Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Программная инженерия (ИУ7)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

         Web-приложение и база данных гипотетической авиакомпании

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент            группы ИУ7-61 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   Баранин А.В.

(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

Москва, 2014

Реферат

В курсовом проекте выполнена разработка базы данных гипотетической авиакомпании «SkyShark» и Web-приложения для работы с ней на языке C#.

Проведено:

1. Анализ технического задания и составление первичного представления о сущностях БД.

2. Составление схемы базы данных.

3. Определение ограничений, накладываемых на домены.

4. Разработка Web-приложения и реализация в нём бизнес-логики.

Расчётно-пояснительная записка содержит 28 листов формата А4, 1 таблицу,   
3 изображения.

Оглавление

[Реферат 3](#_Toc398201481)

[Оглавление 4](#_Toc398201482)

[Введение 5](#_Toc398201483)

[1. Аналитическая часть 6](#_Toc398201484)

[1.1 Описание предметной области 6](#_Toc398201485)

[1.2 Определение требований к структуре базы данных 6](#_Toc398201486)

[1.3 Определение типов данных 7](#_Toc398201487)

[1.4 Определение бизнес-правил 7](#_Toc398201488)

[1.5 Разработка модели данных 8](#_Toc398201489)

[1.6 Диаграмма базы данных 9](#_Toc398201490)

[2. Конструкторская часть 10](#_Toc398201491)

[2.1 Инициализация базы данных SkySharkDb 10](#_Toc398201492)

[2.2 Подготовка объектно-реляционной модели 13](#_Toc398201493)

[2.3 Разработка слоя бизнес-логики. 15](#_Toc398201494)

[2.4 Анализ методологии 18](#_Toc398201495)

[3. Технологическая часть 20](#_Toc398201496)

[3.1 Платформа ASP.NET 20](#_Toc398201497)

[3.2 Разметка страниц 20](#_Toc398201498)

[3.3 Обработка исключений 22](#_Toc398201499)

[3.4 Представление и редактирование объектов. 23](#_Toc398201500)

[Заключение 27](#_Toc398201501)

[Список использованной литературы 28](#_Toc398201502)

Введение

Базы данных всегда были важнейшей темой при изучении информационных систем. Они являются главным помощником в организации коммерческой деятельности человека, а также незаменимы в областях деятельности, требующих учёта большого количества событий, как в случае с таможенной службой любого государства. Не одно десятилетие было потрачено на совершенствование современных решений. Целью этой работы является ознакомление с современными решениями для разработки приложений, использующих базы данных, от корпорации Microsoft.

За предметную область взята предложенная в [1] гипотетическая авиакомпания «SkyShark», являющаяся неплохим примером простой, но показательной в некоторой необычности бизнес-логики пользовательской базы данных. В процессе разработки приложения были изучены современные средства по работе с SQL-базами данных, включённые в состав платформы   
Microsoft .NET, а так же средства разработки сайтов и их взаимодействия с СУБД.

Приложение для работы требует установленную операционную систему семейства Windows NT и Microsoft .NET Framework 4.5. Приложение работает в тандеме с Microsoft SQL Server, хотя с незначительными изменениями и может быть переделано под Oracle и OLE DB. Web-контент доступен с любого браузера, поддерживающего XHTML.

1. Аналитическая часть

1.1 Описание предметной области

Молодая и развивающаяся лизинговая авиакомпания «SkyShark» нуждается в средстве автоматического учёта пассажиров, а так же их обслуживания. Уже давно распространённая методика онлайн-регистрации позволит увеличить выручку и снизить нагрузку на персонал. Более того, от внедрения такой системы выиграл бы финансовый отдел, которому необходимы средства детального статистического анализа. Представители авиакомпании сформулировали приведённое выше техническое задание, подробно описывающее функции, доступные различным работникам и простым клиентам.

1.2 Определение требований к структуре базы данных

Составление схемы базы данных требует более детального понимания бизнес-модели «SkyShark». Компания предоставляет услуги авиаперевозки физическим лицам, предъявляющим паспорт при оплате билета и посадке. Самолёт вылетает строго в назначенное время, и, для упрощения модели, его вылету ничто не может воспрепятствовать. Самолёт имеет модель, однако, большинство современных лайнеров очень гибки в конфигурации салона. Это означает, что количество свободных мест эконом и бизнес-класса задаётся при планировании вылета менеджером. Самолёт в сочетании с тем фактом, что он вылетит в известное время из известного города и приземлится в известный час в другом городе, формируют первую сущность – Рейс.

На рейс садятся пассажиры. Это физические лица, имеющие паспорт, имя, а так же историю полётов рейсами авиакомпании «SkyShark». Последний факт необходим менеджерам для осуществления программ скидок постоянным клиентам. Пассажир – вторая сущность нашей базы данных.

Стандартная форма покупки билетов на рейс – бронирование. Пассажир связывается с сотрудниками авиакомпании и просит закрепить за ним билет на рейс вплоть до его оплаты, или же до сброса брони сотрудниками авиакомпании. В процессе бронирования клиент указывает свой номер паспорта, имя (если до этого он не летал самолётами «SkyShark»), тип билета (эконом или бизнес-класс) и контактный e-mail. Третья сущность – Бронь.

Процесс оплаты стоимости билета можно также воспринимать как подтверждение брони, даже если человек сразу покупает билет на кассе, без предварительной регистрации. Иногда случается так, что пассажир, оплативший билет, не может в силу сложившихся обстоятельств, им воспользоваться. В таких ситуациях уважающие себя компании предоставляют услугу возврата билета, компенсирующая пассажиру часть его стоимости (в зависимости от времени). Для учёта случаев возврата имеет смысл ввести новую сущность – Возврат, связанную с бронью и отражающую факт отказа пассажира от уже оплаченной услуги.

