Руководство разработчика Firebird

9 марта 2018 — 1.0

Спонсоры документации:
Platinum Sponsor

МОСКОВСКАЯ БИРЖА

Gold Sponsor

IBSurgeon

Руководство разработчика Firebird

Над документом работали:

Денис Симонов, Дмитрий Еманов, Дмитрий Кузьменко, Алексей Ковязин

Редактор:

Денис Симонов

Содержание

О рукодводстве разработчика Firebird

Основное внимание в книге уделено процессу разработки приложений с использованием различных технологий, сред разработки и языков программирования. Помимо этого, рассмотрено как устанавливать Firebird на Windows, Linux, MacOS, Android и конфигурировать его.

В написании книги участвовали – Денис Симонов, Дмитрий Еманов (ведущий архитектор Firebird), Роман Симаков (ведущий разработчик СУБД RedDatabase), Алексей Ковязин и Дмитрий Кузьменко из компании IBSurgeon/iBase.ru

Спонсором книги являются ПАО Московская Биржа и IBSurgeon (iBase.ru).

За основу взяты различные статьи, размещённые на сайте ibase.ru, Руководство по языку SQL СУБД Firebird, ReleaseNotes для различных версий Firebird и другая доступная документация. Часть статей по использованию компонент и драйверов ранее была размещена на habrahabr.

Глава 1

Создание базы данных для примеров

Перед описанием процесса создания приложений на различных языках программирования нам необходимо подготовить базу данных, которую мы будем использовать во всех последующих примерах.

Наше приложение будет работать с базой данных модель, которой представлена на рисунке ниже.

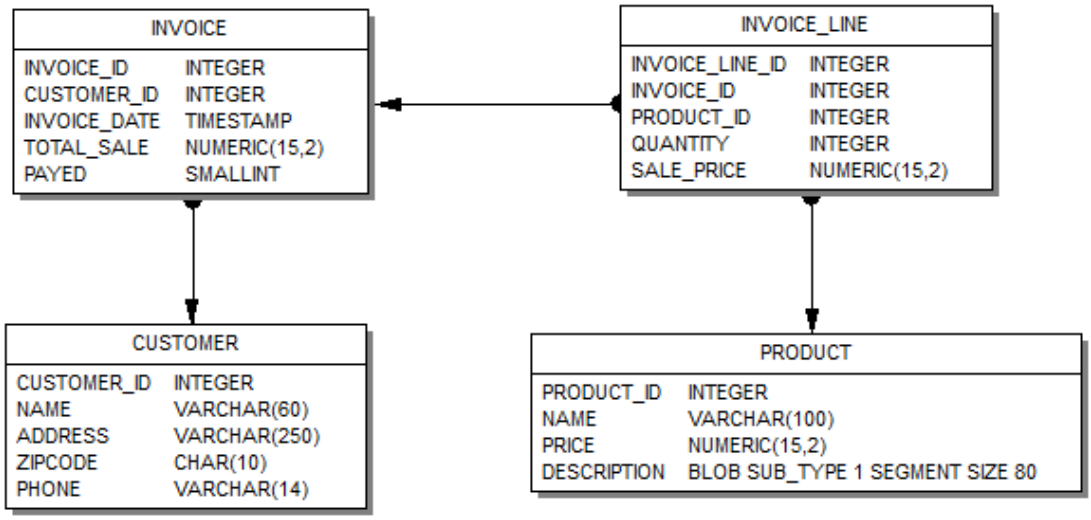

Рис. 1.1. Модель базы данных

> **Важно**
>
> Эта модель является просто примером. Ваша предметная область может быть сложнее, или полностью другой. Модель, используемая в этой статье, максимально упрощена для того, чтобы не загромождать описание работы с компонентами описанием создания и модификации модели данных.

Для создания базы данных из скрипта я буду использовать родной инструмент isql. Вы можете использовать любой другой инструмент администрирования, например FlameRobin, SQLLY Studio, IBExpert и др.

Запустим isql, после появления приглашения ко вводу введём следующий скрипт

```
CREATE DATABASE 'localhost:D:\fbdata\2.5\examples.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
```

```
PAGE_SIZE 8192 DEFAULT CHARACTER SET UTF8;
```

Вы можете создать базу данных от имени любого пользователя необязательно SYSDBA. Пользователь, от имени которого создаётся база данных, становится владельцем БД и обладает полным доступом ко всем объектам метаданных.

Современные версии Firebird поддерживают следующие размеры страниц 4096, 8192 и 16384. Размер страницы 8192 подходит в большинстве случаев.

Необязательное предложение DEFAULT CHARACTER SET задаёт набор символов по умолчанию для строковых типов данных. Наборы символов применяются для типов CHAR, VARCHAR и BLOB. Для работы с символами кириллицы вы можете использовать различные однобайтовые кодировки WIN1251, ISO8859_5, CYRL, DOS866 или универсальную многобайтовую кодировку UTF8.

В настоящее время все современные языки программирования поддерживают работу с UTF8, поэтому выбираем эту кодировку.

Теперь можно выйти из сеанса isql, для этого наберите команду

```
EXIT;
```

При вводе базы данных в эксплуатацию удобней работать с ней с использованием алиасов, кроме того это "повышает безопасность" вашей базы данных в том смысле что путь к первичному файлу базы данных не виден в строке подключения.

В Firebird 2.5 алиас БД задаётся в файле `aliases.conf` следующим образом

```
examples = D:\fbdata\2.5\examples.fdb
```

В Firebird 3.0 алиас БД задаётся в файле `databases.conf`. Кроме задания алиаса БД вы можете также задать некоторые параметры уровня базы данных, например, размер страничного кеша и объём оперативной памяти под сортировку

```
examples = D:\fbdata\3.0\examples.fdb
{
 DefaultDbCachePages = 16K
 TempCacheLimit = 512M
}
```

Теперь составим скрипт для создания базы данных.

Сначала определим некоторые домены, которые будем использовать в определении столбцов.

```
CREATE DOMAIN D_BOOLEAN AS
SMALLINT
```

```
CHECK (VALUE IN (0, 1));

COMMENT ON DOMAIN D_BOOLEAN IS
'Boolean type. 0 - FALSE, 1- TRUE';

CREATE DOMAIN D_MONEY AS
NUMERIC(15,2);

CREATE DOMAIN D_ZIPCODE AS
CHAR(10) CHARACTER SET UTF8
CHECK (TRIM(TRAILING FROM VALUE) SIMILAR TO '[0-9]+');

COMMENT ON DOMAIN D_ZIPCODE IS
'Zip code';
```

> **Примечание**
>
> В Firebird 3.0 есть нативный тип BOOLEAN, но поскольку этот тип появился недавно, то он может не поддерживаться в некоторых драйверах. Кроме того, мы будем строить наши приложения так, чтобы они могли работать одновременно с Firebird 2.5 и Firebird 3.0.

Теперь перейдём к таблицам. Первой таблицей будет таблица заказчиков (CUSTOMER). Для неё как и остальных таблиц создадим последовательность (генератор) и соответствующий триггер для реализации автоинкрементных столбцов.

```
CREATE GENERATOR GEN_CUSTOMER_ID;

CREATE TABLE CUSTOMER (
    CUSTOMER_ID  INTEGER NOT NULL,
    NAME         VARCHAR(60) NOT NULL,
    ADDRESS      VARCHAR(250),
    ZIPCODE      D_ZIPCODE,
    PHONE        VARCHAR(14),
    CONSTRAINT PK_CUSTOMER PRIMARY KEY (CUSTOMER_ID)
);

SET TERM ^ ;

CREATE OR ALTER TRIGGER CUSTOMER_BI FOR CUSTOMER
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.CUSTOMER_ID IS NULL) THEN
    NEW.CUSTOMER_ID = NEXT VALUE FOR GEN_CUSTOMER_ID;
END
^

SET TERM ; ^

COMMENT ON TABLE CUSTOMER IS
'Customers';

COMMENT ON COLUMN CUSTOMER.CUSTOMER_ID IS
'Customer Id';
```

```
COMMENT ON COLUMN CUSTOMER.NAME IS
'Name';

COMMENT ON COLUMN CUSTOMER.ADDRESS IS
'Address';

COMMENT ON COLUMN CUSTOMER.ZIPCODE IS
'Zip Code';

COMMENT ON COLUMN CUSTOMER.PHONE IS
'Phone';
```

Примечание

- В Firebird 3.0 вы можете использовать IDENTITY столбцы в качестве автоинкрементных полей. В этом случае создавать генератор и BEFORE INSERT триггер нет необходимости, а скрипт создания таблицы будет выглядеть следующим образом:

```
CREATE TABLE CUSTOMER (
    CUSTOMER_ID  INTEGER GENERATED BY DEFAULT AS IDENTITY,
    NAME         VARCHAR(60) NOT NULL,
    ADDRESS      VARCHAR(250),
    ZIPCODE      D_ZIPCODE,
    PHONE        VARCHAR(14),
    CONSTRAINT PK_CUSTOMER PRIMARY KEY (CUSTOMER_ID)
);
```

- В Firebird 3.0 для использования последовательности (генератора) необходима привилегия USAGE, поэтому в скрипт необходимо добавить следующую строчку:

```
GRANT USAGE ON SEQUENCE GEN_CUSTOMER_ID TO TRIGGER CUSTOMER_BI;
```

Теперь составим скрипт для создание таблицы товаров (PRODUCT).

```
CREATE GENERATOR GEN_PRODUCT_ID;

CREATE TABLE PRODUCT (
    PRODUCT_ID   INTEGER NOT NULL,
    NAME         VARCHAR(100) NOT NULL,
    PRICE        D_MONEY NOT NULL,
    DESCRIPTION  BLOB SUB_TYPE TEXT,
    CONSTRAINT PK_PRODUCT PRIMARY KEY (PRODUCT_ID)
);

SET TERM ^ ;

CREATE OR ALTER TRIGGER PRODUCT_BI FOR PRODUCT
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.PRODUCT_ID IS NULL) THEN
```

```
    NEW.PRODUCT_ID = NEXT VALUE FOR GEN_PRODUCT_ID;
END
^

SET TERM ; ^

COMMENT ON TABLE PRODUCT IS
'Goods';

COMMENT ON COLUMN PRODUCT.PRODUCT_ID IS
'Product Id';

COMMENT ON COLUMN PRODUCT.NAME IS
'Name';

COMMENT ON COLUMN PRODUCT.PRICE IS
'Price';

COMMENT ON COLUMN PRODUCT.DESCRIPTION IS
'Description';
```

Примечание

В Firebird 3.0 необходимо добавить в скрипт команду для выдачи привилегии USAGE на последовательность (генератор)

```
GRANT USAGE ON SEQUENCE GEN_PRODUCT_ID TO TRIGGER PRODUCT_BI;
```

Скрипт создания таблицы INVOICE (счёт-фактуры) выглядит так:

```
CREATE GENERATOR GEN_INVOICE_ID;

CREATE TABLE INVOICE (
    INVOICE_ID    INTEGER NOT NULL,
    CUSTOMER_ID   INTEGER NOT NULL,
    INVOICE_DATE  TIMESTAMP,
    TOTAL_SALE    D_MONEY,
    PAID          D_BOOLEAN DEFAULT 0 NOT NULL,
    CONSTRAINT PK_INVOICE PRIMARY KEY (INVOICE_ID)
);

ALTER TABLE INVOICE ADD CONSTRAINT FK_INVOCE_CUSTOMER
FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID);

CREATE INDEX INVOICE_IDX_DATE ON INVOICE (INVOICE_DATE);

SET TERM ^ ;

CREATE OR ALTER TRIGGER INVOICE_BI FOR INVOICE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.INVOICE_ID IS NULL) THEN
```

```
    NEW.INVOICE_ID = GEN_ID(GEN_INVOICE_ID,1);
END
^

SET TERM ; ^

COMMENT ON TABLE INVOICE IS
'Invoices';

COMMENT ON COLUMN INVOICE.INVOICE_ID IS
'Invoice number';

COMMENT ON COLUMN INVOICE.CUSTOMER_ID IS
'Customer Id';

COMMENT ON COLUMN INVOICE.INVOICE_DATE IS
'The date of issuance invoices';

COMMENT ON COLUMN INVOICE.TOTAL_SALE IS
'Total sum';

COMMENT ON COLUMN INVOICE.PAID IS
'Payed';
```

Для поля INVOICE_DATE создан индекс, поскольку мы будем фильтровать счёт-фактуры по дате, так чтобы они попадали в рабочий период.

> **Примечание**
>
> В Firebird 3.0 необходимо добавить в скрипт команду для выдачи привилегии USAGE на последовательность (генератор)
>
> ```
> GRANT USAGE ON SEQUENCE GEN_INVOICE_ID TO TRIGGER INVOICE_BI;
> ```

Скрипт создания таблицы INVOICE_LINE выглядит следующим образом:

```
CREATE GENERATOR GEN_INVOICE_LINE_ID;

CREATE TABLE INVOICE_LINE (
    INVOICE_LINE_ID  INTEGER NOT NULL,
    INVOICE_ID       INTEGER NOT NULL,
    PRODUCT_ID       INTEGER NOT NULL,
    QUANTITY         NUMERIC(15,0) NOT NULL,
    SALE_PRICE       D_MONEY NOT NULL,
    CONSTRAINT PK_INVOICE_LINE PRIMARY KEY (INVOICE_LINE_ID)
);

ALTER TABLE INVOICE_LINE ADD CONSTRAINT FK_INVOICE_LINE_INVOICE
FOREIGN KEY (INVOICE_ID) REFERENCES INVOICE (INVOICE_ID);

ALTER TABLE INVOICE_LINE ADD CONSTRAINT FK_INVOICE_LINE_PRODUCT
FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID);
```

```
SET TERM ^ ;

CREATE OR ALTER TRIGGER INVOICE_LINE_BI FOR INVOICE_LINE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.INVOICE_LINE_ID IS NULL) THEN
    NEW.INVOICE_LINE_ID = NEXT VALUE FOR GEN_INVOICE_LINE_ID;
END
^

SET TERM ; ^

COMMENT ON TABLE INVOICE_LINE IS
'Invoice lines';

COMMENT ON COLUMN INVOICE_LINE.INVOICE_LINE_ID IS
'Invoice line Id';

COMMENT ON COLUMN INVOICE_LINE.INVOICE_ID IS
'Invoice number';

COMMENT ON COLUMN INVOICE_LINE.PRODUCT_ID IS
'Product Id';

COMMENT ON COLUMN INVOICE_LINE.QUANTITY IS
'Quantity';

COMMENT ON COLUMN INVOICE_LINE.SALE_PRICE IS
'Price';
```

Примечание

В Firebird 3.0 необходимо добавить в скрипт команду для выдачи привилегии USAGE на последовательность (генератор)

```
GRANT USAGE ON SEQUENCE GEN_INVOICE_LINE_ID TO TRIGGER INVOICE_LINE_BI;
```

Часть бизнес логики будет реализовано с помощью хранимых процедур.

Процедура добавления новой счёт-фактуры довольно простая и выглядит следующим образом:

```
SET TERM ^ ;

CREATE OR ALTER PROCEDURE SP_ADD_INVOICE (
    INVOICE_ID INTEGER,
    CUSTOMER_ID INTEGER,
    INVOICE_DATE TIMESTAMP = CURRENT_TIMESTAMP)
AS
BEGIN
  INSERT INTO INVOICE (
    INVOICE_ID,
```

```
    CUSTOMER_ID,
    INVOICE_DATE,
    TOTAL_SALE,
    PAID
  )
  VALUES (
    :INVOICE_ID,
    :CUSTOMER_ID,
    :INVOICE_DATE,
    0,
    0
  );
END
^

SET TERM ; ^

COMMENT ON PROCEDURE SP_ADD_INVOICE IS
'Adding Invoice';

COMMENT ON PARAMETER SP_ADD_INVOICE.INVOICE_ID IS
'Invoice number';

COMMENT ON PARAMETER SP_ADD_INVOICE.CUSTOMER_ID IS
'Customer Id';

COMMENT ON PARAMETER SP_ADD_INVOICE.INVOICE_DATE IS
'Date';

GRANT INSERT ON INVOICE TO PROCEDURE SP_ADD_INVOICE;
```

Процедура изменения счёт-фактуры уже немного сложнее. Добавим в неё следующее условие, если счёт-фактура оплачена, то её редактирование запрещено. Создадим исключение, которое будет возбуждаться когда счёт фактура оплачена.

```
CREATE EXCEPTION E_INVOICE_ALREADY_PAYED 'Change is impossible, invoice paid.';
```

Сама хранимая процедура будет выглядеть следующем образом:

```
SET TERM ^ ;

CREATE OR ALTER PROCEDURE SP_EDIT_INVOICE (
    INVOICE_ID INTEGER,
    CUSTOMER_ID INTEGER,
    INVOICE_DATE TIMESTAMP)
AS
BEGIN
  IF (EXISTS(SELECT *
             FROM INVOICE
             WHERE INVOICE_ID = :INVOICE_ID
               AND PAID = 1)) THEN
    EXCEPTION E_INVOICE_ALREADY_PAYED;
```

```
  UPDATE INVOICE
  SET CUSTOMER_ID = :CUSTOMER_ID,
      INVOICE_DATE = :INVOICE_DATE
  WHERE INVOICE_ID = :INVOICE_ID;
END
^

SET TERM ; ^

COMMENT ON PROCEDURE SP_EDIT_INVOICE IS
'Editing invoice';

COMMENT ON PARAMETER SP_EDIT_INVOICE.INVOICE_ID IS
'Invoice number';

COMMENT ON PARAMETER SP_EDIT_INVOICE.CUSTOMER_ID IS
'Customer Id';

COMMENT ON PARAMETER SP_EDIT_INVOICE.INVOICE_DATE IS
'Date';

GRANT SELECT,UPDATE ON INVOICE TO PROCEDURE SP_EDIT_INVOICE;
```

Примечание

В Firebird 3.0 для исключений требуется привилегия USAGE, поэтому необходимо добавить в скрипт следующую строку

```
GRANT USAGE ON EXCEPTION E_INVOICE_ALREADY_PAYED TO PROCEDURE SP_EDIT_INVOICE;
```

Для удаления счёт-фактуры будем использовать процедуру SP_DELETE_INVOICE. В этой процедуре будем поверять не оплачена ли счёт-фактура, и если она оплачена бросать исключение.

```
SET TERM ^ ;

CREATE OR ALTER PROCEDURE SP_DELETE_INVOICE (
    INVOICE_ID INTEGER)
AS
BEGIN
  IF (EXISTS(SELECT * FROM INVOICE
             WHERE INVOICE_ID = :INVOICE_ID
               AND PAID = 1)) THEN
    EXCEPTION E_INVOICE_ALREADY_PAYED;

  DELETE FROM INVOICE WHERE INVOICE_ID = :INVOICE_ID;
END
^

SET TERM ; ^

COMMENT ON PROCEDURE SP_DELETE_INVOICE IS
'Deleting invoices';
```

```
GRANT SELECT,DELETE ON INVOICE TO PROCEDURE SP_DELETE_INVOICE;
```

Примечание

В Firebird 3.0 для исключений требуется привилегия USAGE, поэтому необходимо добавить в скрипт следующую строку

```
GRANT USAGE ON EXCEPTION E_INVOICE_ALREADY_PAYED TO PROCEDURE SP_DELETE_INVOICE;
```

Добавим ещё одну процедуру для оплаты счёт фактуры.

```
SET TERM ^ ;

CREATE OR ALTER PROCEDURE SP_PAY_FOR_INOVICE (
    INVOICE_ID INTEGER)
AS
BEGIN
  IF (EXISTS(SELECT *
             FROM INVOICE
             WHERE INVOICE_ID = :INVOICE_ID
               AND PAID = 1)) THEN
    EXCEPTION E_INVOICE_ALREADY_PAYED;

  UPDATE INVOICE
  SET PAID = 1
  WHERE INVOICE_ID = :INVOICE_ID;
END
^

SET TERM ; ^

COMMENT ON PROCEDURE SP_PAY_FOR_INOVICE IS
'Payment of invoices';

COMMENT ON PARAMETER SP_PAY_FOR_INOVICE.INVOICE_ID IS
'Invoice number';

GRANT SELECT,UPDATE ON INVOICE TO PROCEDURE SP_PAY_FOR_INOVICE;
```

Примечание

В Firebird 3.0 для исключений требуется привилегия USAGE, поэтому необходимо добавить в скрипт следующую строку

```
GRANT USAGE ON EXCEPTION E_INVOICE_ALREADY_PAYED TO PROCEDURE SP_PAY_FOR_INOVICE;
```

Теперь создадим процедуры для управления позициями счёт-фактуры. Эти процедуры будут проверять не оплачена ли счёт фактура и запрещать любые манипуляции над строками

оплаченных счёт фактур. Кроме того, процедуры будут корректировать сумму счёт-фактуры в зависимости от количества выписанного товара и его стоимости.

Процедура добавления позиции счёт фактуры выглядит так

```
SET TERM ^ ;

CREATE OR ALTER PROCEDURE SP_ADD_INVOICE_LINE (
    INVOICE_ID INTEGER,
    PRODUCT_ID INTEGER,
    QUANTITY INTEGER)
AS
DECLARE sale_price D_MONEY;
DECLARE paid       D_BOOLEAN;
BEGIN
  SELECT
      paid
  FROM
      invoice
  WHERE
       invoice_id = :invoice_id
  INTO :paid;

  -- It does not allow you to edit already paid invoice.
  IF (paid = 1) THEN
    EXCEPTION E_INVOICE_ALREADY_PAYED;

  SELECT
      price
  FROM
      product
  WHERE
       product_id = :product_id
  INTO :sale_price;

  INSERT INTO invoice_line (invoice_line_id,
                            invoice_id,
                            product_id,
                            quantity,
                            sale_price)
  VALUES (NEXT VALUE FOR gen_invoice_line_id,
          :invoice_id,
          :product_id,
          :quantity,
          :sale_price);

  -- Increase the amount of the account.
  UPDATE invoice
  SET total_sale = COALESCE(total_sale, 0) + :sale_price * :quantity
  WHERE invoice_id = :invoice_id;
END
^

SET TERM ; ^

COMMENT ON PROCEDURE SP_ADD_INVOICE_LINE IS
```

```
'Adding line invoices';

COMMENT ON PARAMETER SP_ADD_INVOICE_LINE.INVOICE_ID IS
'Invoice number';

COMMENT ON PARAMETER SP_ADD_INVOICE_LINE.PRODUCT_ID IS
'Product Id';

COMMENT ON PARAMETER SP_ADD_INVOICE_LINE.QUANTITY IS
'Quantity';


GRANT SELECT,UPDATE ON INVOICE TO PROCEDURE SP_ADD_INVOICE_LINE;
GRANT SELECT ON PRODUCT TO PROCEDURE SP_ADD_INVOICE_LINE;
GRANT INSERT ON INVOICE_LINE TO PROCEDURE SP_ADD_INVOICE_LINE;

-- только для Firebird 3.0 и выше
GRANT USAGE ON EXCEPTION E_INVOICE_ALREADY_PAYED TO PROCEDURE SP_ADD_INVOICE_LINE;
GRANT USAGE ON SEQUENCE GEN_INVOICE_LINE_ID TO PROCEDURE SP_ADD_INVOICE_LINE;
```

Процедура редактирования позиции счёт-фактуры выглядит следующим образом:

```
SET TERM ^ ;

CREATE OR ALTER PROCEDURE SP_EDIT_INVOICE_LINE (
    INVOICE_LINE_ID INTEGER,
    QUANTITY INTEGER)
AS
DECLARE invoice_id INT;
DECLARE price      D_MONEY;
DECLARE paid      D_BOOLEAN;
BEGIN
  SELECT
      product.price,
      invoice.invoice_id,
      invoice.paid
  FROM
      invoice_line
      JOIN invoice ON invoice.invoice_id = invoice_line.invoice_id
      JOIN product ON product.product_id = invoice_line.product_id
  WHERE
       invoice_line.invoice_line_id = :invoice_line_id
  INTO :price,
       :invoice_id,
       :paid;

  -- It does not allow you to edit already paid invoice.
  IF (paid = 1) THEN
    EXCEPTION E_INVOICE_ALREADY_PAYED;

  -- Update price and quantity.
  UPDATE invoice_line
  SET sale_price = :price,
      quantity = :quantity
  WHERE invoice_line_id = :invoice_line_id;
```

```
  -- Now update the amount of the account.
  MERGE INTO invoice
  USING (SELECT
              invoice_id,
              SUM(sale_price * quantity) AS total_sale
          FROM invoice_line
          WHERE invoice_id = :invoice_id
          GROUP BY invoice_id) L
  ON invoice.invoice_id = L.invoice_id
  WHEN MATCHED THEN
      UPDATE SET total_sale = L.total_sale;
END
^

SET TERM ; ^

COMMENT ON PROCEDURE SP_EDIT_INVOICE_LINE IS
'Editing invoice line';

COMMENT ON PARAMETER SP_EDIT_INVOICE_LINE.INVOICE_LINE_ID IS
'Invoice line id';

COMMENT ON PARAMETER SP_EDIT_INVOICE_LINE.QUANTITY IS
'Quantity';

GRANT SELECT,UPDATE ON INVOICE_LINE TO PROCEDURE SP_EDIT_INVOICE_LINE;
GRANT SELECT,UPDATE ON INVOICE TO PROCEDURE SP_EDIT_INVOICE_LINE;
GRANT SELECT ON PRODUCT TO PROCEDURE SP_EDIT_INVOICE_LINE;

-- только для Firebird 3.0 и выше
GRANT USAGE ON EXCEPTION E_INVOICE_ALREADY_PAYED TO PROCEDURE SP_EDIT_INVOICE_LINE;
```

Процедура удаления позиции счёт-фактуры выглядит следующим образом:

```
SET TERM ^ ;

CREATE OR ALTER PROCEDURE SP_DELETE_INVOICE_LINE (
    INVOICE_LINE_ID INTEGER)
AS
DECLARE invoice_id INT;
DECLARE price      D_MONEY;
DECLARE quantity   INT;
BEGIN
  IF (EXISTS(SELECT *
             FROM invoice_line
                 JOIN invoice ON invoice.invoice_id = invoice_line.invoice_id
             WHERE invoice.paid = 1
               AND invoice_line.invoice_line_id = :invoice_line_id)) THEN
    EXCEPTION E_INVOICE_ALREADY_PAYED;

  DELETE FROM invoice_line
  WHERE invoice_line.invoice_line_id = :invoice_line_id
  RETURNING invoice_id, quantity, sale_price
  INTO invoice_id, quantity, price;
```

```
  -- Reduce the amount of the account.
  UPDATE invoice
  SET total_sale = total_sale - :quantity * :price
  WHERE invoice_id = :invoice_id;
END
^

SET TERM ; ^

COMMENT ON PROCEDURE SP_DELETE_INVOICE_LINE IS
'Deleting invoce line';

COMMENT ON PARAMETER SP_DELETE_INVOICE_LINE.INVOICE_LINE_ID IS
'Код строки счёт-фактуры';


GRANT SELECT,DELETE ON INVOICE_LINE TO PROCEDURE SP_DELETE_INVOICE_LINE;
GRANT SELECT,UPDATE ON INVOICE TO PROCEDURE SP_DELETE_INVOICE_LINE;

-- только в Firebird 3.0 и выше
GRANT USAGE ON EXCEPTION E_INVOICE_ALREADY_PAYED TO PROCEDURE SP_DELETE_INVOICE_LINE;
```

Теперь необходимо создать роли и выдать необходимые привилегии. Создадим две роли MANAGER и SUPERUSER. Первая будет обладать ограниченным набором привилегий, а вторая практически всеми возможностями системы.

```
CREATE ROLE MANAGER;
CREATE ROLE SUPERUSER;
```

Роль MANAGER имеет возможность читать любую таблицу и управлять счёт-фактурами через соответствующие процедуры:

```
GRANT SELECT ON CUSTOMER TO MANAGER;
GRANT SELECT ON INVOICE TO MANAGER;
GRANT SELECT ON INVOICE_LINE TO MANAGER;
GRANT SELECT ON PRODUCT TO MANAGER;
GRANT EXECUTE ON PROCEDURE SP_ADD_INVOICE TO MANAGER;
GRANT EXECUTE ON PROCEDURE SP_ADD_INVOICE_LINE TO MANAGER;
GRANT EXECUTE ON PROCEDURE SP_DELETE_INVOICE TO MANAGER;
GRANT EXECUTE ON PROCEDURE SP_DELETE_INVOICE_LINE TO MANAGER;
GRANT EXECUTE ON PROCEDURE SP_EDIT_INVOICE TO MANAGER;
GRANT EXECUTE ON PROCEDURE SP_EDIT_INVOICE_LINE TO MANAGER;
GRANT EXECUTE ON PROCEDURE SP_PAY_FOR_INOVICE TO MANAGER;
GRANT USAGE ON SEQUENCE GEN_INVOICE_ID TO MANAGER;
```

Роль SUPERUSER имеет возможность читать любую таблицу. редактировать справочники и вносить изменения в счёт-фактуры через соответствующие процедуры:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON CUSTOMER TO SUPERUSER;
GRANT SELECT ON INVOICE TO SUPERUSER;
```

```
GRANT SELECT ON INVOICE_LINE TO SUPERUSER;
GRANT SELECT, INSERT, UPDATE, DELETE ON PRODUCT TO SUPERUSER;
GRANT EXECUTE ON PROCEDURE SP_ADD_INVOICE TO SUPERUSER;
GRANT EXECUTE ON PROCEDURE SP_ADD_INVOICE_LINE TO SUPERUSER;
GRANT EXECUTE ON PROCEDURE SP_DELETE_INVOICE TO SUPERUSER;
GRANT EXECUTE ON PROCEDURE SP_DELETE_INVOICE_LINE TO SUPERUSER;
GRANT EXECUTE ON PROCEDURE SP_EDIT_INVOICE TO SUPERUSER;
GRANT EXECUTE ON PROCEDURE SP_EDIT_INVOICE_LINE TO SUPERUSER;
GRANT EXECUTE ON PROCEDURE SP_PAY_FOR_INOVICE TO SUPERUSER;
GRANT USAGE ON SEQUENCE GEN_CUSTOMER_ID TO SUPERUSER;
GRANT USAGE ON SEQUENCE GEN_INVOICE_ID TO SUPERUSER;
GRANT USAGE ON SEQUENCE GEN_PRODUCT_ID TO SUPERUSER;
```

Теперь создадим пользователей и назначим им роли:

```
CREATE USER IVAN PASSWORD 'z12a';
CREATE USER ANNA PASSWORD 'lh67';

GRANT MANAGER TO ANNA;
GRANT MANAGER TO IVAN WITH ADMIN OPTION;
GRANT SUPERUSER TO IVAN;
```

> **Примечание**
>
> Пользователь IVAN может передавать роль MANAGER другим пользователям.

Теперь сохраним наш скрипт в файле `examples.sql`. Вы можете скачать готовые файлы скриптов по ссылкам https://github.com/sim1984/example-db_2_5/archive/1.0.zip и https://github.com/sim1984/example-db_3_0/archive/1.0.zip

Выполним наш скрипт в созданной ранее базе данных:

```
isql -user sysdba -password masterkey "localhost:examples" -i "d:\examples-db\examples.sql"
```

Теперь когда база данных создана вы можете наполнить её тестовыми данными. Сделать это можно с помощью различных инструментов. Каким образом наполнить базу данных тестовыми данными мы не будем рассказывать. Вы можете сделать это самостоятельно.

Заключение

Вы можете скачать готовые базы данных по ссылкам https://github.com/sim1984/example-db_2_5/releases/download/1.0/examples.fdb и https://github.com/sim1984/example-db_3_0/releases/download/1.0/examples.fdb.

Глава 2

Разработка приложений Firebird на Delphi

В данной главе будет описан процесс создания приложений для СУБД Firebird с использованием компонентов доступа FireDac™ и среды Embarcadero Delphi™ XE5. FireDac™ является стандартным набором компонентов доступа к различным базам данных начиная с Delphi XE3.

Создание проекта

Создайте новый проект `File->New->VCL Forms Application - Delphi`. В новый проект добавьте новый дата модуль `File->New->Other`, в появившемся мастере выберите `Delphi Projects->Delphi Files->Data Module`. Этот дата модуль будет главным в нашем проекте. Он будет содержать некоторые экземпляры глобальных компонентов доступа, которые должны быть доступны всем формам, которые должны работать с данными. Например, таким компонентом является `TFDConnection`.

Компонент TFDConnection

Компонент `TFDConnection` обеспечивает подключение к различным типам баз данных. Будем указывать экземпляр этого компонента в свойствах `Connection` остальных компонентов FireDac. К какому именно типу баз данных будет происходить подключение, зависит от значения свойства `DriverName`. Для доступа к Firebird нам необходимо выставить это свойство в значение `FB`. Для того чтобы подключение знало, с какой именно библиотекой доступа необходимо работать, разместим в главном дата модуле компонент `TFBPhysFBDriverLink`. Его свойство `VendorLib` позволяет указывать путь до клиентской библиотеки. Если оно не указано, то подключение к Firebird будет осуществляться через библиотеки, зарегистрированные в системе, например в `system32`, что в ряде случаев может быть нежелательно.

Путь к клиентской библиотеке

Мы будем размещать необходимую библиотеку доступа в папке `fbclient`, которая расположена в папке приложения. Для этого в коде на событие `OnCreate` дата модуля пропишем следующий код.

```
// указываем путь до клиентской библиотеки
xAppPath := ExtractFileDir(Application.ExeName) + PathDelim;
```

```
FDPhysFBDriverLink.VendorLib := xAppPath + 'fbclient' + PathDelim + 'fbclient.dll';
```

Важное замечание о "разрядности"

Если вы компилируете 32 разрядное приложение, то вы должны использовать 32 разрядную библиотеку `fbclient.dll`. Для 64 разрядного – 64 разрядную.

Помимо файла `fbclient.dll` в ту же папку желательно поместить библиотеки `msvcp80.dll` и `msvcr80.dll` (для Firebird 2.5), и `msvcp100.dll` и `msvcr100.dll` (для Firebird 3.0). Эти библиотеки можно найти либо в подпапке `bin` (Firebird 2.5), либо в корневой папке сервера (Firebird 3.0).

Для того чтобы приложение правильно отображало собственные ошибки firebird, необходимо также скопировать файл `firebird.msg`.

- Для Firebird 2.5 и в более ранних версиях он должен находиться на один уровень выше каталога клиентской библиотеки, т.е. в нашем случае в каталоге приложения.
- Для Firebird 3 он должен находиться в каталоге клиентской библиотеки, т.е. в каталоге `fbclient`.

Разработка с использованием встроенного сервера

Если вам необходимо чтобы ваше приложение работало без установленного сервера Firebird, т.е. в режиме Embedded, то для Firebird 2.5 необходимо заменить `fbclient.dll` на `fbembed.dll`. Убедитесь что битность библиотеки (64 или 32) соответсвует битности приложения. При желании имя библиотеки можно вынести в конфигурационный файл вашего приложения. Для

Firebird 3.0 ничего изменять не требуется (режим работы зависит от строки подключения и значения параметра `Providers` в файле `firebird.conf` или `databases.conf`).

Подсказка

Даже если ваше приложение будет работать с Firebird в режиме Embedded, разработку удобнее вести под полноценным сервером. Дело в том, что в режиме Embedded Firebird работает в одном адресном пространстве с вашим приложением, что может привести к нежелательным последствиям при возникновении ошибок в вашем приложении. Кроме того, до версии Firebird 2.5 (и в Firebird 3.0 в умалчиваемой конфигурации) любое приложение, подключающееся к базе данных во встроенном режиме, должно иметь возможность получить эксклюзивный доступ к этой базе данных. Как только это соединение будет успешным, никакие другие встроенные соединения не будут возможны. Когда вы подключаетесь к своей базе данных в Delphi IDE, установленное соединение находится в адресном пространстве Delphi, что предотвращает успешную отладку приложения из IDE.

Учитывайте, что Embedded версия Firebird 3.0 требует эксклюзивного доступа, если он сконфигурирован для работы в режиме Super (SuperServer).

Параметры подключения

Параметры подключения к базе данных содержатся в свойстве `Params` (имя пользователя, пароль, набор символов соединения и др.) компонента `TFDConnection`. Если воспользоваться

редактором свойств `TFDConnection` (двойной клик на компоненте), то упомянутые свойства будут заполнены автоматически. Набор этих свойств зависит от типа базы данных.

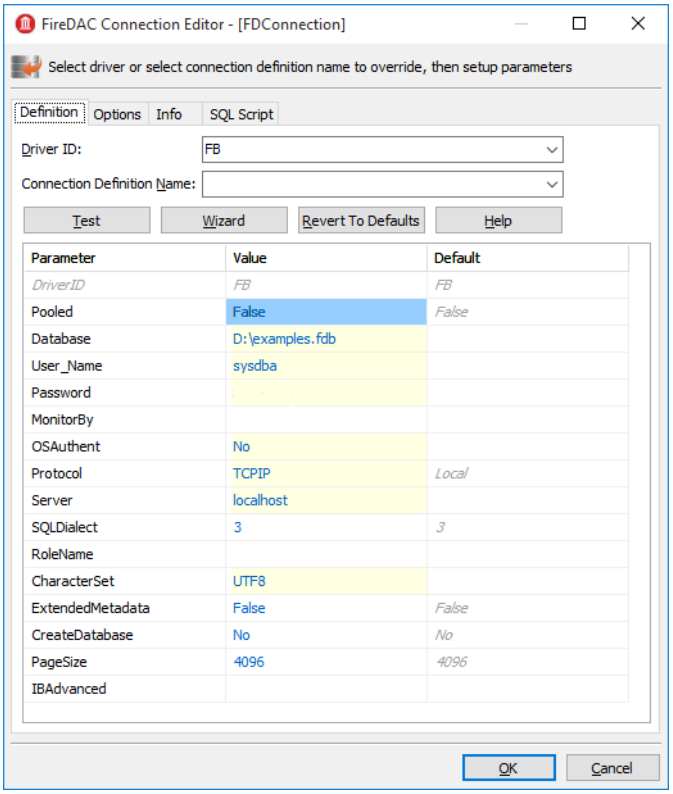

Рис. 2.1. Редактор свойств TFDConnection

Таблица 2.1. Основные свойства компонента TFDConnection

Свойство	Назначение
`Pooled`	Используется ли пул соединений.
`Database`	Путь к базе данных или её псевдоним, определённый в файле конфигурации `aliases.conf` (или `databases.conf`) сервера Firebird.
`User_Name`	Имя пользователя.
`Password`	Пароль.
`OSAuthent`	Используется ли аутентификация средствами операционной системы.
`Protocol`	Протокол соединения. Допускаются следующие значения: • Local — локальный протокол; • NetBEUI — именованные каналы, WNET; • SPX — протокол Novell IPX/SPX (не поддерживается в современных версиях); • TCPIP — TCP/IP.

Свойство	Назначение
Server	Имя сервера или его IP адрес. Если сервер работает на нестандартном порту, то необходимо также указать порт через слэш, например localhost/3051.
SQLDialect	Диалект. Должен совпадать с диалектом базы данных.
RoleName	Имя роли.
CharacterSet	Имя набора символов соединения.
Дополнительные свойства:	
Connected	Управление подсоединением к БД, или проверка состояния соединения. Это свойство должно быть выставлено в True для работы мастеров других компонентов FireDac. Если ваше приложение должно запрашивать данные для авторизации, то важно не забыть сбросить это свойство в False перед компиляцией вашего приложения.
LoginPrompt	Запрашивать ли имя пользователя и пароль при попытке соединения.
Transaction	Компонент TFDTransaction, который будет использоваться в качестве умолчательного для выполнения различных запросов. Если это свойство не назначено явно, TFDConnection создаст себе экземпляр TFDTransaction самостоятельно, его параметры можно указать в свойстве TxOptions.
UpdateTransaction	Компонент TFDTransaction, который будет использоваться в качестве умолчательного для одноимённых свойств компонентов TFDQuery. Если это свойство не назначено явно, будет использовано значение из свойства Transaction этого соединения.

Параметры подключения в конфигурационном файле

Поскольку параметры подключения, за исключением имени пользователя и пароля, обычно не изменяются в процессе эксплуатации приложения, мы будем считывать их из файла конфигурации.

```
// считываем параметры подключения
xIniFile := TIniFile.Create(xAppPath + 'config.ini');
try
  xIniFile.ReadSectionValues('connection', FDConnection.Params);
finally
  xIniFile.Free;
end;
```

Типичный файл конфигурации

Обычно файл конфигурации config.ini содержит примерно следующие строки:

```
[connection]
DriverID=FB
Protocol=TCPIP
Server=localhost/3051
Database=examples
OSAuthent=No
RoleName=
CharacterSet=UTF8
```

Содержимое секции connection можно получить, скопировав содержимое свойства Params компонента TFDConnection после работы мастера.

> **Примечание**
>
> На самом деле общие настройки обычно находятся в `%AppData%\Manufacture\AppName` и сохраняются туда инсталлятором приложения. Однако при разработке удобно чтобы файл настроек лежал где-нибудь поближе, например, в папке с приложением.
>
> Учтите, что если ваше приложение будет устанавливаться в папку `Program Files` и файл настройки будет лежать там же, то либо этот файл будет виртуализироваться в `Program Data`, либо будут проблемы с его модификацией и последующим чтением новых настроек.

Подключение к базе данных

Для подключения к базе данных необходимо изменить свойство `Connected` компонента `TFDConnection` в значение `True` или вызвать метод `Open`. В последний метод можно передать имя пользователя и пароль в качестве параметров.

Небольшая модификация

В нашем приложении мы заменим стандартный диалог соединения с базой данных. Дадим возможность ошибиться при вводе регистрационной информации не более трёх раз, после чего приложение будет закрыто. Для этого напишем следующий код в обработчике события `OnCreate` главного датамодуля.

```
// делаем максимум 3 попытки входа в систему, потом закрываем приложение
xLoginCount := 0;
xLoginPromptDlg := TLoginPromptForm.Create(Self);
while (xLoginCount < MAX_LOGIN_COUNT) and
      (not FDConnection.Connected) do
begin
  try
    if xLoginPromptDlg.ShowModal = mrOK then
      FDConnection.Open(
        xLoginPromptDlg.UserName, xLoginPromptDlg.Password)
    else
      xLoginCount := MAX_LOGIN_COUNT;
  except
    on E: Exception do
    begin
```

```
      Inc(xLoginCount);
      Application.ShowException(E);
    end
  end;
end;
xLoginPromptDlg.Free;

if not FDConnection.Connected then
  Halt;
```

Работа с транзакциями

Клиент Firebird допускает выполнение любых действий только в контексте транзакции, поэтому если вы смогли получить доступ к данным без явного вызова `TFDTransaction.StartTransaction`, то значит где-то в недрах FireDac этот вызов произошёл автоматически. Настоятельно рекомендуем избегать данной практики. Чтобы приложения работали корректно с базами данных, рекомендуется управлять транзакциями вручную, что обозначает старт, фиксацию и откат транзакций с помомщью явных вызовов соответсвующих функций.

Компонент `TFDTransaction` предназначен для явной работы с транзакциями.

Компонент TFDTransaction

`TFDTransaction` имеют три метода для явного управления транзакциями: `StartTransaction`, `Commit` и `Rollback`. В следующей таблице приведены свойства, доступные для настройки этого компонента.

Таблица 2.2. Основные свойства компонента TFDTransaction

Свойство	Назначение
`Connection`	Связь с компонентом FDConnection.
`Options.AutoCommit`	Управляет автоматическим стартом и завершением транзакции. Значение по умолчанию `True`. См. Примечание (1) ниже для получения дополнительной информации о поведении, когда параметр Autocommit имеет значение True.
`Options.AutoStart`	Управляет автоматическим запуском транзакции. По умолчанию `True`.
`Options.AutoStop`	Управляет автоматическим завершением транзакции. По умолчанию `True`.
`Options.DisconnectAction`	Действие, которое будет выполнено при закрытии соединения, если транзакция активна. Значение по умолчанию `xdCommit`. Подробнее о других параметрах см. Примечание (2) ниже.
`Options.EnableNested`	Управляет вложенными транзакциями. Значение по умолчанию `True`. Firebird не поддерживает вложенные

Свойство	Назначение
	транзакции как таковые, но FireDac может имитировать их с помощью точек сохранения. Подробнее см. Примечание (3) ниже.
`Options.Isolation`	Определяет уровень изолированности транзакции. Это самое важное свойство транзакции. Значение по умолчанию `xiReadCommited`. Firebird поддерживает слеюующие варианты: xiSnapshot и xiUnspecified; также xiSerializable, в некоторой степени. Подробнее о доступных уровнях изоляции см. Примечание (4) ниже.
`Options.Params`	Специфичные для Firebird параметры транзакции, которые могут применяться для уточнения параметров транзакции, переопределяя атрибуты, применяемые стандартной реализацией выбранного уровня изоляции. В настоящее время используется только для Firebird и Interbase. Описание аттрибутов и действительных их комбинайция см. в Приимечане (5) ниже.
`Options.ReadOnly`	Указывает является ли транзакция только для чтения. По умолчанию `False`. Если установлено в `True`, то любые изменения в рамках текущей транзакции невозможны, в Firebird в этом случае отсутствует значение read в параметрах транзакции. Установка этого свойства в `True` позволяет СУБД оптимизировать использование ресурсов.

Замечание 1: AutoCommit=True

Если значение свойства `AutoCommit` уставновлено а True, то FireDAC ведёт себя следующим образом:

- Запускается транзакция (если требуется) перед выполнением каждой SQL команды и завершает транзакцию после завершения выполнения SQL команды.
- Если команда выполнена успешно, то транзакция будет заврешена как COMMIT, в противном случае — будет завершена как ROLLBACK.
- Если приложение вызывает метод StartTransaction, то автоматическое управление транзакциями будет отключено, до тех пор, пока транзакция не завершится как Commit или Rollback.

Замечание 2: DisconnectAction

Возможны следующие варианты значений:

- `xdNone` — ничего не будет сделано. Действие будет отдано на откуп СУБД;
- `xdCommit` — подтверждение транзакции;
- `xdRollback` — откат транзакции.

В других компонентах доступа для `DisconnectAction` значение по умолчанию равно `xdRollback`, поэтому необходимо выставлять это свойство вручную в то значение, которое действительно требуется.

Замечание 3: EnableNested

Когда транзакция активна, то следующий вызов `StartTransaction` создаст вложенную транзакцию. FireDAC эмулирует вложенные транзакции, используя точки сохранения. Чтобы отключить вложенные транзакции, установите `EnableNested` в `False`, и последующий вызов `StartTransaction` вызовет исключение.

Замечание 4: Isolation

FireBird имеет три уровня изолированности: READ COMMITTED, SNAPSHOT («concurrency») и SNAPSHOT TABLE STABILITY («consistency», редко используемый). FireDac поддерживает некоторые, но не все конфигурации для READ COMMITTED и SNAPSHOT. Кроме того, он частично использует третий уровень для эмуляции изоляции SERIALIZABLE, которую Firebird не поддерживает.

- `xiReadCommited` — уровень изолированности READ COMMITED. В Firebird такая транзакция стартует с параметрами `read write read_committed rec_version nowait`;
- `xiSnapshot` — уровень изолированности SNAPSHOT. В Firebird такая транзакция стартует с параметрами `read write concurrency wait`;
- `xiUnspecified` — используется уровень изоляции по умолчанию для вашей СУБД (в Firebird это SNAPSHOT, т.е. с параметрами `read write concurrency wait`);
- `xiSerializable` — уровень изолированности SERIALIZABLE. На самом деле в Firebird не существует транзакции с данным уровнем изолированности, но он эмулируется запуском транзакции с параметрами `read write consistency wait`.

Другими параметрами, не поддерживаемыми Firebird, являются:

- `xiDirtyRead` — этого уровня изолированности в Firebird не существует поэтому вместо него будет использован READ COMMITED;
- `xiRepeatableRead` — этого уровня изолированности в Firebird не существует поэтому вместо него будет использован SNAPSHOT.

Замечание 5: Специфичные для Firebird атрибуты транзакций

Атрибуты которые можно настроить в Options.Params:

- `read write`, режим чтения по умолчанию для всех вариантов options.isolation — см. замечание (4) выше. Установить `write` в off если хотите read-only режим. Кром того, вы можете установить `Options.ReadOnly` в True для достижения того же эффекта. Не существует «write-only» транзакций.
- `read_committed`, `concurrency` и `consistency` — уровни изолированности.
- `wait` и `nowait` — это параметры разрешения конфликтов, определяющие, должна ли транзакция ждать разрешения конфликта
- `rec_version` и `no rec_version` — опции которые применимы только к READ COMMITTED транзакциям. По умолчанию `rec_version` даёт этой транзакции читать последнюю зафиксированную версию записи и перезаписывать её, если идентификатор транзакции последней commited версии является более новой (более высокой), чем идентификатор этой транзакции. Параметр `no rec_version` блокирует эту транзакцию от чтения последней commited версии, если любая другая транзакция ожидает обновления.

Несколько транзакций

В отличие от других СУБД в Firebird и Interbase разрешено использовать сколько угодно компонентов `TFDTransaction` привязанных к одному соединению. В нашем приложении мы будем использовать по одной читающей и одной пишущей транзакции на каждый справочник/журнал.

Мы не будем полагаться на автоматический старт и завершение транзакций, а потому во всех транзакциях установим свойства следующим образом `Options.AutoCommit = False`, `Options.AutoStart = False` и `Options.AutoStop = False`.

Датасеты

Работать с данными в FireDac можно при помощи компонент `TFDQuery`, `TFDTable`, `TFDStoredProc`, `TFDCommand`, но `TFDCommand` не является датасетом.

`TFDQuery`, `TFDTable` и `TFDStoredProc` унаследованы от `TFDRdbmsDataSet`. Помимо наборов данных для работы непосредственно с базой данных, в FireDac существует также компонент `TFDMemTable`, который предназначен для работы с набором данных в памяти, является аналогом `TClientDataSet`.

Основным компонентом для работы с наборами данных является `TFDQuery`. Возможностей этого компонента хватает практически для любых целей. Компоненты `TFDTable` и `TFDStoredProc` всего лишь модификации, либо чуть расширенные, либо усеченные. Мы не будем их рассматривать и применять в нашем приложении. При желании вы можете ознакомиться с ними в документации по FireDac.

Назначение компонента — буферизация записей, выбираемых оператором SELECT, для представления этих данных в Grid, а также для обеспечения "редактируемости" записи (текущей в буфере (гриде)). В отличие от компонента `IBX.TIBDataSet` компонент `TFDQuery` не содержит свойств `RefreshSQL`, `InsertSQL`, `UpdateSQL` и `DeleteSQL`. Вместо этого «редактируемость» обеспечивается компонентом `TFDUpdateSQL`, который устанавливается в свойство `UpdateObject`.

> **Свойство RequestLive**
>
> В ряде случаев можно сделать компонент `TFDQuery` редактируемым без установки свойства `UpdateObject` и прописывания запросов Insert/Update/Delete, просто установив свойство `UpdateOptions.RequestLive = True`, при этом модифицирующие запросы будут сгенерированы автоматически. Однако такой подход имеет множество ограничений на основной SELECT запрос, поэтому не стоит полагаться на него.

Компонент TFDQuery

Таблица 2.3. Основные свойства компонента TFDQuery

Свойство	Назначение
`Connection`	Связь с компонентом `FDConnection`.
`MasterSource`	Ссылка на Master-источник данных (`TDataSource`) для `FDQuery`, используемого в качестве Detail.

Свойство	Назначение
`Transaction`	Транзакция, в рамках которой будет выполняться запрос, прописанный в свойстве SQL. Если свойство не указано будет использоваться транзакция по умолчанию для подключения.
`UpdateObject`	Связь с компонентом `FDUpdateSQL`, который обеспечивает «редактируемость» набора данных, когда SELECT запрос не отвечает требованиям для автоматического формирования модифицирующих запросов при установке `UpdateOptions.RequestLive = True`.
`UpdateTransaction`	Транзакция, в рамках которой будут выполняться модифицирующие запросы. Если свойство не указано, будет использована транзакция из свойства `Transaction`.
`UpdateOptions.CheckRequired`	Если свойство `CheckRequired` установлено в `True`, то FireDac контролирует свойство `Required` соответвующих полей, т.е. полей с ограничением NOT NULL. По умолчанию установлено в `True`. Если `CheckRequired=True` и в поле имеющее свойство `Required=True` не присвоено значение, то при вызове метода `Post` будет возбуждено исключение. Это может быть нежелательно в том случае, если значение этого поля может быть присвоено позже в BEFORE триггерах.
`UpdateOptions.EnableDelete`	Определяет, позволяется ли удаление записи из набора данных. Если `EnableDelete=False`, то при вызове метода Delete будет возбуждено исключение.
`UpdateOptions.EnableInsert`	Определяет, позволяется ли вставка записи в набор данных. Если `EnableInsert=False`, то при вызове метода `Insert`/`Append` будет возбуждено исключение.
`UpdateOptions.EnableUpdate`	Определяет, позволяется ли изменение записи в наборе данных. Если `EnableUpdate=False`, то при вызове метода `Edit` будет возбуждено исключение.
`UpdateOptions.FetchGeneratorsPoint`	Управляет моментом получения следующего значения генератора указанного в свойстве `UpdateOptions.GeneratorName` или свойстве `GeneratorName` автоинкрементного поля `AutoGenerateValue = arAutoInc`. По умолчанию используется gpDeferred, что обозначает что следующее значение генератора извлекается до того, как в базу данных будет отправлена новая запись, то есть во время выполнения `Post` или `ApplyUpdates`. Полный набор возможных значений см. В примечании (1) ниже.
`UpdateOptions.GeneratorName`	Имя генератора для извлечения следующего значения автоинкрементного поля.

Свойство	Назначение
`UpdateOptions.ReadOnly`	Указывает, является ли набор данных только для чтения. По умолчанию `False`. Если значение этого свойства установлено в `True`, то значения свойств `EnableDelete`, `EnableInsert` и `EnableUpdate` будут автоматически выставлены в `False`.
`UpdateOptions.RequestLive`	Установка `RequestLive` в `True` делает запрос «живым», т.е. редактируемым, если это возможно. При этом запросы Insert/Update/Delete будут сгенерированы автоматически. Эта опция накладывает множество ограничений на SELECT запрос, введена для обратной совместимости с BDE и не рекомендуется.
`UpdateOptions.UpdateMode`	Управляет, как проверяется была ли запись изменена. Это свойство позволяет контролировать возможную перезапись обновлений в случаях, когда один пользователь выполняет редактирование записи "долго", а другой пользователь одновременно редактирует одну и ту же запись, и завершает обновление раньше. Значение по умолчанию - upWhereKeyOnly. Информацию о доступных режимах см. В примечании (2) ниже.
`CachedUpdates`	Определяет, будет ли набор данных кэшировать изменения без немедленного внесения их в базу данных. Если это свойство установлено в значение `True`, то любые изменения (`Insert`/`Post`, `Update`/`Post`, `Delete`) вносятся в базу данных не сразу, а сохраняется в специальном журнале. Приложение должно явно применить изменения, вызвав метод `ApplyUpdates`. В этом случае все изменения будут выполнены в течение малого промежутка времени и в одной короткой транзакции. По умолчанию значение этого свойства `False`.
`SQL`	Содержит SQL запрос. Если это свойство содержит SELECT запрос, то его необходимо выполнять методом `Open`. В противном случае необходимо использовать методы `Execute` или `ExecSQL`.

Замечание 1: UpdateOptions.FetchGeneratorPoint

Свойство UpdateOptions.FetchGeneratorPoint может принимать следующие значения:

- `gpNone` — значение генератора не извлекается;
- `gpImmediate` — следующее значение генератора извлекается сразу после вызова метода `Insert` или `Append`;
- `gpDeffered` — следующее значение генератора извлекается до публикации новой записи в базе данных, т.е. во время выполнения методов `Post` или `ApplyUpdates`.

Замечание 2: UpdateOptions.UpdateMode

Пользователь во время длительного сеанса редактирования может не знать, что запись была обновлена один или несколько раз в других сеанса редактирования. Это может привести к тому что его изменения перезапищут чужие обновления. Свойство UpdateOptions.UpdateMode позволяет выбирать поведение, чтобы уменьшить или избежать этого риска:

- `upWhereAll` — проверка на существование записи по первичному ключу + проверка всех столбцов на старые значения. Например

```
update table
set ...
where pkfield = :old_ pkfield
  and client_name = :old_client_name
  and info = :old_info
...
```

  То есть, в данном случае запрос поменяет информацию в записи только в том случае, если запись до нас никто не успел изменить. Особенно это важно, если существуют взаимозависимости между значениями столбцов — например, минимальная и максимальная зарплата, и т.п.

- `upWhereCahnged` — проверка записи на существование по первичному ключу + плюс проверка на старые значения только изменяемых столбцов.

```
update table
set ...
where pkfield = :old_ pkfield
  and client_name = :old_client_name
  and info = :old_info
...
```

- `upWhereKeyOnly` (по умолчанию) — проверка записи на существование по первичному ключу.

Последняя проверка соответствует генерируемому автоматически для `UpdateSQL` запросу. Поэтому, при возможных конфликтах обновлений в многопользовательской среде необходимо дописывать условия к where самостоятельно. И, разумеется, также необходимо при реализации аналога `upWhereChanged` удалять лишние изменения столбцов в `update table set ...` - то есть, оставлять в перечне set только действительно изменённые столбцы, иначе запрос перепишет чужие обновления этой записи. Как вы понимаете, это означает необходимость динамического конструирования запроса `UpdateSQL`.

Если вы хотите задать настройки обнаружения конфликтов обновления индивидуально для каждого поля, то вы можете воспользоваться свойством `ProviderFlags` для каждого поля.

Компонент TFDUpdateSQL

Компонент `TFDUpdateSQL` позволяет переопределять SQL команды, сгенерированные для автоматического обновления набора данных. Он может быть использован для внесения обновлений в компоненты `TFDQuery`, `TFDTable` и `TFDStoredProc`. Использование `TFDUpdateSQL` является необязательным для компонентов `TFDQuery` и `TFDTable`, потому что эти компоненты способны автоматически генерировать команды для публикации обновлений из набора данных в СУБД. Использование `TFDUpdateSQL` является обязательным для возможности обновления набора данных `TFDStoredProc`. Рекомендуем применять его всегда,

даже для самых простых случаев, чтобы получать полный контроль над тем какие запросы выполняются в вашем приложении.

Свойства TFDUpdateSQL

Для того чтобы указать SQL команды на этапе проектирования, используйте редактор `TFDUpdateSQL` времени проектирования, который вызывается двойным щелчком по компоненту.

> **Важно**
>
> Для работы многих редакторов времени проектирования FireDac требуется, чтобы было активно подключение к базе данных (`TFDConnection.Connected = True`) и транзакция находилась в режиме автостарта (`TFDTransaction.Options.AutoStart = True`). Но такие настройки могут мешать при работе приложения. Например, пользователь должен входить в программу под своим логином, а `TFDConnection` подключается к базе данных под SYSDBA. Поэтому после каждого использования редакторов времени проектирования рекомендуем проверять свойство `TFDConnection.Connected` и сбрасывать его. Кроме того, вам придётся включать и выключать автостарт транзакции предназначенной только для чтения.

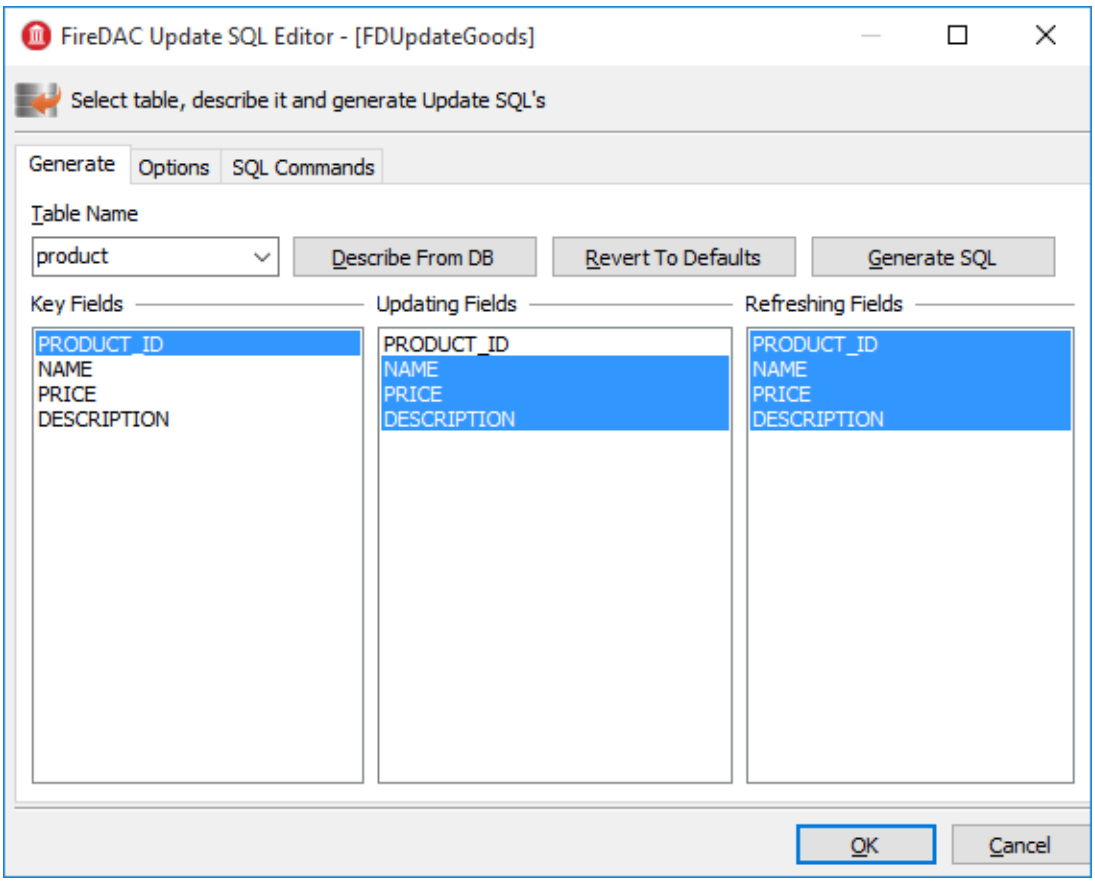

Рис. 2.2. Настройка TFDUpdateSQL. Генерация запросов.

На закладке Generate вы можете упростить себе задачу по написанию Insert/Update/Delete/Refresh запросов. Для этого выберете таблицу для обновления, её ключевые поля, поля для обновления, и поля которые будут перечитаны после обновления, и нажмите на кнопку «Generate SQL». После чего запросы будут сгенерированы автоматически, и вы перейдёте на закладку «SQL Commands», где можете поправить каждый из запросов.

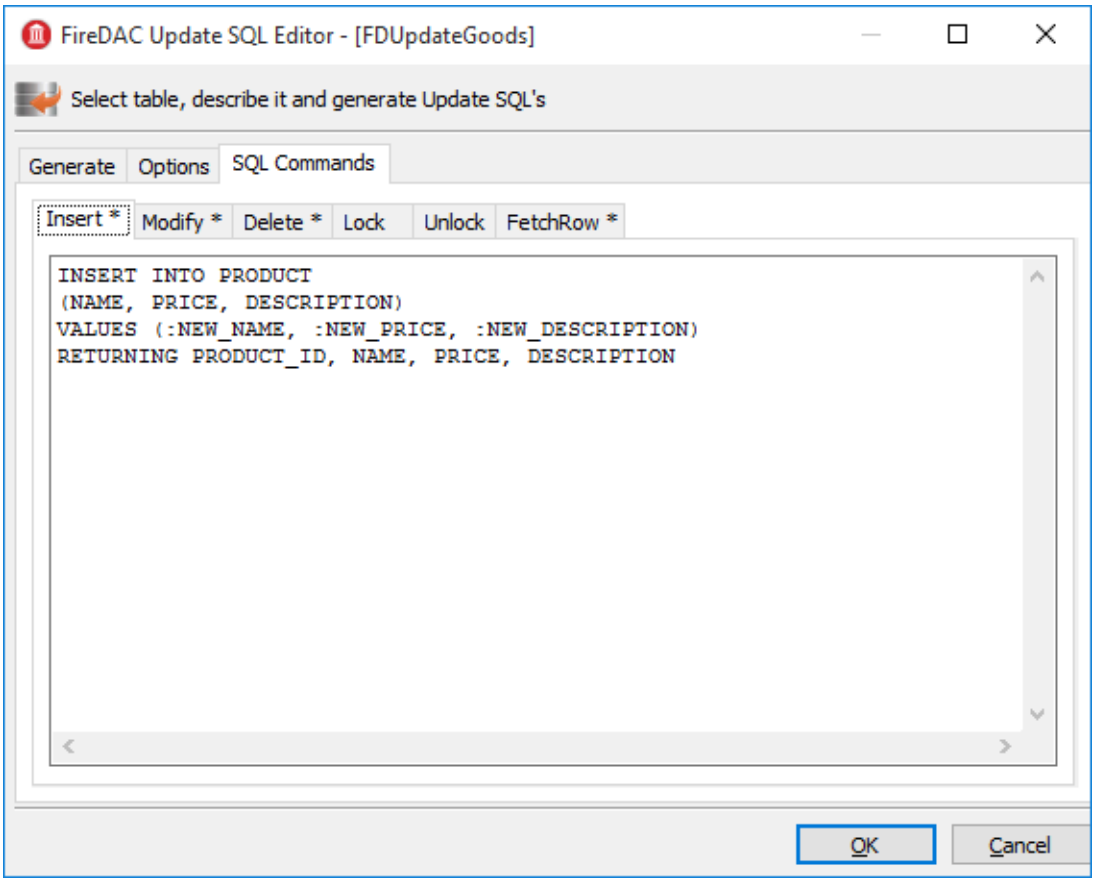

Рис. 2.3. Настройка TFDUpdateSQL. SQL команды.

> **Примечание**
>
> Поскольку product_id не включено в Updating Fields, оно отсутствует в генерируемом запросе insert. Предполагается, что этот столбец заполняется автоматически триггером (с генератором), или же этот это IDENTITY столбец (начиная с Firebird 3.0). При получении значения генератора для этого столбца с сервера, рекомендуется вручную добавить столбец PRODUCT_ID в предложение RETURNING оператора INSERT.

Закладка Options

На закладке Options находятся некоторые свойства, которые могут повлиять на генерацию запросов. Эти свойства не относятся к самому компоненту `TFDUpdateSQL`, а являются ссылками на свойства `UpdateOptions` набора данных, у которого указан текущий `TFDUpdateSQL` в свойстве `UpdateObject`. Так сделано исключительно ради удобства.

Таблица 2.4. Основные свойства компонента TFDUpdateSQL

Свойство	Назначение
`Connection`	Связь с компонентом `TFDConnection`.
`DeleteSQL`	SQL запрос для удаления записи.
`FetchRowSQL`	SQL запрос для возврата одной текущей (обновлённой, вставленной) записи. (`RefreshSQL`)
`InsertSQL`	SQL запрос для вставки записи.

Свойство	Назначение
LockSQL	SQL запрос для блокировки одной текущей записи. (FOR UPDATE WITH LOCK).
ModifySQL	SQL запрос для модификации записи.
UnlockSQL	SQL запрос для разблокировки текущей записи. В Firebird не применяется.

Как вы уже заметили, у компонента TFDUpdateSQL нет свойства Transaction. Это потому, что компонент не выполняет модифицирующие запросы непосредственно, а лишь заменяет автоматически сгенерированные запросы в наборе данных, который является предком TFDRdbmsDataSet.

Компонент TFDCommand

Компонент TFDCommand предназначен для выполнения SQL запросов. Он не является предком TDataSet, а потому удобен лишь для выполнения SQL запросов, не возвращающих набор данных.

Таблица 2.5. Основные свойства компонента TFDCommand

Свойство	Назначение
Connection	Связь с компонентом TFDConnection.
Transaction	Транзакция, в рамках которой будет выполняться SQL команда.
CommandKind	Тип команды. Описание типов команд приведено ниже.
CommandText	Текст SQL запроса.

Типы команд

Обычно тип команды определяется автоматически из текста SQL оператора. Следующие значения доступны для свойства TFDCommand.CommandKind для тех случаев, когда внутренний синтаксический анализатор не может сделать правильные или однозначные предположения, основанные только на тексте запроса:

- skUnknown – неизвестен. В этом случае тип команды будет определятся автоматически по тексту команды внутренним парсером;
- skStartTransaction – команда для старта транзакции;
- skCommit – команда завершения и подтверждения транзакции;
- skRollback – команда завершения и отката транзакции;
- skCreate – команда CREATE … для создания нового объекта метаданных;
- skAlter – команда ALTER … для модификации объекта метаданных;
- skDrop – команда DROP … для удаления объекта метаданных;

- `skSelect` – команда SELECT для выборки данных;
- `skSelectForLock` – команда SELECT … WITH LOCK для блокировки выбранных строк;
- `skInsert` – команда INSERT … для вставки новой записи;
- `skUpdate` – команда UPDATE … для модификации записей;
- `skDelete` – команда DELETE … для удаления записей;
- `skMerge` – команда MERGE INTO …
- `skExecute` – команда EXECUTE PROCEDURE или EXECUTE BLOCK;
- `skStoredProc` – вызов хранимой процедуры;
- `skStoredProcNoCrs` – вызов хранимой процедуры не возвращающей курсор;
- `skStoredProcWithCrs` – вызов хранимой процедуры возвращающей курсор.

Создание справочников

В нашем приложении мы создадим два справочника: справочник товаров и справочник заказчиков. Каждый из справочников представляет собой форму с сеткой `TDBGrid` и инструментальной панелью с кнопками. Бизнес-логика работы со справочником будет находится в отдельном DataModule, который содержит источник данных `TDataSource`, набором данных `TFDQuery`,читающую и пишущую транзакции `TFDTransaction`.

Рассмотрим создание справочников на примере справочника заказчиков.

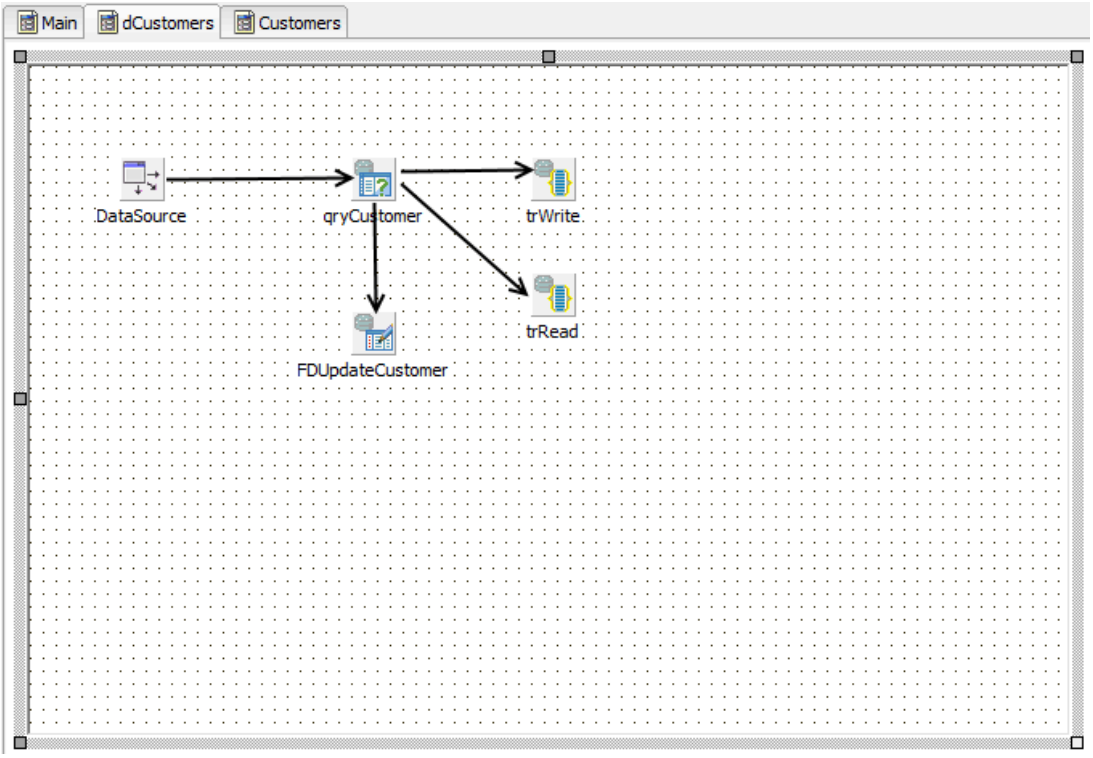

Рис. 2.4. Модуль dCustomers

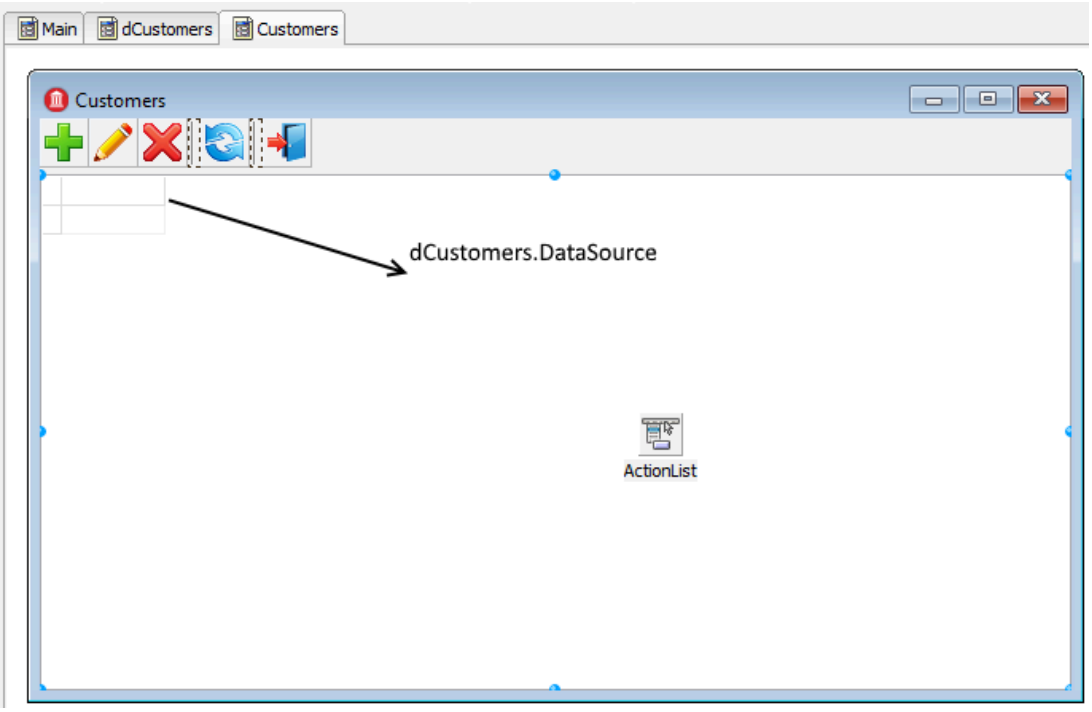

Рис. 2.5. Форма справочника Customers

> **Примечание**
>
> Компонент `DataSource` не виден, потому что находится не на форме, а в модуле `dCustomers`.

Разместим компонент `TFDQuery` в модуле dCustomres с именем `qryCustomers`. Этот набор данных будет указан в свойстве `DataSet` источника данных `DataSource`. В свойстве `Transaction` укажем ReadOnly транзакцию `trRead`, а в свойстве `UpdateTransaction` указываем транзакцию `trWrite`, в свойстве `Connection` — соединение расположенное в главном датамодуле. В свойстве SQL напишем следующий запрос:

