Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной информационной безопасности
электронно-вычислительных систем (КИБЭВС)

Н.А. Новгородова

Безопасность систем баз данных

**Учебно-методические указания к выполнению лабораторных,
практических и самостоятельных работ по дисциплинам
«Безопасность систем баз данных»,
«Базы данных и экспертные системы»,
«Базы данных»**

Томск 2021

Оглавление

ПРАКТИЧЕСКАЯ РАБОТА №1 Знакомство с концептуальным проектированием предметной области. 3

ПРАКТИЧЕСКАЯ РАБОТА №2 Знакомство с реляционным проектированием данных. 8

ЛАБОРАТОРНАЯ РАБОТА № 1 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ MICROSOFT SQL SERVER 9

ЛАБОРАТОРНАЯ РАБОТА № 2 СОЗДАНИЕ ПРОСТОГО ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ В MICROSOFT VISUAL STUDIO 95

ЛАБОРАТОРНАЯ РАБОТА № 3 СОЗДАНИЕ ПРИЛОЖЕНИЯ В MICROSOFT VISUAL STUDIO ДЛЯ РАБОТЫ СО СВЯЗАННЫМИ ТАБЛИЦАМИ БАЗЫ ДАННЫХ 155

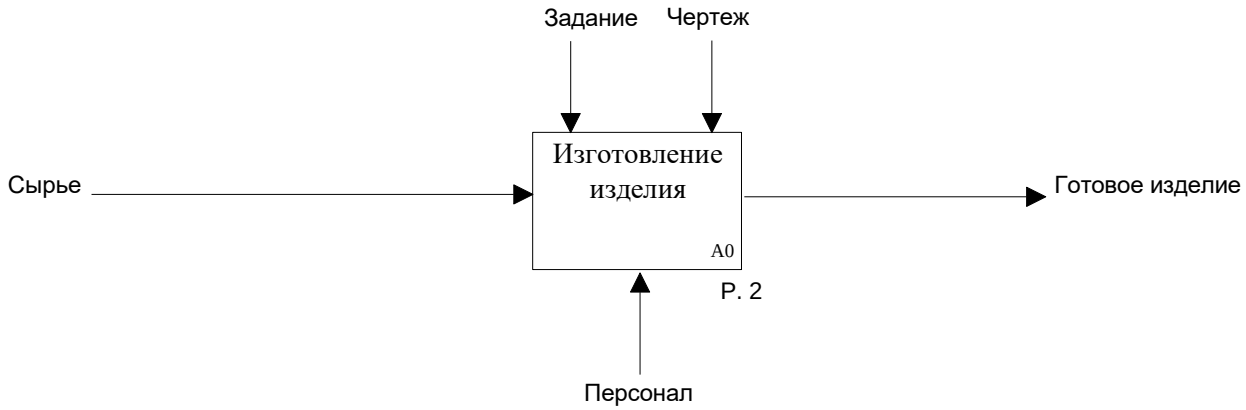
ЛАБОРАТОРНАЯ РАБОТА № 4 SQL ОПЕРАТОРЫ ДЛЯ БД СУБД MICROSOFT SQL SERVER В ПРИЛОЖЕНИИ MICROSOFT VISUAL STUDIO 208

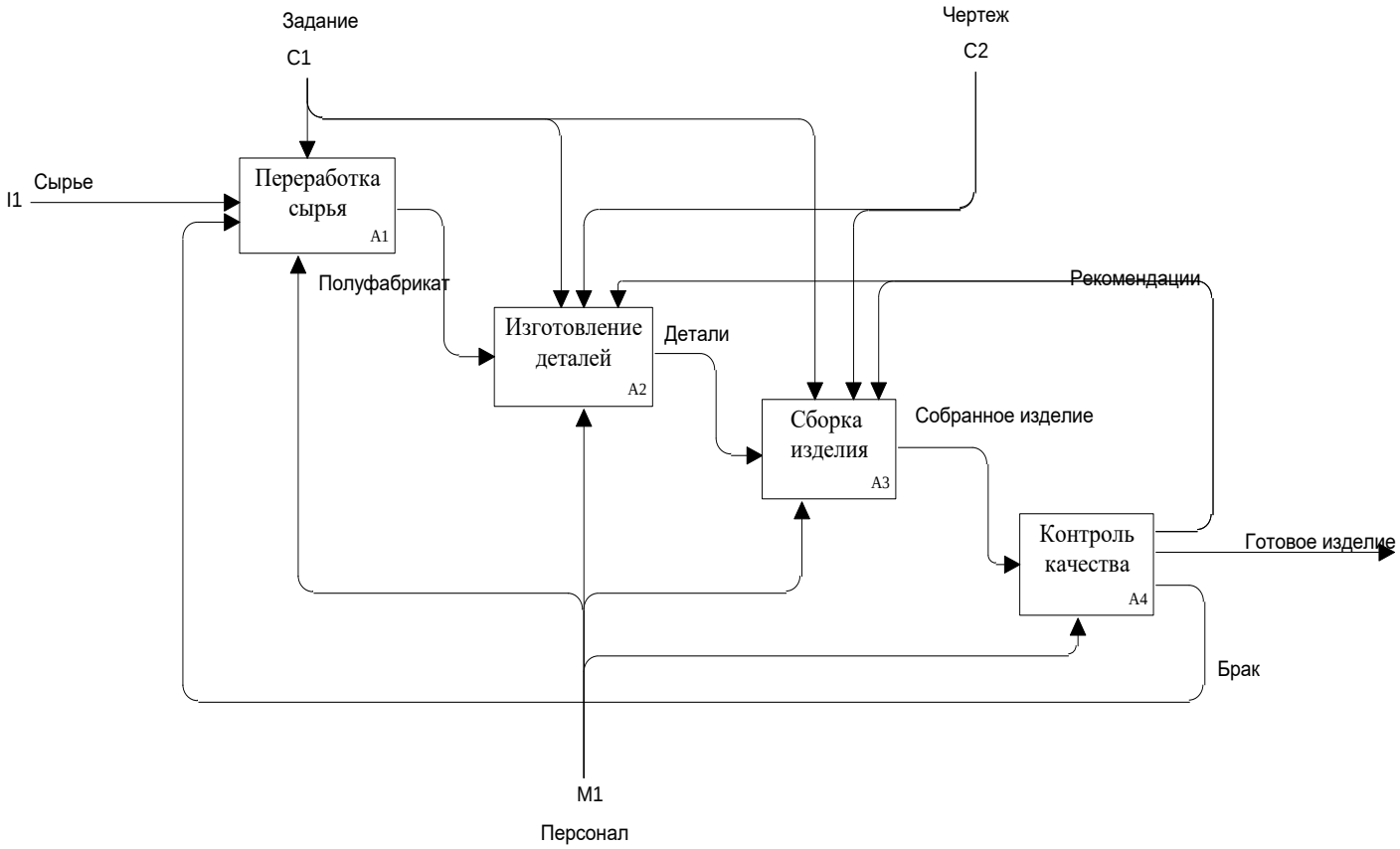
ПРАКТИЧЕСКАЯ РАБОТА №1 Знакомство с концептуальным проектированием предметной области.

Задачи:
1. Выбрать тему предметную область (ПО) и утвердить ее у преподавателя.
2. Неформально описать ПО на повествовательном языке.
3. Описать ПО в методологии функционального моделирования IDEF0 (построить функциональную модель).
3.1 Выделить бизнес-процессы ПО, определить входы и выходы;
3.2 Определить взаимосвязь между бизнес-процессами;
3.3 Выбрать приемлемый уровень декомпозиции функциональной модели;
3.4 Графически представить функциональную модель ПО в методологии IDEF0.
3.5 Графически представить функциональную модель ПО в методологии IDEF0 после внедрения автоматизированной системы.
4. Построить концептуальную информационную модель данных для ПО.
4.1 Определить основные объекты ПО и их характеристики (атрибуты);
4.2 Определить связи между объектами ПО;
4.3 Графически представить концептуальную информационную модель данных.

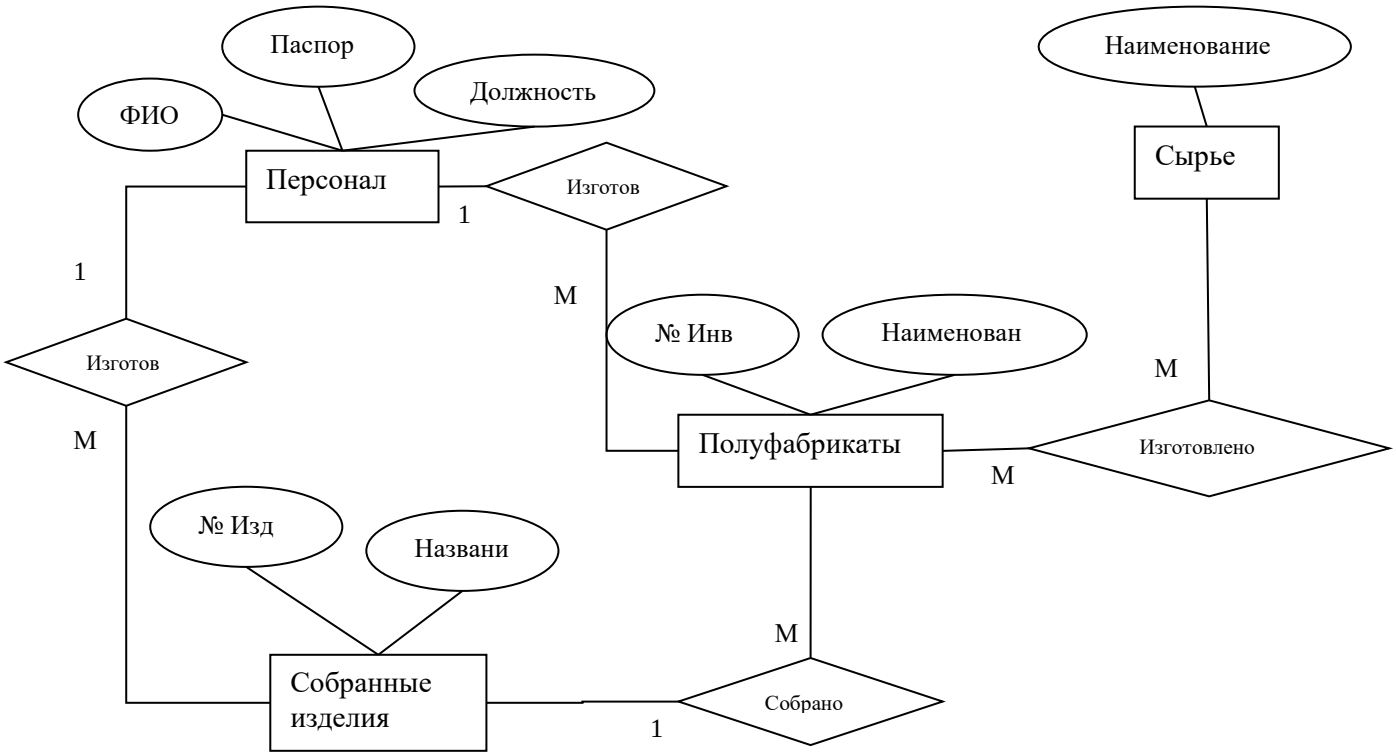
Индивидуальное задание выполнятся в виде отчета, и защищается у преподавателя. Все вышеуказанные задачи должны быть выполнены, и это должно быть отражено в отчете.

Пример построение модели предметной области в методологии IDEF0





Пример построения концептуальной информационной модели данных предметной области



Список предметных областей

1) Складской учет (поступление товара на склад по накладной, продажа товара со склада выпиской счет фактур, учет товаров на складах).

2) Телефонный справочник (поиск по телефону, адресу и владельцу, возможность двух режимов работы: пользовательском - когда возможна справочная работа с базой данных и административном - когда возможно дополнение/изменение информации).

3) Расчет заработной платы работников организации (начисления, удержания, ставки, стаж, тарифная сетка по стажу, табель, районный коэффициент, детские, отчисления и др.).

4) Железнодорожные кассы / авиакассы (продажа билетов, поиск/составление маршрутов, выдача информации о наличие билетов, предварительный заказ билетов/бронирование мест).

5) Работа поликлиники (расписание работы врачей, запись на прием, выбор из возможных вариантов времени и даты клиенту посещения врача, фиксация приема у врачей, оплата услуг, назначенные процедуры и анализы и др.).

6) Деятельность любого магазина/розничной/оптовой точки (сеть магазинов/торговых точек, прайс-лист по текущему состоянию на складе, проверка наличия того или иного товара как внутри одного магазина/торговой точки так и в нескольких, продажа товаров, учет товаров на складах, заказ товаров, история цен

и др.).
7) Аптека (аналогично с магазином/торговой точкой, но, например, предусмотреть замену лекарств аналогами).
8) Туристическое агентство (оформление и продажа путевок в различные страны с различным сервисом и длительностью).
9) Библиотека (картотека, выдача книг читателям и пр., в том числе мультимедиа Библиотека (CD, DVD, кассеты, мини-диски, пластинки и т.п.)).
10) Работа с электронными картами/со счетом в банке (срок действия карты/счета, номер и ПИН код, валидность карты/счета, контроль остатка, осуществление операций по картам/счетам и др.).
11) Видеотека/CDтека (можно предусмотреть прокат, продажу и др.).
12) Автомагазин (аналогично с магазином/торговой точкой, но, например, предусмотреть контроль пробега, год выпуска, марки, БУ и др.).
13) Страховая компания (выдача полисов, страхование имущества и др.).
14) Электронный магазин (имитация электронного магазина: авторизация входа, просмотр каталогов товаров, отображение товаров на складе, помещение в корзину/удаление из корзины, проверка валидности карт и пр.).
15) Электронная кафедра/деканат/представительство ВУЗа (часть его деятельности, учебные планы, состав студентов, преподавателей, сотрудников, нагрузка, успеваемость, расписание, оплата обучения, переводы, академотпуски и пр.).
16) Отдел кадров (ведение приказов о принятии на работу, увольнении, взысканиях, поощрения, перевод на другую должность, стаж).
17) Агентство недвижимости (аренда, продажа квартир, поиск подходящих вариантов и пр.).
18) ГИБДД (учет зарегистрированных и угнанных автомобилей, выдача прав, техосмотр и пр.).
19) Производство/производственный процесс (проекты/разработки, детали, материалы, полуфабрикаты, поставщики, выпуск продукции и пр.)
20) Реализация электронных тестов (список тестов, авторизация тестируемого, оценка, возможность повторного тестирования с отображением новой оценки и прогресса в сравнении с предыдущим тестом и пр.).
21) Сервисный центр (гарантийный и пост гарантийный ремонт различных изделий и пр.).
22) Соревнования (можно как групповые турниры, так и индивидуальное первенство, информация об участниках соревнований, результатах соревнований, призах, судейском персонале и пр.).
23) Агентство недвижимости (продажа, аренда жилья и пр.).
24) Школа (информацию о классах, предметах, учениках, преподавательском составе, распределении помещений, и расписании занятий, информация о родителях учащихся, списки предметов и отчеты об успеваемости учащихся каждого класса и пр.).
25) Паспортный стол (паспорта РФ и зарубежные, выдача, обмен, смена фамилий, места жительства и пр.).
26) Салон красоты (прайс-лист услуг, запись на услуги и пр.).
27) Оператор сотовой связи (сведения об абонентах, их звонках, остатках на счетах

и пр.).
28) ЗАГС (регистрация актов гражданского состояния).
29) Расчет рейтинга по обучению.
30) Метео-служба.
31) Домашняя бухгалтерия.
32) Учет материальных ценностей в хозяйстве.
33) Учет командировочных расходов.
34) Ежедневник.
35) Справочная по принятым и отправленным документам.
36) Тотализатор.
37) УВД.
38) Расселение студентов в общежитии.
39) Казино.
40) Организация, занимающаяся разработкой программного обеспечения.
41) Интернет-кафе.
42) Такси/Таксопарк.
43) Организация концертов/свадеб/юбилеев.
44) Жилищно-коммунальные услуги.

ПРАКТИЧЕСКАЯ РАБОТА №2 Знакомство с реляционным проектированием данных.

Задачи:
1. Определить отношения (таблицы) и связи между отношениями в предметной области на основе концептуальной информационной модели из индивидуального задания №1.
2. Определить атрибуты, а так же первичные и внешние ключи в отношениях (таблицах).
3. Подвергнуть полученные отношения процессу нормализации (использовать только первые три нормальные формы).
4. Полученный после нормализации результат представить в виде реляционной модели данных в методологии IDEF1x.

Индивидуальное задание выполнятся в виде отчета, и защищается у преподавателя. Все вышеуказанные задачи должны быть выполнены, и это должно быть отражено в отчете.

ЛАБОРАТОРНАЯ РАБОТА № 1 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ MICROSOFT SQL SERVER

Ц е л ь р а б о т ы : Реализация проекта учебной базы данных под управлением СУБД **Microsoft SQL Server.** Создание реляционных таблиц и установление межтабличных связей.

1 Краткие теоретические сведения

База данных (БД) — совокупность определенным образом организованных данных в рамках некоторой предметной области.

Например: база данных туристического агентства, библиотеки, отдела кадров, магазина и др.

Данные при этом могут храниться на разных носителях. Современным средством хранения и обработки баз данных является компьютер.

База данных - это объективная форма представления и организации совокупности данных, систематизированных таким образом, чтобы эти данные могли быть найдены и обработаны с помощью ЭВМ.

База данных представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Система управления базой данных (СУБД) - специализированная программа (комплекс программ), предназначенная для организации и ведения БД.

СУБД предназначена для:

- создания БД;
- обеспечения эффективного доступа к данным в БД;
- управления данными в БД;
- поддержания БД в актуальном состоянии для предметной области;
- и др.

Microsoft SQL Server – система управления базами данных (СУБД).

Microsoft SQL Server – система управления реляционными базами данных (РСУБД).

Реляционная модель основана на математическом понятии **отношения** (англ. relation), наглядным представлением которого является **таблица**.

Реляционная база данных (РБД) представляет собой множество взаимосвязанных реляционных таблиц. Таблицы организованы в виде строк и столбцов. Каждая строка таблицы имеет одинаковую структуру и состоит из полей. Строкам таблицы соответствуют кортежи, а столбцам – атрибуты отношения.

Код сотрудника	Фамилия	Имя	Отчество	Должность
1	Петров	Иван	Иванович	Менеджер
2	Петрова	Ирина	Михайловна	Бухгалтер
3	Петров	Иван	Иванович	Менеджер

Рисунок 1.1 – Пример реляционной таблицы БД

Каждая реляционная таблица БД хранит данные об определенном объекте (сущности) предметной области.

Объекты (сущности) – это вовлеченные в деятельность люди, предметы, события, документы, места и т.п., информация о которых должна накапливаться в разрабатываемой БД и которые представляют интерес для пользователей.

На поля в таблицах БД могут быть наложены ограничения. Например:

- уникальность значений;
- значения по умолчанию;
- обязательное указание;
- диапазон значений;
- список значений;
- условия на значения.

Для однозначного определения каждой записи таблица должна иметь **первичный ключ.**

Первичный ключ – это атрибут или набор атрибутов, значения в которых позволяют однозначно определять запись в таблице.

Требования к первичному ключу:

- должен однозначно определять строку таблицы;
- должен быть уникален;
- не должен содержать пустых значений (должен быть обязательно указан, NOT NULL);
- не должен быть избыточным.

Выбор первичного ключа лежит на разработчике БД.

На рисунке 1.1 в качестве первичного ключа может быть выбрано поле **Код сотрудника** при условии, что значения в данном поле будут уникальны, не будут пустыми. Поля по отдельности Фамилия, Имя, Отчество, Должность не могут быть выбраны в качестве первичного ключа, так как могут содержать повторяющиеся значения. Совокупность полей Фамилия, Имя, Отчество, Должность так же не могут быть выбраны в качестве первичного ключа, так как могут быть полные однофамильцы на одной и той же должности.

Связи между реляционными таблицами БД соответствуют связям между объектами (сущностями) предметной области.

Объекты (сущности) предметной области **состоят в связи**, если хотя бы одному экземпляру одного из них можно поставить в соответствии (по определенному правилу) один или более экземпляров другого.

Для реляционных таблиц БД: реляционные таблицы БД **состоят в связи**, если хотя бы одной записи одной из них можно поставить в соответствии (по определенному правилу) одну или более записей в другой таблице.

Внешний ключ (анг.foreign key (FK)) – поле или набор полей, предназначенное для организации связи между данными в таблицах БД.

Внешний ключ - это ссылка полей одной таблицы на поля другой таблицы.

При установке связей между таблицами необходимо помнить про **ссылочную целостность**.

Для ссылочной целостности характерно:

- первичный ключ (PK) таблицы должен содержать уникальные непустые значения для данной таблицы;

- внешний ключ (FK) потомка должен содержать только те значения, которые уже имеются среди значений PRIMARY KEY родителя (или колонок UNIQUE родителя);

- нельзя удалить экземпляр родителя, имеющий хотя бы один связанный с ним экземпляр потомка. Чтобы удалить экземпляр родителя, имеющего связанные потомки, необходимо сначала удалить связанные записи в потомке, а потом удалить экземпляр родителя с ними связанного. ИЛИ Можно установить механизм **каскадного удаления,** при котором удаляется экземпляр родителя вместе со связанными экземплярами потомка;

- нельзя изменить значение в PK (UNIQUE) родителя, имеющем хотя бы один связанный с ним экземпляр потомка. Можно установить механизм **каскадного обновления,** при котором изменяется PK (UNIQUE) родителя и значения FK связанных экземпляров потомка.

Ограничение FOREIGN KEY может содержать пустые значения (значения NULL). Кроме того, тип данных и размер колонки или колонок внешнего ключа должны совпадать со ссылочной колонкой или колонками.

Внешний ключ в таблице может ссылаться и на саму эту таблицу. В таких случаях говорят о **рекурсивном внешнем ключе**. Рекурсивные внешние ключи используются для хранения древовидной структуры данных в реляционной таблице.

Связь характеризуется мощностью связи.

Мощностью связи называется максимальное количество записей одной таблицы БД, связанных с одной записью другой таблицы БД.

Наиболее распространены мощности связи:

- один к одному 1:1;
- один ко многим 1:M;
- многие ко многим M:M.

Связь 1:1: одной записи в одной таблице соответствует одна запись в другой таблице.

Например: **Сотрудники 1 : 1 Дополнительно**

Сотрудники

Код сотрудника	Фамилия	Имя	Отчество	Должность
1	Петров	Иван	Иванович	Менеджер
2	Сидорова	Ирина	Петровна	Бухгалтер
3	Петров	Иван	Иванович	Менеджер

Дополнительно

Код сотрудника (FK)	Адрес	Паспортные данные
1	г. Томск	123456 и далее
2	г. Сургут	654321 и далее
3	г. Томск	789123 и далее

Связь 1:М: наиболее распространена. Одной записи в одной таблице (родительской) соответствует несколько записей в другой таблице (дочерней, потомок). Запись в дочерней таблице связана с одной записью в родительской таблице.

Например: **Сотрудники 1:М Продажи**

Сотрудники

Код сотрудника	Фамилия	Имя	Отчество	Должность
1	Петров	Иван	Иванович	Менеджер
2	Сидорова	Ирина	Петровна	Бухгалтер
3	Петров	Иван	Иванович	Менеджер

Продажи

Номер продажи	Дата	Код сотрудника (FK)
1П	12.02.2008	1
2П	12.02.2008	1
3П	12.02.2008	2
4П	20.02.2008	3
5П	21.02.2008	1

Связь М:М - одной записи первой таблицы может соответствовать несколько записей во второй таблице. И одной записи второй таблицы может соответствовать несколько записей в первой таблице.

Например: **Продажи М:М Блюда**

Продажи

Номер продажи	Дата	Код сотрудника (FK)
1П	12.02.2008	1
2П	12.02.2008	1
3П	12.02.2008	2
4П	20.02.2008	3
5П	21.02.2008	1

Блюда

Код блюда	Название	Тип	Цена меню	Выход
1Б	Оливье	Салаты	100 руб.	250 гр.
2Б	Борщ	Супы	90 руб.	300 гр.
3Б	Цезарь	Салаты	120 руб.	270 гр.

Для моделирования связи М:М в БД вводится дополнительная связующая таблица, которая разбивает связь М:М на связи 1:М и М:1:

Продажи

Номер продажи	Дата	Код сотрудника (FK)
1П	12.02.2008	1
2П	12.02.2008	1
3П	12.02.2008	2
4П	20.02.2008	3
5П	21.02.2008	1

Блюда

Код блюда	Название	Тип	Цена меню	Выход
1Б	Оливье	Салаты	100 руб.	250 гр.
2Б	Борщ	Супы	90 руб.	300 гр.
3Б	Цезарь	Салаты	120 руб.	270 гр.

Продано

Номер продажи (FK)	Код блюда (FK)	Количество	Цена	Скидка
1П	1Б	2	100 руб.	0%
1П	3Б	1	120 руб.	0%
2П	2Б	10	90 руб.	10%
3П	1Б	3	100 руб.	0%
3П	2Б	1	90 руб.	0%
3П	3Б	1	120 руб.	5%

Вторичные ключи устанавливаются по полям, которые часто используются при поиске или сортировке данных. Построенные по вторичным ключам индексы помогут системе значительно быстрее найти нужные значения, хранящиеся в соответствующих полях. В отличие от первичных ключей, поля для вторичных ключей могут содержать неуникальные значения.

Например, можно проиндексировать в таблице Блюда поля Тип, Название и др.

2 Особенности работы с Microsoft SQL Server

СУБД Microsoft SQL Server основана на стандарте SQL с некоторыми добавленными компанией Mictrosoft расширениями. Microsoft SQL Server распознает язык **Transact-SQL**, или, сокращенно, T-SQL.

Существуют различные инструменты, которые позволяют управлять Microsoft SQL Server, преобразуя щелчки мышью на определенных инструментах в соответствующие команды T-SQL, отправляемые ядру базы данных. Например, в **Microsoft Visual Studio, SQL Server Management Studio** и др., которые позволяют разработчикам и администраторам баз данных создавать проекты баз данных и управлять СУБД SQL Server с помощью графического интерфейса пользователя либо с помощью инструкций T-SQL.

В лабораторной работе описываются возможности SQL Server Management Studio для реализации проекта учебной базы данных под управлением СУБД Microsoft SQL Server.

Среда **SQL Server Management Studio** — это интегрированная среда для доступа, настройки, администрирования, разработки компонентов/объектов Microsoft SQL Server и управления ими.

В среде SQL Server Management Studio графические средства сочетаются с набором функциональных редакторов скриптов для доступа разработчиков и администраторов с разным уровнем знаний к SQL Server.

Главным инструментом SQL Server Management Studio является **Object Explorer (Обозреватель объектов)**, который позволяет пользователю просматривать, извлекать объекты сервера, а также ими управлять.

Также есть **SQL Server Management Studio Express** для Express версии сервера, которая является бесплатной. Однако в ней нет поддержки ряда компонентов (Analysis Services, Integration Services и др.).

Среда **Microsoft SQL Server Management Studio Express** — это бесплатная интегрированная среда для доступа, настройки, управления, администрирования и разработки всех компонентов SQL Server, а также объединение простых в использовании графических средств и редакторов с богатыми сценарными возможностями для доступа к SQL Server разработчиков и администраторов с любым уровнем подготовки.

Для доступа к Microsoft SQL Server необходимо наличие у пользователя соответствующей учетной записи для аутентификации.

Microsoft SQL Server предполагает использование двух различных режимов аутентификации:

- регистрационные записи SQL Server - учетные записи создаются и управляются в самой СУБД. Эти записи могут быть использованы только для доступа к серверу баз данных;

- учетные записи Windows - учетные записи Windows создаются для сетевого доступа и могут быть использованы также для доступа к серверу баз данных.

Использование для доступа к Microsoft SQL Server обоих методов называют **смешанным режимом**.

При использовании аутентификации Windows Microsoft SQL Server запрашивает регистрационные данные пользователя у операционной системы. Для регистрации пользователей в Windows используется подсистема Учетные записи пользователей панели управления.

Аутентификация Windows открывает пользователям доступ к базе данных, проверяя их регистрационные данные относительно учетных записей операционной системы. Это позволяет централизовать включение, отключение и применение особых настроек системы безопасности, связанных с учетными записями пользователей. Если какой- либо работник увольняется из организации, то отключение его учетной записи Windows автоматически закрывает ему доступ и к базе данных.

В Microsoft SQL Server можно создать собственные регистрационные записи. Создание отдельной учетной записи для служб Microsoft SQL Server обеспечит большую защиту и надежность, так как при этом можно снабдить ее минимальным набором необходимых разрешений, а недоступность пароля для обычных пользователей уменьшит возможность воспользоваться учетной записью службы для удовлетворения собственных потребностей.

2.1 База данных Microsoft SQL Server

База данных Microsoft SQL Server представляет собой одну логическую единицу, даже несмотря на то, что физически может размещаться в нескольких файлах.

Для создания БД можно использовать команду DDL **CREATE DATABASE** с соответствующими параметрами или воспользоваться соответствующим интерфейсом.

База данных состоит из двух файлов (точнее, из двух наборов файлов):
- файл данных с заданным именем и расширением **. mdf**;
- файл журнала транзакций с расширением **.ldf**.

Файл данных содержит системные и пользовательские таблицы, индексы, представления, хранимые процедуры, пользовательские функции, триггеры и разрешения системы безопасности.

Журнал транзакций с последовательной записью является «центром» SQL Server. Все обновления данных вначале записываются в журнал транзакций и проверяются, что гарантирует запись обновлений в двух

местах. Журнал транзакций содержит не только пользовательские записи, но и индексные записи, разбиения страниц, реорганизации таблиц и многое другое. Файл транзакций содержит не только текущую информацию, но и все обновления файла данных, он, как правило, все время разрастается.

Файлы данных и журнала транзакций по умолчанию размещаются в каталоге, принятом в SQL Server по умолчанию. Место размещения файлов данных и журнала транзакций определяется в окне параметров сервера.

Подобно всем реляционным СУБД, Microsoft SQL Server является таблично ориентированной. После создания базы данных следующим действием является наполнение ее таблицами.

Можно создавать таблицы, используя DDL **CREATE TABLE** или работать в графической среде, например в Visual Studio, SQL Management Studio.

При этом можно создавать таблицы в так называемом конструкторе таблиц (проект таблицы).

2.2 Имена таблиц и столбцов

Microsoft SQL Server допускает использование в именах таблиц и столбцов до 128 символов Unicode, включая пробелы, а также символы верхнего и нижнего регистров.

Если в названии используются пробелы, то в дальнейшем придется заключать их в квадратные скобки. Пробел можно заменить, например, символов подчеркивания или комбинированные слова типа OrderDetail, набранные в верхнем и нижнем регистре. Имена необходимо задавать согласно их смыслу.

