или с литературными источниками, использовавшимися для выполнения курсового проекта, список которых приведен ниже.

Разработанное приложение позволяет просматривать все планируемые мероприятия, а также вносить изменения. Приложения так же редактирует и добавляет данные в базу данных.

Для отображения и редактирования информации в базе данных, а также для выполнения дополнительных функций по обработке данных был создан набор *ASP.NET* страниц. Пользователь имеет возможность перейти к редактированию любой таблицы путём выбора надписи с её именем в списке. Главная страница проекта, изображённая на рисунке 4.1.

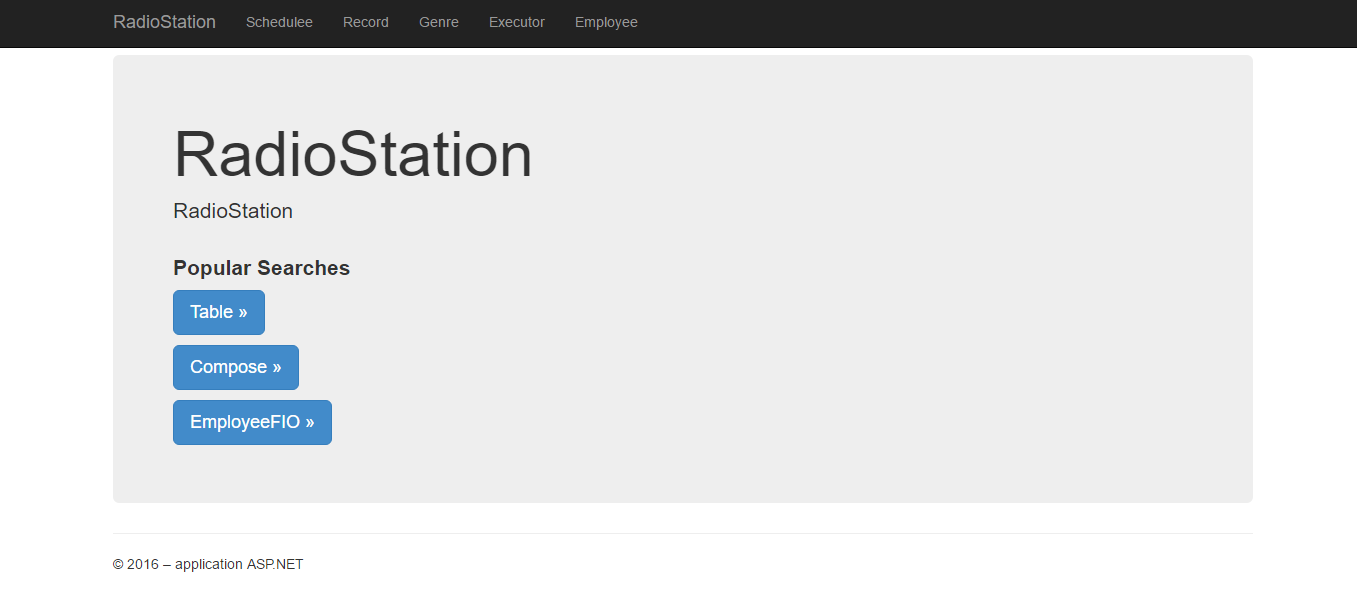


Рисунок 4.1 – Вид главной страницы

При выборе пункта меню «*Shedulee*», отображается страница, представленная на рисунке 4.3. На странице отображается таблица с полями: дата, время, название, номер композиции, длина, рейтинг. Возможно редактирование, добавление, удаление данных в таблицу.

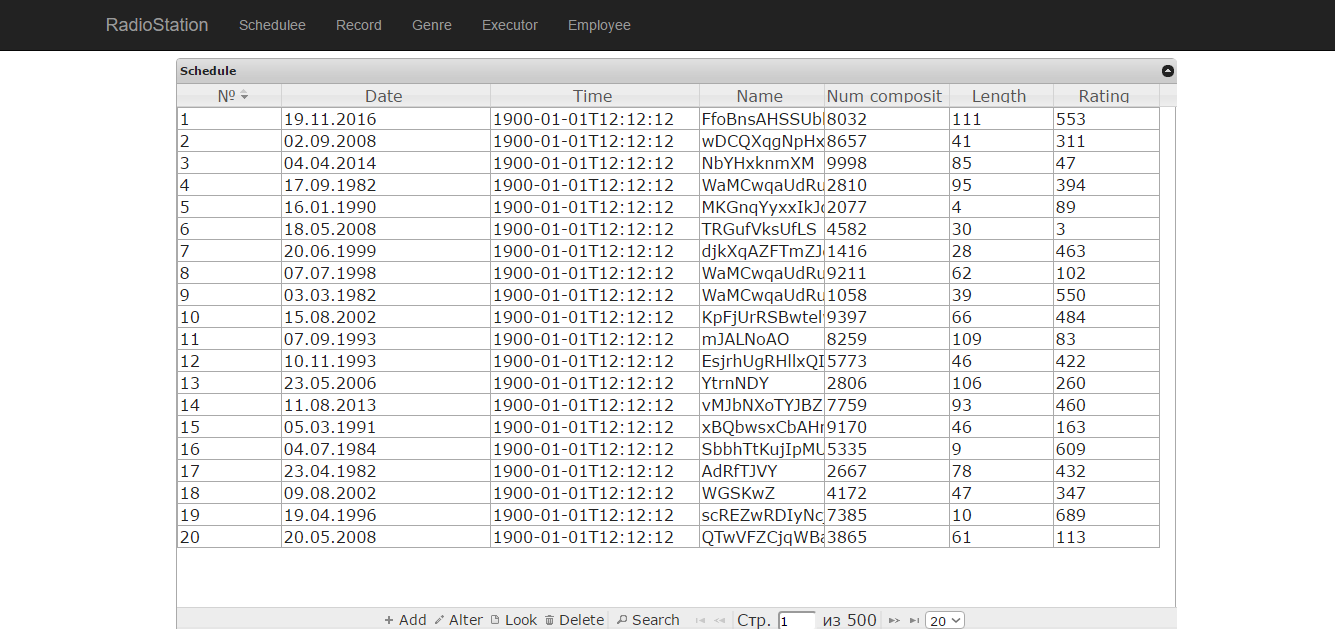


Рисунок 4.3 – Вид страницы «*Shedulee*»

У пользователя имеется возможность поиска нужного ему наименования исполнителя, воспользовавшись соответствующим полем для поиска. Пример показан на рисунке 4.4.