Наконец, требуется обеспечить администрирование логики приложения с помощью системы учётных записей – аккаунтов. Эта система отвечает за распределение ролей между сотрудниками компании, а так же предоставляет информацию о рейсах и права на регистрацию брони. Аккаунт – это пятая сущность.

Нагрузки на сервер относительно небольшие. В день редко совершается более пяти рейсов. Подтверждение брони одного пассажира занимает около минуты, что вызвано рутиной проверки документов и денежными операциями. В среднем один самолёт берёт на борт около ста пятидесяти пассажиров. Из вышесказанного очевидно, что база данных не будет подвержена экстремальным нагрузкам.

1.3 Определение типов данных

Сформулированы следующие пять сущностей: Рейс, Бронь, Пассажир, Отмена Брони, Аккаунт.

таблица 1 таблица атрибутов сущностей системы

|  |  |
| --- | --- |
| Сущность | Атрибуты |
| Рейс | Номер рейса, город вылета\прилёта, дата и время вылета\прилёта, модель самолёта, количество мест эконом\бизнес-класса, цена за билет эконом\бизнес класса |
| Бронь | Номер билета, номер рейса, номер паспорта, класс места, цена с учётом программы скидок, дата и время брони, статус, e-mail |
| Пассажир | Номер паспорта, имя, общий баланс по отношению к компании, количество перелётов, текущий процент скидки |
| Отмена Брони | Номер билета, обратная выплата, аккаунт сотрудника-операциониста, дата и время отмены, комментарий. |
| Аккаунт | Логин, пароль, права, изменён ли пароль (стандарт), e-mail |

1.4 Определение бизнес-правил

* Все вышеперечисленные атрибуты, кроме e-mail в Брони и комментария в Отмене Брони являются обязательными.
* Город вылета и прилёта не должны совпадать.
* Время вылета должно быть меньше времени прилёта.
* Общее число брони определённого класса на рейс не превышает число мест этого класса в самолёте.
* Все цены, кроме обратной выплаты, неотрицательны.
* Номер рейса – строка ASCII не длиннее 20-и символов.
* Имя пассажира – строка Unicode произвольной длины
* Остальные строки – ASCII строки не длиннее 50-и символов.
* Все денежные суммы хранятся в формате десятичной дроби с точностью в 19 разрядов и 4 знака после запятой (тип Decimal(19, 4), эквивалентный SQL Server money).

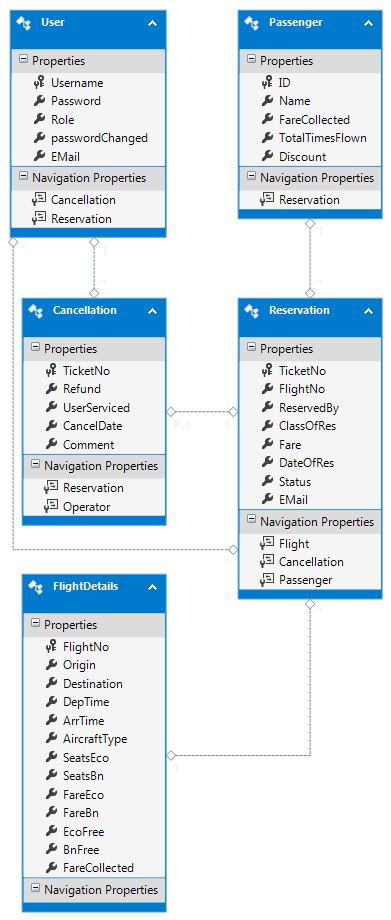
1.5 Разработка модели данных

Платформа .NET включает в себя большое количество различных способов взаимодействия с СУБД. Наиболее низкоуровневый, основанный на языке SQL-запросов механизм слишком стар для использования в 2014 году. Ему на смену пришла технология ADO.NET, предоставляющая обширный набор готовых к использованию классов и процессов разработки. Одним из ключевых компонентов ADO.NET на высоком уровне является компонент Entity Framework (далее EF), включающий в себя набор пространств имён классов-репозиториев, адаптеров, прокси, а так же три основных процесса разработки базы данных:

1. Model First – первична модель базы данных, по которой генерируется DDL-шаблон (Data Definition Language) и метаданные, SQL-инициализатор и классовая иерархия объектно-ориентированной модели этой базы.
2. Code First – первична классовая иерархия и мета-атрибуты, которые компилируются в модель базы данных и SQL-инициализатор через DDL.
3. Database First – первична готовая реляционная база данных, по схеме которой строится объектно-реляционная модель.

В данном конкретном случае я использовал первый процесс – начал с модели базы данных. Её создание происходит в богатом опциями графическом редакторе схем, предназначенном для быстрого создания сущностей, их атрибутов и связей между ними, а также наглядного из представления и лёгкого редактирования. Полученная схема представлена в разделе 1.6.

1.6 Диаграмма базы данных



2. Конструкторская часть

2.1 Инициализация базы данных SkySharkDb

Сгенерированный из шаблона DDL код позволяет создать необходимые для работы логики таблицы. Листинг секции создания таблиц:

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

-- Creating all tables

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

-- Creating table 'UserSet'

CREATE TABLE [dbo].[UserSet] (

[Username] varchar(50) NOT NULL,

[Password] varchar(50) NOT NULL,

[Role] tinyint NOT NULL,

[passwordChanged] bit NOT NULL,

[EMail] varchar(50) NOT NULL

);

GO

-- Creating table 'FlightDetailsSet'