```
SELECT
    customer_id,
    name,
    address,
    zipcode,
    phone
FROM
    customer
ORDER BY name
```

Read-only транзакция

Читающая транзакция `trRead` запускается при отображении формы справочника (событие OnActivate), а завершается при закрытии формы. Для отображения данных в гридах обычно используются режим изолированности READ COMMITED (`Options.Isolation = xiReadCommited`), т.к. он позволяет транзакции видеть чужие, committed изменения базы данных просто путём повторного выполнения запросов (перечитывания данных) без рестарта

транзакции. Поскольку эта транзакция используется только для чтения, установим свойство `Options.ReadOnly` в значение `True`. Таким образом, наша транзакция будет иметь параметры `read read_commited rec_version`. Транзакция с такими параметрами в Firebird может быть открытой сколь угодно долгое время (дни, недели, месяцы), без блокирования других транзакций или влияния на накопление мусора в базе данных (потому что на самом деле, на сервере такая транзакция стартует как committed). Значение свойства `Options.DisconnectAction` установим `xdCommit`, которое хорошо подходит для транзакции только для чтения. Таким образом у читающей транзакции будут следующие свойства:

```
Options.AutoStart = False
Options.AutoCommit = False
Options.AutoStop = False
Options.DisconnectAction = xdCommit
Options.Isolations = xiReadCommited
Options.ReadOnly = True
```

Примечание

Такую транзакцию нельзя использовать для отчётов (особенно если они используют несколько последовательных запросов), потому что транзакция с режимом изолированности READ COMMITED во время перечитывания данных будет видеть все новые committed-изменения.

Для отчётов рекомендуется использовать короткую транзакцию только для чтения с режимом изолированности SNAPSHOT (`Options.Isolation = xiSnapshot` и `Options.ReadOnly= True`). В данном примере работа с отчётами не рассматривается.

Read/Write транзакция

Пишущая транзакция `trWrite` должна быть максимально короткой для того, чтобы не удерживать Oldest Active Transaction, которая не даёт собрать мусор, что в свою очередь приводит к деградации производительности. Поскольку пишущая транзакция очень короткая мы можем использовать уровень изолированности SNAPSHOT. Таким образом, наша пишущая транзакция будет иметь параметры `Options.ReadOnly=False` и `Options.Isolation = xiSnapshot`. Для пишущих транзакций значение свойства `Options.DisconnectAction` по умолчанию не подходит, его необходимо выставить в значение `xdRollback`. Мы не будем полагаться на автоматический старт и завершение транзакции, а будем стартовать и завершать транзакцию явно. Таким образом, наша транзакция должна иметь следующие свойства:

```
Options.AutoStart = False
Options.AutoCommit = False
Options.AutoStop = False
Options.DisconnectAction = xdRollback
Options.Isolations = xiSnapshot
Options.ReadOnly = False
```

SNAPSHOT или READ COMMITTED

На самом деле необязательно устанавливать режим изолированности SNAPSHOT для простых INSERT/UPDATE/DELETE. Однако если у таблицы есть сложные триггеры, или вместо

простых запросов INSERT/UPDATE/DELETE вызывается хранимая процедура, то желательно использовать уровень изолированности SNAPSHOT.

Дело в том, что уровень изолированности READ COMMITED не обеспечивает атомарности оператора в пределах одной транзакции (statement read consistency). Таким образом, оператор SELECT может возвращать данные, которые попали в базу данных после начала выполнения запроса. В принципе режим изолированности SNAPSHOT можно рекомендовать почти всегда, если транзакция будет короткой.

Конфигурация справочника Заказчиков для редактирования

В этом разделе мы сконфигурируем некоторые свойства объектов `qryCustomer` и `FDUpdateCustomer`, чтобы сделать набор данных Заказчиков доступным для редактирования.

Настройки TFDUpdateSQL

Для возможности редактирования набора данных необходимо заполнить свойства `InsertSQL`, `ModifySQL`, `DeleteSQL` и `FetchRowSQL`. Эти свойства могут быть сгенерированы мастером, но после этого может потребоваться некоторая правка. Например вы можете дописать предложение RETURNING, удалить модификацию некоторых столбцов, или же вовсе заменить автоматически сгенерированный запрос на вызов хранимой процедуры.

InsertSQL:

```
INSERT INTO customer (customer_id,
                      name,
                      address,
                      zipcode,
                      phone)
VALUES (:new_customer_id,
        :new_name,
        :new_address,
        :new_zipcode,
        :new_phone)
```

ModifySQL:

```
UPDATE customer
SET name = :new_name,
    address = :new_address,
    zipcode = :new_zipcode,
    phone = :new_phone
WHERE (customer_id = :old_customer_id)
```

DeleteSQL:

```
DELETE FROM customer
WHERE (customer_id = :old_customer_id)
```

FetchRowSQL:

```
SELECT
    customer_id,
    name,
    address,
    zipcode,
    phone
FROM
    customer
WHERE customer_id = :old_customer_id
```

Получение значения генератора

В этом справочнике будем получать значение генератора перед вставкой записи в таблицу. Для этого необходимо установить значение свойств компонента `TFDQuery` в следующие значения `UpdateOptions.GeneratorName = GEN_CUSTOMER_ID` и `UpdateOptions.AutoIncFields = CUSTOMER_ID`. Есть другой способ, когда значение генератора (автоинкрементного поля) возвращается после выполнения INSERT запроса с помощью предложения RETURNING. Этот способ будет показан позже.

Реализация спраочника заказчикоы

Для добавления новой записи и редактирования существующей принято использовать модальные формы, по закрытию которых с результатом `mrOK` изменения вносятся в базу данных. Обычно для создания таких форм используются DBAware компоненты, которые позволяют отображать значения некоторого поля в текущей записи и немедленно вносить изменения в текущую запись набора данных в режимах `Insert`/`Edit`, т.е. до `Post`. Но перевести набор данных в режим Insert/Edit можно только стартовав пишущую транзакцию. Таким образом, если кто-то откроет форму для внесения новой записи и уйдёт на обед, не закрыв эту форму, у нас будет висеть активная транзакция до тех пор, пока сотрудник не вернётся с обеда и не закроет форму. Это в свою очередь приведёт к тому, что активная транзакция будет удерживать сборку мусора, что позже приведёт к снижению производительности. Эту проблему можно решить одним из двух способов:

1. Использовать режим `CachedUpdates`, что позволяет держать транзакцию активной только на очень короткий промежуток времени, а именно на время внесения изменений.

2. Отказаться от применения DBAware компонентов. Однако этот путь потребует от вас дополнительных усилий.

Мы покажем применение обоих способов. Для справочников гораздо удобнее использовать первый способ. Рассмотрим код редактирования записи поставщика

```
procedure TCustomerForm.actEditRecordExecute(Sender: TObject);
```

```
var
  xEditorForm: TEditCustomerForm;
begin
  xEditorForm := TEditCustomerForm.Create(Self);
  try
    xEditorForm.OnClose := CustomerEditorClose;
    xEditorForm.DataSource := Customers.DataSource;
    xEditorForm.Caption := 'Edit customer';
    Customers.Edit;
    xEditorForm.ShowModal;
  finally
    xEditorForm.Free;
  end;
end;
```

Свойство `Customers` инициализируется в событии `OnCreate`:

```
procedure TCustomerForm.FormCreate(Sender: TObject);
begin
  FCustomers := TDMCustomers.Create(Self);
  DBGrid.DataSource := Customers.DataSource;
end;
```

В методе Edit модуля dCustomers перед переводом набора данных в режим редактирования мы устанавливаем ему режим `CachedUpdates`:

```
procedure TdmCustomers.Edit;
begin
  qryCustomer.CachedUpdates := True;
  qryCustomer.Edit;
end;
```

Логика обработки редактирования и добавления записи производится в обработчике события `OnClose` для модальной формы редактирования:

```
procedure TCustomerForm.CustomerEditorClose(Sender: TObject;
  var Action: TCloseAction);
begin
  if TEditCustomerForm(Sender).ModalResult <> mrOK then
  begin
    Customers.Cancel;
    Action := caFree;
    Exit;
  end;

  try
    Customers.Post;
    Customers.Save;

    Action := caFree;
```

```
  except
    on E: Exception do
    begin
      Application.ShowException(E);
      // It does not close the window give the user correct the error
      Action := caNone;
    end;
  end;
end;
```

Кроме того для понимания внутрениих процессов потребуется привести коды методов `Cancel`, `Post` и `Save` модуля данных dCustomer.

```
procedure TdmCustomers.Cancel;
begin
  qryCustomer.Cancel;
  qryCustomer.CancelUpdates;
  qryCustomer.CachedUpdates := False;
end;

procedure TdmCustomers.Post;
begin
  qryCustomer.Post;
end;

procedure TdmCustomers.Save;
begin
  // We do everything in a short transaction
  // In CachedUpdates mode error does not stop running.
  // ApplyUpdates method returns the number of errors.
  // The error can be obtained from the property RowError
  try

    trWrite.StartTransaction;
    if (qryCustomer.ApplyUpdates = 0) then
    begin
      qryCustomer.CommitUpdates;
      trWrite.Commit;
    end
    else
      raise Exception.Create(qryCustomer.RowError.Message);
    qryCustomer.CachedUpdates := False;
  except
    on E: Exception do
    begin
      if trWrite.Active then
        trWrite.Rollback;
      raise;
    end;
  end;
end;
```

Из кода видно, что до тех пор, пока кнопка OK не нажата, пишущая транзакция не стартует вовсе. Таким образом, пишущая транзакция активна только на время переноса данных из буфера набора данных в базу данных. Поскольку мы копим в буфере не более одной записи, транзакция будет активна очень короткое время, что и требовалось.

Использование предложения RETURNING для получения автоинкрементных значений

Справочник товаров делается аналогично справочнику заказчиков. Однако в нём мы продемонстрируем другой способ получения автоинкрементных значений.

Основной запрос будет выглядеть следующим образом:

```
SELECT
    product_id,
    name,
    price,
    description
FROM product
ORDER BY name
```

Свойство компонента `TFDUpdateSQL.InsertSQL` будет содержать следующий запрос:

```
INSERT INTO PRODUCT
(NAME, PRICE, DESCRIPTION)
VALUES (:NEW_NAME, :NEW_PRICE, :NEW_DESCRIPTION)
RETURNING PRODUCT_ID
```

В этом запросе появилось предложение RETURNING, которое вернёт значение поля PRODUCT_ID после изменения его в BEFORE INSERT триггере. В этом случае не имеет смысла выставлять значение свойства `UpdateOptions.GeneratorName`. Кроме того, полю PRODUCT_ID необходимо выставить свойства `Required = False` и `ReadOnly = True`, поскольку значение этого свойства не вносится напрямую. В остальном всё примерно также как это организовано для справочника производителей.

Создание журналов

В нашем приложении будет один журнал «Счёт-фактуры». В отличие от справочников журналы содержат довольно большое количество записей и являются часто пополняемыми.

Счёт-фактура — состоит из заголовка, где описываются общие атрибуты (номер, дата, заказчик …), и строк счёт-фактуры со списком товаром, их количеством, стоимостью и т.д. Для таких документов удобно иметь два грида: в главном отображаются данные о шапке документа, а в детализирующем — список товаров. Таким образом, на форму документа нам потребуется поместить два компонента `TDBGrid`, к каждому из которых привязать свой `TDataSource`, которые в свою очередь будут привязаны к своим `TFDQuery`. В нашем случае набор данных с шапками документы будет называться `qryInvoice`, а со строками документа `qryInvoiceLine`.

Транзакции для журнала счёт-фактур

В свойстве `Transaction` обоих наборов данных укажем ReadOnly транзакцию `trRead`, которая находится в модуле данных `dmInvoice`. В свойстве `UpdateTransaction` указываем транзакцию `trWrite`, в свойстве `Connection` — соединение, расположенное в главном датамодуле.

Фильтрация данных

Большинство журналов содержат поле с датой создания документа. Чтобы уменьшить количество выбираемых данных обычно принято вводить такое понятие как рабочий период для того, чтобы уменьшить объём данных передаваемый на клиента. Рабочий период — это диапазон дат, внутри которого требуются рабочие документы. Поскольку приложение может содержать более одного журнала, то имеет смысл разместить переменные, содержащие дату начала и окончания рабочего периода, в глобальном датамодуле `dmMain`, который, так или иначе, используется всеми модулями, работающими с БД. При старте приложения рабочий период обычно инициализируется датой начала и окончания текущего квартала (могут быть другие варианты). В ходе работы приложения можно изменить рабочий период по желанию пользователя.

Конфигурация журнала

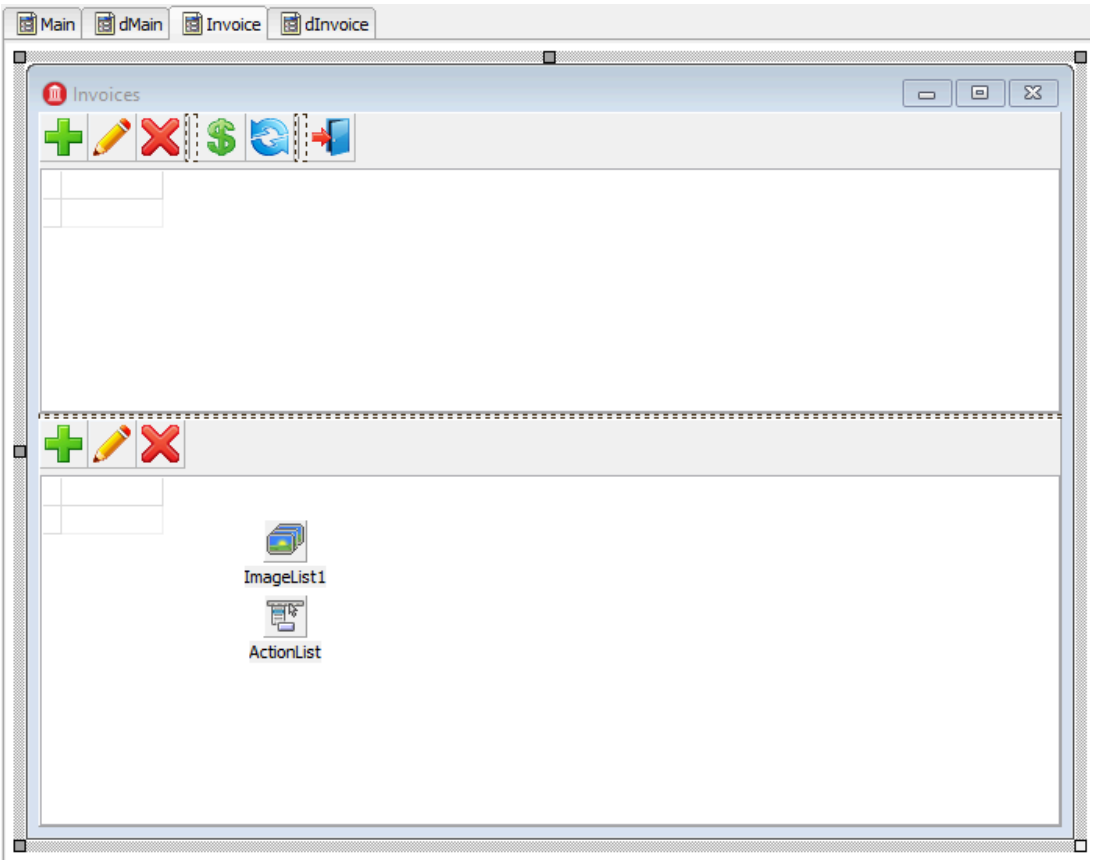

Рис. 2.6. Форма журнала Invoices

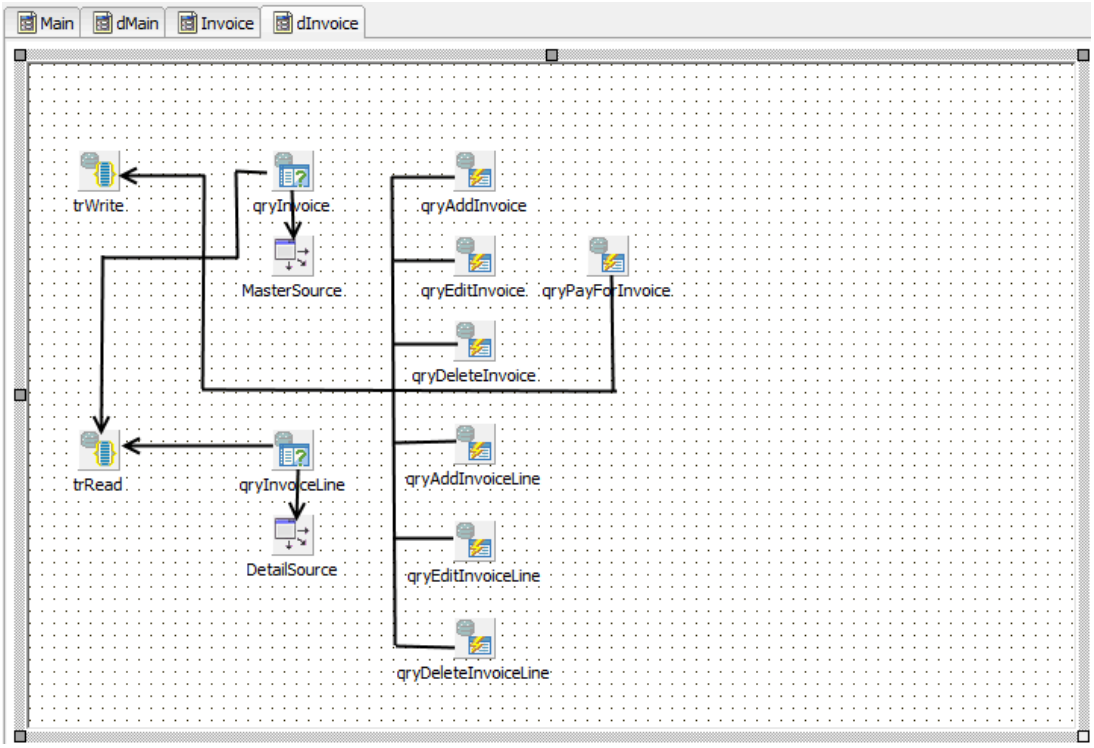

Рис. 2.7. Модуль данных dInvoices

Поскольку чаще всего требуются именно последние введённые документы, то имеет смысл сортировать их по дате в обратном порядке. С учётом вышесказанного, в свойстве SQL набора данных qryInvoice запрос будет выглядеть следующим образом:

```
SELECT
  invoice.invoice_id AS invoice_id,
  invoice.customer_id AS customer_id,
  customer.NAME AS customer_name,
  invoice.invoice_date AS invoice_date,
  invoice.total_sale AS total_sale,
  IIF(invoice.payed=1, 'Yes', 'No') AS payed
FROM
  invoice
  JOIN customer ON customer.customer_id = invoice.customer_id
WHERE invoice.invoice_date BETWEEN :date_begin AND :date_end
ORDER BY invoice.invoice_date DESC
```

При открытии этого набора данных необходимо будет инициализировать параметры запроса:

```
qryInvoice.ParamByName('date_begin').AsSqlTimeStamp := dmMain.BeginDateSt;
qryInvoice.ParamByName('date_end').AsSqlTimeStamp := dmMain.EndDateSt;
qryInvoice.Open;
```

Все операции над счёт-фактурой будем производить с помощью хранимых процедур, хотя в более простых случаях это можно делать и с помощью обычных запросов INSERT/UPDATE/DELETE.

Каждую хранимую процедуру будем выполнять как отдельный запрос в компонентах `TFDCommand`. Этот компонент не является предком `TFDRdbmsDataSet`, не буферизирует данные и возвращает максимум одну строку результата, поэтому его использование несёт меньше накладных расходов для запросов, не возвращающих данные. Поскольку наши хранимые процедуры выполняют модификацию данных, то свойство `Transaction` компонентов `TFDCommand` необходимо установить транзакцию `trWrite`.

> **Примечание**
>
> Хранимые процедуры вставки, редактирования и добавления записи можно также разместить в соответствующих свойствах компонента `TFDUpdateSQL`.

Операции журнала

Для работы с шапкой счёт-фактуры предусмотрено четыре операции: добавление, редактирование, удаление и установка признака «оплачено». Как только счёт-фактура оплачена, мы запрещаем любые её модификации, как в шапке, так и в строках. Это сделано на уровне хранимых процедур. Приведём тексты запросов для вызова хранимых процедур.

qryAddInvoice.CommandText:

```
EXECUTE PROCEDURE sp_add_invoice(
  NEXT VALUE FOR gen_invoice_id,
  :CUSTOMER_ID,
  :INVOICE_DATE
)
```

qryEditInvoice.CommandText:

```
EXECUTE PROCEDURE sp_edit_invoice(
  :INVOICE_ID,
  :CUSTOMER_ID,
  :INVOICE_DATE
)
```

qryDeleteInvoice.CommandText:

```
EXECUTE PROCEDURE sp_delete_invoice(:INVOICE_ID)
```

qryPayForInvoice.CommandText:

```
EXECUTE PROCEDURE sp_pay_for_inovice(:invoice_id)
```

Поскольку наши хранимые процедуры вызываются не из компонента `TFDUpdateSQL`, то после их выполнения необходимо вызвать `qryInvoice.Refresh` для обновления данных в гриде.

Вызов хранимых процедур, для которых не требуется ввод данных, производится следующим образом:

```
procedure TdmInvoice.DeleteInvoice;
begin
  // We do everything in a short transaction
  trWrite.StartTransaction;
  try
    qryDeleteInvoice.ParamByName('INVOICE_ID').AsInteger :=
      Invoice.INVOICE_ID.Value;
    qryDeleteInvoice.Execute;
    trWrite.Commit;
    qryInvoice.Refresh;
  except
    on E: Exception do
    begin
      if trWrite.Active then
        trWrite.Rollback;
      raise;
    end;
  end;
end;
```

Получение подтверждения

Перед выполнением некоторых операций необходимо переспросить об этом пользователя, например при удалении счёт-фактуры:

```
procedure TInvoiceForm.actDeleteInvoiceExecute(Sender: TObject);
begin
  if MessageDlg('Are you sure you want to delete an invoice?', mtConfirmation,
    [mbYes, mbNo], 0) = mrYes then
  begin
    Invoices.DeleteInvoice;
  end;
end;
```

Добавление и редактирование записей

Для добавления новой записи и редактирования существующей, как и в случае со справочниками мы будем использовать модальные формы. В данном случае мы не будем использовать DBAware компоненты. Ещё одна особенность — для выбора заказчика мы будем использовать компонент `TButtonedEdit`. Он будет отображать наименование текущего заказчика, а по нажатию кнопки вызывать модальную форму с гридом для выбора заказчика. Конечно, можно было бы воспользоваться чем-то вроде `TDBLookupCombobox`, но, во-первых заказчиков может быть очень много и пролистывать такой выпадающий список будет неудобно, во-вторых для поиска нужного заказчика одного названия может быть недостаточно.

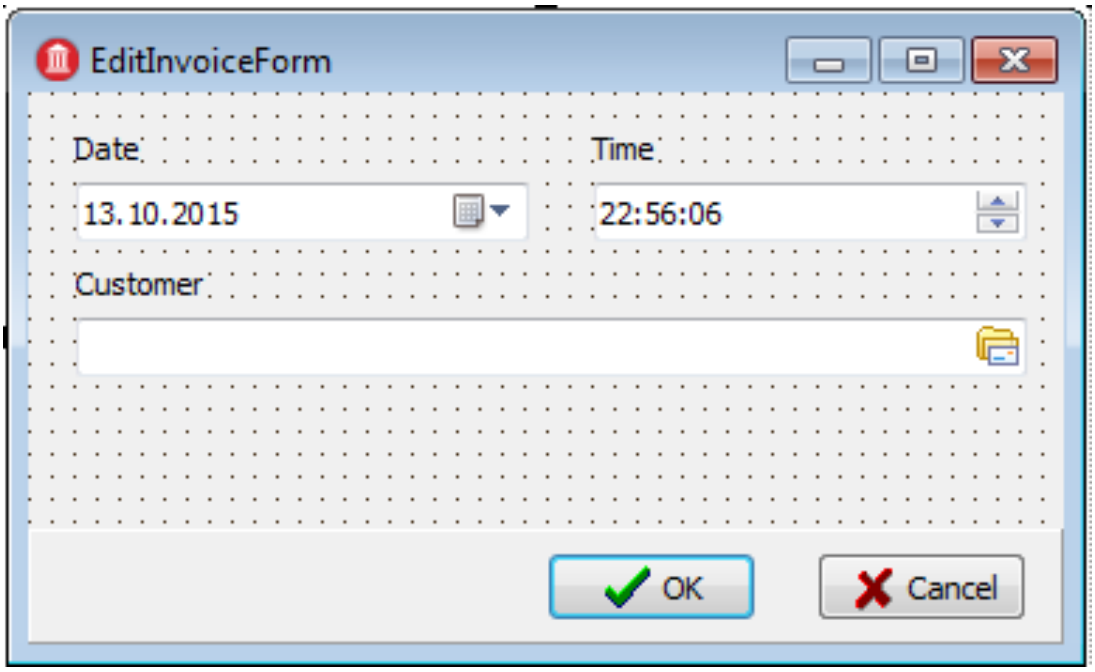

Рис. 2.8. Форма редактирования счёт-фактуры

В качестве модальной окна для выбора заказчика используем ту же форму, что была создана для ввода заказчиков. Код обработчика нажатия кнопки в компоненте `TButtonedEdit` будет выглядеть следующим образом:

```
procedure TEditInvoiceForm.edtCustomerRightButtonClick(Sender: TObject);
var
  xSelectForm: TCustomerForm;
begin
  xSelectForm := TCustomerForm.Create(Self);
  try
    xSelectForm.Visible := False;
    if xSelectForm.ShowModal = mrOK then
    begin
      FCustomerId := xSelectForm.Customers.Customer.CUSTOMER_ID.Value;
      edtCustomer.Text := xSelectForm.Customers.Customer.NAME.Value;
    end;
  finally
    xSelectForm.Free;
  end;
end;
```

Поскольку мы используем не DBAware компоненты, то при вызове формы редактирования нам будет необходимо инициализировать код заказчика и его наименование для отображения.

```
procedure TInvoiceForm.actEditInvoiceExecute(Sender: TObject);
var
  xEditorForm: TEditInvoiceForm;
begin
  xEditorForm := TEditInvoiceForm.Create(Self);
  try
    xEditorForm.OnClose := EditInvoiceEditorClose;
```

```
    xEditorForm.Caption := 'Edit invoice';

    xEditorForm.InvoiceId := Invoices.Invoice.INVOICE_ID.Value;
    xEditorForm.SetCustomer(
      Invoices.Invoice.CUSTOMER_ID.Value,
      Invoices.Invoice.CUSTOMER_NAME.Value);
    xEditorForm.InvoiceDate := Invoices.Invoice.INVOICE_DATE.AsDateTime;

    xEditorForm.ShowModal;
  finally
    xEditorForm.Free;
  end;
end;

procedure TEditInvoiceForm.SetCustomer(ACustomerId: Integer;
  const ACustomerName: string);
begin
  FCustomerId := ACustomerId;
  edtCustomer.Text := ACustomerName;
end;
```

Обработку добавления новой счёт-фактуры и редактирование существующей будем осуществлять в событии закрытия модальной формы, также, как это сделано для справочников. Бизнес логика добавления новой счёт-фактуры находится в модуле данных dmInvoices. Однако здесь мы уже не будем переводить набор данных в режим `CachedUpdates`, поскольку модификация производится с помощью хранимых процедур, и мы не используем DBAware компоненты.

```
procedure TInvoiceForm.actAddInvoiceExecute(Sender: TObject);
var
  xEditorForm: TEditInvoiceForm;
begin
  xEditorForm := TEditInvoiceForm.Create(Self);
  try
    xEditorForm.Caption := 'Add invoice';
    xEditorForm.OnClose := AddInvoiceEditorClose;

    xEditorForm.InvoiceDate := Now;

    xEditorForm.ShowModal;
  finally
    xEditorForm.Free;
  end;
end;

procedure TInvoiceForm.AddInvoiceEditorClose(Sender: TObject;
  var Action: TCloseAction);
var
  xEditorForm: TEditInvoiceForm;
begin
  xEditorForm := TEditInvoiceForm(Sender);

  if xEditorForm.ModalResult <> mrOK then
  begin
```

```
    Action := caFree;
    Exit;
  end;

  try
    Invoices.AddInvoice(xEditorForm.CustomerId, xEditorForm.InvoiceDate);

    Action := caFree;
  except
    on E: Exception do
    begin
      Application.ShowException(E);
      // It does not close the window give the user correct the error
      Action := caNone;
    end;
  end;
end;

procedure TdmInvoice.AddInvoice(ACustomerId: Integer; AInvoiceDate: TDateTime);
begin
  // We do everything in a short transaction
  trWrite.StartTransaction;
  try
    qryAddInvoice.ParamByName('CUSTOMER_ID').AsInteger := ACustomerId;
    qryAddInvoice.ParamByName('INVOICE_DATE').AsSqlTimeStamp :=
      DateTimeToSQLTimeStamp(AInvoiceDate);

    qryAddInvoice.Execute();

    trWrite.Commit;
    qryInvoice.Refresh;
  except
    on E: Exception do
    begin
      if trWrite.Active then
        trWrite.Rollback;

      raise;
    end;
  end;
end;
```

Позиции счёт фактуры

Теперь перейдём к позициям накладной. Набору данных `qryInvoiceLine` установим свойство `MasterSource = MasterSource`, который привязан к `qryInvoice`, а свойство `MasterFields = INVOICE_ID`. В свойстве SQL напишем следующий запрос:

```
SELECT
    invoice_line.invoice_line_id AS invoice_line_id,
    invoice_line.invoice_id AS invoice_id,
    invoice_line.product_id AS product_id,
```

```
    product.name AS productname,
    invoice_line.quantity AS quantity,
    invoice_line.sale_price AS sale_price,
    invoice_line.quantity * invoice_line.sale_price AS total
FROM
    invoice_line
JOIN product ON product.product_id = invoice_line.product_id
WHERE invoice_line.invoice_id = :invoice_id
```

Все модификации, как и в случае с шапкой счёт-фактуры, будем осуществлять с помощью хранимых процедур. Приведём тексты запросов для вызова хранимых процедур.

qryAddInvoiceLine.CommandText:

```
EXECUTE PROCEDURE sp_add_invoice_line(
  :invoice_id,
  :product_id,
  :quantity
)
```

qryEditInvoiceLine.CommandText:

```
EXECUTE PROCEDURE sp_edit_invoice_line(
  :invoice_line_id,
  :quantity
)
```

qryDeleteInvoiceLine.CommandText:

```
EXECUTE PROCEDURE sp_delete_invoice_line(
  :invoice_line_id
)
```

Форма для добавления новой записи и редактирования существующей, как и в случае с шапкой не будет использовать DBAware. Для выбора товара мы будем использовать компонент `TButtonedEdit`. Код обработчика нажатия кнопки в компоненте `TButtonedEdit` будет выглядеть следующим образом:

```
procedure TEditInvoiceLineForm.edtProductRightButtonClick(Sender: TObject);
var
  xSelectForm: TGoodsForm;
begin
  if FEditMode = emInvoiceLineEdit then
    Exit;

  xSelectForm := TGoodsForm.Create(Self);
  try
    xSelectForm.Visible := False;
    if xSelectForm.ShowModal = mrOK then
    begin
```

```
      FProductId := xSelectForm.Goods.Product.PRODUCT_ID.Value;
      edtProduct.Text := xSelectForm.Goods.Product.NAME.Value;
      edtPrice.Text := xSelectForm.Goods.Product.PRICE.AsString;
    end;
  finally
    xSelectForm.Free;
  end;
end;
```

Поскольку мы используем не DBAware компоненты, то при вызове формы редактирования нам будет необходимо инициализировать код товара, его наименование и стоимость для отображения.

```
procedure TInvoiceForm.actEditInvoiceLineExecute(Sender: TObject);
var
  xEditorForm: TEditInvoiceLineForm;
begin
  xEditorForm := TEditInvoiceLineForm.Create(Self);
  try
    xEditorForm.EditMode := emInvoiceLineEdit;
    xEditorForm.OnClose := EditInvoiceLineEditorClose;
    xEditorForm.Caption := 'Edit invoice line';

    xEditorForm.InvoiceLineId := Invoices.InvoiceLine.INVOICE_LINE_ID.Value;
    xEditorForm.SetProduct(
      Invoices.InvoiceLine.PRODUCT_ID.Value,
      Invoices.InvoiceLine.PRODUCTNAME.Value,
      Invoices.InvoiceLine.SALE_PRICE.AsCurrency);
    xEditorForm.Quantity := Invoices.InvoiceLine.QUANTITY.Value;

    xEditorForm.ShowModal;
  finally
    xEditorForm.Free;
  end;
end;

procedure TEditInvoiceLineForm.SetProduct(AProductId: Integer;
  AProductName: string; APrice: Currency);
begin
  FProductId := AProductId;
  edtProduct.Text := AProductName;
  edtPrice.Text := CurrToStr(APrice);
end;
```

Обработку добавления новой позиции и редактирование существующей будем производить в событии закрытия модальной формы.

```
procedure TInvoiceForm.actAddInvoiceLineExecute(Sender: TObject);
var
  xEditorForm: TEditInvoiceLineForm;
begin
  xEditorForm := TEditInvoiceLineForm.Create(Self);
  try
```

```
    xEditorForm.EditMode := emInvoiceLineAdd;
    xEditorForm.OnClose := AddInvoiceLineEditorClose;
    xEditorForm.Caption := 'Add invoice line';

    xEditorForm.Quantity := 1;
    xEditorForm.InvoiceId := Invoices.Invoice.INVOICE_ID.Value;
    xEditorForm.ShowModal;
  finally
    xEditorForm.Free;
  end;
end;

procedure TInvoiceForm.actEditInvoiceLineExecute(Sender: TObject);
var
  xEditorForm: TEditInvoiceLineForm;
begin
  xEditorForm := TEditInvoiceLineForm.Create(Self);
  try
    xEditorForm.EditMode := emInvoiceLineEdit;
    xEditorForm.OnClose := EditInvoiceLineEditorClose;
    xEditorForm.Caption := 'Edit invoice line';

    xEditorForm.InvoiceLineId := Invoices.InvoiceLine.INVOICE_LINE_ID.Value;
    xEditorForm.SetProduct(
      Invoices.InvoiceLine.PRODUCT_ID.Value,
      Invoices.InvoiceLine.PRODUCTNAME.Value,
      Invoices.InvoiceLine.SALE_PRICE.AsCurrency);
    xEditorForm.Quantity := Invoices.InvoiceLine.QUANTITY.Value;

    xEditorForm.ShowModal;
  finally
    xEditorForm.Free;
  end;
end;

procedure TInvoiceForm.AddInvoiceLineEditorClose(Sender: TObject;
  var Action: TCloseAction);
var
  xEditorForm: TEditInvoiceLineForm;
  xCustomerId: Integer;
begin
  xEditorForm := TEditInvoiceLineForm(Sender);

  if xEditorForm.ModalResult <> mrOK then
  begin
    Action := caFree;
    Exit;
  end;

  try
    Invoices.AddInvoiceLine(xEditorForm.ProductId, xEditorForm.Quantity);

    Action := caFree;
  except
    on E: Exception do
    begin
      Application.ShowException(E);
      // It does not close the window give the user correct the error
```

```
      Action := caNone;
    end;
  end;
end;

procedure TInvoiceForm.EditInvoiceLineEditorClose(Sender: TObject;
  var Action: TCloseAction);
var
  xCustomerId: Integer;
  xEditorForm: TEditInvoiceLineForm;
begin
  xEditorForm := TEditInvoiceLineForm(Sender);

  if xEditorForm.ModalResult <> mrOK then
  begin
    Action := caFree;
    Exit;
  end;

  try
    Invoices.EditInvoiceLine(xEditorForm.Quantity);

    Action := caFree;
  except
    on E: Exception do
    begin
      Application.ShowException(E);
      // It does not close the window give the user correct the error
      Action := caNone;
    end;
  end;
end;
```

Теперь приведём код процедур `AddInvoiceLine` и `EditInvoiceLine` модуля данных `dmInvoice`:

```
procedure TdmInvoice.AddInvoiceLine(AProductId: Integer; AQuantity: Integer);
begin
  // We do everything in a short transaction
  trWrite.StartTransaction;
  try
    qryAddInvoiceLine.ParamByName('INVOICE_ID').AsInteger :=
      Invoice.INVOICE_ID.Value;

    if AProductId = 0 then
      raise Exception.Create('Not selected product');

    qryAddInvoiceLine.ParamByName('PRODUCT_ID').AsInteger := AProductId;
    qryAddInvoiceLine.ParamByName('QUANTITY').AsInteger := AQuantity;

    qryAddInvoiceLine.Execute();

    trWrite.Commit;
    qryInvoice.Refresh;
```

```
    qryInvoiceLine.Refresh;

  except
    on E: Exception do
    begin
      if trWrite.Active then
        trWrite.Rollback;
      raise;
    end;
  end;
end;

procedure TdmInvoice.EditInvoiceLine(AQuantity: Integer);
begin
  // We do everything in a short transaction
  trWrite.StartTransaction;
  try
    qryEditInvoiceLine.ParamByName('INVOICE_LINE_ID').AsInteger :=
      InvoiceLine.INVOICE_LINE_ID.Value;
    qryEditInvoiceLine.ParamByName('QUANTITY').AsInteger := AQuantity;

    qryEditInvoiceLine.Execute();

    trWrite.Commit;
    qryInvoice.Refresh;
    qryInvoiceLine.Refresh;

  except
    on E: Exception do
    begin
      if trWrite.Active then
        trWrite.Rollback;
      raise;
    end;
  end;
end;
```

Результат

В итоге у нас получилось приложение которое выглядит следующим образом:

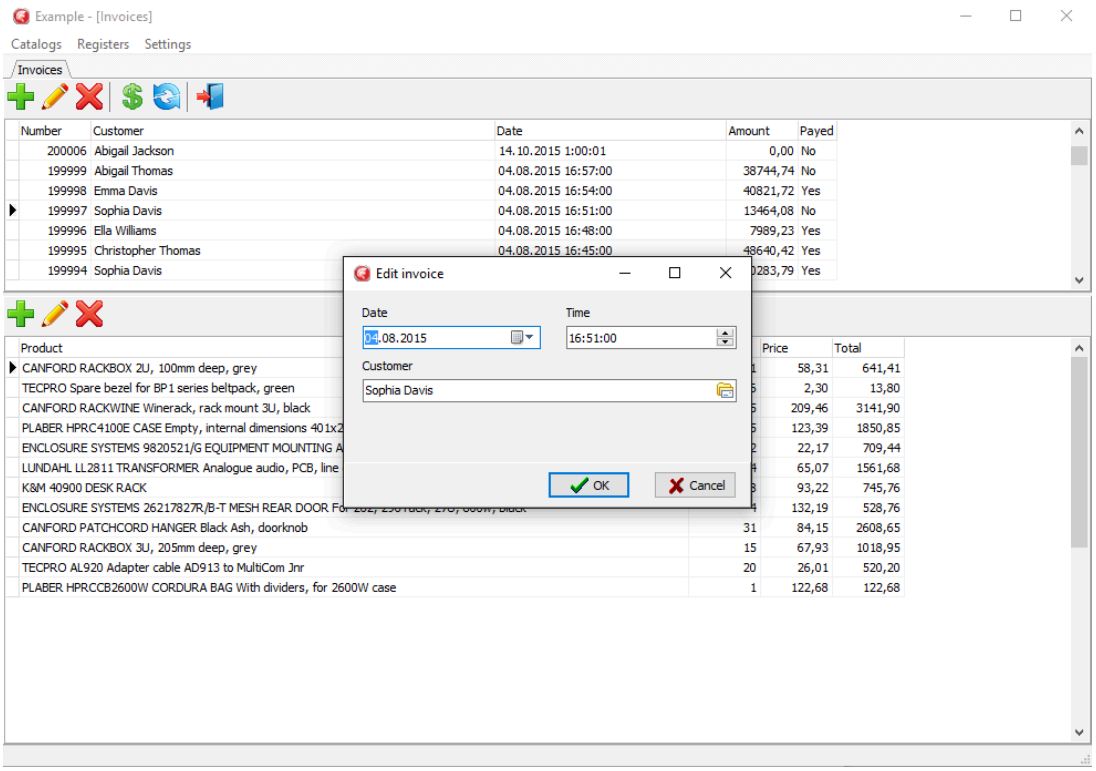

Рис. 2.9. Скриншот работающего приложения

Заключение

Для работы с СУБД Firebird в Delphi существует множество различных компонентов доступа (Interbase Express (IBX), FibPlus, UIB, UniDAC, IBDac, FireDac). FireDac является стандартным набором компонентов доступа к различным базам данных начиная с Delphi XE3.

Все запросы к базе данных происходят в рамках транзакции. Для корректной работы приложений с базой данных желательно управлять транзакциями вручную, то есть явно вызывать методы `StartTransaction`, `Commit` и `Rollback` компонента `TFDTransaction`. Вы можете использовать столько транзакций сколько требует логика вашего приложения. По возможности все транзакции должны быть короткими.

Если требуется длинная читающая транзакция для просмотра журналов или справочников, то такую транзакцию желательно стартовать с параметрами READ READ_COMMITED REC_VERSION. Вы можете стартовать одну такую транзакцию на все журналы/справочники или по одной транзакции на каждый журнал/справочник.

Пишущие транзакции должны быть максимально короткими. Если вы используете сложную логику при редактировании данных (изменение нескольких таблиц, расчёты хранимых агрегатов и др.), то необходимо использовать режим изолированности SNAPSHOT. Для того чтобы не удерживать транзакцию во время редактирования в формах редактирования, необходимо либо отказаться от DBAware компонентов, либо использовать режим `CachedUpdates`. Режим `CachedUpdates` позволяет держать транзакцию активной только на очень короткий промежуток времени, а именно на время внесения изменений.

При построение отчётных форм, особенно когда выполняется множество запросов, необходимо использовать транзакцию с режимом изолированности SNAPSHOT.

Для возможности редактирования набора данных необходимо использовать компонент `TFDUpdateSQL` и заполнить его свойства `InsertSQL`, `ModifySQL`, `DeleteSQL` и `FetchRowSQL`. Эти свойства могут быть сгенерированы мастером, но после этого может потребоваться некоторая правка.

Для работы с автоинкрементными первичными ключами существует 2 способа:

- Предварительное получение значения генератора. Для этого необходимо установить значение свойств компонента `TFDQuery UpdateOptions.GeneratorName` и `UpdateOptions.AutoIncFields`.
- Получение значения первичного ключа с помощью предложения RETURNING, которое необходимо дописать в запросе `InsertSQL`. В этом случае полю необходимо выставить свойства `Required = False` и `ReadOnly = True`, поскольку значение этого свойства не вносится напрямую.

Более сложную бизнес логику удобно реализовать используя хранимые процедуры. Хранимые процедуры, которые не возвращают данные, удобно выполнять с помощью компонента `TFDCommand`.

Исходные коды

Исходные коды примера приложения вы можете получить по ссылке https://github.com/sim1984/FireDacEx

Глава 3

Создание Windows Forms приложений с использованием Entity Framework

В данной главе будет описан процесс создания приложений для СУБД Firebird с использованием компонентов доступа Entity Framework и среды Visual Studio 2015.

ADO.NET Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL.

Способы взаимодействия с базой данных

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

- **Database first**: Entity Framework создаёт набор классов, которые отражают модель конкретной базы данных.
- **Model first**: сначала разработчик создаёт модель базы данных, по которой затем Entity Framework создаёт реальную базу данных на сервере.
- **Code first**: разработчик создаёт класс модели данных, которые будут храниться в БД, а затем Entity Framework по этой модели генерирует базу данных и её таблицы.

В своём приложении мы будем использовать подход Code First, однако вы без труда сможете использовать и другие подходы.

> **Примечание**
>
> На самом деле у нас уже есть база данных. Поэтому будем просто писать код который бы привёл к созданию нашей БД.

Подготовка Visual Studio 2015 для работы с Firebird

Для работы с Firebird вам необходимо установить:

- `FirebirdSql.Data.FirebirdClient.dll`
- `EntityFramework.Firebird.dll`
- DDEX Provider for Visual Studio

Установка первых двух не вызывает никаких сложностей. В настоящий момент они распространяются и устанавливаются в проект с помощью NuGet. А вот последняя библиотека, предназначенная для работы мастеров Visual Studio, устанавливается не так легко и может потратить у вас много сил и времени.

Добрые люди попытались автоматизировать процесс установки и включить установку всех компонентов в один дистрибутив (http://sourceforge.net/projects/firebird-4-8-0-ddex-installer/). Однако в ряде случаев вам может потребоваться ручная установка всех компонентов. В этом случае вам потребуется скачать:

- `FirebirdSql.Data.FirebirdClient-4.10.0.0.msi` (http://sourceforge.net/projects/firebird/files/firebird-net-provider/4.10.0.0/FirebirdSql.Data.FirebirdClient-4.10.0.0.msi/download)
- `EntityFramework.Firebird-4.10.0.0-NET45.7z` (http://sourceforge.net/projects/firebird/files/firebird-net-provider/4.10.0.0/EntityFramework.Firebird-4.10.0.0-NET45.7z/download)
- `DDEXProvider-3.0.2.0.7z` (http://sourceforge.net/projects/firebird/files/firebird-net-provider/DDEX%203.0.2/DDEXProvider-3.0.2.0.7z/download)
- `DDEXProvider-3.0.2.0-src.7z` (http://sourceforge.net/projects/firebird/files/firebird-net-provider/DDEX%203.0.2/DDEXProvider-3.0.2.0-src.7z/download)

Процесс установки

> **Важно**
>
> Поскольку процесс установки требует манипуляций с защищёнными директориями, вам потребуются права администратора.

Шаги

1. Устанавливаем `FirebirdSql.Data.FirebirdClient-4.10.0.0.msi`
2. Распаковываем `EntityFramework.Firebird-4.10.0.0-NET45.7z` в папку с установленным клиентом Firebird. У меня это папка `c:\Program Files (x86)\FirebirdClient\`
3. Необходимо установить сборки Firebird в GAC. Для удобство пописываем в `%PATH%` путь до утилиты gacutil для .NET Framework 4.5. У меня этот путь `c:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools\`
4. Запускаем командную строку cmd от имени администратора и переходим в директорию с установленным клиентом.

```
chdir "c:\Program Files (x86)\FirebirdClient"
```

5. Теперь проверяем что FirebirdSql.Data.FirebirdClient установлен в GAC. Для этого набираем команду

```
gacutil /l FirebirdSql.Data.FirebirdClient
```

```
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.0
с Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

В глобальном кэше сборок содержатся следующие сборки:
  FirebirdSql.Data.FirebirdClient, Version=4.10.0.0, Culture=neutral, PublicKeyToken=3750

Число элементов = 1
```

   Если FirebirdSql.Data.FirebirdClient не был установлен в GAC, то сделаем это с помощью команды

```
gacutil /i FirebirdSql.Data.FirebirdClient.dll
```

6. Теперь установим EntityFramework.Firebird в GAC

```
gacutil /i EntityFramework.Firebird.dll
```

7. Распаковываем `DDEXProvider-3.0.2.0.7z` в удобную директорию. Я распаковал её в `c:\Program Files (x86)\FirebirdDDEX\`

8. Туда же распаковываем `DDEXProvider-3.0.2.0-src.7z` содержимое поддиректории архива `/reg_files/VS2015`

> **Примечание**
>
> Забавно, но по какой-то причине этих файлов нет в предыдущем архиве со скомпилированными dll библиотеками, но они присутствуют в архиве с исходными кодами.

9. Открываем файл `FirebirdDDEXProvider64.reg` с помощью блокнота. Находим строчку, которая содержит `%path%` и меняем его на полный путь к файлу `FirebirdSql.VisualStudio.DataTools.dll`

```
"CodeBase"="c:\\Program Files (x86)\\FirebirdDDEX\\FirebirdSql.VisualStudio.DataTools.dll
```

10. Сохраняем этот файл, запускаем его. На запрос добавить информацию в реестр нажимаем ДА.

11. Теперь нужно отредактировать файл machine.config, в моем случае он находится по пути: `C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config`

    Открываем этот файл блокнотом. Находим секцию

```
<system.data>
    <DbProviderFactories>
```

Добавляем в эту секцию строчку:

```
<add name="FirebirdClient Data Provider"
     invariant="FirebirdSql.Data.FirebirdClient"
     description=".Net Framework Data Provider for Firebird"
     type="FirebirdSql.Data.FirebirdClient.FirebirdClientFactory,
           FirebirdSql.Data.FirebirdClient, Version=4.10.0.0, Culture=neutral,
           PublicKeyToken=3750abcc3150b00c" />
```

> **Примечание**
>
> Всё это действительно для версии 4.10.0.

То же самое проделаем для `machine.config`, который находится в `c:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\`

Установка закончена.

Проверка установки

Для проверки, что всё успешно установилось, запускаем Visual Studio 2015. Находим обозреватель серверов и пытаемся подключиться к одной из существующих баз данных Firebird.

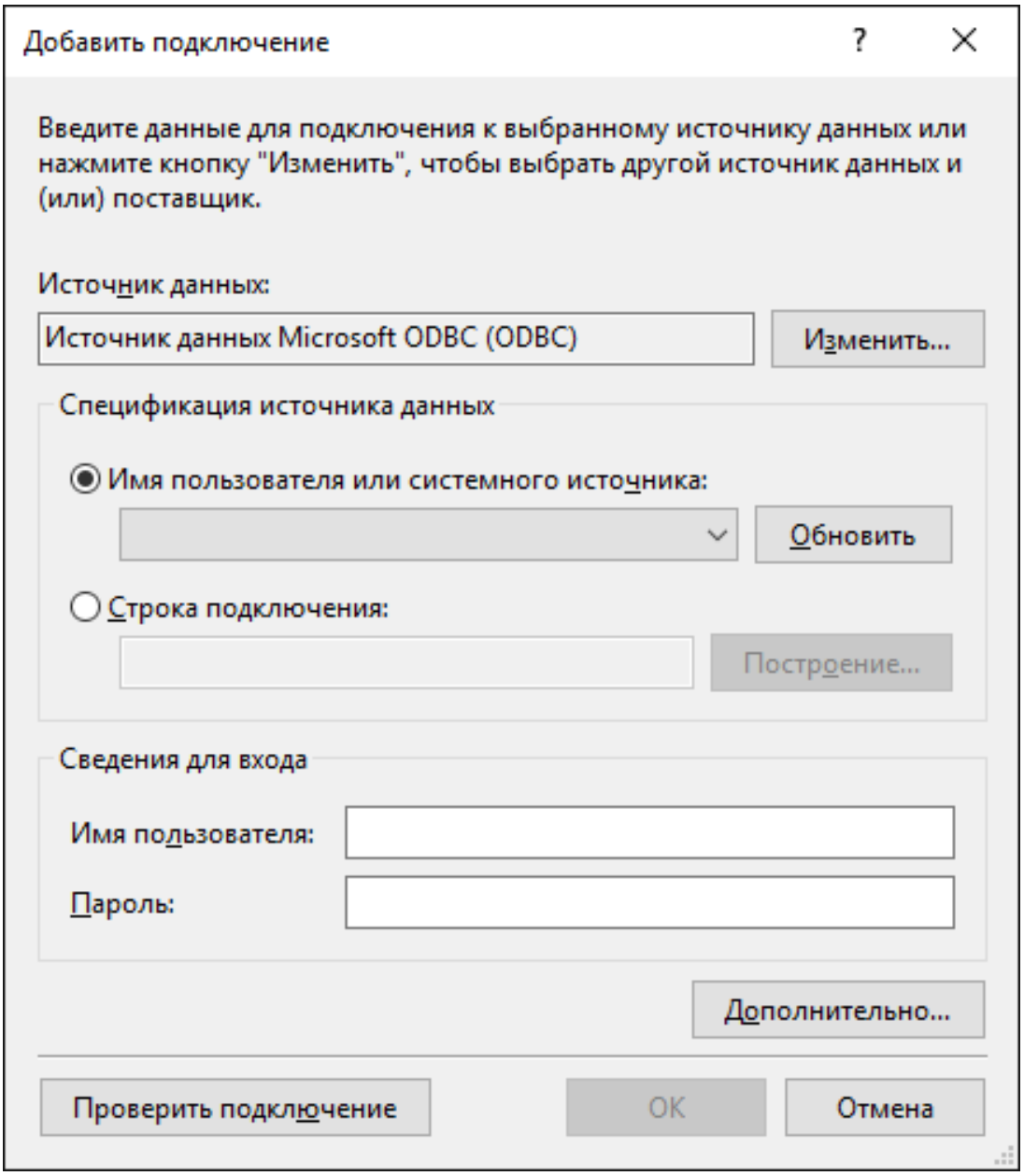

Рис. 3.1. Добавление подключения в Visual Studio

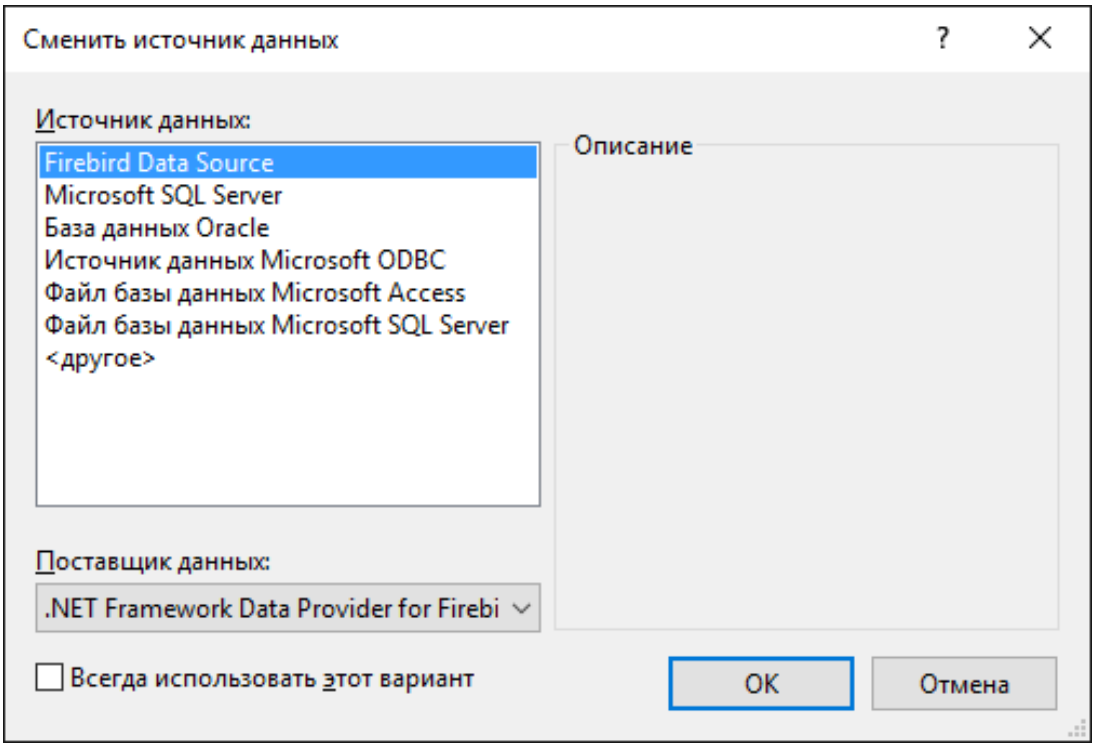

Рис. 3.2. Выбор провайдера подключения

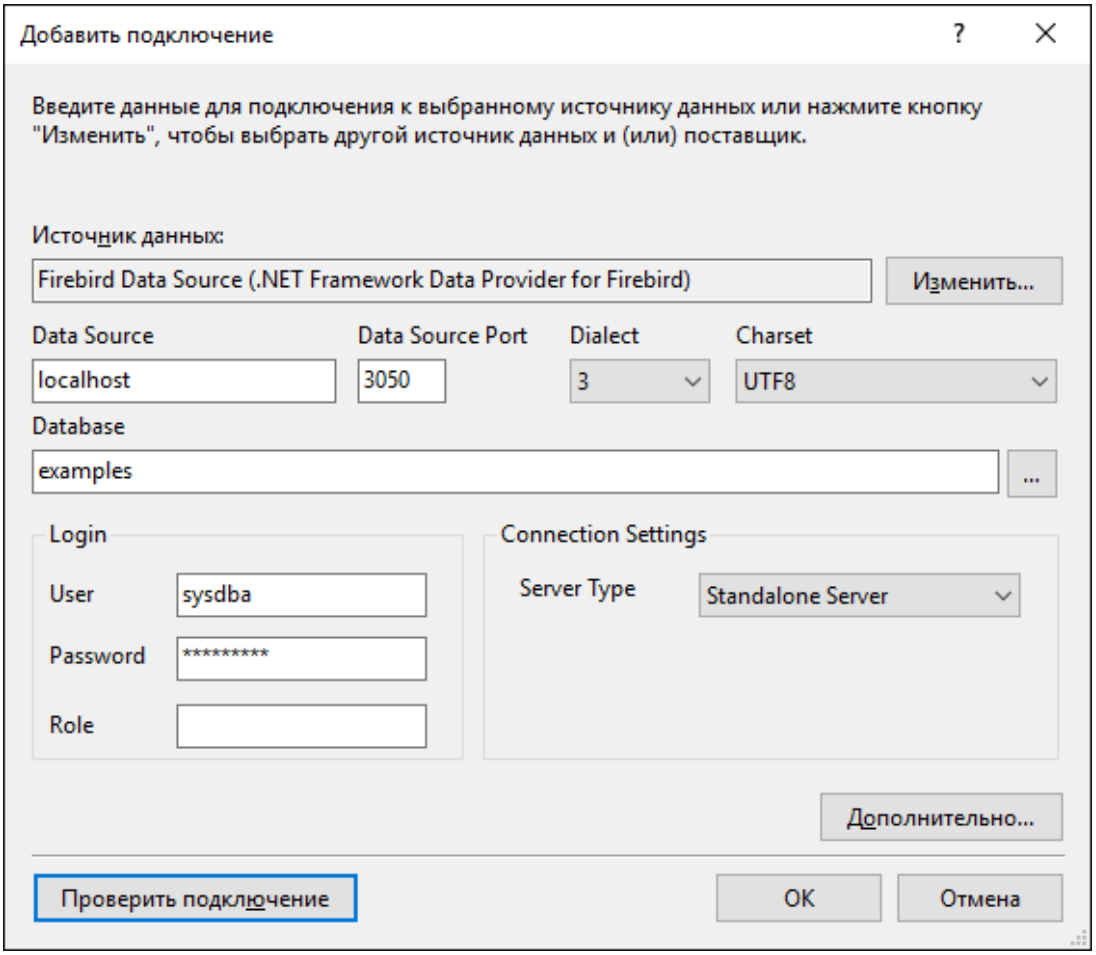

Рис. 3.3. Настройка подключения к Firebird

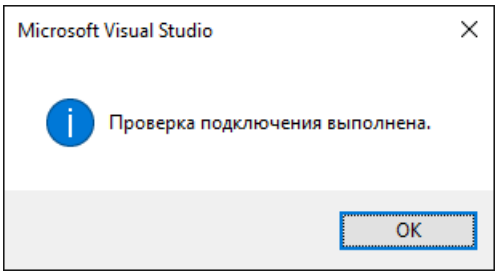

Рис. 3.4. Успешное подключение

Создание проекта

В данной главе мы рассмотрим пример создания Windows Forms приложения. Остальные типы приложений хоть и отличаются, но принципы работы с Firebird через Entity Framework остаются те же.

Добавление пакетов в проект

Прежде всего, после создания Windows Forms проекта нам необходимо добавить с помощью менеджера пакетов NuGet следующие пакеты:

- FirebirdSql.Data.FirebirdClient
- EntityFramework
- EntityFramework.Firebird

Для этого необходимо щёлкнуть правой клавишей мыши по имени проекта в обозревателе решений и в выпадающем меню выбрать пункт «Управление пакетами NuGet».

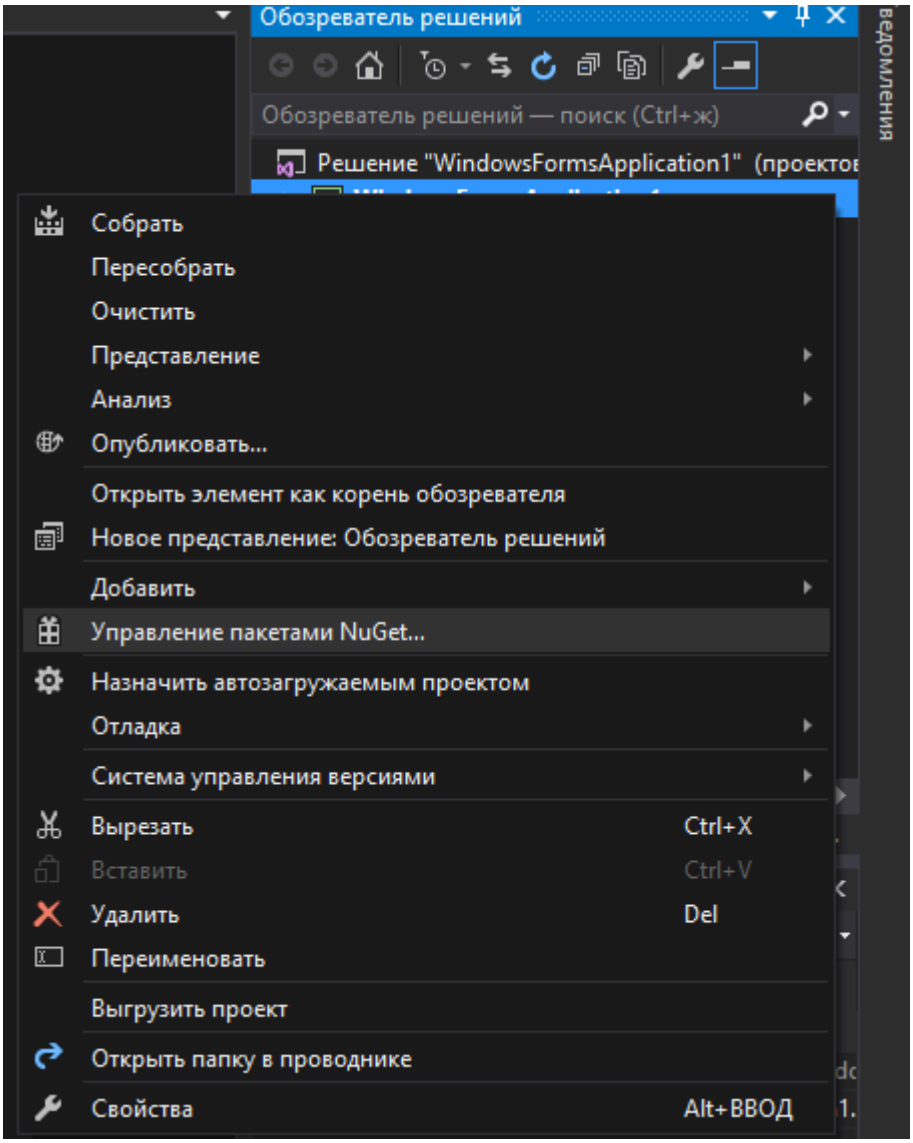

Рис. 3.5. Контекстное меню проекта

В появившемся менеджере пакетов произвести поиск и установку необходимых пакетов.

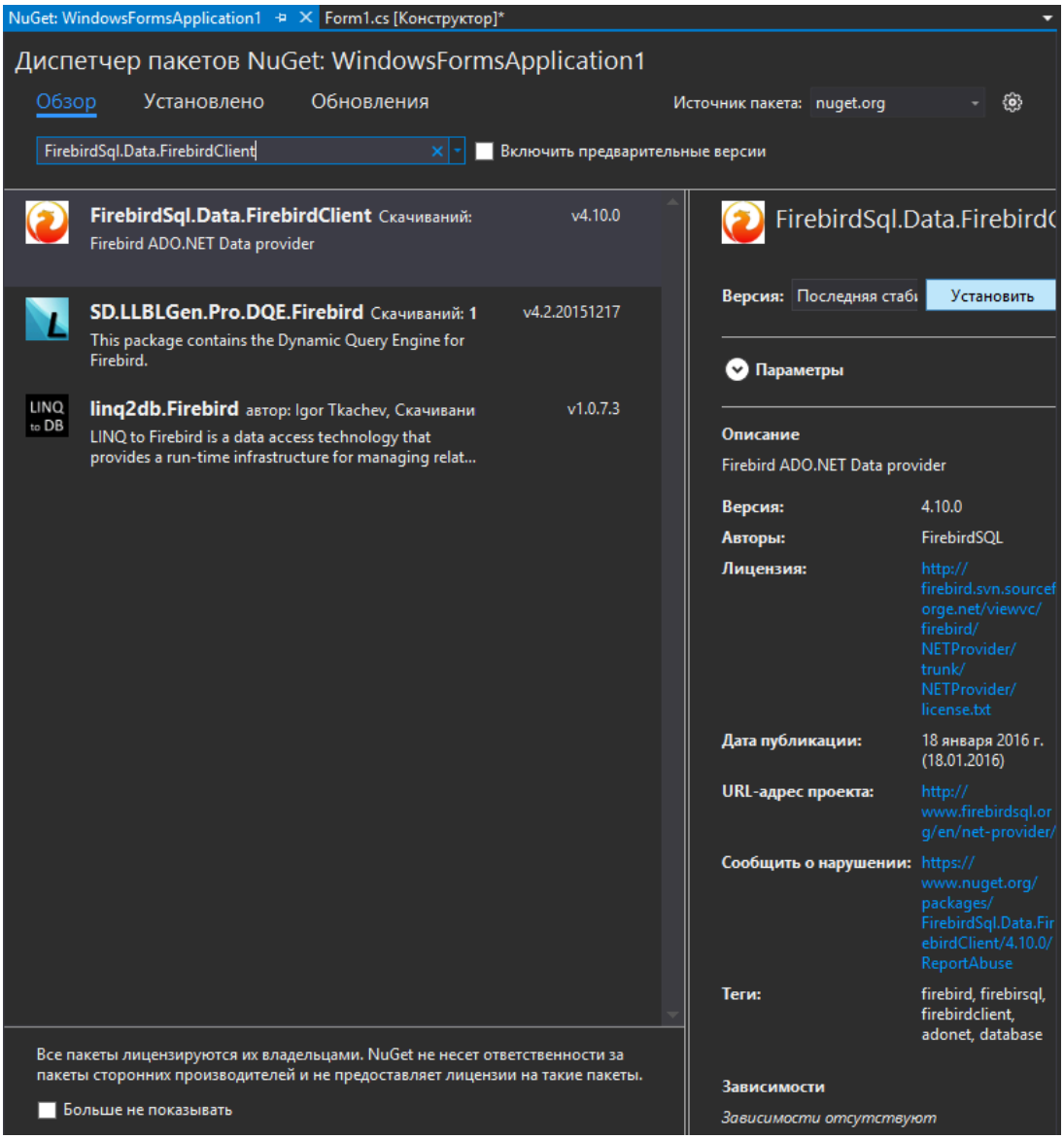

Рис. 3.6. Контекстное меню проекта

Создание EDM модели

В своём приложении мы будем использовать подход Code First.

Для создания модели EDM необходимо щёлкнуть правой клавишей мыши по имени проекта в обозревателе решений и выбрать пункт меню Добавить -> Создать элемент.

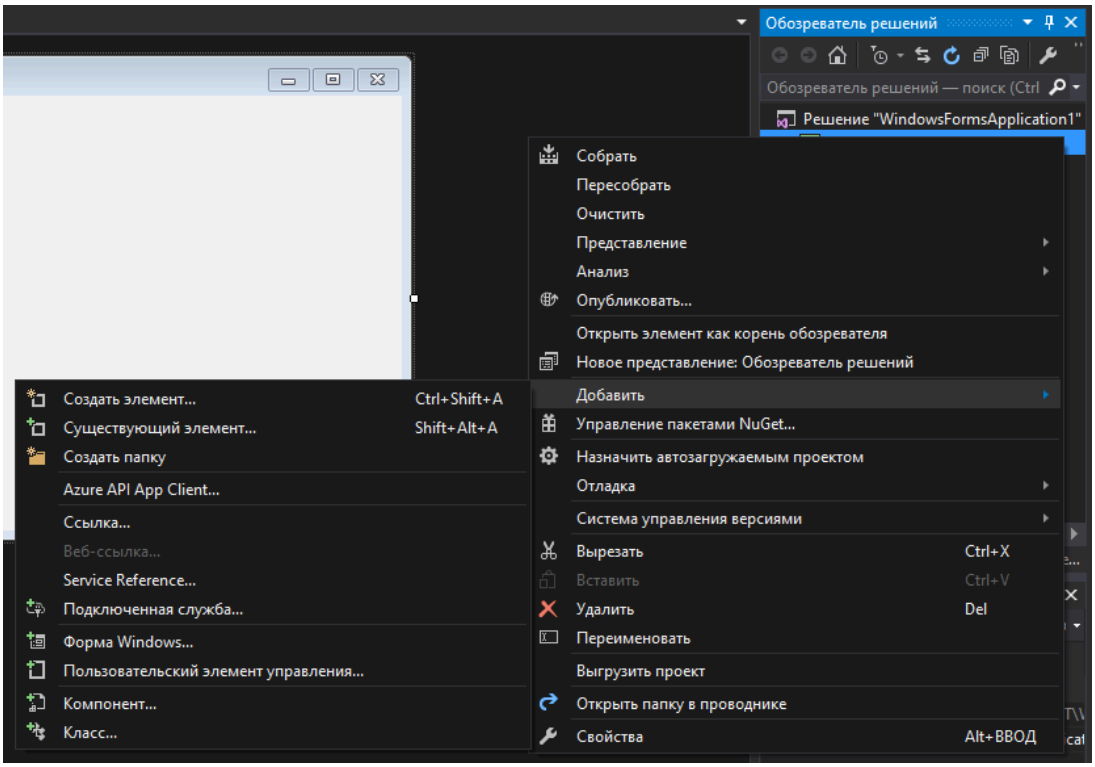

Рис. 3.7. Добавление элемента

Далее в мастере добавления нового элемента выбираем пункт «Модель ADO.NET EDM».

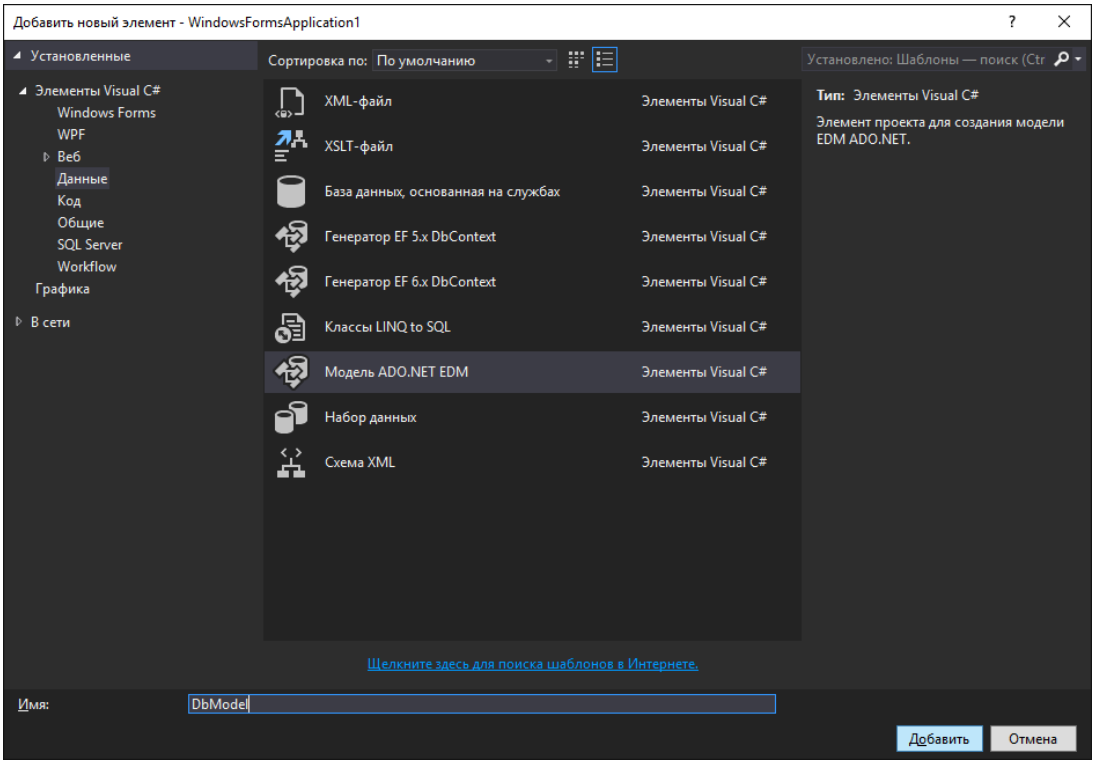

Рис. 3.8. Добавление модели ADO.NET EDM

Поскольку у нас уже существует база данных, то будем генерировать EDM модель из базы данных.

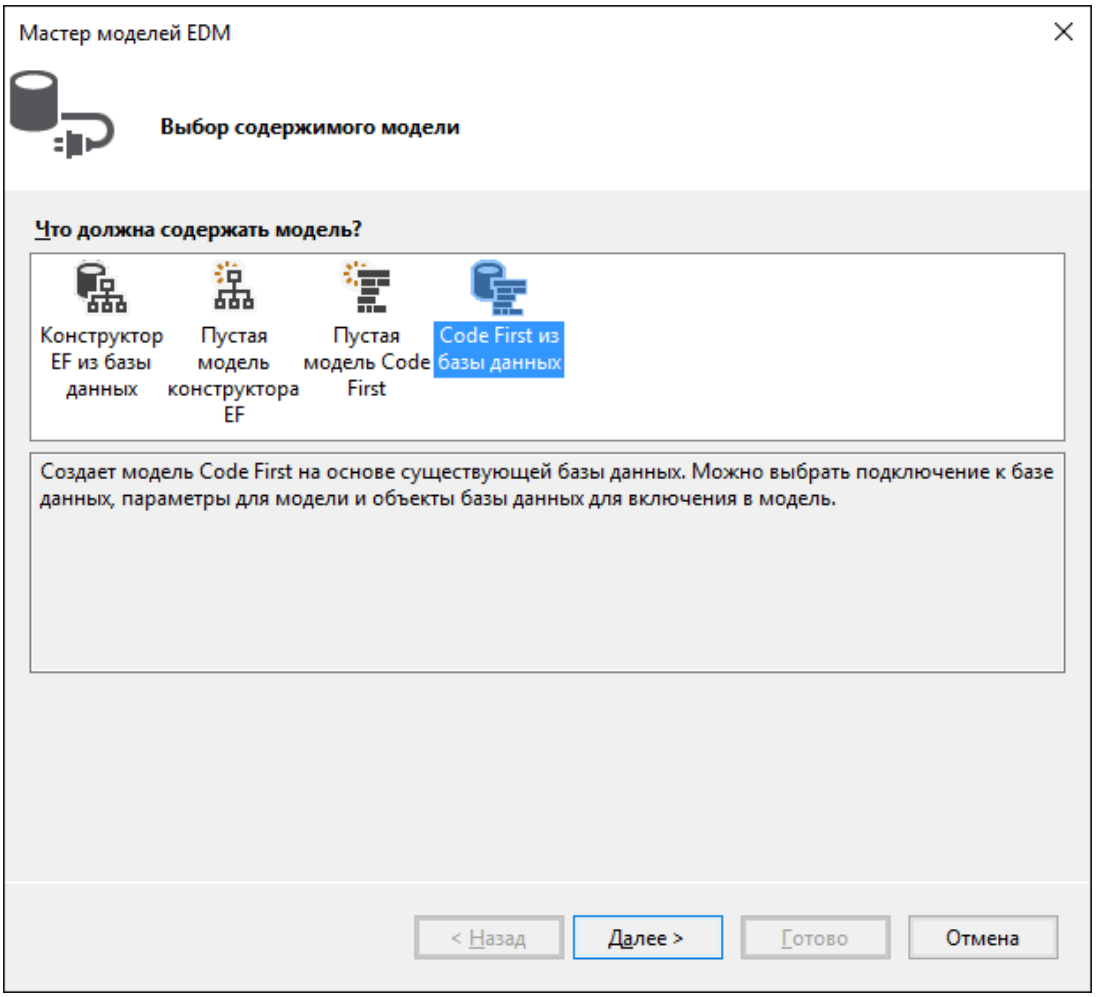

Рис. 3.9. Выбор типа модели

Теперь надо выбрать подключение, из которого будет создана модель. Если Такого подключения нет, то его надо создать.

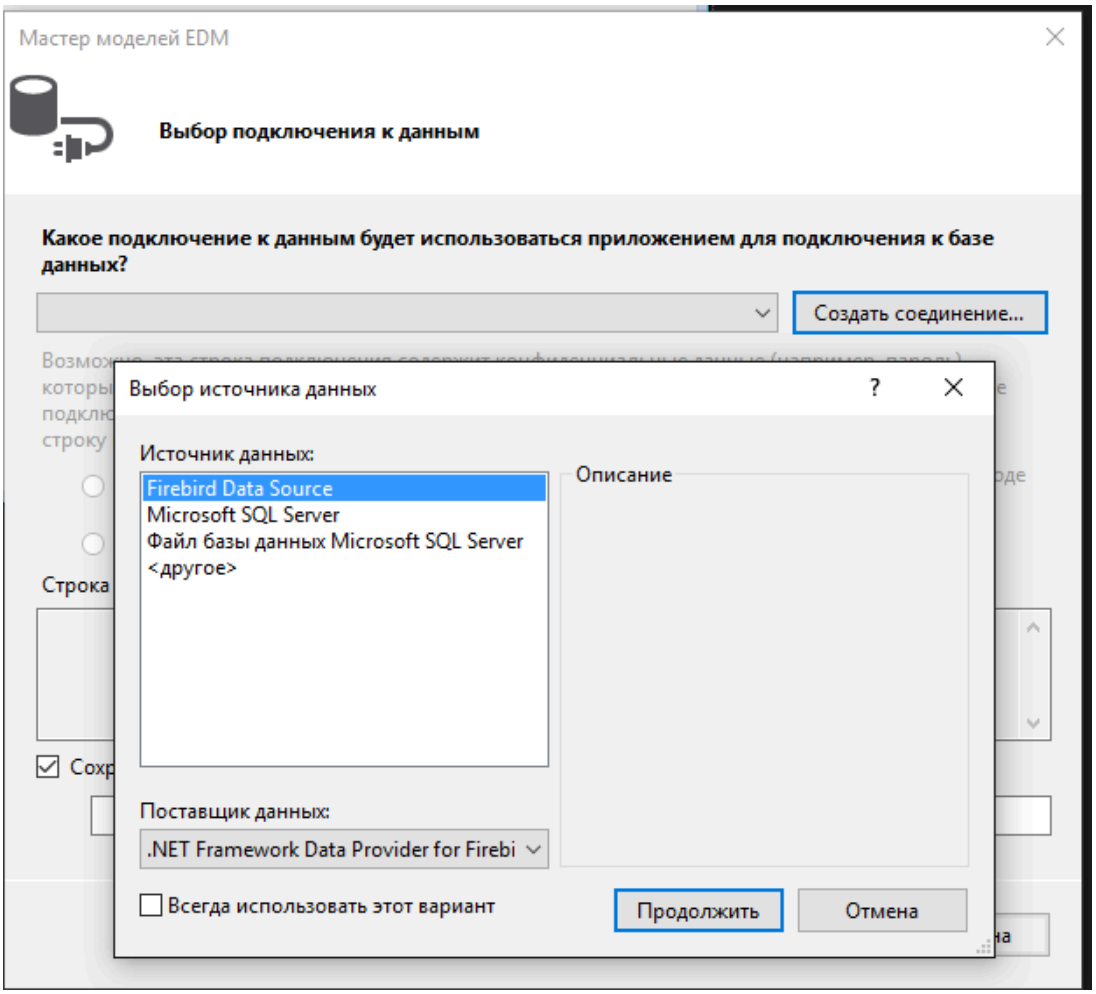

Рис. 3.10. Выбор подключения для модели

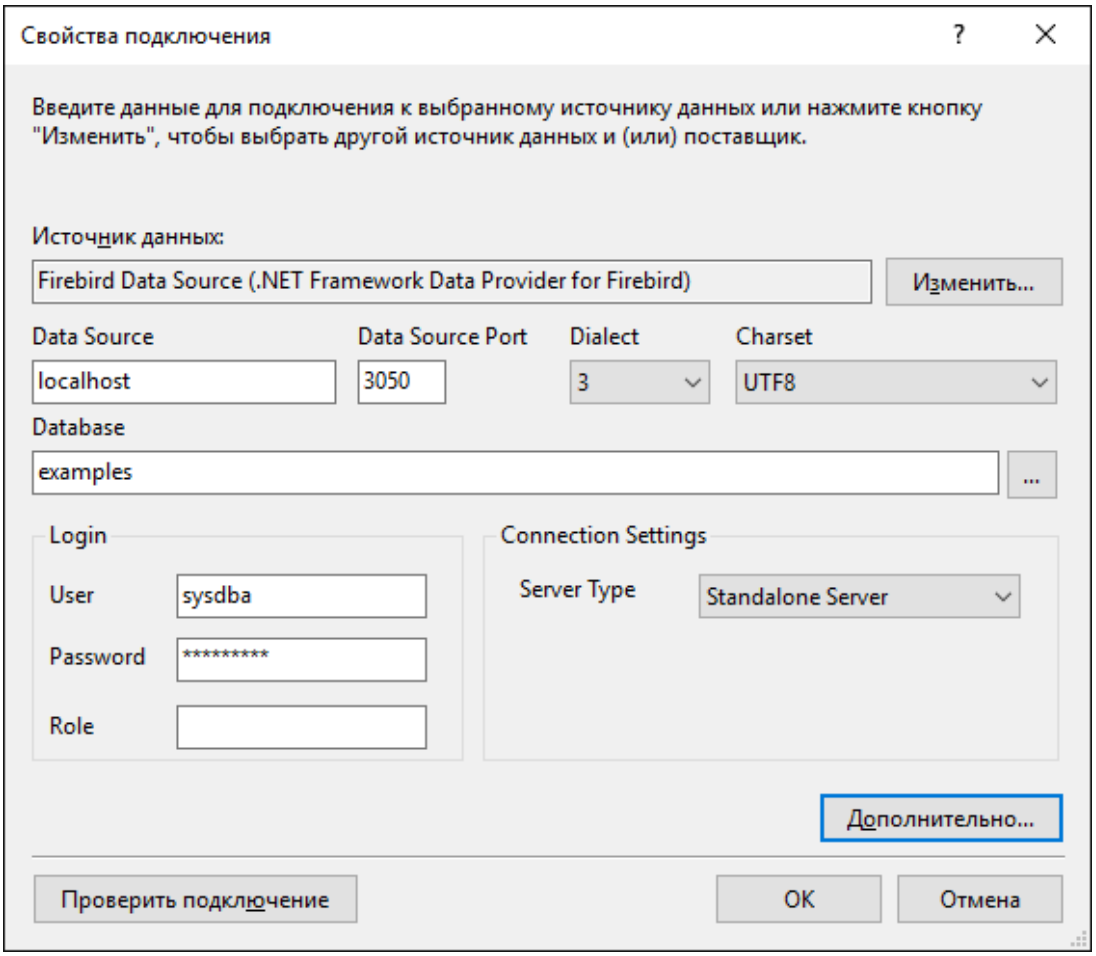

Рис. 3.11. Параметры подключения

Кроме основных параметров подключения могут потребоваться также указать ряд дополнительных параметров, например, уровень изолированности транзакций (по умолчанию Read Commited), использование пула подключений и т.д. Поскольку Entity Framework (как впрочем, ADO.NET в целом) использует отсоединённую модель взаимодействия, при которой каждое подключение и транзакция активна очень короткий промежуток времени, то я бы рекомендовал задать режим изолированности Snapshot.

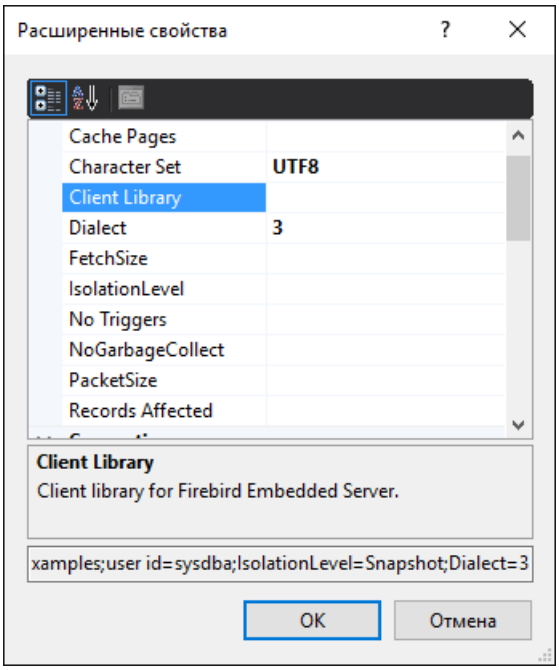

Рис. 3.12. Дополнительные свойства подключения

В процессе работы мастера создания модели у вас спросят, как хранить строку подключения.

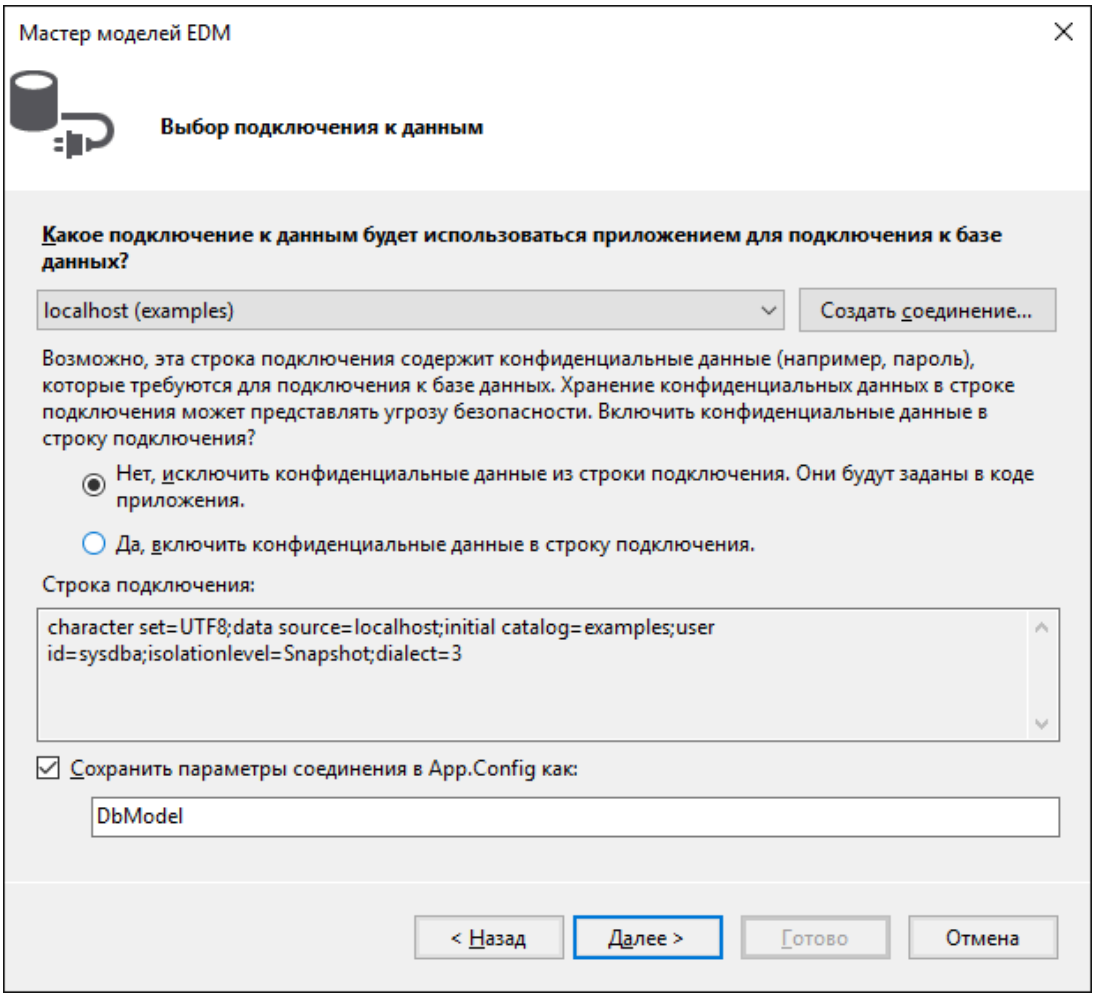

Рис. 3.13. Сохранение строки подключения

Если вы строите веб приложение или трёхзвенку, где все пользователи будут работать с базой данных под одной и той же учётной записью, то смело выбирайте «Да». Если же ваше приложение должно запрашивать учётные данные для соединения с базой данных выбирайте «Нет». Впрочем, с мастерами гораздо более удобно работать, когда у вас выбран пункт «Да». Вы всегда можете это изменить в готовом приложении, просто отредактировав строку подключения в файле конфигурации приложения `<AppName>.exe.conf`. Строка подключения будет сохранена в секции **connectionStrings** примерно в таком виде

```
<add name="DbModel"
     connectionString="character set=UTF8; data source=localhost;
                       initial catalog=examples; port number=3050;
                       user id=sysdba; dialect=3; isolationlevel=Snapshot;
                       pooling=True; password=masterkey;"
     providerName="FirebirdSql.Data.FirebirdClient" />
```

Для того чтобы файл конфигурации перестал хранить конфиденциальную информацию просто удалите из строки подключения *password=masterkey;*

Замечание о работе с Firebird 3.0

К сожалению текущий ADO .Net провайдер для Firebird (версия 5.9.0.0) не поддерживает шифрование сетевого трафика (по умолчанию в Firebird 3.0). Поэтому если вы желаете работать с Firebird 3.0, то вам необходимо изменить некоторые настройки в `firebird.conf` (или в `databases.conf` для конкретной БД), чтобы Firebird работал без использования шифрования сети. Для этого необходимо поменять следующие настройки:

```
WireCrypt = Disabled
```

Далее у вас спросят, какие таблицы и представления должны быть включены модель.

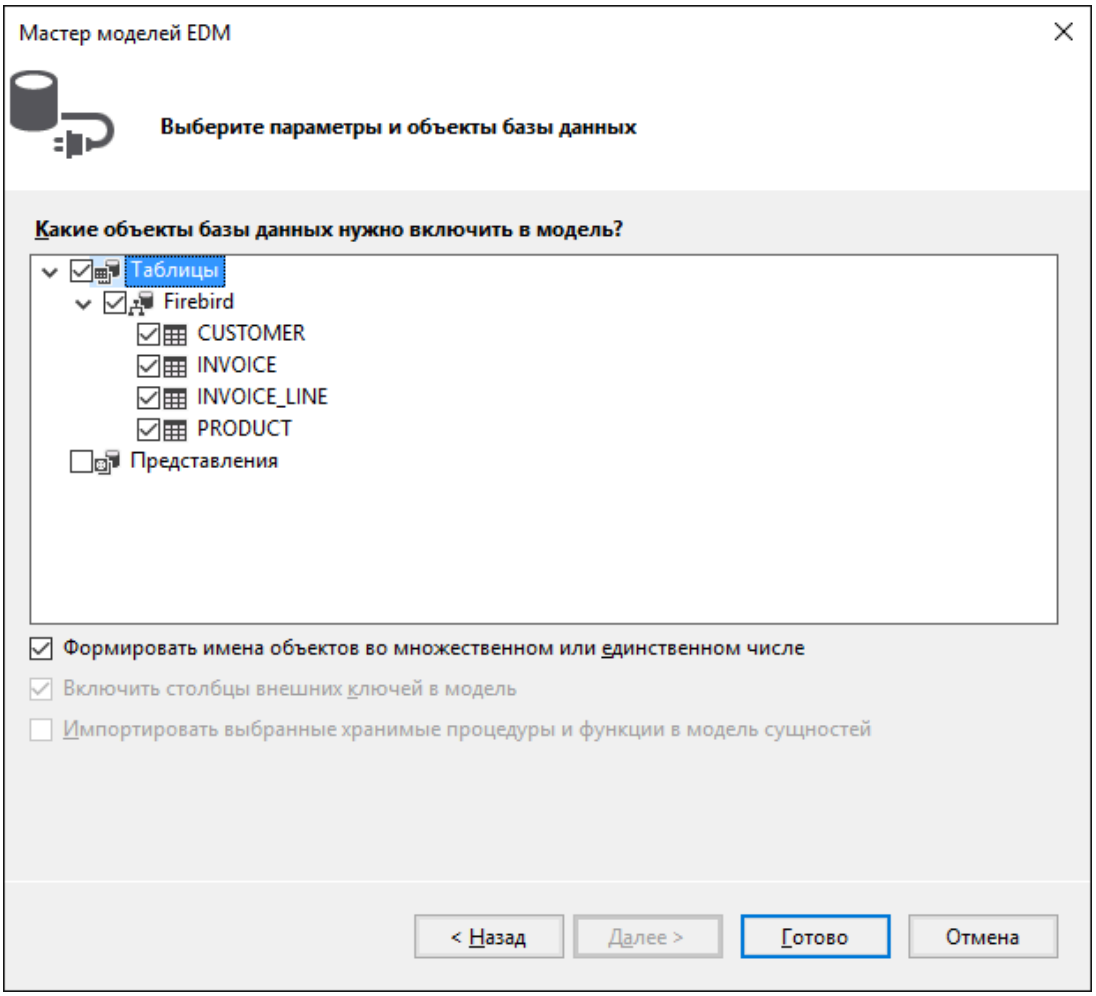

Рис. 3.14. Сохранение строки подключения

Базовая EDM готова.

EDM файлы

После работы этого мастера у вас должно появиться 5 новых файлов. Один файл модели и четыре файла описывающих каждую из сущностей модели.

Файл сущности

Давайте посмотрим один из сгенерированных файлов описывающих сущность INVOICE.

```
[Table("Firebird.INVOICE")]
public partial class INVOICE
{
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
       "CA2214:DoNotCallOverridableMethodsInConstructors")]
    public INVOICE()
    {
        INVOICE_LINES = new HashSet<INVOICE_LINE>();
    }

    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int INVOICE_ID { get; set; }

    public int CUSTOMER_ID { get; set; }

    public DateTime? INVOICE_DATE { get; set; }

    public decimal? TOTAL_SALE { get; set; }

    public short PAYED { get; set; }

    public virtual CUSTOMER CUSTOMER { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
       "CA2227:CollectionPropertiesShouldBeReadOnly")]
    public virtual ICollection<INVOICE_LINE> INVOICE_LINES { get; set; }
}
```

Класс содержат свойства, которые отображают поля таблицы INVOICE. Каждое из таких свойств снабжено атрибутами, описывающими ограничения. Подробнее об различных атрибутах вы можете почитать в документации Майкрософт Code First Data Annotations.

Навигационные свойства и "Ленивая загрузка"

Кроме того, было сгенерировано ещё два навигационных свойства `CUSTOMER` и `INVOICE_LINES`. Первое содержит ссылку на сущность поставщика, второе — коллекцию строк накладных. Оно было сгенерировано потому, что таблица INVOICE_LINE имеет внешний ключ на таблицу INVOICE. Конечно, вы можете удалить это свойство из сущности `INVOICE`, но делать это вовсе не обязательно. Дело в том, что в данном случае свойства `CUSTOMER` и `INVOICE_LINES` использует так называемую «ленивую загрузку». При таком загрузка осуществляется при первом обращении к объекту, т.е. если связанные данные не нужны, то они не подгружаются. Однако при первом же обращении к навигационному свойству эти данные автоматически подгружаются из БД.

При использовании ленивой загрузки надо иметь в виду некоторые моменты при объявлении классов. Так, классы, использующие ленивую загрузку должны быть публичными, а их свойства должны иметь модификаторы **public** и **virtual**.

Файл DbModel

Теперь откроем файл `DbModel.cs` описывающий модель в целом.

```
public partial class DbModel : DbContext
{
    public DbModel()
        : base("name=DbModel")
    {
    }

    public virtual DbSet<CUSTOMER> CUSTOMERS { get; set; }
    public virtual DbSet<INVOICE> INVOICES { get; set; }
    public virtual DbSet<INVOICE_LINE> INVOICE_LINES { get; set; }
    public virtual DbSet<PRODUCT> PRODUCTS { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<CUSTOMER>()
            .Property(e => e.ZIPCODE)
            .IsFixedLength();

        modelBuilder.Entity<CUSTOMER>()
            .HasMany(e => e.INVOICES)
            .WithRequired(e => e.CUSTOMER)
            .WillCascadeOnDelete(false);

        modelBuilder.Entity<PRODUCT>()
            .HasMany(e => e.INVOICE_LINES)
            .WithRequired(e => e.PRODUCT)
            .WillCascadeOnDelete(false);

        modelBuilder.Entity<INVOICE>()
            .HasMany(e => e.INVOICE_LINES)
            .WithRequired(e => e.INVOICE)
            .WillCascadeOnDelete(false);

    }
}
```

Здесь мы видим свойства описывающие набор данных для каждой сущности. А так же задание дополнительных свойств создания модели с помощью Fluent API. Полное описание Fluent API вы может прочитать в документации Microsoft Configuring/Mapping Properties and Types with the Fluent API.

Зададим в методе `OnModelCreating` точность для свойств типа decimal с помощью Fluent API. Для этого допишем следующие строчки

```
            modelBuilder.Entity<PRODUCT>()
                .Property(p => p.PRICE)
                .HasPrecision(15, 2);
```

```
            modelBuilder.Entity<INVOICE>()
                .Property(p => p.TOTAL_SALE)
                .HasPrecision(15, 2);

            modelBuilder.Entity<INVOICE_LINE>()
                .Property(p => p.SALE_PRICE)
                .HasPrecision(15, 2);

            modelBuilder.Entity<INVOICE_LINE>()
                .Property(p => p.QUANTITY)
                .HasPrecision(15, 0);
```

Создание пользовательского интерфейса

В нашем приложении мы создадим два справочника: справочник товаров и справочник заказчиков. Каждый справочник содержит сетку `DataGridView`, панель с кнопками `ToolStrip`, а также компонент `BindingSource`, который служит для упрощения привязки данных к элементам управления в форме.

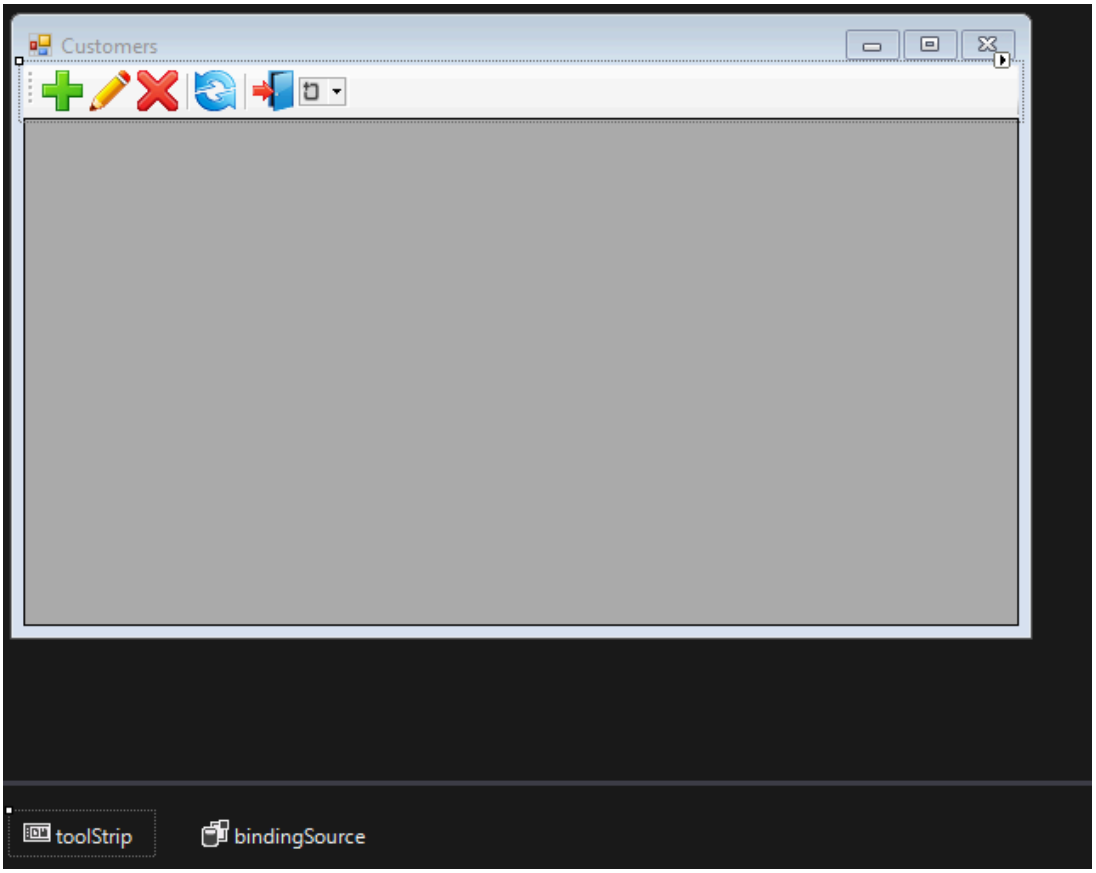

Рис. 3.15. Форма справочника заказчиков

Поскольку по функционалу оба справочника похожи и реализованы схожим образом описывать мы будем только один.

Получение контекста

Для работы с нашей моделью нам потребуется метод для получения контекста (или модели). В принципе для этого достаточно выполнить:

```
DbModel dbContext = new DbModel();
```

Однако, если в строке подключения не хранятся конфиденциальные данные (например, пароль), а мы инициализируем во время авторизации их при старте приложения, то нам потребуется специальный метод для хранения и восстановления строки подключения или сохранение ранее созданного контекста. Для этого создадим специальный класс, который помимо метода для получения контекста будет также содержать некоторые глобальные переменные уровня приложения, например рабочий период.

```
static class AppVariables
{
    private static DbModel dbContext = null;

    /// <summary>
    /// Дата начала рабочего периода
    /// </summary>
    public static DateTime StartDate { get; set; }

    /// <summary>
    /// Дата окончания рабочего периода
    /// </summary>
    public static DateTime FinishDate { get; set; }

    /// <summary>
    /// Returns an instance of the model (context)
    /// </summary>
    /// <returns>Model</returns>
    public static DbModel CreateDbContext() {
        dbContext = dbContext ?? new DbModel();
        return dbContext;
    }
}
```

Сама строка подключения инициализируется при старте приложения, после того как успешно прошла авторизация. Для этого в обработчике события `Load` главной формы напишем следующий код.

```
private void MainForm_Load(object sender, EventArgs e) {
    var dialog = new LoginForm();
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        var dbContext = AppVariables.getDbContext();
```

```
        try
        {
            string s = dbContext.Database.Connection.ConnectionString;
            var builder = new FbConnectionStringBuilder(s);
            builder.UserID = dialog.UserName;
            builder.Password = dialog.Password;

            dbContext.Database.Connection.ConnectionString = builder.ConnectionString;

            // пробуем подключится
            dbContext.Database.Connection.Open();
        }
        catch (Exception ex)
        {
            // отображаем ошибку
            MessageBox.Show(ex.Message, "Error");
            Application.Exit();
        }
    }
    else
        Application.Exit();
}
```

Теперь для получения контекста мы будем использовать статический метод `CreateDbContext`.

```
var dbContext = AppVariables.getDbContext();
```

Работа с данными

Сами по себе сущности модели не содержат никаких данных. Самым простым способом загрузить данные является вызовам метода `Load`, например вот так:

```
private void LoadCustomersData()
{
    dbContext.CUSTOMERS.Load();
    var customers = dbContext.CUSTOMERS.Local;

    bindingSource.DataSource = customers.ToBindingList();
}

private void CustomerForm_Load(object sender, EventArgs e)
{
    LoadCustomersData();

    dataGridView.DataSource = bindingSource;
    dataGridView.Columns["CUSTOMER_ID"].Visible = false;
}
```

Однако такой способ имеет ряд недостатков:

1. Метод `Load` загружает сразу все данные из таблицы CUSTOMER в память.

2. Ленивые свойства (`INVOICES`) хоть и не загружаются сразу, а лишь по мере обращения к ним, всё равно будут загружены при отображении записей в гриде. Причём ровно столько раз, сколько записей будет выведено.

3. Порядок записей неопределён.

Для обхода этих недостатком мы будем использовать технологию LINQ (Language Integrated Query), или точнее **LINQ to Entities**. LINQ to Entities предлагает простой и интуитивно понятный подход для получения данных с помощью выражений, которые по форме близки выражениям языка SQL. С синтаксисом LINQ вы можете ознакомиться по LINQ to Entities.

Методы расшрений LINQ

Методы расширений LINQ могут возвращать два объекта: `IEnumerable` и `IQueryable`. Интерфейс IQueryable наследуется от `IEnumerable`, поэтому по идее объект `IQueryable` это и есть также объект `IEnumerable`. Но между ними есть существенная разница.

Интерфейс `IEnumerable` находится в пространстве имён System.Collections. Объект `IEnumerable` представляет набор данных в памяти и может перемещаться по этим данным только вперёд. При выполнении запроса `IEnumerable` загружает все данные, и если нам надо выполнить их фильтрацию, то сама фильтрация происходит на стороне клиента.

Интерфейс `IQueryable` располагается в пространстве имён System.Linq. Объект `IQueryable` предоставляет удалённый доступ к базе данных и позволяет перемещаться по данным как в прямом порядке от начала до конца, так и в обратном порядке. В процессе создания запроса, возвращаемым объектом которого является `IQueryable`, происходит оптимизация запроса. В итоге в процессе его выполнения тратится меньше памяти, меньше пропускной способности сети.

Свойство `Local` возвращает интерфейс `IEnumerable`. Поэтому мы можем составлять LINQ запросы к нему.

```
private void LoadCustomersData()
{
    var dbContext = AppVariables.getDbContext();
    dbContext.CUSTOMERS.Load();

    var customers =
        from customer in dbContext.CUSTOMERS.Local
        orderby customer.NAME
        select new customer;

    bindingSource.DataSource = customers.ToBindingList();
}
```

Однако как уже сказано этот запрос будет выполняться над данными в памяти. В принципе для маленьких таблиц, которым не требуется предварительная фильтрация это приемлемо.

Для того чтобы LINQ запрос был преобразован в SQL и выполнялся на стороне сервера нам необходимо использовать в LINQ запросе вместо обращения к свойству

`dbContext.CUSTOMERS.Local` обращаться сразу к `dbContext.CUSTOMERS`. В этом случае нам не потребуется предварительный вызов `dbContext.CUSTOMERS.Load();` для загрузки коллекции в память.

IQueryable u BindingList

Однако тут нас подстерегает одна маленькая засада. Объекты `IQueryable` не умеют возвращать `BindingList`. `BindingList` является базовым классом для создания двустороннего механизма привязки данных. Из интерфейса `IQueryable` мы можем получить обычный список посредством вызова `ToList`, но в этом случае мы лишаемся приятных бонусов, таких как сортировка в гриде и многих других. Кстати в .NET Framework 5 это уже исправили и создали специальное расширение. Сделаем своё расширение, которое будет делать тоже самое.

```
public static class DbExtensions
{
    // Внутренний класс для маппинга на него значения генератора
    private class IdResult
    {
        public int Id { get; set; }
    }

    // Преобразование IQueryable в BindingList
    public static BindingList<T> ToBindingList<T>
        (this IQueryable<T> source) where T : class
    {
        return (new ObservableCollection<T>(source)).ToBindingList();
    }

    // Получение следующего значения последовательности
    public static int NextValueFor(this DbModel dbContext, string genName)
    {
        string sql = String.Format(
            "SELECT NEXT VALUE FOR {0} AS Id FROM RDB$DATABASE", genName);
        return dbContext.Database.SqlQuery<IdResult>(sql).First().Id;
    }

    // Отсоединение всех объектов коллекции DbSet от контекста
    // Полезно для обновлении кеша
    public static void DetachAll<T>(this DbModel dbContext, DbSet<T> dbSet)
        where T : class
    {
        foreach (var obj in dbSet.Local.ToList())
        {
            dbContext.Entry(obj).State = EntityState.Detached;
        }
    }

    // Обновление всех изменённых объектов в коллекции
    public static void Refresh(this DbModel dbContext, RefreshMode mode,
       IEnumerable collection)
    {
        var objectContext = ((IObjectContextAdapter)dbContext).ObjectContext;
        objectContext.Refresh(mode, collection);
    }
```

```
    // Обновление объекта
    public static void Refresh(this DbModel dbContext, RefreshMode mode,
         object entity)
    {
        var objectContext = ((IObjectContextAdapter)dbContext).ObjectContext;
        objectContext.Refresh(mode, entity);
    }
}
```

Другие расширения

В этом же классе присутствует ещё несколько расширений:

NextValueFor

предназначен для получения следующего значения генератора. Метод `dbContext.Database.SqlQuery` позволяет выполнять SQL запросы напрямую и отображать их результаты на некоторую сущность (проекцию). Вы можете воспользоваться им, если вам потребуется выполнить SQL запрос напрямую.

DetachAll

предназначен для отсоединения всех объектов коллекции DBSet от контекста. Это необходимо для обновления внутреннего кеша. Дело в том, что в рамках контекста все извлекаемые кешируются и не извлекаются из базы данных снова. Однако это не всегда полезно, поскольку затрудняет получение изменённых записей сделанных в другом контексте.

> **Примечание**
>
> В Web приложениях контекст обычно живёт очень короткое время, а новый контекст имеет не заполненный кеш.

Refresh

предназначен для обновления свойств объекта-сущности. Он полезен для обновления свойств объекта после его редактирования или добавления.

Код для загрузки данных

Таким образом, наш код загрузки данных будет выглядеть так

```
private void LoadCustomersData()
{
    var dbContext = AppVariables.getDbContext();
    // отсоединяем все загруженные объекты
    // это необходимо чтобы обновился внутренний кеш
    // при второй и последующих вызовах этого метода
    dbContext.DetachAll(dbContext.CUSTOMERS);

    var customers =
          from customer in dbContext.CUSTOMERS
```

```
            orderby customer.NAME
            select customer;


    bindingSource.DataSource = customers.ToBindingList();
}

private void CustomerForm_Load(object sender, EventArgs e)
{
    LoadCustomersData();

    dataGridView.DataSource = bindingSource;
    dataGridView.Columns["INVOICES"].Visible = false;
    dataGridView.Columns["CUSTOMER_ID"].Visible = false;
    dataGridView.Columns["NAME"].HeaderText = "Name";
    dataGridView.Columns["ADDRESS"].HeaderText = "Address";
    dataGridView.Columns["ZIPCODE"].HeaderText = "ZipCode";
    dataGridView.Columns["PHONE"].HeaderText = "Phone";
}
```

Добавление заказчика

Код обработчика события на нажатие кнопки добавления выглядит следующим образом.

```
private void btnAdd_Click(object sender, EventArgs e) {
    var dbContext = AppVariables.getDbContext();
    // создание нового экземпляра сущности
    var customer = (CUSTOMER)bindingSource.AddNew();
    // создаём форму для редактирования
    using (CustomerEditorForm editor = new CustomerEditorForm()) {
        editor.Text = "Добавление заказчика";
        editor.Customer = customer;
        // Обработчик закрытия формы
        editor.FormClosing += delegate (object fSender, FormClosingEventArgs fe) {
           if (editor.DialogResult == DialogResult.OK) {
              try {
                 // получаем новое значение генератора
                 // и присваиваем его идентификатору
                 customer.CUSTOMER_ID = dbContext.NextValueFor("GEN_CUSTOMER_ID");
                 // добавляем нового заказчика
                 dbContext.CUSTOMERS.Add(customer);
                 // пытаемся сохранить изменения
                 dbContext.SaveChanges();
                 // и обновить текущую запись
                 dbContext.Refresh(RefreshMode.StoreWins, customer);
              }
              catch (Exception ex) {
                 // отображаем ошибку
                 MessageBox.Show(ex.Message, "Error");
                 // не закрываем форму для возможности исправления ошибки
                 fe.Cancel = true;
              }
           }
           else
```

```
                bindingSource.CancelEdit();

        };
        // показываем модальную форму
        editor.ShowDialog(this);
    }
}
```

При добавлении новой записи мы получаем значение следующего идентификатора с помощью генератора. Мы могли бы не инициализировать значение идентификатора, и в этом случае отработал бы BEFORE INSERT триггер, который всё равно дёрнул бы следующее значение генератора. Однако в этом случае мы не смогли бы обновить вновь добавленную запись.

Редактирование заказчика

Код обработчика события на нажатие кнопки редактирования выглядит следующим образом.

```
private void btnEdit_Click(object sender, EventArgs e) {
    var dbContext = AppVariables.getDbContext();
    // получаем сущность
    var customer = (CUSTOMER)bindingSource.Current;
    // создаём форму для редактирования
    using (CustomerEditorForm editor = new CustomerEditorForm()) {
        editor.Text = "Редактирование заказчика";
        editor.Customer = customer;
        // Обработчик закрытия формы
        editor.FormClosing += delegate (object fSender, FormClosingEventArgs fe) {
            if (editor.DialogResult == DialogResult.OK) {
                try {
                    // пытаемся сохранить изменения
                    dbContext.SaveChanges();
                    dbContext.Refresh(RefreshMode.StoreWins, customer);
                    // обновляем все связанные контролы
                    bindingSource.ResetCurrentItem();
                }
                catch (Exception ex) {
                    // отображаем ошибку
                    MessageBox.Show(ex.Message, "Error");
                    // не закрываем форму для возможности исправления ошибки
                    fe.Cancel = true;
                }
            }
            else
                bindingSource.CancelEdit();

        };
        // показываем модальную форму
        editor.ShowDialog(this);
    }
}
```

Форма для редактирования заказчика выглядит следующим образом.

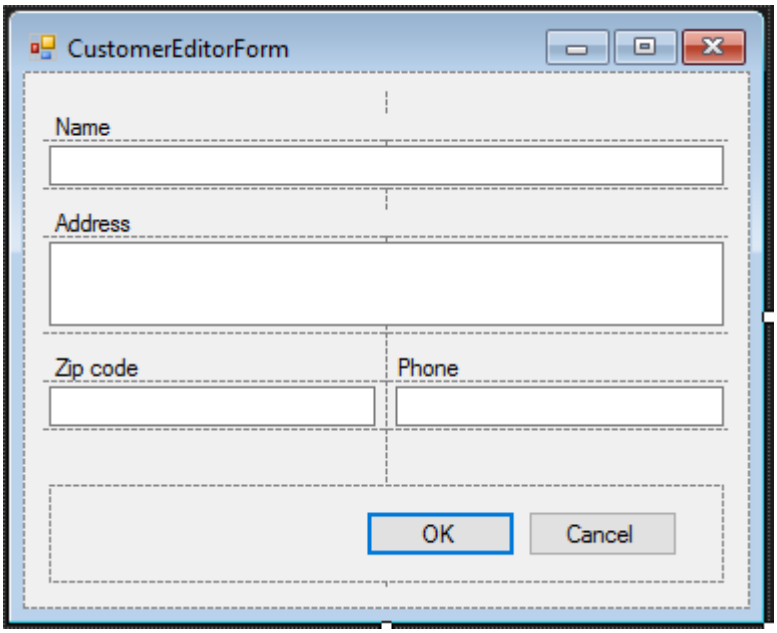

Рис. 3.16. Форма редактирования заказчика

Код привязки к данным очень прост.

```
public CUSTOMER Customer { get; set; }

private void CustomerEditorForm_Load(object sender, EventArgs e)
{
    edtName.DataBindings.Add("Text", this.Customer, "NAME");
    edtAddress.DataBindings.Add("Text", this.Customer, "ADDRESS");
    edtZipCode.DataBindings.Add("Text", this.Customer, "ZIPCODE");
    edtPhone.DataBindings.Add("Text", this.Customer, "PHONE");
}
```

Удаление заказчика

Код обработчика события на нажатие кнопки удаления выглядит следующим образом.

```
private void btnDelete_Click(object sender, EventArgs e) {
    var dbContext = AppVariables.getDbContext();
    var result = MessageBox.Show("Вы действительно хотите удалить заказчика?",
                "Подтверждение",
                MessageBoxButtons.YesNo,
                MessageBoxIcon.Question);
    if (result == DialogResult.Yes) {
       // получаем сущность
       var customer = (CUSTOMER)bindingSource.Current;
       try {
           dbContext.CUSTOMERS.Remove(customer);
           // пытаемся сохранить изменения
           dbContext.SaveChanges();
           // удаляем из связанного списка
```

```
            bindingSource.RemoveCurrent();
        }
        catch (Exception ex) {
            // отображаем ошибку
            MessageBox.Show(ex.Message, "Error");
        }
    }
}
```

Журналы

В нашем приложении будет один журнал «Счёт-фактуры». В отличие от справочников журналы содержат довольно большое количество записей и являются часто пополняемыми.

Счёт-фактура — состоит из заголовка, где описываются общие атрибуты (номер, дата, заказчик …), и строк счёт-фактуры со списком товаром, их количеством, стоимостью и т.д. Для таких документов удобно иметь два грида: в главном отображаются данные о шапке документа, а в детализирующем — список товаров. Таким образом, на форму документа нам потребуется поместить два компонента `DataGridView`, к каждому из которых привязать свой `BindingSource`

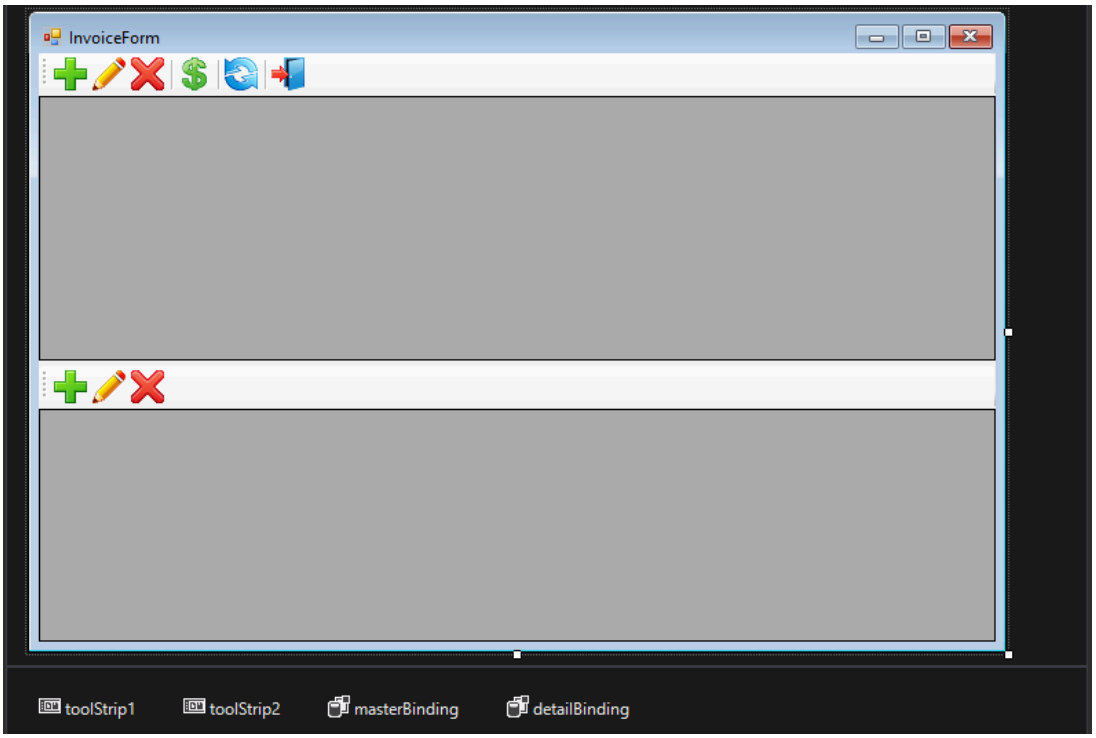

Рис. 3.17. Форма журнала счёт-фактур

Фильтрация данных

Большинство журналов содержат поле с датой создания документа. Чтобы уменьшить количество выбираемых данных обычно принято вводить такое понятие как рабочий период для того, чтобы уменьшить объём данных передаваемый на клиента. Рабочий период — это

диапазон дат, внутри которого требуются рабочие документы. Поскольку приложение может содержать более одного журнала, то имеет смысл разместить переменные, содержащие дату начала и окончания рабочего периода, в глобальном модуле AppVariables (см. Получение контекста), который, так или иначе, используется всеми модулями, работающими с БД. При старте приложения рабочий период обычно инициализируется датой начала и окончания текущего квартала (могут быть другие варианты). В ходе работы приложения можно изменить рабочий период по желанию пользователя.

Поскольку чаще всего требуются именно последние введённые документы, то имеет смысл сортировать их по дате в обратном порядке. Извлекать данные, как и в случае со справочниками будем при помощи LINQ.

Загрузка данных счёт-фактур

С учётом вышесказанного, метод для загрузки данных шапок счёт-фактур будет выглядеть следующим образом:

```
public void LoadInvoicesData() {
    var dbContext = AppVariables.getDbContext();

    // запрос на LINQ преобразуется в SQL
    var invoices =
        from invoice in dbContext.INVOICES
        where (invoice.INVOICE_DATE >= AppVariables.StartDate) &&
              (invoice.INVOICE_DATE <= AppVariables.FinishDate)
        orderby invoice.INVOICE_DATE descending
        select new InvoiceView
        {
            Id = invoice.INVOICE_ID,
            Cusomer_Id = invoice.CUSTOMER_ID,
            Customer = invoice.CUSTOMER.NAME,
            Date = invoice.INVOICE_DATE,
            Amount = invoice.TOTAL_SALE,
            Payed = (invoice.PAYED == 1) ? "Yes" : "No"
        };

    masterBinding.DataSource = invoices.ToBindingList();
}
```

В качестве проекции мы использовали не анонимный тип, а класс `InvoiceView`. Это упрощает приведение типа. Определение класса `InvoiceView` выглядит следующим образом:

```
public class InvoiceView {
    public int Id { get; set; }
    public int Cusomer_Id { get; set; }
    public string Customer { get; set; }
    public DateTime? Date { get; set; }
    public decimal? Amount { get; set; }
    public string Payed { get; set; }
```

```
    public void Load(int Id) {
        var dbContext = AppVariables.getDbContext();

        var invoices =
            from invoice in dbContext.INVOICES
            where invoice.INVOICE_ID == Id
            select new InvoiceView
            {
                Id = invoice.INVOICE_ID,
                Cusomer_Id = invoice.CUSTOMER_ID,
                Customer = invoice.CUSTOMER.NAME,
                Date = invoice.INVOICE_DATE,
                Amount = invoice.TOTAL_SALE,
                Payed = (invoice.PAYED == 1) ? "Yes" : "No"
            };

        InvoiceView invoiceView = invoices.ToList().First();
        this.Id = invoiceView.Id;
        this.Cusomer_Id = invoiceView.Cusomer_Id;
        this.Customer = invoiceView.Customer;
        this.Date = invoiceView.Date;
        this.Amount = invoiceView.Amount;
        this.Payed = invoiceView.Payed;
    }
}
```

Метод `Load` позволяет нам быстро обновить 1 добавленную или обновлённую запись в гриде, вместо того чтобы полностью перезагружать все записи.

Код обработчика события на нажатие кнопки добавления выглядит следующим образом.

```
private void btnAddInvoice_Click(object sender, EventArgs e) {
    var dbContext = AppVariables.getDbContext();
    var invoice = dbContext.INVOICES.Create();

    using (InvoiceEditorForm editor = new InvoiceEditorForm()) {
        editor.Text = "Добавление счёт фактуры";
        editor.Invoice = invoice;
        // Обработчик закрытия формы
        editor.FormClosing += delegate (object fSender, FormClosingEventArgs fe) {
            if (editor.DialogResult == DialogResult.OK) {
                try {
                    // получаем значение генератора
                    invoice.INVOICE_ID = dbContext.NextValueFor("GEN_INVOICE_ID");
                    // добавляем запись
                    dbContext.INVOICES.Add(invoice);
                    // пытаемся сохранить изменения
                    dbContext.SaveChanges();
                    // добавляем проекцию в список для грида
                    ((InvoiceView)masterBinding.AddNew()).Load(invoice.INVOICE_ID);
                }
                catch (Exception ex) {
                    // отображаем ошибку
                    MessageBox.Show(ex.Message, "Error");
                    // не закрываем форму для возможности исправления ошибки
```

```
                    fe.Cancel = true;
                }
            }
        };
        // показываем модальную форму
        editor.ShowDialog(this);
    }
}
```

В отличие от аналогичного метода справочника здесь обновление записи происходит не с помощью вызова `dbContext.Refresh`, а с помощью метода `Load` проекции `InvoiceView`. Дело в том, что `dbContext.Refresh` предназначен для обновления объектов сущностей, а не произвольных проекций, которые могут получаться сложными LINQ запросами.

Код обработчика события на нажатие кнопки редактирования выглядит следующим образом.

```
private void btnEditInvoice_Click(object sender, EventArgs e) {
    // получение контекста
    var dbContext = AppVariables.getDbContext();
    // поиск сущности по идентификатору
    var invoice = dbContext.INVOICES.Find(this.CurrentInvoice.Id);

    if (invoice.PAYED == 1) {
        MessageBox.Show("Изменение не возможно, счёт фактура уже оплачена.", "Ошибка");
        return;
    }

    using (InvoiceEditorForm editor = new InvoiceEditorForm()) {
        editor.Text = "Edit invoice";
        editor.Invoice = invoice;
        // Обработчик закрытия формы
        editor.FormClosing += delegate (object fSender, FormClosingEventArgs fe) {
            if (editor.DialogResult == DialogResult.OK) {
                try {
                    // пытаемся сохранить изменения
                    dbContext.SaveChanges();
                    // перезагружаем проекцию
                    CurrentInvoice.Load(invoice.INVOICE_ID);
                    masterBinding.ResetCurrentItem();
                }
                catch (Exception ex) {
                    // отображаем ошибку
                    MessageBox.Show(ex.Message, "Error");
                    // не закрываем форму для возможности исправления ошибки
                    fe.Cancel = true;
                }
            }
        };
        // показываем модальную форму
        editor.ShowDialog(this);
    }
}
```

Здесь нам потребовалось найти сущность по её идентификатору доступному в текущей записи. Свойство CurrentInvoice предназначено для получения выделенной в гриде счёт-фактуры. Оно реализовано так:

```
public InvoiceView CurrentInvoice {
    get {
        return (InvoiceView)masterBinding.Current;
    }
}
```

Удаление шапки счёт фактуры вы можете сделать самостоятельно.

Оплата счёт-фактуры

Помимо добавления, редактирования и удаления для счёт-фактур мы ввели ещё одну операцию оплаты, код метода реализующего эту операцию выглядит следующим образом:

```
private void btnInvoicePay_Click(object sender, EventArgs e) {
    var dbContext = AppVariables.getDbContext();
    var invoice = dbContext.INVOICES.Find(this.CurrentInvoice.Id);
    try {
        if (invoice.PAYED == 1)
           throw new Exception("Изменение не возможно, счёт фактура уже оплачена.");

        invoice.PAYED = 1;
        // сохраняем изменения
        dbContext.SaveChanges();
        // перезагружаем изменённую запись
        CurrentInvoice.Load(invoice.INVOICE_ID);
        masterBinding.ResetCurrentItem();
    }
    catch (Exception ex) {
        // отображаем ошибку
        MessageBox.Show(ex.Message, "Ошибка");
    }
}
```

Отображение позиций счёт-фактур

Для отображения позиций счёт-фактуры существует два метода:

1. Получать данные по каждой счёт-фактуре из навигационного свойства INVOICE_LINE и отображать содержимое этого сложного свойства (возможно с преобразованиями LINQ) в детейл гриде.

2. Получать данные по каждой счёт-фактуре отдельным LINQ запросом, который будет перевыполняться при перемещении в указателя в мастер гриде.

Каждый из методов имеет свои преимущества и недостатки.

Первый метод предполагает, что при открытии формы счёт-фактуры мы должны сразу извлечь все счёт-фактуры за указанный период и связанные данные по их позициям. Это хоть и выполняется одним SQL запросом, но может занять довольно много времени, и требует значительного объёма оперативной памяти. Этот метод лучше подходит для WEB приложений где вывод записей обычно происходит постранично.

Второй метод несколько более сложен в реализации, но позволяет быстро открывать форму счёт-фактуры и менее требователен к ресурсам, однако при каждом перемещении указателя в мастер гриде будет перевыполняться SQL запрос и загружать сетевой трафик (хотя объём будет невелик).

В нашем приложении я буду использовать второй подход. Для этого необходимо написать обработчик события изменения текущей записи для компонента `BindingSource`.