2.3 Конструктор таблиц

Конструктор таблиц - визуальный инструмент, который позволяет проектировать и визуализировать таблицы базы данных, к которой подключены. Можно использовать конструктор таблиц для создания, редактирования и удаления таблиц, столбцов, ключей, индексов, связей и ограничений.

Конструктор таблиц позволяет создавать новые таблицы, а также модифицировать структуру уже существующих. Интерфейс этого окна схож с интерфейсом аналогичного окна конструкторов прочих приложений.

Открыть конструктор таблиц можно путем создания новой таблицы или открытия существующей таблицы.

2.3.1 Создание новой таблицы

1. Щелкните правой кнопкой мыши на узле **Tables** (**Таблицы**) в обозревателе объектов.

2. Из выпадающего меню выберите пункт Создать таблицу.

3. После этого откроется конструктор таблицы, в котором можно создавать необходимую структуру.

4. Для таблицы введите имена столбцов, выберите типы данных, а также выберете допустимы ли пустые значения для каждого столбца. Кроме того можно установить необходимые ограничения.

5. Из меню Файл (**File**) выберете Сохранить имя таблицы (**Save table name**).

6. В диалоговом окне введите имя таблицы и нажмите кнопку **ОК**.

2.3.2 Отображение свойств таблицы в окне «Свойства»:

1. Щелкните правой кнопкой мыши таблицу в конструкторе таблиц.
2. Выберите Свойства (**Properties)** из контекстного меню.

Например, **свойства таблицы для Microsoft SQL Server:**

- **Name** (Имя) - отображает имя таблицы.
- **Database Name** (Название базы данных) - показывает имя базы данных выбранной таблицы.
- **Description** (Описание) - показывает описание выбранной таблицы. Чтобы увидеть полное описание или изменить его, нажмите **Description** и нажмите кнопку с многоточием (...) справа от свойства.
- **Server** (Сервер) - показывает имя сервера для источника данных.
- **Schema** (Схема) - показывает имя схемы, которой принадлежит эта таблица.
- **Is Replicated** (Реплицируется) - показывает, может ли таблица воспроизводится в другом месте.
- и др.

В конструкторе таблицы каждый столбец таблицы выглядит как строка в сетке, а в столбцах сетки представляют **свойства столбца таблицы**.

2.3.3 Свойства поля /столбца (Column Properties)

Свойства можно настраивать в разделе Свойства столбца (**Column Properties**) в конструкторе таблиц. Чтобы изменить свойства столбцов для баз данных Microsoft SQL Server, выберите столбец в конструкторе таблиц.

Например, **свойства столбца**:

- **Name** (Имя) - показывает название столбца.
- **Database** (База данных) - показывает название базы данных для выбранного столбца.
- **Data Type** (Тип данных) - отображает тип данных выбранного столбца.

- **Is Identity** (идентификатор) – показывает, является ли выбранный столбец идентификатором таблицы.

- **Length** (Длина) - показывает количество символов, допустимое для символьных типов данных.

- **Precision** (Точность) - показывает максимальное число знаков, допустимых для числовых типов данных. Это свойство показывает, 0 для нечисловых типов данных.

- **Scale** (Масштаб) - показывает максимальное количество цифр, которые могут появляться справа от десятичной точки для числовых типов данных. Это свойство показывает, 0 для нечисловых типов данных.

- **Allow Nulls** (Разрешить NULL значения) - показывает, можете ли или нет данный столбец содержать NULL значения.

- **Default Value** (Значение по умолчанию) – значения по умолчанию.

- **Formula** (Формула) - показывает формулу, которую использует выбранный столбец.

- **Description** (Описание) - отображает описание столбца. Чтобы увидеть полное описание или изменить его, нажмите Описание и нажмите кнопку с многоточием (...) справа от свойства.

- **Language** (Язык) - указывает язык, используемый для индексирования столбца.

- **Identity Increment** (Прирост идентификатора) - показывает прирост, который будет добавлен для каждой последующей строки. Если оставить эту ячейку пустой, значение 1 будет назначено по умолчанию.

- **Identity Seed** (начальное значение идентификатора) - показывает значение, присвоенное в первой строке в таблице. Если оставить эту ячейку пустой, значение 1 будет назначено по умолчанию.

- **Is Indexable** (Индексируется) - показывает, имеются ли в выбранном столбце данные, которые могут быть проиндексированы.

- **Is Replicated** (Реплицируется) - показывает, может ли этот столбец воспроизведен в другом месте.

- **Size** (Размер) - показывает размер в байтах для типа данных столбца. Например, NCHAR тип данных может иметь длину от 10 (количество символов), но это будет иметь размер 20 на счет для наборов символов Юникода.

- и др.

2.3.4 Вставка столбцов в таблицу в конструкторе таблицы

1. Откройте необходимую таблицу в конструкторе таблиц.
2. Щелкните правой кнопкой мыши строку в таблице и выберите **Insert Column** (Вставить столбец) из контекстного меню. Пустая строка столбец вставляется.

или

поместите курсор в первую пустую ячейку в столбце **Column Name** (Имя столбца).

3. В столбце **Column Name** (Имя столбца) введите имя столбца в ячейке.

4. Нажмите клавишу **TAB** для перехода к ячейке Тип данных (**Data Type**) и выберите тип данных из выпадающего меню.

5. Продолжите определять какие-либо другие свойства столбца во вкладке **Properties** (Свойства) столбца.

2.3.5 Удаление столбцов из таблицы

1. Выберите столбец, который необходимо удалить.

2. Щелкните правой кнопкой мыши столбец и выберите команду **Delete Column** (Удалить столбец) из контекстного меню.

3. Если столбец участвует в связях, сообщение попросит вас подтвердить удаление выбранных столбцов и их связей.

Если столбец не участвует в проверочном ограничении связи, то столбец буден удален из базы данных.

Если в столбце делается проверочное ограничение связи, то сервер базы данных отклонит изменения, когда вы пытаетесь сохранить свою работу. Чтобы удалить столбец, который участвует в проверочном ограничении связи, необходимо сначала изменить или удалить проверочное ограничение.

2.4 Типы данных столбцов в Microsoft SQL Server

Тип данных столбца служит для двух целей:

- обеспечивает первый уровень целостности данных. Символьные данные не могут быть занесены в столбцы типов даты-времени и числового типа;

- определяет объем дискового пространства, выделяемого для столбца.

2.4.1 Символьные типы данных

SQL Server поддерживает несколько символьных типов данных:

- **char (n)** – символьные данные фиксированной длины, не в Юникоде, с длиной *n* байт. Значение *n* должно находиться в интервале от 1 до 8000. Размер хранения данных этого типа равен *n* байт. Синонимом по стандарту ISO для типа **char** является **character**.

- **varchar** (*n* | **max**) - символьные данные переменной длины, не в Юникоде. *n* может иметь значение от 1 до 8 000. **max** означает, что максимальный размер хранения равен 2^31-1 байт. Размер хранения равен фактической длине данных плюс два байта. Введенные данные могут иметь длину 0 символов. Синонимами по стандарту ISO для типа **varchar** являются типы **char varying** или **character varying**.

- **nchar** (n) - символьные данные фиксированной длины в Юникоде. Аргумент *n* должен иметь значение от 1 до 4000. Размер хранилища вдвое

больше *n* байт. Синонимами по стандарту ISO для типа **nchar** являются типы **national char** и **national character**.

- nvarchar (n | **max**) - символьные данные в Юникоде переменной длины. Аргумент *n* может принимать значение от 1 до 4 000. Аргумент **max** указывает, что максимальный размер хранилища равен 2^31-1 байт. Размер хранилища в байтах вдвое больше числа введенных символов + 2 байта. Введенные данные могут иметь длину в 0 символов. Синонимами по стандарту ISO для типа **nvarchar** являются типы **national char varying** и **national character varying**.

- ntext - представляет символьные данные в Юникоде переменной длины, включающие до 2^30 – 1 (1 073 741 823) символов. Объем занимаемого этим типом пространства (в байтах) в два раза превышает число символов. Синонимом по стандарту ISO для типа **ntext** является **national text**.

- text - представляет данные, отличные от данных Юникод, представленные с использованием кодовой страницы сервера. Максимальная длина данных — 2^31 – 1 (2 147 483 647) символов. Если в кодовой странице сервера используются двухбайтовые символы, объем занимаемого типом пространства все равно не превышает 2 147 483 647 байт. Он может быть менее 2 147 483 647 байт — в зависимости от строки символов.

2.4.2 Числовые типы данных

SQL Server поддерживает несколько числовых типов данных.

- bit - целочисленный тип данных, который может принимать значения 1, 0 или NULL.

Типы точных числовых данных, использующие **целые значения**:

Тип данных	Диапазон	Хранилище
bigint	от -2^63 (-9 223 372 036 854 775 808) до 2^63-1 (9 223 372 036 854 775 807)	8 байт
int	от -2^31 (-2 147 483 648) до 2^31-1 (2 147 483 647)	4 байта
smallint	от -2^15 (-32 768) до 2^15-1 (32 767)	2 байта
tinyint	от 0 до 255	1 байт

- decimal[**(***p*[**,***s*] **)**] и **numeric**[**(***p*[**,***s*] **)**] - числа с фиксированной точностью и масштабом. При использовании максимальной точности числа могут принимать значения в диапазоне от -10^38+1 до 10^38-1. В стандарте ISO синонимом типа **decimal** являются типы **dec** и **dec(***p*, *s***)**. Тип **numeric** функционально равнозначен типу **decimal**.

p (точность)

Максимальное количество десятичных разрядов числа (как слева, так и справа от десятичной запятой). Точность должна принимать значение от 1 до 38. По умолчанию для точности принимается значение 18.

***s* (масштаб)**

Максимальное количество десятичных разрядов числа справа от десятичной запятой. Масштаб может принимать значение от 0 до *p*. Масштаб может быть указан только совместно с точностью. По умолчанию масштаб принимает значение 0; поэтому 0 <= *s* <= *p*. Максимальный размер хранилища зависит от точности.

Точность	**Байты хранилища**
1 - 9	5
10-19	9
20-28	13
29-38	17

Типы данных, представляющие денежные (валютные) значения:

Тип данных	**Диапазон**	**Хранение**
Money	От -922 337 203 685 477,5808 до 922 337 203 685 477,5807	8 байт
smallmoney	От -214 748,3648 до 214 748,3647	4 байта

Типы приблизительных числовых данных, используемые для числовых данных с плавающей запятой. Данные с плавающей запятой являются приблизительными, поэтому не все значения из диапазона могут быть отображены точно.

Тип данных	**Диапазон**	**Хранение**
Float	- 1,79E+308 — -2,23E-308, 0 и 2,23E-308 — 1,79E+308	Зависит от значения *n*
Real	- 3,40E + 38 — -1,18E - 38, 0 и 1,18E - 38 — 3,40E + 38	4 байта

float [**(***n***)**]

Где *n* — это количество битов, используемых для хранения мантиссы числа в формате **float** при экспоненциальном представлении. Определяет точность данных и размер для хранения. Значение параметра *n* должно лежать в пределах от **1** до **53**. Значением по умолчанию для параметра *n* является **53**.

n	**Точность**	**Объем памяти**
1–24	7 знаков	4 байта

25–53 15 знаков 8 байт

2.4.3 Типы данных даты и времени

Тип данных	Format	Диапазон	Точность	Объем памяти (в байтах)	Определяемая пользователем точность в долях секунды	Смещение часового пояса
Time	чч:мм:сс[.нннннн]	От 00:00:00.0000000 до 23:59:59.9999999	100 наносекунд	От 3 до 5	Да	Нет
date	ГГГГ-ММ-ДД	От 0001-01-01 до 9999-12-31	1 день	3	Нет	Нет
smalldatetime	ГГГГ-ММ-ДД чч:мм:сс	От 01.01.1900 до 06.06.2079	1 минута	4	Нет	Нет
datetime	ГГГГ-ММ-ДД чч:мм:сс[.ннн]	От 01.01.1753 до 9999-12-31	0,00333 секунда	8	Нет	Нет
datetime2	ГГГГ-ММ-ДД чч:мм:сс[.нннннн]	От 0001-01-01 00:00:00.0000000 до 9999-12-31 23:59:59.9999999	100 наносекунд	От 6 до 8	Да	Нет
datetimeoffset	ГГГГ-ММ-ДД чч:мм:сс[.нннннн] [+|-]чч:мм	От 0001-01-01 00:00:00.0000000 до 9999-12-31 23:59:59.9999999 (время в формате UTC)	100 наносекунд	От 8 до 10	Да	Да

Функции, получающие компоненты даты и времени:

Функция	Синтаксис	Возвращаемое значение	Тип возвращаемых данных
DATENAME	DATENAME (*datepart*, *date*)	Возвращает строку символов, представляющую указанную часть *datepart* заданной даты.	**nvarchar**
DATEPART	DATEPART (*datepart*, *date*)	Возвращает целое число, представляющее указанный компонент *datepart* указанной даты *date*.	**int**
DAY	DAY (*date*)	Возвращает целое число,	**int**

		представляющее день указанной даты *date*.	
MONTH	MONTH (*date*)	Возвращает целое число, представляющее месяц указанной даты *date*.	**int**
YEAR	YEAR (*date*)	Возвращает целое число, представляющее год указанной даты *date*.	**int**

Функции системной даты и времени меньшей точности:

Функция	Синтаксис	Возвращаемое значение	Тип возвращаемых данных
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	Возвращает значение типа **datetime2(7)**, которое содержит дату и время компьютера, на котором запущен экземпляр SQL Server. Смещение часового пояса не включается.	**datetime**
GETDATE	GETDATE ( )	Возвращает значение типа **datetime2(7)**, которое содержит дату и время компьютера, на котором запущен экземпляр SQL Server. Смещение часового пояса не включается.	**datetime**
GETUTCDATE	GETUTCDATE ( )	Возвращает значение типа **datetime2(7)**, которое содержит дату и время компьютера, на котором запущен	**datetime**

экземпляр SQL Server. Возвращаемые дата и время отображаются в формате UTC.

2.4.4 Прочие типы данных

- image - этот тип представляет двоичные данные переменной длины, включающие от 0 до 2^31 – 1 (2 147 483 647) байт.

Типы двоичных данных фиксированной или переменной длины:

- binary [(*n*)] - двоичные данные фиксированной длины размером в *n* байт, где *n* — значение от 1 до 8000. Размер хранения составляет *n* байт.

- varbinary [(*n* | **max**)] - двоичные данные переменной длины. *n* могут иметь значение от 1 до 8000; **max** означает максимальную длину хранения, которая составляет 2^31-1 байт. Размер хранения — это фактическая длина введенных данных плюс 2 байта. Введенные данные могут иметь размер 0 символов. В ANSI SQL синонимом для **varbinary** является **binary varying**.

Тип **binary** применяется в тех случаях, когда размер данных в столбце поддается предварительной оценке.

Если же размер данных в столбце заранее определить трудно, пользуйтесь типом **varbinary**.

Если элементы данных в столбце превышают в длину 8000 байт, пользуйтесь типом данных **varbinary(max)**.

Тип данных, в котором хранятся XML-данные. Можно хранить экземпляры **xml** в столбце либо в переменной типа **xml**.

Примечание:

Типы данных **ntext**, **text** и **image** в будущей версии Microsoft SQL Server будут удалены. Следует избегать их использования при разработке новых приложений и запланировать изменение приложений, в которых эти типы в данное время используются. Вместо этих типов данных следует использовать типы **nvarchar(max), varchar(max) и varbinary(max).**

2.5 Работа с индексами

Индекс представляет собой механизм, обеспечивающий быстрый доступ к строкам таблицы или для соблюдения определенных ограничений.

2.6 Ограничение уникальности

2.6.1 Создание ограничения уникальности

1. Выберите таблицу, к которой вы хотите добавить ограничение уникальности.
2. Откройте таблицу в конструкторе таблиц.
3. В меню конструктора таблицы выберете **Indexes/Keys**.
4. В диалоге **Indexes/Keys** нажмите кнопку **Add** .
5. В сетке выберите тип (**Type**) и выберете уникальный ключ (**Unique Key**) из раскрывающегося списка справа от свойства.
6. Ограничение уникальности, созданные в базе данных, сохраняются при сохранении таблицы.

2.6.2 Удаление ограничения уникальности

1. В выбранной таблице в режиме конструктора выберите **Indexes/Keys**.
2. В диалоге **Indexes/Keys** выберете уникальный индекс в **Selected Primary/Unique Key and Index list.**
3. Нажмите кнопку **Delete**.
4. Ограничение удаляется из базы данных при сохранении таблицы.

2.7 Ограничение первичного ключа

Ограничение первичного ключа гарантируют уникальность первичного ключа и отсутствие в нем пустых значений.

Таблица может иметь только один первичный ключ. Первичный ключ может быть составным. Если первичный ключ состоит более чем из одного столбца, повторяющиеся значения допускаются в один столбец, но каждая комбинация значений всех столбцов в первичном ключе должны быть уникальными.

2.7.1 Создание первичного ключа

1. В конструкторе таблицы нажмите кнопку выделения строки для столбца, который необходимо определить в качестве первичного ключа. Если хотите выбрать несколько столбцов, нажмите клавишу CTRL для других столбцов.
2. Щелкните правой кнопкой мыши выделения строки для столбца и выберите **Set Primary Key** (Задать первичный ключ). Первичный ключ индекса, под названием "PK_", а затем имя таблицы, создается автоматически, вы можете найти его в диалоговом окне Индексы /Ключи (**Indexes/Keys**).

2.7.2 Удаление ограничения первичного ключа

1. В режиме конструктора таблицы щелкните правой кнопкой мыши на строке с первичным ключом и выберите **Remove Primary Key** (Удалить первичный ключ).

2.8 Свойства для столбца идентификатора автоинкремента (счетчика)

1. Откройте в режиме конструктора необходимую таблицу.
2. Уберите флаг **Allow nulls** для столбца, который нужно изменить.
3. В свойствах столбца (**Column Properties**) раскройте свойство **Identity Specification**.
4. Щелкните ячейку сетки **Is Identity** и выберете **Yes** (Да) в раскрывающемся списке.
5. Введите значение в ячейке **Identity Seed**, которое будет назначено на первую строку в таблице. 1 будет назначен по умолчанию.
6. Введите значение в ячейке **Identity Increment** значение приращения автоинкремента. Это значение приращения, которые будут добавлены для каждой последующей записи. 1 будет назначен по умолчанию.

IDENTITY [(seed , increment)]

Аргументы:

seed Значение, присваиваемое самой первой строке, загружаемой в таблицу.

increment Значение приращения, которое прибавляется к значению идентификатора предыдущей загруженной строки.

Необходимо указывать либо оба аргумента (и seed, и increment), либо не указывать ни одного из них. Если ничего не указано, применяется значение по умолчанию (1,1).

2.9 Значение по умолчанию

Считаются свойством столбца, так как никаких проверок ввода данных пользователем не проводится, также не форсируется поддержание целостности данных. В то же время значения, заданные по умолчанию, являются ценным параметром столбца.

Значение по умолчанию может быть одним из следующих:

- допустимое статическое числовое или символьное значение;
- системная функция, такая как GetDate() и др.
- функция, определенная пользователем;
- пустое значение.

Тип значения по умолчанию должен быть совместим с типом данных столбца.

Указание значения по умолчанию для столбца:

1. Выберите столбец, для которого вы хотите задать значение по умолчанию.
2. В колонке на вкладке **Properties** (Свойства) введите новое значение по умолчанию **Default Value or Binding.**

Если поле строкового типа, то значение по умолчанию для строк текста, заключите в одинарные кавычки ('). Чтобы ввести числовые умолчанию, введите число. Для ввода объекта /функции можете ввести имя объекта / функции без единой кавычки.

2.10 Допустимость NULL значений

NULL значение можно расценивать как неизвестное. Как правило, NULL значения появляются при неполном вводе пользователем данных в строку. Допустимы ли в столбце NULL значения, определяется с помощью атрибута столбца NULL или NOT NULL

Если NULL значения не допускаются, пользователю, который вводит данные в таблице, необходимо ввести значение в поле, так как запись не может быть сохранена в базе данных.

Изменение параметров по NULL столбцу:

1. Выберите столбец и на вкладке **Properties** (Свойства) колонке установите или уберите флаг **Allow nulls** (допускать NULL значения).

2.11 Ограничение проверки

Ограничения проверки (**check constraint**) являются проверкой целостности данных на уровне строк.

Могут быть диапазон, список, условия на значения и др. Как правило, это небольшая формула, которая возвращает булево значение true или false. В ограничения проверки могут быть включены такие операторы, как =,= >, <, >=, <=, BETWEEN, LIKE, IN и др. Можно создать сложное выражение, используя логические операторы, чтобы проверить данные по нескольким условиям. Ограничения проверки могут содержать пользовательские функции, которые могут выполнять последовательность инструкций. Ограничения проверки полезны для поддержания общих правил допустимости данных, а также простых правил бизнес-логики.

Примеры операторов:

Оператор	Описание	Пример
=	**Равно**	**Цена =1000** записи с ценой равной

		1000 **Тип = ’Фрукты’** записи с типом фрукты
<>	**Не равно**	**Количество <>500** записи с количеством не равным 500
>	**Больше**	**Цена >1000** записи с ценой больше 1000
<	**Меньше**	**Количество <10** записи в количестве менее 10
>=	**Больше или равно**	**Цена >=1000** записи с ценой больше или равной 1000
<=	**Меньше или равно**	**Количество <=10** записи в количестве менее или равным 10
AND	**Логическое И** Таблица истинности:	**Цена>=100 AND Цена <=500** записи с ценой от 100 до 500 включительно
OR	**Логическое ИЛИ** Таблица истинности:	**Тип = ’Овощи’ OR Тип = ’Фрукты’** записи с типом фрукты или овощи
NOT	**Реверс значения логического выражения** Таблица истинности:	**Тип NOT ‘Фрукты’** типы кроме фруктов

LIKE	Для отбора данных по шаблону. Синтаксис: **<выражение>LIKE<шаблон>** При этом в шаблоне	**Название LIKE '_a%'** записи, в названии которых вторая буква а

A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

A	NOT A
1	0
0	1

	используются обозначения: _ - один символ, % - любое количество символов, [список] - любой символ из списка.	**Фамилия LIKE '[А,Б]%'** фамилии, начинающиеся на букву А или Б **Тип LIKE '[А-Д]%'** записи с типом, начинающимся на буквы от А до Д
IN	Позволяет выбрать значения из заданного списка. Синтаксис: **<выражение>IN(<выражение>,|; <выражение>,|;...)**	**Пол IN ('мужской'; 'женский')** записи с полом мужской или женский
BETWEEN	Проверяет, находится ли (или нет) выражение в заданном диапазоне. Синтаксис: **<выражение>BETWEEN <нижнее значение> AND <верхнее значение>**	**Цена BETWEEN 1000 AND 12000** записи с ценой от 1000 до 12000 включите
И др.		

2.11.1 Создание ограничения проверки

1. Откройте конструктор таблицы для таблицы, которая будет содержать ограничение.
2. Щелкните правой кнопкой мыши в конструкторе таблицы и выберете **Check Constraints** из контекстного меню.
3. Нажмите кнопку **Add** (Добавить).
4. Введите имя для ограничения в **Constraint name**.
5. В поле выражения (**Expression**) введите выражение типа SQL выражений для проверки ограничения. Например:
   Город = 'Томск'
   Индекс LIKE '[0-9][0-9][0-9][0-9][0-9]'
   Цена >=0
   и др.
6. Разверните окно с проверочными ограничениями для установки, когда ограничение обеспечивается:

- **Check Existing Data on Creation or Enabling** - для проверки ограничения на данные, которые существовали прежде, чем создано ограничение, проверка существующих данных на создание и включение.

- **Enforce For Replication** - для обеспечения соблюдения ограничений, когда операции репликации происходит на этой таблице.

- **Enforce for INSERTs and UPDATEs** - для обеспечения соблюдения ограничений, когда строка этой таблицы вставляется или обновляется,

проверить Обеспечение для вставок и обновлений.

2.11.2 Удаление проверочного ограничения

1. В меню конструктора таблицы нажмите кнопку **Check Constraints**.
2. В диалоге **Check Constraints** выберете необходимое ограничение в списке ограничений.
3. Нажмите кнопку **Delete** (Удалить).

Ограничение будет удалено из базы данных при сохранении таблицы.

2.12 Определение вычисляемого столбца

1. Выберите столбец, для которого хотите задать формулу вычисляемого столбца.
2. В колонке на вкладке **Properties** (Свойства) в **спецификации вычисляемого столбца** укажите в **Formula** (Формула) выражение для этого столбца.

2.13 Изменение точности типа данных столбцов

Свойство Точность **(Precision)** относится к **decimal** и **numeric** типам.

1. Выберите столбец, для которого вы хотите изменить точность.
2. В колонке на вкладке **Properties** (Свойства) щелкните на ячейке для свойства точности **Precision** и введите значение для точности.

2.14 Установка сортировки для колонки

1. Выберите столбец, для которого хотите установить параметры сортировки (**collation**).
2. Нажмите **Collation** и нажмите кнопку с многоточием (...) справа от свойства. Появится диалоговое окно для сортировки (**Collation dialog box**).
3. Выберите настройки параметров сортировки для выбранного столбца в диалоговом окне Параметры сортировки.

2.15 Установка связей

Связи можно устанавливать различными способами.

2.15.1 Создание внешнего ключа в конструкторе таблиц

1. Откройте конструктор таблицы, в которой будет находиться внешний ключ.

2. В меню конструктора таблицы нажмите кнопку связей/отношений **(Relationships**) или в конструкторе таблиц щелкните на кнопке **Relationships** панели инструментов.
3. В диалоговом окне **Foreign Key Relationships** нажмите кнопку **Add** (Добавить).
4. Нажмите на связь в этом списке связей.
5. Нажмите **Tables and Columns Specification** и нажмите кнопку с многоточием (...) справа от свойства.
6. В диалоге **Tables and Columns** в раскрывающемся списке выберите родительскую таблицу в **Primary Key Table** в связи.
7. Затем укажите дочернюю таблицу из списка в **Foreign key table**.
8. Укажите соответствующие поля для связи из выпадающего списка в диалоге **Tables and Columns.**
9. Укажите имя для связи.
10. Выберите **ОК**, чтобы создать связь.

2.15.2 Диалог внешнего ключа (Foreign Key Relationships)

Используйте это окно для создания или изменения связей между таблицами в базе данных.

Внешний ключ может ссылаться на ограничение первичного ключа или ограничение уникальности в другой таблице.

Сложности, создаваемые ссылочной целостностью, сводятся к недопустимости удаления или модификации строки родительской таблицы до тех пор, пока существуют строки дочерней таблицы, ссылающиеся на нее. Если последние остаются после удаления строки родительской таблицы, то нарушаются требования ссылочной целостности. Решение этой проблемы сводится к модификации строк дочерней таблицы как части транзакции, изменяющей строки родительской таблицы.

Чтобы получить доступ к диалоговому окну внешнего ключа, откройте конструктор для таблицы с внешним ключом, щелкните правой кнопкой мыши сетку определения таблицы и нажмите кнопку связей (**Relationships**).

В данном диалоге:

- **Add** - создание новых связей.

- **Delete** - удаляет выбранную связь в списке связей.

- **Check Existing Data on Creation or Re-Enabling** - проверить имеющиеся данные на создание или повторное включение.

- **Tables and Columns Specifications** – показывает какие таблицы и по каким столбцам связаны, что выступает в качестве внешнего ключа и первичного (или уникального) ключа в связях. Чтобы изменить или определить эти значения, нажмите кнопку с многоточием (...) справа от свойства.

- **Foreign Key Base Table** - внешнего ключа таблица. Показывает, какая таблица содержит столбец, выступающий в качестве внешнего ключа в выбранной связи.

- **Foreign Key Columns** - столбцы внешнего ключа. Показывает, какой столбец действует как внешний ключ в выбранной связи.

- **Primary/Unique Key Base Table** - первичный / уникальный ключ родительской таблицы. Показывает, какая таблица содержит столбец, выступающий в качестве первичного (или уникального) ключа в выбранной связи.

- **Primary/Unique Key Columns** - показывает, какой столбец действует как первичный (или уникальный) ключ в выбранной связи.

- **Name** (Имя) - показывает имя связи.

- **Description** (Описание) – показывает описание связей.

- **Enforce for Replications** - поддерживать репликации, если таблицы реплицируются в другой базе данных.

- **Enforce Foreign Key Constraint** - обеспечить ограничения внешнего ключа.

- **INSERT and UPDATE Specification** - расширение для отображения информации для правил удалении и правил обновлений.

- **Delete Rule** – правило на удаление.

- **Update Rule** – правило на обновление.

Для строк дочерней таблицы возможны четыре варианта действий, выбираемых в параметрах **Delete Rule** и **Update Rule** диалога **Foreign Key Relationships**:

- **No Action** - строки таблицы не изменяются. Их присутствие будет предохранять строки родительской таблицы от удаления и модификации. Используйте этот вариант, когда дочерние строки сопровождают значения родительских строк. Вы не сможете изменить или удалить родительские строки, если существуют связанные с ними дочерние.

- **Cascade** - операции удаления и модификации, выполняемые на родительских строках, выполняются также и на дочерних. Используйте этот вариант, когда дочерние данные теряют смысл в отсутствие родительских. Используются механизмы **каскадного удаления** и/или **каскадного обновления** соответственно.

- **Set Null** - этот вариант оставляет дочерние строки нетронутыми, однако изменяет значение внешнего ключа на NULL (разумеется, предполагается допустимость наличия NULL значений). Используйте этот вариант, если хотите разрешить удаление родительских строк без влияния на существование дочерних.

- **Set Default** – родительские строки могут удаляться и изменяться, при этом значения внешнего ключа изменяются на значения по умолчанию, заданные для данного столбца. Этот вариант аналогичен варианту **Set Null**, за исключением того, что на этот раз внешнему ключу присваивается определенное значение.

2.15.3 Удаление ограничения внешнего ключа

1. В меню конструктора таблицы нажмите кнопку связей/отношений **Relationships**.
2. В диалоге **Foreign Key Relationships** выберите связь в этом списке связей.
3. Нажмите кнопку **Delete** (Удалить).

2.16 Диаграмма базы данных (Database Diagram)

Создание диаграмм баз данных можно проводить с помощью инструмента **Database Diagram**, который берет за основу структуры таблиц, созданные в узле **Table**, создает графическое представление таблиц и связей между ними.

Этот инструмент имеет собственный узел в каждой из баз данных. Каждая база данных может иметь множество диаграмм, так как в каждой из них может быть представлен ее конкретный модуль. Вы можете создать одну или несколько диаграмм, иллюстрирующих некоторые или все таблицы, столбцы, ключи и связи в ней. Каждая таблица базы данных может появиться на разном количестве диаграмм. Таким образом, можно создавать различные диаграммы для визуализации различных частей базы данных или, чтобы подчеркнуть различные аспекты дизайна. Это облегчает работу с крупными базами данных.