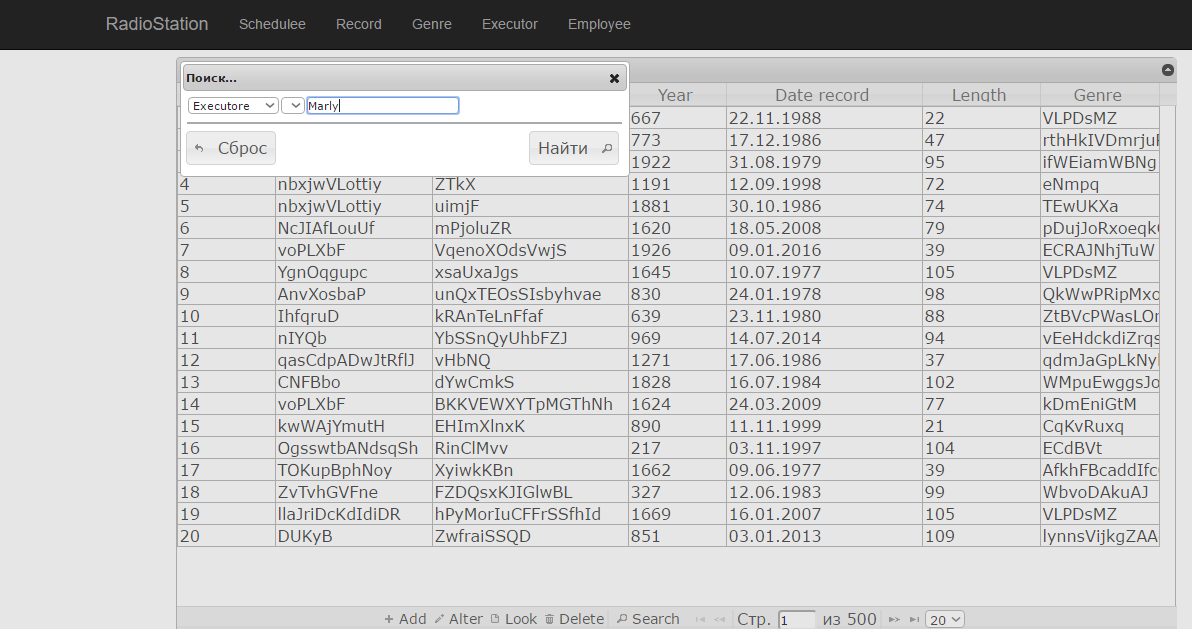


Рисунок 4.4 – Поиск по предприятию в таблице «*Record*»

Так же пользователь может добавить новую запись, нажав на кнопку “*Add*”. Пример показан на рисунке 4.5.

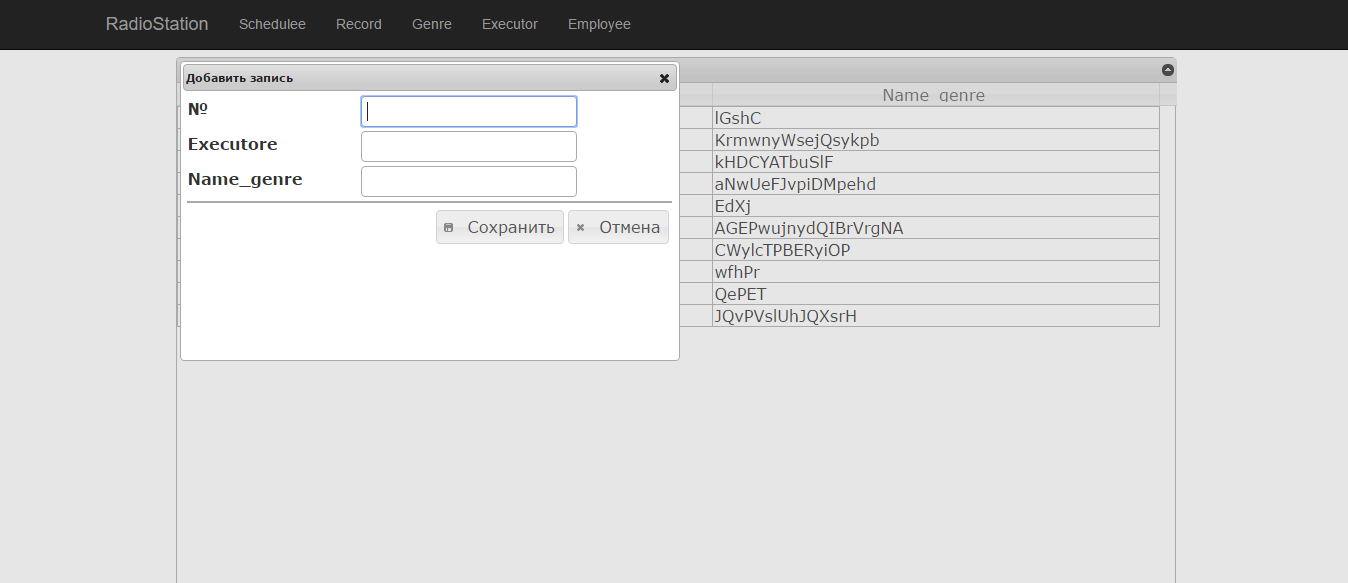


Рисунок 4.5 – Добавление новой записи в таблицу «*Executor*»

Если пользователь хочет отредактировать определенную запись, то у него имеется такая возможность, нажав на кнопку “*Edit*”. Пример показан на рисунке 4.6.

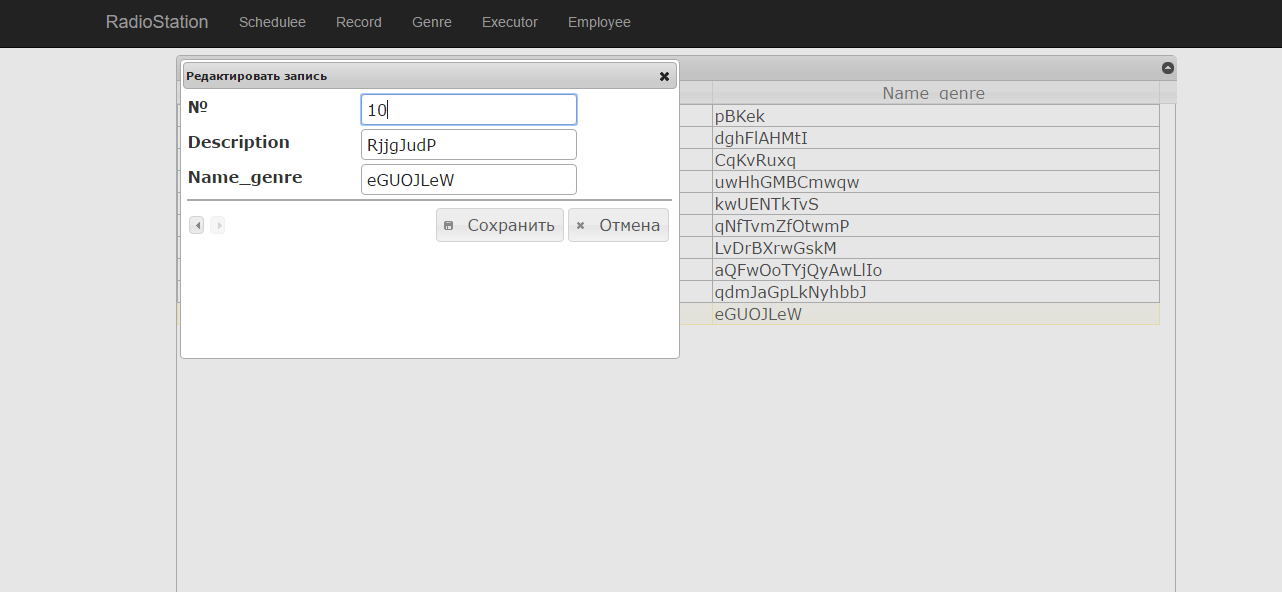


Рисунок 4.6 – Редактирование записи в таблице «*Genre*»

Так же пользователь может удалить запись из таблицы, воспользовавшись кнопкой “*Delete*”. Пример показан на рисунке 4.7.