```
private void masterBinding_CurrentChanged(object sender, EventArgs e) {
   LoadInvoiceLineData(this.CurrentInvoice.Id);
   detailGridView.DataSource = detailBinding;
}
```

Метод для загрузки данных о позициях счёт-фактуры выглядит следующим образом:

```
private void LoadInvoiceLineData(int? id) {
    var dbContext = AppVariables.getDbContext();

    var lines =
        from line in dbContext.INVOICE_LINES
        where line.INVOICE_ID == id
        select new InvoiceLineView
        {
            Id = line.INVOICE_LINE_ID,
            Invoice_Id = line.INVOICE_ID,
            Product_Id = line.PRODUCT_ID,
            Product = line.PRODUCT.NAME,
            Quantity = line.QUANTITY,
            Price = line.SALE_PRICE,
            Total = Math.Round(line.QUANTITY * line.SALE_PRICE, 2)
        };

    detailBinding.DataSource = lines.ToBindingList();
}
```

В качестве проекции мы использовали класс `InvoiceLineView`.

```
public class InvoiceLineView {
   public int Id { get; set; }
   public int Invoice_Id { get; set; }
   public int Product_Id { get; set; }
   public string Product { get; set; }
```

```
    public decimal Quantity { get; set; }
    public decimal Price { get; set; }
    public decimal Total { get; set; }
}
```

Замечу, что в отличие от класса `InvoiceView` здесь отсутствует метод для загрузки одной текущей записи. Здесь скорость перезагрузки детейл грида не настолько критична, поскольку один документ не содержит тысячи позиций, однако при желании вы можете реализовать такой метод.

Добавим специальное свойство для получения текущей строки документа выделенной в детейл гриде.

```
public InvoiceLineView CurrentInvoiceLine {
    get {
        return (InvoiceLineView)detailBinding.Current;
    }
}
```

Работа с хранимыми процедурами

В методах для добавления, редактирования и удаления мы покажем, как работать с хранимыми процедурами в Entity Framework. Например, метод для добавления новой записи выглядит так:

```
private void btnAddInvoiceLine_Click(object sender, EventArgs e) {
    var dbContext = AppVariables.getDbContext();
    // получаем текущую счёт-фактуру
    var invoice = dbContext.INVOICES.Find(this.CurrentInvoice.Id);
    // проверяем не оплачена ли счёт-фактура
    if (invoice.PAYED == 1) {
        MessageBox.Show("Невозможно изменение, счёт-фактура оплачена.", "Error");
        return;
    }
    // создаём позицию счёт-фактуры
    var invoiceLine = dbContext.INVOICE_LINES.Create();
    invoiceLine.INVOICE_ID = invoice.INVOICE_ID;
    // создаём редактор позиции счёт фактуры
    using (InvoiceLineEditorForm editor = new InvoiceLineEditorForm()) {
        editor.Text = "Add invoice line";
        editor.InvoiceLine = invoiceLine;
        // Обработчик закрытия формы
        editor.FormClosing += delegate (object fSender, FormClosingEventArgs fe) {
            if (editor.DialogResult == DialogResult.OK) {
              try {
                 // создаём параметры ХП
                 var invoiceIdParam = new FbParameter("INVOICE_ID", FbDbType.Integer);
                 var productIdParam = new FbParameter("PRODUCT_ID", FbDbType.Integer);
                 var quantityParam = new FbParameter("QUANTITY", FbDbType.Integer);
                 // инициализируем параметры значениями
```

```
                invoiceIdParam.Value = invoiceLine.INVOICE_ID;
                productIdParam.Value = invoiceLine.PRODUCT_ID;
                quantityParam.Value = invoiceLine.QUANTITY;
                // выполняем хранимую процедуру
                dbContext.Database.ExecuteSqlCommand(
                  "EXECUTE PROCEDURE SP_ADD_INVOICE_LINE("
                       + "@INVOICE_ID, @PRODUCT_ID, @QUANTITY)",
                  invoiceIdParam,
                  productIdParam,
                  quantityParam);
                // обновляем гриды
                // перезагрузка текущей записи счёт-фактуры
                CurrentInvoice.Load(invoice.INVOICE_ID);
                // перезагрузка всех записей детейл грида
                LoadInvoiceLineData(invoice.INVOICE_ID);
                // обновляем связанные данные
                masterBinding.ResetCurrentItem();
            }
            catch (Exception ex) {
                // отображаем ошибку
                MessageBox.Show(ex.Message, "Error");
                // не закрываем форму для возможности исправления ошибки
                fe.Cancel = true;
            }
        }
    };
    // показываем модальную форму
    editor.ShowDialog(this);
  }
}
```

Здесь обновление записи мастер грида требуется потому, что одно из его полей (TotalSale) содержит агрегированную информацию по строкам документа.

Метод для обновления записи реализован так.

```
private void btnEditInvoiceLine_Click(object sender, EventArgs e) {
    var dbContext = AppVariables.getDbContext();
    // получаем текущую счёт-фактуру
    var invoice = dbContext.INVOICES.Find(this.CurrentInvoice.Id);
    // проверяем не оплачена ли счёт-фактура
    if (invoice.PAYED == 1) {
        MessageBox.Show("Изменение не возможно, счёт фактура оплачена.", "Error");
        return;
    }
    // получаем текущую позицию счёт-фактуры
    var invoiceLine = invoice.INVOICE_LINES
        .Where(p => p.INVOICE_LINE_ID == this.CurrentInvoiceLine.Id)
        .First();
    // создаём редактор позиции счёт фактуры
    using (InvoiceLineEditorForm editor = new InvoiceLineEditorForm()) {
        editor.Text = "Edit invoice line";
        editor.InvoiceLine = invoiceLine;
```

```
        // Обработчик закрытия формы
        editor.FormClosing += delegate (object fSender, FormClosingEventArgs fe) {
           if (editor.DialogResult == DialogResult.OK) {
               try {
                   // создаём параметры ХП
                   var idParam = new FbParameter("INVOICE_LINE_ID", FbDbType.Integer);
                   var quantityParam = new FbParameter("QUANTITY", FbDbType.Integer);
                   // инициализируем параметры значениями
                   idParam.Value = invoiceLine.INVOICE_LINE_ID;
                   quantityParam.Value = invoiceLine.QUANTITY;
                   // выполняем хранимую процедуру
                   dbContext.Database.ExecuteSqlCommand(
                      "EXECUTE PROCEDURE SP_EDIT_INVOICE_LINE("
                         + "@INVOICE_LINE_ID, @QUANTITY)",
                      idParam,
                      quantityParam);
                   // обновляем гриды
                   // перезагрузка текущей записи счёт-фактуры
                   CurrentInvoice.Load(invoice.INVOICE_ID);
                   // перезагрузка всех записей детейл грида
                   LoadInvoiceLineData(invoice.INVOICE_ID);
                   // обновляем связанные контролы
                   masterBinding.ResetCurrentItem();
               }
               catch (Exception ex) {
                   // отображаем ошибку
                   MessageBox.Show(ex.Message, "Error");
                   // не закрываем форму для возможности исправления ошибки
                   fe.Cancel = true;
               }
           }
        };

        // показываем модальную форму
        editor.ShowDialog(this);
    }
}
```

Удаление позиции счёт-фактуры

Метод для удаления записи реализован так.

```
private void btnDeleteInvoiceLine_Click(object sender, EventArgs e) {
    var result = MessageBox.Show(
                 "Вы действительно хотите удалить строку счёт-фактуры?",
                 "Подтверждение",
                 MessageBoxButtons.YesNo,
                 MessageBoxIcon.Question);
    if (result == DialogResult.Yes) {
       var dbContext = AppVariables.getDbContext();
       // получаем текущую счёт-фактуру
       var invoice = dbContext.INVOICES.Find(this.CurrentInvoice.Id);
       try {
           // проверяем не оплачена ли счёт-фактура
```

```
            if (invoice.PAYED == 1)
                throw new Exception("Не возможно удалить запись, счёт-фактура оплачена.");
            // создаём параметры ХП
            var idParam = new FbParameter("INVOICE_LINE_ID", FbDbType.Integer);
            // инициализируем параметры значениями
            idParam.Value = this.CurrentInvoiceLine.Id;
            // выполняем хранимую процедуру
            dbContext.Database.ExecuteSqlCommand(
                "EXECUTE PROCEDURE SP_DELETE_INVOICE_LINE(@INVOICE_LINE_ID)",
                idParam);

            // обновляем гриды
            // перезагрузка текущей записи счёт-фактуры
            CurrentInvoice.Load(invoice.INVOICE_ID);
            // перезагрузка всех записей детейл грида
            LoadInvoiceLineData(invoice.INVOICE_ID);
            // обновляем связанные контролы
            masterBinding.ResetCurrentItem();
        }
        catch (Exception ex) {
            // отображаем ошибку
            MessageBox.Show(ex.Message, "Error");
        }
    }
}
```

Выбор из справочника товаров

В методах для добавления и редактирования позиций счёт-фактуры мы использовали форму для редактирования.

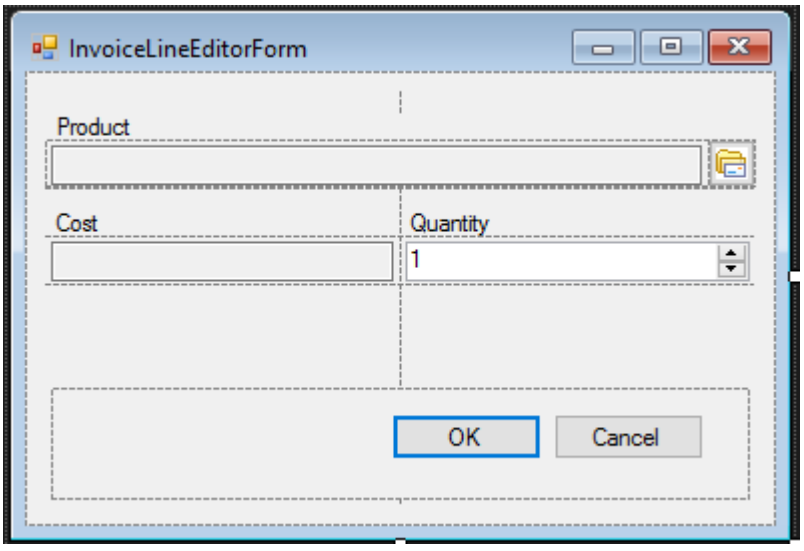

Рис. 3.18. Форма редактора позиций счёт-фактур

Для отображения товара мы будем использовать `TextBox`. По нажатию кнопки, расположенной рядом, будет вызываться модальная форма с гридом для выбора товара. В качестве модальной окна для выбора продукта используем ту же форму, что была создана для их отображения. Код обработчика нажатия кнопки и инициализации формы будет выглядеть следующим образом:

```
public partial class InvoiceLineEditorForm : Form {
    public InvoiceLineEditorForm() {
        InitializeComponent();
    }


    public INVOICE_LINE InvoiceLine { get; set; }

    private void InvoiceLineEditorForm_Load(object sender, EventArgs e) {
        if (this.InvoiceLine.PRODUCT != null) {
           edtProduct.Text = this.InvoiceLine.PRODUCT.NAME;
           edtPrice.Text = this.InvoiceLine.PRODUCT.PRICE.ToString("F2");
           btnChooseProduct.Click -= this.btnChooseProduct_Click;
        }
        if (this.InvoiceLine.QUANTITY == 0)
           this.InvoiceLine.QUANTITY = 1;
        edtQuantity.DataBindings.Add("Value", this.InvoiceLine, "QUANTITY");
    }

    private void btnChooseProduct_Click(object sender, EventArgs e) {
       GoodsForm goodsForm = new GoodsForm();
       if (goodsForm.ShowDialog() == DialogResult.OK) {
          InvoiceLine.PRODUCT_ID = goodsForm.CurrentProduct.Id;
          edtProduct.Text = goodsForm.CurrentProduct.Name;
          edtPrice.Text = goodsForm.CurrentProduct.Price.ToString("F2");
       }
    }
}
```

Работа с транзакциями

Когда мы вызываем при добавлении, обновлении, удалении метод `SaveChanges()`, то фактически Entity Framework неявно стартует и завершает транзакцию. Поскольку используется отсоединённая модель, то все операции происходят в рамках одной транзакции. Кроме того EF автоматически стартует и завершает транзакцию при каждом извлечении данных. Рассмотрим работу автоматических транзакций на следующем примере. Допустим нам необходимо сделать скидку на товары, выделенные в гриде. Код без явного использования транзакций будет выглядеть следующим образом:

```
var dbContext = AppVariables.getDbContext();
foreach (DataGridViewRow gridRows in dataGridView.SelectedRows) {
    int id = (int)gridRows.Cells["Id"].Value;
    // здесь происходит неявный старт и завершение транзакции
    var product = dbContext.PRODUCTS.Find(id);
    // скидка 10%
    decimal discount = 10.0m;
    product.PRICE = product.PRICE * (100 - discount) /100;
}
// здесь происходит неявный старт и завершение транзакции
```

```
// все изменения происходят за одну транзакцию
dbContext.SaveChanges();
```

Допустим, мы выбрали 10 товаров. В этом случае будет неявно использовано 10 транзакций для поиска товара по идентификатору и одиннадцатая для сохранения изменений. В данном случае можно использовать всего одну транзакцию, если использовать явное управление транзакциями. Например, вот так:

```
var dbContext = AppVariables.getDbContext();
// явный старт транзакции по умолчанию
using (var dbTransaction = dbContext.Database.BeginTransaction()) {
    string sql =
         "UPDATE PRODUCT " +
         "SET PRICE =  PRICE * ROUND((100 - @DISCOUNT)/100, 2) " +
         "WHERE PRODUCT_ID = @PRODUCT_ID";
    try {
        // создаём параметры запроса
        var idParam = new FbParameter("PRODUCT_ID", FbDbType.Integer);
        var discountParam = new FbParameter("DISCOUNT", FbDbType.Decimal);
        // создаём SQL команду для обновления записей
        var sqlCommand = dbContext.Database.Connection.CreateCommand();
        sqlCommand.CommandText = sql;
        // указываем команде, какую транзакцию использовать
        sqlCommand.Transaction = dbTransaction.UnderlyingTransaction;
        sqlCommand.Parameters.Add(discountParam);
        sqlCommand.Parameters.Add(idParam);
        // подготавливаем команду
        sqlCommand.Prepare();
        // для всех выделенных записей в гриде
        foreach (DataGridViewRow gridRows in dataGridView.SelectedRows) {
            int id = (int)gridRows.Cells["Id"].Value;
            // инициализируем параметры запроса
            idParam.Value = id;
            discountParam.Value = 10.0m; // скидка 10%
            // выполняем sql запрос
            sqlCommand.ExecuteNonQuery();
        }
        dbTransaction.Commit();
    }
    catch (Exception ex) {
        dbTransaction.Rollback();
        MessageBox.Show(ex.Message, "error");
    }
}
```

В данном случае мы стартовали транзакцию с параметрами по умолчанию. Для того чтобы задавать свои параметры транзакции необходимо использовать метод `UseTransaction`.

```
private void btnDiscount_Click(object sender, EventArgs e) {
    DiscountEditorForm editor = new DiscountEditorForm();
```

```
    editor.Text = "Enter discount";
    if (editor.ShowDialog() != DialogResult.OK)
        return;


    bool needUpdate = false;

    var dbContext = AppVariables.getDbContext();
    var connection = dbContext.Database.Connection;
    // явный старт транзакции
    using (var dbTransaction = connection.BeginTransaction(IsolationLevel.Snapshot)) {
        dbContext.Database.UseTransaction(dbTransaction);
        string sql =
            "UPDATE PRODUCT " +
            "SET PRICE =  ROUND(PRICE * (100 - @DISCOUNT)/100, 2) " +
            "WHERE PRODUCT_ID = @PRODUCT_ID";
        try {
            // создаём параметры запроса
            var idParam = new FbParameter("PRODUCT_ID", FbDbType.Integer);
            var discountParam = new FbParameter("DISCOUNT", FbDbType.Decimal);
            // создаём SQL команду для обновления записей
            var sqlCommand = connection.CreateCommand();
            sqlCommand.CommandText = sql;
            // указываем команде, какую транзакцию использовать
            sqlCommand.Transaction = dbTransaction;
            sqlCommand.Parameters.Add(discountParam);
            sqlCommand.Parameters.Add(idParam);
            // подготавливаем команду
            sqlCommand.Prepare();
            // для всех выделенных записей в гриде
            foreach (DataGridViewRow gridRows in dataGridView.SelectedRows) {
                int id = (int)gridRows.Cells["PRODUCT_ID"].Value;
                // инициализируем параметры запроса
                idParam.Value = id;
                discountParam.Value = editor.Discount;
                // выполняем sql запрос
                needUpdate = (sqlCommand.ExecuteNonQuery() > 0) || needUpdate;
            }
            dbTransaction.Commit();
        }
        catch (Exception ex) {
            dbTransaction.Rollback();
            MessageBox.Show(ex.Message, "error");
            needUpdate = false;
        }
    }
    // перезагружаем содержимое грида
    if (needUpdate) {
       // для всех выделенных записей в гриде
       foreach (DataGridViewRow gridRows in dataGridView.SelectedRows) {
            var product = (PRODUCT)bindingSource.List[gridRows.Index];
            dbContext.Refresh(RefreshMode.StoreWins, product);
       }
       bindingSource.ResetBindings(false);
    }

}
```

Ну вот. Теперь у нас для всего набора обновлений используется всего одна транзакция, и нет лишних команд для поиска данных. Осталось только добавить диалог для ввода значения скидки и обновление данных в гриде. Попробуйте сделать это самостоятельно.

Результат

В заключении приведём скриншот готового приложения.

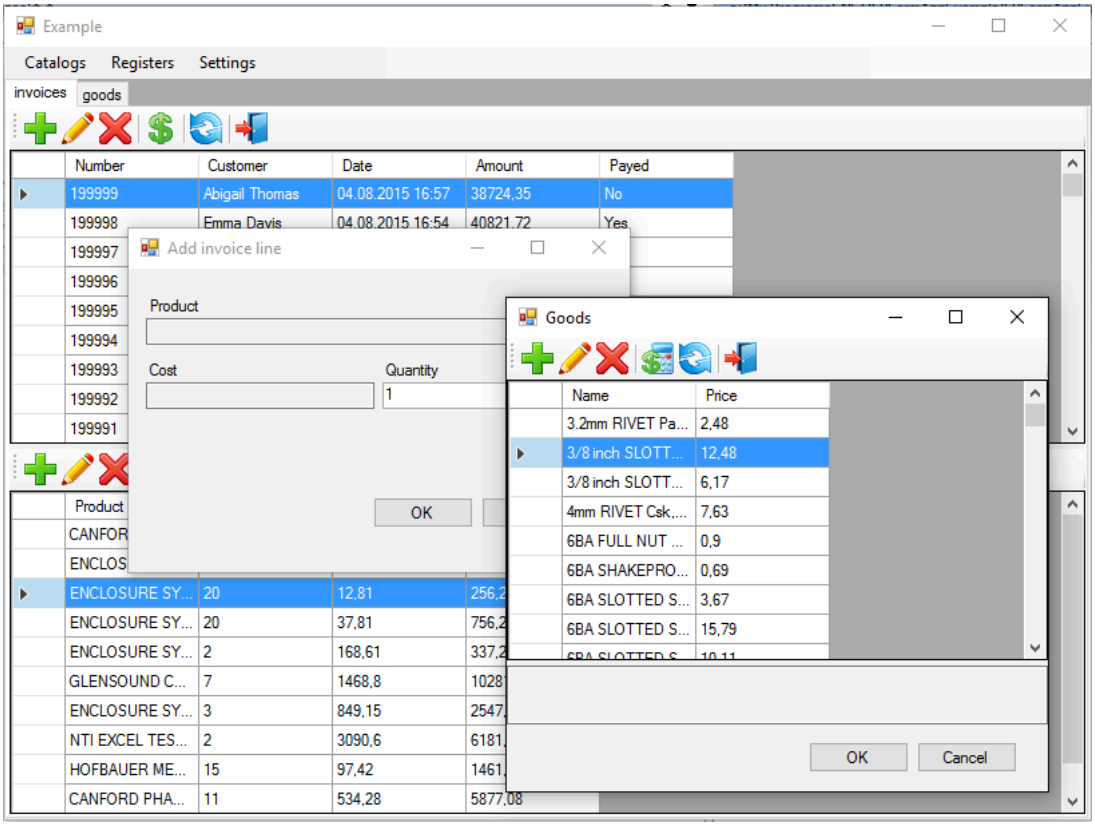

Рис. 3.19. Скриншот готового приложения

Исходный код

Исходные коды примера приложения вы можете скачать по ссылке https://github.com/sim1984/FBFormAppExample.

Глава 4

Создание Web приложений с использованием Entity Framework

В данной главе будет описан процесс создания Web приложений для СУБД Firebird с использованием Entity Framework и среды Visual Studio 2015.

В данной главе обсуждаются особенности создания именно Web приложений, базовые принципы работы с Entity Framework и Firebird описаны в предыдущей главе Создание приложений с использованием Entity Framework.

.NET Frameworks

Платформа .NET предоставляет два основных фреймворка для создания web приложений: ASP.NET Web Forms и ASP.NET MVC. Я предпочитаю использовать паттерн MVC, поэтому в дальнейшем будет описываться именно эта технология.

ASP.NET MVC Platform

Платформа **ASP.NET MVC** представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна MVC.

Концепция паттерна (шаблона) MVC (model-view-controller) предполагает разделение приложения на три компонента:

- **Контроллер** (controller). Контроллеры осуществляют взаимодействие с пользователем, работу с моделью, а также выбор представления, отображающего пользовательский интерфейс. В приложении MVC представления только отображают данные, а контроллер обрабатывает вводимые данные и отвечает на действия пользователя. Например, контроллер может обрабатывать строковые значения запроса и передавать их в модель, которая может использовать эти значения для отправки запроса в базу данных.

- **Представление** (view) — это собственно визуальная часть или пользовательский интерфейс приложения. Пользовательский интерфейс обычно создаётся на основе данных модели.

- **Модель** (model). Объекты моделей являются частями приложения, реализующими логику для работы данными приложения. Объекты моделей часто получают и сохраняют состояние модели в базе данных.

Взаимодействие Model-View-Controller

Общую схему взаимодействия этих компонентов можно представить следующим образом:

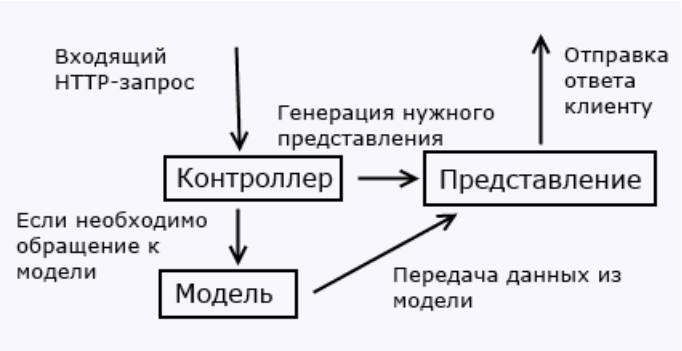

Рис. 4.1. Структура MVC паттерна

Шаблон MVC позволяет создавать приложения, различные аспекты которых (логика ввода, бизнес-логика и логика интерфейса) разделены, но достаточно тесно взаимодействуют друг с другом. Эта схема указывает расположение каждого вида логики в приложении. Пользовательский интерфейс располагается в представлении. Логика ввода располагается в контроллере. Бизнес-логика находится в модели. Это разделение позволяет работать со сложными структурами при создании приложения, так как обеспечивает одновременную реализацию только одного аспекта. Например, разработчик может сконцентрироваться на создании представления отдельно от бизнес-логики.

Более полную информацию о технологии ASP.NET MVC вы можете найти на сайте сообщества ASP.NET .

Программный стек

Помимо библиотек для работы с Firebird, Entity Framework и MVC.NET нам потребуется множество JavaScript библиотек для поддержки отзывчивого интерфейса, таких как jquery, jquery-ui, Bootstrap, jqGrid. В этом примере мы постарались приблизить интерфейс веб-приложения к настольным приложениям, активно применяя грибы для отображения и модальные окна для ввода данных.

Подготовка Visual Studio 2015 для работы с Firebird

Для работы Visual Studio с СУБД Firebird вам придётся проделать несколько дополнительных шагов, которые подробно были описаны в предыдущей главе "Создание приложений с использованием Entity Framework" в разделе Подготовка Visual Studio 2015 для работы с Firebird.

Создание проекта

Посмотрим как создаётся каркас MVC.NET приложения с помощью мастеров Visual Studio.

Итак, откроем Visual Studio 2015 Файл->Создать->Проект и создадим новый проект. Назовём новый проект FBMVCExample.

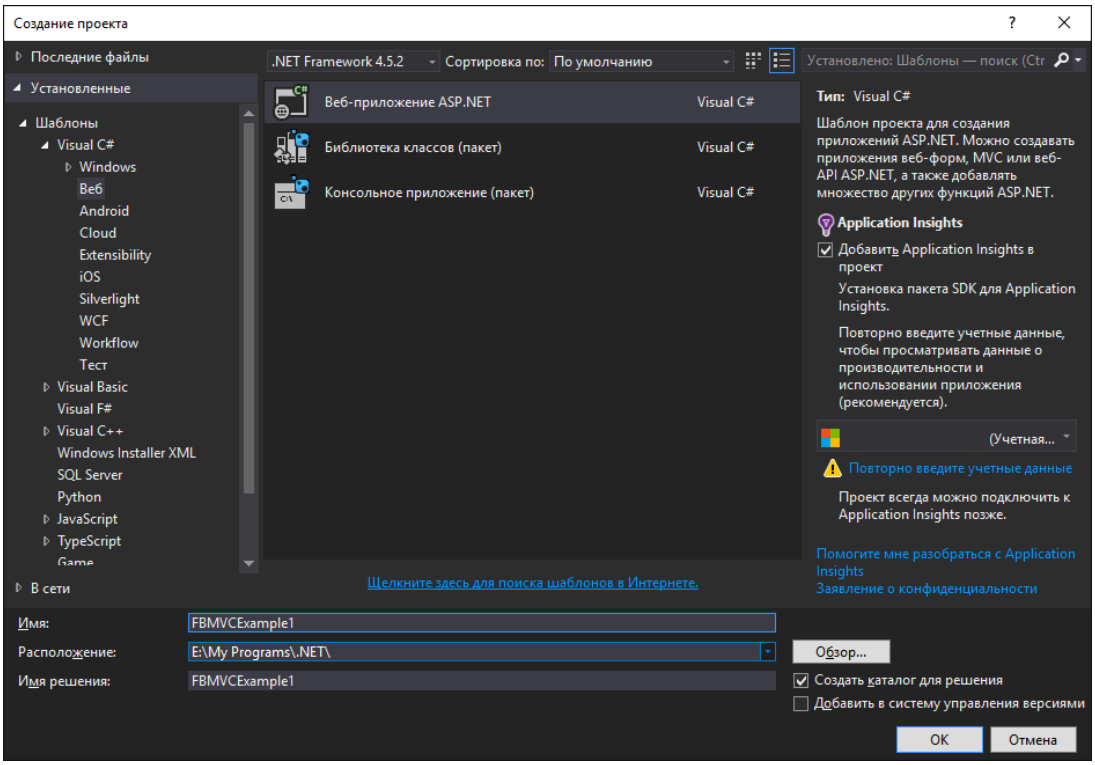

Рис. 4.2. Создание проекта

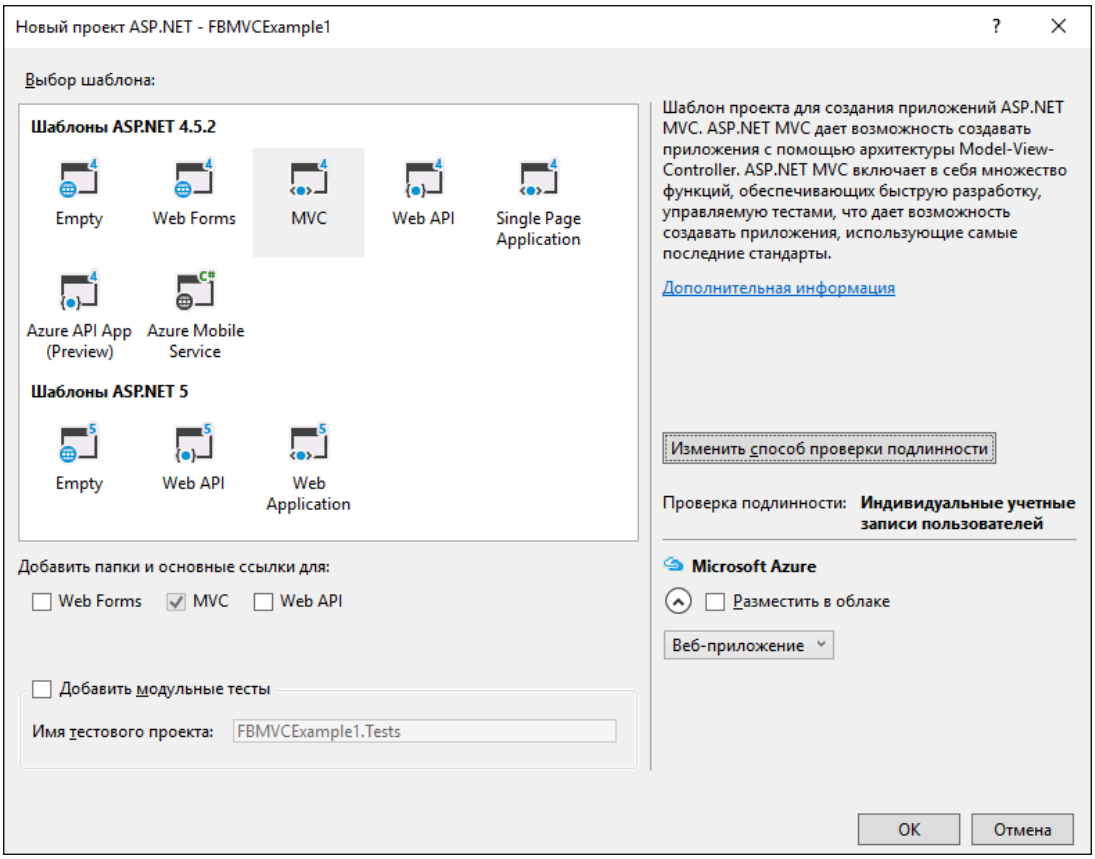

Рис. 4.3. Создание MVC.NET проекта

Изменим способ проверки подлинности. В данный момент создадим веб приложение без проверки подлинности. К этому вопросу мы вернёмся чуть позже.

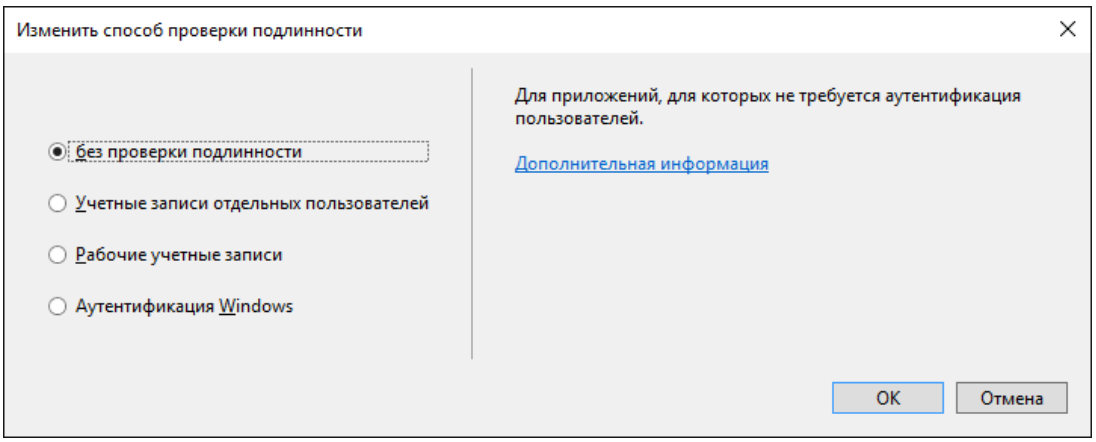

Рис. 4.4. Выбор способа проверки подлинности

Структура проекта

После этого будет создан проект, который практически не обладает никакой функциональностью, хотя уже имеет базовую структуру.

Таблица 4.1. Структура папок MVC.NET проекта

Папка или файл	Описание
`/App_Data`	В эту папку помещаются закрытые данные веб приложения, такие как XML-файлы или файлы базы данных.
`/App_Start`	Эта папка содержит ряд основных настроек конфигурации для проекта, в том числе определение маршрутов и фильтров.
`/Content`	Сюда помещается статическое содержимое, такое как CSS-файлы и изображения. Это является необязательным соглашением. Вы можете хранить файлы стилей в любом подходящем месте.
`/Controllers`	Сюда помещаются классы контроллеров. Это необязательное соглашение. Вы можете классы контроллеров где угодно.
`/Models`	Сюда помещаются классы моделей представлений и моделей предметной области, хотя все кроме простейших приложений выигрывают от определения модели предметной области в отдельном проекте. Это необязательное соглашение. Вы можете размещать классы моделей в любом удобном месте.
`/Scripts`	Эта папка предназначена для хранения библиотек JavaScript, используемых в приложении. По умолчанию Visual Studio добавляет библиотеки jQuery и несколько

Папка или файл	Описание
	других популярных JavaScript-библиотек. Это необязательное соглашение.
`/Views`	В этой папке хранятся представления и частичные представления, обычно сгруппированные вместе в папках с именами контроллеров, с которыми они связаны.
`/Views/Shared`	В этой папке хранятся компоновки и представления, не являющиеся специфичными для какого-либо контроллера.
`/Views/Web.config`	Это конфигурационный файл. В нем содержится конфигурационная информация, которая обеспечивает обработку представлений с помощью ASP.NET и предотвращает их обслуживание веб-сервером IIS, а также пространства имён, по умолчанию импортируемые в представления.
`/Global.asax`	Это глобальный класс приложения ASP.NET. В файле его кода (`Global.asax.cs`) регистрируется конфигурация маршрутов, а также предоставляется любой код, который должен выполняться при запуске или завершении приложения либо в случае возникновения необработанного исключения.
`/Web.config`	Конфигурационный файл для приложения.

Добавление отсутствующих пакетов

Теперь добавим необходимые пакеты с помощью менеджера пакетов NuGet. Нам потребуются установить недостающие пакеты:

- FirebirdSql.Data.FirebirdClient
- EntityFramework (автоматически добавлен мастером)
- EntityFramework.Firebird
- Bootstrap (автоматически добавлен мастером)
- jQuery (автоматически добавлен мастером)
- jQuery.UI.Combined
- Respond (автоматически добавлен мастером)
- Newtonsoft.Json
- Moderninzr (автоматически добавлен мастером)
- Trirand.jqGrid

> **Примечание**
>
> Не все пакеты, предоставляемые NuGet, являются библиотеками последних версий. Особенно это касается JavaScript библиотек. Вы можете подключать последние версии JavaScript библиотек, используя CDN или просто скачать их, заменив библиотеки, предоставленные NuGet.

Для этого необходимо щёлкнуть правой клавишей мыши по имени проекта в обозревателе решений и в выпадающем меню выбрать пункт «Управление пакетами NuGet».

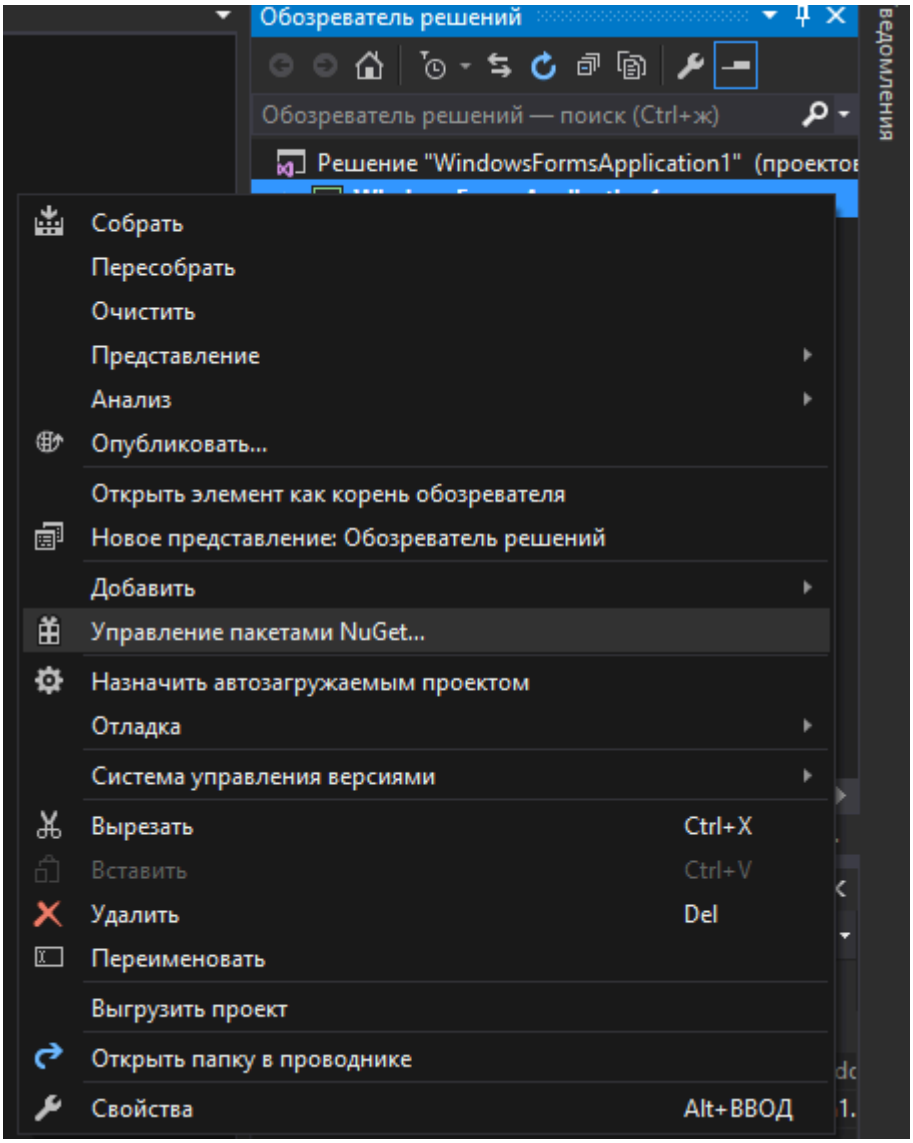

Рис. 4.5. Контекстное меню обозревателя решений

В появившемся менеджере пакетов произвести поиск и установку необходимых пакетов.

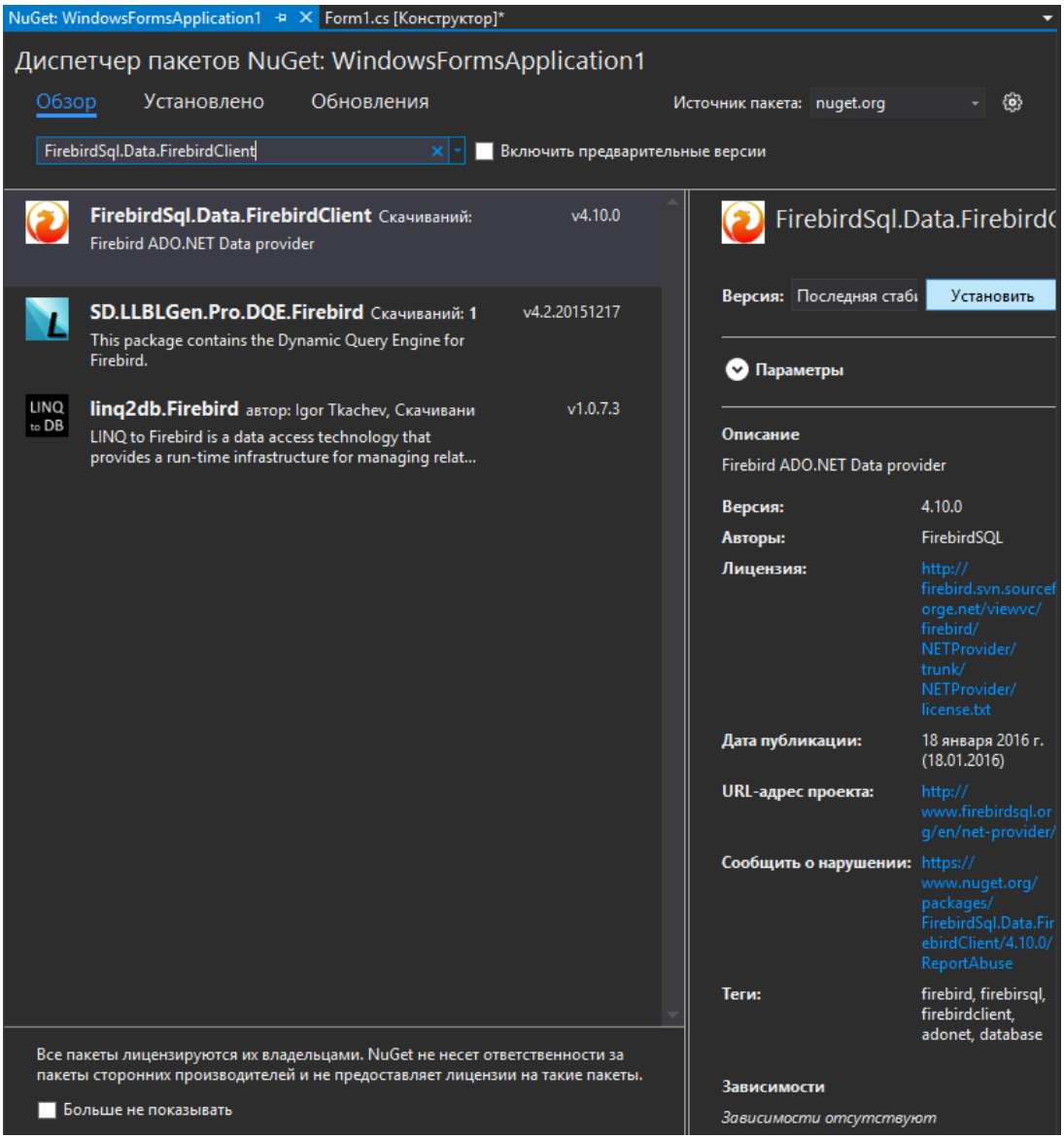

Рис. 4.6. Менеджер пакетов NuGet

Создание EDM модели

Если у вас уже есть Windows Forms приложение, которое использует Entity Framework, то вы просто можете перенести кассы моделей в папку `Models`. В противном случае вам необходимо будет создать их с нуля. Подробно процесс создания EDM модели описан в предыдущей главе «Создание приложений с использованием Entity Framework» (см. раздел «Создание EDM модели»).

Существует одно небольшое отличие. В процессе работы мастера создания модели у вас спросят, как хранить строку подключения.

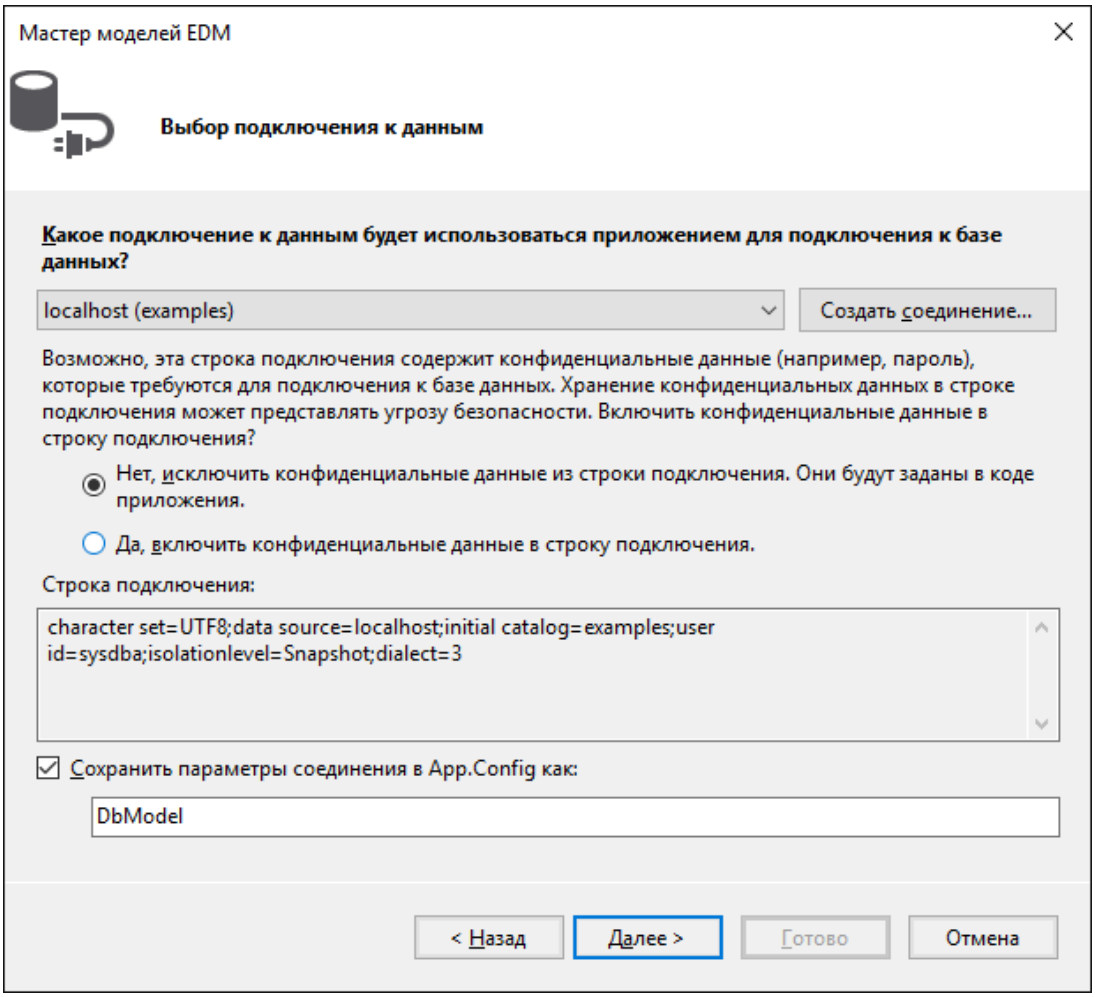

Рис. 4.7. Сохранение строки подключения

Поскольку мы создаём веб приложение, где все пользователи будут работать с базой данных под одной и той же учётной записью, то смело выбираем «Да». В качестве имени пользователя может быть указан любой пользователь с достаточными привилегиями. Желательно не использовать пользователя SYSDBA, поскольку он обладает повышенными привилегиями, которые не требуются для функционирования веб приложения. Вы всегда можете это изменить в готовом приложении, просто отредактировав строку подключения в файле конфигурации приложения *AppName*`.exe.conf`. Строка подключения будет сохранена в секции connectionStrings примерно в таком виде

```
<add name="DbModel"
     connectionString="character set=UTF8; data source=localhost;
                       initial catalog=examples; port number=3050;
                       user id=sysdba; dialect=3; isolationlevel=Snapshot;
                       pooling=True; password=masterkey;"
     providerName="FirebirdSql.Data.FirebirdClient" />
```

Создание пользовательского интерфейса справочников

Создание контроллера заказчиков

Итак, создадим наш первый контроллер. Он будет служить для отображения и ввода данных о поставщиках.

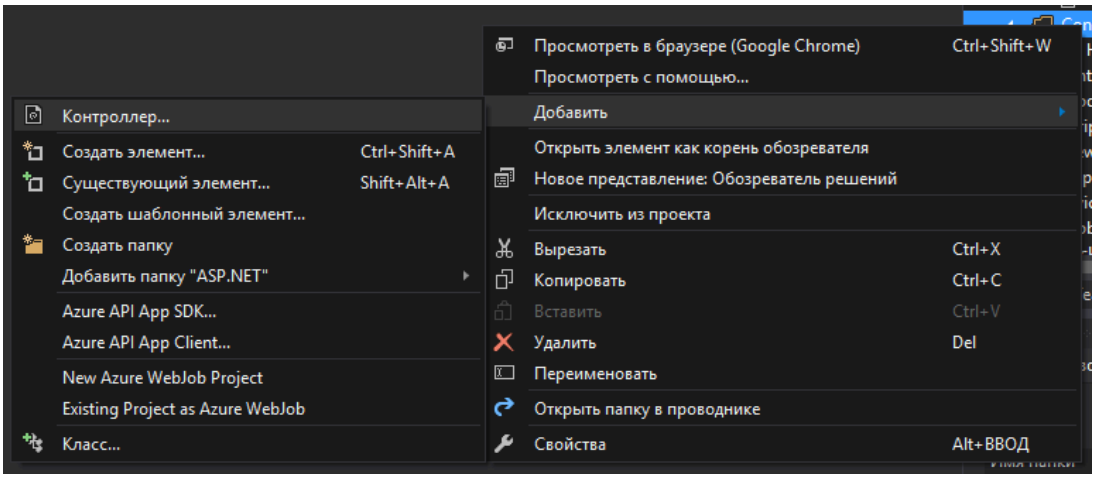

Рис. 4.8. Меню добавления контроллера

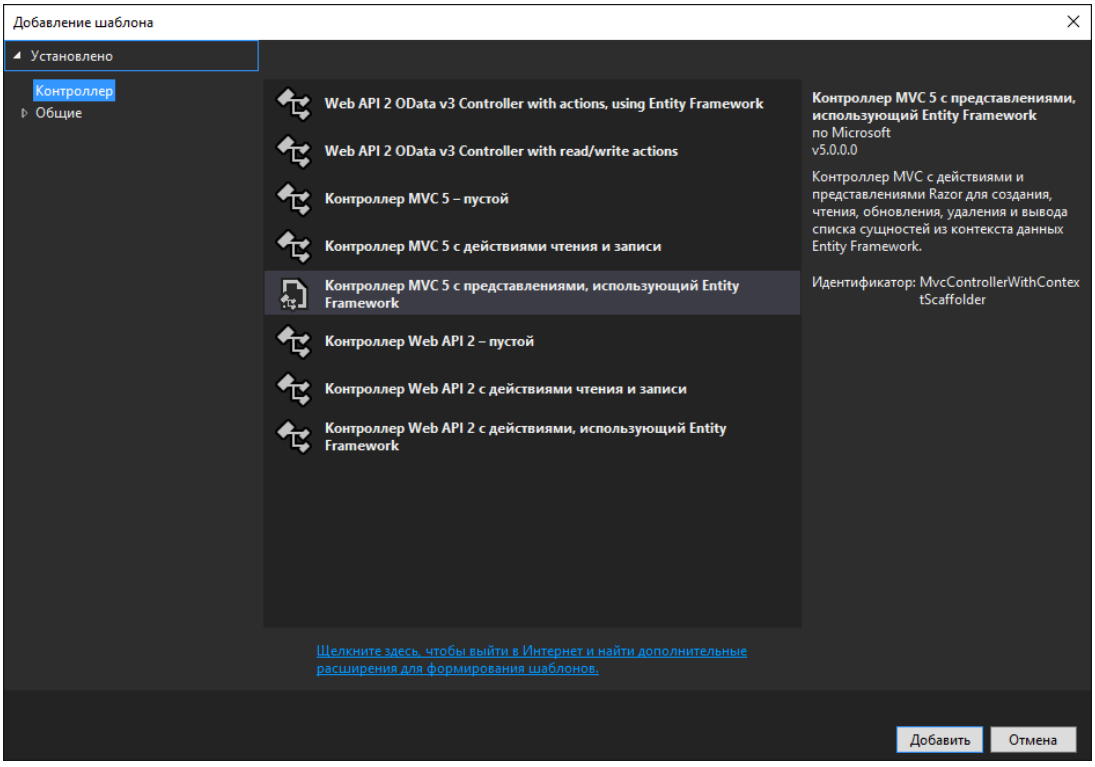

Рис. 4.9. Добавление контроллера

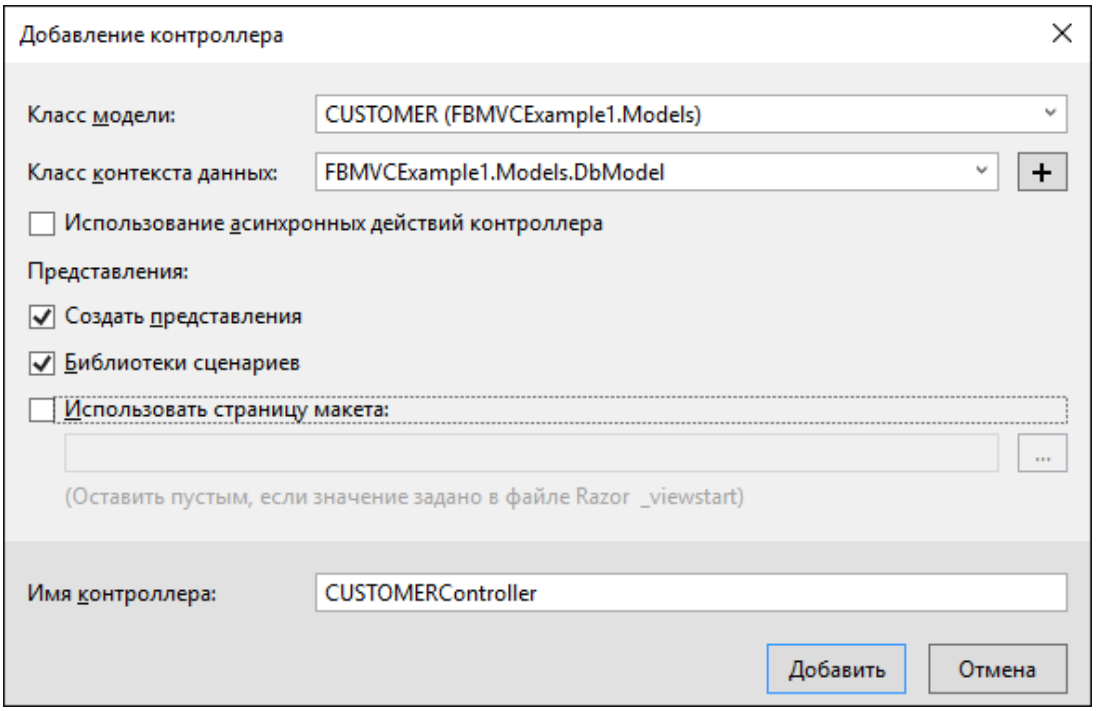

Рис. 4.10. Мастер создания контроллера

После этих действий будет создан контроллер `CustomerController` и 5 представлений:

1. для отображения списка поставщиков
2. детализации поставщика
3. форма для добавления поставщика
4. форма для редактирования поставщика
5. форма для удаления поставщика

Поскольку мы будем активно применять технологию Ajax и библиотеку jqGrid, то нам будет достаточно всего одного представления для отображения списка поставщиков в виде таблицы, остальные действия будут выполняться с помощью jqGrid.

Уменьшение накладных расходов

При разработке web-приложений необходимо понимать как уменьшить накладные расходы, свзяанные с передачей данных по глобальной сети и подключениями к базе данных. Существуют методы которые могут нам в этом.

Ограничение объёма возвращаемых данных

Список поставщиков может оказаться довольно большим. В отличие от настольных приложений в Web приложениях обычно не принято возвращать весь большой список, потому что это может сильно замедлить загрузку страницы. Вместо этого обычно используют постраничное разбиение данных, или динамическую дозагрузку данных, когда при прокрутке пользователь достигает конца страницы (или грида). В нашем примере мы воспользуемся первым вариантом.

Уменьшение количества подключений к базе данных

Ещё одной особенностью при создании веб приложений является то, что в них отсутствует постоянное соединение с базой данных. Это обусловлено тем, что сам скрипт формирования страницы «живёт» не дольше чем время для формирования ответа на запрос пользователя. Само по себе соединение с базой данных — это довольно дорогой ресурс, поэтому его надо экономить. Конечно, для уменьшения времени установления соединения с базой данных придумали пул соединений, но всё равно желательно, чтобы соединение с базой данных происходило только тогда когда это действительно необходимо.

Современные браузеры помогут нам

Одним из способов снижения количества взаимодействий с базой данных является перенос проверки правильности введённых данных на сторону браузера. К счастью современные HTML5 и JavaScript библиотеки могут это делать. Например, вы можете проверять обязательность поля на форме ввода, или максимальную длину строковых полей.

Адаптация контроллера для работы с jqGrid

Итак давайте изменим контроллер `CustomerController` для того чтобы он работал с jqGrid. В тексте контроллера сделаны поясняющие комментарии.

```
public class CustomerController : Controller
{
    private DbModel db = new DbModel();

    // Отображение представления
    public ActionResult Index()
    {
        return View();
    }

    // Получение данных в виде JSON для грида
    public ActionResult GetData(int? rows, int? page, string sidx, string sord,
        string searchField, string searchString, string searchOper)
    {
        // получаем номер страницы, количество отображаемых данных
        int pageNo = page ?? 1;
        int limit = rows ?? 20;
        // вычисляем смещение
        int offset = (pageNo - 1) * limit;

        // строим запрос для получения поставщиков
        var customersQuery =
            from customer in db.CUSTOMERS
            select new
            {
                CUSTOMER_ID = customer.CUSTOMER_ID,
                NAME = customer.NAME,
                ADDRESS = customer.ADDRESS,
                ZIPCODE = customer.ZIPCODE,
                PHONE = customer.PHONE
            };
        // добавлением в запрос условия поиска, если он производится
```

```
        if (searchField != null)
        {
            switch (searchOper)
            {
                case "eq":
                    customersQuery = customersQuery.Where(
                        c => c.NAME == searchString);
                    break;
                case "bw":
                    customersQuery = customersQuery.Where(
                        c => c.NAME.StartsWith(searchString));
                    break;
                case "cn":
                    customersQuery = customersQuery.Where(
                        c => c.NAME.Contains(searchString));
                    break;
            }
        }
        // получаем общее количество поставщиков
        int totalRows = customersQuery.Count();
        // добавляем сортировку
        switch (sord) {
            case "asc":
                customersQuery = customersQuery.OrderBy(
                    customer => customer.NAME);
                break;
            case "desc":
                customersQuery = customersQuery.OrderByDescending(
                    customer => customer.NAME);
                break;
        }

        // получаем список поставщиков
        var customers = customersQuery
             .Skip(offset)
             .Take(limit)
             .ToList();

        // вычисляем общее количество страниц
        int totalPages = totalRows / limit + 1;

        // создаём результат для jqGrid
        var result = new
        {
            page = pageNo,
            total = totalPages,
            records = totalRows,
            rows = customers
        };
        // преобразуем результат в JSON
        return Json(result, JsonRequestBehavior.AllowGet);
    }

    // Добавление нового поставщика
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(
      [Bind(Include = "NAME,ADDRESS,ZIPCODE,PHONE")] CUSTOMER customer)
```

```
    {
        // проверяем правильность модели
        if (ModelState.IsValid)
        {
            // получаем новый идентификатор с помощью генератора
            customer.CUSTOMER_ID = db.NextValueFor("GEN_CUSTOMER_ID");
            // добавляем модель в список
            db.CUSTOMERS.Add(customer);
            // сохраняем модель
            db.SaveChanges();
            // возвращаем успех в формате JSON
            return Json(true);
        }
        else {
            // соединяем ошибки модели в одну строку
            string messages = string.Join("; ", ModelState.Values
                                    .SelectMany(x => x.Errors)
                                    .Select(x => x.ErrorMessage));
            // возвращаем ошибку в формате JSON
            return Json(new { error = messages });
        }
    }

    // Редактирование поставщика
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit(
      [Bind(Include = "CUSTOMER_ID,NAME,ADDRESS,ZIPCODE,PHONE")] CUSTOMER customer)
    {
        // проверяем правильность модели
        if (ModelState.IsValid)
        {
            // помечаем модель как изменённую
            db.Entry(customer).State = EntityState.Modified;
            // сохраняем модель
            db.SaveChanges();
            // возвращаем успех в формате JSON
            return Json(true);
         }
         else {
            // соединяем ошибки модели в одну строку
            string messages = string.Join("; ", ModelState.Values
                                    .SelectMany(x => x.Errors)
                                    .Select(x => x.ErrorMessage));
            // возвращаем ошибку в формате JSON
            return Json(new { error = messages });
         }
    }

    // Удаление поставщика
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Delete(int id)
    {
        // ищем поставщика по идентификатору
        CUSTOMER customer = db.CUSTOMERS.Find(id);
        // удаляем поставщика
        db.CUSTOMERS.Remove(customer);
```

```
        // сохраняем модель
        db.SaveChanges();
        // возвращаем успех в формате JSON
        return Json(true);
    }

    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            db.Dispose();
        }
        base.Dispose(disposing);
    }
}
```

Метод `Index` служит для отображения представления `Views/Cusomter/Index.cshtml`. Само представление будет представлено чуть позже. В общем, это представление представляет собой шаблон html страницы с разметкой и JavaScript для инициализации jqGrid. Сами данные будут получены в асинхронном режиме в формате JSON с помощью технологии Ajax. В зависимости от выбранной сортировки, номера страницы и параметров поиска формируется HTTP запрос, который будет обработан действием `GetData`. Параметры http запроса отображаются на входные аргументы метода `GetData`. В соответствии с этими параметры мы формируем LINQ запрос, и отправляем полученный результат в формате JSON.

> **Примечание**
>
> Для разбора параметров запроса формируемого jqGrid и упрощения построения модели существуют различные библиотеки. Мы не использовали их в наших примерах, и поэтому код может быть несколько громоздким. Однако вы всегда можете улучшить его.

Метод `Create` предназначен для добавления новой записи о поставщике. Параметры HTTP запроса с типом POST (у метода указан атрибут [HttpPost]) будут отображены на модель `Cusotmer`. Обратите внимание на строку

```
[Bind(Include = "NAME,ADDRESS,ZIPCODE,PHONE")] CUSTOMER customer
```

Здесь Bind указывает, какие параметры HTTP запроса отображать на свойства модели.

Аттрибут ValidateAntiforgeryToken

Обратите внимание на атрибут **ValidateAntiforgeryToken**, он предназначен для противодействия подделке межсайтовых запросов, производя верификацию токенов при обращении к методу действия. Наличие этого атрибута требует чтобы в HTTP запросе присутствовал дополнительный параметр __RequestVerificationToken. Этот параметр автоматически добавляется в каждую форму в которой указан хелпер `@Html.AntiForgeryToken()`. Однако библиотека jqGrid использует динамически формируемые Ajax запросы, а не заранее созданные веб формы. Давайте исправим это. Для этого изменим обобщённое представлении `Views/Shared/_Layout.cshtml` следующим образом

```
<!DOCTYPE html>
<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>@ViewBag.Title - приложение ASP.NET</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/jquery-ui")

    <link href="~/Content/jquery.jqGrid/ui.jqgrid.css"
          rel="stylesheet" type="text/css" />
    <link href="~/Content/jquery.jqGrid/ui.jqgrid-bootstrap.css"
          rel="stylesheet" type="text/css" />
    <link href="~/Content/jquery.jqGrid/ui.jqgrid-bootstrap-ui.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>
</head>
<body>
    @Html.AntiForgeryToken()
    <script>
      // получение AntiForgery токена
      function GetAntiForgeryToken() {
        var tokenField =
          $("input[type='hidden'][name$='RequestVerificationToken']");
        if (tokenField.length == 0) {
            return null;
        } else {
            return {
                name: tokenField[0].name,
                value: tokenField[0].value
            };
        }
      }

      // добавляем префильтр на все ajax запросы
      // он будет добавлять к любому POST ajax запросу
      // AntiForgery токен
      $.ajaxPrefilter(
          function (options, localOptions, jqXHR) {
              if (options.type !== "GET") {
                  var token = GetAntiForgeryToken();
                  if (token !== null) {
                      if (options.data.indexOf("X-Requested-With") === -1) {
                          options.data = "X-Requested-With=XMLHttpRequest"
                              + ((options.data === "") ? "" : "&" + options.data);
                      }
                      options.data = options.data + "&" + token.name + '='
                                  + token.value;
                  }
              }
          }
      );
```

```
      // инициализируем общие свойства модуля jqGrid
      $.jgrid.defaults.width = 780;
      $.jgrid.defaults.responsive = true;
      $.jgrid.defaults.styleUI = 'Bootstrap';
    </script>
    <!-- Навигационное меню -->
    <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>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Поставщики", "Index", "Customer")</li>
                    <li>@Html.ActionLink("Товары", "Index", "Product")</li>
                    <li>@Html.ActionLink("Накладные", "Index", "Invoice")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - приложение ASP.NET</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

Бандлы

Бандлы предназначены для упрощения подключения JavaScript скриптов и файлов стилей. С помощью хелпера Styles.Render подключаются бандлы стилей, а с помощью хелпера Scripts.Render — бандлы скриптов.

Регистрация бандлов осуществляется в файле `BundleConfig.cs`, который находится в папке `App_Start`:

```
public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
                        "~/Scripts/jquery-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
                        "~/Scripts/jquery.validate*"));
```

```
    bundles.Add(new ScriptBundle("~/bundles/jquery-ui").Include(
                        "~/Scripts/jquery-ui-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
                        "~/Scripts/modernizr-*"));

    bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
                      "~/Scripts/bootstrap.js",
                      "~/Scripts/respond.js"));

    bundles.Add(new StyleBundle("~/Content/css").Include(
                      "~/Content/jquery-ui.min.css",
                      "~/Content/themes/ui-darkness/jquery-ui.min.css",
                      "~/Content/themes/ui-darkness/theme.css",
                      "~/Content/bootstrap.min.css",
                      "~/Content/Site.css"
                ));
}
```

Здесь метод `RegisterBundles` добавляет все создаваемые бандлы в коллекцию bundles. Объявление бандла выглядит следующим образом:

```
new ScriptBundle("~/bundles/jquery").Include("~/Scripts/jquery-{version}.js")
```

В конструктор `ScriptBundle` передаётся виртуальный путь бандла. А с помощью метода `Include` в данный бандл включаются конкретные файлы скриптов.

В выражении `"~/Scripts/jquery-{version}.js"` параметр `{version}` является заменителем для любого символьного обозначения версии скрипта. Это очень удобно, поскольку через некоторое время мы можем поменять версию библиотеки, но при этом в коде нам ничего не придётся менять, так как система уже автоматически примет новую версию.

Выражение `"~/Scripts/jquery.validate*"` с помощью знака звёздочки заменяет остальную часть строки. Например, это выражение подключит в бандл сразу два файла: `jquery.validate.js` и `jquery.validate.unobtrusive.js` (и их минимизированные версии), так как их названия начинаются с `jquery.validate*`.

То же самое касается и создания бандлов стилей, только в этом случае используется класс `StyleBundle`.