2.16.1 Создание новой диаграммы базы данных

1. В обозревателе объектов щелкните правой кнопкой мыши узел Диаграммы базы данных (**Database Diagrams**) или диаграмму в этом узле.
2. Выберите **Создать диаграмму базы данных** в контекстном меню. Появится окно **Add Table** (Добавление таблицы).
3. Выберите необходимые таблицы в списке таблиц и нажмите кнопку Добавить (**Add**).

   Таблицы графически отображаются в новой диаграмме базы данных.

Вы можете продолжать добавлять или удалять таблицы, изменять существующие таблицы и изменять связи между таблицами.

После сохранения диаграммы базы данных вы можете сохранить все изменения, внесенные в нее, в том числе изменения, внесенные в таблицы, столбцы и другие объекты базы данных.

2.16.2 Сохранение диаграммы базы данных

1. В меню **File** (Файл) выберете **Save** (Сохранить).
2. Если это новая схема, введите имя диаграммы.
   Если вы внесли изменения в существующей диаграмме, в диалоговом окне Сохранить появляется список изменений, которые будут сохранены в базе данных при сохранении схемы.
3. Нажмите **Yes** (или **OK** если это новая диаграмма) для обновления базы данных по диаграмме.

2.16.3 Открытие диаграммы базы данных

1. Раскройте узел Диаграммы базы данных (**Database Diagrams**).
2. Дважды щелкните название диаграммы базы данных, которую требуется открыть.

Или щелкните правой кнопкой мыши имя диаграммы базы данных, которую требуется открыть, а затем выберете **Изменить**.

Диаграмма откроется в конструкторе, из которого вы можете редактировать диаграммы.

2.16.4 Удаление диаграммы базы данных

1. Раскройте узел Диаграммы базы данных (**Database Diagrams**).
2. Щелкните правой кнопкой мыши диаграмму базы данных, которую хотите удалить.
3. Выберите команду Удалить (**Delete)** в контекстном меню.
4. Сообщение попросит вас подтвердить удаление. Выберите **Да**.

2.16.5 Добавление таблицы в существующую диаграмму

1. Убедитесь, что вы подключены к диаграмме базы данных, которую необходимо изменить.
2. Выберите таблицу в папке Таблицы (**Tables**).
3. Перетащите таблицу в диаграмму базы данных, удерживая левую кнопку мыши.
4. Отпустите кнопку мыши.
   или
   Щелкните правой кнопкой мыши в конструкторе диаграммы и выберете **Добавить таблицу**.

2.16.7 Удаление таблицы из диаграммы базы данных

1. В диаграмме базы данных выберите таблицу, которую необходимо удалить.

2. Щелкните правой кнопкой мыши таблицу и выберите **Удалить таблицу из диаграммы** (**Remove Table from Diagram**) из контекстного меню.

   Или нажмите клавишу **ESC**.

Таблицы удаляются из диаграммы, но они продолжают существовать в базе данных.

2.16.8 Проведение связей в диаграмме базы данных

1. В диаграмму базы данных добавьте таблицы, которые необходимо связать.
2. Свяжите ключевое (уникальное) поле родительской таблицы с полем внешнего ключа дочерней таблицы путем перетаскивания, удерживая левую кнопку мыши.
3. В появившемся диалоговом окне **Tables and Columns** проверьте правильность установки связи.
4. Нажмите **ОК**, если связь устанавливается верно.
5. В появившемся диалоговом окне **Foreign Key Relationships** проверьте правильность установленной связи, установите, если необходимо, параметры в данном окне.
6. Нажмите **ОК**, если связь устанавливается верно.

3 Задание на лабораторную работу №1

Создайте базу данных для своей предметной области в выбранной СУБД.

Лабораторная работа выполняется по результатам разработанной структуры БД в индивидуальном задании согласно варианту.

Требования к базе данных (БД):

- БД должна быть реляционной;
- БД должна быть нормализована по 3НФ (иди БКНФ);
- БД должна состоять минимум из 3-4 связанных таблиц (на каждого студента);
- в таблицах БД должны быть наложены ограничения на поля таблиц (по усмотрению разработчиков решается, на какие поля будут наложены ограничения). Должно быть не менее 3 ограничений на разные поля таблиц (без учета ограничений PRIMARY KEY, FOREIGN KEY). Ограничения проверки, значения по умолчанию обязательно;
- должна быть обеспечена ссылочная целостность;
- в БД должны быть обязательно поля разных типов данных, в том числе поля:
  - позволяющие хранить рисунки;
  - позволяющие хранить длинный текст;
  - значения в которых должны соответствовать списку значений, например Пол мужской или женский;
  - значения логического типа.

Ниже приведен пример создания БД для кафе (описание БД приведено в разделе 4) под управлением СУБД Microsoft SQL Server (раздел 5).

4 Описание базы данных для кафе

Создадим следующую структуру базы данных (рисунок 4.1) для кафе.

Рисунок 4.1 – Структура БД для кафе

4.1 Описание таблицы Блюда

Имя поля	Тип данных поля	Описание	Ограничения
Код_блюда	Числовой	Код блюда в кафе. Уникальный идентификатор блюда.	Первичный ключ. Условие на значение: >=0
Название	Текстовый	Название блюда.	Размер поля 40. Обязательное поле.
Тип	Текстовый	Тип блюда (салаты, десерты, супы и др.).	Размер поля 30. Обязательное поле.
Цена_меню	Денежный	Цена блюда по меню.	Обязательное поле. Значение по умолчанию 0. Условие на значение: >=0
Выход	Числовой	Количество готового блюда в соответствующих единицах.	Значение по умолчанию 0. Условие на значение: >=0

Единицы	Текстовый	Единицы измерения	Размер поля 5. Допустимы значения "гр." или "мл." или "шт." Значение по умолчанию “гр.”

Фирменное_ блюдо	Логический	Указывает является ли данное блюдо фирменным	Да/Нет
Состав	Длинный текст	Описание состава блюда.	
Фото	Фотография	Фотография готового блюда.	

Примечание:

Фирменное блюдо — это особенное блюдо, которое готовится шеф-поваром. Это такое блюдо, которое нельзя попробовать в других кафе. Фирменное блюдо может представлять собой горячее, горячую или холодную закуску, салат, десерт или напиток.

4.2 Описание таблицы Сотрудники

Имя поля	Тип данных поля	Описание	Ограничения
Код_сотрудника	Счетчик	Уникальный идентификатор сотрудника.	Первичный ключ.
Фамилия	Текстовый	Фамилия сотрудника	Размер поля 30. Обязательное поле.
Имя	Текстовый	Имя сотрудника	Размер поля 30. Обязательное поле.
Отчество	Текстовый	Отчество сотрудника	Размер поля 30.
Должность	Текстовый	Должность сотрудника	Размер поля 30. Обязательное поле.
Пол	Текстовый	Пол сотрудника	Размер поля 10. Условие на значение:

			‘мужской’ или ‘женский’
Семейное_положение	Текстовый	Семейное положение	Размер поля 15. Условие на значение: ‘Женат’ или ‘Замужем’ или ‘Холост’ или ‘Не замужем’
Дети	Логический	Имеются ли дети у сотрудника	Да/Нет

4.3 Описание таблицы Продажи

Имя поля	Тип данных поля	Описание	Ограничения
Номер_продажи	Счетчик	Уникальный идентификатор продажи.	Первичный ключ.
Дата	Дата/время	Дата продажи	
Код_сотрудника	Числовой	Идентификатор сотрудника, который оформил продажу. Ссылается на таблицу Сотрудники.	Внешний ключ. Обязательное поле.
Номер_столика	Текстовый	Номер столика в кафе	Размер поля 10.

4.4 Описание таблицы Продано

Имя поля	Тип данных поля	Описание	Ограничения
Номер_продажи	Числовой	Номер продажи. Ссылается на таблицу Продажи.	Входит в состав первичного ключа. Внешний ключ.
Код_блюда	Числовой	Код блюда. Ссылается на таблицу Блюда.	Входит в состав первичного ключа. Внешний ключ.
Количество	Числовой	Количество проданных блюд	Целое число. Обязательное поле.

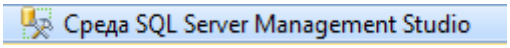
			Значение по умолчанию 1. Условие на значение: >0
Цена	Денежный	Цена блюда на момент продажи	Обязательное поле. Значение по умолчанию: 0 Условие на значение: >=0
Скидка	Числовой	Скидка на конкретное блюдо в продажи	Значение по умолчанию 0. Условие на значение: >=0 и <=1

Примечание:

В лабораторной работе №1 для удобства объяснения и понимания предметной области используются названия таблиц и полей на русском языке.

---

5 Создание БД под управлением СУБД Microsoft SQL Server для кафе

В данной работе описываются возможности среды SQL Server Management Studio  **для реализации проекта учебной базы данных под управлением СУБД Microsoft SQL Server .**

В среде **SQL Server Management Studio** графические средства сочетаются с набором функциональных редакторов скриптов для доступа разработчиков и администраторов с разным уровнем знаний к SQL Server.

Главным инструментом SQL Server Management Studio является **Object Explorer (Обозреватель объектов)**, который позволяет пользователю просматривать, извлекать объекты сервера, а также ими управлять.

Для использования обозревателя объектов следует сначала подключиться к серверу. На панели инструментов обозревателя объектов нажмите кнопку **Соединить** и выберите тип сервера из раскрывающегося списка. Открывается диалоговое окно **Соединение с сервером**. Для подключения следует указать, по крайней мере, **имя сервера и правильные данные для проверки подлинности**.

При подключении к серверу можно указать дополнительные данные соединения в диалоговом окне Соединение с сервером. В диалоговом окне Соединение с сервером сохраняются использованные в последний раз настройки, которые будут использоваться для новых соединений, например для открытия новых окон редактора кода.

Чтобы задать выборочные настройки соединения, выполните следующее.

1. На панели инструментов обозревателя объектов нажмите кнопку Соединить, а затем выберите тип сервера, к которому производится соединение. Откроется диалоговое окно Соединение с сервером.
2. В поле Имя сервера введите имя экземпляра SQL Server.
3. Нажмите **Параметры**. В диалоговом окне Соединение с сервером будут показаны дополнительные **параметры**.
4. Набор доступных параметров зависит от типа сервера.

Для **компонента Database Engine** доступны следующие параметры.

Параметр	Описание
Подключиться к базе данных	Позволяет выбрать одну из доступных баз данных на сервере. Список содержит только те базы данных, для просмотра которых имеется разрешение.
Сетевой протокол	Позволяет выбрать один из вариантов: «Общая память»,

	«TCP/IP» или «Именованные каналы».
Размер сетевого пакета	Задается в байтах. Значение по умолчанию равно 4 096 байтам.
Время ожидания соединения	Задается в секундах. Значение по умолчанию равно 15 секундам.
Время ожидания выполнения	Задается в секундах. Значение по умолчанию (0) означает, что выполнение никогда не будет прервано из-за превышения времени ожидания.
Шифрование соединения	Задает принудительное шифрование.

Чтобы добавить заданный сервер в список зарегистрированных серверов, перейдите на вкладку Зарегистрированный сервер, затем щелкните позицию, где следует поместить сервер, и завершите соединение.

В среде SQL Server Management Studio имеется множество средств для этапов разработки и администрирования. Некоторые средства можно использовать для любого компонента SQL Server, другие — только для определенных компонентов.

Обозреватель объектов может быть использован для всех компонентов SQL Server.

Инструмент	**Назначение**
Обозреватель объектов	Обзор серверов, создание и размещение объектов, управление источниками данных и просмотр журналов. Доступ к этому средству осуществляется из меню Вид.

В лабораторной работе используется СУБД **Microsoft® SQL Server® 2012 Express (Microsoft SQL Server 2012 Express LocalDB)** — мощная и надежная бесплатная система управления данными, обеспечивающая функциональное и надежное хранилище данных для веб-сайтов и настольных приложений.

SQL Server Express LocalDB — это упрощенная версия SQL Server, которая обладает множеством возможностей по программированию баз данных, аналогичных SQL Server.

SQL Server Express LocalDB выполняется в пользовательском режиме, и его можно установить более быстро, с меньшим числом требований и без конфигурации.

В следующей таблице описаны различия между SQL Server Express LocalDB и SQL Server Express.

	SQL Server Express LocalDB	**SQL Server Express**
Тип базы данных при создании базы данных, основанной на службах	В Visual Studio 2012 и Visual Studio 2013 — SQL Server Express LocalDB	В Visual Studio 2010 и более ранних версиях — SQL Server Express
Имя экземпляра SQL Server в средствах и параметрах	(LocalDB)\v11.0	SQLEXPRESS
Значение источника данных в строке подключения	(LocalDB)\v11.0	.\SQLEXPRESS
Значение AttachDbFilename в строке подключения	путь к файлу	путь к файлу
Требуется пользовательский экземпляр ("User Instance=True" в строке подключения)	Нет	Да
Расширение файла базы данных	MDF	MDF

Преимущества SQL Server Express LocalDB:

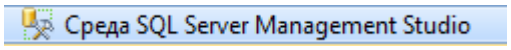
- SQL Server Express LocalDB совместима с основанными на службах выпусками SQL Server для функций, включаемых SQL Server Express LocalDB. В SQL Server можно переместить все базы данных или код Transact-SQL из SQL Server Express LocalDB в SQL Server или SQL Azure, не выполняя никаких действий по обновлению. Поэтому можно использовать SQL Server Express LocalDB для разработки приложений, ориентированных на все выпуски SQL Server.
- SQL Server Express LocalDB поддерживает те же оптимизатор запросов и средство обработки запросов, что и более высокие выпуски SQL Server.

5.1 Создание БД для кафе

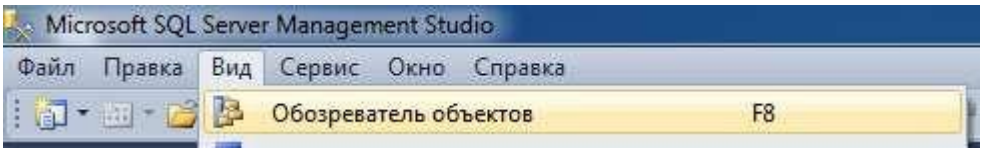
Создадим базу данных, структура и описание которой приведены в разделе 4.

Создайте для своей БД отдельную папку, например: C:\DB_Cafe. В данную папку будем в дальнейшем сохранять свою БД.

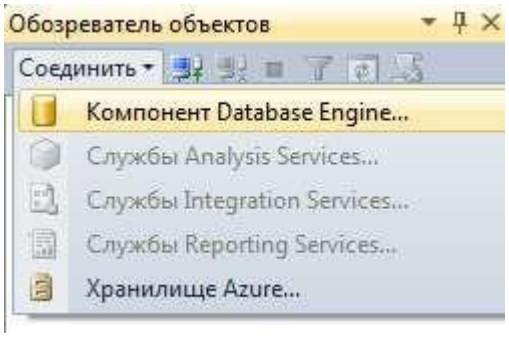
Запустите среду SQL Server Management Studio:



Выберете меню **Вид (View)**, в нем **Обозреватель объектов (Object Explorer)** или нажмите F8:



В окне **Обозреватель объектов (Object Explorer)** выберете **Компонент Database Engine**:



В диалоговом окне **Соединение с сервером** укажите:

Microsoft SQL Server предоставляет средства разделенного управления объектами сервера. Выберете необходимый тип аутентификации:

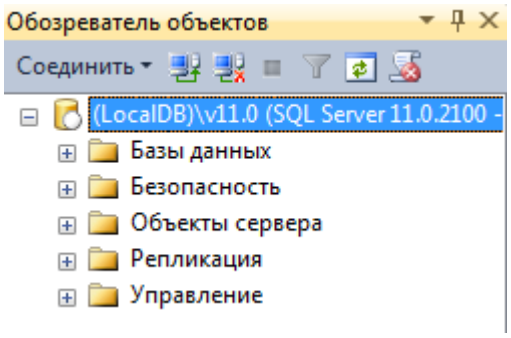
- режим аутентификации Windows (**Windows Authentication**). При установке данный режим предлагается по умолчанию. В этом случае аутентификация пользователя осуществляется операционной системой Windows. Затем SQL Server использует аутентификацию операционной системы для определения уровня доступа;

- регистрационные записи SQL Server (**SQL Server Authentication**) - учетные записи создаются и управляются в самой СУБД.

Использование для доступа к Microsoft SQL Server обоих методов называют **смешанным режимом**. Смешанный режим позволяет проводить аутентификацию пользователя как средствами операционной системы, так и с применением учетных записей Microsoft SQL Server.

Нажмите кнопку **Соединить.**

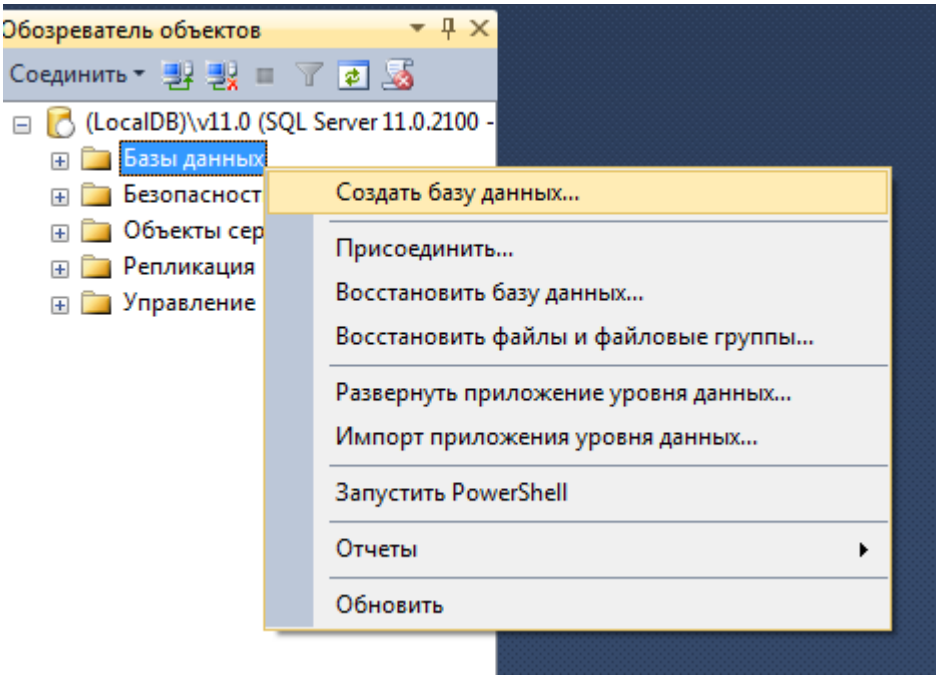
В результате получаем:



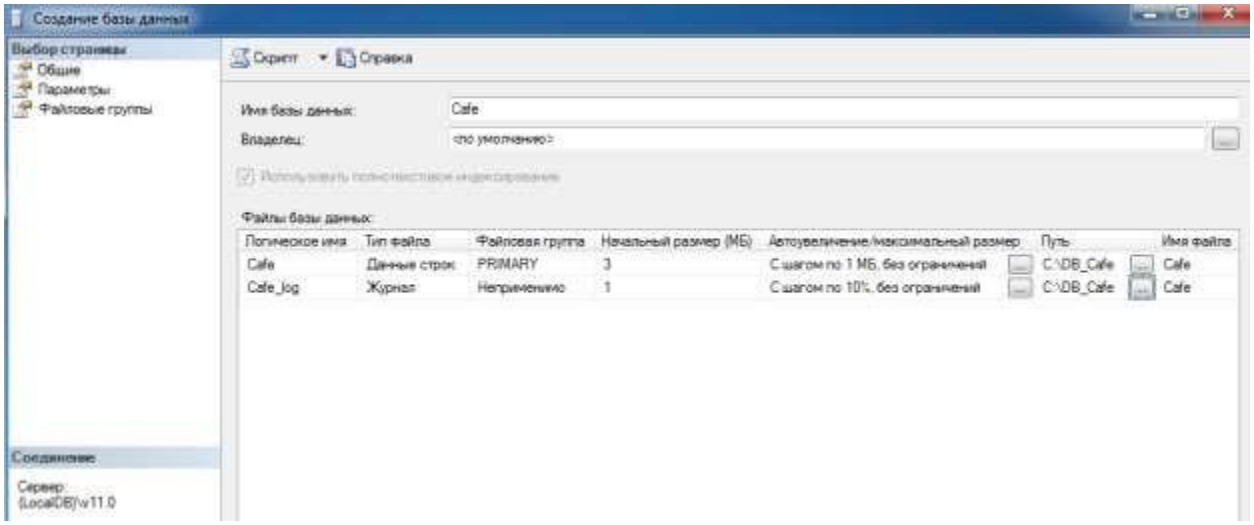
Примечание:

Предварительно у Вас должен быть доступ к SQL Server с правами на создание БД.

На узле **Базы данных** в обозревателе объектов нажмите правую кнопку мыши и выберете **Создать базу данных**:



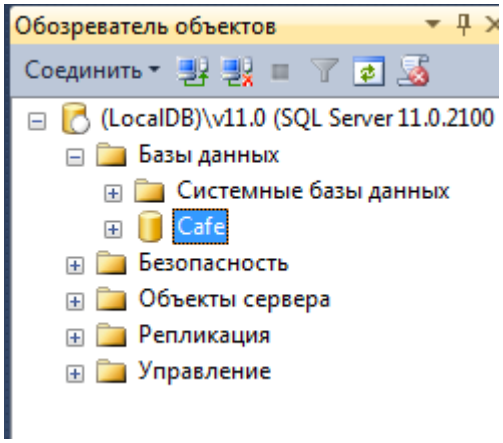
В диалоговом окне укажите для новой базы данных имя **Cafe** и необходимый путь к БД:



Нажмите ОК.

База данных состоит из двух файлов (точнее, из двух наборов файлов):
- файл данных (данные строк) с заданным именем и расширением **Cafe.mdf**;
- файл журнала транзакций с расширением **Cafe_log.ldf**.

В результате получаем:



При создании БД необходимо указать **параметры сортировки** для поддержки необходимого языка.

Параметры сортировки определяют правила, используемые при сортировке и сравнении данных.

Параметры сортировки могут учитывать регистр, диакритические знаки, тип японской азбуки, ширину символов.

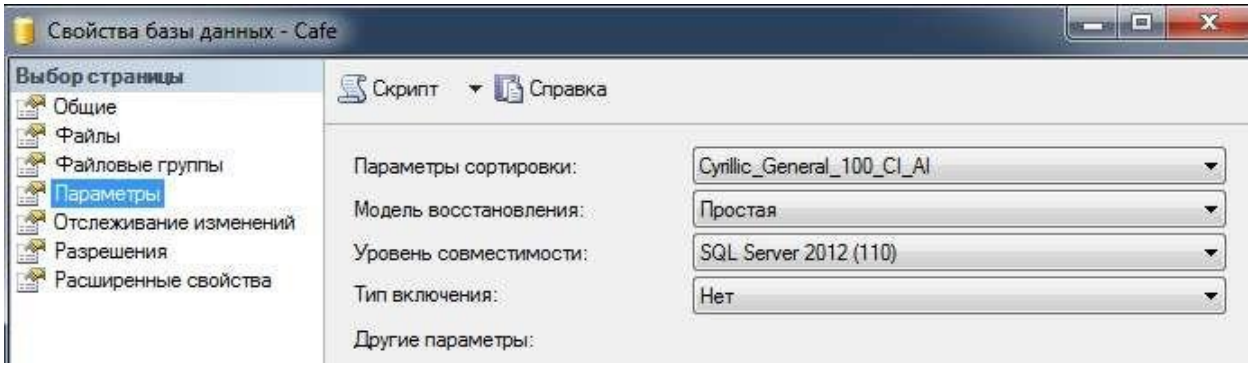
Эти параметры задаются путем их добавления к имени параметров сортировки.

Например, параметр Japanese_Bushu_Kakusu_100_CS_AS_KS_WS определяет параметры сортировки с учетом диакритических знаков, регистра, типа японской азбуки и ширины символов. В следующей таблице описывается режим работы, связанный с этими параметрами.

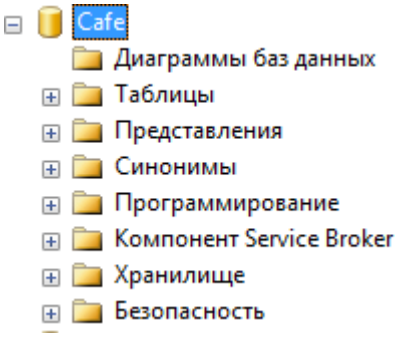
Параметр	Описание
С учетом регистра (**_CS**)	**Различаются буквы верхнего и нижнего регистров**. При выборе этого параметра буквы нижнего регистра при сортировке ставятся перед соответствующими буквами верхнего регистра. Если этот параметр не установлен, параметры сортировки не будут учитывать регистр. То есть при сортировке SQL Server считает буквы верхнего и нижнего регистров идентичными друг другу. Можно явно выбрать **нечувствительность к регистру**, указав параметр **_CI**.
С учетом диакритических знаков (**_AS**)	**Различаются символы с диакритическими знаками и без них.** Например, «a» отлично от «á». Если этот параметр не установлен, параметры сортировки не будут учитывать диакритические знаки. То есть, при сортировке SQL Server считает буквы с диакритическими знаками и без них идентичными друг другу. Можно явно выбрать **нечувствительность к диакритическим знакам**, указав параметр **_AI**.

С учетом типа японской азбуки (**_KS**)	Различаются два вида японской азбуки: Хирагана и Катакана. Если этот параметр не выбран, параметры сортировки не будут учитывать тип японской азбуки. То есть, при сортировке SQL Server рассматривает символы хирагана и катакана как идентичные. Пропуск этого параметра является единственным способом указания нечувствительности к типу японской азбуки.
С учетом ширины символов (**_WS**)	Различаются одинаковые символы, представленные одним и двумя байтами. Если этот параметр не выбран, при сортировке SQL Server рассматривает однобайтовые и двухбайтовые представления одних и тех же символов как идентичные. Пропуск этого параметра является единственным способом указания нечувствительности к ширине символов.

Например, выбор для кириллицы Cyrillic_General_100_CI_AI:



После создания БД в новой базе данных станут доступны ее объекты:



Примечание:

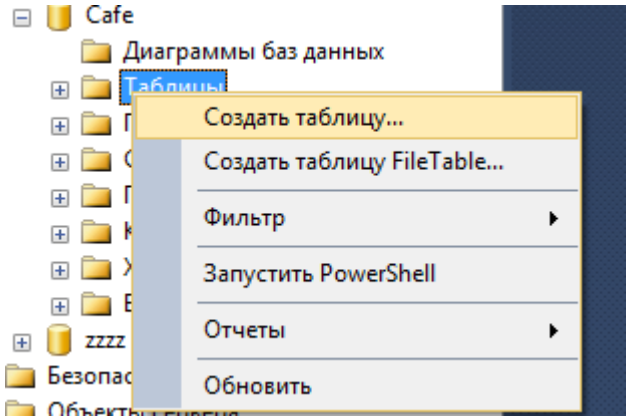
Целью данной лабораторной работы является создание таблиц базы данных, установка межтабличных связей, наложение ограничений на данные в таблицах БД. Остальные объекты БД пока не рассматриваются.

5.2 Создание таблиц в БД

Создайте таблицы «Блюда», «Сотрудники», «Продажи», «Продано» согласно описанию в разделе 4.

5.2.1 Таблица «Блюда»

В нашей БД **Cafe** выделите узел Таблицы (**Tables**), нажмите на нем правую кнопку мыши и выберете пункт для создания новой таблицы БД:



После этого откроется конструктор таблицы.

В появившемся окне **конструктора таблицы** введите соответствующие колонки:

- **Имя столбца (Column Name)** – имя соответствующего столбца;
- **Тип данных (Data Type)** – необходимый тип данных;
- **Разрешить значения Nulls (Allow Nulls)** – допустимы ли NULL значения в данном поле.

Для таблицы **Блюда** задайте в конструкторе таблицы:

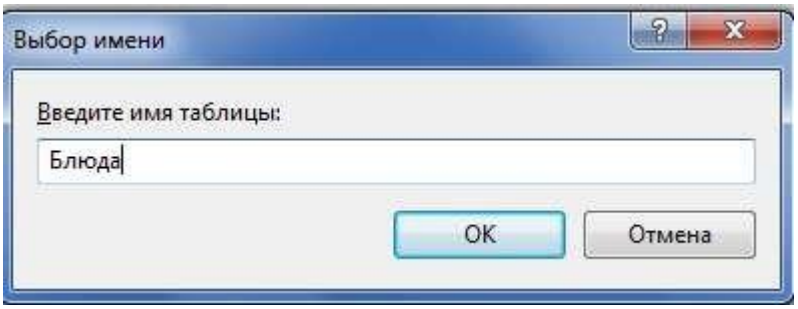
Имя столбца Column Name	**Тип данных Data Type**	**Разрешить значения Nulls Allow Nulls**
Код_блюда	int	
Название	nvarchar(40)	
Тип	nvarchar(30)	
Цена_меню	money	
Выход	int	Флаг установлен
Единицы	nvarchar(5)	Флаг установлен
Фирменное_ блюдо	bit	Флаг установлен
Состав	nvarchar(MAX)	Флаг установлен
Фото	varbinary(MAX)	Флаг установлен

Сохраните таблицу с именем **Блюда**.

Для этого из меню Файл (**File**) выберете **Сохранить имя таблицы** (**Save table name**). В диалоговом окне введите имя таблицы и нажмите кнопку **ОК**.

Или нажмите кнопку **Save Table** либо на панели инструментов, либо вызовом нажатия правой кнопки мыши в конструкторе таблицы.

В появившемся диалоговом окне введите имя таблицы. В данном случае **Блюда:**

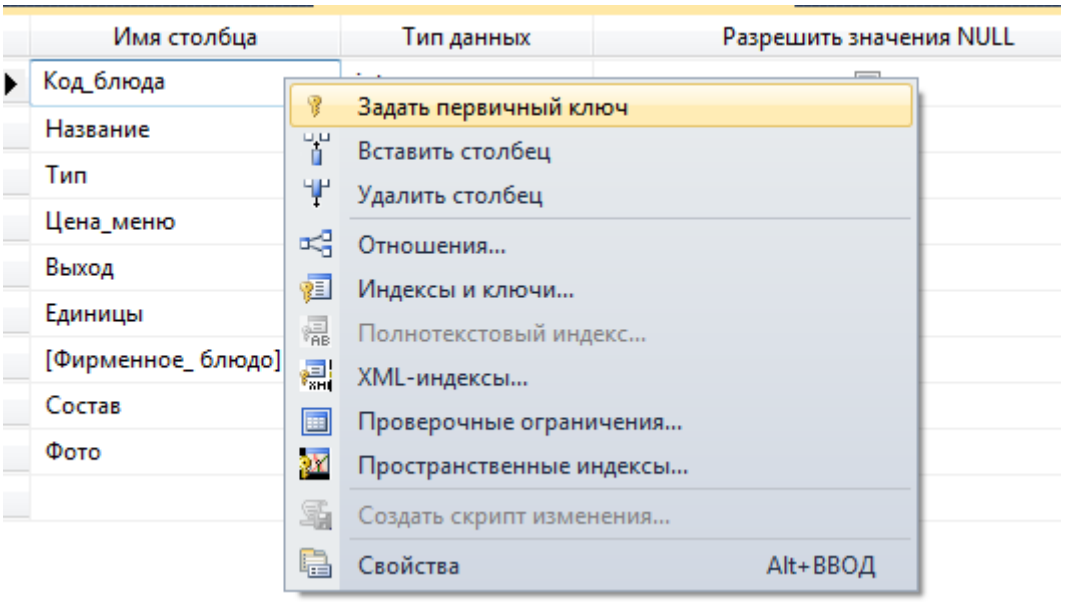


Теперь зададим необходимые ограничения для соответствующих полей.