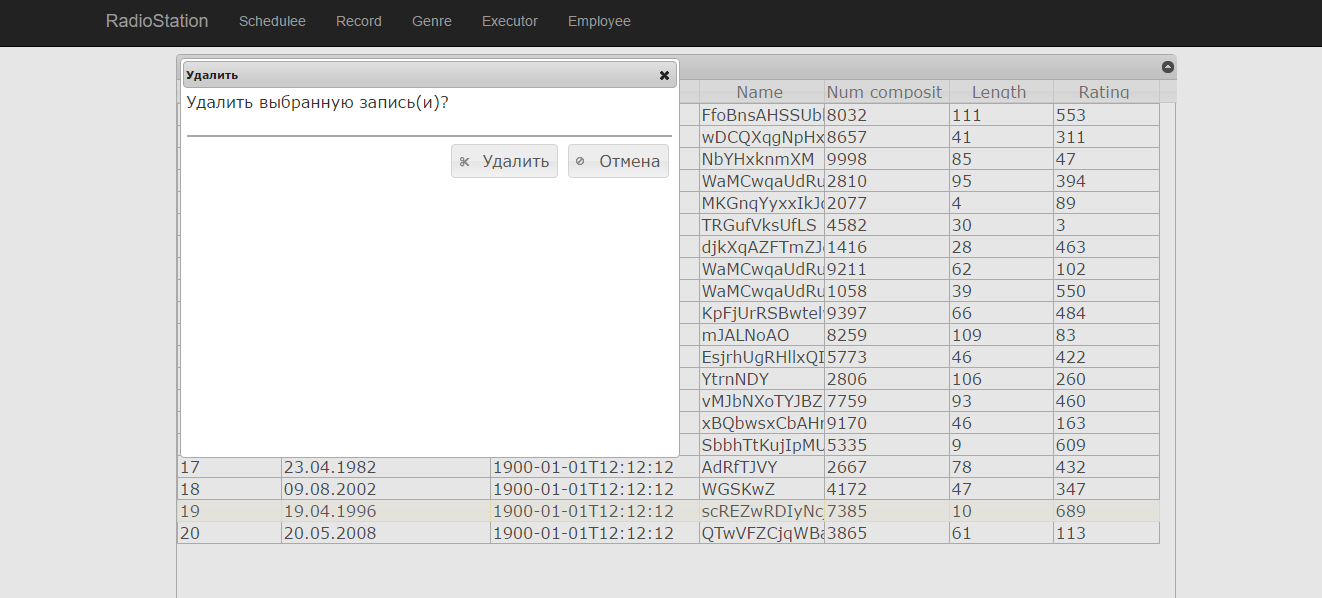


Рисунок 4.7 – Удаление записи из таблицы «*Shedulee*»

При выборе пункта меню «Отдел кадров», отображается страница, показанная на рисунке 4.8. На странице отображена таблица с данными: Ф.И.О., номер планируемого мероприятия, телефон, должность, дата занятия должности, пол, образование.

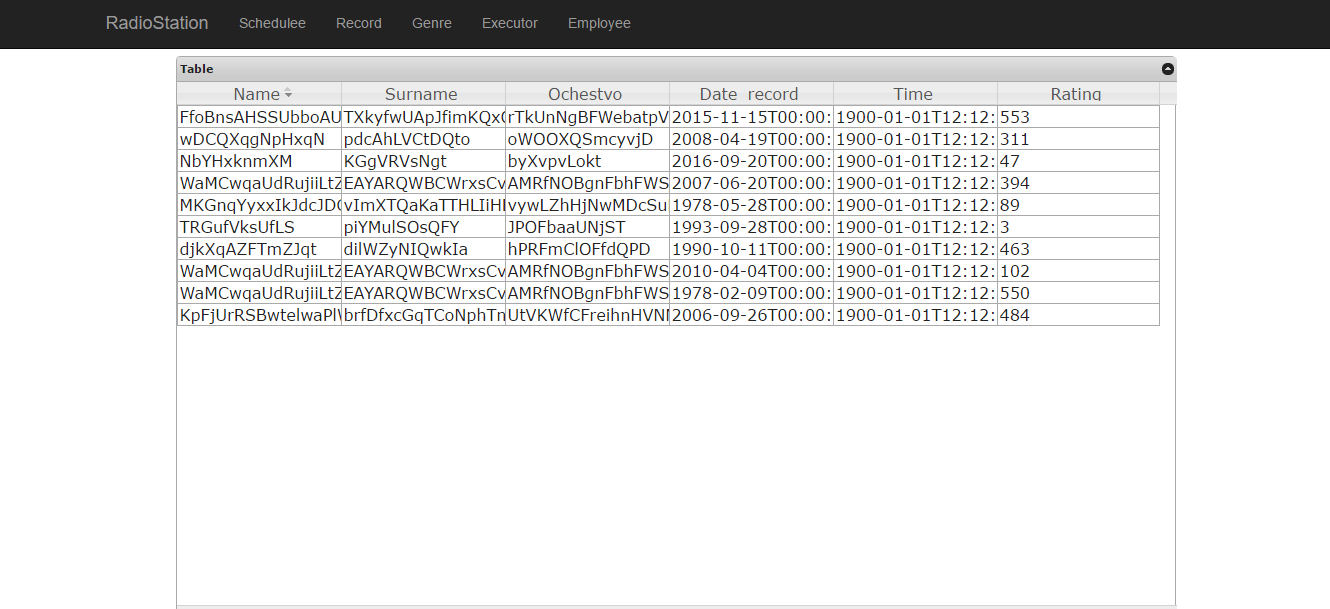


Рисунок 4.8 – Вид страницы «Отдел кадров»

При выборе пункта меню «Музыкальный архив», отображается страница, показанная на рисунке 4.9. На странице представлена таблица с данными: полные сведения обо всех записях,исполнитель, жанр.

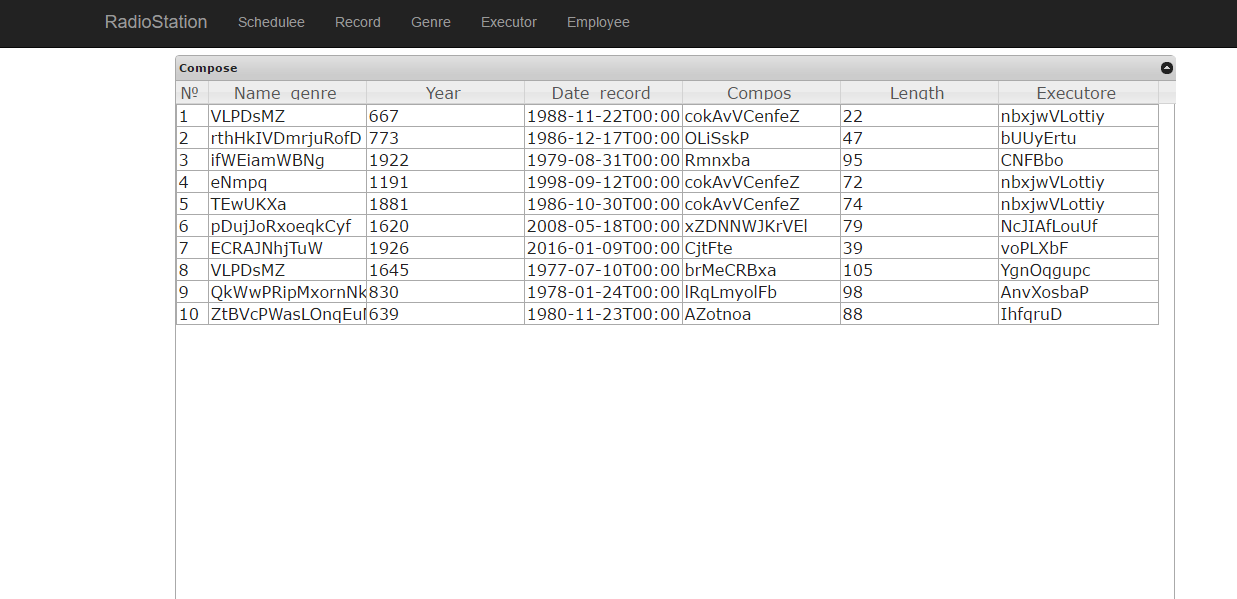


Рисунок 4.9 – Вид страницы «Музыкальный архив»