CREATE TABLE [dbo].[FlightDetailsSet] (

[FlightNo] varchar(20) NOT NULL,

[Origin] nvarchar(max) NOT NULL,

[Destination] nvarchar(max) NOT NULL,

[DepTime] datetime NOT NULL,

[ArrTime] datetime NOT NULL,

[AircraftType] nvarchar(max) NOT NULL,

[SeatsEco] smallint NOT NULL,

[SeatsBn] smallint NOT NULL,

[FareEco] decimal(19,4) NOT NULL,

[FareBn] decimal(19,4) NOT NULL,

[EcoFree] smallint NOT NULL,

[BnFree] smallint NOT NULL,

[FareCollected] decimal(19,4) NOT NULL

);

GO

-- Creating table 'ReservationSet'

CREATE TABLE [dbo].[ReservationSet] (

[TicketNo] int IDENTITY(1,1) NOT NULL,

[FlightNo] varchar(20) NOT NULL,

[ReservedBy] varchar(50) NOT NULL,

[ClassOfRes] tinyint NOT NULL,

[Fare] decimal(19,4) NOT NULL,

[DateOfRes] datetime NOT NULL,

[Status] tinyint NOT NULL,

[EMail] varchar(50) NULL

);

GO

-- Creating table 'CancellationSet'

CREATE TABLE [dbo].[CancellationSet] (

[TicketNo] int NOT NULL,

[Refund] decimal(19,4) NOT NULL,

[UserServiced] varchar(50) NOT NULL,

[CancelDate] datetime NOT NULL,

[Comment] nvarchar(max) NULL

);

GO

-- Creating table 'PassengerSet'

CREATE TABLE [dbo].[PassengerSet] (

[ID] varchar(50) NOT NULL,

[Name] nvarchar(max) NOT NULL,

[FareCollected] decimal(19,4) NOT NULL,

[TotalTimesFlown] int NOT NULL,

[Discount] real NOT NULL

);

GO

-- Creating table 'UserReservation'

CREATE TABLE [dbo].[UserReservation] (

[UserReservation\_Reservation\_Username] varchar(50) NOT NULL,

[Reservation\_TicketNo] int NOT NULL

);

GO

Листинг сценариев генерации ограничений на первичные и внешние ключи:

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

-- Creating all PRIMARY KEY constraints

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

-- Creating primary key on [Username] in table 'UserSet'

ALTER TABLE [dbo].[UserSet]

ADD CONSTRAINT [PK\_UserSet]

PRIMARY KEY CLUSTERED ([Username] ASC);

GO

-- Creating primary key on [FlightNo] in table 'FlightDetailsSet'

ALTER TABLE [dbo].[FlightDetailsSet]

ADD CONSTRAINT [PK\_FlightDetailsSet]

PRIMARY KEY CLUSTERED ([FlightNo] ASC);

GO

-- Creating primary key on [TicketNo] in table 'ReservationSet'

ALTER TABLE [dbo].[ReservationSet]

ADD CONSTRAINT [PK\_ReservationSet]

PRIMARY KEY CLUSTERED ([TicketNo] ASC);

GO

-- Creating primary key on [TicketNo] in table 'CancellationSet'

ALTER TABLE [dbo].[CancellationSet]

ADD CONSTRAINT [PK\_CancellationSet]

PRIMARY KEY CLUSTERED ([TicketNo] ASC);

GO

-- Creating primary key on [ID] in table 'PassengerSet'

ALTER TABLE [dbo].[PassengerSet]

ADD CONSTRAINT [PK\_PassengerSet]

PRIMARY KEY CLUSTERED ([ID] ASC);

GO

-- Creating primary key on [UserReservation\_Reservation\_Username], [Reservation\_TicketNo] in table 'UserReservation'

ALTER TABLE [dbo].[UserReservation]

ADD CONSTRAINT [PK\_UserReservation]

PRIMARY KEY CLUSTERED ([UserReservation\_Reservation\_Username], [Reservation\_TicketNo] ASC);

GO

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

-- Creating all FOREIGN KEY constraints

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

-- Creating foreign key on [FlightNo] in table 'ReservationSet'

ALTER TABLE [dbo].[ReservationSet]

ADD CONSTRAINT [FK\_ReservationFlightDetails]

FOREIGN KEY ([FlightNo])

REFERENCES [dbo].[FlightDetailsSet]

([FlightNo])

ON DELETE NO ACTION ON UPDATE NO ACTION;

-- Creating non-clustered index for FOREIGN KEY 'FK\_ReservationFlightDetails'

CREATE INDEX [IX\_FK\_ReservationFlightDetails]

ON [dbo].[ReservationSet]

([FlightNo]);

GO

-- Creating foreign key on [TicketNo] in table 'CancellationSet'

ALTER TABLE [dbo].[CancellationSet]

ADD CONSTRAINT [FK\_ReservationCancellation]

FOREIGN KEY ([TicketNo])

REFERENCES [dbo].[ReservationSet]

([TicketNo])

ON DELETE NO ACTION ON UPDATE NO ACTION;

GO

-- Creating foreign key on [UserServiced] in table 'CancellationSet'

ALTER TABLE [dbo].[CancellationSet]

ADD CONSTRAINT [FK\_CancellationUser]

FOREIGN KEY ([UserServiced])

REFERENCES [dbo].[UserSet]

([Username])

ON DELETE NO ACTION ON UPDATE NO ACTION;

-- Creating non-clustered index for FOREIGN KEY 'FK\_CancellationUser'

CREATE INDEX [IX\_FK\_CancellationUser]

ON [dbo].[CancellationSet]

([UserServiced]);

GO

-- Creating foreign key on [ReservedBy] in table 'ReservationSet'

ALTER TABLE [dbo].[ReservationSet]

ADD CONSTRAINT [FK\_PassengerReservation]

FOREIGN KEY ([ReservedBy])

REFERENCES [dbo].[PassengerSet]

([ID])

ON DELETE NO ACTION ON UPDATE NO ACTION;

-- Creating non-clustered index for FOREIGN KEY 'FK\_PassengerReservation'