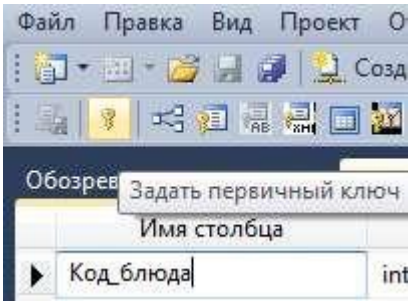
5.2.1.1 Поле Код_блюда

5.2.1.1.1 Установка первичного ключа для поля Код_блюда

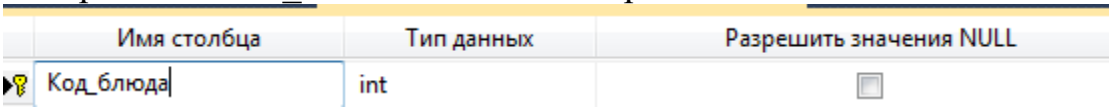
Выделите строку для поля Код_блюда, нажмите на ней правую кнопку мыши и выберете **Задать первичный ключ (Set Primary Key)**:



Или нажмите соответствующую кнопку на панели инструментов:



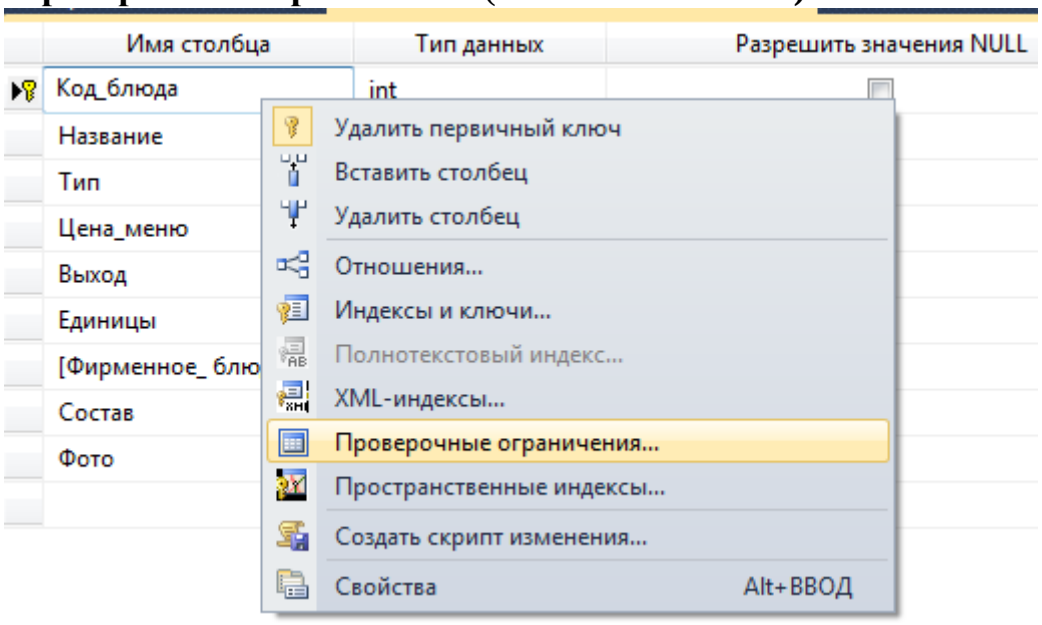
Теперь возле Код_блюда стоит знак первичного ключа:

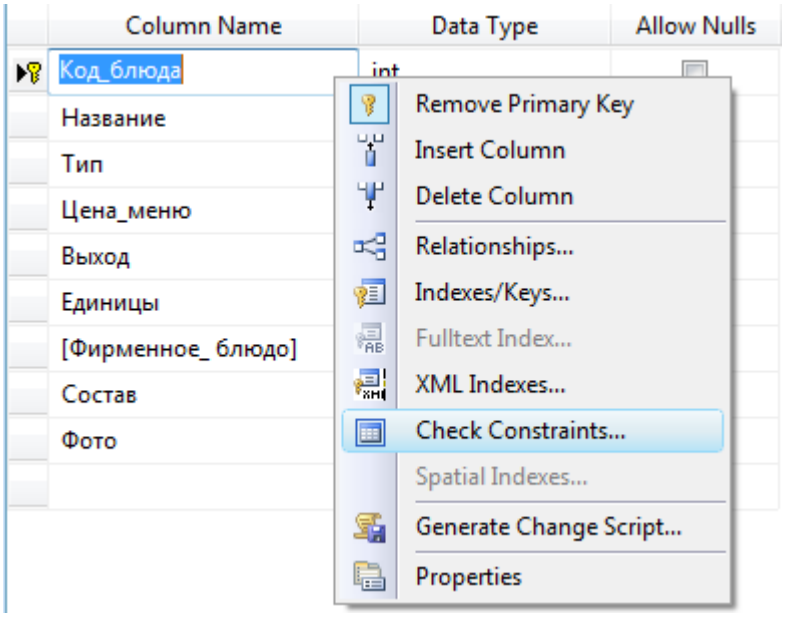


NULL значения для данного поля не допустимы.

5.2.1.1.2 Условие на значение Код_блюда >=0

Нажмите правую кнопку мыши в конструкторе таблицы и выберете пункт **Проверочные ограничения (Check Constraints)**:

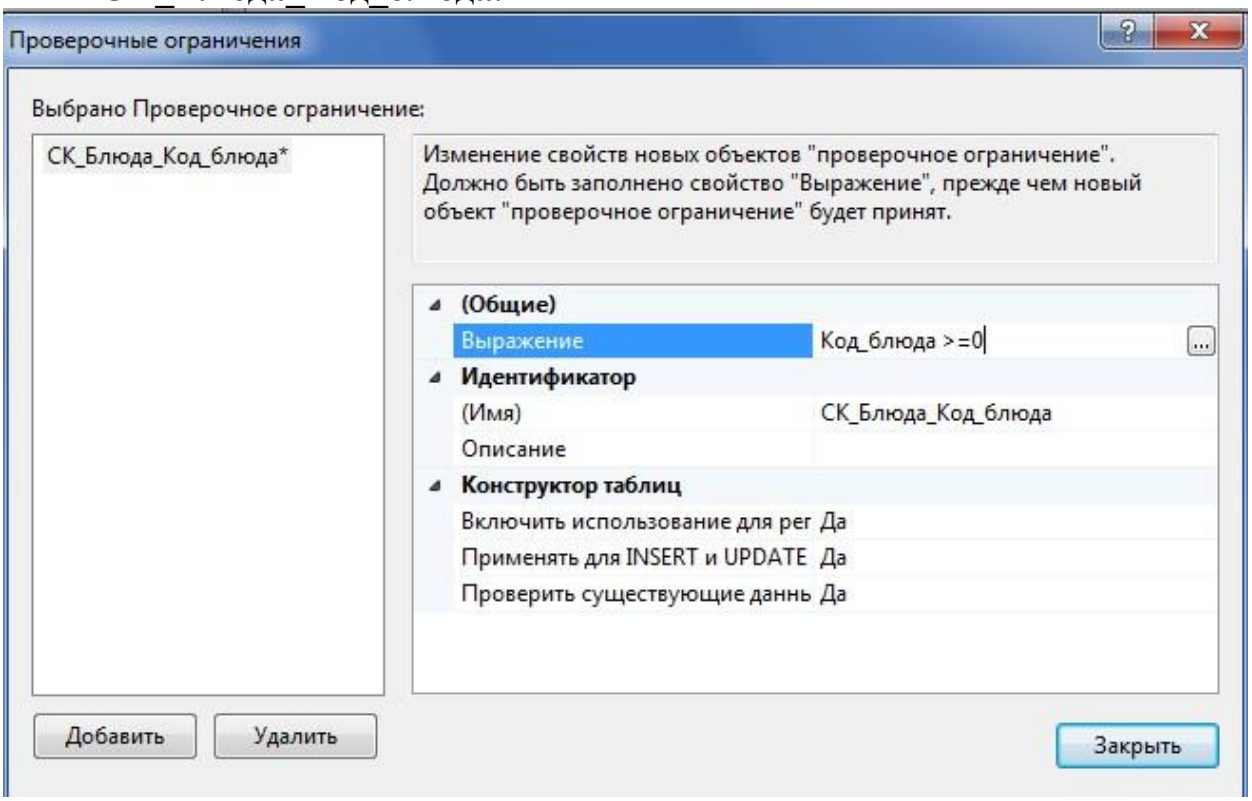


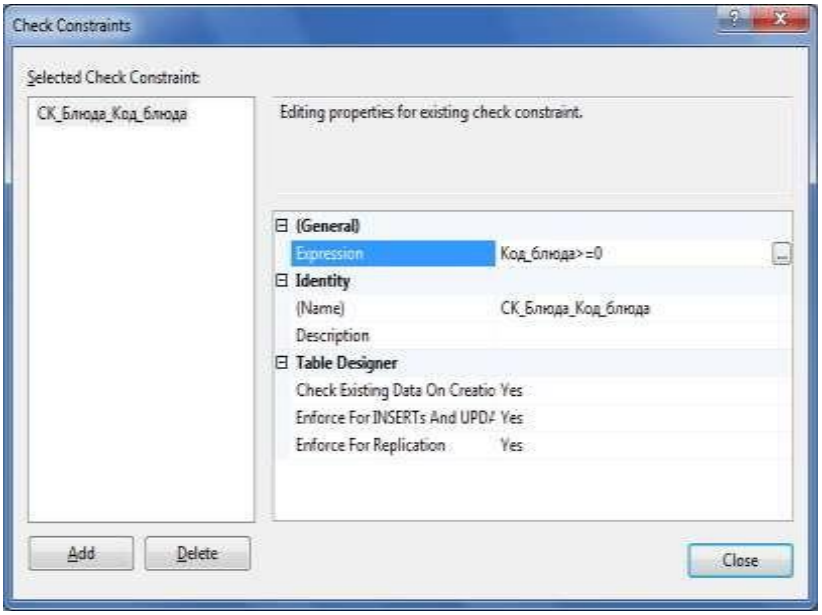


В появившемся диалоговом окне нажмите кнопку **Добавить (Add)**.

Затем в строке **Выражение (Expression)** укажите условие на значения для **Код_блюда >=0**.

Назовите ограничение в свойстве **Имя (Name)** в **СК_Блюда_Код_блюда.**





Нажмите **Закрыть (Close)** и перейдите к конструктору таблицы **Блюда**.

5.2.1.2 Поле Название

Размер поля **текстовый до 40 символов**. **Обязательное поле** (запрет NULL значений):

Имя столбца	Тип данных	Разрешить значения NULL
Код_блюда	int	☐
Название	nvarchar(40)	☐

или

Column Name	Data Type	Allow Nulls
Код_блюда	int	☐
Название	nvarchar(40)	☐

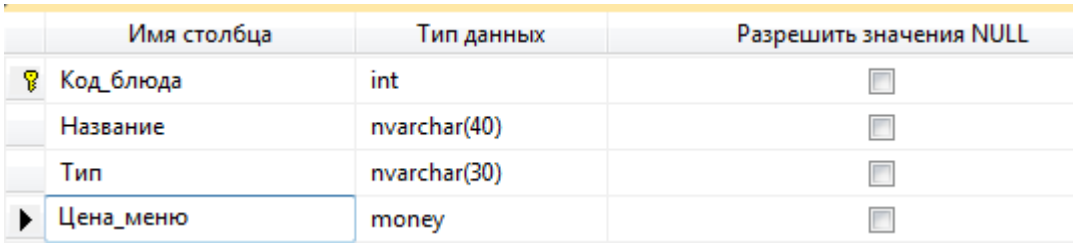
5.2.1.3 Поле Тип

Размер поля **текстовый до 30 символов**. **Обязательное поле** (запрет NULL значений):

Имя столбца	Тип данных	Разрешить значения NULL
Код_блюда	int	☐
Название	nvarchar(40)	☐
Тип	nvarchar(30)	☐

5.2.1.4 Поле Цена_меню

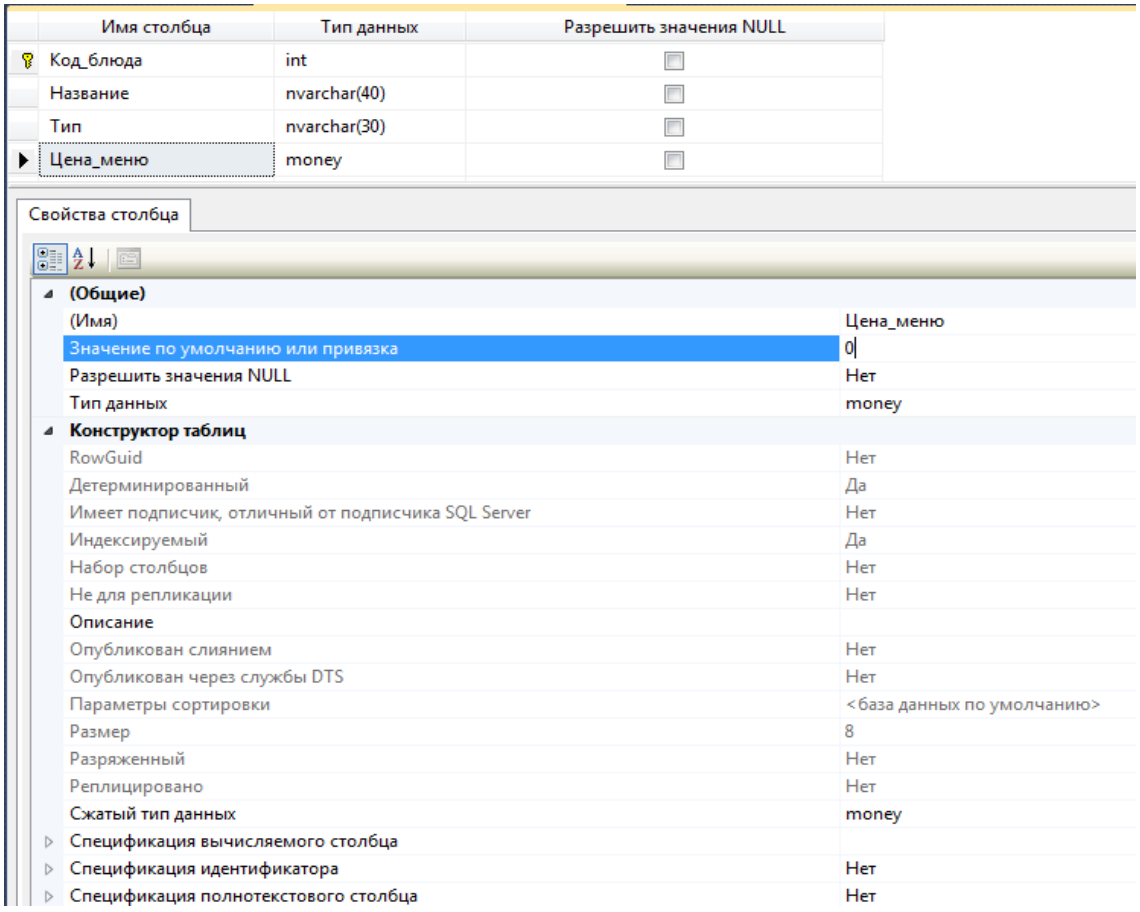
5.2.1.4.1 Тип данных – денежный. Обязательное поле (запрет NULL значений):



	Имя столбца	Тип данных	Разрешить значения NULL
🔑	Код_блюда	int	☐
	Название	nvarchar(40)	☐
	Тип	nvarchar(30)	☐
▶	Цена_меню	money	☐

5.2.1.4.2 Значение по умолчанию 0

Перейдите в окно свойств поля **Свойства столбца (Column Properties)** для поля Цена_меню. В строке **Значения по умолчанию или привязка (Default Value or Binding)** укажите значение 0:

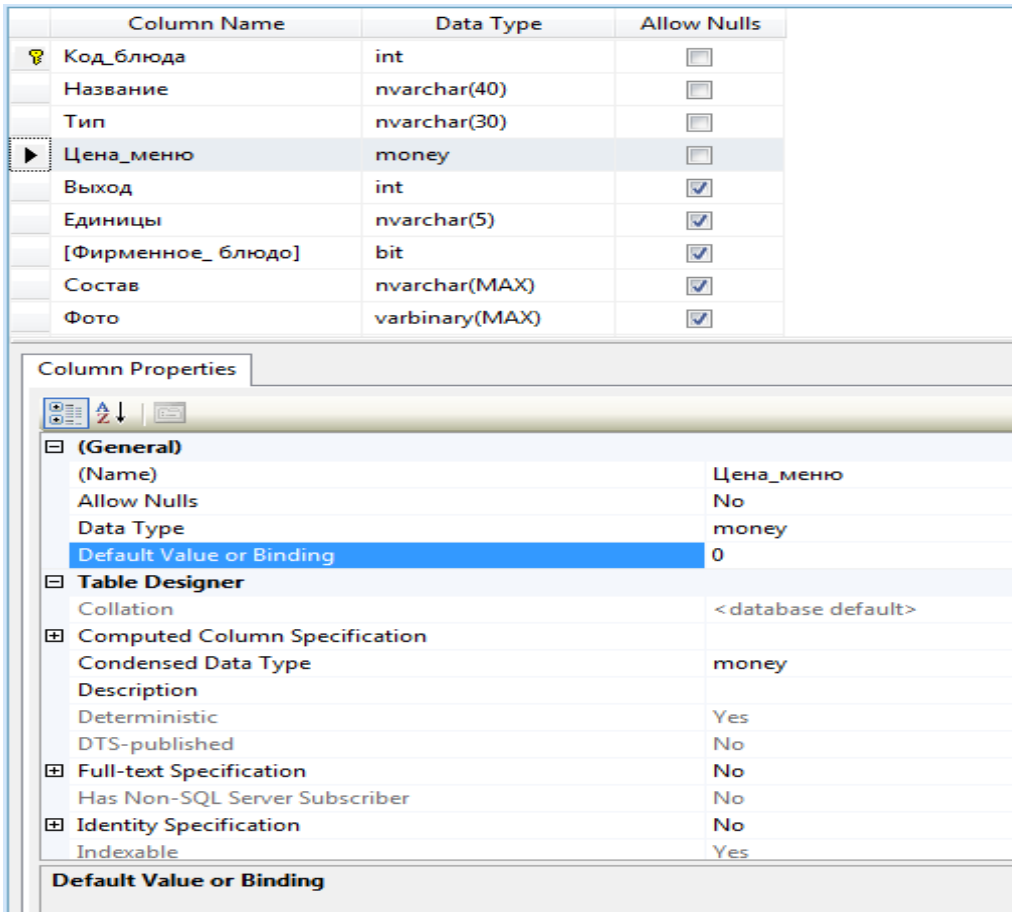


	Имя столбца	Тип данных	Разрешить значения NULL
🔑	Код_блюда	int	☐
	Название	nvarchar(40)	☐
	Тип	nvarchar(30)	☐
▶	Цена_меню	money	☐

Свойства столбца

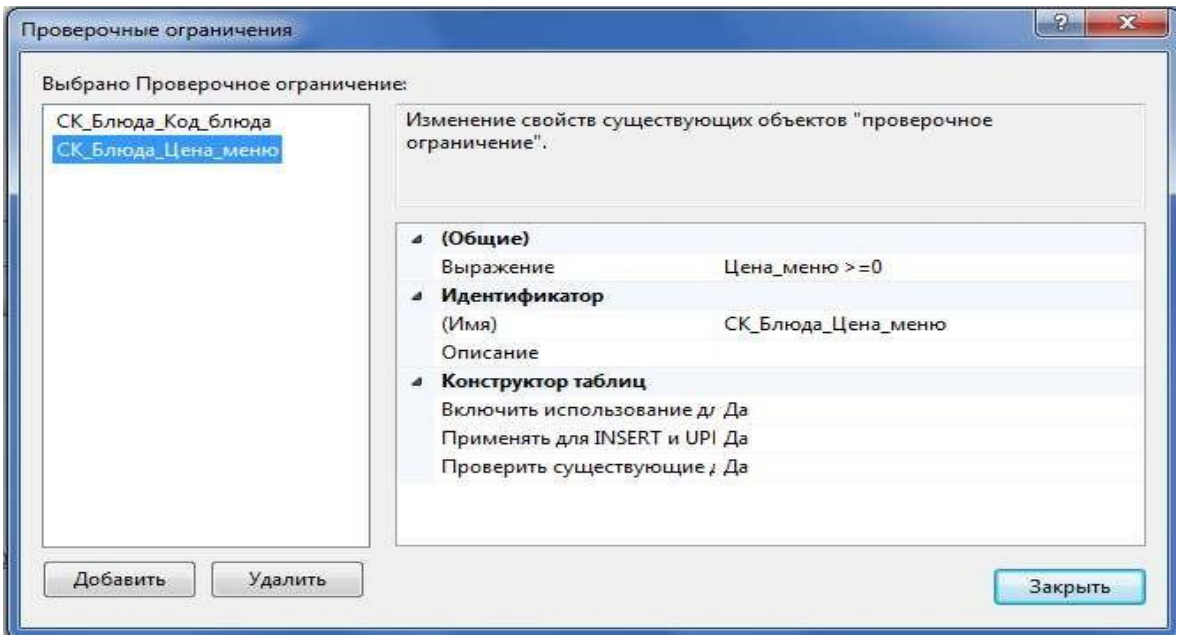
(Общие)	
(Имя)	Цена_меню
Значение по умолчанию или привязка	0
Разрешить значения NULL	Нет
Тип данных	money
Конструктор таблиц	
RowGuid	Нет
Детерминированный	Да
Имеет подписчик, отличный от подписчика SQL Server	Нет
Индексируемый	Да
Набор столбцов	Нет
Не для репликации	Нет
Описание	
Опубликован слиянием	Нет
Опубликован через службы DTS	Нет
Параметры сортировки	<база данных по умолчанию>
Размер	8
Разряженный	Нет
Реплицировано	Нет
Сжатый тип данных	money
Спецификация вычисляемого столбца	
Спецификация идентификатора	Нет
Спецификация полнотекстового столбца	Нет

или

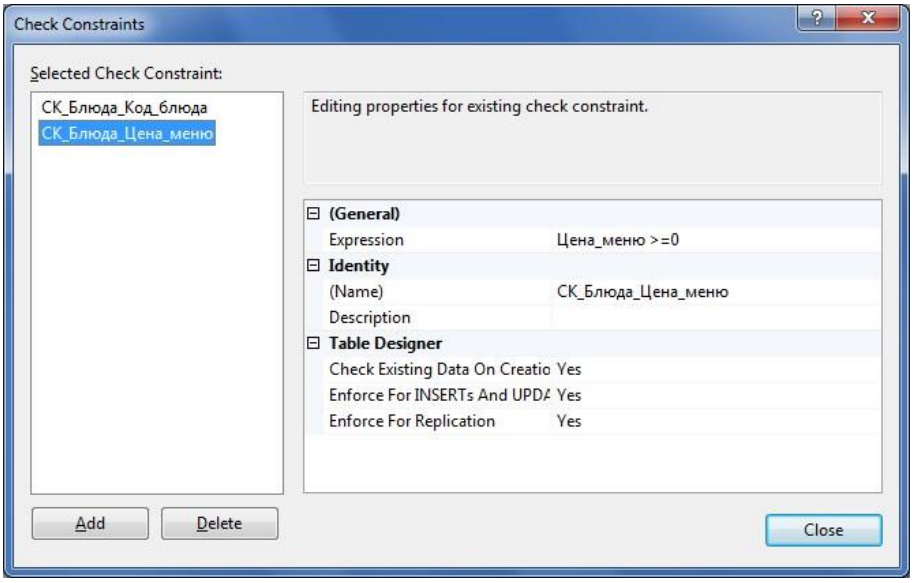


5.2.1.4.3 Условие на значение Цена_меню >=0

Аналогично как для Код_блюда (раздел 5.2.1.1.2).
Назовите ограничение **CK_Блюда_Цена_меню**.



или



5.2.1.5 Поле Выход

5.2.1.5.1 Тип данных – **целые числа, NULL значения допустимы**:

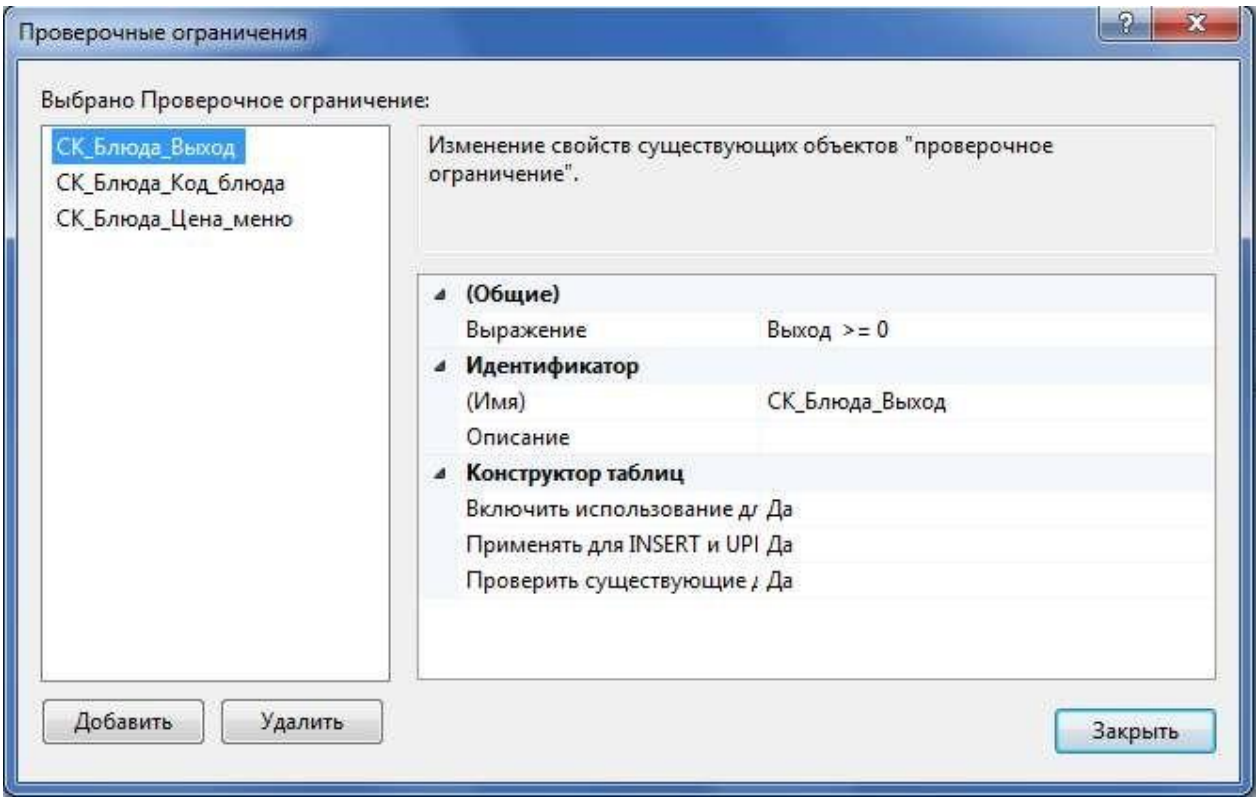
	Имя столбца	Тип данных	Разрешить значения NULL
🔑	Код_блюда	int	☐
	Название	nvarchar(40)	☐
	Тип	nvarchar(30)	☐
	Цена_меню	money	☐
▶	Выход	int	☑

5.2.1.5.2 Значение по умолчанию 0

Аналогично как для поля Цена_меню (раздел 5.2.1.4.2).

5.2.1.5.3 Условие на значение Выход >= 0:

Аналогично как для Код_блюда (раздел 5.2.1.1.2).
Назовите ограничение **CK_Блюда_Выход**.



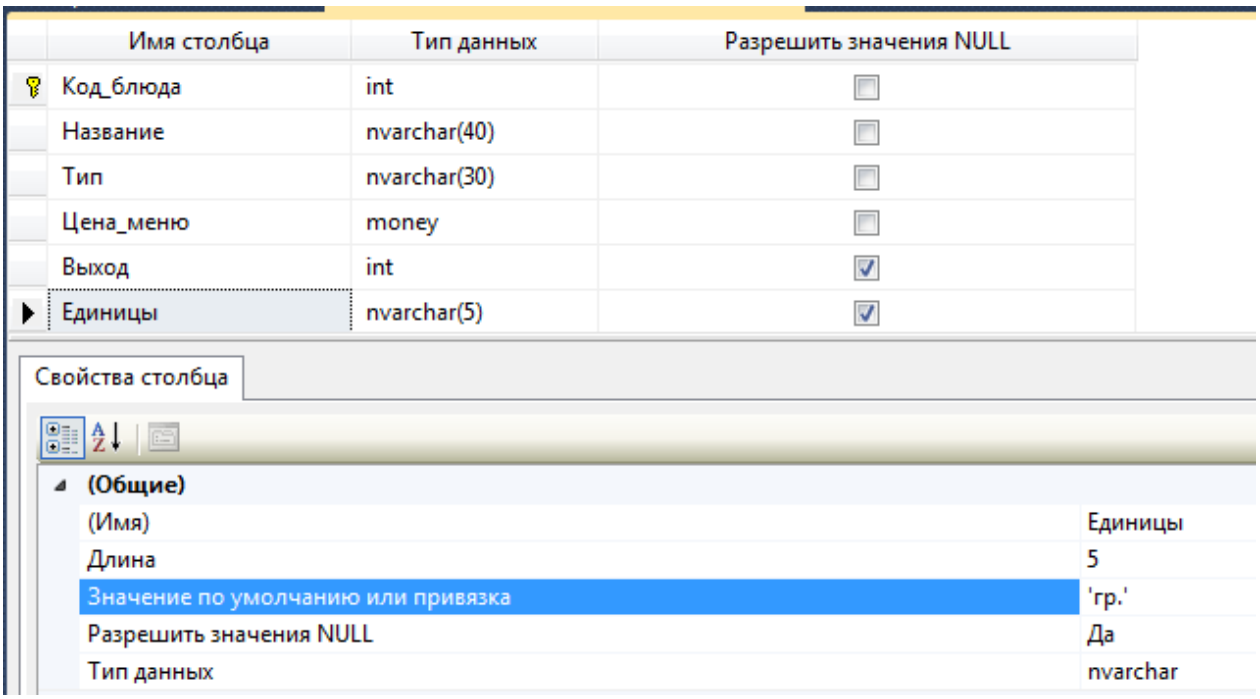
5.2.1.6 Поле Единицы

5.2.1.6.1 Размер поля **текстовый до 5 символов, NULL значения допустимы:**

Имя столбца	Тип данных	Разрешить значения NULL
Код_блюда	int	☐
Название	nvarchar(40)	☐
Тип	nvarchar(30)	☐
Цена_меню	money	☐
Выход	int	☑
Единицы	nvarchar(5)	☑

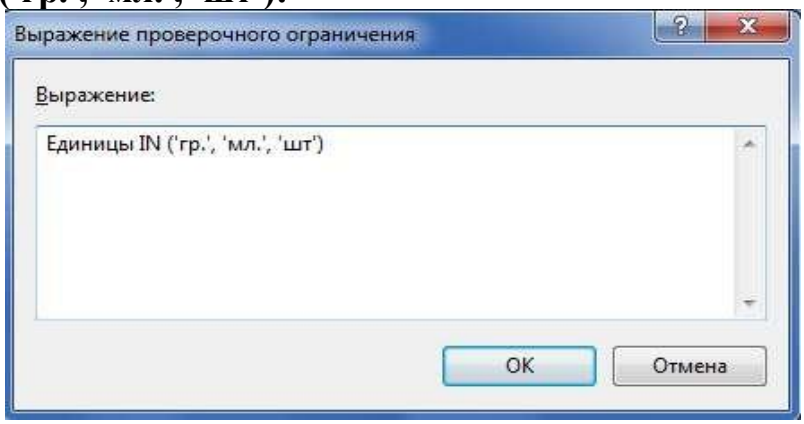
5.2.1.6.2 Значение по умолчанию ‘гр.’