Во время отладки желательно иметь полные версии скриптов и стилей, а при развёртывании приложения – минифицированные. Бандлы позволяют решить эту задачу. Когда приложение находится в режиме отладки, то файле `web.config` параметр <compilation debug="true">. При изменении этого параметра на значение `false` (режим компиляции Release) вместо полных версий JavaScript модулей и файлов CSS стилей будут использоваться минифицированные.

Представления

Из всех автоматически созданных представлений для контроллера `Customer` нам потребуется только один `View/Customer/Index.cshtml`, остальные можно удалить из этой папки.

```
@{
    ViewBag.Title = "Index";
}

<h2>Customers</h2>

<table id="jqg"></table>
<div id="jqg-pager"></div>

<script type="text/javascript">
    $(document).ready(function () {

        var dbGrid = $("#jqg").jqGrid({
            url: '@Url.Action("GetData")', // url для получения данных
            datatype: "json", // формат получения данных
            mtype: "GET",     // тип http запроса
            // описание модели
            colModel: [
                {
                    label: 'Id',           // подпись
                    name: 'CUSTOMER_ID',   // имя поля
                    key: true,             // признак ключевого поля
                    hidden: true           // скрыт
                },
                {
                    label: 'Name',
                    name: 'NAME',
                    width: 250,            // ширина
                    sortable: true,        // разрешена сортировка
                    editable: true,        // разрешено редактирование
                    edittype: "text",      // тип поля в редакторе
                    search: true,          // разрешён поиск
                    searchoptions: {
                        sopt: ['eq', 'bw', 'cn'] // разрешённые операторы поиска
                    },
                    // размер и максимальная длина для поля ввода
                    editoptions: { size: 30, maxlength: 60 },
                    // говорит о том, что поле обязательное
                    editrules: { required: true }
                },
                {
                    label: 'Address',
                    name: 'ADDRESS',
                    width: 300,
                    sortable: false,    // запрещаем сортировку
                    editable: true,     // редактируемое
                    search: false,      // запрещаем поиск
                    edittype: "textarea",
                    editoptions: { maxlength: 250, cols: 30, rows: 4 }
                },
                {
                    label: 'Zip Code',
                    name: 'ZIPCODE',
                    width: 30,
                    sortable: false,
                    editable: true,
                    search: false,
```

```
                edittype: "text",
                editoptions: { size: 30, maxlength: 10 },
            },
            {
                label: 'Phone',
                name: 'PHONE',
                width: 80,
                sortable: false,
                editable: true,
                search: false,
                edittype: "text",
                editoptions: { size: 30, maxlength: 14 },
            }
        ],
        rowNum: 500, // число отображаемых строк
        loadonce: false, // загрузка только один раз
        sortname: 'NAME', // сортировка по умолчанию по столбцу NAME
        sortorder: "asc", // порядок сортировки
        width: window.innerWidth - 80, // ширина грида
        height: 500,      // высота грида
        viewrecords: true, // отображать количество записей
        caption: "Customers", // подпись к гриду
        pager: 'jqg-pager'  // элемент для отображения навигации
    });

    dbGrid.jqGrid('navGrid', '#jqg-pager', {
        search: true,  // поиск
        add: true,     // добавление
        edit: true,    // редактирование
        del: true,     // удаление
        view: true,    // просмотр записи
        refresh: true, // обновление
        // подписи кнопок
        searchtext: "Поиск",
        addtext: "Добавить",
        edittext: "Изменить",
        deltext: "Удалить",
        viewtext: "Смотреть",
        viewtitle: "Выбранная запись",
        refreshtext: "Обновить"
    },
    update("edit"), // обновление
    update("add"), // добавление
    update("del") // удаление
    );

    // функция возвращающая настройки редактора
    function update(act) {
        return {
            closeAfterAdd: true, // закрыть после добавления
            closeAfterEdit: true, // закрыть после редактирования
            width: 400,           // ширина редактора
            reloadAfterSubmit: true, // обновление
            drag: true,              // перетаскиваемый
            // обработчик отправки формы редактирования/удаления/добавления
            onclickSubmit: function (params, postdata) {
                // получаем идентификатор строки
                var selectedRow = dbGrid.getGridParam("selrow");
```

```
                    // устанавливаем url в зависимости от операции
                    switch (act) {
                        case "add":
                            params.url = '@Url.Action("Create")';
                            break;

                        case "edit":
                            params.url = '@Url.Action("Edit")';
                            postdata.CUSTOMER_ID = selectedRow;
                            break;

                        case "del":
                            params.url = '@Url.Action("Delete")';
                            postdata.CUSTOMER_ID = selectedRow;
                            break;
                    }
                },
                // обработчик результатов обработки форм (операций)
                afterSubmit: function (response, postdata) {
                    var responseData = response.responseJSON;
                    // проверяем результат на наличие сообщений об ошибках
                    if (responseData.hasOwnProperty("error")) {
                        if (responseData.error.length) {
                            return [false, responseData.error];
                        }
                    }
                    else {
                        // обновление грида
                        $(this).jqGrid(
                            'setGridParam',
                            {
                                datatype: 'json'
                            }
                        ).trigger('reloadGrid');
                    }
                    return [true, "", 0];
                }
            };
        };

    });
</script>
```

Как видите всё представление состоит из заголовка, таблицы jqg и блока jqg-pager для отображения панели навигации, остальное занимает скрипт по инициализации грида, панели навигации и диалога редактирования. Для правильного отображения грида, размещения элементов ввода в форме редактирования, настройки валидации форм ввода, настройки возможностей сортировки и поиска важно правильно настроить свойства модели. Эта настройка довольно нетривиальна и содержит множество параметров. Я постарался описать используемые параметры в комментариях. Полное описание параметров модели вы можете найти в документации по библиотеки jqGrid в разделе ColModel API.

Обратите внимание, что для параметров редактирования и удаления нам пришлось добавить в параметры запроса идентификатор заказчика

```
case "edit":
    params.url = '@Url.Action("Edit")';
    postdata.CUSTOMER_ID = selectedRow;
    break;

case "del":
    params.url = '@Url.Action("Delete")';
    postdata.CUSTOMER_ID = selectedRow;
    break;
```

Это сделано потому, что jqGrid автоматически не добавляет в форму ввода скрытые колонки грида, хотя, на мой взгляд, это было бы логично, хотя бы для ключевых полей.

Работающая страница справочника поставщиков будет выглядеть следующим образом:

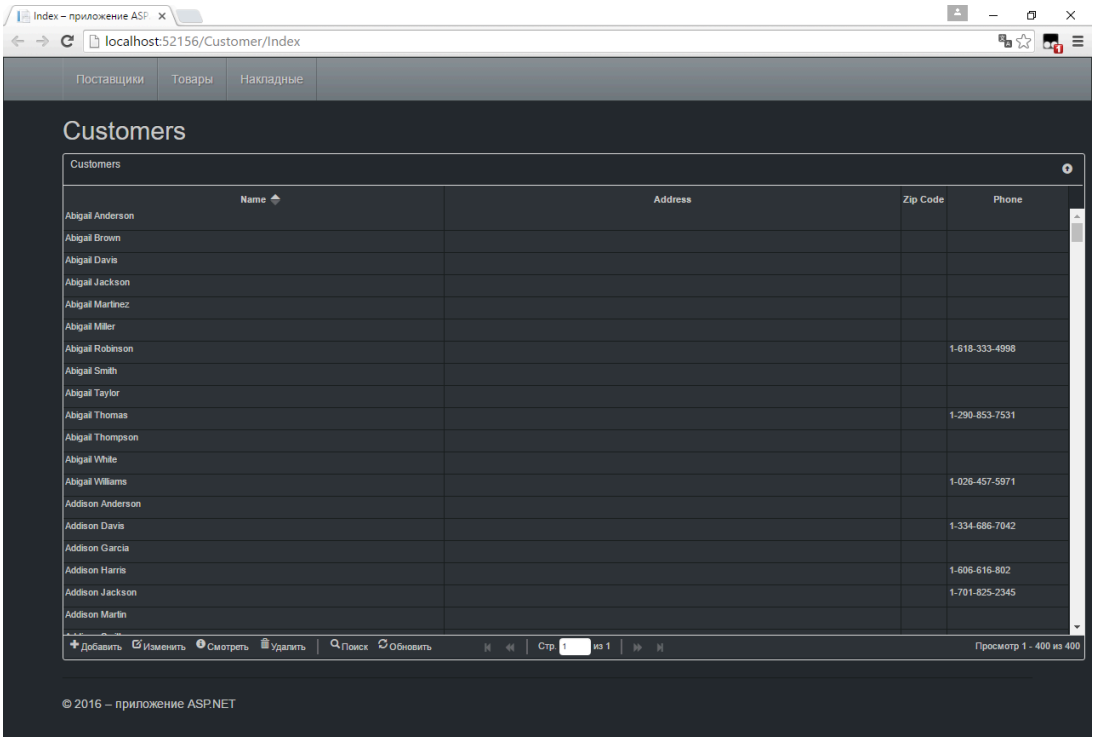

Рис. 4.11. Справочник заказчиков

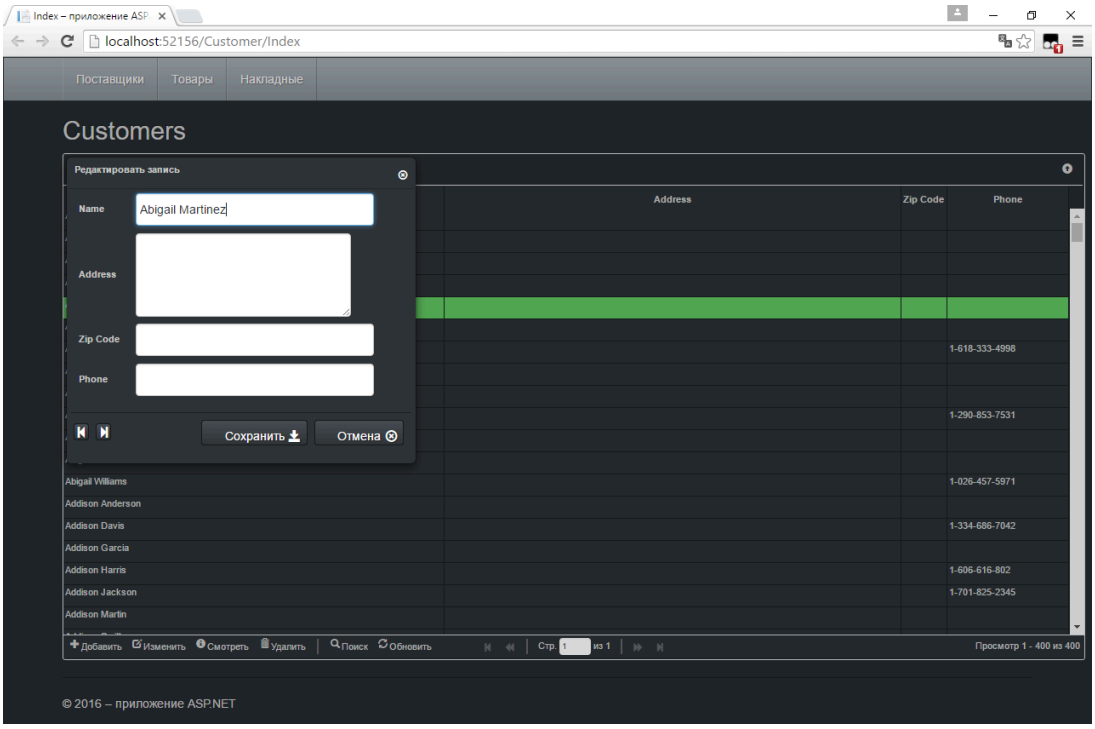

Рис. 4.12. Редактирование заказчика

Контроллер и представление для справочника товаров делаются по аналогии. Здесь мы не будем описывать их подробно, вы можете написать их самостоятельно или найти в исходных кодах, которые прилагаются к данной статье.

Создание пользовательского интерфейса журналов

В нашем приложении будет один журнал «Счёт-фактуры». В отличие от справочников журналы содержат довольно большое количество записей и являются часто пополняемыми.

Счёт-фактура – состоит из заголовка, где описываются общие атрибуты (номер, дата, заказчик …), и строк счёт-фактуры со списком товаром, их количеством, стоимостью и т.д. Для экономии пространства страницы мы сделаем детализирующий грид скрытым. Он будет отображён лишь при клике по иконке со знаком «+», таким образом, у нас получается, что детализирующий грид вложен в главный.

Контроллер для счёт-фактур

Контроллер журнала счёт фактуры должен уметь отдавать данные как по шапкам счёт-фактуры, так и по её позициям. То же самое касается методов для добавления, редактирования и удаления.

```
[Authorize(Roles = "manager")]
public class InvoiceController : Controller
{
    private DbModel db = new DbModel();
```

```
// Отображение представления
public ActionResult Index()
{
    return View();
}

// Получение данных в виде JSON для главного грида
public ActionResult GetData(int? rows, int? page, string sidx, string sord,
    string searchField, string searchString, string searchOper)
{
    // получаем номер страницы, количество отображаемых данных
    int pageNo = page ?? 1;
    int limit = rows ?? 20;
    // вычисляем смещение
    int offset = (pageNo - 1) * limit;

    // строим запрос для получения счёт-фактур
    var invoicesQuery =
        from invoice in db.INVOICES
        where (invoice.INVOICE_DATE >= AppVariables.StartDate) &&
              (invoice.INVOICE_DATE <= AppVariables.FinishDate)
        select new
        {
            INVOICE_ID = invoice.INVOICE_ID,
            CUSTOMER_ID = invoice.CUSTOMER_ID,
            CUSTOMER_NAME = invoice.CUSTOMER.NAME,
            INVOICE_DATE = invoice.INVOICE_DATE,
            TOTAL_SALE = invoice.TOTAL_SALE,
            PAID = invoice.PAID
        };

    // добавлением в запрос условия поиска, если он производится
    // для разных полей доступны разные операторы
    // сравнения при поиске
    if (searchField == "CUSTOMER_NAME")
    {
         switch (searchOper)
         {
             case "eq": // equal
                 invoicesQuery = invoicesQuery.Where(
                    c => c.CUSTOMER_NAME == searchString);
                 break;
             case "bw": // starting with
                 invoicesQuery = invoicesQuery.Where(
                    c => c.CUSTOMER_NAME.StartsWith(searchString));
                 break;
             case "cn": // containing
                 invoicesQuery = invoicesQuery.Where(
                    c => c.CUSTOMER_NAME.Contains(searchString));
                 break;
         }
    }
    if (searchField == "INVOICE_DATE")
    {
        var dateValue = DateTime.Parse(searchString);
        switch (searchOper)
        {
```

```
            case "eq": // =
                invoicesQuery = invoicesQuery.Where(
                    c => c.INVOICE_DATE == dateValue);
                break;
            case "lt": // <
                invoicesQuery = invoicesQuery.Where(
                    c => c.INVOICE_DATE < dateValue);
                break;
            case "le": // <=
                invoicesQuery = invoicesQuery.Where(
                    c => c.INVOICE_DATE <= dateValue);
                break;
            case "gt": // >
                invoicesQuery = invoicesQuery.Where(
                    c => c.INVOICE_DATE > dateValue);
                break;
            case "ge": // >=
                invoicesQuery = invoicesQuery.Where(
                    c => c.INVOICE_DATE >= dateValue);
                break;

        }
    }
    if (searchField == "PAID")
    {
        int iVal = (searchString == "on") ? 1 : 0;
        invoicesQuery = invoicesQuery.Where(c => c.PAID == iVal);
    }

    // получаем общее количество счёт-фактур
    int totalRows = invoicesQuery.Count();

    // добавляем сортировку
    switch (sord)
    {
        case "asc":
            invoicesQuery = invoicesQuery.OrderBy(
                invoice => invoice.INVOICE_DATE);
            break;
        case "desc":
            invoicesQuery = invoicesQuery.OrderByDescending(
                invoice => invoice.INVOICE_DATE);
            break;
    }

    // получаем список счёт-фактур
    var invoices = invoicesQuery
         .Skip(offset)
         .Take(limit)
         .ToList();

    // вычисляем общее количество страниц
    int totalPages = totalRows / limit + 1;

    // создаём результат для jqGrid
    var result = new
    {
        page = pageNo,
```

```
            total = totalPages,
            records = totalRows,
            rows = invoices
        };

        // преобразуем результат в JSON
        return Json(result, JsonRequestBehavior.AllowGet);
    }

    // Получение данных в виде JSON для детализирующего грида
    public ActionResult GetDetailData(int? invoice_id)
    {
        // строим запрос для получения позиций счёт-фактуры
        // отфильтрованный по коду счёт-фактуры
        var lines =
            from line in db.INVOICE_LINES
            where line.INVOICE_ID == invoice_id
            select new
            {
                INVOICE_LINE_ID = line.INVOICE_LINE_ID,
                INVOICE_ID = line.INVOICE_ID,
                PRODUCT_ID = line.PRODUCT_ID,
                Product = line.PRODUCT.NAME,
                Quantity = line.QUANTITY,
                Price = line.SALE_PRICE,
                Total = line.QUANTITY * line.SALE_PRICE
            };

        // получаем список позиций
        var invoices = lines
             .ToList();

        // создаём результат для jqGrid
        var result = new
        {
            rows = invoices
        };

        // преобразуем результат в JSON
        return Json(result, JsonRequestBehavior.AllowGet);
    }


    // Добавление новой шапки счёт-фактуры
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(
       [Bind(Include = "CUSTOMER_ID,INVOICE_DATE")] INVOICE invoice)
    {
        // проверяем правильность модели
        if (ModelState.IsValid)
        {
            try
            {
                var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
                var CUSTOMER_ID = new FbParameter("CUSTOMER_ID", FbDbType.Integer);
                var INVOICE_DATE = new FbParameter("INVOICE_DATE",
                                            FbDbType.TimeStamp);
```

```
            // инициализируем параметры значениями
            INVOICE_ID.Value = db.NextValueFor("GEN_INVOICE_ID");
            CUSTOMER_ID.Value = invoice.CUSTOMER_ID;
            INVOICE_DATE.Value = invoice.INVOICE_DATE;
            // выполняем ХП
            db.Database.ExecuteSqlCommand(
"EXECUTE PROCEDURE SP_ADD_INVOICE(@INVOICE_ID, @CUSTOMER_ID, @INVOICE_DATE)",
                INVOICE_ID,
                CUSTOMER_ID,
                INVOICE_DATE);
                // возвращаем успех в формате JSON
            return Json(true);
        }
        catch (Exception ex)
        {
            // возвращаем ошибку в формате JSON
            return Json(new { error = ex.Message });
        }

    }
    else {
        string messages = string.Join("; ", ModelState.Values
                                .SelectMany(x => x.Errors)
                                .Select(x => x.ErrorMessage));
        // возвращаем ошибку в формате JSON
        return Json(new { error = messages });
    }
}


// Редактирование шапки счёт-фактуры
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(
   [Bind(Include = "INVOICE_ID,CUSTOMER_ID,INVOICE_DATE")] INVOICE invoice)
{
    // проверяем правильность модели
    if (ModelState.IsValid)
    {
        try
        {
            var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
            var CUSTOMER_ID = new FbParameter("CUSTOMER_ID", FbDbType.Integer);
            var INVOICE_DATE = new FbParameter("INVOICE_DATE",
                                                FbDbType.TimeStamp);
            // инициализируем параметры значениями
            INVOICE_ID.Value = invoice.INVOICE_ID;
            CUSTOMER_ID.Value = invoice.CUSTOMER_ID;
            INVOICE_DATE.Value = invoice.INVOICE_DATE;
            // выполняем ХП
            db.Database.ExecuteSqlCommand(
"EXECUTE PROCEDURE SP_EDIT_INVOICE(@INVOICE_ID, @CUSTOMER_ID, @INVOICE_DATE)",
                INVOICE_ID,
                CUSTOMER_ID,
                INVOICE_DATE);
            // возвращаем успех в формате JSON
            return Json(true);
        }
```

```
            catch (Exception ex)
            {
                // возвращаем ошибку в формате JSON
                return Json(new { error = ex.Message });
            }
        }
        else {
            string messages = string.Join("; ", ModelState.Values
                                    .SelectMany(x => x.Errors)
                                    .Select(x => x.ErrorMessage));
            // возвращаем ошибку в формате JSON
            return Json(new { error = messages });
        }
    }


    // Удаление шапки счёт-фактуры
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Delete(int id)
    {
        try
        {
            var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
            // инициализируем параметры значениями
            INVOICE_ID.Value = id;
            // выполняем ХП
            db.Database.ExecuteSqlCommand(
                "EXECUTE PROCEDURE SP_DELETE_INVOICE(@INVOICE_ID)",
                INVOICE_ID);
            // возвращаем успех в формате JSON
            return Json(true);
        }
        catch (Exception ex)
        {
            // возвращаем ошибку в формате JSON
            return Json(new { error = ex.Message });
        }
    }

    // Оплата счёт фактуры
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Pay(int id)
    {
        try
        {
            var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
            // инициализируем параметры значениями
            INVOICE_ID.Value = id;
            // выполняем ХП
            db.Database.ExecuteSqlCommand(
                "EXECUTE PROCEDURE SP_PAY_FOR_INOVICE(@INVOICE_ID)",
                INVOICE_ID);
            // возвращаем успех в формате JSON
            return Json(true);
        }
        catch (Exception ex)
```

```
        {
            // возвращаем ошибку в формате JSON
            return Json(new { error = ex.Message });
        }
    }

    // Добавление позиции счёт фактуры
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult CreateDetail(
       [Bind(Include = "INVOICE_ID,PRODUCT_ID,QUANTITY")] INVOICE_LINE invoiceLine)
    {
        // проверяем правильность модели
        if (ModelState.IsValid)
        {
            try
            {
                var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
                var PRODUCT_ID = new FbParameter("PRODUCT_ID", FbDbType.Integer);
                var QUANTITY = new FbParameter("QUANTITY", FbDbType.Integer);
                // инициализируем параметры значениями
                INVOICE_ID.Value = invoiceLine.INVOICE_ID;
                PRODUCT_ID.Value = invoiceLine.PRODUCT_ID;
                QUANTITY.Value = invoiceLine.QUANTITY;
                // выполняем ХП
                db.Database.ExecuteSqlCommand(
    "EXECUTE PROCEDURE SP_ADD_INVOICE_LINE(@INVOICE_ID, @PRODUCT_ID, @QUANTITY)",
                    INVOICE_ID,
                    PRODUCT_ID,
                    QUANTITY);
                // возвращаем успех в формате JSON
                return Json(true);
            }
            catch (Exception ex)
            {
                // возвращаем ошибку в формате JSON
                return Json(new { error = ex.Message });
            }
        }
        else {
            string messages = string.Join("; ", ModelState.Values
                                    .SelectMany(x => x.Errors)
                                    .Select(x => x.ErrorMessage));
            // возвращаем ошибку в формате JSON
            return Json(new { error = messages });
        }
    }

    // редактирование позиции счёт фактуры
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult EditDetail(
      [Bind(Include = "INVOICE_LINE_ID,INVOICE_ID,PRODUCT_ID,QUANTITY")]
      INVOICE_LINE invoiceLine)
    {
        // проверяем правильность модели
        if (ModelState.IsValid)
        {
```

```
            try
            {
                // Создание параметров
                var INVOICE_LINE_ID = new FbParameter("INVOICE_LINE_ID",
                                                FbDbType.Integer);
                var QUANTITY = new FbParameter("QUANTITY", FbDbType.Integer);
                // инициализируем параметры значениями
                INVOICE_LINE_ID.Value = invoiceLine.INVOICE_LINE_ID;
                QUANTITY.Value = invoiceLine.QUANTITY;
                // выполняем ХП
                db.Database.ExecuteSqlCommand(
    "EXECUTE PROCEDURE SP_EDIT_INVOICE_LINE(@INVOICE_LINE_ID, @QUANTITY)",
                    INVOICE_LINE_ID,
                    QUANTITY);
                // возвращаем успех в формате JSON
                return Json(true);
            }
            catch (Exception ex)
            {
                // возвращаем ошибку в формате JSON
                return Json(new { error = ex.Message });
            }
        }
        else {
            string messages = string.Join("; ", ModelState.Values
                                    .SelectMany(x => x.Errors)
                                    .Select(x => x.ErrorMessage));
            // возвращаем ошибку в формате JSON
            return Json(new { error = messages });
        }
    }

    // Удаление позиции счёт фактуры
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult DeleteDetail(int id)
    {
        try
        {
            // Создание параметров
            var INVOICE_LINE_ID = new FbParameter("INVOICE_LINE_ID",
                                            FbDbType.Integer);
            // инициализируем параметры значениями
            INVOICE_LINE_ID.Value = id;
            // выполняем ХП
            db.Database.ExecuteSqlCommand(
                 "EXECUTE PROCEDURE SP_DELETE_INVOICE_LINE(@INVOICE_LINE_ID)",
                 INVOICE_LINE_ID);
            // возвращаем успех в формате JSON
            return Json(true);
        }
        catch (Exception ex)
        {
            // возвращаем ошибку в формате JSON
            return Json(new { error = ex.Message });
        }
    }
```

```
    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            db.Dispose();
        }
        base.Dispose(disposing);
    }
}
```

В методе `GetDetailData` для получения списка позиций счёт-фактуры нет кода для постраничной навигации. Дело в том, что у конкретной счёт-фактуры не очень много позиций для того чтобы применять к ним постраничную навигацию. Это упрощает код, и делает его быстрее.

На этот раз все действия по модификации данных выполняются в хранимых процедурах, однако вы можете выполнить те же действия с помощью Entity Framework. Тексты хранимых процедур вы можете посмотреть в скрипте создания БД.

Представления для счёт-фактур

Как и для контроллера Customer нам потребуется только одно представление `View/Invoice/Index.cshtml`, остальные можно удалить из этой папки. Сама разметка представления очень проста, а вот JavaScript кода довольно много. Будем описывать js код по частям.

```
@{
    ViewBag.Title = "Index";
}

<h2>Invoices</h2>

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

<script type="text/javascript">

    /**
     * Код для работы с jqGrid
     */
</script>
```

Для начала рассмотрим код для работы с главным гридом. По сути, в нём необходимо только прописать свойства модели (типы и размеры полей, параметры поиска, сортировки, видимости и т.д.).

```
// Грид с инвойсами
var dbGrid = $("#jqg").jqGrid({
    url: '@Url.Action("GetData")', // url для получения данных
    datatype: "json", // формат получения данных
```

```
    mtype: "GET",      // тип http запроса
    // описание модели
    colModel: [
        {
            label: 'Id',          // подпись
            name: 'INVOICE_ID',   // имя поля
            key: true,            // признак ключевого поля
            hidden: true          // скрыт
        },
        {
            label: 'CUSTOMER_ID', // подпись
            name: 'CUSTOMER_ID',  // имя поля
            hidden: true,         // скрыт
            editrules: { edithidden: true, required: true }, // скрытое и требуемое
            editable: true,       // редактируемое
            edittype:'custom',    // собственный тип
            editoptions: {
                custom_element: function (value, options) {
                    // добавляем скрытый input
                    return $("<input>")
                            .attr('type', 'hidden')
                            .attr('rowid', options.rowId)
                            .addClass("FormElement")
                            .addClass("form-control")
                            .val(value)
                            .get(0);
                }
            }
        },
        {
            label: 'Date',
            name: 'INVOICE_DATE',
            width: 60,              // ширина
            sortable: true,         // позволять сортировку
            editable: true,         // редактируемое
            search: true,           // разрешён поиск
            edittype: "text",       // тип поля ввода
            align: "right",         // выравнено по правому краю
            formatter: 'date',      // отформатировано как дата
            sorttype: 'date',       // сортируем как дату
            formatoptions: {        // формат даты
                srcformat: 'd.m.Y H:i:s',
                newformat: 'd.m.Y H:i:s'
            },
            editoptions: {
                // инициализация элемента формы для редактирования
                dataInit: function (element) {
                    // создаём datepicker
                    $(element).datepicker({
                        id: 'invoiceDate_datePicker',
                        dateFormat: 'dd.mm.yy',
                        minDate: new Date(2000, 0, 1),
                        maxDate: new Date(2030, 0, 1)
                    });
                }
            },
            searchoptions: {
                // инициализация элемента формы для поиска
```

```
            dataInit: function (element) {
                // создаём datepicker
                $(element).datepicker({
                    id: 'invoiceDate_datePicker',
                    dateFormat: 'dd.mm.yy',
                    minDate: new Date(2000, 0, 1),
                    maxDate: new Date(2030, 0, 1)
                });
            },
            searchoptions: { // типы поиска
                sopt: ['eq', 'lt', 'le', 'gt', 'ge']
            },
        }
    },
    {
        label: 'Customer',
        name: 'CUSTOMER_NAME',
        width: 250,
        editable: true,
        edittype: "text",
        editoptions: {
            size: 50,
            maxlength: 60,
            readonly: true    // только чтение
        },
        editrules: { required: true },
        search: true,
        searchoptions: {
                    sopt: ['eq', 'bw', 'cn']
        },
    },
    {
        label: 'Amount',
        name: 'TOTAL_SALE',
        width: 60,
        sortable: false,
        editable: false,
        search: false,
        align: "right",
        formatter: 'currency', // форматировать как валюту
        sorttype: 'number',
        searchrules: {
             "required": true,
             "number": true,
             "minValue": 0
        }
    },
    {
        label: 'Paid',
        name: 'PAID',
        width: 30,
        sortable: false,
        editable: true,
        search: true,
        searchoptions: {
            sopt: ['eq']
        },
        edittype: "checkbox", // галочка
```

```
            formatter: "checkbox",
            stype: "checkbox",
            align: "center",
            editoptions: {
                value: "1",
                offval: "0"
            }
        }
    ],
    rowNum: 500, // число отображаемых строк
    loadonce: false, // загрузка только один раз
    sortname: 'INVOICE_DATE', // сортировка по умолчанию по столбцу NAME
    sortorder: "desc", // порядок сортировки
    width: window.innerWidth - 80, // ширина грида
    height: 500, // высота грида
    viewrecords: true, // отображать количество записей
    caption: "Invoices", // подпись к гриду
    pager: '#jpager', // элемент для отображения постраничной навигации
    subGrid: true, // показывать вложенный грид
    // javascript функция для отображения родительского грида
    subGridRowExpanded: showChildGrid,
    subGridOptions: { // опции вложенного грида
         // загружать данные только один раз
         reloadOnExpand: false,
         // загружать строки подгрида только при щелчке по иконке "+"
         selectOnExpand: true
    },
});

// отображение панели навигации
dbGrid.jqGrid('navGrid', '#jpager',
    {
         search: true,  // поиск
         add: true,     // добавление
         edit: true,    // редактирование
         del: true,     // удаление
         view: false,    // просмотр записи
         refresh: true, // обновление

         searchtext: "Поиск",
         addtext: "Добавить",
         edittext: "Изменить",
         deltext: "Удалить",
         viewtext: "Смотреть",
         viewtitle: "Выбранная запись",
         refreshtext: "Обновить"
    },
    update("edit"), // обновление
    update("add"), // добавление
    update("del") // удаление
);
```

Добавим в главный грид ещё «пользовательскую» одну кнопку для оплаты счёт-фактуры.

```
// добавление кнопки для оплаты счёт фактуры
dbGrid.navButtonAdd('#jpager',
    {
        buttonicon: "glyphicon-usd",
        title: "Оплатить",
        caption: "Оплатить",
        position: "last",
        onClickButton: function () {
            // получаем идентификатор текущей записи
            var id = dbGrid.getGridParam("selrow");
            if (id) {
                var url = '@Url.Action("Pay")';
                $.ajax({
                     url: url,
                     type: 'POST',
                     data: { id: id },
                     success: function (data) {
                         // проверяем, не произошла ли ошибка
                         if (data.hasOwnProperty("error")) {
                             alertDialog('Ошибка', data.error);
                         }
                         else {
                             // обновление грида
                             $("#jqg").jqGrid(
                                 'setGridParam',
                                 {
                                     datatype: 'json'
                                 }
                             ).trigger('reloadGrid');
                         }
                     }
                });
            }
        }
    });
```

Диалоги редактирования счёт-фактуры

В отличие от справочников диалоги редактирования для журналов намного сложнее. Зачастую они используют выбор из других справочников. Поэтому такие диалоги редактирования не получится построить стандартными способами jqGrid, однако в этой библиотеки существует возможность построение диалогов по шаблону, которой мы и воспользуемся.

Для выбора заказчика сделаем поле только для чтения и разместим справа от него кнопку для вызова формы с гридом для отображения списка заказчиков.

```
// возвращает свойства для создания диалогов редактирования
function update(act) {
    // шаблон диалога редактирования
    var template = "<div style='margin-left:15px;' id='dlgEditInvoice'>";
    template += "<div>{CUSTOMER_ID} </div>";
    template += "<div> Date: </div><div>{INVOICE_DATE} </div>";
    // поле ввода заказчика с кнопкой
```

```
    template += "<div> Customer <sup>*</sup>:</div>";
    template += "<div>";
    template += "<div style='float: left;'>{CUSTOMER_NAME}</div> ";
    template += "<a style='margin-left: 0.2em;' class='btn'";
    template += " onclick='showCustomerWindow(); return false;'>";
    template += "<span class='glyphicon glyphicon-folder-open'></span>";
    template += " Выбрать</a> ";
    template += "<div style='clear: both;'></div>";
    template += "</div>";
    template += "<div> {PAID} Paid </div>";
    template += "<hr style='width: 100%;'/>";
    template += "<div> {sData} {cData}  </div>";
    template += "</div>";

    return {
        top: $(".container.body-content").position().top + 150,
        left: $(".container.body-content").position().left + 150,
        modal: true,
        drag: true,
        closeOnEscape: true,
        closeAfterAdd: true, // закрыть после добавления
        closeAfterEdit: true, // закрыть после редактирования
        reloadAfterSubmit: true, // обновление
        template: (act != "del") ? template : null,
        onclickSubmit: function (params, postdata) {
                    // получаем идентификатор строки
                    var selectedRow = dbGrid.getGridParam("selrow");
                    switch (act) {
                        case "add":
                            params.url = '@Url.Action("Create")';
                            // получаем идентификатор заказчика для текущей строки
                            postdata.CUSTOMER_ID =
                                $('#dlgEditInvoice input[name=CUSTOMER_ID]').val();
                            break;

                        case "edit":
                            params.url = '@Url.Action("Edit")';
                            postdata.INVOICE_ID = selectedRow;
                            // получаем идентификатор заказчика для текущей строки
                            postdata.CUSTOMER_ID =
                                $('#dlgEditInvoice input[name=CUSTOMER_ID]').val();
                            break;

                        case "del":
                            params.url = '@Url.Action("Delete")';
                            postdata.INVOICE_ID = selectedRow;
                            break;
                    }
                },
                afterSubmit: function (response, postdata) {
                    var responseData = response.responseJSON;
                    // проверяем результат на наличие сообщений об ошибках
                    if (responseData.hasOwnProperty("error")) {
                        if (responseData.error.length) {
                            return [false, responseData.error];
                        }
                    }
                    else {
```

```
                        // обновление грида
                        $(this).jqGrid(
                            'setGridParam',
                            {
                                datatype: 'json'
                            }
                        ).trigger('reloadGrid');
                    }
                    return [true, "", 0];
                }
            };
        };
}
```

Теперь напишем функцию для открытия справочника заказчиков. В этой функции мы будем создавать диалог с помощью библиотеки Bootstrap, в котором будет размещён грид для выбора заказчика. По сути, это тот же самый грид, который мы использовали выше, но размещённый внутри диалогового окна. При нажатии кнопки «OK» идентификатор заказчика и его имя будут записаны в элементы ввода родительского диалога для редактирования счёт-фактуры.

```
/**
  * Отображение окна для выбора справочника заказчиков
  */
function showCustomerWindow() {
    // основной блок диалога
    var dlg = $('<div>')
        .attr('id', 'dlgChooseCustomer')
        .attr('aria-hidden', 'true')
        .attr('role', 'dialog')
        .attr('data-backdrop', 'static')
        .css("z-index", '2000')
        .addClass('modal')
        .appendTo($('body'));

    // блок с содержимым диалога
    var dlgContent = $("<div>")
        .addClass("modal-content")
        .css('width', '730px')
        .appendTo($('<div>')
        .addClass('modal-dialog')
        .appendTo(dlg));

    // блок с шапкой диалога
    var dlgHeader = $('<div>').addClass("modal-header").appendTo(dlgContent);
    // кнопка "X" для закрытия
    $("<button>")
        .addClass("close")
        .attr('type', 'button')
        .attr('aria-hidden', 'true')
        .attr('data-dismiss', 'modal')
        .html("&times;")
        .appendTo(dlgHeader);
    // подпись
    $("<h5>").addClass("modal-title").html("Выбор заказчика").appendTo(dlgHeader);
```

```
// тело диалога
var dlgBody = $('<div>')
    .addClass("modal-body")
    .appendTo(dlgContent);

// подвал диалога
var dlgFooter = $('<div>').addClass("modal-footer").appendTo(dlgContent);
// Кнопка "OK"
$("<button>")
    .attr('type', 'button')
    .addClass('btn')
    .html('OK')
    .on('click', function () {
        var rowId = $("#jqgCustomer").jqGrid("getGridParam", "selrow");
        var row = $("#jqgCustomer").jqGrid("getRowData", rowId);
        // сохраняем идентификатор и имя заказчика
        // в элементы ввода родительской формы
        $('#dlgEditInvoice input[name=CUSTOMER_ID]').val(rowId);
        $('#dlgEditInvoice input[name=CUSTOMER_NAME]').val(row["NAME"]);
        dlg.modal('hide');
    })
    .appendTo(dlgFooter);
// Кнопка "Cancel"
$("<button>")
    .attr('type', 'button')
    .addClass('btn')
    .html('Cancel')
    .on('click', function () { dlg.modal('hide'); })
    .appendTo(dlgFooter);
// добавляем таблицу для отображения заказчиков в тело диалога
$('<table>')
    .attr('id', 'jqgCustomer')
    .appendTo(dlgBody);
// добавляем панель навигации
$('<div>')
    .attr('id', 'jqgCustomerPager')
    .appendTo(dlgBody);

dlg.on('hidden.bs.modal', function () {
    dlg.remove();
});

// отображаем диалог
dlg.modal();

// создание и инициализация jqGrid
var dbGrid = $("#jqgCustomer").jqGrid({
    url: '@Url.Action("GetData", "Customer")', // url для получения данных
    mtype: "GET", // тип http запроса
    datatype: "json", // формат получения данных
    page: 1,
    width: '100%',
    // описание модели
    colModel: [
        {
            label: 'Id',          // подпись
            name: 'CUSTOMER_ID',  // имя поля
```

```
                key: true,              // признак ключевого поля
                hidden: true            // скрытое
            },
            {
                label: 'Name',
                name: 'NAME',
                width: 250,             // ширина
                sortable: true,         // разрешена сортировка
                editable: true,         // разрешено редактирование
                edittype: "text",       // тип поля в редакторе
                search: true,           // разрешён поиск
                searchoptions: {
                    sopt: ['eq', 'bw', 'cn'] // разрешённые операторы поиска
                },
                // размер и максимальная длина для поля ввода
                editoptions: { size: 30, maxlength: 60 },
                // говорит о том что поле обязательное
                editrules: { required: true }
            },
            {
                label: 'Address',
                name: 'ADDRESS',
                width: 300,
                sortable: false,        // запрещаем сортировку
                editable: true,         // редактируемое
                search: false,          // запрещаем поиск
                edittype: "textarea",
                editoptions: { maxlength: 250, cols: 30, rows: 4 }
            },
            {
                label: 'Zip Code',
                name: 'ZIPCODE',
                width: 60,
                sortable: false,
                editable: true,
                search: false,
                edittype: "text",
                editoptions: { size: 30, maxlength: 10 },
            },
            {
                label: 'Phone',
                name: 'PHONE',
                width: 85,
                sortable: false,
                editable: true,
                search: false,
                edittype: "text",
                editoptions: { size: 30, maxlength: 14 },
            }
        ],
        loadonce: false,
        pager: '#jqgCustomerPager',
        rowNum: 500, // число отображаемых строк
        sortname: 'NAME', // сортировка по умолчанию по столбцу NAME
        sortorder: "asc", // порядок сортировки
        height: 500
    });
```

```
    dbGrid.jqGrid('navGrid', '#jqgCustomerPager',
        {
            search: true,  // поиск
            add: false,     // добавление
            edit: false,    // редактирование
            del: false,     // удаление
            view: false,    // просмотр записи
            refresh: true, // обновление

            searchtext: "Поиск",
            viewtext: "Смотреть",
            viewtitle: "Выбранная запись",
            refreshtext: "Обновить"
        }
    );
}
```

Для этого журнала нам осталось написать функцию `showChildGrid`, которая позволяет просматривать и редактировать информацию о позициях накладной. Эта функция будет динамически создавать грид с позициями счёт-фактуры при нажатии на кнопку «+» (для раскрытия деталей). Для загрузки данных о позиции нам будет необходимо передавать первичный ключ выбранной шапки счёт фактуры.

```
// обработчик события раскрытия родительского грида
// принимает два параметра идентификатор родительской записи
// и первичный ключ записи
function showChildGrid(parentRowID, parentRowKey) {
    var childGridID = parentRowID + "_table";
    var childGridPagerID = parentRowID + "_pager";

    // отправляем первичный ключ родительской записи
    // чтобы отфильтровать записи позиций накладной
    var childGridURL = '@Url.Action("GetDetailData")';
    childGridURL = childGridURL + "?invoice_id="
                + encodeURIComponent(parentRowKey)

    // добавляем HTML элементы для отображения таблицы и постраничной навигации
    // как дочерние для выбранной строки в мастер гриде
    $('<table>')
        .attr('id', childGridID)
        .appendTo($('#' + parentRowID));
    $('<div>')
        .attr('id', childGridPagerID)
        .addClass('scroll')
        .appendTo($('#' + parentRowID));

    // создаём и инициализируем дочерний грид
    var detailGrid = $("#" + childGridID).jqGrid({
        url: childGridURL,
        mtype: "GET",
        datatype: "json",
        page: 1,
        colModel: [
            {
```

```
            label: 'Invoice Line ID',
            name: 'INVOICE_LINE_ID',
            key: true,
            hidden: true
        },
        {
            label: 'Invoice ID',
            name: 'INVOICE_ID',
            hidden: true,
            editrules: { edithidden: true, required: true },
            editable: true,
            edittype: 'custom',
            editoptions: {
                custom_element: function (value, options) {
                    // создаём скрытый элемент ввода
                    return $("<input>")
                           .attr('type', 'hidden')
                           .attr('rowid', options.rowId)
                           .addClass("FormElement")
                           .addClass("form-control")
                           .val(parentRowKey)
                           .get(0);
                }
            }
        },
        {
            label: 'Product ID',
            name: 'PRODUCT_ID',
            hidden: true,
            editrules: { edithidden: true, required: true },
            editable: true,
            edittype: 'custom',
            editoptions: {
                custom_element: function (value, options) {
                    // создаём скрытый элемент ввода
                    return $("<input>")
                         .attr('type', 'hidden')
                         .attr('rowid', options.rowId)
                         .addClass("FormElement")
                         .addClass("form-control")
                         .val(value)
                         .get(0);
                }
            }
        },
        {
            label: 'Product',
            name: 'Product',
            width: 300,
            editable: true,
            edittype: "text",
            editoptions: {
                size: 50,
                maxlength: 60,
                readonly: true
            },
            editrules: { required: true }
        },
```

```
            {
                label: 'Price',
                name: 'Price',
                formatter: 'currency',
                editable: true,
                editoptions: {
                    readonly: true
                },
                align: "right",
                width: 100
            },
            {
                label: 'Quantity',
                name: 'Quantity',
                align: "right",
                width: 100,
                editable: true,
                editrules: { required: true, number: true, minValue: 1 },
                editoptions: {
                  dataEvents: [
                    {
                       type: 'change',
                       fn: function (e) {
                         var quantity = $(this).val() - 0;
                         var price =
                           $('#dlgEditInvoiceLine input[name=Price]').val() - 0;
               $('#dlgEditInvoiceLine input[name=Total]').val(quantity * price);
                       }
                    }
                  ],
                  defaultValue: 1
                }
            },
            {
                label: 'Total',
                name: 'Total',
                formatter: 'currency',
                align: "right",
                width: 100,
                editable: true,
                editoptions: {
                    readonly: true
                }
            }
        ],
        loadonce: false,
        width: '100%',
        height: '100%',
        pager: "#" + childGridPagerID
    });

    // отображение панели инструментов
    $("#" + childGridID).jqGrid('navGrid', '#' + childGridPagerID,
        {
            search: false,  // поиск
            add: true,      // добавление
            edit: true,     // редактирование
            del: true,      // удаление
```

```
            refresh: true // обновление
        },
        updateDetail("edit"), // обновление
        updateDetail("add"), // добавление
        updateDetail("del") // удаление
    );

    // функция возвращающая настройки для диалога редактирования
    function updateDetail(act) {
        // шаблон диалога редактирования
        var template = "<div style='margin-left:15px;' id='dlgEditInvoiceLine'>";
        template += "<div>{INVOICE_ID} </div>";
        template += "<div>{PRODUCT_ID} </div>";
        // поле ввода товара с кнопкой
        template += "<div> Product <sup>*</sup>:</div>";
        template += "<div>";
        template += "<div style='float: left;'>{Product}</div> ";
        template += "<a style='margin-left: 0.2em;' class='btn' ";
        template += "onclick='showProductWindow(); return false;'>";
        template += "<span class='glyphicon glyphicon-folder-open'></span>";
        template += " Выбрать</a> ";
        template += "<div style='clear: both;'></div>";
        template += "</div>";
        template += "<div> Quantity: </div><div>{Quantity} </div>";
        template += "<div> Price: </div><div>{Price} </div>";
        template += "<div> Total: </div><div>{Total} </div>";
        template += "<hr style='width: 100%;'/>";
        template += "<div> {sData} {cData}  </div>";
        template += "</div>";

        return {
            top: $(".container.body-content").position().top + 150,
            left: $(".container.body-content").position().left + 150,
            modal: true,
            drag: true,
            closeOnEscape: true,
            closeAfterAdd: true, // закрыть после добавления
            closeAfterEdit: true, // закрыть после редактирования
            reloadAfterSubmit: true, // обновление
            template: (act != "del") ? template : null,
            onclickSubmit: function (params, postdata) {
                var selectedRow = detailGrid.getGridParam("selrow");
                switch (act) {
                    case "add":
                        params.url = '@Url.Action("CreateDetail")';
                        // получаем идентификатор счёт-фактуры
                        postdata.INVOICE_ID =
                            $('#dlgEditInvoiceLine input[name=INVOICE_ID]').val();
                        // получаем идентификатор товара для текущей записи
                        postdata.PRODUCT_ID =
                            $('#dlgEditInvoiceLine input[name=PRODUCT_ID]').val();
                        break;

                    case "edit":
                        params.url = '@Url.Action("EditDetail")';
                        // получаем идентификатор текущей записи
                        postdata.INVOICE_LINE_ID = selectedRow;
                        break;
```

```
                    case "del":
                        params.url = '@Url.Action("DeleteDetail")';
                        // получаем идентификатор текущей записи
                        postdata.INVOICE_LINE_ID = selectedRow;
                        break;
                }
            },
            afterSubmit: function (response, postdata) {
                var responseData = response.responseJSON;
                // проверяем результат на наличие сообщений об ошибках
                if (responseData.hasOwnProperty("error")) {
                    if (responseData.error.length) {
                        return [false, responseData.error];
                    }
                }
                else {
                    // обновление грида
                    $(this).jqGrid(
                       'setGridParam',
                       {
                           datatype: 'json'
                       }
                    ).trigger('reloadGrid');
                }
                return [true, "", 0];
            }
        };
    };
}
```

Вот теперь создание журнала счёт-фактур закончено. Здесь мы не рассмотрели функцию `showProductWindow`, которая предназначена для выбора товара из справочника при заполнении позиций счёт-фактуры. Эта функция полностью аналогична ранее описанной функции `showCustomerWindow`, предназначенной для выбора из справочника заказчиков.

Внимательный читатель мог заметить, что функции для отображения выбора из справочника и отображения справочника почти идентичные. Это можно улучшить выносив эти функции в отдельные файлы скриптов с расширением js. Попробуйте сделать это самостоятельно.

Аутентификация и авторизация

Технология ASP.NET имеет мощный механизм для организации авторизации и аутентификации в .NET приложениях под названием ASP.NET Identity. Инфраструктура OWIN и AspNet Identity позволяют производить как стандартную авторизацию, так и авторизацию через внешние сервисы с помощью аккаунтов в Google, Twitter, Facebook и т.д. Описание технологии ASP.NET Identity является достаточно объёмным и выходит за рамки данной статьи. Вы можете почитать об этой технологии на сайте http://www.asp.net/identity.

А в нашем приложении мы будем использовать чуть более простую модель, основанную на аутентификации форм. Для включения аутентификации форм необходимо сделать изменения в файле конфигурации `web.config`. Находим секцию и внутри этой секции поместим следующую подсекцию:

```
<authentication mode="Forms">
  <forms name="cookies" timeout="2880" loginUrl="~/Account/Login"
         defaultUrl="~/Invoice/Index"/>
</authentication>
```

Установив `mode="Forms"`, мы тем самым подключаем аутентификацию форм. Далее мы задаём ряд параметров. Нам доступен следующий список параметров:

- **coockieless**: определяет, применяются ли куки-наборы и как они используются. Может принимать следующие значения: **UseCookies**(определяет, что куки-наборы будут использоваться всегда вне зависимости от устройства), **UseUri** (куки-наборы никогда не используются), **AutoDetect** (если устройство поддерживает куки-наборы, то они используются, в противном случае они не применяются, при этом проводится тестирование, определяющее, включена ли поддержка), **UseDeviceProfile** (если устройство поддерживает куки-наборы, то они используются, в противном случае они не применяются, в отличие от предыдущего случая тестирование не проводится. Используется по умолчанию).
- **defaultUrl**: определяет путь, по которому осуществляется переход после авторизации
- **domain**: определяет куки-наборы для всего домена. Благодаря этому мы можем использовать одни и те же куки-наборы для главного домена и его субдоменов. По умолчанию имеет значение в качестве пустой строки
- **loginUrl**: адрес для аутентификации пользователя. Значение по умолчанию — "~/Account/Login"
- **name**: задаёт имя для куки-набора. Значение по умолчанию — ".ASPXAUTH"
- **path**: задаёт путь для куки-наборов. Значение по умолчанию — "/"
- **requireSSL**: определяет, требуется ли SSL-соединение для передачи куки-наборов. Значение по умолчанию false
- **timeout**: определяет срок действия куков в минутах

В нашем приложении мы будем хранить данные аутентификации в той же базе данных, что и другие данные, поэтому настройка дополнительной строки подключения нам не потребуется.

Инфраструктура для аутентификации

Теперь надо создать всю необходимую инфраструктуру для аутентификации — модели, контроллеры и представления. Создадим модель `WebUser`, которая будет описывать пользователя:

```
[Table("Firebird.WEBUSER")]
public partial class WEBUSER
{
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
       "CA2214:DoNotCallOverridableMethodsInConstructors")]
    public WEBUSER()
    {
```

```
            WEBUSERINROLES = new HashSet<WEBUSERINROLE>();
        }

        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int WEBUSER_ID { get; set; }

        [Required]
        [StringLength(63)]
        public string EMAIL { get; set; }

        [Required]
        [StringLength(63)]
        public string PASSWD { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
           "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<WEBUSERINROLE> WEBUSERINROLES { get; set; }
    }
```

Добавим ещё две модели: одну для описания ролей `WEBROLE`, и одну для связи ролей с пользователями `WEBUSERINROLE`.

```
    [Table("Firebird.WEBROLE")]
    public partial class WEBROLE
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int WEBROLE_ID { get; set; }

        [Required]
        [StringLength(63)]
        public string NAME { get; set; }

    }

    [Table("Firebird.WEBUSERINROLE")]
    public partial class WEBUSERINROLE
    {

        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int ID { get; set; }

        [Required]
        public int WEBUSER_ID { get; set; }

        [Required]
        public int WEBROLE_ID { get; set; }

        public virtual WEBUSER WEBUSER { get; set; }

        public virtual WEBROLE WEBROLE { get; set; }
    }
```

В классе `DbModel` с помощью Fluent API укажем связи между `WEBUSER` и `WEBUSERINROLE`.

```
…
        public virtual DbSet<WEBUSER> WEBUSERS { get; set; }
        public virtual DbSet<WEBROLE> WEBROLES { get; set; }
        public virtual DbSet<WEBUSERINROLE> WEBUSERINROLES { get; set; }
…
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<WEBUSER>()
                .HasMany(e => e.WEBUSERINROLES)
                .WithRequired(e => e.WEBUSER)
                .WillCascadeOnDelete(false);

…
        }
…
```

Поскольку мы используем технологию Database First, то таблицы в БД могут быть созданы автоматически, но я предпочитаю сам контролировать этот процесс, поэтому приведу здесь скрипт создания дополнительных таблиц.

```
RECREATE TABLE WEBUSER (
  WEBUSER_ID INT NOT NULL,
  EMAIL VARCHAR(63) NOT NULL,
  PASSWD VARCHAR(63) NOT NULL,
  CONSTRAINT PK_WEBUSER PRIMARY KEY(WEBUSER_ID),
  CONSTRAINT UNQ_WEBUSER UNIQUE(EMAIL)
);

RECREATE TABLE WEBROLE (
  WEBROLE_ID INT NOT NULL,
  NAME VARCHAR(63) NOT NULL,
  CONSTRAINT PK_WEBROLE PRIMARY KEY(WEBROLE_ID),
  CONSTRAINT UNQ_WEBROLE UNIQUE(NAME)
);

RECREATE TABLE WEBUSERINROLE (
  ID INT NOT NULL,
  WEBUSER_ID INT NOT NULL,
  WEBROLE_ID INT NOT NULL,
  CONSTRAINT PK_WEBUSERINROLE PRIMARY KEY(ID)
);


ALTER TABLE WEBUSERINROLE
ADD CONSTRAINT FK_WEBUSERINROLE_USER FOREIGN KEY (WEBUSER_ID) REFERENCES WEBUSER (WEBUSER_I

ALTER TABLE WEBUSERINROLE
ADD CONSTRAINT FK_WEBUSERINROLE_ROLE FOREIGN KEY (WEBROLE_ID) REFERENCES WEBROLE (WEBROLE_I
```

```
RECREATE SEQUENCE SEQ_WEBUSER;
RECREATE SEQUENCE SEQ_WEBROLE;
RECREATE SEQUENCE SEQ_WEBUSERINROLE;

SET TERM ^;

RECREATE TRIGGER TBI_WEBUSER
FOR WEBUSER
ACTIVE BEFORE INSERT
AS
BEGIN
  IF (NEW.WEBUSER_ID IS NULL) THEN
    NEW.WEBUSER_ID = NEXT VALUE FOR SEQ_WEBUSER;
END^

RECREATE TRIGGER TBI_WEBROLE
FOR WEBROLE
ACTIVE BEFORE INSERT
AS
BEGIN
  IF (NEW.WEBROLE_ID IS NULL) THEN
    NEW.WEBROLE_ID = NEXT VALUE FOR SEQ_WEBROLE;
END^

RECREATE TRIGGER TBI_WEBUSERINROLE
FOR WEBUSERINROLE
ACTIVE BEFORE INSERT
AS
BEGIN
  IF (NEW.ID IS NULL) THEN
    NEW.ID = NEXT VALUE FOR SEQ_WEBUSERINROLE;
END^

SET TERM ;^
```

Добавим два пользователя и две роли для проверки.

```


INSERT INTO WEBUSER (EMAIL, PASSWD) VALUES ('john', '12345');
INSERT INTO WEBUSER (EMAIL, PASSWD) VALUES ('alex', '123');

COMMIT;

INSERT INTO WEBROLE (NAME) VALUES ('admin');
INSERT INTO WEBROLE (NAME) VALUES ('manager');

COMMIT;

-- Связываем пользователей и роли
INSERT INTO WEBUSERINROLE(WEBUSER_ID, WEBROLE_ID) VALUES(1, 1);
INSERT INTO WEBUSERINROLE(WEBUSER_ID, WEBROLE_ID) VALUES(1, 2);
INSERT INTO WEBUSERINROLE(WEBUSER_ID, WEBROLE_ID) VALUES(2, 2);

COMMIT;
```

> **Замечание о паролях**
>
> Обычно вместо пароля в открытом виде хранят некий хэш от него, например, по алгоритму md5. В нашем примере мы немного упростили аутентификацию.

При регистрации и логине мы не будем напрямую взаимодействовать с моделью `WebUser`. Вместо этого мы будем использовать специальные модели, которые также добавим в проект:

```
namespace FBMVCExample.Models
{
    using System;
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;
    using System.ComponentModel.DataAnnotations.Schema;
    using System.Data.Entity.Spatial;

    // Модель для входа в систему
    public class LoginModel
    {
        [Required]
        public string Name { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }

    // Модель для регистрации нового пользователя
    public class RegisterModel
    {
        [Required]
        public string Name { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Compare("Password", ErrorMessage = "Пароли не совпадают")]
        public string ConfirmPassword { get; set; }
    }
}
```

Эти модели будет использоваться соответственно для представлений логина и регистрации. Эти представление для входа будет выглядеть следующим образом:

```
@model FBMVCExample.Models.LoginModel
```

```
@{
    ViewBag.Title = "Вход";
}

<h2>Вход</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        @Html.ValidationSummary(true)

        <div class="form-group">
            @Html.LabelFor(model => model.Name,
                new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name)
                @Html.ValidationMessageFor(model => model.Name)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Password,
                new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Password)
                @Html.ValidationMessageFor(model => model.Password)
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Вход" class="btn btn-default" />
            </div>
        </div>
    </div>
}

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Соответственно, представление для регистрации будет выглядеть так:

```
@model FBMVCExample.Models.RegisterModel

@{
    ViewBag.Title = "Регистрация";
}

<h2>Регистрация</h2>
```

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        @Html.ValidationSummary(true)

        <div class="form-group">
            @Html.LabelFor(model => model.Name,
                new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name)
                @Html.ValidationMessageFor(model => model.Name)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Password,
                new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Password)
                @Html.ValidationMessageFor(model => model.Password)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.ConfirmPassword,
                new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ConfirmPassword)
                @Html.ValidationMessageFor(model => model.ConfirmPassword)
            </div>
        </div>


        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Зарегистрировать"
                       class="btn btn-default" />
            </div>
        </div>
    </div>
}

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Замечание о пользователях

В данном примере модель, представление и контроллеры для входа и регистрации пользователей предельно упрощены, т.к. обычно пользователь имеет существенно больше атрибутов, чем логин и пароль.

Теперь добавим новый контроллер `AccountController` со следующим содержанием:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Security;
using FBMVCExample.Models;

namespace FBMVCExample.Controllers
{
    public class AccountController : Controller
    {
        public ActionResult Login()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Login(LoginModel model)
        {
            if (ModelState.IsValid)
            {
                // поиск пользователя в бд
                WEBUSER user = null;
                using (DbModel db = new DbModel())
                {
                    user = db.WEBUSERS.FirstOrDefault(
                        u => u.EMAIL == model.Name &&
                             u.PASSWD == model.Password);

                }
                // если нашли пользователя с введённым логином и паролем, то
                // запоминаем его и делаем переадресацию на стартовую страницу
                if (user != null)
                {
                    FormsAuthentication.SetAuthCookie(model.Name, true);
                    return RedirectToAction("Index", "Invoice");
                }
                else
                {
                    ModelState.AddModelError("",
                        "Пользователя с таким логином и паролем не существует");
                }
            }

            return View(model);
        }

        [Authorize(Roles = "admin")]
        public ActionResult Register()
        {
            return View();
        }
```

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Register(RegisterModel model)
        {
            if (ModelState.IsValid)
            {
                WEBUSER user = null;
                using (DbModel db = new DbModel())
                {
                    user = db.WEBUSERS.FirstOrDefault(u => u.EMAIL == model.Name);
                }
                if (user == null)
                {
                    // создаём нового пользователя
                    using (DbModel db = new DbModel())
                    {

                        // получаем новый идентификатор с помощью генератора
                        int userId = db.NextValueFor("SEQ_WEBUSER");
                        db.WEBUSERS.Add(new WEBUSER {
                            WEBUSER_ID = userId,
                            EMAIL = model.Name,
                            PASSWD = model.Password
                        });
                        db.SaveChanges();

                        user = db.WEBUSERS.Where(u => u.WEBUSER_ID == userId)
                                          .FirstOrDefault();

                        // находим роль manager
                        // Эта роль будет ролью по умолчанию, т.е.
                        // будет выдана автоматически при регистрации
                        var defaultRole =
                            db.WEBROLES
                              .Where(r => r.NAME == "manager")
                              .FirstOrDefault();

                        // назначаем вновь добавленному пользователю роль
                        // по умолчанию
                        if (user != null && defaultRole != null)
                        {
                            db.WEBUSERINROLES.Add(new WEBUSERINROLE
                            {
                                WEBUSER_ID = user.WEBUSER_ID,
                                WEBROLE_ID = defaultRole.WEBROLE_ID
                            });
                            db.SaveChanges();
                        }
                    }
                    // если пользователь удачно добавлен в бд
                    if (user != null)
                    {
                        FormsAuthentication.SetAuthCookie(model.Name, true);
                        return RedirectToAction("Login", "Account");
                    }
                }
                else
                {
```

```
                    ModelState.AddModelError("",
                        "Пользователь с таким логином уже существует");
                }
            }

            return View(model);
        }

        public ActionResult Logoff()
        {
            FormsAuthentication.SignOut();
            return RedirectToAction("Login", "Account");
        }
    }
}
```

Обратите внимание на атрибут `[Authorize(Roles = "admin")]`. Он обозначает, что действие по регистрации пользователей может производить только пользователь с ролью admin. Этот механизм называется фильтрами авторизации. Он нём будет сказано чуть позже.

Добавление нового пользователя

При регистрации мы добавляем нового пользователя в БД, а при логине просто смотрим, есть ли такой пользователь. И если пользователь найден, то с помощью аутентификации форм устанавливаем куки

```
FormsAuthentication.SetAuthCookie(model.Name, true);
```

Вся информация о пользователе в Asp.Net MVC хранится в свойстве `HttpContext.User`, которое представляет реализацию интерфейса `IPrincipal`, который определён в пространстве имён System.Security.Principal.

Интерфейс `IPrincipal` определяет свойство `Identity`, которое хранит объект интерфейса `IIdentity`, который описывает текущего пользователя.

Интерфейс `IIdentity` содержит следующие свойства:

- `AuthenticationType`: тип аутентификации
- `IsAuthenticated`: если пользователь аутентифицирован, то возвращает `true`
- `Name`: имя пользователя в системе

Для определения аутентифицирован ли пользователь, ASP.NET MVC принимает от браузера куки, и если пользователь аутентифицирован, у свойства `IIdentity.IsAuthenticated` устанавливается значение `true`, а в свойство `Name` получает в качестве значения имя пользователя.

Теперь добавим элементы авторизации. Для этого воспользуемся механизмом универсальных провайдеров.

Универсальные провайдеры

Универсальные провайдеры предоставляют уже готовый функционал авторизации. Но в то же время эти провайдеры обладают достаточной гибкостью — в частности мы можем их переопределить по своему усмотрению. При этом нам необязательно переопределять и использовать все четыре провайдера. Что довольно удобно, особенно в ситуации, когда нам не нужны все навороты ASP.NET Identity, а требуется построить очень простенькую систему авторизации.

Итак, переопределим провайдер ролей. Для этого добавим через NuGet пакет Microsoft.AspNet.Providers.

Определение провайдера ролей

Теперь определим сам провайдер ролей. Для этого сначала добавим в проект папку `Providers` и затем в него добавим новый класс `MyRoleProvider`:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using FBMVCExample.Models;

namespace FBMVCExample.Providers
{
    public class MyRoleProvider : RoleProvider
    {
        /// <summary>
        /// Возвращает список имён ролей у пользователя
        /// </summary>
        /// <param name="username">Имя пользователя</param>
        /// <returns></returns>
        public override string[] GetRolesForUser(string username)
        {
            string[] roles = new string[] { };
            using (DbModel db = new DbModel())
            {
                // Получаем пользователя
                WEBUSER user = db.WEBUSERS.FirstOrDefault(
                    u => u.EMAIL == username);
                if (user != null)
                {
                    // заполняем массив доступных ролей
                    int i = 0;
                    roles = new string[user.WEBUSERINROLES.Count];
                    foreach (var rolesInUser in user.WEBUSERINROLES)
                    {
                        roles[i] = rolesInUser.WEBROLE.NAME;
                        i++;
                    }
                }
            }
            return roles;
        }
```

```
/// <summary>
/// Создание новой роли
/// </summary>
/// <param name="roleName">Имя роли</param>
public override void CreateRole(string roleName)
{

    using (DbModel db = new DbModel())
    {
        WEBROLE newRole = new WEBROLE() { NAME = roleName };
        db.WEBROLES.Add(newRole);
        db.SaveChanges();
    }
}

/// <summary>
/// Возвращает присутствует ли роль у пользователя
/// </summary>
/// <param name="username">Имя пользователя</param>
/// <param name="roleName">Имя роли</param>
/// <returns></returns>
public override bool IsUserInRole(string username, string roleName)
{
    bool outputResult = false;
    using (DbModel db = new DbModel())
    {
        var userInRole =
            from ur in db.WEBUSERINROLES
            where ur.WEBUSER.EMAIL == username &&
                  ur.WEBROLE.NAME == roleName
            select new { id = ur.ID };

        outputResult = userInRole.Count() > 0;
    }
    return outputResult;
}

public override void AddUsersToRoles(string[] usernames,
    string[] roleNames)
{
    throw new NotImplementedException();
}

public override string ApplicationName
{
    get { throw new NotImplementedException(); }
    set { throw new NotImplementedException(); }
}

public override bool DeleteRole(string roleName,
    bool throwOnPopulatedRole)
{
    throw new NotImplementedException();
}

public override string[] FindUsersInRole(string roleName,
```

```
            string usernameToMatch)
        {
            throw new NotImplementedException();
        }

        public override string[] GetAllRoles()
        {
            throw new NotImplementedException();
        }

        public override string[] GetUsersInRole(string roleName)
        {
            throw new NotImplementedException();
        }

        public override void RemoveUsersFromRoles(string[] usernames,
            string[] roleNames)
        {
            throw new NotImplementedException();
        }

        public override bool RoleExists(string roleName)
        {
            throw new NotImplementedException();
        }
    }
}
```

В целях демонстрации переопределено три метода. Первый из них — `GetRolesForUser` позволяет получать набор ролей для определённого пользователя. Второй метод — `CreateRole` — предполагает создание роли. И третий метод — `IsUserInRole` — определяет, выполняет ли пользователь определённую роль в системе.