При выборе пункта меню «Сетка вещания», отображается страница, показанная на рисунке 4.10. На странице представлена таблица с данными: полные сведения о рабочем времени всех сотрудников на текущую неделю; сведения о времени работы определенного сотрудника; сведения о датах трансляции определенных композиций; сведения о композициях, которые не транслировались в заданном диапазоне времени.

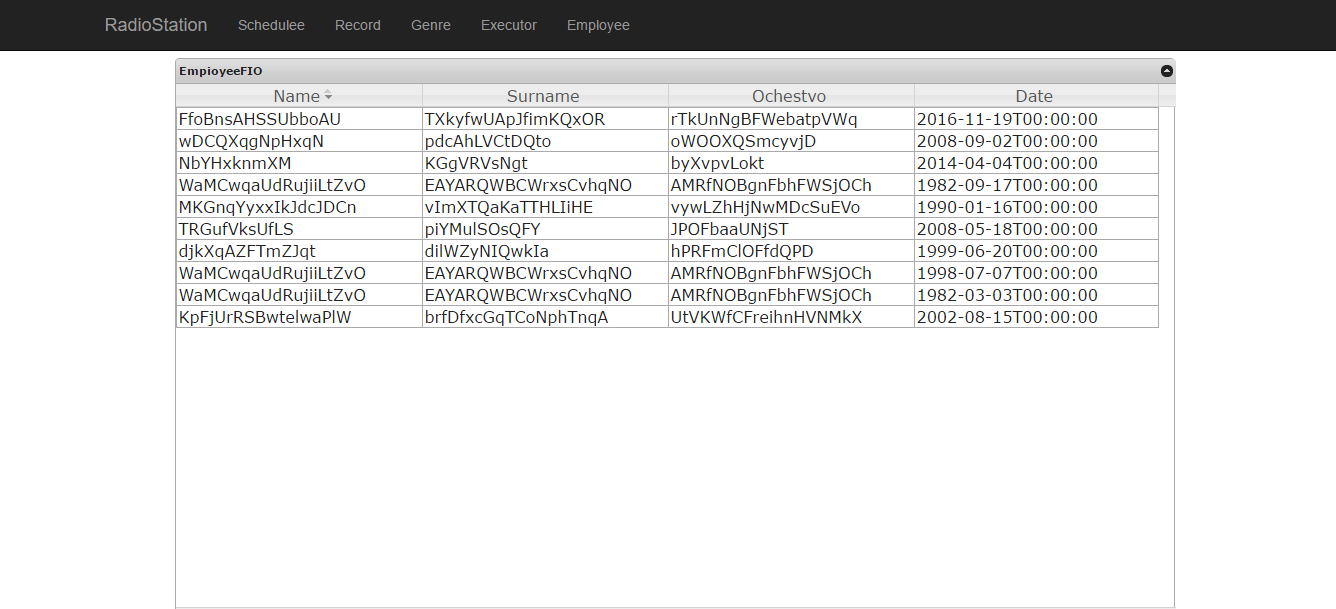


Рисунок 4.10 – Вид страницы «Сетка вещания»

1. **РУКОВОДСТВОПРОГРАММИСТА**

Разработанная информационная система позволяет добавлять, редактировать, удалять и отображать данные о сотрудниках, об их должностях, послужных списках. А также данные о мероприятиях, их подробном описании. Приложение «Радиостанция» позволит автоматизировать работу сотрудников, что уменьшит их труд и затрачиваемое время.

Для нормального функционирования программы необходимы следующие технические средства и программное обеспечение:

– процессор *Intel Pentium III* 1 ГГц и выше;

– объем оперативного запоминающего устройства 512 Мб;

– видеоадаптер должен поддерживать разрешение 1024х768 точек

– монитор с возможностью поддерживать разрешение 1024х768 с частотой не менее 75 Гц, отображающий 256 и более цветов

– устройства ввода информации – клавиатура и манипулятор типа «мышь»;

– 10 Мб свободного места на жёстком диске;

– операционная система *Windows Vista*/7/8

– *Microsoft.NET Framework* 4.5

Никаких ограничений и требований на режим работы и запуска программы не налагается. Для контроля правильности выполнения программы предусмотрены сообщения пользователю о некорректно введенных данных. Средства восстановления программы после сбоя не предусмотрены.

Для исключения бесполезной, неинформативной или противоречивой информации, введённой пользователем, в приложении осуществляется проверка вводимых данных. Критерии проверки могут быть самыми разными, начиная с того, вводились ли данные вообще, и заканчивая проверкой типа данных.

На рисунке 5.1 показана возможность изменения количества отображаемых строк в таблице.

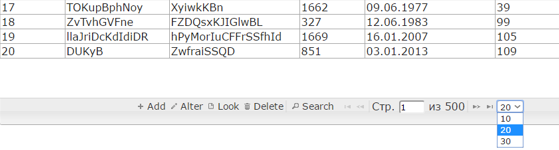


Рисунок 5.1 – Изменение количества строк

Ввод корректных данных, обеспечивается использованием выпадающих списков, которые появляются, если наши связи сделаны правильно. Вместо поля *Id* будет, автоматически сформирован список из следующего поля стоящего после первичного ключа. На рисунке 5.2 изображен пример такого выпадающего списка.

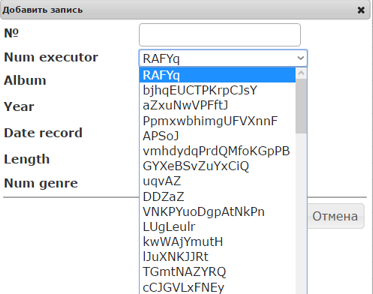


Рисунок 5.2 – Выпадающий список

Для отображение табличных данных был использован плагин *JQuery.Grid*. Благодаря которому, пользователю предоставляется удобный интерфейс. А также новые данные подгружаются с помощью технологии *AJAX*. На рисунке 5.3 изображен пример табличных данных.