Аналогично как для поля Цена_меню (раздел 5.2.1.4.2) только укажите значение 'гр.'

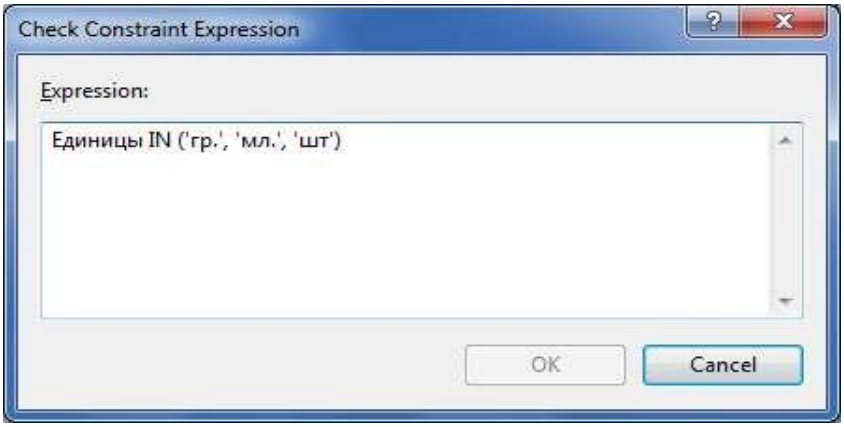


5.2.1.6.3 Допустимы значения ‘гр.’ или ‘мл.’ или ‘шт.’:

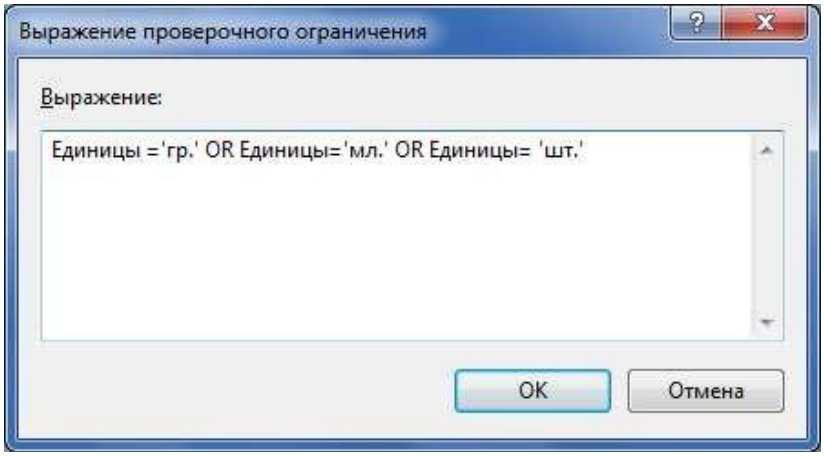
В строке **Выражение (Expression)** укажите условие на значения **Единицы IN ('гр.', 'мл.', 'шт'):**



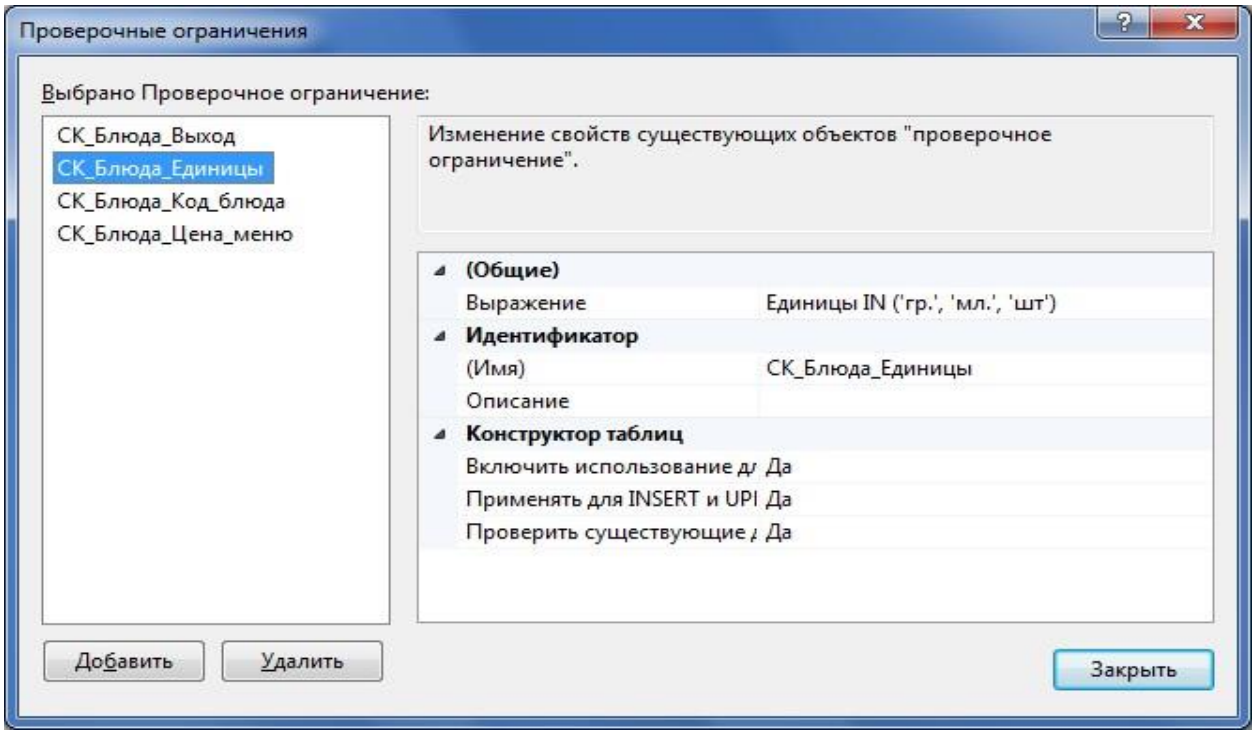
или



Или **Единицы ='гр.' OR Единицы='мл.' OR Единицы= 'шт.'**



Назовите ограничение в свойстве **Имя (Name)** в **СК_Блюда_Единицы.**



5.2.1.7 Поле Фирменное_блюдо

Выбран тип данных **bit - целочисленный тип данных,** который может принимать значения 1, 0 или NULL.

Имя столбца	Тип данных	Разрешить значения NULL
Код_блюда	int	☐
Название	nvarchar(40)	☐
Тип	nvarchar(30)	☐
Цена_меню	money	☐
Выход	int	☑
Единицы	nvarchar(5)	☑
Фирменное_ блюдо	bit	☑

5.2.1.8 Поле Состав

Был выбран тип данных:

nvarchar (max) - символьные данные в Юникоде переменной длины. Аргумент **max** указывает, что максимальный размер хранилища равен 2^31-1 байт.

Имя столбца	Тип данных	Разрешить значения NULL
Код_блюда	int	☐
Название	nvarchar(40)	☐
Тип	nvarchar(30)	☐
Цена_меню	money	☐
Выход	int	☑
Единицы	nvarchar(5)	☑
[Фирменное_ блюдо]	bit	☑
Состав	nvarchar(MAX)	☑

5.2.1.9 Поле Фото

Был выбран тип данных:

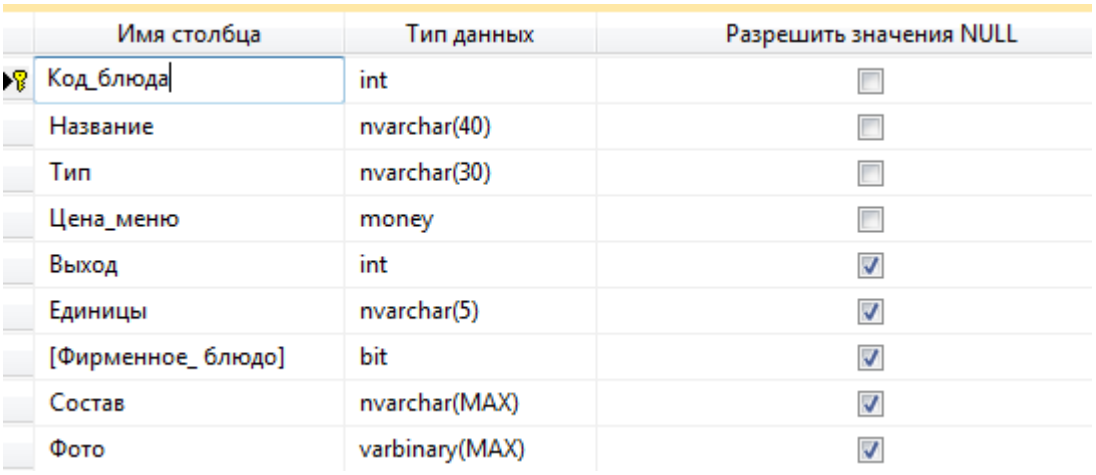
varbinary(max) - двоичные данные переменной длины. **max** означает максимальную длину хранения, которая составляет 2^31-1 байт.

Имя столбца	Тип данных	Разрешить значения NULL
Код_блюда	int	☐
Название	nvarchar(40)	☐
Тип	nvarchar(30)	☐
Цена_меню	money	☐
Выход	int	☑
Единицы	nvarchar(5)	☑
[Фирменное_ блюдо]	bit	☑
Состав	nvarchar(MAX)	☑
Фото	varbinary(MAX)	☑

5.2.1.10 Сохраните таблицу **Блюда**.

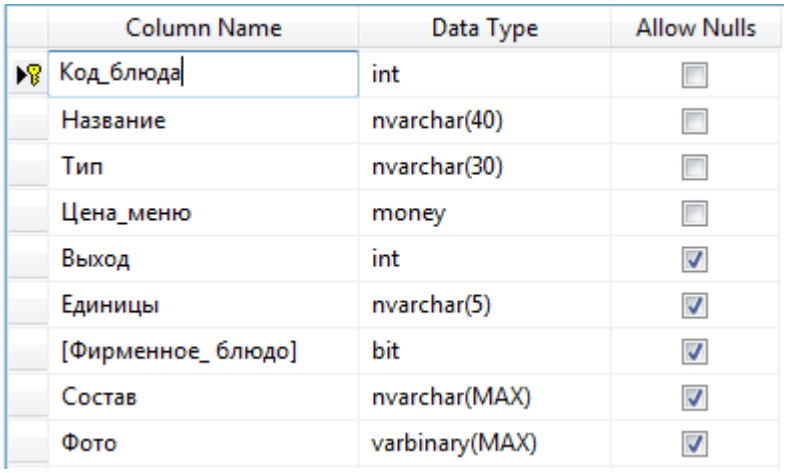
После указания ограничений на поля, сохраните таблицу Блюда.

5.2.1.11 Конструктор таблицы **Блюда** имеет вид:



Имя столбца	Тип данных	Разрешить значения NULL
Код_блюда	int	☐
Название	nvarchar(40)	☐
Тип	nvarchar(30)	☐
Цена_меню	money	☐
Выход	int	☑
Единицы	nvarchar(5)	☑
[Фирменное_ блюдо]	bit	☑
Состав	nvarchar(MAX)	☑
Фото	varbinary(MAX)	☑

или



Column Name	Data Type	Allow Nulls
Код_блюда	int	☐
Название	nvarchar(40)	☐
Тип	nvarchar(30)	☐
Цена_меню	money	☐
Выход	int	☑
Единицы	nvarchar(5)	☑
[Фирменное_ блюдо]	bit	☑
Состав	nvarchar(MAX)	☑
Фото	varbinary(MAX)	☑

5.2.2 Таблица «Сотрудники»

Проведите создание таблицы **Сотрудники** аналогичным образом с необходимыми ограничениями.

Для таблицы **Сотрудники** задайте в конструкторе таблицы:

Имя столбца Column Name	**Тип данных Data Type**	**Разрешить значения Nulls Allow Nulls**
Код_сотрудника	int	
Фамилия	nvarchar(30)	
Имя	nvarchar(30)	
Отчество	nvarchar(30)	Флаг установлен
Должность	nvarchar(30)	
Пол	nvarchar(10)	Флаг установлен
Семейное_положение	nvarchar(15)	Флаг установлен
Дети	bit	Флаг установлен

Теперь зададим необходимые ограничения для соответствующих полей.

5.2.2.1 Поле Код_сотрудника

5.2.2.1.1 Установка первичного ключа

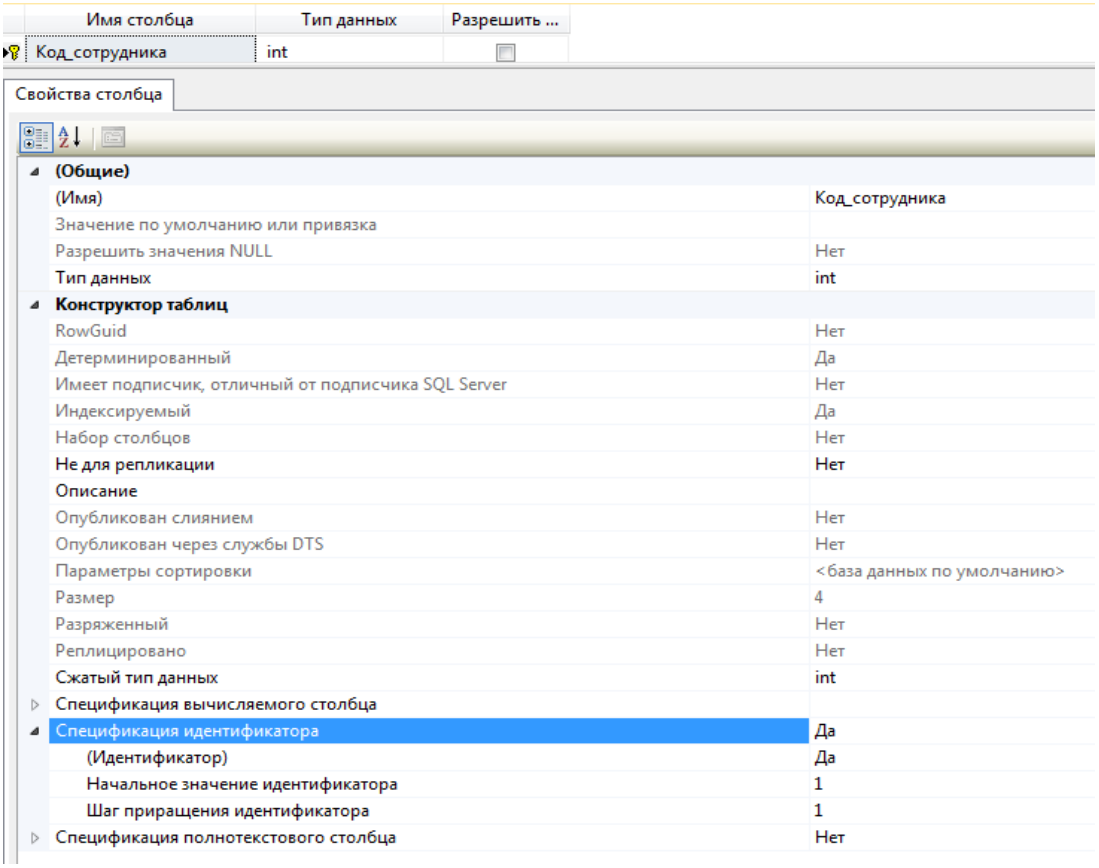
Данное поле является первичным ключом. Установите первичный ключ по аналогии с разделом 5.2.1.1.1.

5.2.2.1.2 Тип данных счетчик

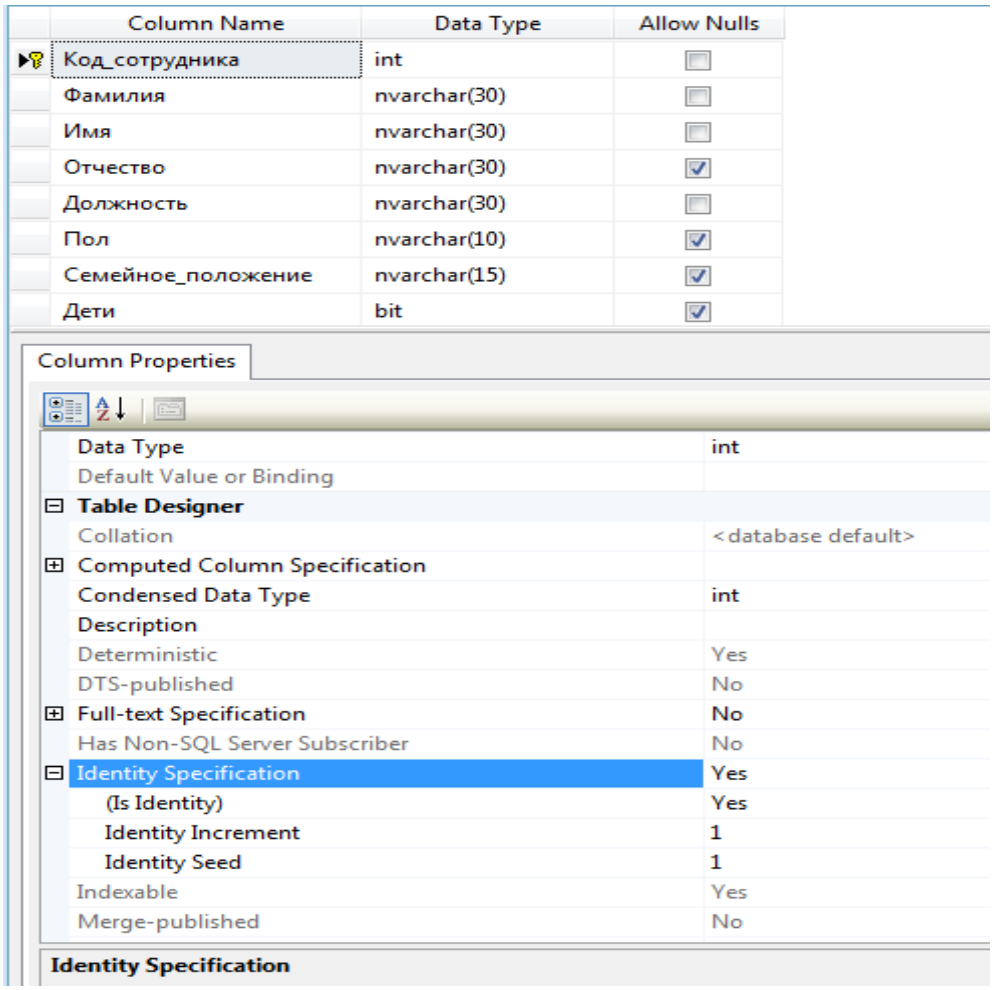
Тип данных для данного поля решено установить в счетчик (автоинкремент). Значения в данном поле будут указываться, начиная с 1 с шагом 1. Данное поле редактировать нельзя.

Для того, чтобы указать данный тип данных счетчик для Код_сотрудника необходимо выделить поле Код_сотрудника, перейти в свойства для данного поля и указать в **Спецификация идентификатора (Identity Specification)**:

- свойство **Идентификатор (Is Identity)** в **Да (Yes);**
- свойство **Начальное значение идентификатора** в **1;**
- свойство **Шаг приращения идентификатора** в **1.**

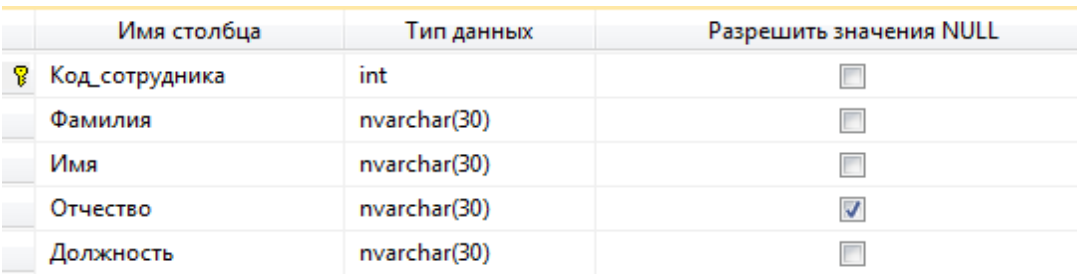


или



5.2.2.2 Поля Фамилия, Имя, Отчество, Должность

Размер поля **текстовый до 30 символов**. Поля Фамилия, Имя, Должность - **обязательные поля** (запрет NULL значений), в поле Отчество NULL значения допустимы:



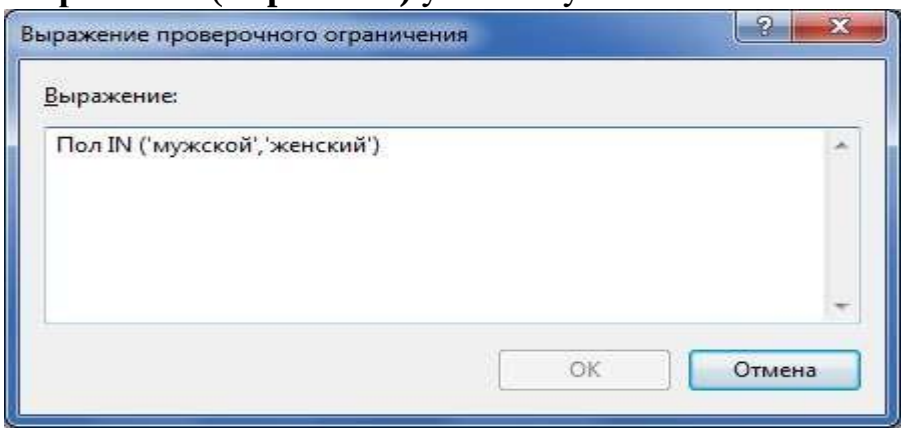
5.2.2.3 Поле Пол

Размер поля **текстовый до 10 символов**. **NULL значения допустимы**:

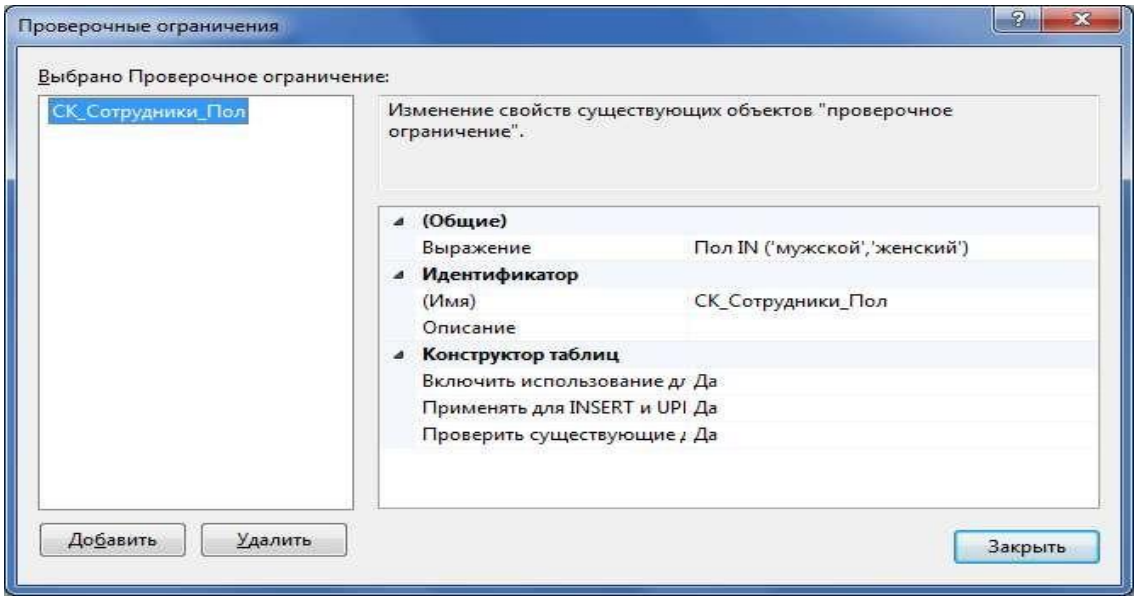
Имя столбца	Тип данных	Разрешить значения NULL
Код_сотрудника	int	☐
Фамилия	nvarchar(30)	☐
Имя	nvarchar(30)	☐
Отчество	nvarchar(30)	☑
Должность	nvarchar(30)	☐
Пол	nvarchar(10)	☑

Условие на значение: ‘мужской’ OR ‘женский’:

Для этого в **Проверочные ограничения (Check Constraints)** в строке **Выражение (Expression)** укажите условие:



Назовите ограничение в свойстве **Имя (Name)** в **CK_Сотрудники_Пол.**



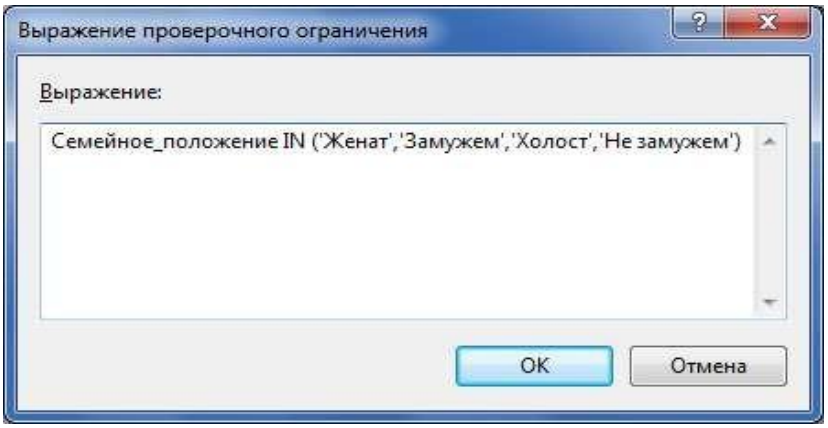
5.2.2.4 Поле Семейное_положение

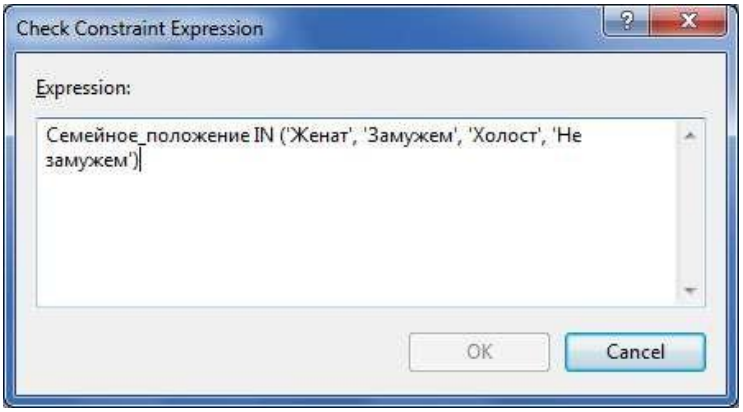
Размер поля **текстовый до 15 символов. NULL значения допустимы**:

	Имя столбца	Тип данных	Разрешить значения NULL
🔑	Код_сотрудника	int	☐
	Фамилия	nvarchar(30)	☐
	Имя	nvarchar(30)	☐
	Отчество	nvarchar(30)	☑
	Должность	nvarchar(30)	☐
	Пол	nvarchar(10)	☑
▶	Семейное_положение	nvarchar(15)	☑

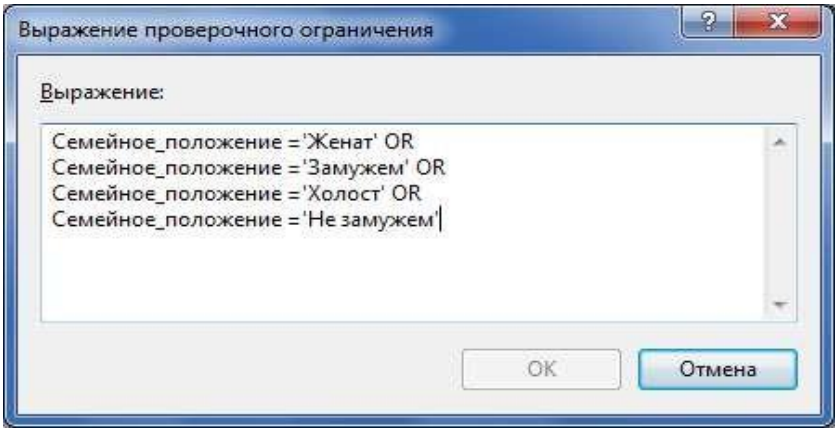
Условие на значение:

Семейное_положение IN ('Женат','Замужем','Холост','Не замужем')

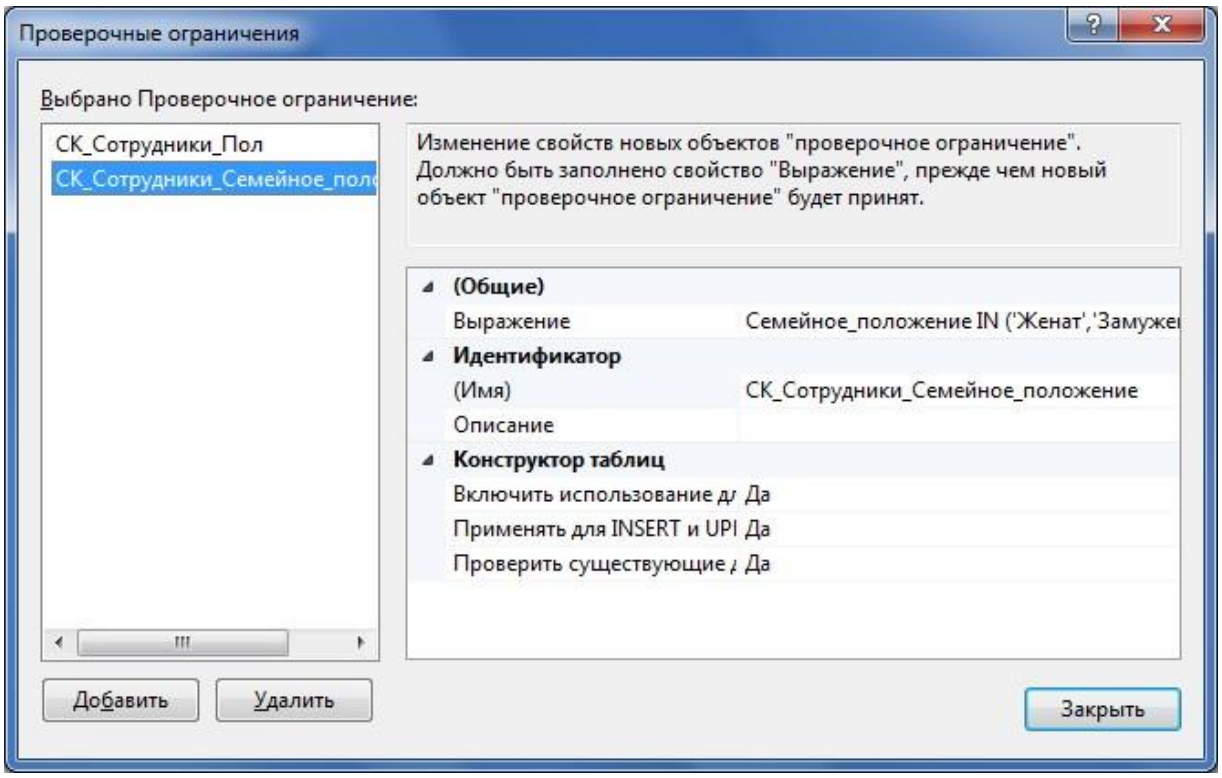




или



Назовите ограничение в свойстве **Имя (Name)** в **CK_Сотрудники_Семейное_положение**.