Конфигурирование провайдера ролей

Чтобы использовать провайдер ролей в приложении, надо добавить его определение в файл конфигурации. Откроем файл `web.config` и удалим из него определение провайдеров, которые были добавлены автоматически при добавлении пакета Microsoft.AspNet.Providers. И добавим туда вместо этого в пределах узла system.web добавим наш провайдер:

```
<system.web>
    <authentication mode="Forms">
      <forms name="cookies" timeout="2880" loginUrl="~/Account/Login"
             defaultUrl="~/Invoice/Index"/>
    </authentication>
    <roleManager enabled="true" defaultProvider="MyRoleProvider">
      <providers>
        <add name="MyRoleProvider"
             type="FBMVCExample.Providers.MyRoleProvider" />
      </providers>
    </roleManager>
  </system.web>
```

Авторизация доступа к действиям контроллера

И теперь мы можем разграничить доступ к методам различных контроллеров с помощью атрибута `Authorize`. Мы уже видели его применение в контроллере `AccountController`:

```
[Authorize(Roles = "admin")]
public ActionResult Register()
{
…
```

Данный фильтр можно применять как на уровне контроллера в целом, так и для отдельного действия контроллера. Давайте добавим разграничение прав для наших трёх основных контроллеров `CustomerController`, `InvoiceController` и `ProductController`. В нашем случае пользователь с ролью manager может смотреть и править данные во всех трёх таблицах. Установка фильтра для контроллера `InvoiceController` будет выглядеть следующим образом:

```
[Authorize(Roles = "manager")]
public class InvoiceController : Controller
{
    private DbModel db = new DbModel();

    // Отображение представления
    public ActionResult Index()
    {
        return View();
    }
…
```

Исходные коды

Вы можете скачать исходные тексты по ссылке https://github.com/sim1984/FBMVCExample

Глава 5

Создание Web приложений на PHP

В этой главе мы рассмотрим процесс создания web приложения с использованием СУБД Firebird на языке PHP. Предполагается что у вас есть веб-сервер, такой как Apache HTTP Server или Nginx с установленным и настроеным PHP, а так же установленный Firebird сервер.

Взаимодействие PHP и Firebird

Для возможности подключения к СУБД Firebird необходимо установить драйвер.

Обзор драйверов для работы с Firebird

В PHP есть два драйвера для работы с СУБД Firebird:

- РасширениеFirebird/Interbase (ibase_ функции);
- PDO драйвер для Firebird.

Клиентская библиотека Firebird

Оба драйвера требуют, чтобы у вас была установлена клиентская библиотека `fbclient.dll` (для UNIX подобных систем `fbclient.so`) соответствующей разрядности. Т.е. если у вас установлен 64-х разрядный PHP, то его расширениям требуется 64-х разрядная библиотека, для 32-х разрядного PHP — 32-х разрядная библиотека.

> **Замечание для пользователей Win32/Win64**
>
> Для работы этих драйверов в системной переменной Windows PATH должны быть доступен DLL-файл `fbclient.dll`. Хотя копирование DLL-файлов из директории PHP в системную папку Windows также решает проблему (потому что системная директория по умолчанию находится в переменной PATH), это не рекомендуется.

Обзор расширения Firebird/Interbase

Расширение Firebird/Interbase появилось раньше и является наиболее проверенным.

Для установки расширения Firebird/Interbase в конфигурационном файле `php.ini` необходимо раскомментировать строку

```
extension=php_interbase.dll
```

или для UNIX подобных систем строку

```
extension=php_interbase.so
```

Установка Fb/IB Extension в Linux

В Linux это расширение в зависимости от дистрибутива можно установить одной из следующих команд (необходимо уточнить поддерживаемые версии, возможно, необходимо подключить сторонний репозиторий):

```
apt-get install php5-firebird

rpm -ihv php5-firebird

yum install php70w-interbase

zypper install php5-firebird
```

> **Подсказка**
>
> В некоторых случаях вам может потребоваться подключить дополнительные репозитарии для разрешения зависимсотей.

Стиль программирования

Это расширение использует процедурный подход к написанию программ. Функции с префиксом `ibase_` могут возвращать или принимать в качестве одного из параметров идентификатор соединения, транзакции, подготовленного запроса или курсора (результат SELECT запроса). Этот идентификатор имеет тип resource. Все выделенные ресурсы необходимо освобождать, как только они больше не требуются. Я не буду описывать каждую из функций подробно, вы можете посмотреть их описание по адресу http://php.net/ibase, вместо этого приведу несколько небольших примеров с комментариями.

```
<?php

$db = 'localhost:example';
$username = 'SYSDBA';
$password = 'masterkey';

// Подключение к БД
$dbh = ibase_connect($db, $username, $password);
$sql = 'SELECT login, email FROM users';
// Выполняем запрос
$rc = ibase_query($dbh, $sql);
// Получаем результат построчно в виде объекта
while ($row = ibase_fetch_object($rc)) {
    echo $row->email, "\n";
}
// Освобождаем хэндл связанный с результатом запроса
ibase_free_result($rc);
// Освобождаем хэндл связанный с подключением
ibase_close($dbh);
```

ibase_ для соединения с базой данных

Вместо функции `ibase_connect` вы можете применять функцию `ibase_pconnect`, которая создаёт так называемые постоянные соединения. В этом случае при вызове `ibase_close` соединение не закрывается, все связанные с ней ресурсы освобождаются, транзакция по умолчанию подтверждается, другие виды транзакций откатываются. Такое соединение может быть использовано повторно в другой сессии, если параметры подключения совпадают. В некоторых случаях постоянные соединения могут значительно повысить эффективность вашего веб приложения. Это особенно заметно, если затраты на установку соединения велики. Они позволяют дочернему процессу на протяжении всего жизненного цикла использовать одно и то же соединение вместо того, чтобы создавать его при обработке каждой страницы, которая взаимодействует с SQL-сервером. Этим постоянные соединения напоминают работу с пулом соединений. Подробнее о постоянных соединениях вы может прочитать по ссылке http://php.net/persistent-connections.

> **Важно**
>
> Многие ibase функции позволяют не передавать в них идентификатор соединения (транзакции, подготовленного запроса). В этом случае эти функции используют идентификатор последнего установленного соединения (начатой транзакции). Я не рекомендую так делать, в особенности, если ваше веб приложение может использовать более одного подключения.

ibase_query

Функция `ibase_query` выполняет SQL запрос и возвращает идентификатор результата или `true`, если запрос не возвращает набор данных. Эта функция помимо идентификатора подключения (транзакции) и текста SQL запроса может принимать переменное число аргументов в качестве значений параметров SQL запроса. В этом случае наш пример выглядит следующим образом:

```
// …
$sql = 'SELECT login, email FROM users WHERE id=?';
$id = 1;
// Выполняем запрос
$rc = ibase_query($dbh, $sql, $id);
// Получаем результат построчно в виде объекта
if ($row = ibase_fetch_object($rc)) {
    echo $row->email, "\n";
}
// Освобождаем хэндл связанный с результатом запроса
ibase_free_result($rc);
// …
```

Очень часто параметризованные запросы используются многократно с различным набором значений параметров, в этом случае для повышения производительности рекомендуется использовать подготовленные запросы. В этом случае сначала необходимо сначала получить идентификатор подготовленного запроса с помощью функции `ibase_prepare`, а затем выполнять подготовленный запрос с помощью функции `ibase_execute`.

```
// …
$sql = 'SELECT login, email FROM users WHERE id=?';
// Подготавливаем запрос
$sth = ibase_prepare($dbh, $sql);
$id = 1;
// Выполняем запрос
$rc = ibase_execute($sth, $id);
// Получаем результат построчно в виде объекта
if ($row = ibase_fetch_object($rc)) {
    echo $row->email, "\n";
}
// Освобождаем хэндл связанный с результатом запроса
ibase_free_result($rc);
// Освобождаем подготовленный запрос
ibase_free_query($sth);
// …
```

Подготовленные запросы гораздо чаще используются, когда необходима массовая заливка данных.

```
// …
$sql = 'INSERT INTO users(login, email) VALUES(?, ?)';
// Подготавливаем запрос
$sth = ibase_prepare($dbh, $sql);
$users = [["user1", "user1@gmail.com"], ["user2", "user2@gmail.com"]];
// Выполняем запрос
foreach ($users as $user)) {
  ibase_execute($sth, $user[0], $user[1]);
}
// Освобождаем подготовленный запрос
ibase_free_query($sth);
// …
```

По последнему примеру можно увидеть один из недостатков этого расширения, а именно, функции с переменным числом аргументов не очень удобны для параметризованных запросов. Этот недостаток проявляется особенно ярко, если вы пытаетесь написать универсальный класс для исполнения любых запросов. Гораздо удобнее было бы, если параметры можно было передавать одним массивом. Конечно, существуют обходные пути вроде вот такого:

```
function fb_execute ($stmt, $data)
{
   if (!is_array($data))
       return ibase_execute($stmt, $data);
   array_unshift($data, $stmt);
   $rc = call_user_func_array('ibase_execute', $data);
   return $rc;
}
```

Расширение Firebird/Interbase не работает с именованными параметрами запроса.

ibase_trans

По умолчанию расширение Firebird/Interbase автоматически стартует транзакцию по умолчанию после подключения. Транзакцию по умолчанию подтверждается при закрытии соединения функцией `ibase_close`. Её можно подтвердить или откатить раньше, если вызвать методы `ibase_commit` или `ibase_rollback` передав в них идентификатор соединения, или не передавая ни чего (если вы используете единственное соединение).

Если вам необходимо явное управление транзакциями, то необходимо стартовать транзакцию с помощью функции ibase_trans. Если параметры транзакции не указаны, то транзакция будет начата с параметрами `IBASE_WRITE | IBASE_CONCURRENCY | IBASE_WAIT`. Описание констант для задания параметров транзакции можно найти по ссылке php.net/manual/ru/ibase.constants.php. Транзакцию необходимо завершать с помощью функции `ibase_commit` или `ibase_rollback` передавая в эти функции идентификатор транзакции. Если вместо этих функций использовать функции `ibase_commit_ret` или `ibase_rollback_ret`, то транзакция будет завершаться как `COMMIT RETAIN` или `ROLLBACK RETAIN`.

> **Примечание**
>
> Умолчательные параметры транзакции подходят для большинства случаев. Дело в том что соединение с базой данных, как и все связанные с ним ресурсы существуют максимум до конца работы PHP скрипта. Даже если вы используете постоянные соединения, то все связанные ресурсы будут освобождены после вызова функции `ibase_close`. Несмотря на сказанное, настоятельно рекомендую завершать все выделенные ресурсы явно, вызывая соответствующие ibase_ функции.
>
> Пользоваться функциями `ibase_commit_ret` и `ibase_rollback_ret` настоятельно не рекомендую, так как это не имеет смысла. `COMMIT RETAIN` и `ROLLBACK RETAIN` были введены для того, чтобы в настольных приложениях сохранять открытыми курсоры при завершении транзакции.

```
$sql = 'INSERT INTO users(login, email) VALUES(?, ?)';
// Подготавливаем запрос
$sth = ibase_prepare($dbh, $sql);
$users = [["user1", "user1@gmail.com"], ["user2", "user2@gmail.com"]];
$trh = ibase_trans($dbh, IBASE_WRITE | IBASE_CONCURRENCY | IBASE_WAIT);
try {
  // Выполняем запрос
  foreach ($users as $user)) {
    $r = ibase_execute($sth, $user[0], $user[1]);
    // Если произошла ошибка, бросаем исключение
    if ($r === false)
      throw new \Exception(ibase_errmsg());
  }
  ibase_commit($trh);
}
catch(\Exception $e) {
  ibase_rollback($trh);
     echo $e->getMessage();
```

```
}
// Освобождаем подготовленный запрос
ibase_free_query($sth);
```

> **Важно**
>
> ibase функции не бросают исключение в случае возникновения ошибки. Часть функций возвращают `false`, если произошла ошибка. Обращаю ваше внимание, что результат сравнивать с `false` необходимо строгим оператором сравнения `===`. Потенциально ошибка может возникнуть поле вызова любой ibase функции. Текст ошибки можно узнать с помощью функции `ibase_errmsg`. Код ошибки можно получить с помощью функции `ibase_errcode`.

Функции Service API

Расширение Firebird/Interbase позволяет взаимодействовать с сервером Firebird не только посредством SQL запросов, но и используя Service API (см. функции `ibase_service_attach`, `ibase_service_detach`, ibase_server_info, `ibase_maintain_db`, `ibase_db_info`, `ibase_backup`, `ibase_restore`). Эти функции позволяют получить информацию о сервере Firebird, сделать резервное копирование, восстановление или получить статистику. Эта функциональность требуется в основном для администрирования БД, поэтому мы не будем рассматривать её подробно.

Функции для работы с событиями

Расширение Firebird/Interbase так же поддерживает работу с событиями Firebird (см. функции `ibase_set_event_handler`, `ibase_free_event_handler`, `ibase_wait_event`).

Обзор расширения PDO (драйвер Firebird)

Расширение PDO предоставляет обобщённый интерфейс для доступа к различным типам БД. Каждый драйвер базы данных, в котором реализован этот интерфейс, может представить специфичный для базы данных функционал в виде стандартных функций расширения.

PDO и все основные драйверы внедрены в PHP как загружаемые модули. Чтобы их использовать, требуется их просто включить, отредактировав файл `php.ini` следующим образом:

```
extension=php_pdo.dll
```

> **Примечание**
>
> Этот шаг необязателен для версий PHP 5.3 и выше, так как для работы PDO больше не требуются DLL.

Специфичные для Firebird библиотеки

Далее нужно выбрать DLL конкретных баз данных и либо загружать их во время выполнения функцией `dl()`, либо включить их в `php.ini` после `php_pdo.dll`. Например:

```
extension=php_pdo.dll
```

```
extension=php_pdo_firebird.dll
```

Эти DLL должны лежать в директории `extension_dir`.

В Linux это расширение в зависимости от дистрибутива можно установить одной из следующих команд (необходимо уточнить поддерживаемые версии, возможно, необходимо подключить сторонний репозиторий):

```
apt-get install php5-firebird

rpm -ihv php5-firebird

yum install php70w-firebird

zypper install php5-firebird
```

Стиль программирования

PDO использует объектно-ориентированный подход к написанию программ. Какой именно драйвер будет использоваться в PDO, зависит от строки подключения, называемой так же DSN (Data Source Name). DSN состоит из префикса, который и определяет тип базы данных, и набора параметров в виде *<ключ>=<значение>*, разделённых точкой с запятой «;». Допустимый набор параметров зависит от типа базы данных. Для работы с Firebird строка подключения должна начинаться с префикса *firebird:* и иметь вид, описанный в документации в разделе PDO_FIREBIRD DSN.

Соединение с базой данных

Соединения устанавливаются автоматически при создании объекта PDO от его базового класса. Конструктор класса принимает аргументы для задания источника данных (DSN), а также необязательные имя пользователя и пароль (если есть). Четвёртым аргументом можно передать массив специфичных для драйвера настроек подключения в формате *ключ=>значение*.

```
$dsn = 'firebird:dbname=localhost:example;charset=utf8;';
$username = 'SYSDBA';
$password = 'masterkey';
try {
  // Подключение к БД
  $dbh = new \PDO($dsn, $username, $password,
                  [\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION]);
  $sql = 'SELECT login, email FROM users';
  // Выполняем запрос
  $query = $dbh->query($sql);
  // Получаем результат построчно в виде объекта
  while ($row = $query->fetch(\PDO::FETCH_OBJ)) {
    echo $row->email, "\n";
  }
    $query->closeCursor(); // Закрываем курсор
} catch (\PDOException $e) {
  echo $e->getMessage();
}
```

Постоянные соединенения

Для того чтобы PDO использовал постоянные соединения необходимо в конструктор PDO в массиве свойств передать `PDO::ATTR_PERSISTENT => true`.

Обработка исключений

Установив свойство `\PDO::ATTR_ERRMODE` в значение `\PDO::ERRMODE_EXCEPTION`, мы установили режим, при котором любая ошибка, в том числе и ошибка при подключении к БД, будет возбуждать исключение `\PDOException`. Работать в таком режиме гораздо удобнее, чем проверять наличие ошибки после каждого вызова ibase_ функций.

Запросы

Метод `query` выполняет SQL запрос и возвращает результирующий набор в виде объекта `\PDOStatement`. В этот метод помимо SQL запросы вы можете передать способ возвращения значений при фетче. Это может быть столбец, экземпляр заданного класса, объект. Различные способы вызова вы можете посмотреть по ссылке http://php.net/manual/ru/pdo.query.php.

Запросы не возвращающие наборы данных

Если необходимо выполнить SQL запрос, не возвращающий набор данных, то вы можете воспользоваться методом exec, который возвращает количество задействованных строк. Этот метод не обеспечивает выполнение подготовленных запросов.

Параметризированные запросы

Если в запросе используются параметры, то необходимо пользоваться подготовленными запросами. В этом случае вместо метода query необходимо вызвать метод prepare. Этот метод возвращает объект класса `\PDOStatement`, который инкапсулирует в себе методы для работы с подготовленными запросами и их результатами. Для выполнения запроса необходимо вызвать метод execute, который может принимать в качестве аргумента массив с именованными или неименованными параметрами. Результат выполнения селективного запроса можно получить с помощью методов fetch, fetchAll, fetchColumn, fetchObject. Методы `fetch` и `fetchAll` могут возвращать результаты в различном виде: ассоциативный массив, объект или экземпляр определённого класса. Последнее довольно часто используется в MVC паттерне при работе с моделями.

```
$dsn = 'firebird:dbname=localhost:example;charset=utf8;';
$username = 'SYSDBA';
$password = 'masterkey';
try {
  // Подключение к БД
  $dbh = new \PDO($dsn, $username, $password,
                  [\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION]);
  $sql = 'INSERT INTO users(login, email) VALUES(?, ?)';
  $users = [
    ["user1", "user1@gmail.com"],
    ["user2", "user2@gmail.com"]
  ];

  // Подготавливаем запрос
```

```
  $query = $dbh->prepare($sql);
  // Выполняем запрос
  foreach ($users as $user)) {
      $query->execute($user);
  }
} catch (\PDOException $e) {
  echo $e->getMessage();
}
```

Пример использования именованных параметров.

```
$dsn = 'firebird:dbname=localhost:example;charset=utf8;';
$username = 'SYSDBA';
$password = 'masterkey';
try {
  // Подключение к БД
  $dbh = new \PDO($dsn, $username, $password,
                  [\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION]);
  $sql = 'INSERT INTO users(login, email) VALUES(:login, :email)';
     $users = [
    [":login" => "user1", ":email" => "user1@gmail.com"],
    [":login" => "user2", ":email" => "user2@gmail.com"]
  ];
  // Подготавливаем запрос
  $query = $dbh->prepare($sql);
  // Выполняем запрос
  foreach ($users as $user)) {
      $query->execute($user);
  }
} catch (\PDOException $e) {
  echo $e->getMessage();
}
```

Примечание

Для поддержки именованных параметров PDO производит предобработку запроса и заменяет параметры вида :paramname на «?», сохраняя при этом массив соответствия между именем параметра и номерами его позиций в запросе. По этой причине оператор EXECUTE BLOCK не будет работать, если внутри него используются переменные маркированные двоеточием. На данный момент нет никакой возможности заставить работать PDO с оператором EXECUTE BLOCK иначе, например, задать альтернативный префикс параметров, как это сделано в некоторых компонентах доступа.

Связвание

Передать параметры в запрос можно и другим способом, используя так называемое связывание. Метод bindValue привязывает значение к именованному или неименованному параметру. Метод bindParam привязывает переменную к именованному или неименованному параметру. Последний метод особенно полезен для хранимых процедур, которые возвращают значение через OUT или IN OUT параметр (в Firebird механизм возврата значений из хранимых процедур другой).

```
$dsn = 'firebird:dbname=localhost:example;charset=utf8;';
$username = 'SYSDBA';
$password = 'masterkey';
try {
  // Подключение к БД
  $dbh = new \PDO($dsn, $username, $password,
                  [\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION]);
  $sql = 'INSERT INTO users(login, email) VALUES(:login, :email)';
  $users = [
    ["user1", "user1@gmail.com"],
    ["user2", "user2@gmail.com"]
  ];

  // Подготавливаем запрос
  $query = $dbh->prepare($sql);
  // Выполняем запрос
  foreach ($users as $user)) {
    $query->bindValue(":login", $user[0]);
    $query->bindValue(":email", $user[1]);
    $query->execute();
  }
} catch (\PDOException $e) {
  echo $e->getMessage();
}
```

Важно

Нумерация неименованных параметров в методах `bindParam` и `bindValue` начинается с 1.

```
$dsn = 'firebird:dbname=localhost:example;charset=utf8;';
$username = 'SYSDBA';
$password = 'masterkey';
try {
  // Подключение к БД
  $dbh = new \PDO($dsn, $username, $password,
                  [\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION]);
  $sql = 'INSERT INTO users(login, email) VALUES(?, ?)';
  $users = [
    ["user1", "user1@gmail.com"],
    ["user2", "user2@gmail.com"]
  ];

  // Подготавливаем запрос
  $query = $dbh->prepare($sql);
  // Выполняем запрос
  foreach ($users as $user)) {
    $query->bindValue(1, $user[0]);
    $query->bindValue(2, $user[1]);
    $query->execute();
  }
} catch (\PDOException $e) {
```

```
  echo $e->getMessage();
}
```

Транзакции

По умолчанию PDO автоматически подтверждает транзакцию после выполнения каждого SQL запроса, если вам необходимо явное управление транзакциями, то необходимо стартовать транзакцию с помощью метода `\PDO::beginTransaction`. По умолчанию транзакция стартует с параметрами `CONCURRENCY | WAIT | READ_WRITE`. Завершить транзакцию можно методом `\PDO::commit` или `\PDO::rollback`.

```
$dsn = 'firebird:dbname=localhost:example;charset=utf8;';
$username = 'SYSDBA';
$password = 'masterkey';
try {
  // Подключение к БД
  $dbh = new \PDO($dsn, $username, $password,
                  [\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION]);
  // Стартуем транзакцию для обеспечения согласованности между запросами
  $dbh->beginTransaction();
  // Получаем пользователей из одной таблицы
  $users_stmt = $dbh->prepare('SELECT login, email FROM old_users');
  $users_stmt->execute();
  $users = $users_stmt->fetchAll(\PDO::FETCH_OBJECT);
  $users_stmt->closeCursor();

  // И переносим их в другую
  $sql = 'INSERT INTO users(login, email) VALUES(?, ?)';
  // Подготавливаем запрос
  $query = $dbh->prepare($sql);
  // Выполняем запрос
  foreach ($users as $user)) {
    $query->bindValue(1, $user->LOGIN);
    $query->bindValue(2, $user->EMAIL]);
    $query->execute();
  }
  // Подтверждаем транзакцию
  $dbh->commit();
} catch (\PDOException $e) {
  // Если соединение произошло и транзакция стартовала, откатываем её
  if ($dbh &&  $dbh->inTransaction())
    $dbh->rollback();
  echo $e->getMessage();
}
```

К сожалению метод `beginTransaction` не предоставляет возможности изменить параметры транзакции, однако вы можете сделать хитрый трюк, задав параметры транзакции оператором SET TRANSACTION.

```
$dbh = new \PDO($dsn, $username, $password);
$dbh->setAttribute(\PDO::ATTR_AUTOCOMMIT, false);
$dbh->exec("SET TRANSACTION READ ONLY ISOLATION LEVEL READ COMMITTED NO WAIT");
// Выполняем действия в транзакции
// …
$dbh->exec("COMMIT");
$dbh->setAttribute(\PDO::ATTR_AUTOCOMMIT, true);
```

Сравнение драйверов

Ниже представлена сводная таблица возможностей различных драйверов для работы с Firebird.

Таблица 5.1. Сравнение расширений для работы с Firebird

Возможность	Расширение Firebird/ Interbase	PDO
Парадигма программирования	Функциональная	Объектно-ориентированная
Поддерживаемые БД	Firebird, Interbase, Yaffil и другие клоны Interbase.	Любая БД, для которой существует PDO драйвер, в том числе Firebird.
Работа с параметрами запросов	Только неименованные параметры, работать не очень удобно, поскольку используется функция с переменным числом аргументов.	Только неименованные параметры, работать не очень удобно, поскольку используется функция с переменным числом аргументов.
Обработка ошибок	Проверка результата функций `ibase_errmsg`, `ibase_errcode`. Ошибка может произойти после вызова любой ibase функции при этом исключение не будет возбуждено.	Есть возможность установить режим, при котором любая ошибка приведёт к возбуждению исключения.
Управление транзакциями	Даёт возможность задать параметры транзакции.	Не даёт возможность задать параметры транзакции. Есть обходной путь через выполнение оператора SET TRANSACTION.
Специфичные возможности Interbase/Firebird	Есть возможность работать с расширениями Service API (backup, restore, получение статистики и т.д.), а также с событиями базы данных.	Не позволяет использовать специфичные возможности, с которыми невозможно работать, используя SQL.

Из приведённой таблицы видно, что большинству фреймворков гораздо удобнее пользоваться PDO.

Выбор фреймворка для построения WEB приложения

Небольшие web сайты можно писать, не используя паттерн MVC. Однако чем больше становится ваш сайт, тем сложнее его поддерживать, особенно если над ним работает не один человек. Поэтому при разработке нашего web приложения сразу договоримся об использовании этого паттерна.

Итак, мы решили использовать паттерн MVC. Однако написание приложение с использованием этого паттерна не такая простая задача как кажется, особенно если мы не пользуемся сторонними библиотеками. Если всё писать самому, то необходимо решить множество задач: автозагрузка файлов `.php`, включающих определение классов, маршрутизация и др. Для решения этих задач было создано большое количество фреймворков, например Yii, Laravel, Symphony, Kohana и многие другие. Лично мне нравится Laravel, поэтому далее я буду описывать создание приложения с использованием этого фреймворка.

Установка Laravel

Прежде чем устанавливать Laravel вам необходимо убедится, что ваше системное окружение соответствует требованиям.

- PHP >= 5.5.9
- PDO расширение для PHP (для версии 5.1+)
- MCrypt расширение для PHP (для версии 5.0)
- OpenSSL (расширение для PHP)
- Mbstring (расширение для PHP)
- Tokenizer (расширение для PHP)

Установка composer

Laravel использует Composer для управления зависимостями. Поэтому сначала установите Composer, а затем Laravel.

Самый простой способ установить composer под Windows — это скачать и запустить инсталлятор Composer-Setup.exe. Инсталлятор установит Composer и настроит PATH, так что вы можете вызвать Composer из любой директории в командной строке.

Если необходимо установить Composer вручную, то необходимо запустить скрипт

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('SHA384', 'composer-setup.php') ===
'aa96f26c2b67226a324c27919f1eb05f21c248b987e6195cad9690d5c1ff713d53020a02
ac8c217dbf90a7eacc9d141d') { echo 'Installer verified'; } else { echo
'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Этот скрипт осуществляет следующие действия:

- Скачивает инсталлятор в текущую директорию
- Проверяет инсталлятор с помощью SHA-384
- Запускает скрипт инсталляции
- Удаляет скрипт инсталляции

После запуска этого скрипта у вас появится файл `composer.phar` (phar – это архив) — по сути это PHP скрипт, который может принимать несколько команд (install, update, ...) и умеет скачивать и распаковывать библиотеки. Если вы работаете под windows, то вы можете облегчить себе задачу, создав файл `composer.bat` и поместив его в PATH. Для этого необходимо выполнить команду

```
echo @php "%~dp0composer.phar" %*>composer.bat
```

Подробнее об установке composer можно ознакомится по адресу https://getcomposer.org/doc/00-intro.md.

Установка Laravel

Теперь устанавливаем сам Laravel

```
composer global require "laravel/installer"
```

Создание проекта

Если установка прошла успешно, то приступаем к созданию каркаса проекта.

```
laravel new fbexample
```

Ждём завершения, после чего у нас будет создан каркас проекта. Описание структуры каталогов можно ознакомится по адресу http://laravel.su/docs/5.2/structure по Laravel.

Структура нашего проекта

Нас будут интересовать следующие каталоги:

- `app` – основной каталог нашего приложения. В корне каталога будут размещены модели. В подкаталоге Http находится все, что касается работы с браузером. В подкаталоге `Http/Controllers` — наши контроллеры.
- `config` – каталог файлов конфигурации. Подробней о конфигурировании будет написано ниже.
- `public` – корневой каталог web приложения (DocumentRoot). Содержит статические файлы - css, js, изображения и т.п.

- `resources` - здесь находятся шаблоны (Views), файлы локализации и, если таковые имеются, рабочие файлы LESS, SASS и js-приложения на фреймворках типа ReactJS, AngularJS или Ember, которые потом собираются внешним инструментом в папку `public`.

  В корне папки нашего приложения есть файл `composer.json`, который описывает, какие пакеты, потребуются нашему приложению помимо тех, что уже есть в Laravel. Нам потребуется два таких пакета: zofe/rapyd-laravel - для быстрого построения интерфейса с сетками (grids) и диалогами редактирования, и sim1984/laravel-firebird - расширение для работы с СУБД Firebird. Пакет sim1984/laravel-firebird является форком пакета «jacquestvanzuydam/laravel-firebird» с расширенными возможностями поэтому его установка несколько отличается. Позже планируется перенос некоторых возможностей моего пакета в официальный. Не забудьте установить параметр minimum-stability равный dev, так как пакет не является стабильным, а так же добавить ссылки на репозиторий.

```
   "repositories": [
        {
            "type": "package",
            "package": {
                "version": "dev-master",
                "name": "sim1984/laravel-firebird",
                "source": {
                    "url": "https://github.com/sim1984/laravel-firebird",
                    "type": "git",
                    "reference": "master"
                },
                "autoload": {
                    "classmap": [""]
                }
            }
        }
    ],
```

  В секции require добавьте требуемые пакеты следующим образом:

```
        "zofe/rapyd": "2.2.*",
        "sim1984/laravel-firebird": "dev-master"
```

Теперь можно запустить обновление пакетов командой (запускать надо в корне веб приложения)

```
composer update
```

Конфигурация

После выполнения этой команды новые пакеты будут установлены в ваше приложение. Теперь можно приступить к настройке. Для начала выполним команду

```
php artisan vendor:publish
```

которая создаст дополнительные файлы конфигурации для пакета zofe/rapyd.

В файле `config/app.php` добавим два новых провайдера. Для этого добавим две новых записи в ключ providers

```
        Zofe\Rapyd\RapydServiceProvider::class,
        Firebird\FirebirdServiceProvider::class,
```

Теперь перейдём к файлу `config/databases.conf`, который содержит настройки подключения к базе данных. Добавим в ключ connections следующие строки

```
       'firebird' => [
            'driver' => 'firebird',
            'host' => env('DB_HOST', 'localhost'),
            'port' => env('DB_PORT', '3050'),
            'database' => env('DB_DATABASE', 'examples'),
            'username' => env('DB_USERNAME', 'SYSDBA'),
            'password' => env('DB_PASSWORD', 'masterkey'),
            'charset' => env('DB_CHARSET', 'UTF8'),
            'engine_version' => '3.0.0',
        ],
```

Поскольку мы будем использовать наше подключение в качестве подключения по умолчанию, установим следующее

```
'default' => env('DB_CONNECTION', 'firebird'),
```

Обратите внимание на функцию env, которая используется для чтения переменных окружения приложения из специального файла `.env`, находящегося в корне проекта. Исправим в этом файле `.env` следующие строки

```
DB_CONNECTION=firebird
DB_HOST=localhost
DB_PORT=3050
DB_DATABASE=examples
DB_USERNAME=SYSDBA
DB_PASSWORD=masterkey
```

В файле конфигурации `config/rapyd.php` изменим отображение дат так, чтобы они были в формате принятом в России:

```
'fields' => [
        'attributes' => ['class' => 'form-control'],
        'date' => [
```

```
            'format' => 'd.m.Y',
        ],
        'datetime' => [
            'format' => 'd.m.Y H:i:s',
            'store_as' => 'Y-m-d H:i:s',
        ],
    ],
```

Первоначальная настройка закончена, теперь мы можем приступить непосредственно к написанию логики web приложения.

Создание моделей

Фреймворк Laravel поддерживает ORM Eloquent. ORM Eloquent - красивая и простая реализация паттерна ActiveRecord для работы с базой данных. Каждая таблица имеет соответствующий класс-модель, который используется для работы с этой таблицей. Модели позволяют читать данные из таблиц и записывать данные в таблицу.

Инструментарий для создания моделей

Создадим модель заказчиков, для упрощения этого процесса в Laravel есть artisan команда.

```
php artisan make:model Customer
```

Этой командой мы создаём шаблон модели. Теперь изменим нашу модель так, чтобы она выглядела следующим образом:

```
namespace App;

use Firebird\Eloquent\Model;

class Customer extends Model
{
    /**
     * Таблица, связанная с моделью
     *
     * @var string
     */
    protected $table = 'CUSTOMER';

    /**
     * Первичный ключ модели
     *
     * @var string
     */
    protected $primaryKey = 'CUSTOMER_ID';

    /**
```

```
     * Наша модель не имеет временной метки
     *
     * @var bool
     */
    public $timestamps = false;

    /**
     * Имя последовательности для генерации первичного ключа
     *
     * @var string
     */
    protected $sequence = 'GEN_CUSTOMER_ID';
}
```

Важно

Мы используем модифицированную модель `Firebird\Eloquent\Model` из пакета sim1984/laravel-firebird в качестве базовой. Она позволяет воспользоваться последовательностью, указанной в свойстве `$sequence`, для генерирования значения идентификатора первичного ключа.

По аналогии создадим модель товаров – `Product`.

```
namespace App;

use Firebird\Eloquent\Model;

class Product extends Model
{
    /**
     * Таблица, связанная с моделью
     *
     * @var string
     */
    protected $table = 'PRODUCT';

    /**
     * Первичный ключ модели
     *
     * @var string
     */
    protected $primaryKey = 'PRODUCT_ID';

    /**
     * Наша модель не имеет временной метки
     *
     * @var bool
     */
    public $timestamps = false;

    /**
     * Имя последовательности для генерации первичного ключа
```

```
     *
     * @var string
     */
    protected $sequence = 'GEN_PRODUCT_ID';
}
```

Теперь создадим модель для шапки счёт-фактуры.

```
namespace App;

use Firebird\Eloquent\Model;

class Invoice extends Model {

    /**
     * Таблица, связанная с моделью
     *
     * @var string
     */
    protected $table = 'INVOICE';

    /**
     * Первичный ключ модели
     *
     * @var string
     */
    protected $primaryKey = 'INVOICE_ID';

    /**
     * Наша модель не имеет временной метки
     *
     * @var bool
     */
    public $timestamps = false;

    /**
     * Имя последовательности для генерации первичного ключа
     *
     * @var string
     */
    protected $sequence = 'GEN_INVOICE_ID';

    /**
     * Заказчик
     *
     * @return \App\Customer
     */
    public function customer() {
        return $this->belongsTo('App\Customer', 'CUSTOMER_ID');
    }

    /**
     * Позиции счёт фактуры
```

```
     * @return \App\InvoiceLine[]
     */
    public function lines() {
        return $this->hasMany('App\InvoiceLine', 'INVOICE_ID');
    }

    /**
     * Оплата
     */
    public function pay() {
        $connection = $this->getConnection();

        $attributes = $this->attributes;

        $connection->executeProcedure('SP_PAY_FOR_INOVICE',
            [$attributes['INVOICE_ID']]);
    }

}
```

В этой модели можно заметить несколько дополнительных функций. Метод `customer` возвращает заказчика связанного со счёт фактурой через поле `CUSTOMER_ID`. Для осуществления такой связи используется метод `belongsTo`, в который передаются имя класса модели и имя поле связи. Метод `lines` возвращают позиции счёт-фактуры, которые представлены коллекцией моделей `InvoiceLine` (будет описана далее). Для осуществления связи один ко многим в методе `lines` используется метод `hasMany`, в который передаётся имя класса модели и поле связи. Подробнее о задании отношений между сущностями вы можете почитать в разделе Отношения документации Laravel.

Метод `pay` осуществляет оплату счёт фактуры. Для этого вызывается хранимая процедура `SP_PAY_FOR_INVOICE`. В неё передаётся идентификатор счёт фактуры. Значение любого поля (атрибута модели) можно получить из свойства `attributes`. Вызов хранимой процедуры осуществляется с помощью метода `executeProcedure`. Этот метод доступен только при использовании расширения sim1984/laravel-firebird.

Модель позиций счёт-фактур

Теперь создадим модель для позиций счёт фактуры.

```
namespace App;

use Firebird\Eloquent\Model;
use Illuminate\Database\Eloquent\Builder;

class InvoiceLine extends Model {

    /**
     * Таблица, связанная с моделью
     *
     * @var string
     */
```

```
    protected $table = 'INVOICE_LINE';

    /**
     * Первичный ключ модели
     *
     * @var string
     */
    protected $primaryKey = 'INVOICE_LINE_ID';

    /**
     * Наша модель не имеет временной метки
     *
     * @var bool
     */
    public $timestamps = false;

    /**
     * Имя последовательности для генерации первичного ключа
     *
     * @var string
     */
    protected $sequence = 'GEN_INVOICE_LINE_ID';

    /**
      * Массив имён вычисляемых полей
      *
      * @var array
      */
    protected $appends = ['SUM_PRICE'];

    /**
     * Товар
     *
     * @return \App\Product
     */
    public function product() {
        return $this->belongsTo('App\Product', 'PRODUCT_ID');
    }

    /**
     * Сумма по позиции
     *
     * @return double
     */
    public function getSumPriceAttribute() {
        return $this->SALE_PRICE * $this->QUANTITY;
    }

    /**
     * Добавление объекта модели в БД
     * Переопределяем этот метод, т.к. в данном случаем мы работаем с помощью ХП
     *
     * @param  \Illuminate\Database\Eloquent\Builder  $query
     * @param  array  $options
     * @return bool
     */
    protected function performInsert(Builder $query, array $options = []) {
```

```
        if ($this->fireModelEvent('creating') === false) {
            return false;
        }

        $connection = $this->getConnection();

        $attributes = $this->attributes;

        $connection->executeProcedure('SP_ADD_INVOICE_LINE', [
            $attributes['INVOICE_ID'],
            $attributes['PRODUCT_ID'],
            $attributes['QUANTITY']
        ]);

        // We will go ahead and set the exists property to true,
        // so that it is set when the created event is fired, just in case
        // the developer tries to update it during the event. This will allow
        // them to do so and run an update here.
        $this->exists = true;

        $this->wasRecentlyCreated = true;

        $this->fireModelEvent('created', false);

        return true;
    }

    /**
     * Сохранение изменений текущего экземпляра модели в БД
     * Переопределяем этот метод, т.к. в данном случаем мы работаем с помощью ХП
     *
     * @param  \Illuminate\Database\Eloquent\Builder  $query
     * @param  array  $options
     * @return bool
     */
    protected function performUpdate(Builder $query, array $options = []) {
        $dirty = $this->getDirty();

        if (count($dirty) > 0) {
            // If the updating event returns false, we will cancel
            // the update operation so developers can hook Validation systems
            // into their models and cancel this operation if the model does
            // not pass validation. Otherwise, we update.
            if ($this->fireModelEvent('updating') === false) {
                return false;
            }

            $connection = $this->getConnection();

            $attributes = $this->attributes;

            $connection->executeProcedure('SP_EDIT_INVOICE_LINE', [
                $attributes['INVOICE_LINE_ID'],
                $attributes['QUANTITY']
            ]);
```

```
            $this->fireModelEvent('updated', false);
        }
    }

    /**
     * Удаление текущего экземпляра модели в БД
     * Переопределяем этот метод, т.к. в данном случаем мы работаем с помощью ХП
     *
     * @return void
     */
    protected function performDeleteOnModel() {

        $connection = $this->getConnection();

        $attributes = $this->attributes;

        $connection->executeProcedure('SP_DELETE_INVOICE_LINE',
            [$attributes['INVOICE_LINE_ID']]);

    }
}
```

В этой модели есть метод `product`, которая возвращает продукт (модель `App/Product`), указанный в позиции счёт фактуры. Связь осуществляется по полю `PRODUCT_ID` с помощью метода `belongsTo`.

Вычисляемое поле SumPrice вычисляется с помощью функции `getSumPriceAttribute`. Для того чтобы это вычисляемое поле было доступно в модели, его имя должно быть указано в массиве имён вычисляемых полей `$appends`.

Операции

В этой модели мы переопределили операции insert, update и delete так, чтобы они выполнялись, используя хранимые процедуры. Эти хранимые процедуры помимо собственно операций вставки, редактирования и удаления пересчитывают сумму в шапке накладной. Этого можно было бы и не делать, но тогда пришлось бы выполнять в одной транзакции модификацию нескольких моделей. Как это сделать будет показано далее.

Как Laravel оперирует данными

Теперь немного поговорим о том, как работать с моделями в Laravel для выборки, вставки, редактирования и удаления данных. Laravel оперирует данными с помощью конструктора запросов. Полное описание синтаксиса и возможностей этого конструктора вы можете найти по ссылке https://laravel.com/docs/5.2/queries. Например, для получения всех строк поставщиков вы можете выполнить следующий запрос

```
$customers = DB::table('CUSTOMER')->get();
```

Этот конструктор запросов является довольно мощным средством для построения и выполнения SQL запросов. Вы можете выполнять также фильтрация, сортировку и соединения таблиц, например

```
DB::table('users')
        ->join('contacts', function ($join) {
            $join->on('users.id', '=', 'contacts.user_id')->orOn(...);
        })
        ->get()
```

Однако гораздо удобнее работать с использованием моделей. Описание моделей Eloquent ORM и синтаксиса запроса к ним можно найти по ссылке https://laravel.com/docs/5.2/eloquent. Так для получения всех элементов коллекции поставщиков необходимо выполнить следующий запрос

```
$customers = Customer::all();
```

Следующий запрос вернёт первые 20 поставщиков отсортированных по алфавиту.

```
$customers = App\Customer::select()
               ->orderBy('name')
               ->take(20)
               ->get();
```

Сложные модели

Для сложных моделей связанные отношения или коллекции отношений могут быть получены через динамические атрибуты. Например, следующий запрос вернёт позиции счёт-фактуры с идентификатором 1.

```
$lines = Invoice::find(1)->lines;
```

Добавление записей осуществляется через создание экземпляра модели, инициализации его свойств и сохранение модели с помощью метода `save`.

```
$flight = new Flight;
$flight->name = $request->name;
$flight->save();
```

Для изменения запись её необходимо найти, изменить необходимые атрибуты и сохранить методом `save`.

```
$flight = App\Flight::find(1);
$flight->name = 'New Flight Name';
$flight->save();
```

Для удаления записи её необходимо найти и вызвать метод `delete`.

```
$flight = App\Flight::find(1);
$flight->delete();
```

Удалить запись по ключу можно и гораздо быстрее с помощью метода `destroy`. В этом случае можно удалить модель не получая её экземпляр.

```
App\Flight::destroy(1);
```

Существуют и другие способы удаления записей, например «мягкое» удаление. Подробно о способах удаления вы можете прочитать по ссылке https://laravel.com/docs/5.2/eloquent#deleting-models.

Транзакции

Теперь поговорим немного о транзакциях. Что это такое я рассказывать не буду, а лишь покажу, как их можно использовать совместно с Eloquent ORM.

```
DB::transaction(function () {
  // Создаём новую позицию в счёт фактуре
  $line = new App\InvoiceLine();
  $line->CUSTOMER_ID = 45;
  $line->PRODUCT_ID = 342;
  $line->QUANTITY = 10;
  $line->COST = 12.45;
  $line->save();

  // добавляем сумму позиции по строке к сумме накладной
  $invoice = App\Invoice::find($line->CUSTOMER_ID);
  $invoice->INVOICE_SUM += $line->SUM_PRICE;
  $invoice->save();
});
```

Всё что находится в функции обратного вызова, которая является аргументом метода `transaction`, выполняется в рамках одной транзакции.

Создание контроллеров и настройка маршрутизации

Фреймворк Laravel имеет мощную подсистему маршрутизации. Вы можете отображать ваши маршруты, как на простые функции обратного вызова, так и на методы контроллеров. Простейшие примеры маршрутов выглядят вот так

```
Route::get('/', function () {
  return 'Hello World';
});

Route::post('foo/bar', function () {
  return 'Hello World';
});
```

В первом случае мы регистрируем обработчик GET запроса для корня сайта, во втором – для POST запроса с маршрутом `/foo/bar`.

Вы можете зарегистрировать маршрут сразу на несколько типов HTTP запросов, например

```
Route::match(['get', 'post'], 'foo/bar', function () {
  return 'Hello World';
});
```

Из маршрута можно извлекать часть адреса и использовать его в качестве параметров функции-обработчика

```
Route::get('posts/{post}/comments/{comment}', function ($postId, $commentId) {
  //
});
```

Параметры маршрута всегда заключаются в фигурные скобки.

Подробнее о возможности настройки маршрутизации вы можете посмотреть в документации глава Маршрутизация. Маршруты настраиваются в файле `app/Http/routes.php` в Laravel 5.2 и `routes/wep.php` в Laravel 5.3.

Использование контроллеров для обработки запросов

Вместо того чтобы описывать обработку всех запросов в едином файле маршрутизации, мы можем организовать её используя классы контроллеров, которые позволяют группировать связанные обработчики запросов в отдельные классы. Контроллеры хранятся в папке `app/Http/Controllers`.

Все Laravel контроллеры должны расширять базовый класс контроллера `App\Http\Controllers\Controller`, присутствующий в Laravel по умолчанию. Подробнее о написании контроллеров вы можете почитать по ссылке https://laravel.com/docs/5.2/controllers.

Контроллер заказчиков

Напишем наш первый контроллер, который будет отвечать за вывод списка заказчиков и его редактирование.

```
<?php

/*
 * Контроллер заказчиков
 */

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Customer;


class CustomerController extends Controller
{

    /**
     * Отображает список заказчиков
     *
     * @return Response
     */
    public function showCustomers()
    {
        // запрашиваем из модели первые 20 заказчиков
        // отсортированных по алфавиту
        $customers = Customer::select()
                    ->orderBy('NAME')
                    ->take(20)
                    ->get();
   var_dump($customers);
    }

}
```

Теперь необходимо связать методы контроллера с маршрутом. Для этого в `routes.php` (`web.php`) необходимо внести строку

```
Route::get('/customers', 'CustomerController@showCustomers');
```

Здесь имя контроллера отделено от имени метода символом @.

Для быстрого построения интерфейса с сетками и диалогами редактирования будем использовать пакет zofe/rapyd. Мы его уже подключили ранее. Классы пакета zofe/rapyd берут на

себя построение типичных запросов к моделям Eloquent ORM. Изменим контроллер заказчиков так, чтобы он выводил данные в сетку (grid), позволял производить их фильтрацию, а также добавлять, редактировать и удалять записи через диалоги редактирования.

```
<?php

/*
 * Контроллер заказчиков
 */

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Customer;


class CustomerController extends Controller {

    /**
     * Отображает список заказчиков
     *
     * @return Response
     */
    public function showCustomers() {
        // Подключаем виджет для поиска
        $filter = \DataFilter::source(new Customer);
        // Поиск будет по наименованию поставщика
        $filter->add('NAME', 'Наименование', 'text');
        // Задаём подпись кнопке поиска
        $filter->submit('Поиск');
        // Добавляем кнопку сброса фильтра и задаём её подпись
        $filter->reset('Сброс');

        // Создам сетку для отображения отфильтрованных данных
        $grid = \DataGrid::source($filter);

        // выводимые столбцы
        // Поле, подпись, сортируемый
        $grid->add('NAME', 'Наименование', true);
        $grid->add('ADDRESS', 'Адрес');
        $grid->add('ZIPCODE', 'Индекс');
        $grid->add('PHONE', 'Телефон');

        // Добавляем кнопки для просмотра, редактирования и удаления записи
        $grid->edit('/customer/edit', 'Редактирование', 'show|modify|delete');
        // Добавляем кнопку добавления заказчика
        $grid->link('/customer/edit', "Добавление заказчика", "TR");
        // задаём сортировку
        $grid->orderBy('NAME', 'asc');
        // задаём количество записей на страницу
        $grid->paginate(10);
        // отображаем шаблон customer и передаём в него фильтр и грид
        return view('customer', compact('filter', 'grid'));
    }

    /**
```

```
     * Добавление, редактирование и удаление заказчика
     *
     * @return Response
     */
    public function editCustomer() {
        if (\Input::get('do_delete') == 1)
            return "not the first";
        // создаём редактор
        $edit = \DataEdit::source(new Customer());
        // задаём подпись диалога в зависимости от типа операции
        switch ($edit->status) {
            case 'create':
                $edit->label('Добавление заказчика');
                break;
            case 'modify':
                $edit->label('Редактирование заказчика');
                break;
            case 'do_delete':
                $edit->label('Удаление заказчика');
                break;
            case 'show':
                $edit->label('Карточка заказчика');
                // добавляем ссылку для возврата назад на список заказчиков
                $edit->link('customers', 'Назад', 'TR');
                break;
        }
        // задаём что после операций добавления, редактирования и удаления
        // возвращаемся к списку заказчиков
        $edit->back('insert|update|do_delete', 'customers');
        // Добавляем редакторы определённого типа, задаём им подпись
        // и связываем их с атрибутами модели
        $edit->add('NAME', 'Наименование', 'text')->rule('required|max:60');
        $edit->add('ADDRESS', 'Адрес', 'textarea')
             ->attributes(['rows' => 3])
             ->rule('max:250');
        $edit->add('ZIPCODE', 'Индекс', 'text')->rule('max:10');
        $edit->add('PHONE', 'Телефон', 'text')->rule('max:14');
        // отображаем шаблон customer_edit и передаём в него редактор
        return $edit->view('customer_edit', compact('edit'));
    }
}
```

Шаблонизатор blade

Laravel по умолчанию использует шаблонизатор blade. Метод `view` находит необходимый шаблон в директории `resources/views`, делает необходимые замены в нём и возвращает текст HTML страницы. Кроме того, она передаёт в него переменные, которые становятся доступными в шаблоне. Описание синтаксиса шаблонов blade вы можете найти по адресу https://laravel.com/docs/5.2/blade.

Шаблон для отображения заказчиков

Шаблон для отображения заказчиков выглядит следующим образом:

```
@extends('example')

@section('title','Заказчики')

@section('body')

    <h1>Заказчики</h1>
    <p>
        {!! $filter !!}
        {!! $grid !!}
    </p>
@stop
```

Данный шаблон унаследован от шаблона example и переопределяет его секцию body. Переменные `$filter` и `$grid` содержат HTML код для осуществления фильтрации и отображения данных в сетке. Шаблон example является общим для всех страниц.

```
@extends('master')
@section('title', 'Пример работы с Firebird')

@section('body')

  <h1>Пример</h1>

  @if(Session::has('message'))
    <div class="alert alert-success">
        {!! Session::get('message') !!}
    </div>
  @endif

  <p>Пример работы с Firebird.<br/>
  </p>
@stop


@section('content')

    @include('menu')

    @yield('body')

@stop
```

Этот шаблон сам унаследован от шаблона master, кроме того он подключает шаблон menu.

Меню довольно простое, состоит из трёх пунктов Заказчики, Продукты и Счёт фактуры.

```
<nav class="navbar main">
    <div class="navbar-header">
```

```
        <button type="button" class="navbar-toggle"
                data-toggle="collapse" data-target=".main-collapse">
            <span class="sr-only"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
        </button>
    </div>
    <div class="collapse navbar-collapse main-collapse">
        <ul class="nav nav-tabs">
            <li @if (Request::is('customer*'))
                class="active"@endif>{!! link_to("customers", "Заказчики") !!}</li>
            <li @if (Request::is('product*'))
                class="active"@endif>{!! link_to("products", "Товары") !!}</li>
            <li @if (Request::is('invoice*'))
                class="active"@endif>{!! link_to("invoices", "Счёт фактуры") !!}</li>
        </ul>
    </div>
</nav>
```

В шаблоне master подключаются css стили и JavaScript файлы с библиотеками.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>@yield('title', 'Пример Web приложения на Firebird')</title>
        <meta name="description" content="@yield('description',
             'Пример Web приложения на Firebird')" />
        @section('meta', '')


<link href="http://fonts.googleapis.com/css?family=Bitter" rel="stylesheet"
      type="text/css" />
<link href="//netdna.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css"
      rel="stylesheet">
<link href="//maxcdn.bootstrapcdn.com/font-awesome/4.1.0/css/font-awesome.min.css"
      rel="stylesheet">


        {!! Rapyd::styles(true) !!}
    </head>

    <body>
        <div id="wrap">
            <div class="container">
                <br />
                <div class="row">
                    <div class="col-sm-12">
                        @yield('content')
                    </div>
                </div>
```

```
            </div>
        </div>

        <div id="footer">
        </div>

<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
<script src="//netdna.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js">
</script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.pjax/1.9.6
/jquery.pjax.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/riot/2.2.4
/riot+compiler.min.js"></script>

        {!! Rapyd::scripts() !!}
    </body>
</html>
```

Шаблон редактора заказчика customer_edit выглядит следующим образом

```
@extends('example')

@section('title', 'Редактирование заказчика')

@section('body')
    <p>
        {!! $edit !!}
    </p>
@stop
```

Контроллер товаров

Контроллер товаров сделан аналогично контроллеру поставщиков.

```
<?php

/*
 * Контроллер товаров
 */

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Product;


class ProductController extends Controller {
    /**
     * Отображает список продуктов
```

```
     *
     * @return Response
     */
    public function showProducts() {
        // Подключаем виджет для поиска
        $filter = \DataFilter::source(new Product);
        // Поиск будет по наименованию продукта
        $filter->add('NAME', 'Наименование', 'text');
        $filter->submit('Поиск');
        $filter->reset('Сброс');

        // Создам сетку для отображения отфильтрованных данных
        $grid = \DataGrid::source($filter);

        // выводимые столбцы сетки
        // Поле, подпись, сортируемый
        $grid->add('NAME', 'Наименование', true);
        // задаём формат с 2 знаками после запятой
        $grid->add('PRICE|number_format[2,., ]', 'Стоимость');

        $grid->row(function($row) {
            // Денежные величины прижимаем вправо
            $row->cell('PRICE')->style("text-align: right");
        });
        // Добавляем кнопки для просмотра, редактирования и удаления записи
        $grid->edit('/product/edit', 'Редактирование', 'show|modify|delete');
        // Добавляем кнопку добавления товара
        $grid->link('/product/edit', "Добавление товара", "TR");
        // задаём сортировку
        $grid->orderBy('NAME', 'asc');
        // задаём количество записей на страницу
        $grid->paginate(10);
        // отображаем шаблон customer и передаём в него фильтр и грид
        return view('product', compact('filter', 'grid'));
    }

    /**
     * Добавление, редактирование и удаление продуктов
     *
     * @return Response
     */
    public function editProduct() {
        if (\Input::get('do_delete') == 1)
            return "not the first";
        // создаём редактор
        $edit = \DataEdit::source(new Product());
        // задаём подпись диалога в зависимости от типа операции
        switch ($edit->status) {
            case 'create':
                $edit->label('Добавление товара');
                break;
            case 'modify':
                $edit->label('Редактирование товара');
                break;
            case 'do_delete':
                $edit->label('Удаление товара');
                break;
```

```
            case 'show':
                $edit->label('Карточка товара');
                $edit->link('products', 'Назад', 'TR');
                break;
        }
        // задаём что после операций добавления, редактирования и удаления
        // возвращаемся к списку товаров
        $edit->back('insert|update|do_delete', 'products');
        // Добавляем редакторы определённого типа, задаём им подпись
        // и связываем их с атрибутами модели
        $edit->add('NAME', 'Наименование', 'text')->rule('required|max:100');
        $edit->add('PRICE', 'Стоимость', 'text')->rule('max:19');
        $edit->add('DESCRIPTION', 'Описание', 'textarea')
             ->attributes(['rows' => 8])
             ->rule('max:8192');
        // отображаем шаблон product_edit и передаём в него редактор
        return $edit->view('product_edit', compact('edit'));
    }
}
```

Контроллер счёт-фактур

Контроллер счёт фактур является более сложным. В него добавлена дополнительная функция оплаты счёта. Оплаченные счёт фактуры подсвечиваются другим цветом. При просмотре счёт фактуры отображаются так же её позиции. Во время редактирования счёт фактуры есть возможность редактировать и её позиции. Далее я приведу текст контроллера с подробными комментариями.

```
<?php

/*
 * Контроллер счёт фактур
 */

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Invoice;
use App\Customer;
use App\Product;
use App\InvoiceLine;

class InvoiceController extends Controller {

    /**
     * Отображает список счёт-фактур
     *
     * @return Response
     */
    public function showInvoices() {
        // Модель счёт фактур будет одновременно
        // выбирать связанных поставщиков
        $invoices = Invoice::with('customer');
```

```
        // Подключаем виджет для поиска
        $filter = \DataFilter::source($invoices);
        // Позволяем фильтровать по диапазону дат
        $filter->add('INVOICE_DATE', 'Дата', 'daterange');
        // и фильтровать по имени заказчика
        $filter->add('customer.NAME', 'Заказчик', 'text');
        $filter->submit('Поиск');
        $filter->reset('Сброс');

        // Создам сетку для отображения отфильтрованных данных
        $grid = \DataGrid::source($filter);

        // выводимые столбцы сетки
        // Поле, подпись, сортируемый
        // для даты задаём дополнительную функцию,
        // которая преобразует дату в строку
        $grid->add('INVOICE_DATE|strtotime|date[d.m.Y H:i:s]', 'Дата', true);
        // для денег задам формат с двумя знаками после запятой
        $grid->add('TOTAL_SALE|number_format[2,., ]', 'Сумма');
        $grid->add('customer.NAME', 'Заказчик');
        // Значение boolean отображаем как Да/Нет
        $grid->add('PAID', 'Оплачено')
             ->cell(function( $value, $row) {
                      return $value ? 'Да' : 'Нет';
                    });
        // задаём функцию обработки каждой строки
        $grid->row(function($row) {
            // Денежные величины прижимаем вправо
            $row->cell('TOTAL_SALE')->style("text-align: right");
            // окрашиваем оплаченные накладные в другой цвет
            if ($row->cell('PAID')->value == 'Да') {
                $row->style("background-color: #ddffee;");
            }
        });

        // Добавляем кнопки для просмотра, редактирования и удаления записи
        $grid->edit('/invoice/edit', 'Редактирование', 'show|modify|delete');
        // Добавляем кнопку добавления счёт-фактуры
        $grid->link('/invoice/edit', "Добавление счёта", "TR");
        // задаём сортировку
        $grid->orderBy('INVOICE_DATE', 'desc');
        // задаём количество записей на страницу
        $grid->paginate(10);
        // отображаем шаблон customer и передаём в него фильтр и грид
        return view('invoice', compact('filter', 'grid'));
    }

    /**
     * Добавление, редактирование и удаление счет фактуры
     *
     * @return Response
     */
    public function editInvoice() {
        // Получаем текст сохранённой ошибки, если она была
        $error_msg = \Request::old('error_msg');
        // создаём редактор счёт фактуры
        $edit = \DataEdit::source(new Invoice());
        // если счёт оплачен, то генерируем ошибку при попытке его редактирования
```

```
if (($edit->model->PAID) && ($edit->status === 'modify')) {
    $edit->status = 'show';
    $error_msg = 'Редактирование не возможно. Счёт уже оплачен.';
}
// если счёт оплачен, то генерируем ошибку при попытке его удаления
if (($edit->model->PAID) && ($edit->status === 'delete')) {
    $edit->status = 'show';
    $error_msg = 'Удаление не возможно. Счёт уже оплачен.';
}
// задаём подпись диалога в зависимости от типа операции
switch ($edit->status) {
    case 'create':
        $edit->label('Добавление счета');
        break;
    case 'modify':
        $edit->label('Редактирование счета');
        break;
    case 'do_delete':
        $edit->label('Удаление счета');
        break;
    case 'show':
        $edit->label('Счет');
        $edit->link('invoices', 'Назад', 'TR');
        // Если счёт фактуры не оплачена, показываем кнопку оплатить
        if (!$edit->model->PAID)
            $edit->link('invoice/pay/' . $edit->model->INVOICE_ID,
                        'Оплатить', 'BL');
        break;
}

// задаём что после операций добавления, редактирования и удаления
// возвращаемся к списку счет фактур
$edit->back('insert|update|do_delete', 'invoices');

// Задаём для поля дата, что оно обязательное
// По умолчанию ставится текущая дата
$edit->add('INVOICE_DATE', 'Дата', 'datetime')
        ->rule('required')
        ->insertValue(date('Y-m-d H:i:s'));

// Добавляем поле для ввода заказчика. При наборе имени заказчика
// будет отображаться список подсказок
$edit->add('customer.NAME', 'Заказчик', 'autocomplete')
        ->rule('required')
        ->options(Customer::lists('NAME', 'CUSTOMER_ID')
        ->all());
// добавляем поле, которое будет отображать сумму накладной, только для чтения
$edit->add('TOTAL_SALE', 'Сумма', 'text')
        ->mode('readonly')
        ->insertValue('0.00');
// Добавляем галочку Оплачено
$paidCheckbox = $edit->add('PAID', 'Оплачено', 'checkbox')
        ->insertValue('0')
        ->mode('readonly');
$paidCheckbox->checked_output = 'Да';
$paidCheckbox->unchecked_output = 'Нет';

// создаём грид для отображения строк счет фактуры
```

```
        $grid = $this->getInvoiceLineGrid($edit->model, $edit->status);
        // отображаем шаблон invoice_edit и передаём в него редактор и
        // грид для отображения позиций
        return $edit->view('invoice_edit', compact('edit', 'grid', 'error_msg'));
    }

    /**
     * Оплата счёт фактуры
     *
     *  @return Response
     */
    public function payInvoice($id) {
        try {
            // находим счёт фактуру по идентификатору
            $invoice = Invoice::findOrFail($id);
            // вызываем процедуру оплаты
            $invoice->pay();
        } catch (\Illuminate\Database\QueryException $e) {
            // если произошла ошибка, то
            // выделяем текст исключения
            $pos = strpos($e->getMessage(), 'E_INVOICE_ALREADY_PAYED');
            if ($pos !== false) {
                // перенаправляем на страницу редактора и отображаем там ошибку
                return redirect('invoice/edit?show=' . $id)
                       ->withInput(['error_msg' => 'Счёт уже оплачен']);
            } else
                throw $e;
        }
        // перенаправляем на страницу редактора
        return redirect('invoice/edit?show=' . $id);
    }

    /**
     * Получение сетки для строк счета фактуры
     * @param \App\Invoice $invoice
     * @param string $mode
     * @return \DataGrid
     */
    private function getInvoiceLineGrid(Invoice $invoice, $mode) {
        // Получаем строки счёт фактуры
        // Для каждой позиции счёта будет инициализироваться
        // связанный с ним продукт
        $lines = InvoiceLine::with('product')
               ->where('INVOICE_ID', $invoice->INVOICE_ID);

        // Создам сетку для отображения позиций накладной
        $grid = \DataGrid::source($lines);
        // выводимые столбцы сетки
        // Поле, подпись, сортируемый
        $grid->add('product.NAME', 'Наименование');
        $grid->add('QUANTITY', 'Количество');
        $grid->add('SALE_PRICE|number_format[2,., ]', 'Стоимость')
             ->style('min-width: 8em;');
        $grid->add('SUM_PRICE|number_format[2,., ]', 'Сумма')
             ->style('min-width: 8em;');
        // задаём функцию обработки каждой строки
        $grid->row(function($row) {
```

```
            $row->cell('QUANTITY')->style("text-align: right");
            // Денежные величины приживаем вправо
            $row->cell('SALE_PRICE')->style("text-align: right");
            $row->cell('SUM_PRICE')->style("text-align: right");
        });

        if ($mode == 'modify') {
      // Добавляем кнопки для просмотра, редактирования и удаления записи
            $grid->edit('/invoice/editline', 'Редактирование', 'modify|delete');
      // Добавляем кнопку добавления заказчика
            $grid->link('/invoice/editline?invoice_id=' . $invoice->INVOICE_ID,
                        "Добавление позиции", "TR");
        }

        return $grid;
    }

    /**
     * Добавление, редактирование и удаление позиций счет фактуры
     *
     * @return Response
     */
    public function editInvoiceLine() {
        if (\Input::get('do_delete') == 1)
            return "not the first";

        $invoice_id = null;
        // создаём редактор позиции счёт фактуры
        $edit = \DataEdit::source(new InvoiceLine());
        // задаём подпись диалога в зависимости от типа операции
        switch ($edit->status) {
            case 'create':
                $edit->label('Добавление позиции');
                $invoice_id = \Input::get('invoice_id');
                break;
            case 'modify':
                $edit->label('Редактирование позиции');
                $invoice_id = $edit->model->INVOICE_ID;
                break;
            case 'delete':
                $invoice_id = $edit->model->INVOICE_ID;
                break;
            case 'do_delete':
                $edit->label('Удаление позиции');
                $invoice_id = $edit->model->INVOICE_ID;
                break;
        }
        // формируем url для возврата
        $base = str_replace(\Request::path(), '', strtok(\Request::fullUrl(), '?'));
        $back_url = $base . 'invoice/edit?modify=' . $invoice_id;
        // устанавливаем страницу для возврата
        $edit->back('insert|update|do_delete', $back_url);
        $edit->back_url = $back_url;
        // добавляем скрытое поле с кодом счёт фактуры
        $edit->add('INVOICE_ID', '', 'hidden')
             ->rule('required')
             ->insertValue($invoice_id)
             ->updateValue($invoice_id);
```

```
        // Добавляем поле для ввода товара. При наборе имени товара
        // будет отображаться список подсказок
        $edit->add('product.NAME', 'Наименование', 'autocomplete')
             ->rule('required')
             ->options(Product::lists('NAME', 'PRODUCT_ID')->all());
        // поле для ввода количества
        $edit->add('QUANTITY', 'Количество', 'text')
             ->rule('required');
        // отображаем шаблон invoice_line_edit и передаём в него редактор
        return $edit->view('invoice_line_edit', compact('edit'));
    }
}
```

Редактор счёт-фактур

Редактор счёт фактур имеет не стандартный для zofe/rapyd вид, поскольку нам необходимо выводить сетку с позициями счёт фактур. Для этого мы изменили шаблон invoice_edit следующим образом.

```
@extends('example')

@section('title','Редактирование счета')

@section('body')

    <div class="container">
        {!! $edit->header !!}

        @if($error_msg)
            <div class="alert alert-danger">
                <strong>Ошибка!</strong> {{ $error_msg }}
            </div>
        @endif

        {!! $edit->message !!}

        @if(!$edit->message)

            <div class="row">
                <div class="col-sm-4">
                    {!! $edit->render('INVOICE_DATE') !!}
                    {!! $edit->render('customer.NAME') !!}
                    {!! $edit->render('TOTAL_SALE') !!}
                    {!! $edit->render('PAID') !!}
                </div>
            </div>

            {!! $grid !!}

        @endif

        {!! $edit->footer !!}
    </div>
@stop
```

Изменение маршрутов

Теперь, когда все контроллеры написаны, изменим маршруты так, чтобы наш сайт на стартовой странице открывал список счёт фактур. Напоминаю, что маршруты настраиваются в файле `app/Http/routes.php` в Laravel 5.2 и `routes/wep.php` в Laravel 5.3.