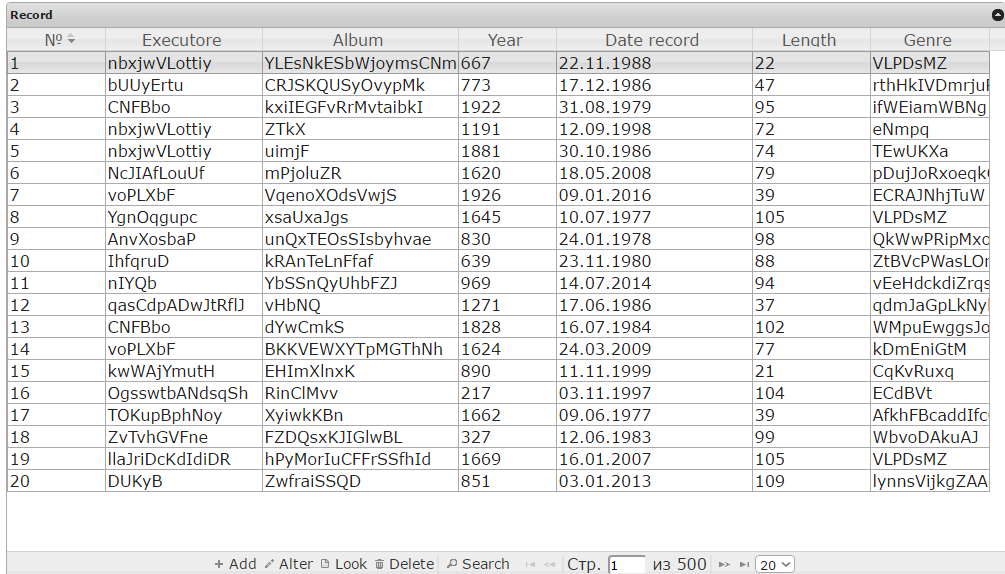


Рисунок 5.3 – Табличные данные

**ЗАКЛЮЧЕНИЕ**

Курсовой проект - этап обучения студента, где проявляются навыки выполнения самостоятельной научно-исследовательской работы и овладения методикой исследования и эксперимента при решении актуальной задачи в области избранной студентом специальности.

В данном курсовом проекте достигнуты все поставленные задачи и цели, а именно было разработано программное обеспечение с продуманным и простым интерфейсом.

Итогом курсового проекта стало *Web*-приложение базы данных, осуществляющее переход из одной страницы на другую и выполнения операций с данными, а именно: добавление, удаление, просмотр, редактирование. Сначала был проведён анализ предметной области, затем был разработан алгоритм работы приложения. Приложение имеет простой и понятный интерфейс, который позволяет пользователям без труда работать с данной ИС, поскольку при разработке были использованы общепринятые стандарты.

В ходе выполнения работы была изучена технология *ASP.NET MVC*, усовершенствованы навыки *Web*–разработки, работы с *ORM Entity Framework*.

Учитывая экономическую эффективность программного продукта, современность и перспективность выбранных технологий и подходов при разработки программного продукта можно с уверенностью сказать, что данный продукт будет, не только экономически обоснован и перспективен, но и прост и удобен в поддержке.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для высших учебных заведений / Под ред. проф. А.Д. Хомоненко. – 5-е изд., доп. – М.: Бином-Пресс; СПб.: КОРОНА принт, 2006. -736 с.
2. Эспозито Д. Знакомство с Microsoft ASP.NET 2.0 - Москва; Санкт-Петербург: Русская редакция: Питер, 2006. - 490 с.
3. Википедия, свободная энциклопедия / Д.Уэльс и Л.Сэнгер.//Основы ASP.NET [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org – Дата доступа: 18.05.2013.
4. Гаряка А.А. Основы ASP.NET 2.0/ [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/department/se/aspdotnet – Дата доступа: 22.05.2013.
5. Магдануров Г. Разработка веб-приложений на ASP.NET/ [Электронный ресурс]. – Режим доступа:

http://www.intuit.ru/department/internet/webdevaspnet – Дата доступа: 20.05.2013.

1. Горев, А. Эффективная работа с СУБД / А.Горев, Р.Ахаян, С.Макашарипов. – СПб.: Питер, 1997. – 74 с.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Распечатка программных модулей**

Контроллер таблицы “Record” (RecordController.cs).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using WebApplication1.Models;

using Newtonsoft.Json;

using System.Data.Entity;

namespace WebApplication1.Controllers

{

public class RecordController : Controller

{

RadioStationContext db = new RadioStationContext();

public ActionResult Index()

{

return View();

}

public JsonResult RecordToListGenre()

{

var all = db.Genres.Select(o => new { o.Number\_genre, o.Name\_genre }).ToList();

return Json(all, JsonRequestBehavior.AllowGet);

}

public JsonResult RecordToListExecutor()

{

var all = db.Executors.Select(o => new { o.Number\_executor, o.Executore }).ToList();

return Json(all, JsonRequestBehavior.AllowGet);

}

public string GetData()

{

var record = db.Records.Include(c => c.Genre).Include(p => p.Executor).Select(

t => new { t.Number\_composition, t.Number\_executor,t.Executor.Executore, t.Album, t.Year, t.Date\_record, t.Length, t.Number\_genre,t.Genre.Name\_genre });

return JsonConvert.SerializeObject(record);

}

[HttpPost]

public ActionResult Edit(Record record)

{

if (ModelState.IsValid)

{

db.Entry(record).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(record);

}

[HttpPost]

public ActionResult Create(Record record)

{

record.Number\_composition = 0;

if (ModelState.IsValid)

{

db.Records.Add(record);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(record);

}

[HttpPost]

public ActionResult Delete(int id)

{

Record record = db.Records.Find(id);

db.Records.Remove(record);

db.SaveChanges();

return RedirectToAction("Index");

}

}

}

Представление таблицы “Record” (Index.cshtml).

@{

Layout = null;

}

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home Page</title>

<link href="/Content/bootstrap.css" rel="stylesheet" />

<link href="/Content/site.css" rel="stylesheet" />

<script src="/Scripts/modernizr-2.6.2.js"></script>

<title>Record</title>

<script src="~/Scripts/jquery-3.1.1.min.js" type="text/javascript"></script>

<link href="~/Content/themes/base/jquery.ui.all.css" rel="stylesheet" type="text/css" />

<link href="~/Content/jquery.jqGrid/ui.jqgrid.css" rel="stylesheet" type="text/css" />

<script src="~/Scripts/jquery.jqGrid.min.js" type="text/javascript"></script>

<script src="~/Scripts/i18n/grid.locale-ru.js" type="text/javascript"></script>

<style>

.ui-jqgrid .ui-pg-input {

height: 20px;

}

.ui-jqgrid .ui-jqgrid-pager {

height: 40px;

}

.ui-jqgrid .ui-jqgrid-pager .ui-pg-div {

font-size: 14px;

}

</style>

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="/">RadioStation</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a href="http://localhost:10778/Schedulee/Index">Schedulee</a></li>

<li><a href="http://localhost:10778/Record/Index">Record</a></li>

<li><a href="http://localhost:10778/Genre/Index">Genre</a></li>

<li><a href="http://localhost:10778/Executor/Index">Executor</a></li>

<li><a href="http://localhost:10778/Employee/Index">Employee</a></li>

</ul>

</div>

</div>

</div>

<div class="block5" align="center">

<table id="jqg"></table>

<div id="jpager"></div>

<script type="text/javascript">

$(document).ready(function () {

$("#jqg").jqGrid({

url: '@Url.Action("GetData")',

datatype: "json",

colNames: ['№', 'Num executor', 'Executore', 'Album', 'Year', 'Date record', 'Length', 'Num genre', 'Genre'],

colModel: [

{ name: 'Number\_composition', index: 'Number\_composition', width: 50, stype: 'text', sorttype: "number", editable: true },