5.2.2.5 Поле Дети

Выбран тип данных **bit - целочисленный тип данных**, который может принимать значения 1, 0 или NULL.

NULL значения для данного поля допустимы:

	Имя столбца	Тип данных	Разрешить значения NULL
🔑	Код_сотрудника	int	☐
	Фамилия	nvarchar(30)	☐
	Имя	nvarchar(30)	☐
	Отчество	nvarchar(30)	☑
	Должность	nvarchar(30)	☐
	Пол	nvarchar(10)	☑
	Семейное_положение	nvarchar(15)	☑
▶	Дети	bit	☑

5.2.2.6 Сохраните таблицу с именем **Сотрудники**.

5.2.2.7 Конструктор таблицы **Сотрудники** имеет вид:

	Имя столбца	Тип данных	Разрешить значения NULL
🔑	Код_сотрудника	int	☐
	Фамилия	nvarchar(30)	☐
	Имя	nvarchar(30)	☐
	Отчество	nvarchar(30)	☑
	Должность	nvarchar(30)	☐
	Пол	nvarchar(10)	☑
	Семейное_положение	nvarchar(15)	☑
	Дети	bit	☑

или

	Column Name	Data Type	Allow Nulls
🔑	Код_сотрудника	int	☐
	Фамилия	nvarchar(30)	☐
	Имя	nvarchar(30)	☐
	Отчество	nvarchar(30)	☑
	Должность	nvarchar(30)	☐
	Пол	nvarchar(10)	☑
	Семейное_положение	nvarchar(15)	☑
	Дети	bit	☑

5.2.3 Таблица «Продажи»

Проведите создание таблицы **Продажи** аналогичным образом с необходимыми ограничениями.

Для таблицы **Продажи** задайте в конструкторе таблицы:

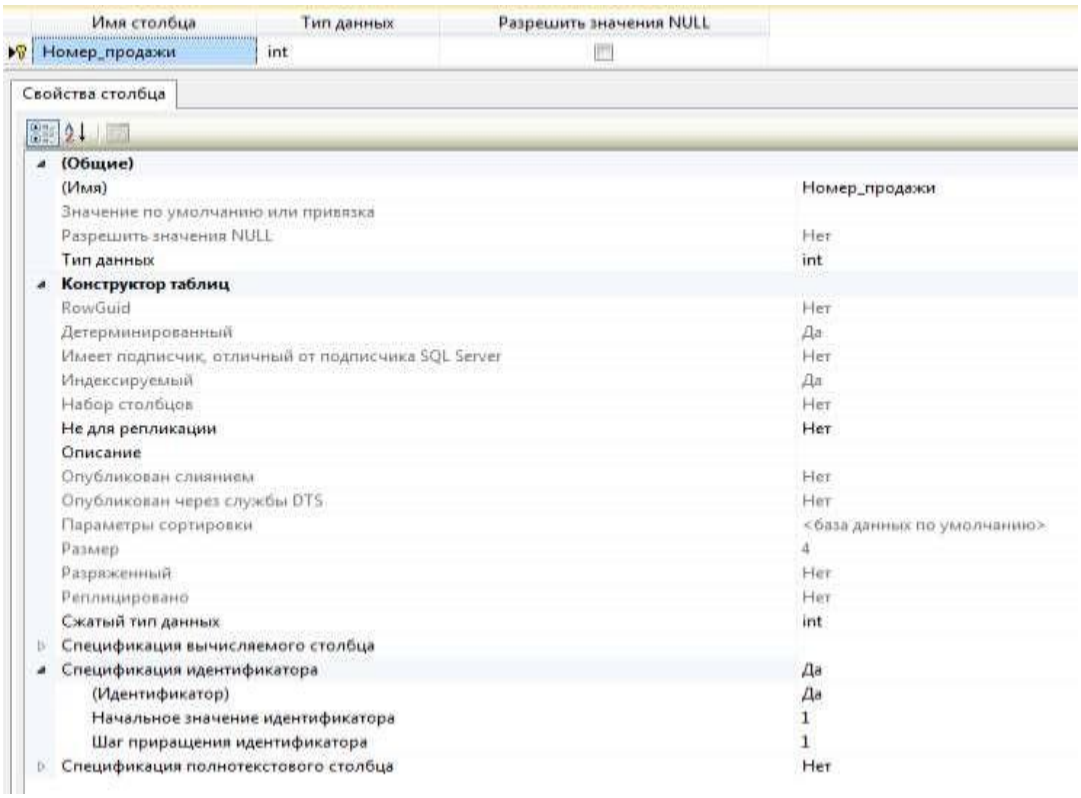
Имя столбца Column Name	Тип данных Data Type	Разрешить значения Nulls Allow Nulls
Номер_продажи	int	
Дата	smalldatetime	Флаг установлен
Код_сотрудника	int	
Номер_столика	nvarchar(10)	Флаг установлен

Теперь зададим необходимые ограничения для соответствующих полей.

5.2.3.1 Поле Номер_продажи

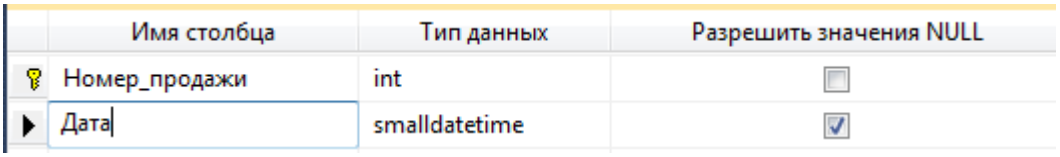
Данное поле **является первичным ключом**. Установите первичный ключ для поля Номер_продажи по аналогии с разделом 5.2.1.1.1.

Тип данных для данного поля решено установить в **счетчик (автоинкремент)**. Установите для Номер_продажи тип счетчик по аналогии с разделом 5.2.2.1.2.



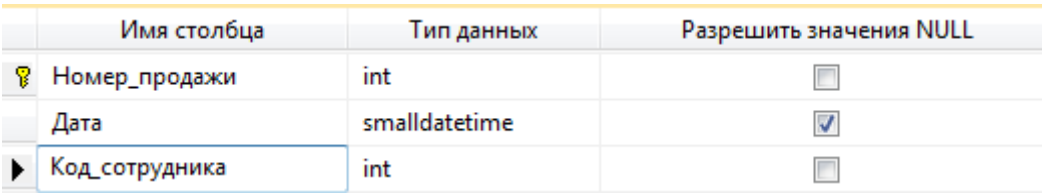
5.2.3.2 Поле дата

Выбран тип данных **краткий формат даты, NULL значения допустимы**:



5.2.3.3 Поле Код_сотрудника

5.2.3.3.1 Тип данных – **целые числа, NULL значения не допустимы**:

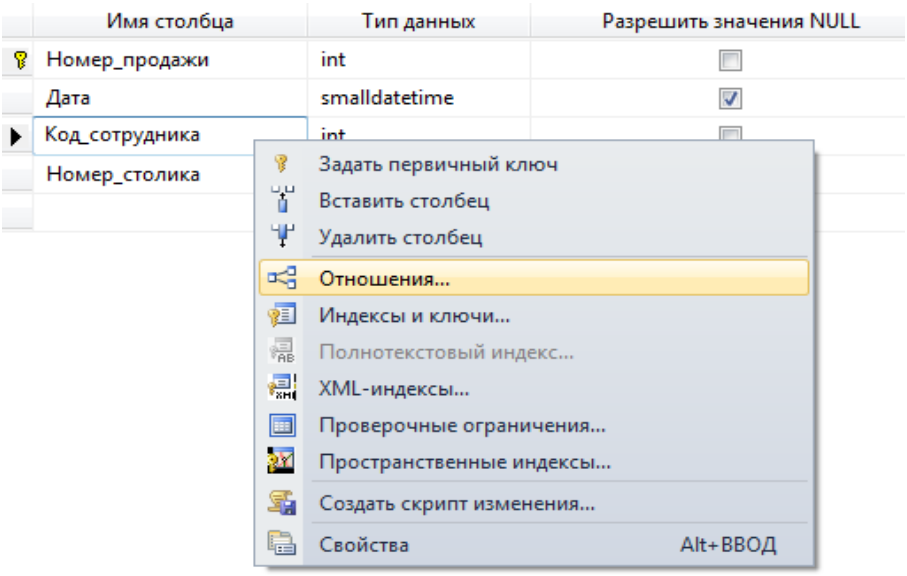


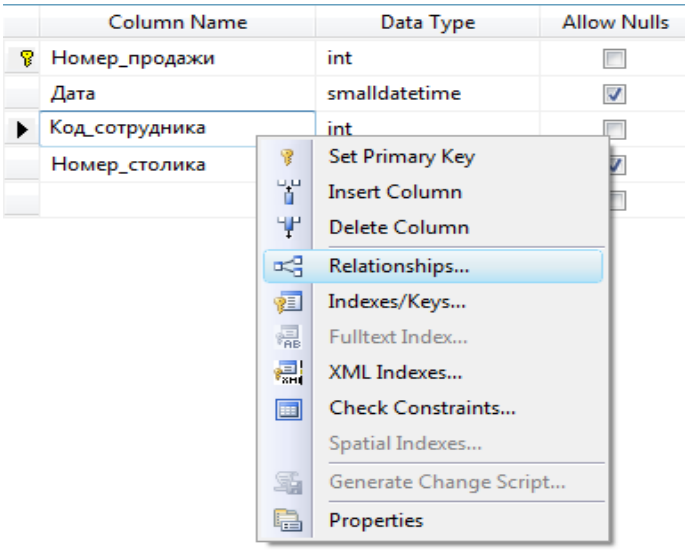
5.2.3.3.2 Внешний ключ

Данное поле является полем внешнего ключа. Родительской таблицей является таблица **Сотрудники**. Установить связь между таблицами **Сотрудники** и **Продажи** можно различными способами.

Создадим в данном случае связь через конструктор таблицы **Продажи**.

В меню конструктора таблицы **Продажи** нажмите кнопку связей/отношений (**Relationships**) или в конструкторе таблиц щелкните на кнопке **Отношения (Relationships)** панели инструментов:



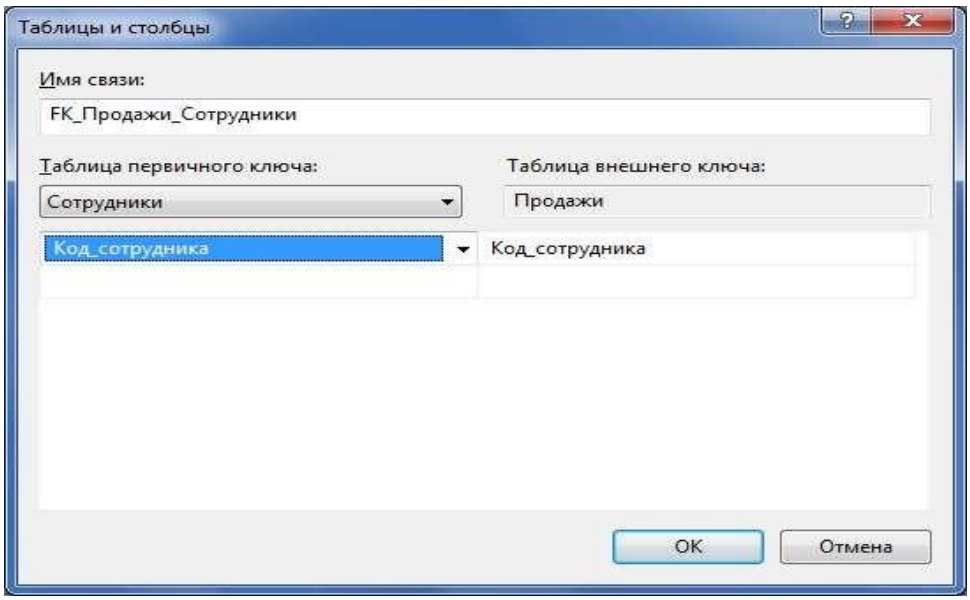


В диалоговом окне **Связи по внешнему ключу (Foreign Key Relationships)** нажмите кнопку **Добавить (Add)** для добавления связей.

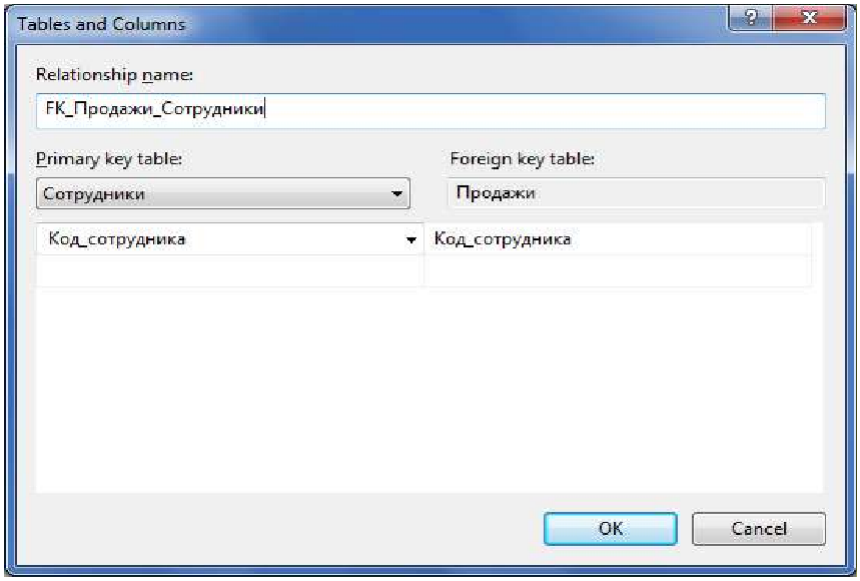
Выделите **Спецификация таблиц и столбцов (Tables and Columns Specification)** и нажмите кнопку с многоточием (...) справа от свойства.

В появившемся диалоговом окне **Таблицы и столбцы (Tables and Columns)** укажите:

- **Таблица первичного ключа (Primary Key Table) –** родительскую таблицу **Сотрудники;**
- **Таблица внешнего ключа (Foreign Key Table)** – таблицу внешнего ключа **Продажи**;
- укажите, что связь устанавливается по Код_сотрудника в обеих таблицах:



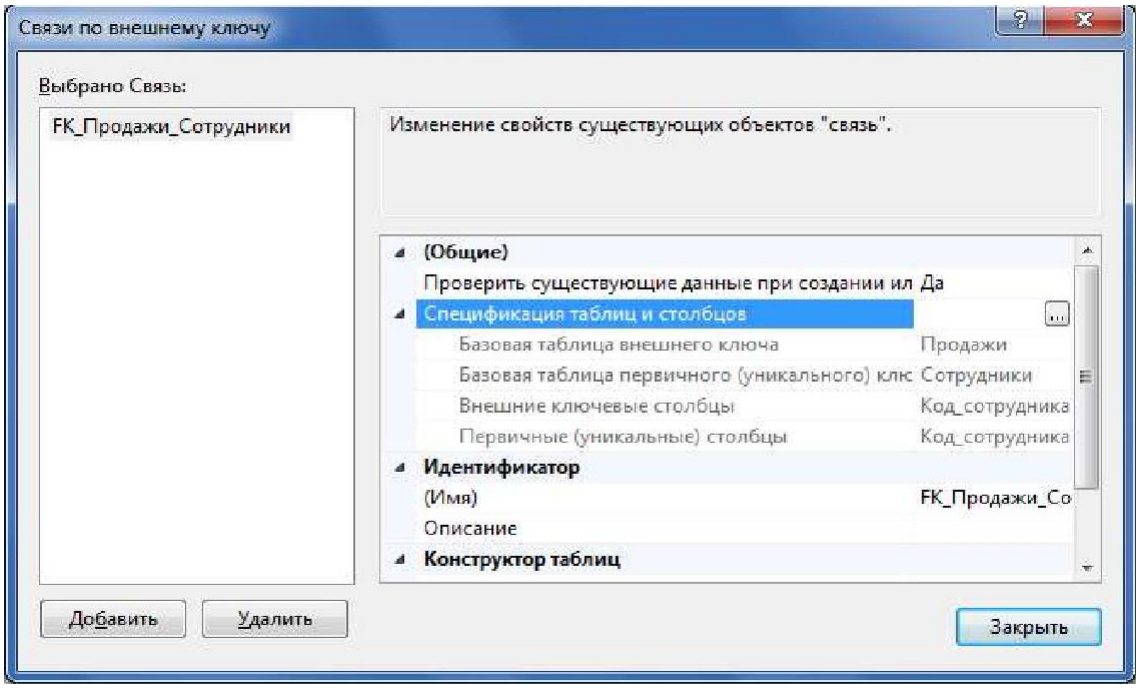
или



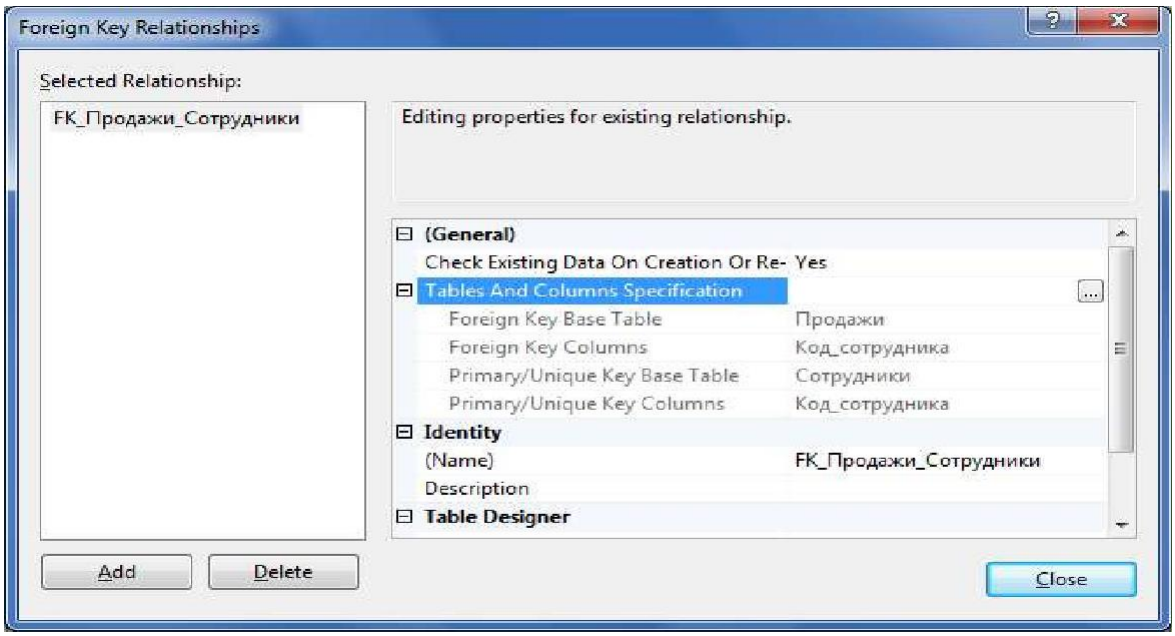
Нажмите кнопку ОК.

В результате в **Спецификация таблиц и столбцов (Tables and Columns Specifications)**:

- **Базовая таблица внешнего ключа (Foreign Key Base Table) –** таблица **Продажи**;
- **Базовая таблица первичного (уникального) ключа (Primary/Unique Key Base Table) –** таблица **Сотрудники;**
- **Внешние ключевые столбцы (Foreign Key Columns) –** Код_сотрудника**;**
- **Первичные (уникальные) столбцы (Primary/Unique Key Columns) –** Код_сотрудника.



или



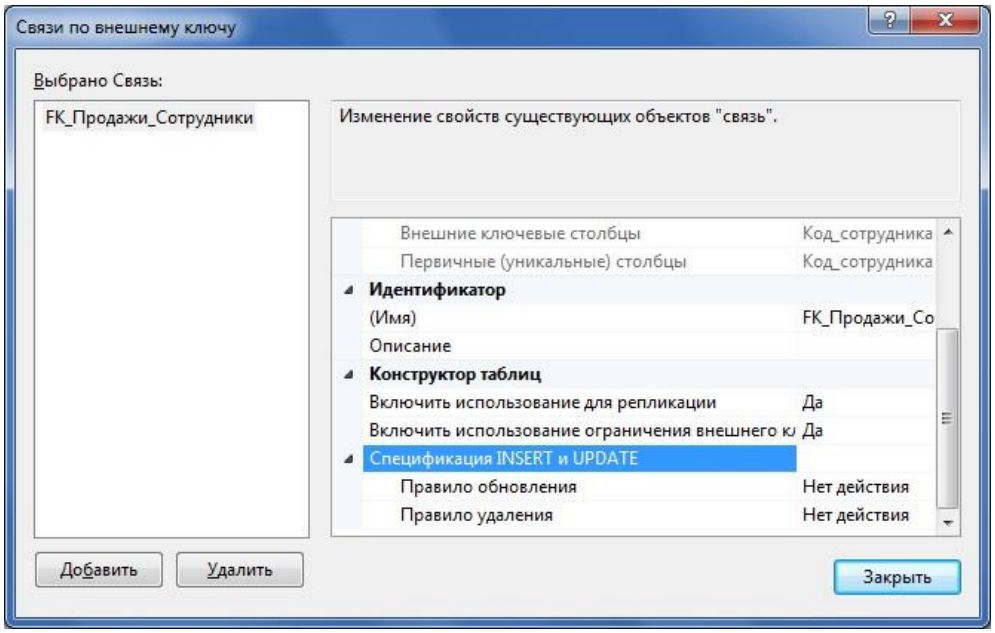
Для данной связи между таблицами **Сотрудники** и **Продажи** механизм **каскадного обновления связанных записей** не устанавливаем, так как Код_сотрудника в Сотрудники является типом данных счетчик, а его нельзя редактировать.

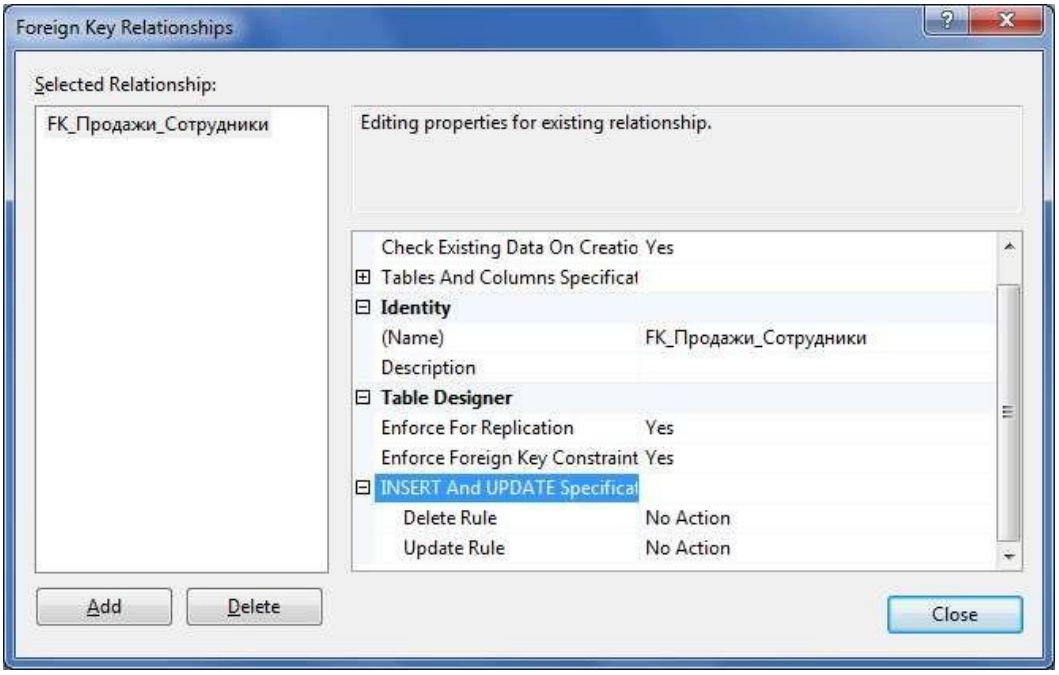
Для данной связи между таблицами **Сотрудники** и **Продажи механизм каскадного удаления связанных записей** не устанавливаем, чтобы при удалении данных о сотруднике, связанные с ним продажи автоматически не удалялись. Чтобы удалить сотрудника необходимо сначала удалить все его продажи, а затем сведения по самому сотруднику.

Таким образом, в **Спецификация INSERT и UPDATE (INSERT and UPDATE Specification)** в свойствах**:**

- Правило обновления (Update Rule) - Нет действия (No Action);

- Правило удаления (Delete Rule) – Нет действия (No Action).





5.2.3.4 Поле Номер_столика

Размер поля **текстовый до 10 символов. NULL значения допустимы**:

Имя столбца	Тип данных	Разрешить значения NULL
Номер_продажи	int	☐
Дата	smalldatetime	☑
Код_сотрудника	int	☐
Номер_столика	nvarchar(10)	☑

5.2.3.5 Сохраните таблицу с именем **Продажи**.

5.2.3.6 Конструктор таблицы **Продажи** имеет вид:

Имя столбца	Тип данных	Разрешить значения NULL
Номер_продажи	int	☐
Дата	smalldatetime	☑
Код_сотрудника	int	☐
Номер_столика	nvarchar(10)	☑

5.2.4 Таблица «Продано»

Проведите создание таблицы **Продано** аналогичным образом с необходимыми ограничениями.

Для таблицы **Продано** задайте в конструкторе таблицы:

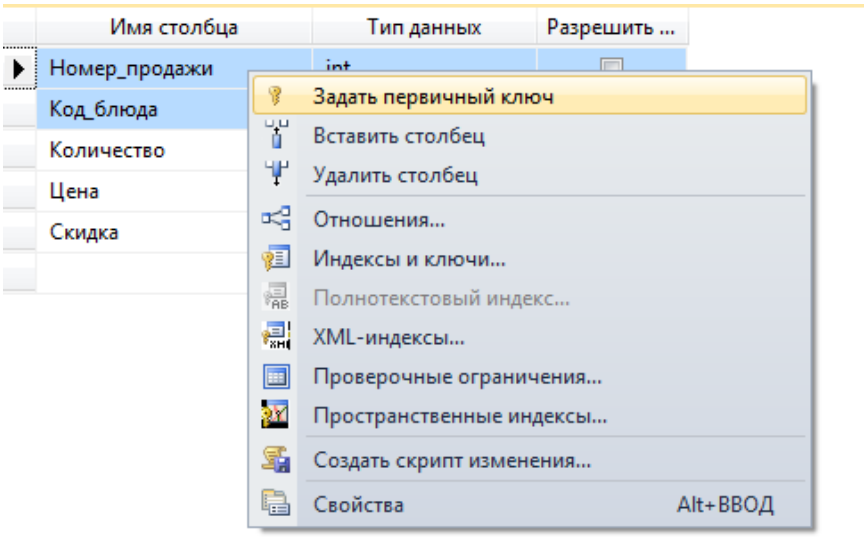
Имя столбца Column Name	Тип данных Data Type	Разрешить значения Nulls Allow Nulls
Номер_продажи	int	
Код_блюда	int	
Количество	int	
Цена	money	
Скидка	decimal(3,2)	Флаг установлен

Теперь зададим необходимые ограничения для соответствующих полей.

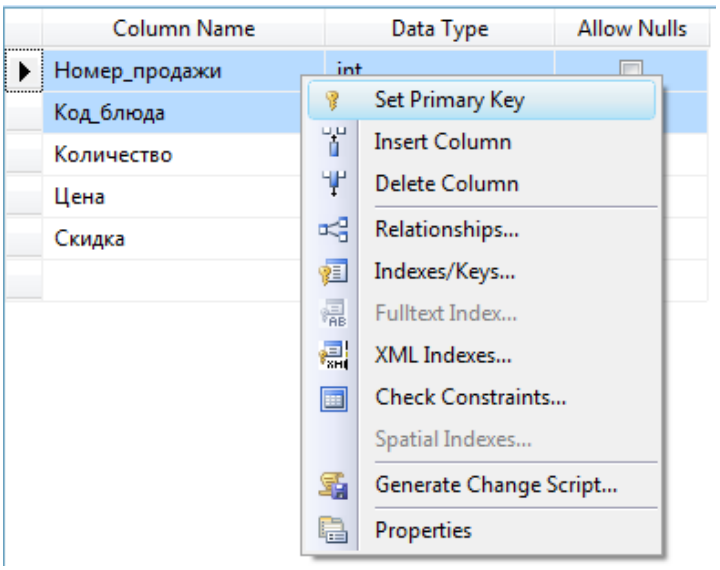
5.2.4.1 Поля Номер_продажи и Код_блюда

Оба этих поля входят в состав первичного ключа.