```
// Корневой маршрут
Route::get('/', 'InvoiceController@showInvoices');

Route::get('/customers', 'CustomerController@showCustomers');
Route::any('/customer/edit', 'CustomerController@editCustomer');

Route::get('/products', 'ProductController@showProducts');
Route::any('/product/edit', 'ProductController@editProduct');

Route::get('/invoices', 'InvoiceController@showInvoices');
Route::any('/invoice/edit', 'InvoiceController@editInvoice');
Route::any('/invoice/pay/{id}', 'InvoiceController@payInvoice');
Route::any('/invoice/editline', 'InvoiceController@editInvoiceLine');
```

Здесь маршрут `/invoice/pay/{id}` выделяет идентификатор счёт фактуры из адреса и передаёт его в метод `payInvoice`. Остальные маршруты не требуют отдельного пояснения.

Результат

Напоследок приведу несколько скриншотов получившегося веб приложения.

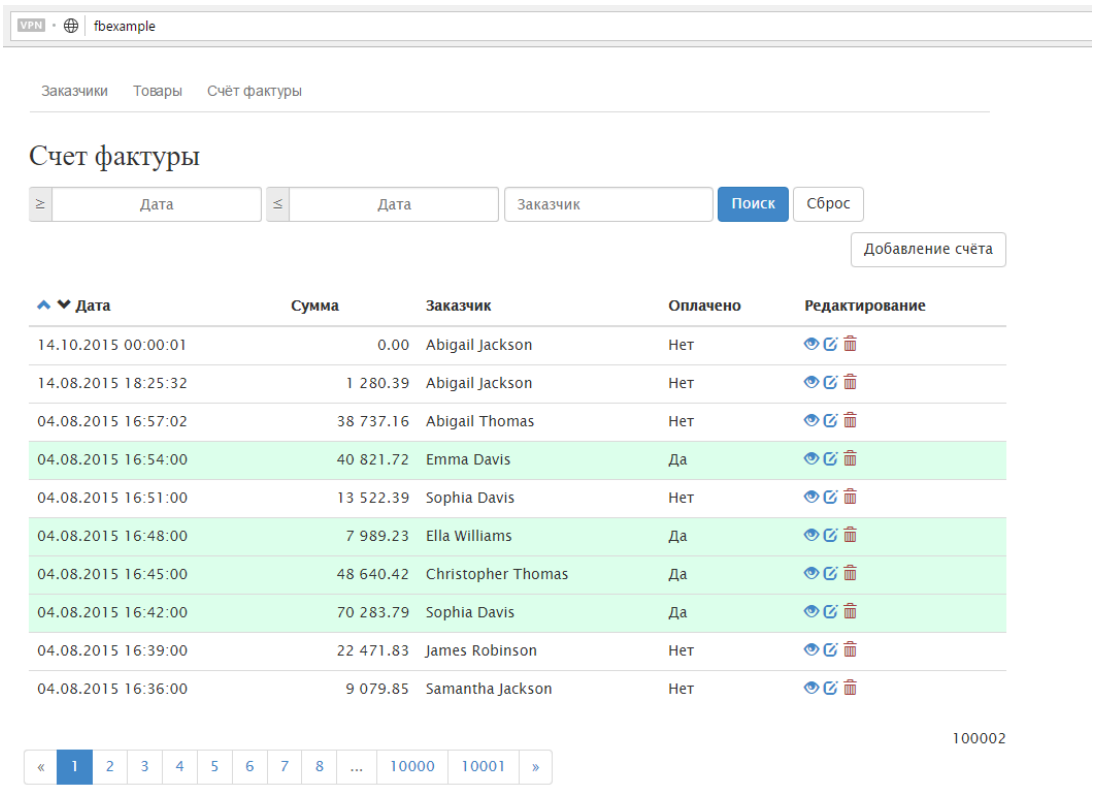

Рис. 5.1. Страница с гридом счёт-фактуры

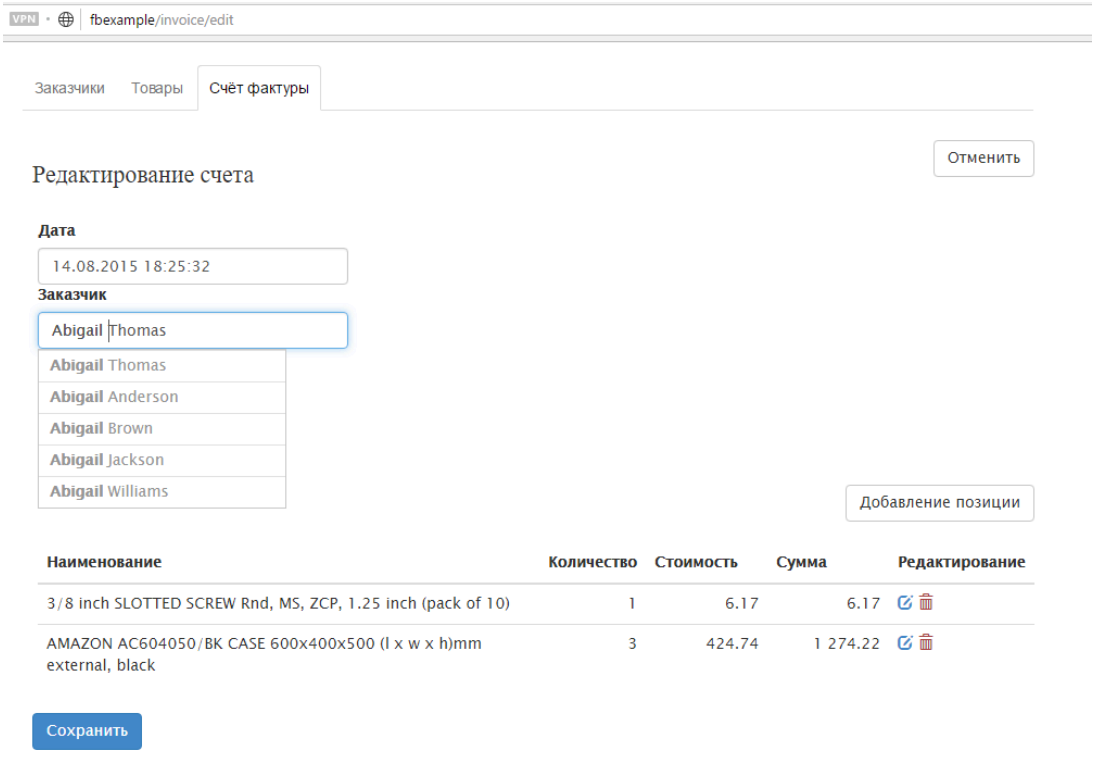

Рис. 5.2. Страница с редактором счёт-фактуры

Исходный код

На этом мой пример закончен. Исходные коды вы можете скачать по ссылке https://github.com/sim1984/phpfbexample.

Глава 6

Создание приложений с использованием jOOQ и Spring MVC

В данной главе будет описан процесс создания web приложения на языке Java с использованием фреймворка Spring MVC, библиотеки jOOQ и СУБД Firebird.

Для упрощения разработки вы можете воспользоваться одной из распространённых IDE для Java (NetBeanse, IntelliJ IDEA, Eclipse, JDeveloper или др.). Лично я использовал NetBeanse. Для тестирования и отладки нам так же потребуется установить один и веб-серверов или серверов приложения (Apache Tomcat или Glass Fish). Создаём проект на основе шаблона Maven проекта веб-приложения.

Организация структуры папок

После создания проекта на основе шаблона необходимо преобразовать его структуру папок так чтобы она была корректной для Spring 4. Если проект создавался в среде NetBeans 8.2, то необходимо выполнить следующие шаги:

1. Удалить файл `index.html`
2. Создать папку `WEB-INF` внутри папки `Web Pages`
3. Внутри папки `WEB-INF` создать папки `jsp`, `jspf` и `resources`
4. Внутри папки `resources` создаём папки `js` и `CSS`
5. Внутри папки `jsp` создаём файл `index.jsp`

После наших манипуляций структура папок должна выглядеть следующим образом.

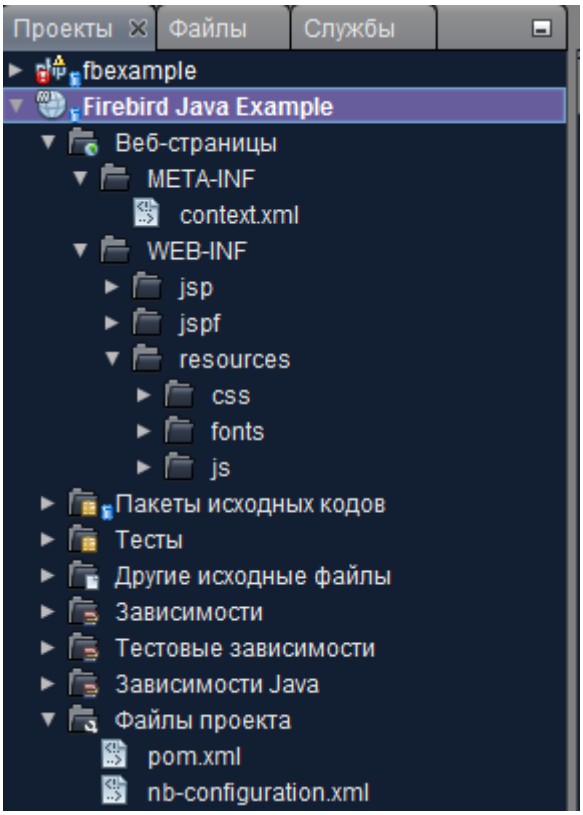

Рис. 6.1. Структура каталогов Spring MVC проекта

В папке `WEB-INF/jsp` будут размещаться jsp странице, а в папке `WEB-INF/jspf` части страниц, которые будут подключены в другие странице с помощью инструкции

```
<%@ include file ="<имя файла>" %>
```

Папка `resource` предназначена для размещения статических веб ресурсов. В папке `WEB-INF/resources/css` будут размещаться файлы каскадных таблиц стилей, в папке `WEB-INF/resources/fonts` – файлы шрифтов, в папке `WEB-INF/resources/js` – файлы JavaScript и сторонние JavaScript библиотеки.

Теперь поправим файл `pom.xml` и пропишем в него общие свойства приложения, зависимости от пакетов библиотек (Spring MVC, Jaybird, JDBC пул, JOOQ) и свойства JDBC подключения.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>ru.ibase</groupId>
    <artifactId>fbjavaex</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>Firebird Java Example</name>
```

```
<properties>
    <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring.version>4.3.4.RELEASE</spring.version>
    <jstl.version>1.2</jstl.version>
    <javax.servlet.version>3.0.1</javax.servlet.version>
    <jaybird.version>3.0.0</jaybird.version>
    <org.jooq.version>3.9.2</org.jooq.version>
    <db.url>jdbc:firebirdsql://localhost:3050/examples</db.url>
    <db.driver>org.firebirdsql.jdbc.FBDriver</db.driver>
    <db.username>SYSDBA</db.username>
    <db.password>masterkey</db.password>
</properties>


<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>7.0</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>${javax.servlet.version}</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>${jstl.version}</version>
    </dependency>

    <!-- Работа с JSON -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.8.5</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.8.5</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.8.5</version>
    </dependency>


    <!-- Spring -->
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
</dependency>

<!-- JDBC -->

<dependency>
    <groupId>org.firebirdsql.jdbc</groupId>
    <artifactId>jaybird-jdk18</artifactId>
    <version>${jaybird.version}</version>
</dependency>

<!-- Пул коннектов -->

<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.4</version>
</dependency>

<!-- jOOQ -->

<dependency>
    <groupId>org.jooq</groupId>
    <artifactId>jooq</artifactId>
    <version>${org.jooq.version}</version>
</dependency>

<dependency>
    <groupId>org.jooq</groupId>
    <artifactId>jooq-meta</artifactId>
```

```
            <version>${org.jooq.version}</version>
        </dependency>

        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jooq-codegen</artifactId>
            <version>${org.jooq.version}</version>
        </dependency>

        <!-- Testing -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <type>jar</type>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
            <scope>test</scope>
        </dependency>

    </dependencies>

    <build>

        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                    <compilerArguments>
                        <endorseddirs>${endorsed.dir}</endorseddirs>
                    </compilerArguments>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.3</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <version>2.6</version>
                <executions>
                    <execution>
                        <phase>validate</phase>
                        <goals>
```

```
                        <goal>copy</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>${endorsed.dir}</outputDirectory>
                        <silent>true</silent>
                        <artifactItems>
                            <artifactItem>
                                <groupId>javax</groupId>
                                <artifactId>javaee-endorsed-api</artifactId>
                                <version>7.0</version>
                                <type>jar</type>
                            </artifactItem>
                        </artifactItems>
                    </configuration>
                </execution>
            </executions>
        </plugin>
        <!-- Jetty -->
        <plugin>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-maven-plugin</artifactId>
            <version>9.4.8.v20171121</version>
        </plugin>
        <!-- The jOOQ code generator plugin -->
        <plugin>
            <groupId>org.jooq</groupId>
            <artifactId>jooq-codegen-maven</artifactId>
            <version>${org.jooq.version}</version>

            <executions>
                <execution>
                    <id>generate-firebird</id>
                    <phase>generate-sources</phase>
                    <goals>
                        <goal>generate</goal>
                    </goals>
                    <configuration>
                        <jdbc>
                            <driver>${db.driver}</driver>
                            <url>${db.url}</url>
                            <user>${db.username}</user>
                            <password>${db.password}</password>
                            <properties>
                                <property>
                                    <key>charSet</key>
                                    <value>utf-8</value>
                                </property>
                            </properties>
                        </jdbc>

                        <generator>
                            <name>org.jooq.util.JavaGenerator</name>

                            <database>
                                <!-- Тип базы данных. Формат:
                                org.util.[database].[database]Database -->
                                <name>org.jooq.util.firebird.FirebirdDatabase</name>
```

```
                                    <inputSchema></inputSchema>

<!-- Все объекты, которые генерируются из
вашей схемы (Регулярное выражение Java. Используйте фильтры, чтобы
ограничить количество объектов). Следите за чувствительностью к
регистру. В зависимости от вашей базы данных, это может быть важно! -->
                                    <includes>.*</includes>

<!-- Объекты, которые исключаются при генерации из вашей схемы.
(Регулярное выражение Java). В данном случае мы исключаем системные
таблицы RDB$, таблицы мониторинга MON$ и псевдотаблицы
безопасности SEC$. -->
                                    <excludes>
                                        RDB\$.*
                                        | MON\$.*
                                        | SEC\$.*
                                    </excludes>
                                </database>

                                <target>
                                    <!-- имя пакета -->
                                    <packageName>ru.ibase.fbjavaex.exampledb</packageName>

                                    <!-- директория для сгенерированых классов -->
                                    <directory>src/main/java/</directory>
                                </target>
                            </generator>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

После того как вы прописали все необходимые зависимости, желательно перезагрузить POM, чтобы загрузить все необходимые библиотеки. Если этого не сделать, то в процессе работы с проектом могут возникать ошибки. В NetBeans это делается следующим образом

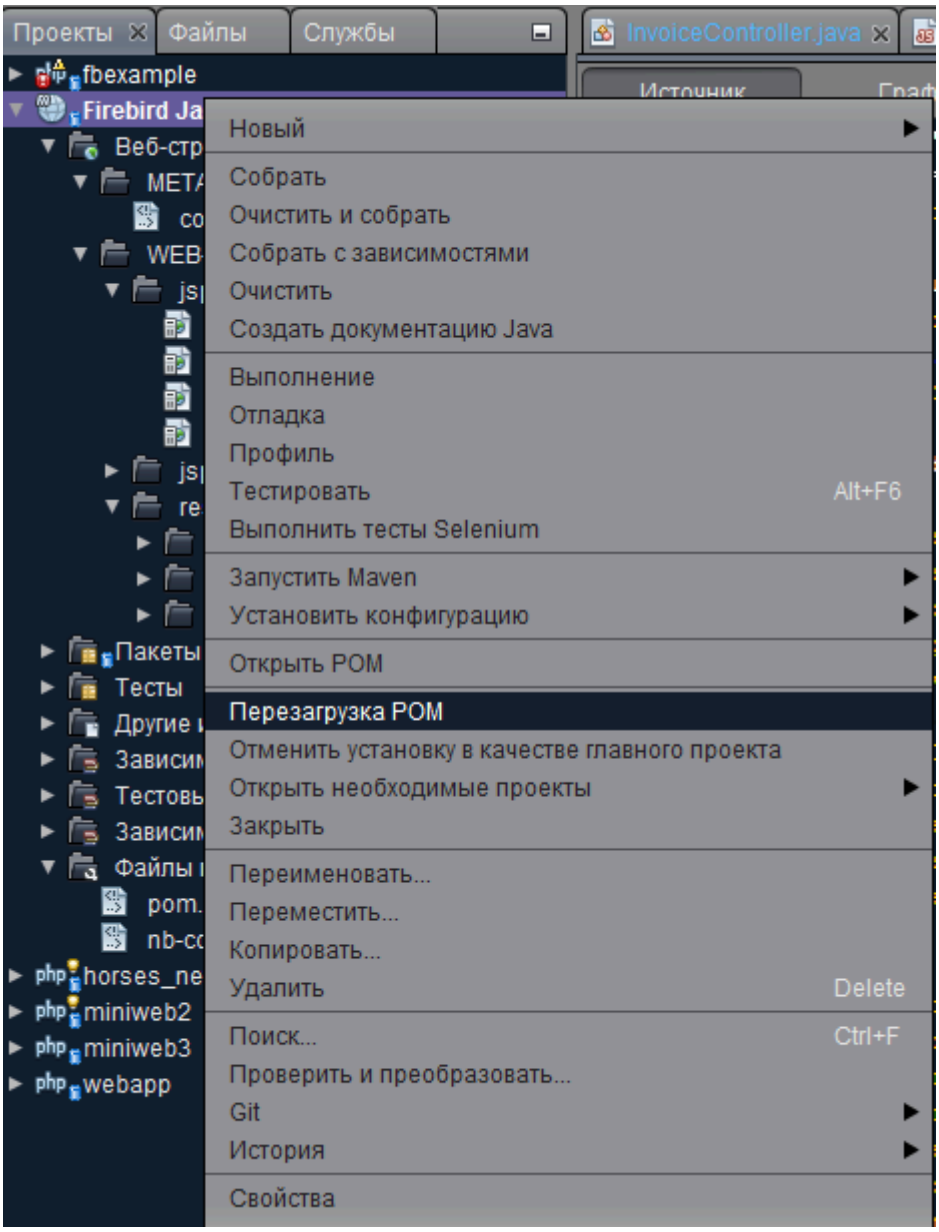

Рис. 6.2. Перезагрузка POM

Кодирование конфигурации

Мне не очень нравится конфигурирование через xml, поэтому я буду работать через классы конфигурации Java.

```
package ru.ibase.fbjavaex.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
```

```
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import org.springframework.web.servlet.view.JstlView;
import org.springframework.web.servlet.view.UrlBasedViewResolver;
import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
import org.springframework.http.converter.HttpMessageConverter;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import java.util.List;

@Configuration
@ComponentScan("ru.ibase.fbjavaex")
@EnableWebMvc
public class WebAppConfig extends WebMvcConfigurerAdapter {

    @Override
    public void configureMessageConverters(
      List<HttpMessageConverter<?>> httpMessageConverters) {
        MappingJackson2HttpMessageConverter jsonConverter =
            new MappingJackson2HttpMessageConverter();
        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS,
                               false);
        jsonConverter.setObjectMapper(objectMapper);
        httpMessageConverters.add(jsonConverter);
    }

    @Bean
    public UrlBasedViewResolver setupViewResolver() {
        UrlBasedViewResolver resolver = new UrlBasedViewResolver();
        resolver.setPrefix("/WEB-INF/jsp/");
        resolver.setSuffix(".jsp");
        resolver.setViewClass(JstlView.class);
        return resolver;
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**")
                .addResourceLocations("/WEB-INF/resources/");
    }
}
```

В данном конфигурационном классе мы задаём место поиска веб ресурсов и JSP представлений. Метод `configureMessageConverters` устанавливает, что дата должна сериализоваться в строковое представление (по умолчанию сериализуется в числовом представлении как timestamp).

Написание кода WebInitializer

Теперь избавимся от файла `Web.xml` вместо него создадим файл `WebInitializer.java`.

```
package ru.ibase.fbjavaex.config;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration.Dynamic;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class WebInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        AnnotationConfigWebApplicationContext ctx =
            new AnnotationConfigWebApplicationContext();
        ctx.register(WebAppConfig.class);
        ctx.setServletContext(servletContext);
        Dynamic servlet = servletContext.addServlet("dispatcher",
                                                    new DispatcherServlet(ctx));
        servlet.addMapping("/");
        servlet.setLoadOnStartup(1);
    }

}
```

Осталось сконфигурировать IoC контейнеры для внедрения зависимостей. К этому шагу мы вернёмся позже, а сейчас перейдём к генерации классов для работы с базой данных через jOOQ.

Генерации классов для работы с базой данных через jOOQ

Работу с базой данных будем вести с помощью библиотеки jOOQ. jOOQ позволяет строить SQL запросы из объектов jOOQ и кода (наподобие LINQ). jOOQ имеет более тесную интеграцию с базой данных, чем ORM, поэтому кроме простых CRUD SQL запросов используемых в Active Record, позволяет использовать дополнительные возможности. Например, jOOQ умеет работать с хранимыми процедурами и функциями, последовательностями, использовать оконные функции и другие, специфичные для определённой СУБД, возможности. Полная документация по работе с jOOQ находится по адресу http://www.jooq.org/doc/3.9/manual-single-page/

Классы jOOQ

Классы jOOQ для работы с базой данных генерируются на основе схемы базы данных, описанной в главе Создание базы данных для примеров.

Для генерации классов jOOQ, работающих с нашей БД, необходимо скачать следующие бинарные файлы по ссылке http://www.jooq.org/download или через maven репозиторий:

- `jooq-3.9.2.jar` — главная библиотека, которая включается в наше приложение для работы с jOOQ.
- `jooq-meta-3.9.2.jar` — утилита, которая включается в вашу сборку для навигации по схеме базы данных через сгенерированные объекты;
- `jooq-codegen-3.9.2.jar` — утилита, которая включается в вашу сборку для генерации схемы базы данных.

Кроме того для подключения к БД Firebird через JDBC вам потребуется скачать драйвер jaybird-full-3.0.0.jar.

Конфигурация для генерации классов схемы базы данных

Теперь надо создать файл конфигурации `example.xml`, который будет использован для генерации классов схемы БД.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration xmlns="http://www.jooq.org/xsd/jooq-codegen-3.8.0.xsd">
  <!-- Конфигурация подключения к БД -->
  <jdbc>
    <driver>org.firebirdsql.jdbc.FBDriver</driver>
    <url>jdbc:firebirdsql://localhost:3050/examples</url>
    <user>SYSDBA</user>
    <password>masterkey</password>
    <properties>
      <property>
        <key>charSet</key>
        <value>utf-8</value>
      </property>
    </properties>
  </jdbc>

  <generator>
    <name>org.jooq.util.JavaGenerator</name>

    <database>
      <!-- Тип базы данных. Формат:
           org.util.[database].[database]Database -->
      <name>org.jooq.util.firebird.FirebirdDatabase</name>

      <inputSchema></inputSchema>

      <!-- Все объекты, которые генерируются из вашей схемы
           (Регулярное выражение Java. Используйте фильтры, чтобы ограничить
           количество объектов).
           Следите за чувствительностью к регистру. В зависимости от вашей
           базы данных, это может быть важно! -->
      <includes>.*</includes>

      <!-- Объекты, которые исключаются при генерации из вашей схемы.
           (Регулярное выражение Java).
           В данном случае мы исключаем системные таблицы RDB$, таблицы
           мониторинга MON$ и псевдотаблицы безопасности SEC$. -->
      <excludes>
          RDB\$.*
```

```
        | MON\$.*
        | SEC\$.*
      </excludes>
    </database>

    <target>
      <!-- Имя пакета в который будут выгружены сгенерированные классы -->
      <packageName>ru.ibase.fbjavaex.exampledb</packageName>

      <!-- Директория для размещения сгенерированных классов.
           Здесь используется структура директорий Maven. -->
      <directory>e:/OpenServer/domains/localhost/fbjavaex/src/main/java/</directory>
    </target>
  </generator>
</configuration>
```

Генерация классов схемы

Теперь переходим в командную строку и выполняем следующую команду:

```
java -cp jooq-3.9.2.jar;jooq-meta-3.9.2.jar;jooq-codegen-3.9.2.jar;
jaybird-full-3.0.0.jar;. org.jooq.util.GenerationTool example.xml
```

Данная команда создаст необходимые классы и позволит писать на языке Java запросы к объектам БД. Подробнее с процессом генерации классов вы можете ознакомиться по ссылке https://www.jooq.org/doc/3.9/manual-single-page/#code-generation.

Генерация классов схемы при сборке приложения

Для упрощения процесса генерации классов схемы мы включили соотвествующие разделы в `pom.xml`. Таким образом генерация классов схем происходит при сборке приложения с помощью Maven.

Внедрение зависимостей

В Spring внедрение зависимостей (Dependency Injection (DI)) осуществляется через Spring IoC (Inversion of Control) контейнер. Внедрение зависимостей, является процессом, согласно которому объекты определяют свои зависимости, т.е. объекты, с которыми они работают, через аргументы конструктора/фабричного метода или свойства, которые были установлены или возвращены фабричным методом. Затем контейнер inject (далее "внедряет") эти зависимости при создании бина. Подробнее о внедрении зависимостей вы можете почитать по ссылке http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans.

Конфигурация IoC контейнеров

Я не сторонник xml конфигурации, поэтому мы будем использовать подход на основе аннотаций и Java-конфигурации. Основными признаками и частями Java-конфигурации IoC контейнера являются классы с аннотацией *`@Configuration`* и методы с аннотацией *`@Bean`*.

Анотация @Bean

Аннотация `@Bean` используется для указания того, что метод создает, настраивает и инициализирует новый объект, управляемый Spring IoC контейнером. Такие методы можно использовать как в классах с аннотацией `@Configuration`. Наш IoC контейнер будет возвращать пул подключений, менеджер транзакций, транслятор исключений (преобразует исключения `SQLException` в специфичные для Spring исключения `DataAccessException`), DSL контекст (стартовая точка, для построения всех запросов используя Fluent API), а также менеджеры для реализации бизнес логики и гриды для отображения данных.

```
/**
 * Конфигурация IoC контейнера
 * для осуществления внедрения зависимостей.
 */

package ru.ibase.fbjavaex.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;
import org.apache.commons.dbcp.BasicDataSource;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy;
import org.jooq.impl.DataSourceConnectionProvider;
import org.jooq.DSLContext;
import org.jooq.impl.DefaultDSLContext;
import org.jooq.impl.DefaultConfiguration;
import org.jooq.SQLDialect;
import org.jooq.impl.DefaultExecuteListenerProvider;

import ru.ibase.fbjavaex.exception.ExceptionTranslator;

import ru.ibase.fbjavaex.managers.*;
import ru.ibase.fbjavaex.jqgrid.*;

/**
 * Конфигурационный класс Spring IoC контейнера
 */
@Configuration
public class JooqConfig {


    /**
     * Возвращает пул коннектов
     *
     * @return
     */
    @Bean(name = "dataSource")
    public DataSource getDataSource() {
        BasicDataSource dataSource = new BasicDataSource();
        // определяем конфигурацию подключения
        dataSource.setUrl("jdbc:firebirdsql://localhost:3050/examples");
        dataSource.setDriverClassName("org.firebirdsql.jdbc.FBDriver");
```

```
        dataSource.setUsername("SYSDBA");
        dataSource.setPassword("masterkey");
        dataSource.setConnectionProperties("charSet=utf-8");
        return dataSource;
    }

    /**
     * Возвращает менеджер транзакций
     *
     * @return
     */
    @Bean(name = "transactionManager")
    public DataSourceTransactionManager getTransactionManager() {
        return new DataSourceTransactionManager(getDataSource());
    }

    @Bean(name = "transactionAwareDataSource")
    public TransactionAwareDataSourceProxy getTransactionAwareDataSource() {
        return new TransactionAwareDataSourceProxy(getDataSource());
    }

    /**
     * Возвращает провайдер подключений
     *
     * @return
     */
    @Bean(name = "connectionProvider")
    public DataSourceConnectionProvider getConnectionProvider() {
        return new DataSourceConnectionProvider(getTransactionAwareDataSource());
    }

    /**
     * Возвращает транслятор исключений
     *
     * @return
     */
    @Bean(name = "exceptionTranslator")
    public ExceptionTranslator getExceptionTranslator() {
        return new ExceptionTranslator();
    }

    /**
     * Возвращает конфигурацию DSL контекста
     *
     * @return
     */
    @Bean(name = "dslConfig")
    public org.jooq.Configuration getDslConfig() {
        DefaultConfiguration config = new DefaultConfiguration();
        // используем диалект SQL СУБД Firebird
        config.setSQLDialect(SQLDialect.FIREBIRD);
        config.setConnectionProvider(getConnectionProvider());
        DefaultExecuteListenerProvider listenerProvider =
          new DefaultExecuteListenerProvider(getExceptionTranslator());
        config.setExecuteListenerProvider(listenerProvider);
        return config;
    }
```

```
    /**
     * Возвращает DSL контекст
     *
     * @return
     */
    @Bean(name = "dsl")
    public DSLContext getDsl() {
        org.jooq.Configuration config = this.getDslConfig();
        return new DefaultDSLContext(config);
    }

    /**
     * Возвращает менеджер заказчиков
     *
     * @return
     */
    @Bean(name = "customerManager")
    public CustomerManager getCustomerManager() {
        return new CustomerManager();
    }

    /**
     * Возвращает грид с заказчиками
     *
     * @return
     */
    @Bean(name = "customerGrid")
    public JqGridCustomer getCustomerGrid() {
        return new JqGridCustomer();
    }

    /**
     * Возвращает менеджер продуктов
     *
     * @return
     */
    @Bean(name = "productManager")
    public ProductManager getProductManager() {
        return new ProductManager();
    }

    /**
     * Возвращает грид с товарами
     *
     * @return
     */
    @Bean(name = "productGrid")
    public JqGridProduct getProductGrid() {
        return new JqGridProduct();
    }

    /**
     * Возвращает менеджер счёт фактур
     *
     * @return
     */
```

```
    @Bean(name = "invoiceManager")
    public InvoiceManager getInvoiceManager() {
        return new InvoiceManager();
    }

    /**
     * Возвращает грид с заголовками счёт фактур
     *
     * @return
     */
    @Bean(name = "invoiceGrid")
    public JqGridInvoice getInvoiceGrid() {
        return new JqGridInvoice();
    }

    /**
     * Возвращает грид с позициями счёт фактуры
     *
     * @return
     */
    @Bean(name = "invoiceLineGrid")
    public JqGridInvoiceLine getInvoiceLineGrid() {
        return new JqGridInvoiceLine();
    }

    /**
     * Возвращает рабочий период
     *
     * @return
     */
    @Bean(name = "workingPeriod")
    public WorkingPeriod getWorkingPeriod() {
        return new WorkingPeriod();
    }

}
```

Построение SQL запросов используя jOOQ

Прежде чем рассматривать реализацию менеджеров и сеток (grids) расскажем, как работать с базой данных через jOOQ. Здесь будут изложены лишь краткие сведения о построении запросов, полную документацию по этому вопросу вы можете найти в главе sql-building документации jOOQ.

Класс `org.jooq.impl.DSL` является основным классом, от которого вы будете создавать все объекты jOOQ. Он выступает в роли статической фабрики для табличных выражений, выражений столбцов (или полей), условных выражений и многих других частей запроса.

`DSLContext` ссылается на объект `org.jooq.Configuration`, который настраивает поведение jOOQ, при выполнении запросов. В отличие от статического `DSL`, `DSLContext` позволяет создавать SQL-операторы, которые уже "настроены" и готовы к выполнению. В нашем приложении `DSLContext` создаётся в классе конфигурации `JooqConfig` в методе `getDsl`.

Конфигурация для `DSLContext` возвращается методом `getDslConfig`. В этом методе мы указали, что будем использовать диалект SQL СУБД Firebird, провайдер подключений (определяет, как мы получаем подключение через JDBC) и слушатель выполнения SQL запросов.

jOOQ DSL

jOOQ поставляется с собственным DSL (или Domain Specific Language), который эмулирует SQL в Java. Это означает, что вы можете писать SQL-операторы почти так, как если бы Java изначально поддерживал их, примерно так же, как .NET в C# делает это с помощью LINQ к SQL.

jOOQ использует неформальную BNF нотацию, которая моделирует унифицированный SQL диалект, подходящий для большинства СУБД. В отличие от других, более простых фреймворков, которые используют "Fluent API" или "метод цепочек", иерархия интерфейса BNF на основе jOOQ не позволяет плохой синтаксис запросов.

Давайте рассмотрим простой запрос на языке SQL

```
SELECT *
 FROM author a
 JOIN book b ON a.id = b.author_id
WHERE a.year_of_birth > 1920
  AND a.first_name = 'Paulo'
ORDER BY b.title
```

В jOOQ он будет выглядеть следующим образом:

```
Result<Record> result =
dsl.select()
   .from(AUTHOR.as("a"))
   .join(BOOK.as("b")).on(a.ID.equal(b.AUTHOR_ID))
   .where(a.YEAR_OF_BIRTH.greaterThan(1920)
   .and(a.FIRST_NAME.equal("Paulo")))
   .orderBy(b.TITLE)
   .fetch();
```

Классы `AUTHOR` и `BOOK`, описывающие соответствующие таблицы должны быть сгенерированы заранее. Процесс генерации классов jOOQ по заданной схеме БД был описан выше.

В данном случае мы задали таблицам `AUTHOR` и `BOOK` алиас с помощью конструкции `as`. Без использования алиасов этот запрос выглядел бы следующим образом

```
Result<Record> result =
dsl.select()
   .from(AUTHOR)
```

```
    .join(BOOK).on(AUTHOR.ID.equal(BOOK.AUTHOR_ID))
    .where(AUTHOR.YEAR_OF_BIRTH.greaterThan(1920)
    .and(AUTHOR.FIRST_NAME.equal("Paulo")))
    .orderBy(BOOK.TITLE)
    .fetch();
```

Теперь посмотрим более сложный запрос с использованием агрегатных функций и группировки.

```
SELECT AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME, COUNT(*)
FROM AUTHOR
  JOIN BOOK ON AUTHOR.ID = BOOK.AUTHOR_ID
WHERE BOOK.LANGUAGE = 'DE'
  AND BOOK.PUBLISHED > '2008-01-01'
GROUP BY AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME
  HAVING COUNT(*) > 5
ORDER BY AUTHOR.LAST_NAME ASC NULLS FIRST
  OFFSET 1 ROWS
  FETCH FIRST 2 ROWS ONLY
```

В jOOQ он будет выглядеть так:

```
dsl.select(AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME, count())
    .from(AUTHOR)
    .join(BOOK).on(BOOK.AUTHOR_ID.equal(AUTHOR.ID))
    .where(BOOK.LANGUAGE.equal("DE"))
    .and(BOOK.PUBLISHED.greaterThan("2008-01-01"))
    .groupBy(AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME)
    .having(count().greaterThan(5))
    .orderBy(AUTHOR.LAST_NAME.asc().nullsFirst())
    .limit(2)
    .offset(1)
    .fetch();
```

Заметьте ограничение на количество возвращаемых записей, будет сгенерировано в соответствии с указанным диалектом SQL. В примере выше использовался диалект `FIREIRD_3_0`. Если бы был указан диалект `FIREBIRD_2_5` или просто `FIREBIRD`, то использовалось бы предложение ROWS вместо OFFSET … FETCH.

Вы можете собирать запрос по частям. Это позволяет менять его динамически, что можно использовать для изменения порядка сортировки или добавления дополнительных параметров фильтрации.

```
SelectFinalStep<?> select
    = dsl.select()
         .from(PRODUCT);
```

```
SelectQuery<?> query = select.getQuery();
switch (searchOper) {
    case "eq":
         query.addConditions(PRODUCT.NAME.eq(searchString));
         break;
    case "bw":
         query.addConditions(PRODUCT.NAME.startsWith(searchString));
         break;
    case "cn":
         query.addConditions(PRODUCT.NAME.contains(searchString));
         break;
}
switch (sOrd) {
    case "asc":
         query.addOrderBy(PRODUCT.NAME.asc());
         break;
    case "desc":
         query.addOrderBy(PRODUCT.NAME.desc());
         break;
}
return query.fetchMaps();
```

Именованные и неименованные параметры

По умолчанию каждый раз, когда вы используете в запросе литера строк, дат и чисел, а также подставляете внешние переменные, jOOQ делает привязку этой переменной или литерала через неименованные параметры. Например, следующее выражение на языке Java

```
dsl.select()
   .from(BOOK)
   .where(BOOK.ID.equal(5))
   .and(BOOK.TITLE.equal("Animal Farm"))
   .fetch();
```

Эквивалентно полной форме записи

```
dsl.select()
   .from(BOOK)
   .where(BOOK.ID.equal(val(5)))
   .and(BOOK.TITLE.equal(val("Animal Farm")))
   .fetch();
```

и преобразуется в sql запрос

```
SELECT *
```

```
FROM BOOK
WHERE BOOK.ID = ?
  AND BOOK.TITLE = ?
```

Вам не нужно беспокоиться какой индекс у соответствующего параметра, значения автоматически будут привязаны к нужному параметру. Если нужно изменить значение параметра, то вы можете сделать это, выбрав нужный параметр по номеру индекса (индексация начинается с 1).

```
Select<?> select =
  dsl.select()
     .from(BOOK)
     .where(BOOK.ID.equal(5))
     .and(BOOK.TITLE.equal("Animal Farm"));
Param<?> param = select.getParam("2");
Param.setValue("Animals as Leaders");
```

Другим способом присвоить параметру новое значение является вызов метода `bind`.

```
Query query1 =
  dsl.select()
     .from(AUTHOR)
     .where(LAST_NAME.equal("Poe"));
query1.bind(1, "Orwell");
```

Кроме того, jOOQ поддерживает именованные параметры. В этом случае их надо явно создавать, используя `org.jooq.Param`.

```
// Create a query with a named parameter. You can then use that name for
// accessing the parameter again
Query query1 =
  dsl.select()
     .from(AUTHOR)
     .where(LAST_NAME.equal(param("lastName", "Poe")));
Param<?> param1 = query.getParam("lastName");

// Or, keep a reference to the typed parameter in order
// not to lose the <T> type information:
Param<String> param2 = param("lastName", "Poe");
Query query2 =
  dsl.select()
     .from(AUTHOR)
     .where(LAST_NAME.equal(param2));

// You can now change the bind value directly on the Param reference:
```

```
param2.setValue("Orwell");
```

Другим способом присвоить параметру новое значение является вызов метода `bind`.

```
// Or, with named parameters
Query query2 =
  dsl.select()
     .from(AUTHOR)
     .where(LAST_NAME.equal(param("lastName", "Poe")));
query2.bind("lastName", "Orwell");
```

Возврат значений из селективных запросов

jOOQ предоставляет множество способов извлечения данных из SQL запросов. Мы не будем перечислять здесь все способы, подробнее вы можете прочитать о них в главе Fetching документации jOOQ. Мы в своём примере будем пользоваться возвратом в список карт (метод `fetchMaps`), который удобно использовать для сериализации результата в JSON.

Другие типы запросов

Теперь посмотрим, как выглядят другие типы запросов. Например, следующий запрос для вставки записи

```
INSERT INTO AUTHOR
       (ID, FIRST_NAME, LAST_NAME)
VALUES (100, 'Hermann', 'Hesse');
```

в jOOQ будет выглядеть так

```
dsl.insertInto(AUTHOR,
        AUTHOR.ID, AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME)
   .values(100, "Hermann", "Hesse")
   .execute();
```

Вот такой запрос для обновления записи

```
UPDATE AUTHOR
   SET FIRST_NAME = 'Hermann',
       LAST_NAME = 'Hesse'
```

```
 WHERE ID = 3;
```

с использованием jOOQ записывается следующим образом

```
dsl.update(AUTHOR)
   .set(AUTHOR.FIRST_NAME, "Hermann")
   .set(AUTHOR.LAST_NAME, "Hesse")
   .where(AUTHOR.ID.equal(3))
   .execute();
```

Запрос на удаление записи

```
DELETE FROM AUTHOR
 WHERE ID = 100;
```

выглядит так

```
dsl.delete(AUTHOR)
   .where(AUTHOR.ID.equal(100))
   .execute();
```

Кроме того, в jOOQ можете строить более сложные модифицирующие запросы, например MERGE.

Хранимые процедуры в jOOQ

Большим преимуществом jOOQ является поддержка работы с хранимыми процедурами. Хранимые процедуры извлекаются в пакет *.Routines.* после чего с ними можно удобно работать, например следующий код на Java

```
int invoiceId = dsl.nextval(GEN_INVOICE_ID).intValue();

spAddInvoice(dsl.configuration(),
             invoiceId,
             customerId,
             invoiceDate);
```

эквивалентен получению следующего значения генератора с помощью SQL запроса

```
SELECT NEXT VALUE FOR GEN_INVOICE_ID FROM RDB$DATABASE
```

и последующего вызова хранимой процедуры

```
EXECUTE PROCEDURE SP_ADD_INVOICE(:INVOICE_ID, :CUSTOMER_ID, :INVOICE_DATE);
```

jOOQ также предоставляет вам средства для построения простых DDL запросов, но мы не будет их рассматривать здесь.

Работа с транзакциями

По умолчанию jOOQ работает в режиме авто подтверждения транзакции, т.е. на каждый SQL оператор стартует новая транзакция, и если в процессе выполнения SQL оператора не было ошибок транзакция подтверждается, в противном случае откатывается. По умолчанию используется транзакция с параметрами `READ_WRITE READ_COMMITED REC_VERSION WAIT`. То есть те же самые что используются JDBC драйвером. Изменить режим изолированности по умолчанию можно через параметры пула соединений (см. `BasicDataSource.setDefaultTransactionIsolation` в методе `getDataSource` класса конфигурации `JooqConfig`).

Явные транзакции

В jOOQ существует несколько способов явного управления транзакциями. Поскольку мы разрабатываем приложение с использованием Spring Framework, будем использовать менеджер транзакций заданный в конфигурации (JooqConfig). Получить менеджер транзакций можно, объявив в классе свойство `txMgr` следующим образом:

```
@Autowired
private DataSourceTransactionManager txMgr;
```

В этом случае типичный сценарий работы с транзакцией выглядит следующим образом:

```
TransactionStatus tx = txMgr.getTransaction(new DefaultTransactionDefinition());
try {
    // действия внутри транзакции
    for (int i = 0; i < 2; i++)
        dsl.insertInto(BOOK)
           .set(BOOK.ID, 5)
           .set(BOOK.AUTHOR_ID, 1)
           .set(BOOK.TITLE, "Book 5")
           .execute();
```

```
    // подтверждение транзакции
    txMgr.commit(tx);
}
catch (DataAccessException e) {
    // откат транзакции
    txMgr.rolback(tx);
}
```

Однако Spring позволяет осуществить подобный сценарий намного проще с помощью аннотации `@Transactional` указанной перед методом класса. В этом случае все действия, производимые в методе, будут обёрнуты транзакцией.

```
    /**
     * Удаление заказчика
     *
     * @param customerId
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void delete(int customerId) {
        this.dsl.deleteFrom(CUSTOMER)
                .where(CUSTOMER.CUSTOMER_ID.eq(customerId))
                .execute();
    }
```

Параметры транзакции

Параметр *propagation* задаёт, каким образом будет вестись работа с транзакциями, если наш метод вызывается из внешней транзакции.

- `Propagation.REQUIRED` — выполняться в существующей транзакции, если она есть, иначе создавать новую;
- `Propagation.MANDATORY` — выполняться в существующей транзакции, если она есть, иначе генерировать исключение;
- `Propagation.SUPPORTS` — выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции;
- `Propagation.NOT_SUPPORTED` — всегда выполняться вне транзакции. Если есть существующая, то она будет остановлена;
- `Propagation.REQUIRES_NEW` — всегда выполняться в новой независимой транзакции. Если есть существующая, то она будет остановлена до окончания выполнения новой транзакции;
- `Propagation.NESTED` — если есть текущая транзакция, выполняться в новой, так называемой, вложенной транзакции. Если вложенная транзакция будет отменена, то это не повлияет на внешнюю транзакцию; если будет отменена внешняя транзакция, то будет отменена и вложенная. Если текущей транзакции нет, то просто создаётся новая;

- `Propagation.NEVER` — всегда выполнять вне транзакции, при наличии существующей генерировать исключение.

Параметр *`isolation`* указывает режим изолированности транзакции. Поддерживается 5 значений: `DEFAULT`, `READ_UNCOMMITTED`, `READ_COMMITTED`, `REPEATABLE_READ`, `SERIALIZABLE`. Если указано значение параметра `DEFAULT`, то будет использоваться умолчательный режим изолированности транзакции. Остальные режимы изолированности взяты из SQL стандарта. В Firebird несколько другие режимы изолированности и полностью соответствует всем критериям лишь режим READ_COMMITED. Таким образом, режим в JDBC `READ_COMMITTED` отображает на `read_committed` в Firebird, `REPEATABLE_READ` — `concurrency` (Snapshot), а `SERIALIZABLE` — `consistency`. Кроме того, помимо режима изолированности, Firebird поддерживает дополнительные параметры транзакции (`NO RECORD_VERSION` и `RECORD_VERSION`, `WAIT` и `NO WAIT`). Вы можете настроить отображение стандартных уровней изолированности на параметры транзакции Firebird с помощью задания свойств JDBC соединения (подробнее см. в Jaybird 2.1 JDBC driver Java Programmer's Manual в главе Using transactions). Если ваша транзакция работает более чем с 1 запросом, то рекомендуется режим изолированности `REPEATABLE_READ` для обеспечения согласованности данных.

В аннотации `@Transactional` вы можете задать, является ли транзакция только для чтения с помощью свойства `readOnly`. По умолчанию транзакция находится в режиме read-write.

Написание кода приложения

Данные нашего приложения мы будем отображать с помощью JavaScript компонента jqGrid. В настоящий момент jqGrid распространяется по коммерческой лицензии, но бесплатен для некоммерческих целей. Вместо него вы можете воспользоваться форком free-jqGrid. Для отображения данных в данном гриде и элементов постраничной навигации нам требуется вернуть данные в формате JSON, структура которых выглядит следующим образом.

```
{
    total: 100,
    page: 3,
    records: 3000,
    rows: [
       {id: 1, name: "Ada"},
       {id: 2, name: "Smith"},
      …
    ]
}
```

где

- `total` — общее количество страниц;
- `page` — номер текущей страницы;
- `records` — общее количество записей;
- `rows` — массив записей на текущей странице.

Создадим класс который описывает данную структуру:

```
package ru.ibase.fbjavaex.jqgrid;

import java.util.List;
import java.util.Map;

/**
 * Класс описывающий структуру которая используется jqGrid
 * Предназначен для сериализации в JSON
 *
 * @author Simonov Denis
 */
public class JqGridData {

    /**
     * Total number of pages
     */
    private final int total;

    /**
     * The current page number
     */
    private final int page;

    /**
     * Total number of records
     */
    private final int records;

    /**
     * The actual data
     */
    private final List<Map<String, Object>> rows;

    /**
     * Конструктор
     *
     * @param total
     * @param page
     * @param records
     * @param rows
     */
    public JqGridData(int total, int page, int records,
                      List<Map<String, Object>> rows) {
        this.total = total;
        this.page = page;
        this.records = records;
        this.rows = rows;
    }

    /**
     * Возвращает общее количество страниц
     *
     * @return
```

```
     */
    public int getTotal() {
        return total;
    }

    /**
     * Возвращает текущую страницу
     *
     * @return
     */
    public int getPage() {
        return page;
    }

    /**
     * Возвращает общее количество записей
     *
     * @return
     */
    public int getRecords() {
        return records;
    }

    /**
     * Возвращает список карт
     * Это массив данных для отображения в гриде
     *
     * @return
     */
    public List<Map<String, Object>> getRows() {
        return rows;
    }
}
```

Теперь напишем абстрактный класс, который будет возвращать вышеописанную структуру в зависимости от условий поиска и сортировки. Этот класс будет предком классов возвращающие подобные структуры для конкретных сущностей.

```
/*
 * Абстрактный класс для работы с JqGrid
 */
package ru.ibase.fbjavaex.jqgrid;

import java.util.Map;
import java.util.List;
import org.jooq.DSLContext;
import org.springframework.beans.factory.annotation.Autowired;

/**
 * Работа с JqGrid
 *
 * @author Simonov Denis
 */
```

```
public abstract class JqGrid {


    @Autowired(required = true)
    protected DSLContext dsl;

    protected String searchField = "";
    protected String searchString = "";
    protected String searchOper = "eq";
    protected Boolean searchFlag = false;
    protected int pageNo = 0;
    protected int limit = 0;
    protected int offset = 0;
    protected String sIdx = "";
    protected String sOrd = "asc";

    /**
     * Возвращает общее количество записей
     *
     * @return
     */
    public abstract int getCountRecord();

    /**
     * Возвращает структуру для сериализации в JSON
     *
     * @return
     */
    public JqGridData getJqGridData() {
        int recordCount = this.getCountRecord();
        List<Map<String, Object>> records = this.getRecords();

        int total = 0;
        if (this.limit > 0) {
            total = recordCount / this.limit + 1;
        }

        JqGridData jqGridData = new JqGridData(
            total,
            this.pageNo,
            recordCount,
            records);
        return jqGridData;
    }


    /**
     * Возвращает количество записей на странице
     *
     * @return
     */
    public int getLimit() {
        return this.limit;
    }

    /**
     * Возвращает смещение для извлечения первой записи на странице
```

```
     *
     * @return
     */
    public int getOffset() {
        return this.offset;
    }

    /**
     * Возвращает имя поля для сортировки
     *
     * @return
     */
    public String getIdx() {
        return this.sIdx;
    }

    /**
     * Возвращает порядок сортировки
     *
     * @return
     */
    public String getOrd() {
        return this.sOrd;
    }

    /**
     * Возвращает номер текущей страницы
     *
     * @return
     */
    public int getPageNo() {
        return this.pageNo;
    }

    /**
     * Возвращает массив записей как список карт
     *
     * @return
     */
    public abstract List<Map<String, Object>> getRecords();

    /**
     * Возвращает поле для поиска
     *
     * @return
     */
    public String getSearchField() {
        return this.searchField;
    }

    /**
     * Возвращает значение для поиска
     *
     * @return
     */
    public String getSearchString() {
        return this.searchString;
```

```
    }

    /**
     * Возвращает операцию поиска
     *
     * @return
     */
    public String getSearchOper() {
        return this.searchOper;
    }

    /**
     * Устанавливает ограничение на количество выводимых записей
     *
     * @param limit
     */
    public void setLimit(int limit) {
        this.limit = limit;
    }

    /**
     * Устанавливает количество записей, которые надо пропустить
     *
     * @param offset
     */
    public void setOffset(int offset) {
        this.offset = offset;
    }

    /**
     * Устанавливает сортировку
     *
     * @param sIdx
     * @param sOrd
     */
    public void setOrderBy(String sIdx, String sOrd) {
        this.sIdx = sIdx;
        this.sOrd = sOrd;
    }

    /**
     * Устанавливает номер текущей страницы
     *
     * @param pageNo
     */
    public void setPageNo(int pageNo) {
        this.pageNo = pageNo;
        this.offset = (pageNo - 1) * this.limit;
    }

    /**
     * Устанавливает условие поиска
     *
     * @param searchField
     * @param searchString
     * @param searchOper
     */
```

```
    public void setSearchCondition(String searchField, String searchString,
                                   String searchOper) {
        this.searchFlag = true;
        this.searchField = searchField;
        this.searchString = searchString;
        this.searchOper = searchOper;
    }
}
```

Важно

Данный класс содержит свойство DSLContext dsl, которое будет использоваться для построения запросов на выборку данных с помощью jOOQ.

Создание справочников

Теперь мы можем приступить к созданию справочников. Мы опишем процесс создания справочников на примере справочника заказчиков. Справочник продуктов создаётся схожим образом, и при желании вы можете просмотреть его исходный код по ссылке, приведённой в конце этой главы.

Сначала реализуем класс для работы с jqGrid, он будет наследоваться от нашего абстрактного класса `ru.ibase.fbjavaex.jqgrid.JqGrid` описанного выше. В нём имеется возможность поиска и разнонаправленной сортировки по полю NAME. В листинге исходного кода будут приведены поясняющие комментарии.

```
package ru.ibase.fbjavaex.jqgrid;

import org.jooq.*;
import java.util.List;
import java.util.Map;

import static ru.ibase.fbjavaex.exampledb.Tables.CUSTOMER;

/**
 * Грид заказчиков
 *
 * @author Simonov Denis
 */
public class JqGridCustomer extends JqGrid {

    /**
     * Добавление условия поиска
     *
     * @param query
     */
    private void makeSearchCondition(SelectQuery<?> query) {
        switch (this.searchOper) {
            case "eq":
                // CUSTOMER.NAME = ?
```

```
                query.addConditions(CUSTOMER.NAME.eq(this.searchString));
                break;
            case "bw":
                // CUSTOMER.NAME STARTING WITH ?
                query.addConditions(CUSTOMER.NAME.startsWith(this.searchString));
                break;
            case "cn":
                // CUSTOMER.NAME CONTAINING ?
                query.addConditions(CUSTOMER.NAME.contains(this.searchString));
                break;
        }
    }


    /**
     * Возвращает общее количество записей
     *
     * @return
     */
    @Override
    public int getCountRecord() {
        // запрос, возвращающий количество записей
        SelectFinalStep<?> select
            = dsl.selectCount()
                 .from(CUSTOMER);

        SelectQuery<?> query = select.getQuery();
        // если мы осуществляем поиск, то добавляем условие поиска
        if (this.searchFlag) {
            makeSearchCondition(query);
        }
        // возвращаем количество
        return (int) query.fetch().getValue(0, 0);
    }

    /**
     * Возвращает записи грида
     *
     * @return
     */
    @Override
    public List<Map<String, Object>> getRecords() {
        // Базовый запрос на выборку
        SelectFinalStep<?> select =
            dsl.select()
               .from(CUSTOMER);

        SelectQuery<?> query = select.getQuery();
        // если мы осуществляем поиск, то добавляем условие поиска
        if (this.searchFlag) {
            makeSearchCondition(query);
        }
        // задаём порядок сортировки
        switch (this.sOrd) {
            case "asc":
                query.addOrderBy(CUSTOMER.NAME.asc());
                break;
```

```
            case "desc":
                query.addOrderBy(CUSTOMER.NAME.desc());
                break;
        }
        // ограничиваем количество записей
        if (this.limit != 0) {
            query.addLimit(this.limit);
        }
        // смещение
        if (this.offset != 0) {
            query.addOffset(this.offset);
        }
        // возвращаем массив карт
        return query.fetchMaps();
    }
}
```

Класс CustomerManager

Добавление, редактирование и удаление заказчика мы будем осуществлять через класс `CustomerManager`, который является своеобразным бизнес-слоем между соответствующим контроллером и базой данных. Все операции в этом слое мы будем осуществлять в транзакции с уровнем изолированности Snapshot.

```
package ru.ibase.fbjavaex.managers;

import org.jooq.DSLContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Isolation;

import static ru.ibase.fbjavaex.exampledb.Tables.CUSTOMER;
import static ru.ibase.fbjavaex.exampledb.Sequences.GEN_CUSTOMER_ID;

/**
 * Менеджер заказчиков
 *
 * @author Simonov Denis
 */
public class CustomerManager {

    @Autowired(required = true)
    private DSLContext dsl;

    /**
     * Добавление заказчика
     *
     * @param name
     * @param address
     * @param zipcode
     * @param phone
```

```
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void create(String name, String address, String zipcode, String phone) {
        if (zipcode != null) {
            if (zipcode.trim().isEmpty()) {
                zipcode = null;
            }
        }

        int customerId = this.dsl.nextval(GEN_CUSTOMER_ID).intValue();

        this.dsl
                .insertInto(CUSTOMER,
                        CUSTOMER.CUSTOMER_ID,
                        CUSTOMER.NAME,
                        CUSTOMER.ADDRESS,
                        CUSTOMER.ZIPCODE,
                        CUSTOMER.PHONE)
                .values(
                        customerId,
                        name,
                        address,
                        zipcode,
                        phone
                )
                .execute();
    }

    /**
     * Редактирование заказчика
     *
     * @param customerId
     * @param name
     * @param address
     * @param zipcode
     * @param phone
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void edit(int customerId, String name, String address,
                     String zipcode, String phone) {

        if (zipcode != null) {
            if (zipcode.trim().isEmpty()) {
                zipcode = null;
            }
        }

        this.dsl.update(CUSTOMER)
                .set(CUSTOMER.NAME, name)
                .set(CUSTOMER.ADDRESS, address)
                .set(CUSTOMER.ZIPCODE, zipcode)
                .set(CUSTOMER.PHONE, phone)
                .where(CUSTOMER.CUSTOMER_ID.eq(customerId))
                .execute();
    }
```

```
    /**
     * Удаление заказчика
     *
     * @param customerId
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void delete(int customerId) {
        this.dsl.deleteFrom(CUSTOMER)
                .where(CUSTOMER.CUSTOMER_ID.eq(customerId))
                .execute();
    }
}
```

Класс контроллера заказчиков

Теперь перейдём к написанию контроллера. Классы контроллеров начинаются с аннотации `@Controller`. Для определения действий контроллера необходимо добавить аннотацию `@RequestMapping` перед методом и указать в ней маршрут, по которому будет вызвано действие контроллера. Маршрут указывается в параметре *`value`*. В параметре *`method`* можно указать метод HTTP запроса (PUT, GET, POST, DELETE). Входной точкой нашего контроллера будет метод `index`, он отвечает за отображение JSP страницы (представления). Эта страница содержит разметку для отображения грида, панель инструментов и навигации.

Данные для отображения загружаются асинхронно компонентом jqGrid (маршрут `/customer/getdata`). С данным маршрутом связан метод `getData`.

Метод getData

Метод `getData` содержит дополнительную аннотацию `@ResponseBody`, которая говорит о том, что наш метод возвращает объект для сериализации в один из форматов. В аннотации `@RequestMapping` задан параметр `produces = MediaType.APPLICATION_JSON`, что обозначает, что возвращаемый объект будет сериализован в формат JSON. Именно в этом методе мы работаем с классом `JqGridCustomer` описанном выше. Аннотация `@RequestParam` позволяет извлечь значение параметра из HTTP запроса. Данный метод класса работает с GET запросами. Параметр *`value`* в аннотации `@RequestParam` задаёт имя параметра HTTP запроса для извлечения. Параметр *`required`* задаёт, является ли параметр HTTP запроса обязательным. Параметр *`defaultValue`* задаёт значение по умолчанию, которое будет подставлено в случае отсутствия HTTP параметра.

Методы действий контроллера Заказчиков

Метод `addCustomer` предназначен для добавления нового заказчика. Он связан с маршрутом `/customer/create`, и в отличие от предыдущего метода работает с POST запросом. Метод возвращает `{success: true}` в случае успешного добавления, и объект с текстом ошибки в случае ошибки. Данный метод работает с классом бизнес слоя `CustomerManager`.

Метод `editCustomer` связан с маршрутом `/customer/edit` и предназначен для редактирования заказчика. Метод `deleteCustomer` связан с маршрутом `/customer/delete` и предназначен для удаления заказчика.

```
package ru.ibase.fbjavaex.controllers;

import java.util.HashMap;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestParam;
import javax.ws.rs.core.MediaType;

import org.springframework.beans.factory.annotation.Autowired;

import ru.ibase.fbjavaex.managers.CustomerManager;

import ru.ibase.fbjavaex.jqgrid.JqGridCustomer;
import ru.ibase.fbjavaex.jqgrid.JqGridData;


/**
 * Контроллер заказчиков
 *
 * @author Simonov Denis
 */
@Controller
public class CustomerController {


    @Autowired(required = true)
    private JqGridCustomer customerGrid;

    @Autowired(required = true)
    private CustomerManager customerManager;


    /**
     * Действие по умолчанию
     * Возвращает имя JSP страницы (представления) для отображения
     *
     * @param map
     * @return имя JSP шаблона
     */
    @RequestMapping(value = "/customer/", method = RequestMethod.GET)
    public String index(ModelMap map) {
        return "customer";
    }

    /**
     * Возвращает данные в формате JSON для jqGrid
     *
     * @param rows количество строк на страницу
     * @param page номер страницы
     * @param sIdx поле для сортировки
     * @param sOrd порядок сортировки
     * @param search должен ли осуществляться поиск
     * @param searchField поле поиска
```

```
     * @param searchString значение поиска
     * @param searchOper операция поиска
     * @return JSON для jqGrid
     */
    @RequestMapping(value = "/customer/getdata",
            method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public JqGridData getData(
            // количество записей на странице
            @RequestParam(value = "rows", required = false,
                    defaultValue = "20") int rows,
            // номер текущей страницы
            @RequestParam(value = "page", required = false,
                    defaultValue = "1") int page,
            // поле для сортировки
            @RequestParam(value = "sidx", required = false,
                    defaultValue = "") String sIdx,
            // направление сортировки
            @RequestParam(value = "sord", required = false,
                    defaultValue = "asc") String sOrd,
            // осуществляется ли поиск
            @RequestParam(value = "_search", required = false,
                    defaultValue = "false") Boolean search,
            // поле поиска
            @RequestParam(value = "searchField", required = false,
                    defaultValue = "") String searchField,
            // значение поиска
            @RequestParam(value = "searchString", required = false,
                    defaultValue = "") String searchString,
            // операция поиска
            @RequestParam(value = "searchOper", required = false,
                    defaultValue = "") String searchOper,
            // фильтр
            @RequestParam(value="filters", required=false,
                    defaultValue="") String filters) {
        customerGrid.setLimit(rows);
        customerGrid.setPageNo(page);
        customerGrid.setOrderBy(sIdx, sOrd);
        if (search) {
            customerGrid.setSearchCondition(searchField, searchString, searchOper);
        }

        return customerGrid.getJqGridData();
    }

    @RequestMapping(value = "/customer/create",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> addCustomer(
            @RequestParam(value = "NAME", required = true,
                    defaultValue = "") String name,
            @RequestParam(value = "ADDRESS", required = false,
                    defaultValue = "") String address,
            @RequestParam(value = "ZIPCODE", required = false,
                    defaultValue = "") String zipcode,
            @RequestParam(value = "PHONE", required = false,
```

```
                        defaultValue = "") String phone) {
        Map<String, Object> map = new HashMap<>();
        try {
            customerManager.create(name, address, zipcode, phone);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    @RequestMapping(value = "/customer/edit",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> editCustomer(
            @RequestParam(value = "CUSTOMER_ID", required = true,
                        defaultValue = "0") int customerId,
            @RequestParam(value = "NAME", required = true,
                        defaultValue = "") String name,
            @RequestParam(value = "ADDRESS", required = false,
                        defaultValue = "") String address,
            @RequestParam(value = "ZIPCODE", required = false,
                        defaultValue = "") String zipcode,
            @RequestParam(value = "PHONE", required = false,
                        defaultValue = "") String phone) {
        Map<String, Object> map = new HashMap<>();
        try {
            customerManager.edit(customerId, name, address, zipcode, phone);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    @RequestMapping(value = "/customer/delete",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> deleteCustomer(
            @RequestParam(value = "CUSTOMER_ID", required = true,
                        defaultValue = "0") int customerId) {
        Map<String, Object> map = new HashMap<>();
        try {
            customerManager.delete(customerId);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }
}
```

Отображение заказчиков

JSP страница для отображения справочника заказчиков не содержит ничего особенного: разметку с основными частями страницы, таблицу для отображения грида и блок для отображения панели навигации. JSP шаблоны не очень продвинутое средство, при желании вы можете заменить их на другие системы шаблонов, которые поддерживают наследование. В файле `../jspf/head.jspf` содержатся общие скрипты и стили для всех страниц сайта, а файл `../jspf/menu.jspf` главное меню сайта. Мы не будем приводить их код, он довольно простой и при желании вы можете посмотреть его в исходных кодах проекта.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="cp" value="${pageContext.request.servletContext.contextPath}"
       scope="request" />

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>An example of a Spring MVC application using Firebird
               and jOOQ</title>

        <!-- Scripts and styles -->
        <%@ include file="../jspf/head.jspf" %>
        <script src="${cp}/resources/js/jqGridCustomer.js"></script>
    </head>
    <body>
        <!-- Navigation menu -->
        <%@ include file="../jspf/menu.jspf" %>

        <div class="container body-content">

            <h2>Customers</h2>

            <table id="jqGridCustomer"></table>
            <div id="jqPagerCustomer"></div>

            <hr/>
            <footer>
                <p>&copy; 2016 - An example of a Spring MVC application
                using Firebird and jOOQ</p>
            </footer>
        </div>

<script type="text/javascript">
    $(document).ready(function () {
        JqGridCustomer({
            baseAddress: '${cp}'
        });
    });
</script>

    </body>
</html>
```

Основная логика на стороне клиента сосредоточена в JavaScript модуле `/resources/js/jqGridCustomer.js`

```
var JqGridCustomer = (function ($) {

  return function (options) {
      var jqGridCustomer = {
          dbGrid: null,
          options: $.extend({
              baseAddress: null,
              showEditorPanel: true
          }, options),
          // return model description
          getColModel: function () {
              return [
                  {
                      label: 'Id',
                      name: 'CUSTOMER_ID', // field name
                      key: true,
                      hidden: true
                  },
                  {
                      label: 'Name',
                      name: 'NAME',
                      width: 240,
                      sortable: true,
                      editable: true,
                      edittype: "text", // field type in the editor
                      search: true,
                      searchoptions: {
                          // allowed search operators
                          sopt: ['eq', 'bw', 'cn']
                      },
                      // size and maximum length for the input field
                      editoptions: {size: 30, maxlength: 60},
                      editrules: {required: true}
                  },
                  {
                      label: 'Address',
                      name: 'ADDRESS',
                      width: 300,
                      sortable: false, // prohibit sorting
                      editable: true,
                      search: false, // prohibit search
                      edittype: "textarea", // Memo field
                      editoptions: {maxlength: 250, cols: 30, rows: 4}
                  },
                  {
                      label: 'Zip Code',
                      name: 'ZIPCODE',
                      width: 30,
                      sortable: false,
                      editable: true,
                      search: false,
```

```
                edittype: "text",
                editoptions: {size: 30, maxlength: 10}
            },
            {
                label: 'Phone',
                name: 'PHONE',
                width: 80,
                sortable: false,
                editable: true,
                search: false,
                edittype: "text",
                editoptions: {size: 30, maxlength: 14}
            }
        ];
    },
    // grid initialization
    initGrid: function () {
        // url to retrieve data
        var url = jqGridCustomer.options.baseAddress
                + '/customer/getdata';
        jqGridCustomer.dbGrid = $("#jqGridCustomer").jqGrid({
            url: url,
            datatype: "json", // data format
            mtype: "GET", // request type
            colModel: jqGridCustomer.getColModel(),
            rowNum: 500, // number of rows displayed
            loadonce: false, // load only once
            sortname: 'NAME', // Sorting by NAME by default
            sortorder: "asc",
            width: window.innerWidth - 80,
            height: 500,
            viewrecords: true, // display the number of records
            guiStyle: "bootstrap",
            iconSet: "fontAwesome",
            caption: "Customers",
            // navigation item
            pager: 'jqPagerCustomer'
        });
    },
    // editing options
    getEditOptions: function () {
      return {
        url: jqGridCustomer.options.baseAddress + '/customer/edit',
        reloadAfterSubmit: true,
        closeOnEscape: true,
        closeAfterEdit: true,
        drag: true,
        width: 400,
        afterSubmit: jqGridCustomer.afterSubmit,
        editData: {
          // In addition to the values from the form, pass the key field
          CUSTOMER_ID: function () {
            // get the current row
            var selectedRow = jqGridCustomer.dbGrid.getGridParam("selrow");
            // get the value of the field CUSTOMER_ID
            var value = jqGridCustomer.dbGrid.getCell(selectedRow,
                        'CUSTOMER_ID');
            return value;
```

```
            }
          }
        };
      },
      // Add options
      getAddOptions: function () {
        return {
          url: jqGridCustomer.options.baseAddress + '/customer/create',
          reloadAfterSubmit: true,
          closeOnEscape: true,
          closeAfterAdd: true,
          drag: true,
          width: 400,
          afterSubmit: jqGridCustomer.afterSubmit
        };
      },
      // Edit options
      getDeleteOptions: function () {
        return {
          url: jqGridCustomer.options.baseAddress + '/customer/delete',
          reloadAfterSubmit: true,
          closeOnEscape: true,
          closeAfterDelete: true,
          drag: true,
          msg: "Delete the selected customer?",
          afterSubmit: jqGridCustomer.afterSubmit,
          delData: {
            // pass the key field
            CUSTOMER_ID: function () {
              var selectedRow = jqGridCustomer.dbGrid.getGridParam("selrow");
              var value = jqGridCustomer.dbGrid.getCell(selectedRow,
                          'CUSTOMER_ID');
              return value;
            }
          }
        };
      },
      // initializing the navigation bar with editing dialogs
      initPagerWithEditors: function () {
          jqGridCustomer.dbGrid.jqGrid('navGrid', '#jqPagerCustomer',
              {
                   // buttons
                   search: true,
                   add: true,
                   edit: true,
                   del: true,
                   view: true,
                   refresh: true,
                   // button captions
                   searchtext: "Search",
                   addtext: "Add",
                   edittext: "Edit",
                   deltext: "Delete",
                   viewtext: "View",
                   viewtitle: "Selected record",
                   refreshtext: "Refresh"
              },
              jqGridCustomer.getEditOptions(),
```

```
                jqGridCustomer.getAddOptions(),
                jqGridCustomer.getDeleteOptions()
            );
        },
        // initialize the navigation bar without editing dialogs
        initPagerWithoutEditors: function () {
            jqGridCustomer.dbGrid.jqGrid('navGrid', '#jqPagerCustomer',
                {
                    // buttons
                    search: true,
                    add: false,
                    edit: false,
                    del: false,
                    view: false,
                    refresh: true,
                    // button captions
                    searchtext: "Search",
                    viewtext: "View",
                    viewtitle: "Selected record",
                    refreshtext: "Refresh"
                }
            );
        },
        // initialize the navigation bar
        initPager: function () {
            if (jqGridCustomer.options.showEditorPanel) {
                jqGridCustomer.initPagerWithEditors();
            } else {
                jqGridCustomer.initPagerWithoutEditors();
            }
        },
        // initialize
        init: function () {
            jqGridCustomer.initGrid();
            jqGridCustomer.initPager();
        },
        // processor of the results of processing forms (operations)
        afterSubmit: function (response, postdata) {
            var responseData = response.responseJSON;
            // check the result for error messages
            if (responseData.hasOwnProperty("error")) {
                if (responseData.error.length) {
                    return [false, responseData.error];
                }
            } else {
                // if an error was not returned, refresh the grid
                $(this).jqGrid(
                        'setGridParam',
                        {
                            datatype: 'json'
                        }
                ).trigger('reloadGrid');
            }
            return [true, "", 0];
        }
    };
    jqGridCustomer.init();
    return jqGridCustomer;
```

```
  };
})(jQuery);
```

Визуальные элементы

Сетка jqGrid создаётся в методе `initGrid` и привязывается к html элементу с идентификатором jqGridCustomer. Описание столбцов (колонок) грида возвращается методом `getColModel`. Каждый столбец в jqGrid имеет достаточно много возможных свойств. В исходном коде присутствуют комментарии, объясняющие свойства столбцов. Подробнее о конфигурировании модели столбцов jqGrid вы можете прочитать в документации проекта jqGrid в разделе ColModel API.

Панель навигации может быть создана с кнопками редактирования или без них (методы `initPagerWithEditors` и `initPagerWithoutEditors` соответственно). Конструктор панели прикрепляет её к элементу с идентификатором jqPagerCustomer. Опции создания панели навигации описаны в разделе Navigator документации jqGrid.

Функции `getEditOptions`, `getAddOptions`, `getDeleteOptions` возвращают опции диалогов редактирования, добавления и удаления соответственно. Свойство `url` указывает, по какому адресу будут отправлены данные после нажатия кнопки OK в диалоге. Свойство `afterSubmit` — событие, происходящее после отправки данных на сервер и получения ответа от него. В методе `afterSubmit` проверяется, не вернул ли наш контроллер ошибку. Если ошибки не было, то производится обновление грида, в противном случае ошибка сообщается пользователю. Обратите внимание на свойство `editData`. Оно позволяет задать значения дополнительных полей, которые не участвуют в диалоге редактирования. Дело в том, что диалоги редактирования не включают в себя значение скрытых полей, а отображать автоматически генерируемые ключи не сильно хочется.

Создание журналов

В отличие от справочников журналы содержат довольно большое количество записей и являются часто пополняемыми. Большинство журналов содержат поле с датой создания документа. Чтобы уменьшить количество выбираемых данных обычно принято вводить такое понятие как рабочий период для того, чтобы уменьшить объём данных передаваемый на клиента. Рабочий период — это диапазон дат, внутри которого требуются рабочие документы. Рабочий период описывается классом `WorkingPeriod`. Этот класс создаётся через бин workingPeriod в классе конфигурации `ru.ibase.fbjavaex.config.JooqConfig`.