CREATE INDEX [IX\_FK\_PassengerReservation]

ON [dbo].[ReservationSet]

([ReservedBy]);

GO

-- Creating foreign key on [UserReservation\_Reservation\_Username] in table 'UserReservation'

ALTER TABLE [dbo].[UserReservation]

ADD CONSTRAINT [FK\_UserReservation\_User]

FOREIGN KEY ([UserReservation\_Reservation\_Username])

REFERENCES [dbo].[UserSet]

([Username])

ON DELETE NO ACTION ON UPDATE NO ACTION;

GO

-- Creating foreign key on [Reservation\_TicketNo] in table 'UserReservation'

ALTER TABLE [dbo].[UserReservation]

ADD CONSTRAINT [FK\_UserReservation\_Reservation]

FOREIGN KEY ([Reservation\_TicketNo])

REFERENCES [dbo].[ReservationSet]

([TicketNo])

ON DELETE NO ACTION ON UPDATE NO ACTION;

-- Creating non-clustered index for FOREIGN KEY 'FK\_UserReservation\_Reservation'

CREATE INDEX [IX\_FK\_UserReservation\_Reservation]

ON [dbo].[UserReservation]

([Reservation\_TicketNo]);

GO

2.2 Подготовка объектно-реляционной модели

В процессе разработки слоя бизнес-логики был найден довольно существенный недостаток процесса Model First – заданные в дизайнере домены не транслируются ни в SQL CHECK ограничения, ни в метамодель классов, что удивительно, так как это значительно увеличивает время подготовки DAL (Data Access Layer). Генерируемые ил DDL классы не содержат мета-таблиц и не пригодны для валидации модели на уровне бизнес-логики. А раз этой же валидации нет и на уровне SQL-сервера, для обеспечения корректности работы приложения были вручную созданы мета-таблицы для сгенерированных классов.

На листинге ниже код класс пассажира и его мета-класс:

public partial class Passenger

{

public Passenger()

{

this.FareCollected = 0.0m;

this.TotalTimesFlown = 0;

this.Discount = 0F;

this.Reservation = new HashSet<Reservation>();

}

public string ID { get; set; }

public string Name { get; set; }

public decimal FareCollected { get; set; }

public int TotalTimesFlown { get; set; }

public float Discount { get; set; }

public virtual ICollection<Reservation> Reservation { get; set; }

}

[MetadataType(typeof(PassengerMetadata))]

public partial class Passenger

{

public class PassengerMetadata

{

[Key]

[Required(AllowEmptyStrings = false)]

[InverseProperty("Passenger")]

[MaxLength(50, ErrorMessage = "Максимальная длина номера паспорта - 50")]

[Display(Name="Номер паспорта")]

public string ID { get; set; }

[Required(AllowEmptyStrings = false)]

[Display(Name = "ФИО")]

public string Name { get; set; }

[Required]

[Range(typeof(decimal), "0", "999999999999999")]

[DisplayFormat(DataFormatString = "0.00")]

[Display(Name = "Всего оплатил")]

public decimal FareCollected { get; set; }

[Required]

[Range(0, int.MaxValue)]

[Display(Name = "Кол-во перелётов")]

public int TotalTimesFlown { get; set; }

[Required]

[Range(0.0, 1.0)]

[Display(Name = "Скидка")]

public float Discount { get; set; }

[InverseProperty("Passenger")]

public virtual ICollection<Reservation> Reservation { get; set; }

}

}

Стоит уделить отдельное внимание на так-называемые Navigation Properties. В приведённом выше классе Passenger присутствует свойство Reservation, которое отвечает за отношение один-ко-многим между пассажиром и бронью. EF генерирует в объектной модели свойства, отвечающие за отношения и позволяющие непосредственно переходить по связям. Аттрибут [InverseProperty("Passenger")] явно указывает на «обратный» внешний ключ в классе Reservation.

В случае же отношения многие-ко-многим, как в отношении Аккаунт-Бронь, EF сгенерировал дополнительную таблицу:

CREATE TABLE [dbo].[UserReservation] (

[UserReservation\_Reservation\_Username] varchar(50) NOT NULL,

[Reservation\_TicketNo] int NOT NULL

);

В ней будут храниться пары Пользователь – его Бронь с целью оптимизации поиска. При этом у этой таблицы нет отображения в объектно-реляционной модели: о её существовании мы узнаём лишь из свойств класса User, возвращающих коллекцию объектов класса Reservation.

2.3 Разработка слоя бизнес-логики.

BLL – Business Logic Layer. Согласно принципам абстракции логики от представления данных, которые проповедует ADO.NET, правила прикладного приложения не должны содержать специфичного для SQL-запросов кода. В подтверждение продуктивности такого подхода, себя очень хорошо зарекомендовал второй компонент ADO.NET – Linq. Универсальный в рамках платформы .NET строго типизированный и динамически компилируемый язык запросов к коллекциям (Linq to Entities) и соединениям с СУБД (Linq to SQL, не рекомендуемый уже самими Microsoft как нарушающий абстракцию) позволяет быстро и надёжно формулировать привычные SQL-запросы к разнородным объектам. На сегодняшний день Linq to Entities - основной рекомендованный способ декларативного доступа к любым коллекциям в среде .NET.

В компоненте EF6 DAL представлен реализацией паттерна репозитория, классом System.Data.Entity. DbContext, имеющим следующие атрибуты:

public partial class SkySharkDbContainer : DbContext

{

public SkySharkDbContainer()

: base("name=SkySharkDbContainer")

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

throw new UnintentionalCodeFirstException();

}

public virtual DbSet<User> UserSet { get; set; }

public virtual DbSet<FlightDetails> FlightDetailsSet { get; set; }

public virtual DbSet<Reservation> ReservationSet { get; set; }

public virtual DbSet<Cancellation> CancellationSet { get; set; }

public virtual DbSet<Passenger> PassengerSet { get; set; }

}

SkySharkDbContainer – его наследник, реализующий схему нашей базы данных. Его атрибуты – коллекции сущностей нашей модели, к которым следует применять запросы. Несколько не приведённых здесь методов отвечают за синхронизацию состояния контекста БД и СУБД – совершение транзакций.