Выделите строки для Номер_продажи и Код_блюда, нажмите правую кнопку мыши и выберете **Задать первичный ключ (Set Primary Key)**:



или



Имя столбца	Тип данных	Разрешить значения NULL
Номер_продажи	int	☐
Код_блюда	int	☐

Кроме того, данные поля **являются внешними ключами**. Для данных полей можно установить связи по аналогии с внешним ключом Код_сотрудника в Продажи (раздел 5.2.3.3.2) или можно воспользоваться средствами диаграммы базы данных (сделаем так в данном случае, см. раздел 5.3).

5.2.4.2 Поле Количество

5.2.4.2.1 Тип данных – **целые числа, NULL значения не допустимы**:

Имя столбца	Тип данных	Разрешить значения NULL
Номер_продажи	int	☐
Код_блюда	int	☐
Количество	int	☐

5.2.4.2.2 Значение по умолчанию 1:

Аналогично как для поля Цена_меню (раздел 5.2.1.4.2), только в **Значения по умолчанию или привязка (Default Value or Binding)** укажите значение 1.

Имя столбца	Тип данных	Разрешить значения NULL
Номер_продажи	int	☐
Код_блюда	int	☐
Количество	int	☐

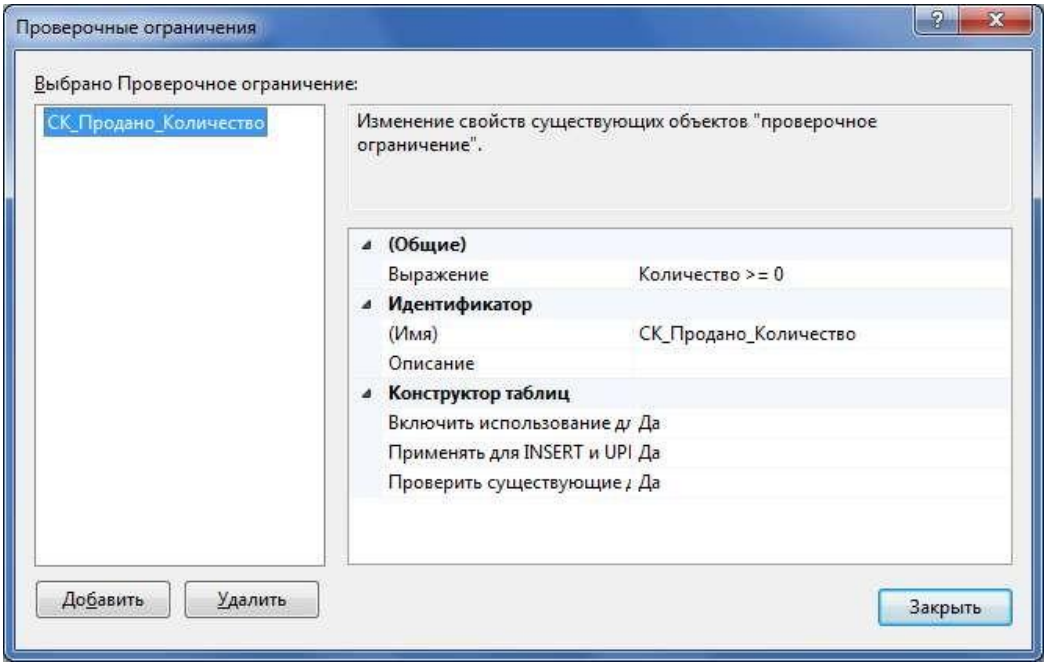
Свойства столбца

(Общие)

(Имя)	Количество
Значение по умолчанию или привязка	1
Разрешить значения NULL	Нет
Тип данных	int

5.2.4.2.3 Условие на значение >= 0

Аналогично как для Код_блюда (раздел 5.2.1.1.2).
Назовите ограничение **CK_Продано_Количество**.



5.2.4.3 Поле Цена

5.2.4.3.1 Тип данных – **денежный, NULL значения не допустимы**:

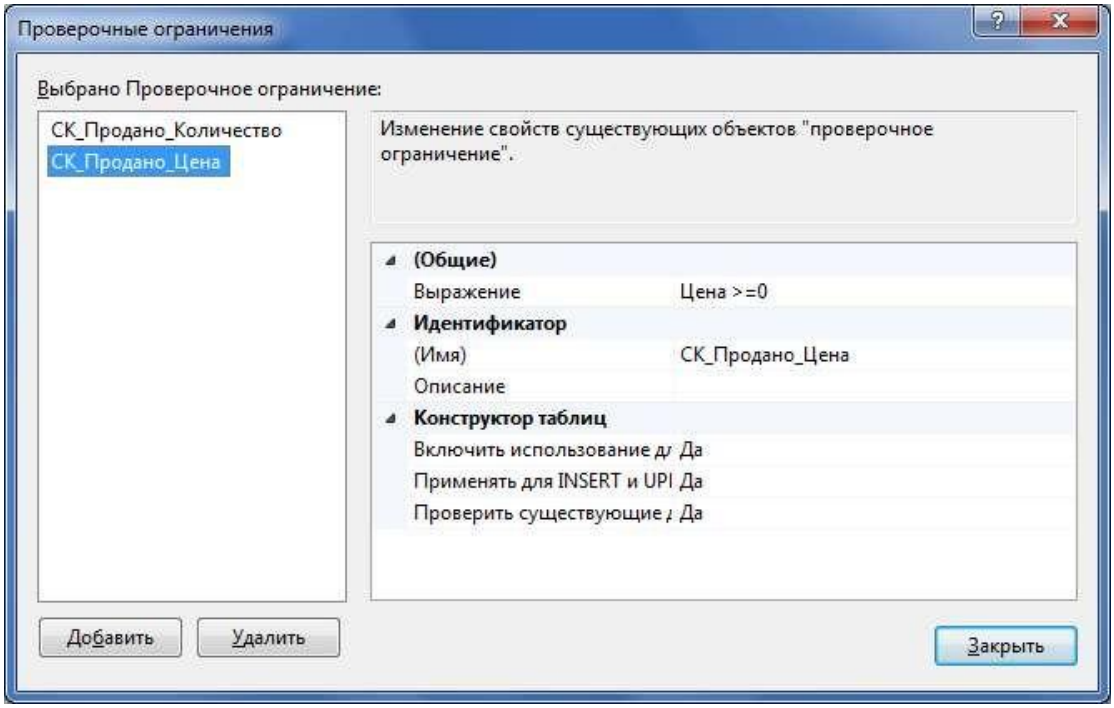
	Имя столбца	Тип данных	Разрешить значения NULL
🔑	Номер_продажи	int	☐
🔑	Код_блюда	int	☐
	Количество	int	☐
▶	Цена	money	☐

5.2.4.3.2 Значение по умолчанию 0:

Аналогично как для поля Цена_меню (раздел 5.2.1.4.2).

5.2.4.3.3 Условие на значение >= 0

Аналогично как для Код_блюда (раздел 5.2.1.1.2).
Назовите ограничение **CK_Продано_Цена**.



5.2.4.4 Поле Скидка

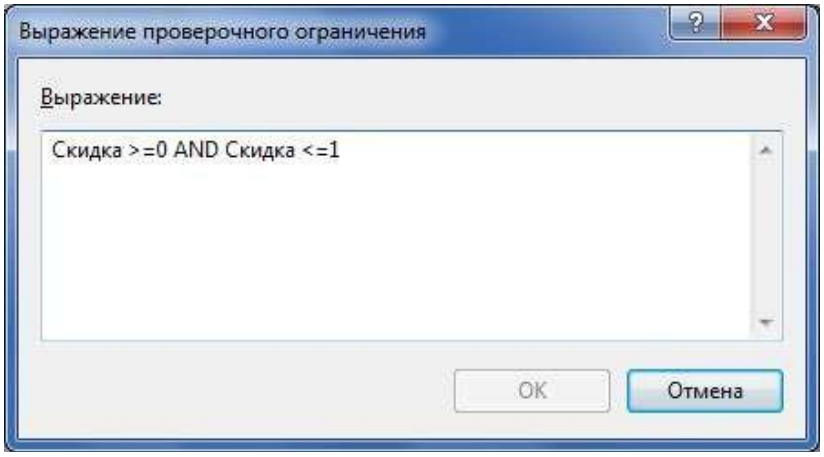
	Имя столбца	Тип данных	Разрешить значения NULL
🔑	Номер_продажи	int	☐
🔑	Код_блюда	int	☐
	Количество	int	☐
	Цена	money	☐
▶	Скидка	decimal(3, 2)	☑

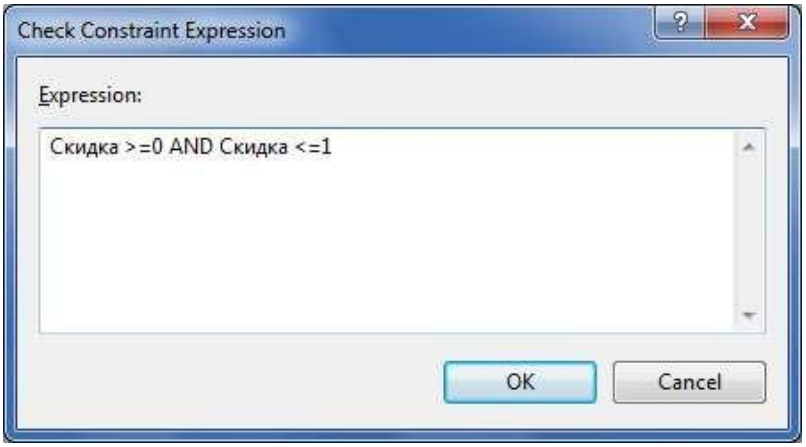
5.2.4.4.1 Значение по умолчанию 0

Аналогично как для поля Цена_меню (раздел 5.2.1.4.2).

5.2.4.4.2 Диапазон скидки от 0 до 1

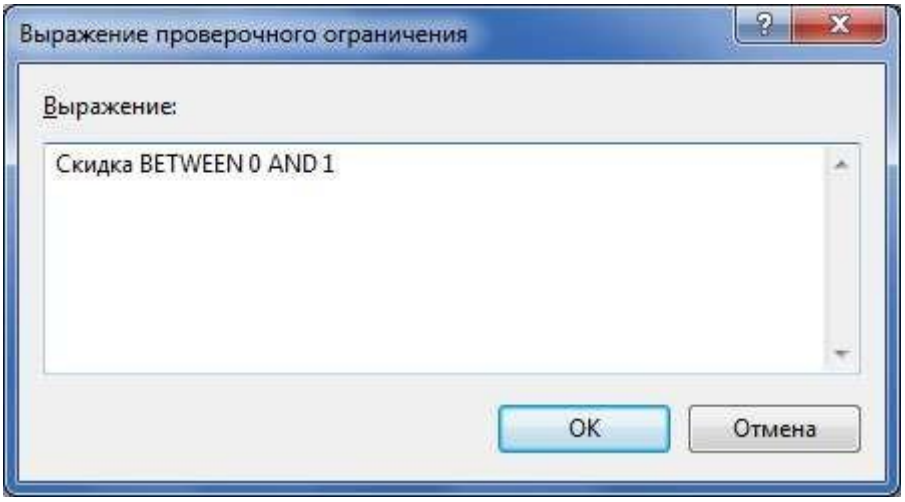
Аналогично как для Код_блюда (раздел 5.2.1.1.2), только в строке Выражение (**Expression)** укажите условие на значения **Скидка >=0 AND Скидка <=1:**



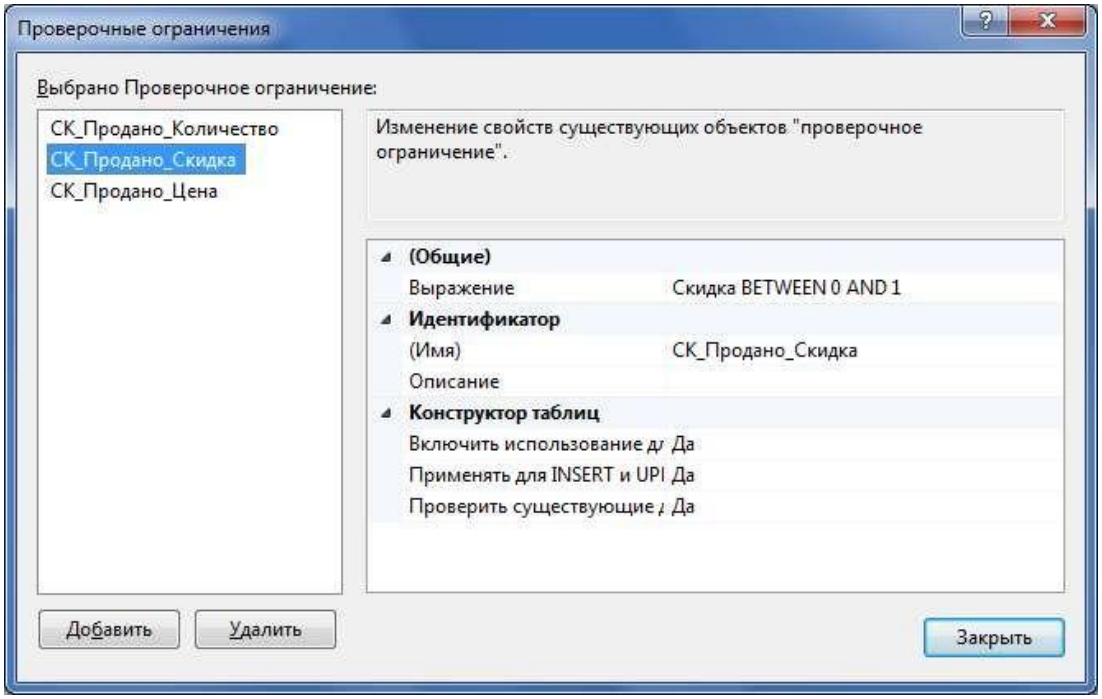


Или

Скидка BETWEEN 0 AND 1:



Назовите это ограничение **CK_Продано_Скидка.**



5.2.4.4.3 Тип данных для данного поля решено установить в **decimal(3,2)**, который позволит задать нам требуемую точность для скидок. В нашем случае диапазон скидки от 0 до 1, всего при записи скидки будут участвовать:

3 (точность) - максимальное количество десятичных разрядов числа (как слева, так и справа от десятичной запятой).

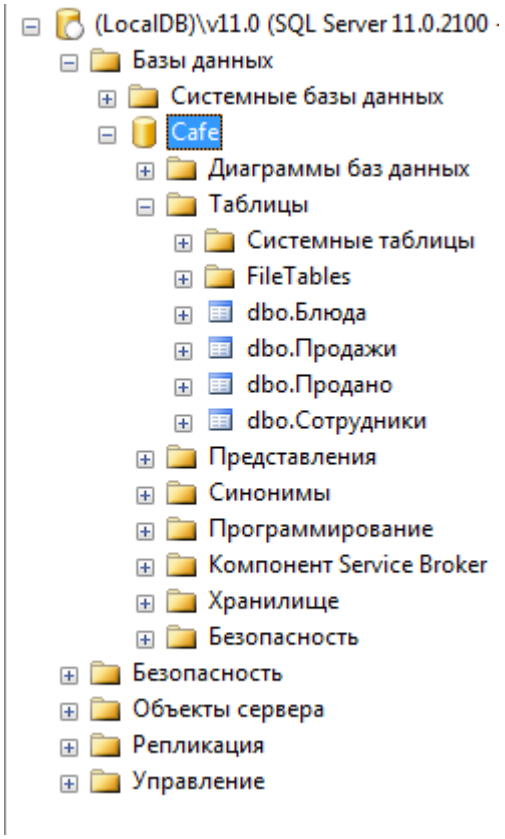
2 (масштаб) - максимальное количество десятичных разрядов числа справа от десятичной запятой.

5.2.4.5 Сохраните таблицу с именем **Продано.**

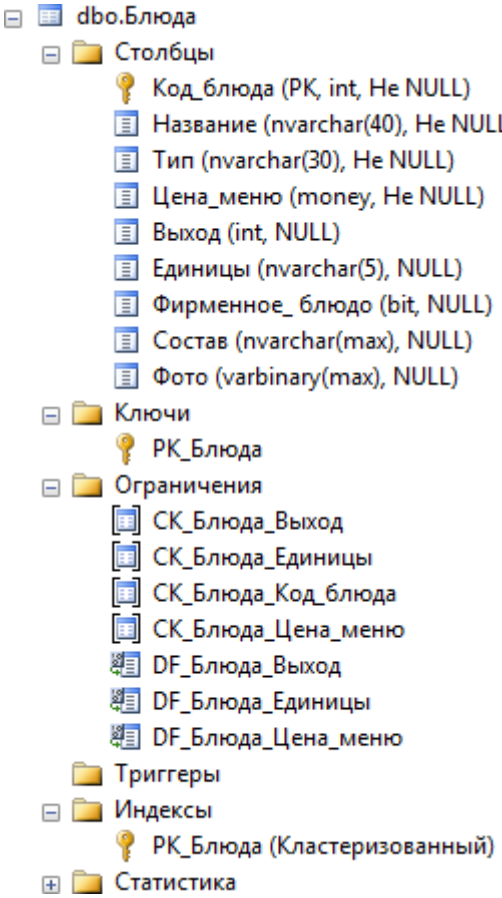
5.2.4.6 Конструктора таблицы **Продано** имеет вид:

Имя столбца	Тип данных	Разрешить значения NULL
Номер_продажи	int	☐
Код_блюда	int	☐
Количество	int	☐
Цена	money	☐
Скидка	decimal(3, 2)	☑

Таким образом, получаем таблицы:



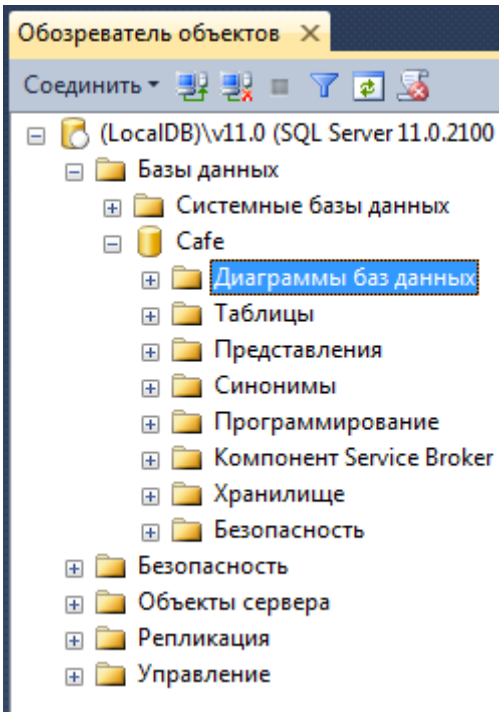
Например, для таблицы Блюда:



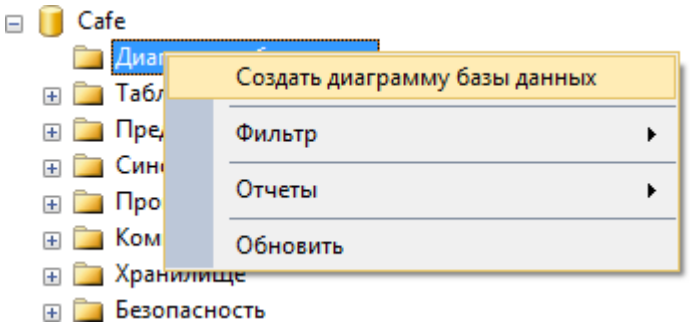
5.3 Создание диаграммы базы данных

Создадим диаграмму для нашей базы данных.

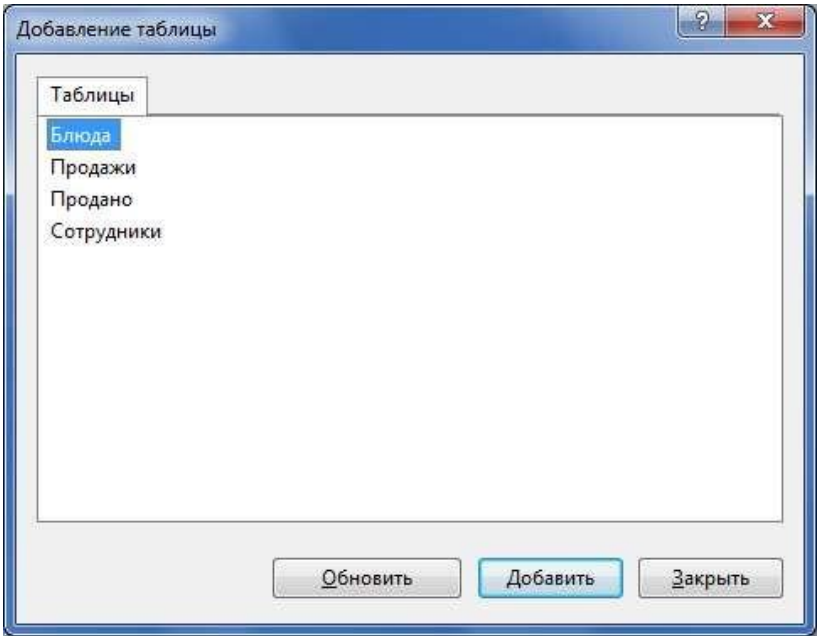
В окне **Обозреватель объектов** выделите узел **Диаграммы баз данных (Database Diagrams):**



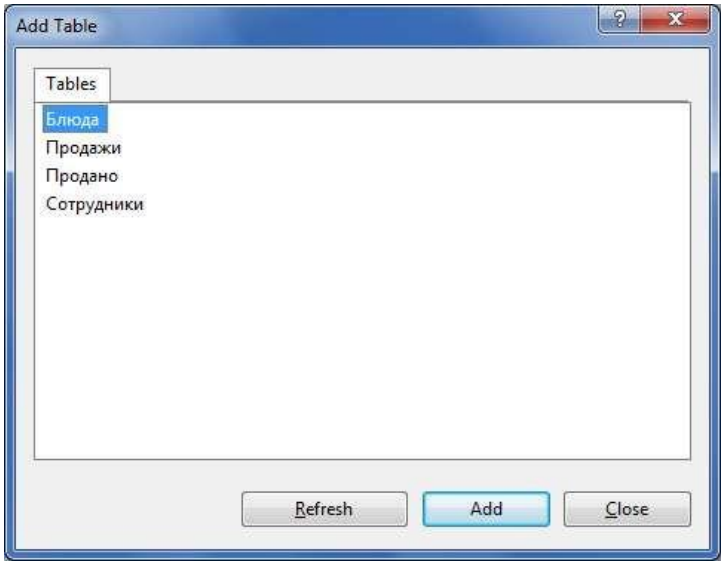
Нажмите правую кнопку на **Диаграммы баз данных (Database Diagrams)** и выберете пункт **Создать диаграмму базы данных**:



В появившемся диалоговом окне **Добавление таблицы (Add Table)** необходимо выбрать таблицы, добавляемые в диаграмму:

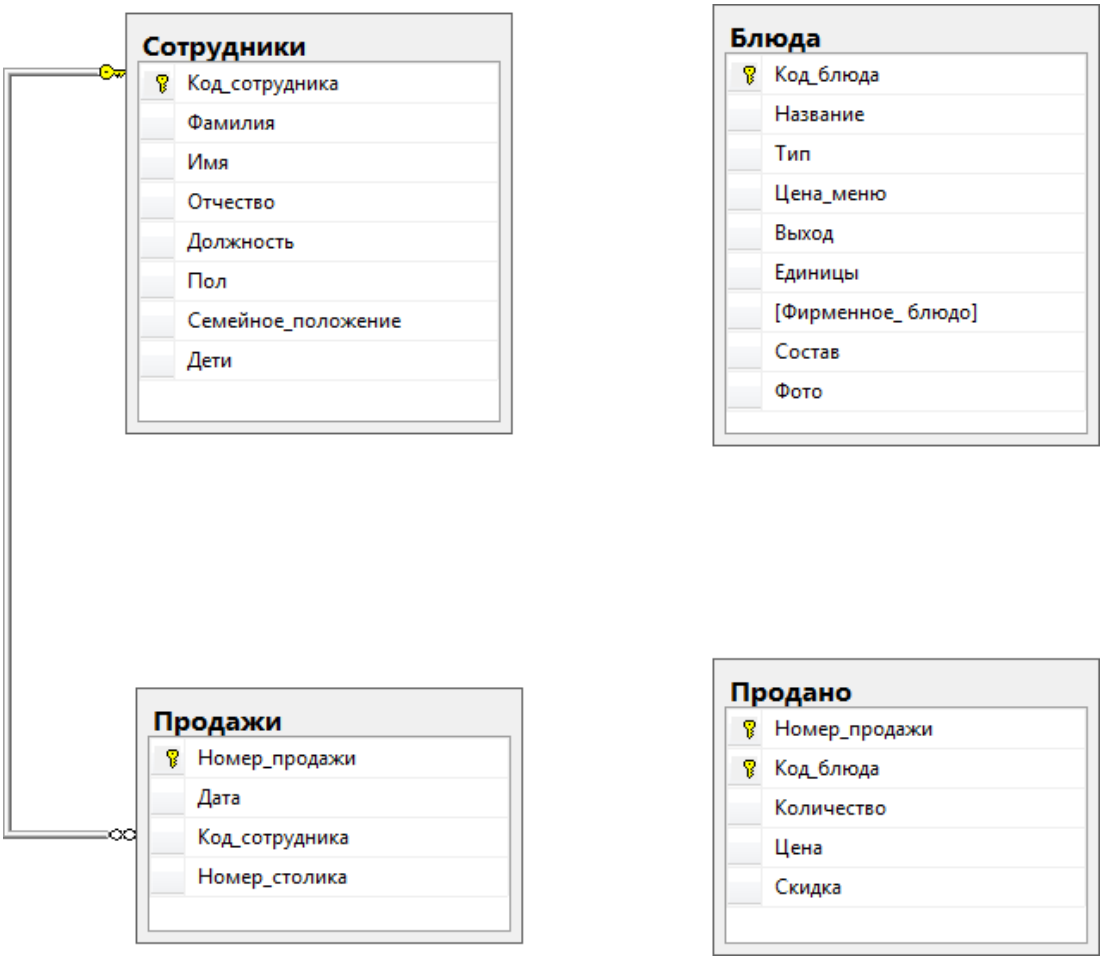


или



Выберете все таблицы БД.

Получаем в диаграмме:



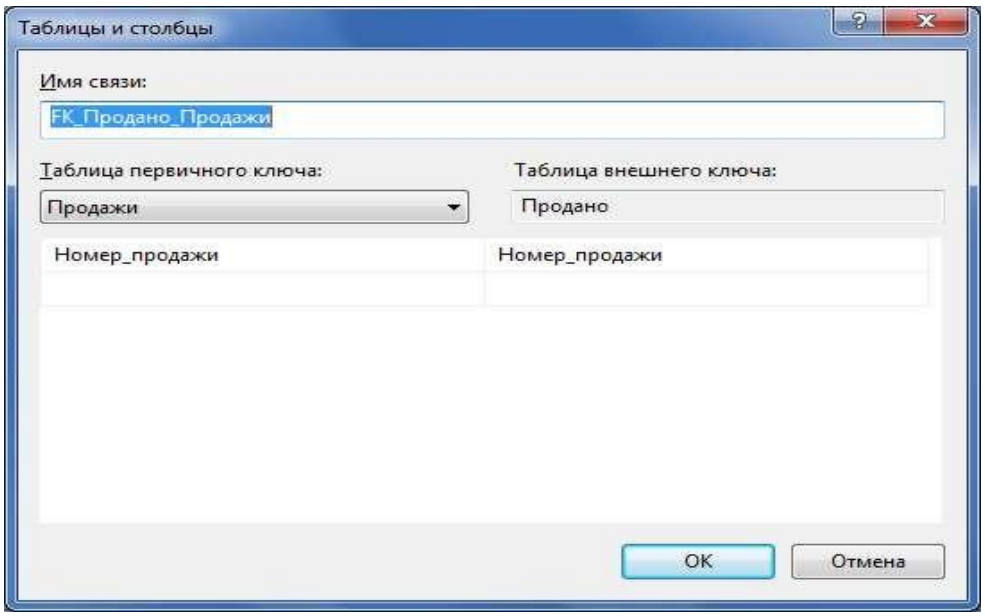
Обратите внимание, что таблицы **Сотрудники** и **Продажи уже связаны, так как данную связь мы установили в разделе 5.2.3.3.2.**

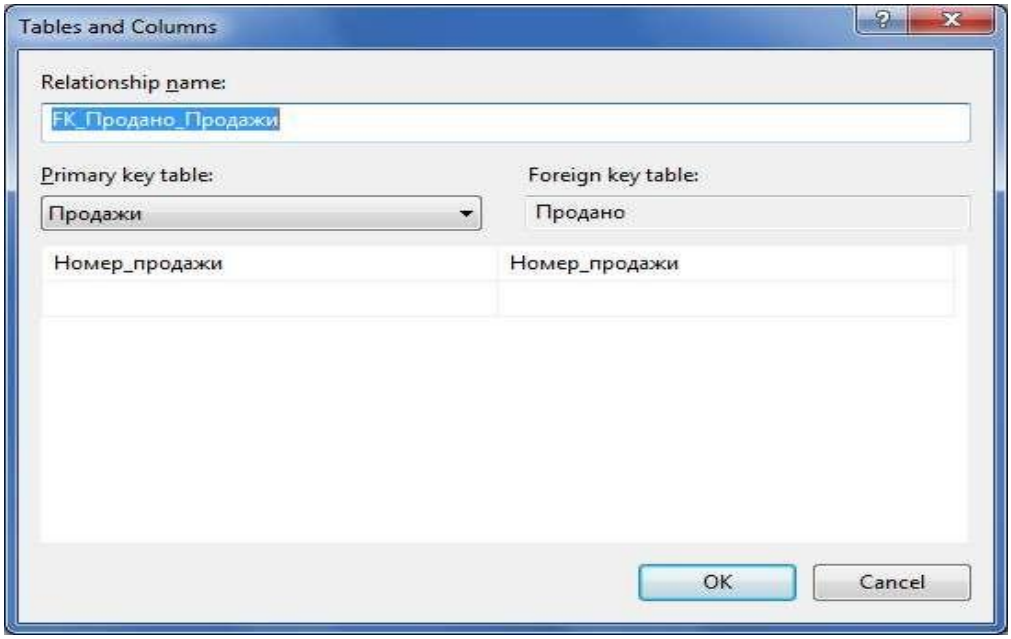
Теперь установим связь между остальными таблицами БД согласно структуре на рисунке 4.1.