```
package ru.ibase.fbjavaex.config;

import java.sql.Timestamp;
import java.time.LocalDateTime;

/**
 * Working period
 *
 * @author Simonov Denis
 */
public class WorkingPeriod {
```

```
    private Timestamp beginDate;
    private Timestamp endDate;

    /**
     * Contructor
     */
    WorkingPeriod() {
        // in real applications is calculated from the current date
        this.beginDate = Timestamp.valueOf("2015-06-01 00:00:00");
        this.endDate = Timestamp.valueOf(LocalDateTime.now().plusDays(1));
    }

    /**
     * Returns the start date of the work period
     *
     * @return
     */
    public Timestamp getBeginDate() {
        return this.beginDate;
    }

    /**
     * Returns the end date of the work period
     *
     * @return
     */
    public Timestamp getEndDate() {
        return this.endDate;
    }

    /**
     * Setting the start date of the work period
     *
     * @param value
     */
    public void setBeginDate(Timestamp value) {
        this.beginDate = value;
    }

    /**
     * Setting the end date of the work period
     *
     * @param value
     */
    public void setEndDate(Timestamp value) {
        this.endDate = value;
    }

    /**
     * Setting the working period
     *
     * @param beginDate
     * @param endDate
     */
    public void setRangeDate(Timestamp beginDate, Timestamp endDate) {
        this.beginDate = beginDate;
```

```
        this.endDate = endDate;
    }
}
```

В нашем приложении будет один журнал «Счёт-фактуры». Счёт-фактура – состоит из заголовка, где описываются общие атрибуты (номер, дата, заказчик …), и позиций счёт-фактуры (наименование товара, количество, стоимостью и т.д.). Шапка счёт-фактуру отображается в основной сетке, а позиции могут быть просмотрены в детализирующей сетке, которая раскрывается по щелчку по значку «+» на нужном документе.

Реализуем класс для просмотра шапок счёт-фактуры через jqGrid, он будет наследоваться от нашего абстрактного класса `ru.ibase.fbjavaex.jqgrid.JqGrid`, описанного выше. В нём имеется возможность поиска наименованию заказчика и дате счёта. Кроме того данный класс поддерживает сортировку по дате в обоих направлениях.

```
package ru.ibase.fbjavaex.jqgrid;

import java.sql.*;
import org.jooq.*;

import java.util.List;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import ru.ibase.fbjavaex.config.WorkingPeriod;

import static ru.ibase.fbjavaex.exampledb.Tables.INVOICE;
import static ru.ibase.fbjavaex.exampledb.Tables.CUSTOMER;

/**
 * Grid handler for the invoice journal
 *
 * @author Simonov Denis
 */
public class JqGridInvoice extends JqGrid {

    @Autowired(required = true)
    private WorkingPeriod workingPeriod;

    /**
     * Adding a search condition
     *
     * @param query
     */
    private void makeSearchCondition(SelectQuery<?> query) {
        // adding a search condition to the query,
        // if it is produced for different fields,
        // different comparison operators are available when searching.
        if (this.searchString.isEmpty()) {
            return;
        }

        if (this.searchField.equals("CUSTOMER_NAME")) {
            switch (this.searchOper) {
```

```
                case "eq": // equal
                  query.addConditions(CUSTOMER.NAME.eq(this.searchString));
                  break;
                case "bw": // starting with
                  query.addConditions(CUSTOMER.NAME.startsWith(this.searchString));
                  break;
                case "cn": // containing
                  query.addConditions(CUSTOMER.NAME.contains(this.searchString));
                  break;
            }
        }
        if (this.searchField.equals("INVOICE_DATE")) {
            Timestamp dateValue = Timestamp.valueOf(this.searchString);

            switch (this.searchOper) {
                case "eq": // =
                  query.addConditions(INVOICE.INVOICE_DATE.eq(dateValue));
                  break;
                case "lt": // <
                  query.addConditions(INVOICE.INVOICE_DATE.lt(dateValue));
                  break;
                case "le": // <=
                  query.addConditions(INVOICE.INVOICE_DATE.le(dateValue));
                  break;
                case "gt": // >
                  query.addConditions(INVOICE.INVOICE_DATE.gt(dateValue));
                  break;
                case "ge": // >=
                  query.addConditions(INVOICE.INVOICE_DATE.ge(dateValue));
                  break;
            }
        }
    }

    /**
     * Returns the total number of records
     *
     * @return
     */
    @Override
    public int getCountRecord() {
        SelectFinalStep<?> select
                = dsl.selectCount()
                     .from(INVOICE)
                     .where(INVOICE.INVOICE_DATE.between(
                               this.workingPeriod.getBeginDate(),
                               this.workingPeriod.getEndDate()));

        SelectQuery<?> query = select.getQuery();

        if (this.searchFlag) {
            makeSearchCondition(query);
        }

        return (int) query.fetch().getValue(0, 0);
    }
```

```
    /**
     * Returns the list of invoices
     *
     * @return
     */
    @Override
    public List<Map<String, Object>> getRecords() {
        SelectFinalStep<?> select = dsl.select(
                INVOICE.INVOICE_ID,
                INVOICE.CUSTOMER_ID,
                CUSTOMER.NAME.as("CUSTOMER_NAME"),
                INVOICE.INVOICE_DATE,
                INVOICE.PAID,
                INVOICE.TOTAL_SALE)
            .from(INVOICE)
            .innerJoin(CUSTOMER).on(CUSTOMER.CUSTOMER_ID.eq(INVOICE.CUSTOMER_ID))
            .where(INVOICE.INVOICE_DATE.between(
                       this.workingPeriod.getBeginDate(),
                       this.workingPeriod.getEndDate()));

        SelectQuery<?> query = select.getQuery();
        // add a search condition
        if (this.searchFlag) {
            makeSearchCondition(query);
        }
        // add sorting
        if (this.sIdx.equals("INVOICE_DATE")) {
            switch (this.sOrd) {
               case "asc":
                 query.addOrderBy(INVOICE.INVOICE_DATE.asc());
                 break;
               case "desc":
                 query.addOrderBy(INVOICE.INVOICE_DATE.desc());
                 break;
            }
        }
        // limit the number of records and add an offset
        if (this.limit != 0) {
            query.addLimit(this.limit);
        }
        if (this.offset != 0) {
            query.addOffset(this.offset);
        }

        return query.fetchMaps();
    }
}
```

Позиции счёт-фактур

Класс для просмотра позиций счёт-фактуры через jqGrid несколько проще. Во-первых, его записи отфильтрованы по коду шапки счёт фактуры, а во-вторых в нём мы не будем реализовывать поиск и пользовательскую сортировку.

```
package ru.ibase.fbjavaex.jqgrid;

import org.jooq.*;

import java.util.List;
import java.util.Map;

import static ru.ibase.fbjavaex.exampledb.Tables.INVOICE_LINE;
import static ru.ibase.fbjavaex.exampledb.Tables.PRODUCT;

/**
 * The grid handler for the invoice items
 *
 * @author Simonov Denis
 */
public class JqGridInvoiceLine extends JqGrid {

    private int invoiceId;


    public int getInvoiceId() {
        return this.invoiceId;
    }

    public void setInvoiceId(int invoiceId) {
        this.invoiceId = invoiceId;
    }

    /**
     * Returns the total number of records
     *
     * @return
     */
    @Override
    public int getCountRecord() {
        SelectFinalStep<?> select
            = dsl.selectCount()
                .from(INVOICE_LINE)
                .where(INVOICE_LINE.INVOICE_ID.eq(this.invoiceId));

        SelectQuery<?> query = select.getQuery();

        return (int) query.fetch().getValue(0, 0);
    }


    /**
     * Returns invoice items
     *
```

```
     * @return
     */
    @Override
    public List<Map<String, Object>> getRecords() {
        SelectFinalStep<?> select = dsl.select(
                INVOICE_LINE.INVOICE_LINE_ID,
                INVOICE_LINE.INVOICE_ID,
                INVOICE_LINE.PRODUCT_ID,
                PRODUCT.NAME.as("PRODUCT_NAME"),
                INVOICE_LINE.QUANTITY,
                INVOICE_LINE.SALE_PRICE,
                INVOICE_LINE.SALE_PRICE.mul(INVOICE_LINE.QUANTITY).as("TOTAL"))
            .from(INVOICE_LINE)
            .innerJoin(PRODUCT).on(PRODUCT.PRODUCT_ID.eq(INVOICE_LINE.PRODUCT_ID))
            .where(INVOICE_LINE.INVOICE_ID.eq(this.invoiceId));

        SelectQuery<?> query = select.getQuery();
        return query.fetchMaps();
    }
}
```

Класс InvoiceManager

Добавлять, редактировать, удалять счёт фактуры (и их позиции), а также оплачивать их, мы будем через класс `ru.ibase.fbjavaex.managers.InvoiceManager`, который является своеобразным бизнес слоем. Все операции в этом слое мы будем осуществлять в транзакции с уровнем изолированности Snapshot. В этом классе все операции с базой данных осуществляются с помощью вызовов хранимых процедур (это не является обязательным, просто показан один из вариантов).

```
package ru.ibase.fbjavaex.managers;


import java.sql.Timestamp;
import org.jooq.DSLContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Isolation;

import static ru.ibase.fbjavaex.exampledb.Sequences.GEN_INVOICE_ID;
import static ru.ibase.fbjavaex.exampledb.Routines.spAddInvoice;
import static ru.ibase.fbjavaex.exampledb.Routines.spEditInvoice;
import static ru.ibase.fbjavaex.exampledb.Routines.spPayForInovice;
import static ru.ibase.fbjavaex.exampledb.Routines.spDeleteInvoice;
import static ru.ibase.fbjavaex.exampledb.Routines.spAddInvoiceLine;
import static ru.ibase.fbjavaex.exampledb.Routines.spEditInvoiceLine;
import static ru.ibase.fbjavaex.exampledb.Routines.spDeleteInvoiceLine;

/**
 * Invoice manager
 *
```

```
 * @author Simonov Denis
 */
public class InvoiceManager {

    @Autowired(required = true)
    private DSLContext dsl;

    /**
     * Add invoice
     *
     * @param customerId
     * @param invoiceDate
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void create(Integer customerId,
                       Timestamp invoiceDate) {
        int invoiceId = this.dsl.nextval(GEN_INVOICE_ID).intValue();

        spAddInvoice(this.dsl.configuration(),
            invoiceId,
            customerId,
            invoiceDate);
    }

    /**
     * Edit invoice
     *
     * @param invoiceId
     * @param customerId
     * @param invoiceDate
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void edit(Integer invoiceId,
                     Integer customerId,
                     Timestamp invoiceDate) {
        spEditInvoice(this.dsl.configuration(),
            invoiceId,
            customerId,
            invoiceDate);
    }

    /**
     * Payment of invoices
     *
     * @param invoiceId
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void pay(Integer invoiceId) {
        spPayForInovice(this.dsl.configuration(),
            invoiceId);
    }

    /**
     * Delete invoice
```

```
     *
     * @param invoiceId
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void delete(Integer invoiceId) {
        spDeleteInvoice(this.dsl.configuration(),
            invoiceId);
    }

    /**
     * Add invoice item
     *
     * @param invoiceId
     * @param productId
     * @param quantity
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void addInvoiceLine(Integer invoiceId,
                               Integer productId,
                               Integer quantity) {
        spAddInvoiceLine(this.dsl.configuration(),
            invoiceId,
            productId,
            quantity);
    }

    /**
     * Edit invoice item
     *
     * @param invoiceLineId
     * @param quantity
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void editInvoiceLine(Integer invoiceLineId,
                                Integer quantity) {
        spEditInvoiceLine(this.dsl.configuration(),
            invoiceLineId,
            quantity);
    }

    /**
     * Delete invoice item
     *
     * @param invoiceLineId
     */
    @Transactional(propagation = Propagation.REQUIRED,
                   isolation = Isolation.REPEATABLE_READ)
    public void deleteInvoiceLine(Integer invoiceLineId) {
        spDeleteInvoiceLine(this.dsl.configuration(),
            invoiceLineId);
    }
}
```

Контроллер счёт-фактур

Теперь перейдём к написанию контроллера. Входной точкой нашего контроллера будет метод `index`, он отвечает за отображение JSP страницы (представления). Эта страница содержит разметку для отображения грида, панель инструментов и навигации.

Данные для отображения шапок счёт фактуры загружаются асинхронно компонентом jqGrid (маршрут `/invoice/getdata`). С данным маршрутом связан метод `getData` (аналогично справочникам). Позиции счёт фактуры возвращаются методом `getDetailData` (маршрут `/invoice/getdetaildata`). В этот метод передаётся код счёт фактуры, на которой был раскрыт детализирующий грид. Методы `addInvoice`, `editInvoice`, `payInvoice`, `deleteInvoice` осуществляют добавление, редактирование, оплату и удаление счёт фактуры. Методы `addInvoiceLine`, `editInvoiceLine`, `deleteInvoiceLine` осуществляют добавление, редактирование и удаление позиции счёт фактуры.

```
package ru.ibase.fbjavaex.controllers;

import java.sql.Timestamp;
import java.util.HashMap;
import java.util.Map;
import java.util.Date;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.beans.PropertyEditorSupport;

import javax.ws.rs.core.MediaType;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.WebDataBinder;
import ru.ibase.fbjavaex.jqgrid.JqGridInvoice;
import ru.ibase.fbjavaex.jqgrid.JqGridInvoiceLine;

import ru.ibase.fbjavaex.managers.InvoiceManager;

import ru.ibase.fbjavaex.jqgrid.JqGridData;


/**
 * Invoice controller
 *
 * @author Simonov Denis
 */
@Controller
public class InvoiceController {

    @Autowired(required = true)
    private JqGridInvoice invoiceGrid;
```

```
    @Autowired(required = true)
    private JqGridInvoiceLine invoiceLineGrid;

    @Autowired(required = true)
    private InvoiceManager invoiceManager;

    /**
     * Describe how a string is converted to a date
     * from the input parameters of the HTTP request
     *
     * @param binder
     */
    @InitBinder
    public void initBinder(WebDataBinder binder)    {
        binder.registerCustomEditor(Timestamp.class,
                new PropertyEditorSupport() {
            @Override
            public void setAsText(String value) {
              try {
                if ((value == null) || (value.isEmpty())) {
                  setValue(null);
                } else {
                  Date parsedDate = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss")
                        .parse(value);
                  setValue(new Timestamp(parsedDate.getTime()));
                }
              } catch (ParseException e) {
                throw new java.lang.IllegalArgumentException(value);
              }
            }
        });
    }

    /**
     * Default action
     * Returns the JSP name of the page (view) to display
     *
     * @param map
     * @return JSP page name
     */
    @RequestMapping(value = "/invoice/", method = RequestMethod.GET)
    public String index(ModelMap map) {

        return "invoice";
    }

    /**
     * Returns a list of invoices in JSON format for jqGrid
     *
     * @param rows number of entries per page
     * @param page current page number
     * @param sIdx sort field
     * @param sOrd sorting order
     * @param search search flag
     * @param searchField search field
     * @param searchString search value
     * @param searchOper comparison operation
```

```
     * @param filters filter
     * @return
     */
    @RequestMapping(value = "/invoice/getdata",
            method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public JqGridData getData(
            @RequestParam(value = "rows", required = false,
                          defaultValue = "20") int rows,
            @RequestParam(value = "page", required = false,
                          defaultValue = "1") int page,
            @RequestParam(value = "sidx", required = false,
                          defaultValue = "") String sIdx,
            @RequestParam(value = "sord", required = false,
                          defaultValue = "asc") String sOrd,
            @RequestParam(value = "_search", required = false,
                          defaultValue = "false") Boolean search,
            @RequestParam(value = "searchField", required = false,
                          defaultValue = "") String searchField,
            @RequestParam(value = "searchString", required = false,
                          defaultValue = "") String searchString,
            @RequestParam(value = "searchOper", required = false,
                          defaultValue = "") String searchOper,
            @RequestParam(value = "filters", required = false,
                          defaultValue = "") String filters) {

        if (search) {
            invoiceGrid.setSearchCondition(searchField, searchString, searchOper);
        }
        invoiceGrid.setLimit(rows);
        invoiceGrid.setPageNo(page);

        invoiceGrid.setOrderBy(sIdx, sOrd);

        return invoiceGrid.getJqGridData();
    }

    /**
     * Add invoice
     *
     * @param customerId customer id
     * @param invoiceDate invoice date
     * @return
     */
    @RequestMapping(value = "/invoice/create",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> addInvoice(
            @RequestParam(value = "CUSTOMER_ID", required = true,
                          defaultValue = "0") Integer customerId,
            @RequestParam(value = "INVOICE_DATE", required = false,
                          defaultValue = "") Timestamp invoiceDate) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.create(customerId, invoiceDate);
            map.put("success", true);
```

```
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Edit invoice
     *
     * @param invoiceId invoice id
     * @param customerId customer id
     * @param invoiceDate invoice date
     * @return
     */
    @RequestMapping(value = "/invoice/edit",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> editInvoice(
            @RequestParam(value = "INVOICE_ID", required = true,
                          defaultValue = "0") Integer invoiceId,
            @RequestParam(value = "CUSTOMER_ID", required = true,
                          defaultValue = "0") Integer customerId,
            @RequestParam(value = "INVOICE_DATE", required = false,
                          defaultValue = "") Timestamp invoiceDate) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.edit(invoiceId, customerId, invoiceDate);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Pays an invoice
     *
     * @param invoiceId invoice id
     * @return
     */
    @RequestMapping(value = "/invoice/pay",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> payInvoice(
            @RequestParam(value = "INVOICE_ID", required = true,
                          defaultValue = "0") Integer invoiceId) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.pay(invoiceId);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }
```

```
    /**
     * Delete invoice
     *
     * @param invoiceId invoice id
     * @return
     */
    @RequestMapping(value = "/invoice/delete",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> deleteInvoice(
            @RequestParam(value = "INVOICE_ID", required = true,
                          defaultValue = "0") Integer invoiceId) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.delete(invoiceId);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Returns invoice item
     *
     * @param invoice_id invoice id
     * @return
     */
    @RequestMapping(value = "/invoice/getdetaildata",
            method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public JqGridData getDetailData(
            @RequestParam(value = "INVOICE_ID", required = true) int invoice_id) {

        invoiceLineGrid.setInvoiceId(invoice_id);

        return invoiceLineGrid.getJqGridData();

    }

    /**
     * Add invoice item
     *
     * @param invoiceId invoice id
     * @param productId product id
     * @param quantity quantity of products
     * @return
     */
    @RequestMapping(value = "/invoice/createdetail",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> addInvoiceLine(
            @RequestParam(value = "INVOICE_ID", required = true,
```

```
                        defaultValue = "0") Integer invoiceId,
            @RequestParam(value = "PRODUCT_ID", required = true,
                        defaultValue = "0") Integer productId,
            @RequestParam(value = "QUANTITY", required = true,
                        defaultValue = "0") Integer quantity) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.addInvoiceLine(invoiceId, productId, quantity);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Edit invoice item
     *
     * @param invoiceLineId invoice item id
     * @param quantity quantity of products
     * @return
     */
    @RequestMapping(value = "/invoice/editdetail",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> editInvoiceLine(
            @RequestParam(value = "INVOICE_LINE_ID", required = true,
                        defaultValue = "0") Integer invoiceLineId,
            @RequestParam(value = "QUANTITY", required = true,
                        defaultValue = "0") Integer quantity) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.editInvoiceLine(invoiceLineId, quantity);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Delete invoice item
     *
     * @param invoiceLineId invoice item id
     * @return
     */
    @RequestMapping(value = "/invoice/deletedetail",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> deleteInvoiceLine(
            @RequestParam(value = "INVOICE_LINE_ID", required = true,
                        defaultValue = "0") Integer invoiceLineId) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.deleteInvoiceLine(invoiceLineId);
```

```
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }
}
```

В целом, контроллер счёт фактур похож на контроллеры справочников за двумя исключениями:

1. Контроллер отображает и работает с данными, как главного, так и детализирующего грида.
2. Счёт фактуры отфильтрованы по полю дата так, чтобы в выборку попадали только те счёт фактуры, которые входят в рабочий период.

Работа с датами в Java

При работе с датами в Java существует много особенностей.

Тип `java.sql.Timestamp` в Java поддерживает точность до наносекунд, в то время как в Firebird максимальная точность типа TIMESTAMP составляет десятитысячную долю секунды. На самом деле это не является большой проблемой.

Типы даты и времени в Java поддерживают работу временными зонами. С другой стороны, в настоящее время Firebird не поддерживает тип TIMESTAMP WITH TIMEZONE. В этом случае Java считает, что даты в базе данных хранятся в часовом поясе сервера (а не в UTC как вы могли бы подумать). Однако при сериализации в JSON время будет преобразовано в UTC. Это надо учитывать при обработке времени в JavaScript.

> **Важно**
>
> Java берёт смещение времени из собственной базы временных зон, а не из операционной системы. Это обстоятельство существенно повышает требования к актуальности версии JDK. Если у вас установлена древняя JDK, то работа с датой и временем может вестись не верно.

По умолчанию дата сериализуется в JSON в числовом представлении (как число наносекунд прошедших с 1 января 1970). Это не всегда удобно. Для того чтобы дата сериализовалась в текстовом представлении в классе `WebAppConfig`, описанном выше, в методе `configureMessageConverters` свойству конфигурации `SerializationFeature.WRITE_DATES_AS_TIMESTAMPS` преобразования даты указать значение `false`.

```
@Configuration
@ComponentScan("ru.ibase.fbjavaex")
@EnableWebMvc
public class WebAppConfig extends WebMvcConfigurerAdapter {

    @Override
    public void configureMessageConverters(
      List<HttpMessageConverter<?>> httpMessageConverters) {
        MappingJackson2HttpMessageConverter jsonConverter =
            new MappingJackson2HttpMessageConverter();
        ObjectMapper objectMapper = new ObjectMapper();
```

```
        objectMapper.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS,
                               false);
        jsonConverter.setObjectMapper(objectMapper);
        httpMessageConverters.add(jsonConverter);
    }
…
}
```

Метод `initBinder` контроллера `InvoiceController` описывает, каким образом текстовое представление даты, присылаемое браузером, преобразуется в значение типа Timestamp.

Отображение счёт-фактур

JSP страница содержит разметку для отображения сетки с шапками счёт-фактур и панель навигации. Позиции счёт фактур отображаются при раскрытии счёт шапки фактуры, как выпадающий грид.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="cp" value="${pageContext.request.servletContext.contextPath}"
       scope="request" />

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>An example of a Spring MVC application using Firebird and jOOQ</title>

        <!-- Scripts and styles -->
        <%@ include file="../jspf/head.jspf" %>
        <script src="${cp}/resources/js/jqGridProduct.js"></script>
        <script src="${cp}/resources/js/jqGridCustomer.js"></script>
        <script src="${cp}/resources/js/jqGridInvoice.js"></script>
    </head>
    <body>
        <!-- Navigation menu -->
        <%@ include file="../jspf/menu.jspf" %>

        <div class="container body-content">

            <h2>Invoices</h2>

            <table id="jqGridInvoice"></table>
            <div id="jqPagerInvoice"></div>

            <hr />
            <footer>
                <p>&copy; 2016 - An example of a Spring MVC application using
                   Firebird and jOOQ</p>
            </footer>
        </div>

        <script type="text/javascript">
```

```
            var invoiceGrid = null;
            $(document).ready(function () {
                invoiceGrid = JqGridInvoice({
                    baseAddress: '${cp}'
                });
            });
        </script>

    </body>
</html>
```

Основная логика на стороне клиента сосредоточена в JavaScript модуле `/resources/js/jqGridInvoice.js`

```
var JqGridInvoice = (function ($, jqGridProductFactory, jqGridCustomerFactory) {

    return function (options) {
        var jqGridInvoice = {
            dbGrid: null,
            detailGrid: null,
            options: $.extend({
                baseAddress: null
            }, options),
            // return invoice model description
            getInvoiceColModel: function () {
                return [
                    {
                        label: 'Id',
                        name: 'INVOICE_ID', // field name
                        key: true,
                        hidden: true
                    },
                    {
                        label: 'Customer Id'
                        name: 'CUSTOMER_ID',
                        hidden: true,
                        editrules: {edithidden: true, required: true},
                        editable: true,
                        edittype: 'custom', // custom type
                        editoptions: {
                            custom_element: function (value, options) {
                                // add hidden input
                                return $("<input>")
                                        .attr('type', 'hidden')
                                        .attr('rowid', options.rowId)
                                        .addClass("FormElement")
                                        .addClass("form-control")
                                        .val(value)
                                        .get(0);
                            }
                        }
                    },
                    {
                        label: 'Date',
```

```
                    name: 'INVOICE_DATE',
                    width: 60,
                    sortable: true,
                    editable: true,
                    search: true,
                    edittype: "text", // input type
                    align: "right",
                    // format as date
                    formatter: jqGridInvoice.dateTimeFormatter,
                    sorttype: 'date', // sort as date
                    formatoptions: {
                        srcformat: 'Y-m-d\TH:i:s', // input format
                        newformat: 'd.m.Y H:i:s'   // output format
                    },
                    editoptions: {
                        // initializing the form element for editing
                        dataInit: function (element) {
                            // creating datepicker
                            $(element).datepicker({
                                id: 'invoiceDate_datePicker',
                                dateFormat: 'dd.mm.yy',
                                minDate: new Date(2000, 0, 1),
                                maxDate: new Date(2030, 0, 1)
                            });
                        }
                    },
                    searchoptions: {
                        // initializing the form element for searching
                        dataInit: function (element) {
                            // создаём datepicker
                            $(element).datepicker({
                                id: 'invoiceDate_datePicker',
                                dateFormat: 'dd.mm.yy',
                                minDate: new Date(2000, 0, 1),
                                maxDate: new Date(2030, 0, 1)
                            });
                        },
                        searchoptions: { // search types
                            sopt: ['eq', 'lt', 'le', 'gt', 'ge']
                        }
                    }
                },
                {
                    label: 'Customer',
                    name: 'CUSTOMER_NAME',
                    width: 250,
                    editable: true,
                    edittype: "text",
                    editoptions: {
                        size: 50,
                        maxlength: 60,
                        readonly: true
                    },
                    editrules: {required: true},
                    search: true,
                    searchoptions: {
                        sopt: ['eq', 'bw', 'cn']
                    }
```

```
                },
                {
                    label: 'Amount',
                    name: 'TOTAL_SALE',
                    width: 60,
                    sortable: false,
                    editable: false,
                    search: false,
                    align: "right",
                    // foramt as currency
                    formatter: 'currency',
                    sorttype: 'number',
                    searchrules: {
                        "required": true,
                        "number": true,
                        "minValue": 0
                    }
                },
                {
                    label: 'Paid',
                    name: 'PAID',
                    width: 30,
                    sortable: false,
                    editable: true,
                    search: true,
                    searchoptions: {
                        sopt: ['eq']
                    },
                    edittype: "checkbox",
                    formatter: "checkbox",
                    stype: "checkbox",
                    align: "center",
                    editoptions: {
                        value: "1",
                        offval: "0"
                    }
                }
            ];
        },
        initGrid: function () {
            // url to retrieve data
            var url = jqGridInvoice.options.baseAddress + '/invoice/getdata';
            jqGridInvoice.dbGrid = $("#jqGridInvoice").jqGrid({
                url: url,
                datatype: "json", // data format
                mtype: "GET", // http request type
                // model description
                colModel: jqGridInvoice.getInvoiceColModel(),
                rowNum: 500, // number of rows displayed
                loadonce: false, // load only once
                // default sort by INVOICE_DATE column
                sortname: 'INVOICE_DATE',
                sortorder: "desc", // sorting order
                width: window.innerWidth - 80,
                height: 500,
                viewrecords: true, // display the number of entries
                guiStyle: "bootstrap",
                iconSet: "fontAwesome",
```

```
            caption: "Invoices",
            // pagination element
            pager: '#jqPagerInvoice',
            subGrid: true, // show subGrid
            // javascript function to display the child grid
            subGridRowExpanded: jqGridInvoice.showChildGrid,
            subGridOptions: {
                // load only once
                reloadOnExpand: false,
                // load the subgrid string only when you click on the "+"
                selectOnExpand: true
            }
        });
    },
    // date format function
    dateTimeFormatter: function(cellvalue, options, rowObject) {
        var date = new Date(cellvalue);
        return date.toLocaleString().replace(",", "");
    },
    // returns a template for the editing dialog
    getTemplate: function () {
      var template = "<div style='margin-left:15px;' id='dlgEditInvoice'>";
      template += "<div>{CUSTOMER_ID} </div>";
      template += "<div> Date: </div><div>{INVOICE_DATE}</div>";
      // customer input field with a button
      template += "<div> Customer <sup>*</sup>:</div>";
      template += "<div>";
      template += "<div style='float: left;'>{CUSTOMER_NAME}</div> ";
      template += "<a style='margin-left: 0.2em;' class='btn' ";
      template += "onclick='invoiceGrid.showCustomerWindow(); ";
      template += "return false;'>";
      template += "<span class='glyphicon glyphicon-folder-open'>";
      template += "</span>Select</a> ";
      template += "<div style='clear: both;'></div>";
      template += "</div>";
      template += "<div> {PAID} Paid </div>";
      template += "<hr style='width: 100%;'/>";
      template += "<div> {sData} {cData}  </div>";
      template += "</div>";
      return template;
    },
    // date conversion in UTC
    convertToUTC: function(datetime) {
      if (datetime) {
          var dateParts = datetime.split('.');
          var date = dateParts[2].substring(0, 4) + '-' +
                     dateParts[1] + '-' + dateParts[0];
          var time = dateParts[2].substring(5);
          if (!time) {
              time = '00:00:00';
          }
          var dt = Date.parse(date + 'T' + time);
          var s = dt.getUTCFullYear() + '-' +
                  dt.getUTCMonth() + '-' +
                  dt.getUTCDay() + 'T' +
                  dt.getUTCHour() + ':' +
                  dt.getUTCMinute() + ':' +
                  dt.getUTCSecond() + '  GMT';
```

```
            return s;
        } else
            return null;
      },
      // returns the options for editing invoices
      getEditInvoiceOptions: function () {
        return {
          url: jqGridInvoice.options.baseAddress + '/invoice/edit',
          reloadAfterSubmit: true,
          closeOnEscape: true,
          closeAfterEdit: true,
          drag: true,
          modal: true,
          top: $(".container.body-content").position().top + 150,
          left: $(".container.body-content").position().left + 150,
          template: jqGridInvoice.getTemplate(),
          afterSubmit: jqGridInvoice.afterSubmit,
          editData: {
            INVOICE_ID: function () {
              var selectedRow = jqGridInvoice.dbGrid.getGridParam("selrow");
              var value = jqGridInvoice.dbGrid
                          .getCell(selectedRow, 'INVOICE_ID');
              return value;
            },
            CUSTOMER_ID: function () {
              return $('#dlgEditInvoice input[name=CUSTOMER_ID]').val();
            },
            INVOICE_DATE: function () {
              var datetime = $('#dlgEditInvoice input[name=INVOICE_DATE]')
                             .val();
              return jqGridInvoice.convertToUTC(datetime);
            }
          }
        };
      },
      // returns options for adding invoices
      getAddInvoiceOptions: function () {
        return {
          url: jqGridInvoice.options.baseAddress + '/invoice/create',
          reloadAfterSubmit: true,
          closeOnEscape: true,
          closeAfterAdd: true,
          drag: true,
          modal: true,
          top: $(".container.body-content").position().top + 150,
          left: $(".container.body-content").position().left + 150,
          template: jqGridInvoice.getTemplate(),
          afterSubmit: jqGridInvoice.afterSubmit,
          editData: {
            CUSTOMER_ID: function () {
              return $('#dlgEditInvoice input[name=CUSTOMER_ID]').val();
            },
            INVOICE_DATE: function () {
              var datetime = $('#dlgEditInvoice input[name=INVOICE_DATE]')
                             .val();
              return jqGridInvoice.convertToUTC(datetime);
            }
          }
```

```
    };
  },
  // returns the options for deleting invoices
  getDeleteInvoiceOptions: function () {
    return {
      url: jqGridInvoice.options.baseAddress + '/invoice/delete',
      reloadAfterSubmit: true,
      closeOnEscape: true,
      closeAfterDelete: true,
      drag: true,
      msg: "Delete the selected invoice?",
      afterSubmit: jqGridInvoice.afterSubmit,
      delData: {
        INVOICE_ID: function () {
          var selectedRow = jqGridInvoice.dbGrid.getGridParam("selrow");
          var value = jqGridInvoice.dbGrid
                      .getCell(selectedRow, 'INVOICE_ID');
          return value;
        }
      }
    };
  },
  initPager: function () {
      // display the navigation bar
      jqGridInvoice.dbGrid.jqGrid('navGrid', '#jqPagerInvoice',
          {
               search: true,
               add: true,
               edit: true,
               del: true,
               view: false,
               refresh: true,

               searchtext: "Search",
               addtext: "Add",
               edittext: "Edit",
               deltext: "Delete",
               viewtext: "View",
               viewtitle: "Selected record",
               refreshtext: "Refresh"
          },
          jqGridInvoice.getEditInvoiceOptions(),
          jqGridInvoice.getAddInvoiceOptions(),
          jqGridInvoice.getDeleteInvoiceOptions()
      );
      // Add a button to pay the invoice
      var urlPay = jqGridInvoice.options.baseAddress + '/invoice/pay';
      jqGridInvoice.dbGrid.navButtonAdd('#jqPagerInvoice',
          {
             buttonicon: "glyphicon-usd",
             title: "Pay",
             caption: "Pay",
             position: "last",
             onClickButton: function () {
               // get the id of the current record
               var id = jqGridInvoice.dbGrid.getGridParam("selrow");
               if (id) {
                 $.ajax({
```

```
                        url: urlPay,
                        type: 'POST',
                        data: {INVOICE_ID: id},
                        success: function (data) {
                            // Check if an error has occurred
                            if (data.hasOwnProperty("error")) {
                                jqGridInvoice.alertDialog('Ошибка',
                                                          data.error);
                            } else {
                                // refresh grid
                                $("#jqGridInvoice").jqGrid(
                                    'setGridParam',
                                    {
                                        datatype: 'json'
                                    }
                                ).trigger('reloadGrid');
                            }
                        }
                    });
                }
            }
        }
    );
},
init: function () {
    jqGridInvoice.initGrid();
    jqGridInvoice.initPager();
},
afterSubmit: function (response, postdata) {
    var responseData = response.responseJSON;
    // Check if an error has occurred
    if (responseData.hasOwnProperty("error")) {
        if (responseData.error.length) {
            return [false, responseData.error];
        }
    } else {
        // refresh grid
        $(this).jqGrid(
                'setGridParam',
                {
                    datatype: 'json'
                }
        ).trigger('reloadGrid');
    }
    return [true, "", 0];
},
getInvoiceLineColModel: function (parentRowKey) {
  return [
      {
        label: 'Invoice Line ID',
        name: 'INVOICE_LINE_ID',
        key: true,
        hidden: true
      },
      {
        label: 'Invoice ID',
        name: 'INVOICE_ID',
        hidden: true,
```

```
        editrules: {edithidden: true, required: true},
        editable: true,
        edittype: 'custom',
        editoptions: {
          custom_element: function (value, options) {
              // create hidden input
              return $("<input>")
                     .attr('type', 'hidden')
                     .attr('rowid', options.rowId)
                     .addClass("FormElement")
                     .addClass("form-control")
                     .val(parentRowKey)
                     .get(0);
          }
        }
      },
      {
        label: 'Product ID',
        name: 'PRODUCT_ID',
        hidden: true,
        editrules: {edithidden: true, required: true},
        editable: true,
        edittype: 'custom',
        editoptions: {
          custom_element: function (value, options) {
              // create hidden input
              return $("<input>")
                     .attr('type', 'hidden')
                     .attr('rowid', options.rowId)
                     .addClass("FormElement")
                     .addClass("form-control")
                     .val(value)
                     .get(0);
          }
        }
      },
      {
        label: 'Product',
        name: 'PRODUCT_NAME',
        width: 300,
        editable: true,
        edittype: "text",
        editoptions: {
            size: 50,
            maxlength: 60,
            readonly: true
        },
        editrules: {required: true}
      },
      {
        label: 'Price',
        name: 'SALE_PRICE',
        formatter: 'currency',
        editable: true,
        editoptions: {
            readonly: true
        },
        align: "right",
```

```
            width: 100
          },
          {
            label: 'Quantity',
            name: 'QUANTITY',
            align: "right",
            width: 100,
            editable: true,
            editrules: {required: true, number: true, minValue: 1},
            editoptions: {
              dataEvents: [{
                type: 'change',
                fn: function (e) {
                  var quantity = $(this).val() - 0;
                  var price =
                    $('#dlgEditInvoiceLine input[name=SALE_PRICE]').val()-0;
                  var total = quantity * price;
                  $('#dlgEditInvoiceLine input[name=TOTAL]').val(total);
                }
              }],
              defaultValue: 1
            }
          },
          {
            label: 'Total',
            name: 'TOTAL',
            formatter: 'currency',
            align: "right",
            width: 100,
            editable: true,
            editoptions: {
                readonly: true
            }
          }
        ];
      },
      // returns the options for editing the invoice item
      getEditInvoiceLineOptions: function () {
        return {
          url: jqGridInvoice.options.baseAddress + '/invoice/editdetail',
          reloadAfterSubmit: true,
          closeOnEscape: true,
          closeAfterEdit: true,
          drag: true,
          modal: true,
          top: $(".container.body-content").position().top + 150,
          left: $(".container.body-content").position().left + 150,
          template: jqGridInvoice.getTemplateDetail(),
          afterSubmit: jqGridInvoice.afterSubmit,
          editData: {
            INVOICE_LINE_ID: function () {
              var selectedRow = jqGridInvoice.detailGrid
                                .getGridParam("selrow");
              var value = jqGridInvoice.detailGrid
                          .getCell(selectedRow, 'INVOICE_LINE_ID');
              return value;
            },
            QUANTITY: function () {
```

```
            return $('#dlgEditInvoiceLine input[name=QUANTITY]').val();
          }
        }
      };
    },
    // returns options for adding an invoice item
    getAddInvoiceLineOptions: function () {
      return {
        url: jqGridInvoice.options.baseAddress + '/invoice/createdetail',
        reloadAfterSubmit: true,
        closeOnEscape: true,
        closeAfterAdd: true,
        drag: true,
        modal: true,
        top: $(".container.body-content").position().top + 150,
        left: $(".container.body-content").position().left + 150,
        template: jqGridInvoice.getTemplateDetail(),
        afterSubmit: jqGridInvoice.afterSubmit,
        editData: {
          INVOICE_ID: function () {
            var selectedRow = jqGridInvoice.dbGrid.getGridParam("selrow");
            var value = jqGridInvoice.dbGrid
                        .getCell(selectedRow, 'INVOICE_ID');
            return value;
          },
          PRODUCT_ID: function () {
            return $('#dlgEditInvoiceLine input[name=PRODUCT_ID]').val();
          },
          QUANTITY: function () {
            return $('#dlgEditInvoiceLine input[name=QUANTITY]').val();
          }
        }
      };
    },
    // returns the option to delete the invoice item
    getDeleteInvoiceLineOptions: function () {
      return {
        url: jqGridInvoice.options.baseAddress + '/invoice/deletedetail',
        reloadAfterSubmit: true,
        closeOnEscape: true,
        closeAfterDelete: true,
        drag: true,
        msg: "Delete the selected item?",
        afterSubmit: jqGridInvoice.afterSubmit,
        delData: {
          INVOICE_LINE_ID: function () {
            var selectedRow = jqGridInvoice.detailGrid
                              .getGridParam("selrow");
            var value = jqGridInvoice.detailGrid
                        .getCell(selectedRow, 'INVOICE_LINE_ID');
            return value;
          }
        }
      };
    },
    // Event handler for the parent grid expansion event
    // takes two parameters: the parent record identifier
    // and the primary record key
```

```
        showChildGrid: function (parentRowID, parentRowKey) {
            var childGridID = parentRowID + "_table";
            var childGridPagerID = parentRowID + "_pager";
            // send the primary key of the parent record
            // to filter the entries of the invoice items
            var childGridURL = jqGridInvoice.options.baseAddress
                             + '/invoice/getdetaildata';
            childGridURL = childGridURL + "?INVOICE_ID="
                         + encodeURIComponent(parentRowKey);
            // add HTML elements to display the table and page navigation
            // as children for the selected row in the master grid
            $('<table>')
                    .attr('id', childGridID)
                    .appendTo($('#' + parentRowID));
            $('<div>')
                    .attr('id', childGridPagerID)
                    .addClass('scroll')
                    .appendTo($('#' + parentRowID));
            // create and initialize the child grid
            jqGridInvoice.detailGrid = $("#" + childGridID).jqGrid({
                url: childGridURL,
                mtype: "GET",
                datatype: "json",
                page: 1,
                colModel: jqGridInvoice.getInvoiceLineColModel(parentRowKey),
                loadonce: false,
                width: '100%',
                height: '100%',
                guiStyle: "bootstrap",
                iconSet: "fontAwesome",
                pager: "#" + childGridPagerID
            });
            // displaying the toolbar
            $("#" + childGridID).jqGrid(
                'navGrid', '#' + childGridPagerID,
                {
                    search: false,
                    add: true,
                    edit: true,
                    del: true,
                    refresh: true
                },
                jqGridInvoice.getEditInvoiceLineOptions(),
                jqGridInvoice.getAddInvoiceLineOptions(),
                jqGridInvoice.getDeleteInvoiceLineOptions()
            );
        },
        // returns a template for the invoice item editor
        getTemplateDetail: function () {
          var template = "<div style='margin-left:15px;' ";
          template += "id='dlgEditInvoiceLine'>";
          template += "<div>{INVOICE_ID} </div>";
          template += "<div>{PRODUCT_ID} </div>";
          // input field with a button
          template += "<div> Product <sup>*</sup>:</div>";
          template += "<div>";
          template += "<div style='float: left;'>{PRODUCT_NAME}</div> ";
          template += "<a style='margin-left: 0.2em;' class='btn' ";
```

```
        template += "onclick='invoiceGrid.showProductWindow(); ";
        template += "return false;'>";
        template += "<span class='glyphicon glyphicon-folder-open'>";
        template += "</span> Select</a> ";
        template += "<div style='clear: both;'></div>";
        template += "</div>";
        template += "<div> Quantity: </div><div>{QUANTITY} </div>";
        template += "<div> Price: </div><div>{SALE_PRICE} </div>";
        template += "<div> Total: </div><div>{TOTAL} </div>";
        template += "<hr style='width: 100%;'/>";
        template += "<div> {sData} {cData}  </div>";
        template += "</div>";
        return template;
    },
    // Display selection window from the goods directory.
    showProductWindow: function () {
        var dlg = $('<div>')
                  .attr('id', 'dlgChooseProduct')
                  .attr('aria-hidden', 'true')
                  .attr('role', 'dialog')
                  .attr('data-backdrop', 'static')
                  .css("z-index", '2000')
                  .addClass('modal')
                  .appendTo($('body'));

        var dlgContent = $("<div>")
                  .addClass("modal-content")
                  .css('width', '760px')
                  .appendTo($('<div>')
                          .addClass('modal-dialog')
                          .appendTo(dlg));

        var dlgHeader = $('<div>').addClass("modal-header")
                                  .appendTo(dlgContent);
        $("<button>")
                  .addClass("close")
                  .attr('type', 'button')
                  .attr('aria-hidden', 'true')
                  .attr('data-dismiss', 'modal')
                  .html("&times;")
                  .appendTo(dlgHeader);
        $("<h5>").addClass("modal-title")
                   .html("Select product")
                   .appendTo(dlgHeader);
        var dlgBody = $('<div>')
                  .addClass("modal-body")
                  .appendTo(dlgContent);
        var dlgFooter = $('<div>').addClass("modal-footer")
                                  .appendTo(dlgContent);
        $("<button>")
          .attr('type', 'button')
          .addClass('btn')
          .html('OK')
          .on('click', function () {
              var rowId = $("#jqGridProduct")
                          .jqGrid("getGridParam", "selrow");
              var row = $("#jqGridProduct")
                        .jqGrid("getRowData", rowId);
```

```
                $('#dlgEditInvoiceLine input[name=PRODUCT_ID]')
                    .val(row["PRODUCT_ID"]);
                $('#dlgEditInvoiceLine input[name=PRODUCT_NAME]')
                    .val(row["NAME"]);
                $('#dlgEditInvoiceLine input[name=SALE_PRICE]')
                    .val(row["PRICE"]);
                var price = $('#dlgEditInvoiceLine input[name=SALE_PRICE]')
                    .val()-0;
                var quantity = $('#dlgEditInvoiceLine input[name=QUANTITY]')
                    .val()-0;
                var total = Math.round(price * quantity * 100) / 100;
                $('#dlgEditInvoiceLine input[name=TOTAL]').val(total);
                dlg.modal('hide');
            })
            .appendTo(dlgFooter);

        $("<button>")
              .attr('type', 'button')
              .addClass('btn')
              .html('Cancel')
              .on('click', function () {
                  dlg.modal('hide');
              })
              .appendTo(dlgFooter);

        $('<table>')
              .attr('id', 'jqGridProduct')
              .appendTo(dlgBody);
        $('<div>')
              .attr('id', 'jqPagerProduct')
              .appendTo(dlgBody);

        dlg.on('hidden.bs.modal', function () {
              dlg.remove();
        });
        dlg.modal();

        jqGridProductFactory({
            baseAddress: jqGridInvoice.options.baseAddress
        });
    },
    // Display the selection window from the customer's directory.
    showCustomerWindow: function () {
        // the main block of the dialog
        var dlg = $('<div>')
                .attr('id', 'dlgChooseCustomer')
                .attr('aria-hidden', 'true')
                .attr('role', 'dialog')
                .attr('data-backdrop', 'static')
                .css("z-index", '2000')
                .addClass('modal')
                .appendTo($('body'));
        // block with the contents of the dialog
        var dlgContent = $("<div>")
                .addClass("modal-content")
                .css('width', '730px')
                .appendTo($('<div>')
                        .addClass('modal-dialog')
```

```
                    .appendTo(dlg));
// block with dialog header
var dlgHeader = $('<div>').addClass("modal-header")
                          .appendTo(dlgContent);
// button "X" for closing
$("<button>")
        .addClass("close")
        .attr('type', 'button')
        .attr('aria-hidden', 'true')
        .attr('data-dismiss', 'modal')
        .html("&times;")
        .appendTo(dlgHeader);
// title of dialog
$("<h5>").addClass("modal-title")
         .html("Select customer")
         .appendTo(dlgHeader);
// body of dialog
var dlgBody = $('<div>')
        .addClass("modal-body")
        .appendTo(dlgContent);
// footer of dialog
var dlgFooter = $('<div>').addClass("modal-footer")
                          .appendTo(dlgContent);
// "OK" button
$("<button>")
    .attr('type', 'button')
    .addClass('btn')
    .html('OK')
    .on('click', function () {
        var rowId = $("#jqGridCustomer")
                    .jqGrid("getGridParam", "selrow");
        var row = $("#jqGridCustomer")
                  .jqGrid("getRowData", rowId);
        // Keep the identifier and the name of the customer
        // in the input elements of the parent form.
        $('#dlgEditInvoice input[name=CUSTOMER_ID]')
            .val(rowId);
        $('#dlgEditInvoice input[name=CUSTOMER_NAME]')
            .val(row["NAME"]);
        dlg.modal('hide');
    })
    .appendTo(dlgFooter);
// "Cancel" button
$("<button>")
        .attr('type', 'button')
        .addClass('btn')
        .html('Cancel')
        .on('click', function () {
            dlg.modal('hide');
        })
        .appendTo(dlgFooter);
// add a table to display the customers in the body of the dialog
$('<table>')
        .attr('id', 'jqGridCustomer')
        .appendTo(dlgBody);
// add the navigation bar
$('<div>')
        .attr('id', 'jqPagerCustomer')
```

```
                    .appendTo(dlgBody);
                dlg.on('hidden.bs.modal', function () {
                    dlg.remove();
                });
                // display dialog
                dlg.modal();
                jqGridCustomerFactory({
                    baseAddress: jqGridInvoice.options.baseAddress
                });
            },
            // A window for displaying the error.
            alertDialog: function (title, error) {
                var alertDlg = $('<div>')
                        .attr('aria-hidden', 'true')
                        .attr('role', 'dialog')
                        .attr('data-backdrop', 'static')
                        .addClass('modal')
                        .appendTo($('body'));
                var dlgContent = $("<div>")
                        .addClass("modal-content")
                        .appendTo($('<div>')
                                .addClass('modal-dialog')
                                .appendTo(alertDlg));
                var dlgHeader = $('<div>').addClass("modal-header")
                                          .appendTo(dlgContent);
                $("<button>")
                        .addClass("close")
                        .attr('type', 'button')
                        .attr('aria-hidden', 'true')
                        .attr('data-dismiss', 'modal')
                        .html("&times;")
                        .appendTo(dlgHeader);
                $("<h5>").addClass("modal-title")
                         .html(title)
                         .appendTo(dlgHeader);
                $('<div>')
                        .addClass("modal-body")
                        .appendTo(dlgContent)
                        .append(error);
                alertDlg.on('hidden.bs.modal', function () {
                    alertDlg.remove();
                });
                alertDlg.modal();
            }
        };
        jqGridInvoice.init();
        return jqGridInvoice;
    };
})(jQuery, JqGridProduct, JqGridCustomer);
```

Отображение и редактирование позиций счёт-фактур

В журнале счёт фактур, основная сетка используется для отображения шапок, а раскрывающаяся по клику для отображения позиций. Для отображения дочернего грида

свойству `subGrid` присвоено значение `true`. Дочерняя сетка отображается, используя событие `subGridRowExpanded`, которое связано с методом `showChildGrid`.

Позиции фильтруются по первичному ключу счёт фактуры. Помимо основных кнопок панель навигации для шапки счёт фактуры добавлена пользовательская кнопка для оплаты счёт фактуры с помощью функции `jqGridInvoice.dbGrid.navButtonAdd` (см. метод `initPager`).

Диалоги

В отличие от справочников диалоги редактирования для журналов намного сложнее. Зачастую они используют выбор из других справочников. Поэтому такие диалоги редактирования не получится построить автоматически с помощью jqGrid, однако в этой библиотеки существует возможность построение диалогов по шаблону, которой мы и воспользуемся.

Шаблон диалога возвращает функцией `getTemplate`. Открытие справочника заказчиков для выбора заказчика осуществляется функцией `invoiceGrid.showCustomerWindow()`. Она использует функции уже описанного ранее модуля JqGridCustomer. После выбора заказчика из модального окна его код подставляется в поле CUSTOMER_ID. В свойстве `editData` опций редактирования и добавления описаны поля, которые надо будет передать на сервер, используя предварительную обработку или из невидимых полей.

Обработка дат

Теперь вернёмся к обработке дат. Как я уже говорил, контроллер `InvoiceController` возвращает дату в UTC, нам же необходимо отобразить её в текущей часовой зоне. Для этого зададим функцию форматирования даты `jqGridInvoice.dateTimeFormatter` через свойство `formatter`, соответствующего поля INVOICE_DATE.

При отправке данных на сервер нам необходимо сделать обратную операцию – перевести время из текущей временной зоны в UTC. За это отвечает функция `convertToUTC`.

Для редактора позиций счёт фактуры, так же используется пользовательский шаблон, который возвращается функцией `getTemplateDetail`. Открытие окна для выбора из справочника товаров осуществляется функцией `invoiceGrid.showProductWindow()`. Эта функция использует функции модуля JqGridProduct.

Код модуля JqGridInvoice подробно прокомментирован так, чтобы вы могли понять логику его работы. Дополнительные пояснения вы можете найти в нём.

Результат

Напоследок приведу несколько скриншотов получившегося веб приложения.

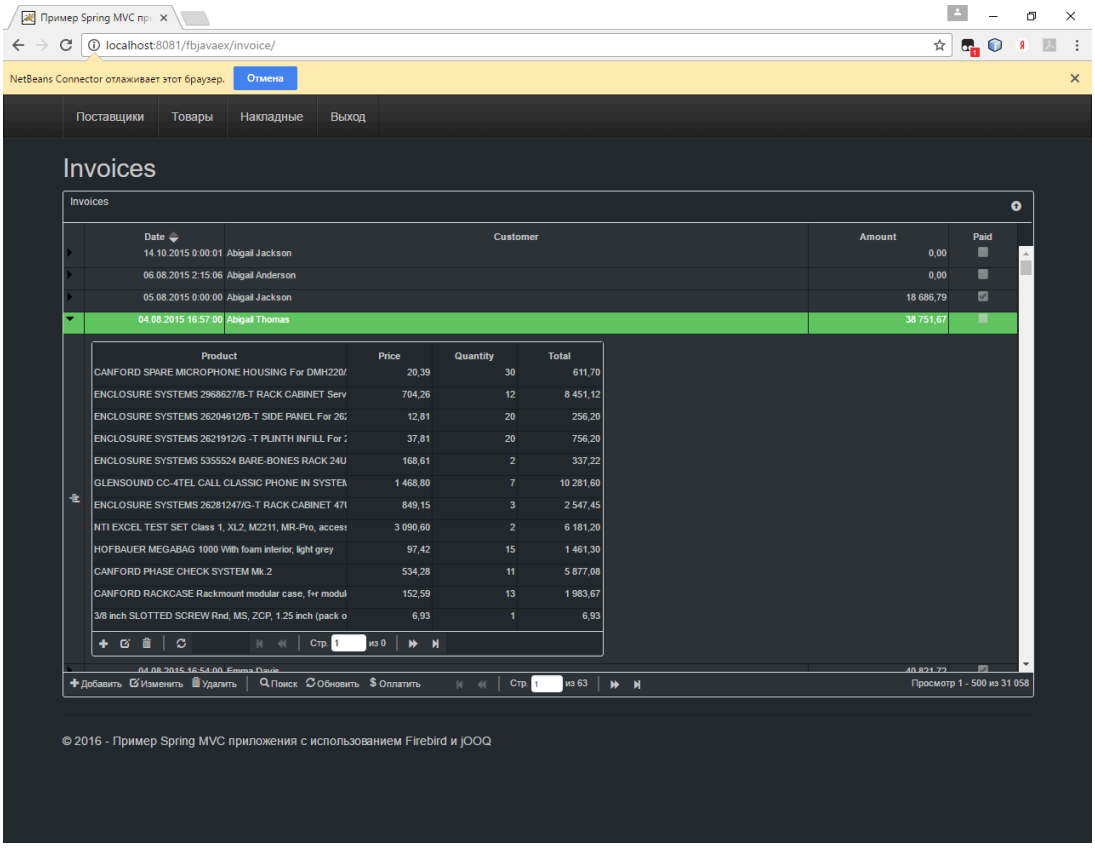

Рис. 6.3. Журнал счёт-фактур

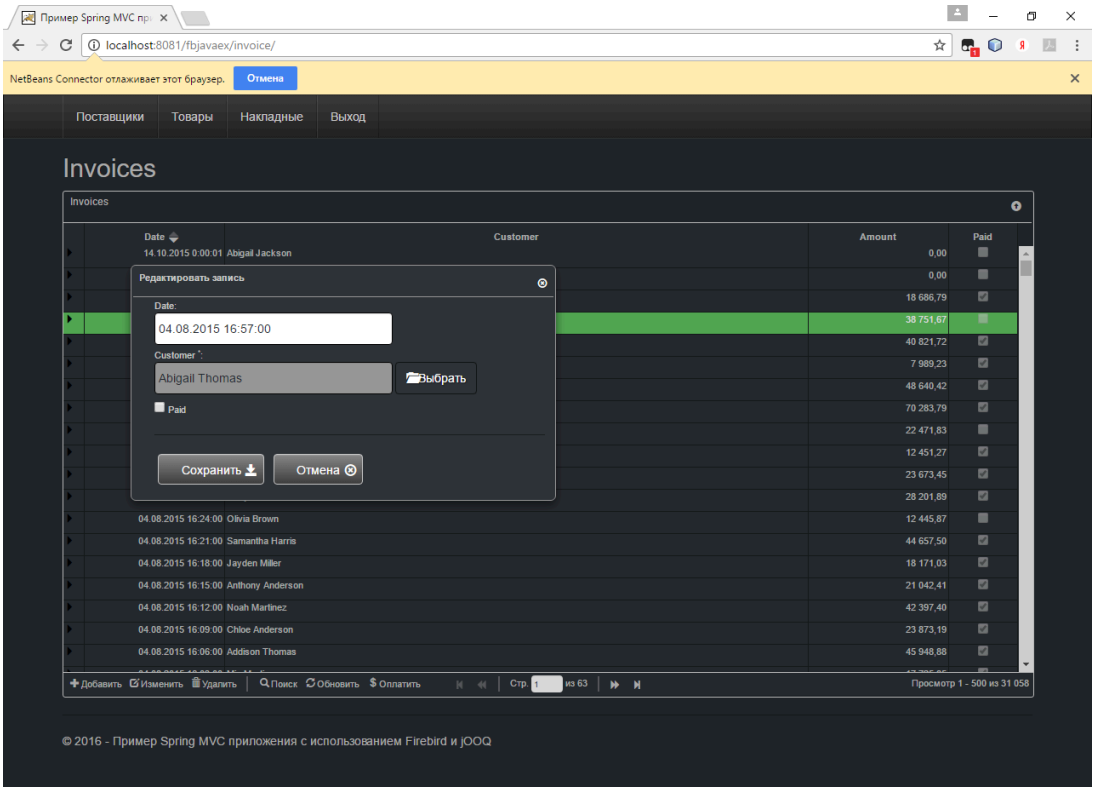

Рис. 6.4. Редактирование счёт-фактуры

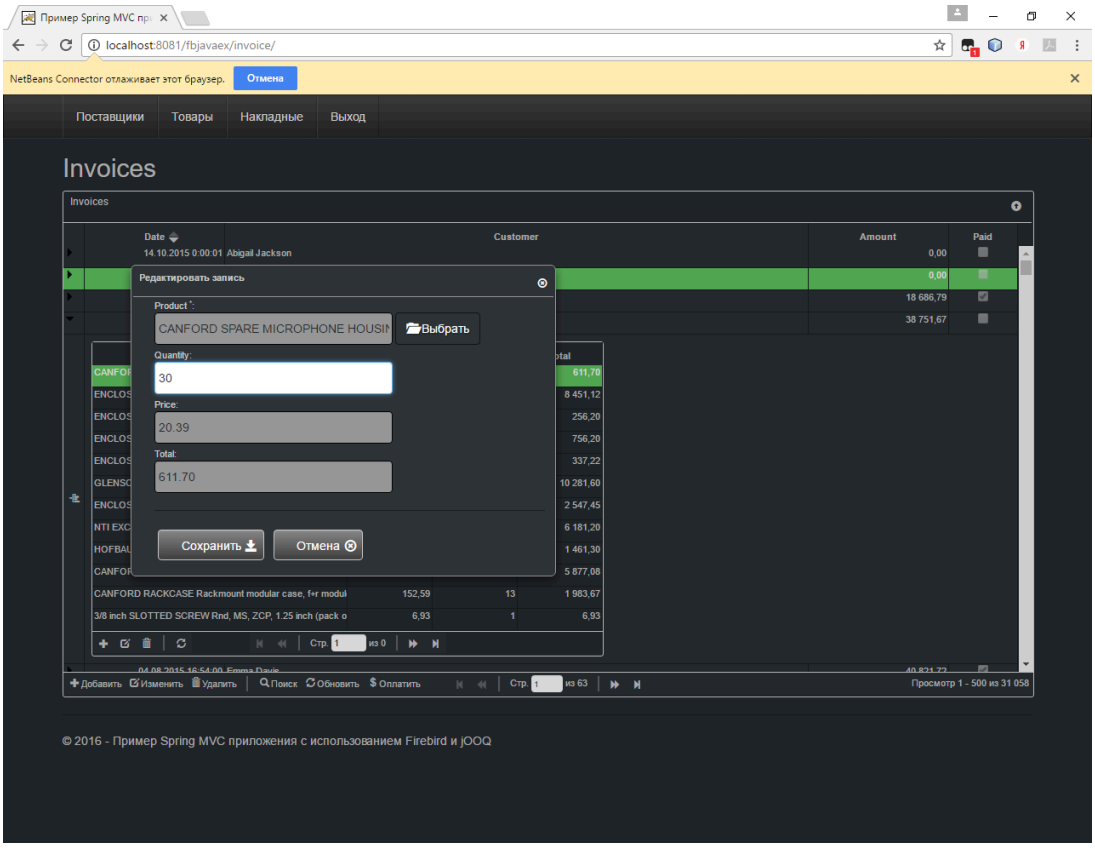

Рис. 6.5. Редактирование позиции счёт-фактуры

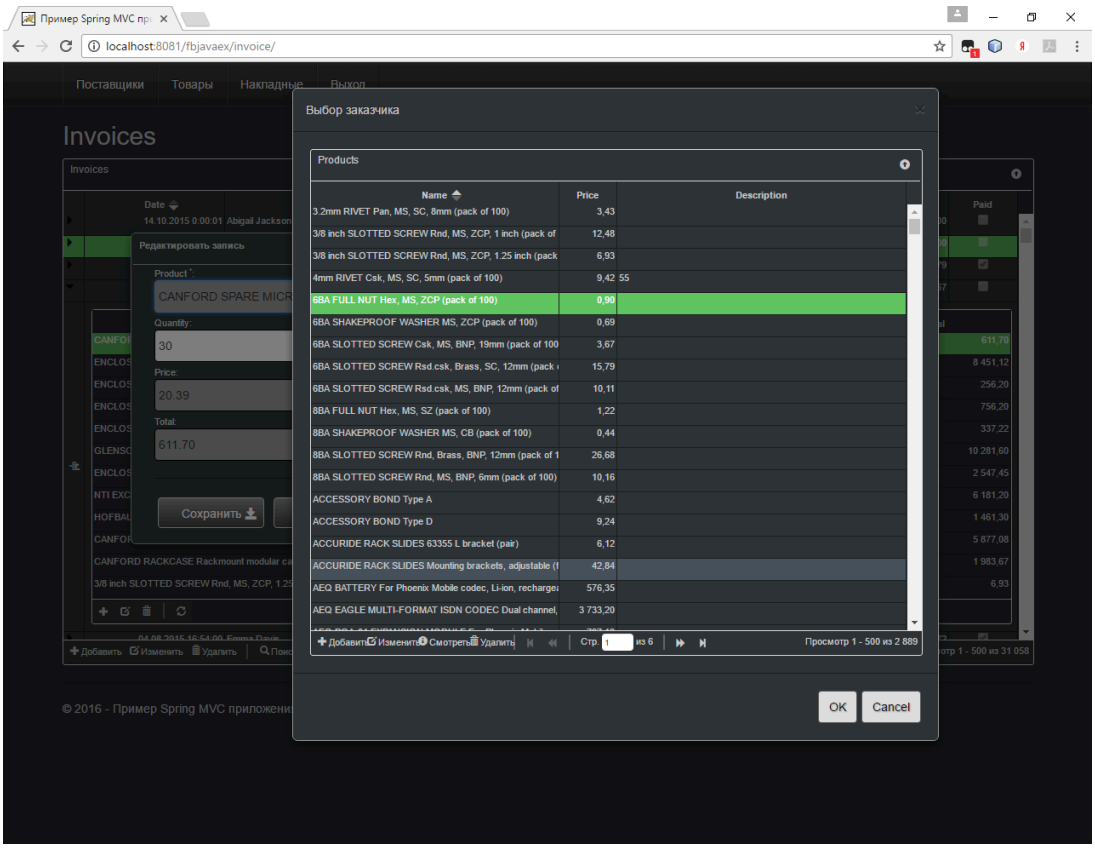

Рис. 6.6. Выбор из справочника в диалоге счёт-фактуры

Исходный код

На этом мой пример закончен. Исходные коды вы можете скачать по ссылке https://github.com/sim1984/fbjavaex.

Алфавитный указатель