Ниже приведён листинг реализации бизнес-логики класса FlightDetails (Рейс):

public partial class FlightDetails

{

// Вызвать после конструктора и инициализировать вспомогательные поля

public void Initialize()

{

EcoFree = SeatsEco;

BnFree = SeatsBn;

FareCollected = 0.0m;

}

public static IQueryable<FlightDetails> getNotDepartedFlights(SkySharkDbContainer context)

{

DateTime now = DateTime.Now;

return (from flight in context.FlightDetailsSet

where flight.DepTime > now

select flight);

}

public static IQueryable<FlightDetails> getAllFlights(SkySharkDbContainer context)

{

return context.FlightDetailsSet;

}

}

Бизнес-логика класса Бронь (Reservation):

public partial class Reservation

{

*// Составление брони*

public void apply\_reservation(SkySharkDbContainer context, string flightNo, string reservedBy)

{

if (flightNo != null)

FlightNo = flightNo;

if (reservedBy != null)

ReservedBy = reservedBy;

Passenger = context.PassengerSet.Find(ReservedBy);

if (Passenger == null)

throw new PassengerNotFoundException(ReservedBy, "Не найден пассажир с таким номером паспорта");

DateOfRes = DateTime.Now;

context.ReservationSet.Add(this);

foreach (var user in context.UserSet.Where(u => u.EMail == EMail))

user.Reservation.Add(this);

Flight = context.FlightDetailsSet.Find(FlightNo);

occupy\_place();

calculate\_fare();

Status = ReservationStatus.Unprocessed;

}

// Подтверждение брони

public void confirm(SkySharkDbContainer context)

{

if (Status == ReservationStatus.Cancelled)

throw new InvalidOperationException("Нельзя подтвердить отменённую бронь.");

calculate\_fare();

Passenger.FareCollected += Fare;

Flight.FareCollected += Fare;

Status = ReservationStatus.Confirmed;

}

// Вычисление цены билета

private void calculate\_fare()

{

if (ClassOfRes == FlightClass.Eco)

Fare = (decimal)(1.0 - Passenger.Discount) \* Flight.FareEco;

if (ClassOfRes == FlightClass.Business)

Fare = (decimal)(1.0 - Passenger.Discount) \* Flight.FareBn;

}

// Отмена неоплаченной брони

public void cancel(SkySharkDbContainer context)

{

if (Status == ReservationStatus.Confirmed)

throw new InvalidOperationException("Нельзя отменить оплаченную бронь.");

foreach (var user in context.UserSet.Where(u => u.EMail == EMail))

user.Reservation.Remove(this);

Status = ReservationStatus.Cancelled;

free\_place();

context.ReservationSet.Remove(this);

}

// Возмещение оплаченной брони

public void refund(SkySharkDbContainer context, string operatorId)

{

if (Status != ReservationStatus.Confirmed)

throw new InvalidOperationException("Вернуть можно лишь оплаченный билет.");

DateTime curTime = DateTime.Now;

DataBase.Cancellation canc = new Cancellation();

// За сутки и более возврат в 100%

// За два часа и более возврат 25%

// Иначе штраф 5000 р

if (curTime <= Flight.DepTime.AddDays(-1))

{

canc.initialize\_from\_ticket(this, Fare, "Возврат более чем за сутки", operatorId);

Passenger.FareCollected -= Fare;

Flight.FareCollected -= Fare;

Fare = 0;

}

else

if (curTime <= Flight.DepTime.AddHours(-2))

{

canc.initialize\_from\_ticket(this, Fare \* 0.25m, "Возврат более чем за два часа", operatorId);

Passenger.FareCollected -= Fare;

Flight.FareCollected -= Fare \* 0.25m;

Fare \*= 0.75m;

}

else

{

canc.initialize\_from\_ticket(this, Fare - 5000m, "Несвоевременный возврат", operatorId);

Passenger.FareCollected -= Fare;

Flight.FareCollected -= Fare + 5000m;

Fare = 5000m;

}

free\_place();

context.CancellationSet.Add(canc);

Status = ReservationStatus.Cancelled;

}

// Служебные функции

private void occupy\_place()

{

if (ClassOfRes == FlightClass.Eco)

Flight.EcoFree--;

if (ClassOfRes == FlightClass.Business)

Flight.BnFree--;

}

private void free\_place()

{

if (ClassOfRes == FlightClass.Eco)

Flight.EcoFree++;

if (ClassOfRes == FlightClass.Business)

Flight.BnFree++;

}

public bool IsCancelled() { return (Status == ReservationStatus.Cancelled); }

}

Для демонстрации синтаксиса Linq хорошо подойдут методы служебного класса TimeIntervalReport, необходимого в вычислениях статистики по операциям за месяц.