{

name: 'Number\_executor', index: 'Number\_executor', width: 80, sortable: true, sorttype: "number", editable: true, editrules: { required: true, edithidden: true }, hidden: true, edittype: 'select', stype: 'select',

editoptions: {

dataUrl: '@Url.Action("RecordToListExecutor", "Record")',

buildSelect: createSelectList

},

},

{ name: 'Executore', index: 'Executore', width: 80, sortable: true, sorttype: "number", editable: false },

{ name: 'Album', index: 'Album', width: 100, sortable: true, editable: true },

{ name: 'Year', index: 'Year', width: 50, stype: 'text', sorttype: "number", editable: true },

{ name: 'Date\_record', index: 'Date\_record', width: 100, formatter: "date", editable: true },

{ name: 'Length', index: 'Length', width: 60, stype: 'text', sorttype: "number", editable: true },

{

name: 'Number\_genre', index: 'Number\_genre', width: 60, stype: 'text', sorttype: "number", editable: true, editrules: { required: true, edithidden: true }, hidden: true, edittype: 'select', stype: 'select', editoptions: {

dataUrl: '@Url.Action("RecordToListGenre", "Record")',

buildSelect: createSelectList1

},

},

{ name: 'Name\_genre', index: 'Genre', width: 60, stype: 'text', sorttype: "number", editable: false },

],

rowNum: 20,

rowList: [10, 20, 30],

width: 1000,

height: 500,

pager: '#jpager',

loadonce: true,

sortname: 'Number\_composition',

sortorder: "desc",

caption: "Record"

});

$("#jqg").jqGrid('navGrid', '#jpager', {

search: true,

searchtext: "Search",

refresh: false,

add: true, // добавление

del: true, // удаление

edit: true, // редактирование

view: true, // просмотр записи

viewtext: "Look",

viewtitle: "Selected entry",

addtext: "Add",

edittext: "Alter",

deltext: "Delete"

},

update("edit"), // обновление

update("add"), // добавление

update("del") // удаление

);

function update(act) {

return {

closeAfterAdd: true, // закрыть после добавления

height: 400,

width: 500,

closeAfterEdit: true, // закрыть после редактирования

reloadAfterSubmit: true, // обновление

drag: true,

onclickSubmit: function (params) {

var list = $("#jqg");

var selectedRow = list.getGridParam("selrow");

rowData = list.getRowData(selectedRow);

if (act === "add")

params.url = '@Url.Action("Create")';

else if (act === "del")

params.url = '@Url.Action("Delete")';

else if (act === "edit")

params.url = '@Url.Action("Edit")';

},

afterSubmit: function (response, postdata) {

// обновление грида

$(this).jqGrid('setGridParam', { datatype: 'json' }).trigger('reloadGrid')

return [true, "", 0]

}

};

};

});

createSelectList = function (data) {

var response, s = '<select>', i, l, ri, k1, k2;

if (typeof (data) === "string") {

//var leng = data.length - 1;

response = jQuery.parseJSON(data);

}

else {

response = jQuery.parseJSON(data.responseText);

s += '<option value="">Select...</option>';

}

if (response && response.length) {

for (i = 0, l = response.length; i < l; i += 1) {

k1 = response[i];

s += '<option value="' + k1.Number\_executor + '">' + k1.Executore + '</option>';

}

}

return s + '</select>';

}

createSelectList1 = function (data) {

var response, s = '<select>', i, l, ri, k1, k2;

if (typeof (data) === "string") {

//var leng = data.length - 1;

response = jQuery.parseJSON(data);

}

else {

response = jQuery.parseJSON(data.responseText);

s += '<option value="">Select...</option>';

}

if (response && response.length) {

for (i = 0, l = response.length; i < l; i += 1) {

k1 = response[i];

s += '<option value="' + k1.Number\_genre + '">' + k1.Name\_genre + '</option>';

}

}

return s + '</select>';

}

</script>

</div>

<hr />

<footer>

<p>&copy; 2016 – application ASP.NET</p>

</footer>

</body>

</html>

Контроллер таблицы “Genre” (GenreController.cs).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using WebApplication1.Models;

using Newtonsoft.Json;

using System.Data.Entity;

namespace WebApplication1.Controllers

{

public class GenreController : Controller

{

RadioStationContext db = new RadioStationContext();

public ActionResult Index()

{

return View();

}

public string GetData()

{

var genre = db.Genres.Select(t => new { t.Number\_genre, t.Name\_genre, t.Description});

return JsonConvert.SerializeObject(genre);

}

[HttpPost]

public ActionResult Edit(Genre genre)

{

if (ModelState.IsValid)

{

db.Entry(genre).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(genre);

}

[HttpPost]

public ActionResult Create(Genre genre)

{

genre.Number\_genre = 0;

if (ModelState.IsValid)

{

db.Genres.Add(genre);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(genre);

}

[HttpPost]

public ActionResult Delete(int id)

{

Genre genre = db.Genres.Find(id);

db.Genres.Remove(genre);

db.SaveChanges();

return RedirectToAction("Index");

}

}

}

Представление таблицы “Genre” (Index.cshtml).

@{

Layout = null;

}

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home Page</title>

<link href="/Content/bootstrap.css" rel="stylesheet" />

<link href="/Content/site.css" rel="stylesheet" />

<script src="/Scripts/modernizr-2.6.2.js"></script>

<title>Genre</title>

<script src="~/Scripts/jquery-3.1.1.min.js" type="text/javascript"></script>

<link href="~/Content/themes/base/jquery.ui.all.css" rel="stylesheet" type="text/css" />

<link href="~/Content/jquery.jqGrid/ui.jqgrid.css" rel="stylesheet" type="text/css" />

<script src="~/Scripts/jquery.jqGrid.min.js" type="text/javascript"></script>

<script src="~/Scripts/i18n/grid.locale-ru.js" type="text/javascript"></script>

<style>

.ui-jqgrid .ui-pg-input {

height: 20px;

}

.ui-jqgrid .ui-jqgrid-pager {

height: 40px;

}

.ui-jqgrid .ui-jqgrid-pager .ui-pg-div {

font-size: 14px;

}

body{

background: url(3.jpeg) no-repeat;

background-size: 100%;

background-attachment: fixed;

}

}

</style>

</head>

<body >

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="/">RadioStation</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a href="http://localhost:10778/Schedulee/Index">Schedulee</a></li>

<li><a href="http://localhost:10778/Record/Index">Record</a></li>

<li><a href="http://localhost:10778/Genre/Index">Genre</a></li>

<li><a href="http://localhost:10778/Executor/Index">Executor</a></li>

<li><a href="http://localhost:10778/Employee/Index">Employee</a></li>

</ul>

</div>

</div>

</div>

<div class="block5" align="center">

<table id="jqg"></table>

<div id="jpager"></div>

<script type="text/javascript">

$(document).ready(function () {

$("#jqg").jqGrid({

url: '@Url.Action("GetData")',

datatype: "json",

colNames: ['№', 'Description', 'Name\_genre'],

colModel: [

{ name: 'Number\_genre', index: 'Number\_genre', width: 30, stype: 'text', sorttype: "number", editable: true },

{ name: 'Description', index: 'Description', width: 150, sortable: true, editable: true },

{ name: 'Name\_genre', index: 'Name\_genre', width: 150, sortable: true, editable: true },

],

rowNum: 10,

rowList: [10, 20, 30],

width: 1000,

height: 500,

pager: '#jpager',

loadonce: true,

sortname: 'Number\_genre',

sortorder: "desc",

caption: "Genre"

});