Выбор рейсов во временном интервале:

var flights = from f in context.FlightDetailsSet

where ((f.DepTime >= start) && (f.DepTime < start))

select f;

Вычисление суммарной прибыли с этих рейсов:

TotalFare = flights.Select(f => f.FareCollected).DefaultIfEmpty(0m).Sum();

Вычисление общей суммы компенсаций за интервал времени:

TotalRefunds =

(from f in flights

join r in context.ReservationSet

on f.FlightNo equals r.FlightNo

where r.Status == ReservationStatus.Cancelled

join c in context.CancellationSet

on r.TicketNo equals c.TicketNo

select c.Refund).DefaultIfEmpty(0m).Sum();

Вычисление пассажиропотока:

TotalPassengersTravelled =

(from f in flights

select f.SeatsEco + f.SeatsBn - f.EcoFree - f.BnFree).DefaultIfEmpty(0).Sum();

2.4 Анализ методологии

Приведённые выше отрывки кода – практически вся нетривиальная часть BLL. Ранее включаемая в бизнес-логику валидация аттрибутов благодаря мощнейшим средствам мета-документации платформы .NET будет вынесена на уровень Web-приложения. Привычные же отображения, хранимые процедуры и триггеры с помощью Linq to Entity заменяются на достаточно элегантный C# код, который динамически компилируется средой в соответствующие им SQL-запросы. Такой подход сильно отличается от традиционной разработки приложения базы данных, в первую очередь, в сторону большей скорости разработки. Из недостатков, традиционно – некоторое уменьшение производительности запросов и сложности в освоении технологии.

Код резервной архивации базы данных:

string dataTime = DateTime.Now.ToString("yyyy-MM-dd") + "-" + DateTime.Now.ToString("HH-mm");

string directory = HttpContext.Current.Server.MapPath("~/") + "backups\\" + dataTime + "\\";

string fileName = directory + dataTime + ".bak";

if (!Directory.Exists(directory))

Directory.CreateDirectory(directory);

db.Database.ExecuteSqlCommand(System.Data.Entity.TransactionalBehavior.DoNotEnsureTransaction, "BACKUP DATABASE SkySharkDb TO DISK = '" + fileName + "' WITH FORMAT, MEDIANAME = 'SkySharkDb\_backups', NAME = 'Full Backup of SkySharkDb'");

3. Технологическая часть

3.1 Платформа ASP.NET

ASP.NET – развитие старой технологии ASP корпорации Microsoft для создания динамических веб-приложений, работающих на базе CLR (Common Language Runtime, исполняющая среда .NET) и оптимизированных для серверов IIS. Проблема быстрой разработки Web-приложений сейчас актуальна как никогда, и ASP.NET – её решение от Microsoft.

В свою очередь, ASP.NET в современной версии предоставляет на выбор две основных парадигмы разработки Web: MVC Framework и Web Forms. Вторая значительно старше и более задокументирована, поэтому при выборе фреймворка для разработки Web-слоя выбор пал на неё.

3.2 Разметка страниц

Структура ASP.NET приложения сильно отличается от привычного .exe файла. Проект компилируется в библиотеку и набор файлов, которые в свою очередь передаются в обработку установленному на компьютере серверу IIS. Тот в свою очередь загружает эту библиотеку и обеспечивает коду в ней необходимую среду выполнения.

Структура исходников ASP.NET проекта основана на .aspx файлах, во многом напоминающих простую HTML разметку. В процессе работы сервера, .aspx файлы динамически компилируются в экземпляры класса страницы, дополненные каким угодно функционалом среды .NET. При запросе клиента страницы она рендерится в обычную HTML разметку и javaScript, который передаётся в браузер и интерпретируется. Страницы в ASP.NET WebForms представлены классовой иерархией, которую могут поддерживаться и разметки, реализуя механизм Master-страниц.

Разметка мастер страницы:

<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="MasterForm.Master.cs" Inherits="WebPortal.WebForms.MasterForm" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>SkyShark Airlines master page</title>

<link rel="stylesheet" type="text/css" href="MasterForm.css"/>

<asp:ContentPlaceHolder id="head" runat="server" />

</head>

<body style="margin: 0px">

<form id="form1" runat="server">

<asp:LinkButton id="pageHeader" runat="server" Font-Underline="False" PostBackUrl="~/WebForms/Common/Default.aspx" ValidationGroup="generalVG">

SkyShark Airlines

</asp:LinkButton>

<asp:Panel ID="tabPanel" runat="server" SkinID="skn\_tabPanel">

<asp:Panel id="left\_tabs" runat="server">

<asp:LinkButton ID="viewFlightsLink" CssClass="tabLink" runat="server" SkinID="skn\_greyTabHyperlinkMedium" Text="Рейсы" ValidationGroup="generalVG" PostBackUrl="~/WebForms/Common/FlightList.aspx"/>

<asp:LinkButton ID="companyDescLink" CssClass="tabLink" runat="server" SkinId="skn\_greyTabHyperlinkMedium" Text="О компании" PostBackUrl="~/WebForms/Common/About.aspx" ValidationGroup="generalVG"/>

<asp:LinkButton ID="manageUsers" CssClass="tabLink" runat="server" SkinId="skn\_greyTabHyperlinkMedium" Text="Аккаунты" PostBackUrl="~/WebForms/NetworkAdministrator/ManageUsers.aspx" Visible="False" ValidationGroup="generalVG"/>

...

Список страниц:

* MasterForm.Master – страница-шаблон для всех остальных страниц, задаёт цвет фона, основные стили интерфейса (CSS-файл), панель инструментов, глобальные параметры.
* About.aspx – страница с надписью о приложении.
* ChangePassword.aspx – страница для смены пароля текущего пользователя.
* Default.aspx – страница–приветствие. Она же домашняя страница, рис. 1.
* ErrorPage.aspx – для печати сообщений о критических ошибках.
* FlightList.aspx – для вывода списка рейсов
* Login.aspx – форма логина
* PersonalTickets.aspxz – личный кабинет.
* RegisterClient.aspx – создание новой учётной записи клиента
* ReserveSeat.aspx – оформление брони
* FinReport.aspx – финансовый отчёт
* NewFlight.aspx – авторизация нового рейса
* ReservationList.aspx – работа с бронью
* SearchReservations.aspx – поиск оформленной брони
* BackUp.aspx – резервное копирование БД
* ManageUsers.aspx – управление учётными записями пользователей.



рис. 1

3.3 Обработка исключений

Страницы разделены по папкам, соответствующим необходимому классу доступа аккаунта. В коде загрузки каждой страницы проверяется роль текущей сессии:

DataBase.UserRole role = (DataBase.UserRole)Session["Privilege"];

if (role != DataBase.UserRole.Administrator)

Response.Redirect("~/WebForms/Common/Default.aspx");

На каждой странице внизу расположена метка для вывода сообщений об ошибках:

<asp:Label ID="errorLabel" runat="server" Text="Ошибка" Visible="False" Font-Size="Medium" ForeColor="Red" CssClass="errorLabel"></asp:Label>

try

{

//Любой небезопасный код

}

catch (Exception error)

{

Global.print\_error(error, errorLabel);

}

Любые операции ввода и редактирования значений классов из модели базы данных проходят проверку на уровне клиента простыми валидаторами, а затем на уровне сервера теми же валидаторами и более сложными правилами. Информация об атрибутах сущностей берётся из составленных мета-данных (2.2). Сбор совершённых на странице ошибок ввода осуществляет компонент ValidationSummary:

<asp:ValidationSummary runat="server" CssClass="greyColor validationSummaryMiddle" ShowModelStateErrors="True" ValidationGroup="gridViewVG"/>

На случай ошибок уровня загрузки страницы предусмотрен переход на ErrorPage.aspx с печатью сообщения об ошибке:

private void Page\_Error(object sender, EventArgs e)

{

Session["Error"] = Server.GetLastError();

Response.Redirect("~/WebForms/Common/ErrorPage.aspx");

}

3.4 Представление и редактирование объектов.

Любое приложение, использующее базы данных, каким-либо образом позволяет пользователю получить данные с возможностью их изменения по каким-либо правилам. Наиболее подходящими для этих задач являются компоненты GridView (хорош для изображения и редактирования таблиц) и FormView (нужен для редактирования или создания одной конкретной сущности).

Ниже представлена разметка компонента GridView для отображения списка рейсов:

<asp:GridView ID="flightGridView" runat="server" AllowPaging="True" CssClass="userGridView" AllowSorting="True" PageSize="15" AutoGenerateColumns="False" ItemType="WebPortal.DataBase.FlightDetails" SelectMethod="flightGridView\_GetData" DataKeyNames="FlightNo" OnRowCommand="flightGridView\_RowCommand">

<Columns>

<asp:DynamicField DataField="FlightNo" ReadOnly="true" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="Origin" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="Destination" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="DepTime" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="ArrTime" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="AircraftType" ValidationGroup="gridViewVG" Visible="false"/>

<asp:DynamicField DataField="SeatsEco" ValidationGroup="gridViewVG" Visible="false"/>

<asp:DynamicField DataField="EcoFree" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="SeatsBn" ValidationGroup="gridViewVG" Visible="false"/>

<asp:DynamicField DataField="BnFree" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="FareEco" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="FareBn" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="FareCollected" ValidationGroup="gridViewVG" Visible="false"/>

<asp:TemplateField ShowHeader="False">

<ItemTemplate>

<asp:Button ID="ReserveButton" runat="server" CausesValidation="false" CommandName="Reserve"

Text="Бронировать" SkinID="skn\_reserveBtn" CommandArgument='<%# Eval("FlightNo")%>' Enabled='<%# (Convert.ToInt16(Eval("EcoFree")) > 0) ||(Convert.ToInt16(Eval("BnFree")) > 0) %>' />

</ItemTemplate>

</asp:TemplateField>

<asp:TemplateField ShowHeader="False" Visible="false">

<ItemTemplate>

<asp:Button ID="manageReservationsBtn" runat="server" CausesValidation="false" CommandName="manageReservations"

Text="Билеты" SkinID="skn\_reserveBtn" CommandArgument='<%# Eval("FlightNo")%>'/>

</ItemTemplate>

</asp:TemplateField>

</Columns>

</asp:GridView>

В приведённом выше листинге стоит обратить внимание на разметку столбцов таблицы с помощью компонентов DynamicField. Классы DynamicEntity и DynamicField входят в состав очень полезного набора классов, объединённого под именем DynamicData. Это семейство классов создано для динамического создания визуальных компонентов для отображения и редактирования произвольных структур данных с помощью мета-данных. Этот набор классов, хотя он и требует ручной доработки, существенно ускоряет программирование сценариев редактирования, отображения и добавления.

Ниже предоставлен отредактированный шаблон редактируемого поля типа DateTime, а также рис. 2 с визуальным представлением компонента FormView, использующего и этот шаблон в том числе.

<%@ Control Language="C#" CodeBehind="DateTime\_Edit.ascx.cs" Inherits="WebPortal.DateTime\_EditField" %>

<juice:Datepicker runat="server" ID="t1" TargetControlID="TextBox1" DateFormat="dd/mm/yy"/>

<asp:TextBox ID="TextBox1" runat="server" CssClass="DDTextBox" Text='<%# FieldValueEditString %>' Columns="12"></asp:TextBox>

<div style="display:inline">

чч

<asp:TextBox ID="Hours" runat="server" Width="15" Text="00" Columns="2"></asp:TextBox>

мм

<asp:TextBox ID="Minutes" runat="server" Width="15" Text="00" Columns="2"></asp:TextBox>

</div>

<asp:RequiredFieldValidator runat="server" ID="RequiredFieldValidator1" CssClass="DDControl DDValidator" ControlToValidate="TextBox1" Display="Static" Enabled="false" />

<asp:RegularExpressionValidator runat="server" ID="RegularExpressionValidator1" CssClass="DDControl DDValidator" ControlToValidate="TextBox1" Display="Static" Enabled="false" />