$("#jqg").jqGrid('navGrid', '#jpager', {

search: true,

searchtext: "Search",

refresh: false,

add: true, // добавление

del: true, // удаление

edit: true, // редактирование

view: true, // просмотр записи

viewtext: "Look",

viewtitle: "Selected entry",

addtext: "Add",

edittext: "Alter",

deltext: "Delete"

},

update("edit"), // обновление

update("add"), // добавление

update("del") // удаление

);

function update(act) {

return {

closeAfterAdd: true, // закрыть после добавления

height: 300,

width: 500,

closeAfterEdit: true, // закрыть после редактирования

reloadAfterSubmit: true, // обновление

drag: true,

onclickSubmit: function (params) {

var list = $("#jqg");

var selectedRow = list.getGridParam("selrow");

rowData = list.getRowData(selectedRow);

if (act === "add")

params.url = '@Url.Action("Create")';

else if (act === "del")

params.url = '@Url.Action("Delete")';

else if (act === "edit")

params.url = '@Url.Action("Edit")';

},

afterSubmit: function (response, postdata) {

// обновление грида

$(this).jqGrid('setGridParam', { datatype: 'json' }).trigger('reloadGrid')

return [true, "", 0]

}

};

};

});

</script>

</div>

</body>

</html>

Контролер таблицы “Executor” (ExecutorControler.cs).

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web.Mvc;

using WebApplication1.Models;

using Newtonsoft.Json;

using System.Collections.Generic;

namespace WebApplication1.Controllers

{

public class ExecutorController : Controller

{

RadioStationContext db = new RadioStationContext();

public ActionResult Index()

{

return View();

}

public string GetData()

{

var executor = db.Executors.Select(

t => new { t.Number\_executor, t.Executore, t.Compos });

return JsonConvert.SerializeObject(executor);

}

[HttpPost]

public ActionResult Edit(Executor executor)

{

if (ModelState.IsValid)

{

db.Entry(executor).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(executor);

}

[HttpPost]

public ActionResult Create(Executor executor)

{

executor.Number\_executor = 0;

if (ModelState.IsValid)

{

db.Executors.Add(executor);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(executor);

}

[HttpPost]

public ActionResult Delete(int id)

{

Executor executor = db.Executors.Find(id);

db.Executors.Remove(executor);

db.SaveChanges();

return RedirectToAction("Index");

}

}

}

Представление таблицы “Executor” (Index.cshtml).

@{

Layout = null;

}

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home Page</title>

<link href="/Content/bootstrap.css" rel="stylesheet" />

<link href="/Content/site.css" rel="stylesheet" />

<script src="/Scripts/modernizr-2.6.2.js"></script>

<title>Executor</title>

<script src="~/Scripts/jquery-3.1.1.min.js" type="text/javascript"></script>

<link href="~/Content/themes/base/jquery.ui.all.css" rel="stylesheet" type="text/css" />

<link href="~/Content/jquery.jqGrid/ui.jqgrid.css" rel="stylesheet" type="text/css" />

<script src="~/Scripts/jquery.jqGrid.min.js" type="text/javascript"></script>

<script src="~/Scripts/i18n/grid.locale-ru.js" type="text/javascript"></script>

<style>

.ui-jqgrid .ui-pg-input {

height: 20px;

}

.ui-jqgrid .ui-jqgrid-pager {

height: 40px;

}

.ui-jqgrid .ui-jqgrid-pager .ui-pg-div {

font-size: 14px;

}

</style>

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="/">RadioStation</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a href="http://localhost:10778/Schedulee/Index">Schedulee</a></li>

<li><a href="http://localhost:10778/Record/Index">Record</a></li>

<li><a href="http://localhost:10778/Genre/Index">Genre</a></li>

<li><a href="http://localhost:10778/Executor/Index">Executor</a></li>

<li><a href="http://localhost:10778/Employee/Index">Employee</a></li>

</ul>

</div>

</div>

</div>

<div class="block5" align="center">

<table id="jqg"></table>

<div id="jpager"></div>

<script type="text/javascript">

$(document).ready(function () {

$("#jqg").jqGrid({

url: '@Url.Action("GetData")',

datatype: "json",

colNames: ['№', 'Executore', 'Name\_genre'],

colModel: [

{ name: 'Number\_executor', index: 'Number\_executor', width: 30, stype: 'text', sorttype: "number", editable: true },

{ name: 'Executore', index: 'Executore', width: 150, sortable: true, editable: true },

{ name: 'Compos', index: 'Compos', width: 150, sortable: true, editable: true },

],

rowNum: 10,

rowList: [10, 20, 30],

width: 1000,

height: 500,

pager: '#jpager',

loadonce: true,

sortname: 'Number\_executor',

sortorder: "desc",

caption: "Executor"

});

$("#jqg").jqGrid('navGrid', '#jpager', {

search: true,

searchtext: "Search",

refresh: false,

add: true, // добавление

del: true, // удаление

edit: true, // редактирование

view: true, // просмотр записи

viewtext: "Look",

viewtitle: "Selected entry",

addtext: "Add",

edittext: "Alter",

deltext: "Delete"

},

update("edit"), // обновление

update("add"), // добавление

update("del") // удаление

);

function update(act) {

return {

closeAfterAdd: true, // закрыть после добавления

height: 300,

width: 500,

closeAfterEdit: true, // закрыть после редактирования

reloadAfterSubmit: true, // обновление

drag: true,

onclickSubmit: function (params) {

var list = $("#jqg");

var selectedRow = list.getGridParam("selrow");

rowData = list.getRowData(selectedRow);

if (act === "add")

params.url = '@Url.Action("Create")';

else if (act === "del")

params.url = '@Url.Action("Delete")';

else if (act === "edit")

params.url = '@Url.Action("Edit")';

},

afterSubmit: function (response, postdata) {

// обновление грида

$(this).jqGrid('setGridParam', { datatype: 'json' }).trigger('reloadGrid')

return [true, "", 0]

}

};

};

});

</script>

</div>

<hr />

<footer>

<p>&copy; 2016 – application ASP.NET</p>

</footer>

</body>

</html>

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Экранные копии Web страниц**

Отображение страницы «*Record*» показан на рисунке Б.1. На странице отображена таблица с данными: наименование композиции, исполнитель, альбом, год, жанр, дата записи, длительность, рейтинг. Возможно редактирование, добавление, удаление данных.