<asp:DynamicValidator runat="server" ID="DynamicValidator1" CssClass="DDControl DDValidator" ControlToValidate="TextBox1" Display="Static" />

<asp:CustomValidator runat="server" ID="DateValidator" CssClass="DDControl DDValidator" ControlToValidate="TextBox1" Display="Static" EnableClientScript="false" Enabled="false" OnServerValidate="DateValidator\_ServerValidate" />

<asp:CustomValidator runat="server" ID="HoursValidator" CssClass="DDControl DDValidator" ControlToValidate="Hours" Display="Static" EnableClientScript="false" Enabled="true" OnServerValidate="HoursValidator\_ServerValidate" />

<asp:CustomValidator runat="server" ID="MinutesValidator" CssClass="DDControl DDValidator" ControlToValidate="Minutes" Display="Static" EnableClientScript="false" Enabled="true" OnServerValidate="MinutesValidator\_ServerValidate" />

<asp:RequiredFieldValidator runat="server" ID="RequiredFieldValidator2" CssClass="DDControl DDValidator" ControlToValidate="Hours" Display="Static" Enabled="false" />

<asp:RequiredFieldValidator runat="server" ID="RequiredFieldValidator3" CssClass="DDControl DDValidator" ControlToValidate="Minutes" Display="Static" Enabled="false" />

<asp:RegularExpressionValidator runat="server" ID="RegularExpressionValidator2" CssClass="DDControl DDValidator" ControlToValidate="Hours" Display="Static" Enabled="false" ValidationExpression="^dd$"/>

<asp:RegularExpressionValidator runat="server" ID="RegularExpressionValidator3" CssClass="DDControl DDValidator" ControlToValidate="Minutes" Display="Static" Enabled="false" ValidationExpression="^dd$"/>

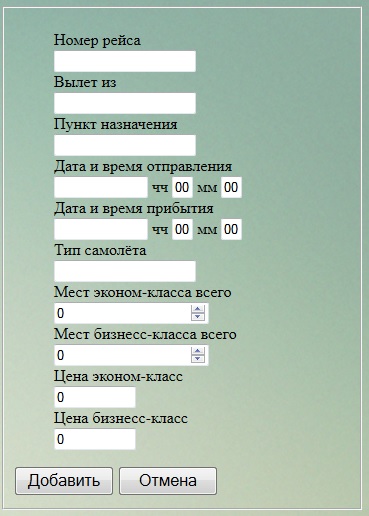


рис. 2

Пример таблицы, полученной с помощью компонента GridView, можно увидеть на рис 3.

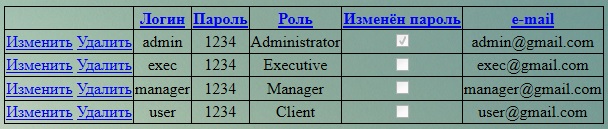


рис. 3

Разметка этой таблицы:

<asp:GridView ID="usersGridView" runat="server" AllowPaging="True" CssClass="userGridView" AllowSorting="True" PageSize="15" AutoGenerateColumns="False" ItemType="WebPortal.DataBase.User" SelectMethod="usersGridView\_GetData" UpdateMethod="usersGridView\_UpdateItem" DeleteMethod="usersGridView\_DeleteItem" DataKeyNames="Username">

<Columns>

<asp:CommandField ValidationGroup="gridViewVG" ShowEditButton="True" ShowDeleteButton="True" EditText="Изменить" DeleteText="Удалить" CancelText="Отмена" />

<asp:DynamicField DataField="Username" HeaderText="Логин" ReadOnly="true" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="Password" HeaderText="Пароль" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="Role" HeaderText="Роль" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="passwordChanged" HeaderText="Изменён пароль" ValidationGroup="gridViewVG"/>

<asp:DynamicField DataField="EMail" HeaderText="e-mail" ValidationGroup="gridViewVG"/>

</Columns>

</asp:GridView>

Заключение

По данным компании Netcraft на октябрь 2011 года, более 21 млн сайтов обслуживаются веб-сервером IIS, что составляет 12,46 % от общего числа веб-сайтов. При этом большая часть приложений, запущенных на IIS, базируется на технологии ASP.NET. Проведённое в процессе выполнения курсовой работы ознакомление с процессом разработки Web-приложений на платформе .NET позволило широко использовать развитый функционал платформы, многократно адаптировавшейся под нужды программистов со всего мира. Быстрая разработка небольших бизнес-приложений становится абсолютно реальной задачей при грамотном использовании средств разработки, а выложенные под лицензией Apache 2.0 исходники классов ASP.NET удовлетворят нужды в кастомизации процесса разработки в случае неудовлетворения функционалом.

В процессе реализации программы был получен ценный опыт организации доступа к данным, а так же новые знания, касающиеся языков HTML, CSS, JavaScript, а также процесса взаимодействия браузера и веб-сервера. К этому можно добавить полученное представление о современных тенденциях в развитии платформы .NET.

Список использованной литературы

1. “Microsoft C# Professional Projects” Geetahjali Arora, Balusubramaniam, Auaswamy, Nitin Pandey, 2002 год.
2. «ASP.NET tutorials» Набор обучающих статей с сайта <http://www.asp.net/web-forms/tutorials> за 2014 год.
3. «Entity Framework documentation» Раздел документации на портале Data Developer Center, подраздел MSDN, 2014 год.
4. Историческая справка о технологиях ASP и ADO с <http://en.wikipedia.org/>.
5. «Managing DbContext the right way with Entity Framework 6: an in-depth guide» Mehdi El Gueddari 2014 год.
6. «101 LINQ Samples» цикл примеров работы Linq с <http://code.msdn.microsoft.com>, 2012 год.
7. Документация по классам и пространствам имён платформы .NET на сайте MSDN.