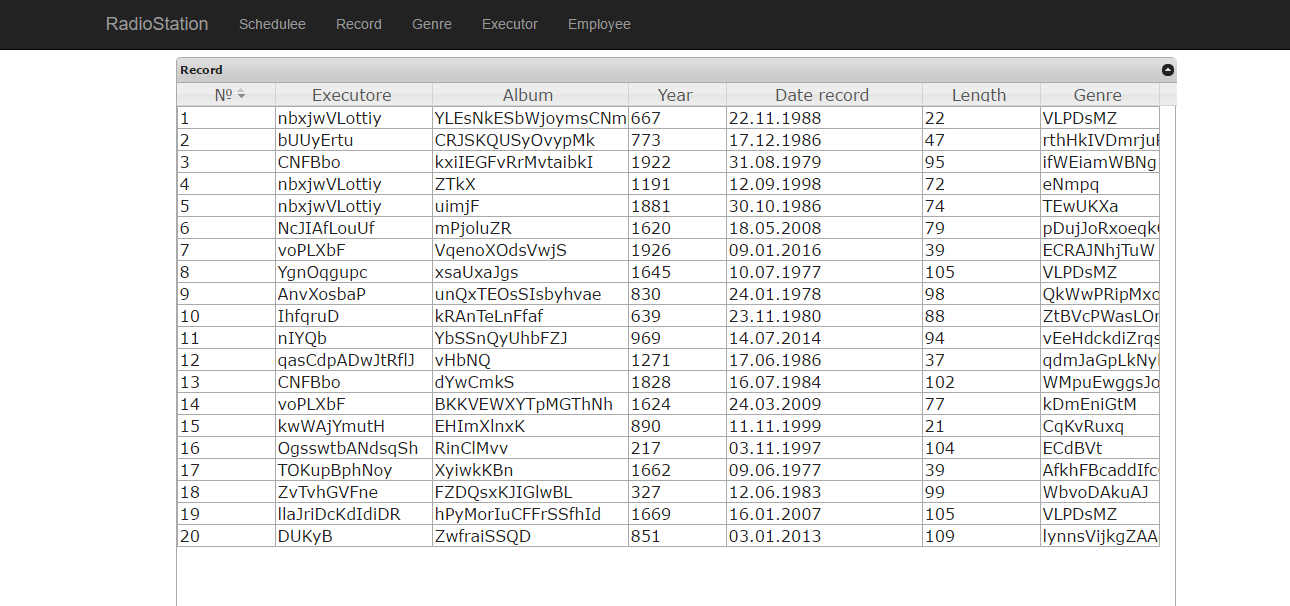


Рисунок Б.1 – Вид страницы «*Record*»

Отображение страницы «*Genre*» показан на рисунке Б.2. На странице отображена таблица с данными: наименование, описание. Имеется возможность редактирования, добавления, удаления данных.

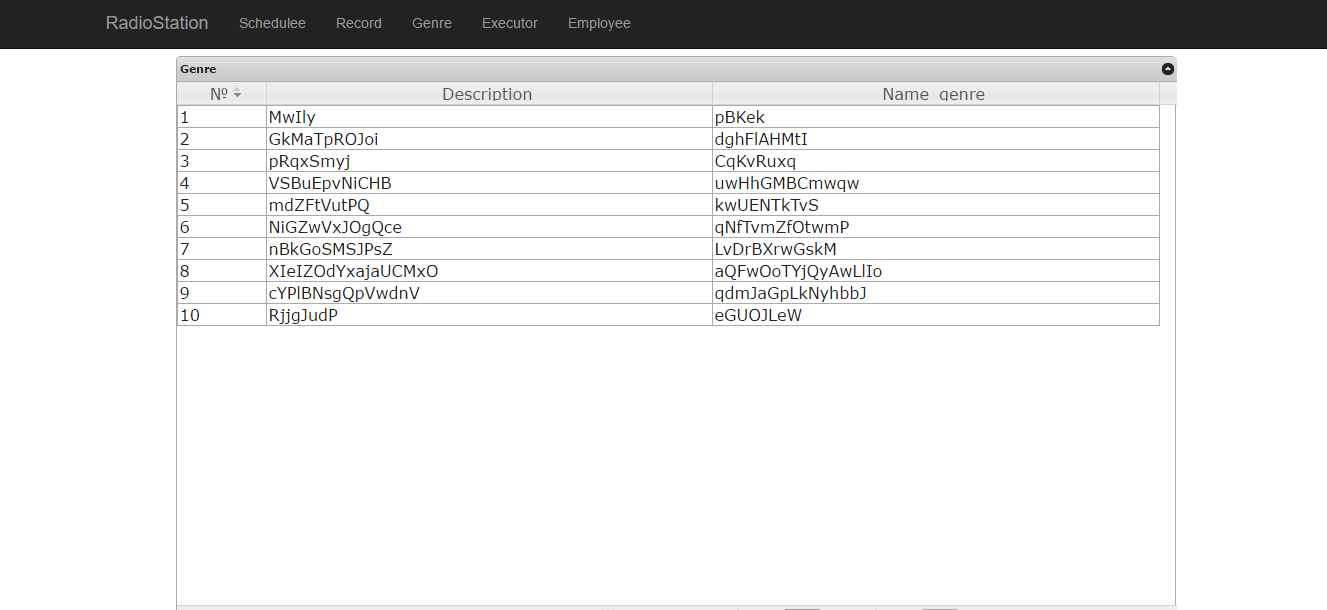


Рисунок Б.2 – Вид страницы «*Genre*»

Отображение страницы «*Executor*» показана на рисунке Б.3. На странице представлена таблица с данными: наименование исполнителя, наименование жанра. Возможно редактирование, добавление, удаление данных.

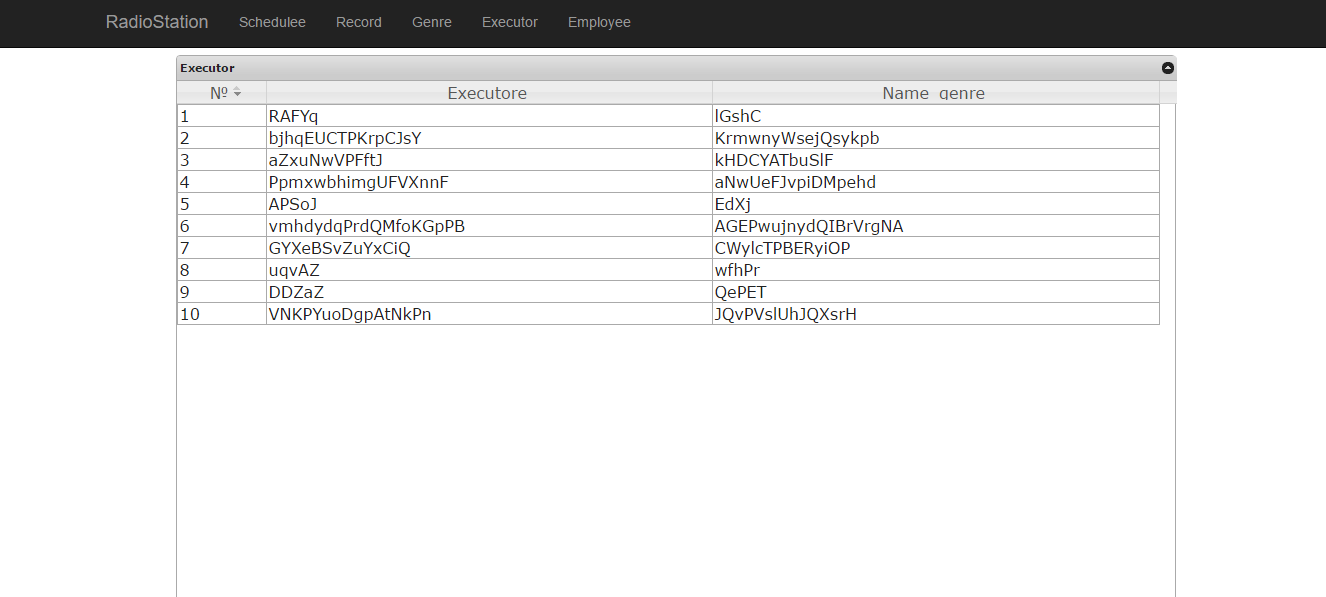


Рисунок Б.3 – Вид страницы «*Executor*»

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Чертеж схемы базы данных и структуры *Web* приложения**