5.3.1 Установка связи между Продажи и Продано

Свяжите ключевое поле **Номер_продажи** простым перетаскиванием названия поля от родительской таблицы **Продажи** к дочерней таблице **Продано** по полю **Номер_продажи**:

В появившемся диалоговом окне проверьте правильность установки связи:





Нажмите ОК, если связь устанавливается верно. В нашем случае по Номер_продажи.

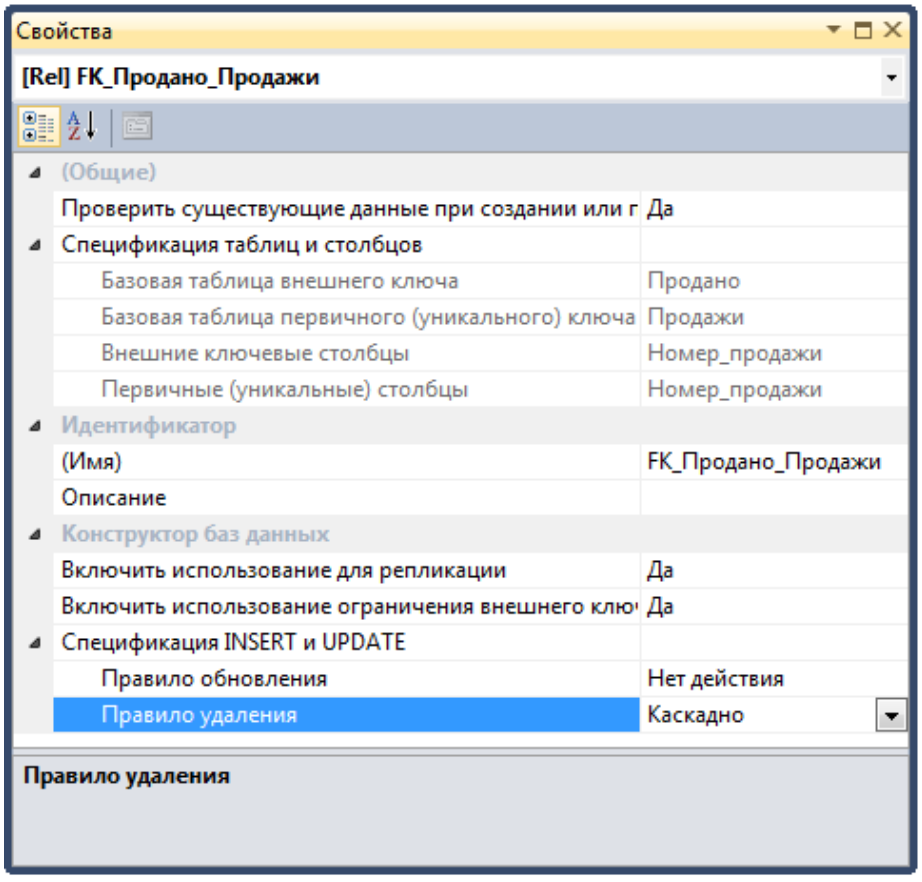
Для данной связи между таблицами **Продажи** и **Продано** механизм **каскадного обновления связанных записей** не устанавливаем, так как Номер_продажи в **Продажи** является типом данных счетчик, а его нельзя редактировать.

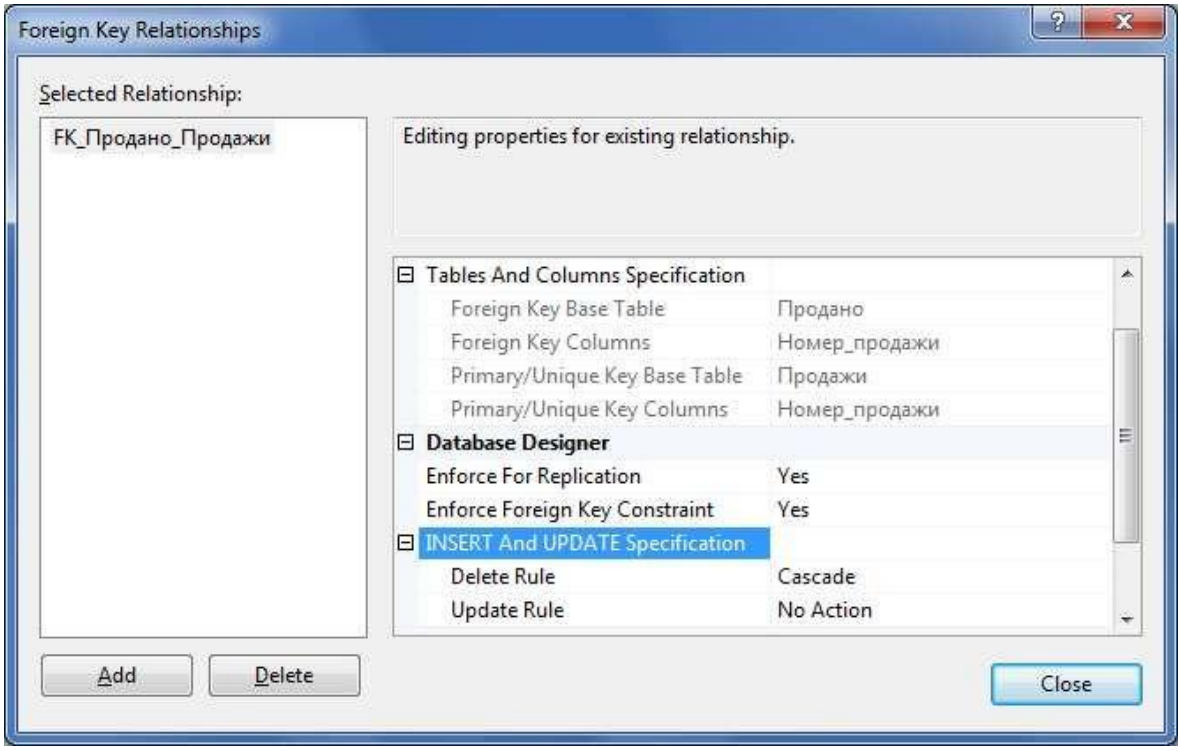
Устанавливаем для данной связи между таблицами **Продажи** и **Продано механизм каскадного удаления связанных записей**, чтобы при удалении данных о продаже, связанные с ней блюда в **Продано**, автоматически удалялись.

Таким образом, в **Спецификация INSERT и UPDATE (INSERT and UPDATE Specification)** в свойствах**:**

- Правило удаления (Delete Rule) – Каскадно (Cascade);

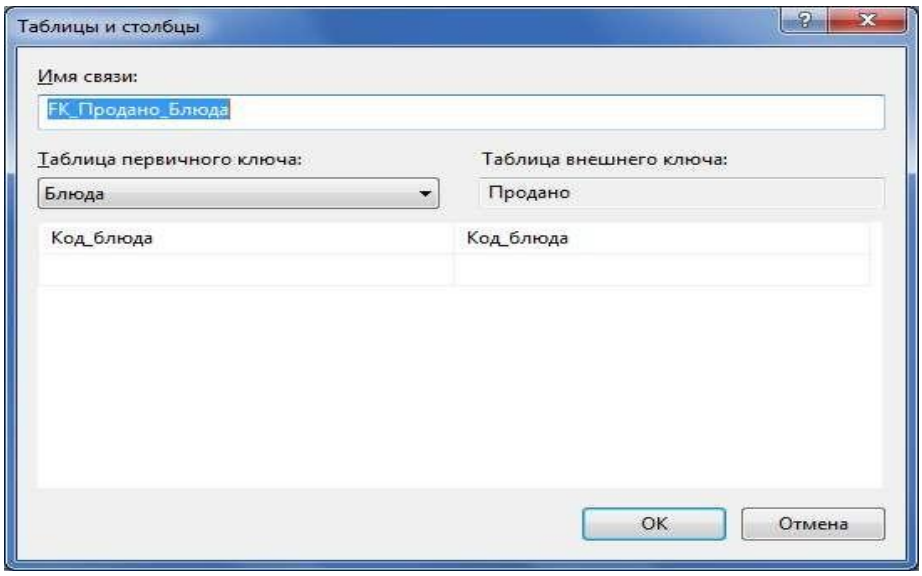
- Правило обновления (Update Rule) – Нет действия (No Action).

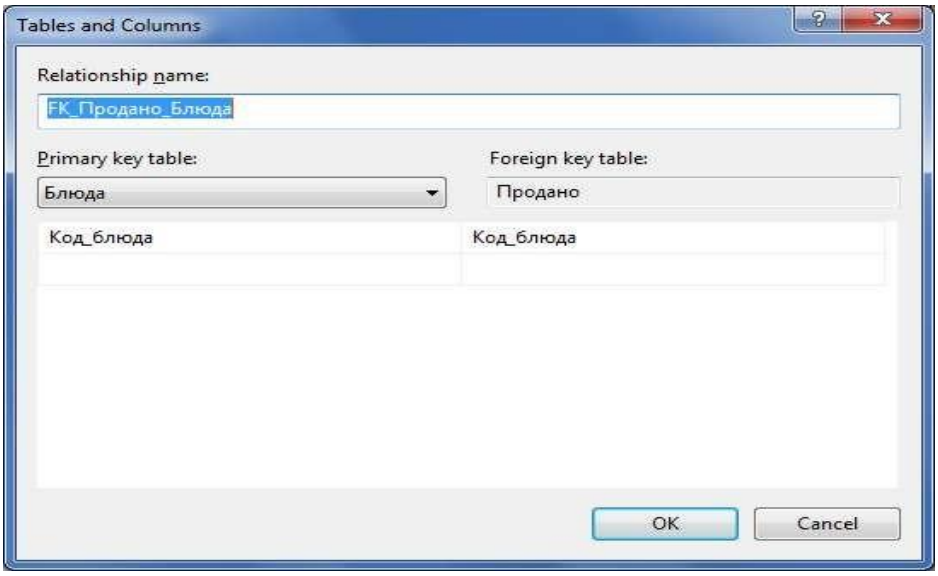




5.3.2 Установка связи между Блюда и Продано

По аналогии установите связь между **Блюда** и **Продано** по полю **Код_блюда**:





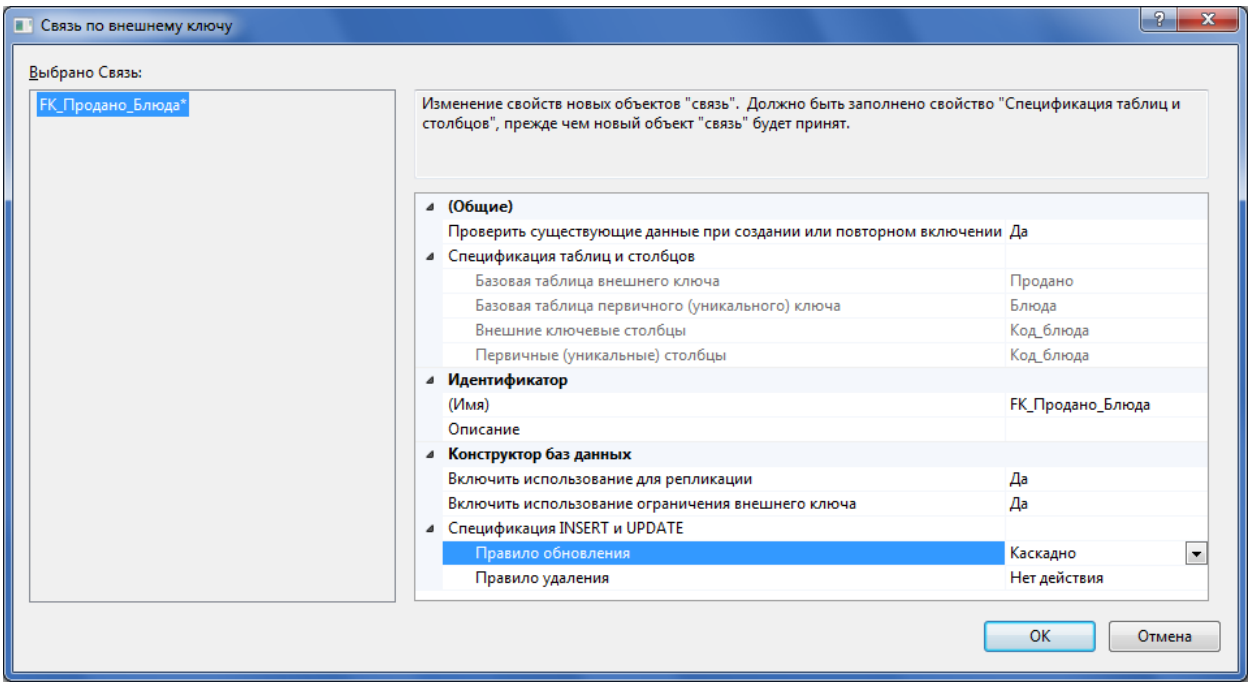
Устанавливаем для данной связи между таблицами **Блюда** и **Продано** механизм **каскадного обновления связанных записей**, так как Код_блюда в **Блюда** может поменяться (например, введение новой формы меню, а блюдо неизменно). При этом необходимо, чтобы связанные с данным блюдом продажи автоматически среагировали на смену значения в Код_блюда.

Механизм каскадного удаления связанных записей для данной связи не устанавливаем, чтобы при удалении данных о блюде, связанные с ним продажи в **Продано**, автоматически не удалялись.

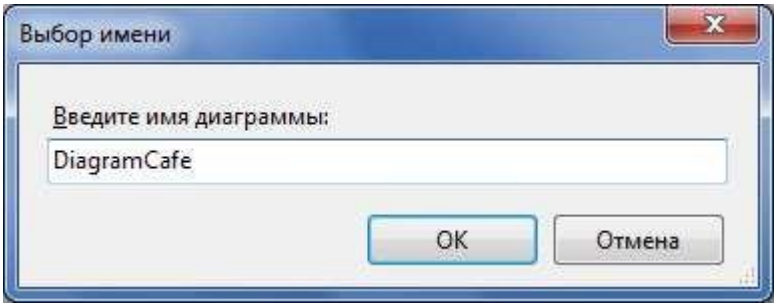
Таким образом, в **Спецификация INSERT и UPDATE (INSERT and UPDATE Specification)** в свойствах**:**

- Правило обновления (Update Rule) – Каскадно (Cascade);

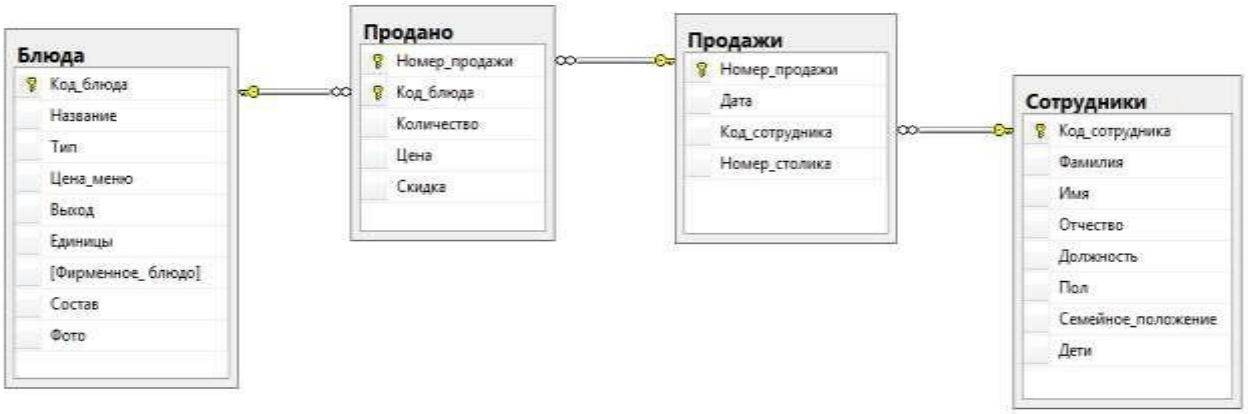
- Правило удаления (Delete Rule) – Нет действия (No Action).



5.3.3 Сохраните полученную диаграмму с именем **DiagramCafe**.



5.3.4 В итоге получаем схему данных:



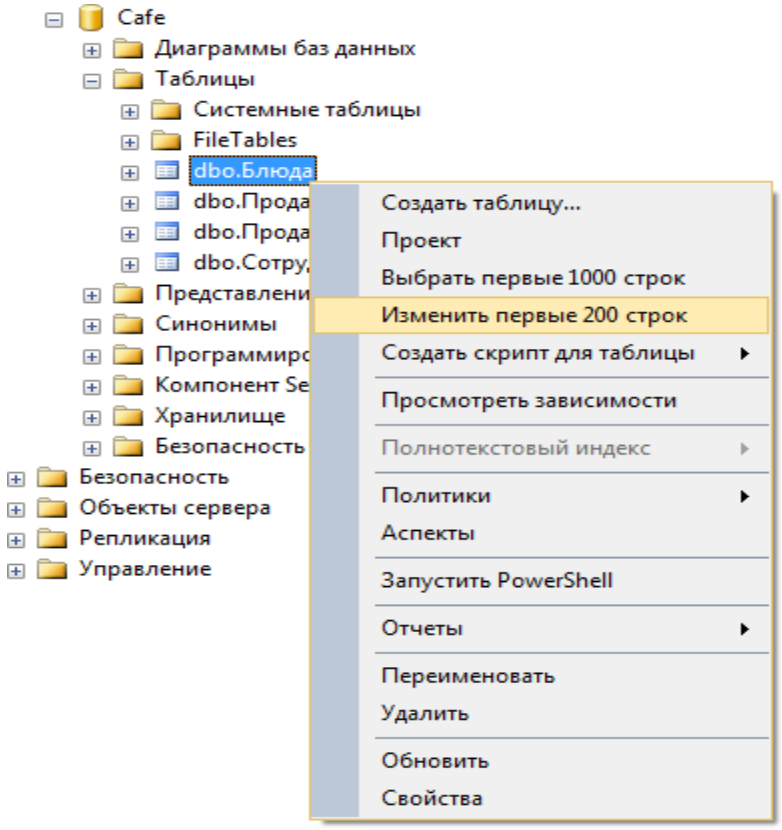
Что соответствует разработанной ранее логической модели, приведенной на рисунке 4.1.

5.4 Заполнение таблиц данными

5.4.1 Заполните по 3-5 строк в каждой созданной таблице.

Для того, чтобы заполнять данные в определенной таблице:
- выделите данную таблицу в узле **Таблицы (Tables):**
- нажмите правую кнопку мыши и выберете **Изменить первые 200 строк:**

Например, для таблицы **Блюда**:



	Код_блюда	Название	Тип	Цена_меню	Выход	Единицы	Фирменное_ блюдо	Состав	Фото
▶	1	Грибной ...	Супы ...	120,0000	250	мл.	True	Грибы, лук, с...	NULL
	2	Салат вкусный ...	Салаты ...	140,0000	300	гр.	True	Форель, зеле...	NULL
	3	Борщ ...	Супы ...	120,0000	250	мл.	False	Картофель, с...	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

5.4.2 Отследите обеспечение целостности

Например, в таблице **Блюда.**

При заполнении данных в данной таблицы были наложены ограничения целостности, нарушение которых будут приводить к соответствующим ошибкам.

Например:

- если повторить значения в первичном ключе **Код_блюда**;
- если оставить пустое значение в первичном ключе **Код_блюда**;
- в **Код_блюда, Цена_меню, Выход** были введены отрицательные числа;
- не заполнено **Название блюда**;
- не заполнен **Тип, Цена_меню**;
- в **Единицы** введено значение, отличное от допустимых "гр." или "мл." или "шт.";
- указан не соответствующий тип данных в **Код_блюда, Выход**;
- указан не соответствующий тип данных в **Цена_меню**:

- ссылочная целостность:

Данная таблица **Блюда** связана с таблицей **Продано**. Для данной связи было установлено обеспечение целостности данных, а также установлен механизм **каскадного обновления связанных записей**, так как Код_блюда в Блюдах может поменяться (например, введение новой формы меню, а блюдо неизменно). Механизм **каскадного удаления связанных записей** для данной связи не установлен, чтобы при удалении данных о блюде, связанные с ним продажи в **Продано**, автоматически не удалялись.

Нарушение данных условий приведет к следующим ошибкам:

- при удалении блюда в таблице Блюда, записи о котором есть в таблице Продано. Чтобы удалить блюдо в данном случае необходимо сначала удалить связанные записи в таблице Продано, а потом удалить блюдо из таблицы Блюда.
- если в таблице Продано указать не существующий в таблице Блюда Код_блюда.

Примечание:

посмотрите, что если для определенного блюда изменить его код, то он автоматически изменится в таблице Продано, так как был установлен механизм **каскадного обновления связанных записей.**

По аналогии опишите наложенные ограничения и для остальных таблиц в своей БД.

6 **Оформление отчета к лабораторной работе**

Оформить отчёт со следующим содержанием:

6.1 Титульный лист.
6.2 Введение (цель работы, постановка задачи).
6.3 Ход выполнения заданий и описание результатов.

Приведите скриншоты полученной схемы данных, конструкторов таблиц, режимов таблиц с заполненными данными. Опишите наложенные ограничения на Вашу БД. Покажите отработку ограничений целостности в Вашей БД.
6.4 Заключение (выводы).

Отчет должен быть оформлен согласно требованиям ОС ТУСУР.

7 Контрольные вопросы

7.1 Что такое база данных?
7.2 Что такое СУБД?
7.3 Что представляет собой реляционная таблица?
7.4 Какие ограничения можно наложить на данные в БД?
7.5 Что такое первичный ключ? Требования к первичному ключу.
7.6 Что такое внешний ключ?
7.7 Что характерно для ссылочной целостности?
7.8 Что подразумевает каскадное удаление и каскадное обновление?
7.9 Охарактеризуйте связь один к одному 1:1.
7.10 Охарактеризуйте связь один ко многим 1:M.
7.11 Охарактеризуйте связь многие ко многим M:M.
7.12 Какие режимы аутентификации поддерживает Microsoft SQL Server?
7.13 Как можно создать БД Microsoft SQL Server через SQL Server Management Studio?
7.14 Как можно создать таблицу Microsoft SQL Server?
7.15 Приведите примеры типов данных в Microsoft SQL Server.
7.16 Приведите примеры свойств полей в Microsoft SQL Server.
7.17 Приведите примеры операторов, использующихся для ограничений проверки.
7.18 Как можно установить связи между таблицами БД в Microsoft SQL Server?

ЛАБОРАТОРНАЯ РАБОТА № 2 СОЗДАНИЕ ПРОСТОГО ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ В MICROSOFT VISUAL STUDIO

Цель работы:

1) Изучение основных компонент для работы с данными в базах данных в Microsoft Visual Studio;
2) Создание простого приложения в Microsoft Visual Studio для работы с базой данных для своей предметной области (на примере базы данных для кафе).

1 Введение

В предлагаемом лабораторном практикуме рассматриваются основные этапы проектирования автоматизированной информационной системы на учебном примере в среде Microsoft Visual Studio.

В качестве языка программирования в среде Microsoft Visual Studio для платформы Net Framework выбран C# (произносится Си Шарп).

Данный полнофункциональный объектно-ориентированный язык программирования специально для платформы .NET разработала компания Microsoft.

.NET Framework - программная технология от компании Microsoft, предназначенная для создания различных программ, в том числе WEB-приложений.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java.

Язык C# находит широкое применение для разработки приложений для настольных и серверных Windows, Web-приложений, для мобильных платформ, таких как Windows Mobile и др.

В качестве учебного примера рассматривается пример реализации программы для кафе с использованием библиотек классов Microsoft.NET Framework, языка программирования C#, системы управления базами данных Microsoft SQL Server.

2 Информация о базе данных

Разрабатываемое приложение предназначено для работы с базой данных для кафе (см. лабораторную работу №1).

Структура БД для кафе в рассматриваемом примере:

В рассматриваемом приложении в качестве СУБД может использоваться Microsoft SQL Server.

Примечание:

В данном примере в БД для удобства объяснения и понимания предметной области используются названия таблиц и полей на русском языке.

---

3 Краткие теоретические сведения

В платформе .NET определено множество типов (организованных в соответствующие пространства имен) для взаимодействия с локальными и удаленными хранилищами данных. Общее название пространств имен с этими типами - ADO.NET.

ADO.NET - это технология доступа к базам данных, специально оптимизированная для нужд построения систем на платформе .NET.

Технология ADO.NET ориентирована на архитектуру многоуровневых приложений, которая в настоящее время стала фактическим стандартом для создания распределенных систем.

Основные отличительные особенности ADO.NET:

- ADO расширяет концепцию объектов-наборов записей в базе данных новым типом DataSet, который представляет локальную копию сразу множества взаимосвязанных таблиц. При помощи объекта DataSet пользователь может локально производить различные операции с содержимым базы данных, будучи физически рассоединен с СУБД, и после завершения этих операций передавать внесенные изменения в базу данных при помощи соответствующего "адаптера данных" (data adapter);

- в ADO.NET реализована поддержка представления данных в XML-совместимых форматах. В ADO.NET сформированные для локальной обработки наборы данных представлены в формате XML (в этом же формате они и передаются с сервера баз данных). Данные в форматах XML очень удобно передавать при помощи обычного HTTP, решает многие проблемы с установлением соединений через брандмауэры;

- ADO.NET - это библиотека управляемого кода и взаимодействие с ней производится как с обычной сборкой .NET. Типы ADO.NET используют возможности управления памятью CLR и могут использоваться во многих .NET - совместимых языках. При этом обращение к типам ADO.NET (и их членам) производится практически одинаково вне зависимости от того, какой язык используется.

Все типы ADO.NET предназначены для выполнения единого набора задач:

- установить соединение с хранилищем данных;
- создать и заполнить данными объект DataSet;
- отключиться от хранилища данных и вернуть изменения, внесенные в объект DataSet обратно в хранилище данных.

Объект DataSet - это тип данных, представляющий локальный набор данных и информацию об отношениях между ними.

На практике можно создать на клиенте объект DataSet, который будет представлять полную копию удаленной базы данных.

После создания объекта DataSet и его заполнения данными можно программными средствами производить запросы к нему и перемещаться по таблицам, выполнять все операции, как при работе с обычными базами

данных: добавлять в таблицы новые записи, удалять и изменять существующие, применять к ним фильтры и т.п. После того как клиент завершит внесение изменений, информация о них будет отправлена в хранилище данных для обработки.

В состав ADO.NET включены управляемые провайдера для взаимодействия с различными СУБД.

В любом приложении ADO.NET необходимо использовать, по крайней мере, одно пространство имен - System.Data. В основном эти типы представляют собой объектные представления примитивов для работы с базами данных - таблицами, строками, столбцами, ограничениями и т. п. Наиболее часто используемые типы System.Data представлены ниже:

Тип	**Назначение**
DataColumnCollection, DataColumn	DataColumn представляет один столбец в объекте DataTable, DataColumnCollection - все столбцы
ConstraintCollection, Constraint	Constraint - объектно-ориентированная оболочка вокруг ограничения (например, внешнего ключа или уникальности), наложенного на один или несколько DataColumn, ConstraintCollection - все ограничения в объекте DataTable
DataRowCollection, Data Row	DataRow представляет единственную строку в DataTable, DataRowCollection - все строки в DataTable
DataRowView, Data View	DataRowView позволяет создавать настроенное представление единственной строки, DataView - созданное программным образом представление объекта DataTable, которое может быть использовано для сортировки, фильтрации, поиска, редактирования и перемещения
DataSet	Объект, создаваемый в оперативной памяти на клиентском компьютере. DataSet состоит из множества объектов DataTable и информации об отношениях между ними

ForeignKeyConstraint, UniqueConstraint	ForeignKeyConstraint представляет ограничение, налагаемое на набор столбцов в таблицах, связанных отношениями первичный - внешний ключ. UniqueConstraint - ограничение,

	при помощи которого гарантируется, что в столбце не будет повторяющихся записей
DataRelationCollection, DataRelation, DataTableCollection, DataTable	Тип DataRelationCollection представляет набор всех отношений (то есть объектов DataRelation) между таблицами в DataSet. Тип DataTableCollection представляет набор всех таблиц (объектов DataTable) в DataSet

В традиционных системах клиент-сервер при запуске приложения пользователем автоматически устанавливается связь с базой данных, которая поддерживается в "активном" состоянии до тех пор, пока приложение не будет закрыто. Такой метод работы с данными становится непрактичным, поскольку подобные приложения трудно масштабируются. Например, такая прикладная система может работать достаточно быстро и эффективно при наличии 8-10 пользователей, но она может стать полностью неработоспособной, если с ней начнут работать 100, 200 и более пользователей. Каждое открываемое соединение с базой данных "потребляет" достаточно много системных ресурсов сервера, они становятся занятыми поддержкой и обслуживанием открытых соединений, их практически не остается на процессы непосредственной обработки данных.

По этой причине, в ADO.NET используется модель работы пользователя в отрыве от источника данных. Приложения подключаются к базе данных только па небольшой промежуток времени. Соединение устанавливается только тогда, когда клиент удаленного компьютера запрашивает на сервере данные. После того, как сервер подготовил необходимый набор данных, сформировал и отправил их клиенту, связь приложения с сервером сразу же обрывается, и клиент просматривает полученную информацию уже не в связи с сервером.

В объектной модели ADO.NET можно выделить несколько уровней.

Уровень данных. Это по сути дела базовый уровень, на котором располагаются сами данные (например, таблицы базы данных СУБД). На данном уровне обеспечивается физическое хранение информации на магнитных носителях и манипуляция с данными на уровне исходных таблиц (выборка, сортировка, добавление, удаление, обновление и т. п.).

Уровень бизнес-логики. Это набор объектов, определяющих, с какой базой данных предстоит установить связь и какие действия необходимо будет выполнить с содержащейся в ней информацией. Для установления связи с базами данных используется объект DataConnection. Для хранения команд, выполняющих какие либо действия над данными, используется объект DataAdapter. И, наконец, если выполнялся процесс выборки информации из базы данных, для хранения результатов выборки

используется объект DataSet. Объект DataSet представляет собой набор

данных "вырезанных" из таблиц основного хранилища, который может быть передан любой программе-клиенту, способной либо отобразить эту информацию конечному пользователю, либо выполнить какие-либо манипуляции с полученными данными.

Уровень приложения. Это набор объектов, позволяющих хранить и отображать данные на компьютере конечного пользователя. Для хранения информации используется уже знакомый нам объект DataSet, а для отображения данных имеется довольно большой набор элементов управления (DataGridView, TextBox, ComboBox и т. д.).

В Visual Studio .Net можно вести разработку приложений различных типов. В первую очередь это традиционные Windows-приложения (на основе Windows-форм), которые реализованы в виде exe-файлов, запускаемых на компьютере пользователя. Ну и конечно, Web-приложения (на основе Web-форм), которые работают в оболочке браузера. Для хранения данных на уровне обоих типов приложений используется объект DataSet.

Обмен данными между приложениями и уровнем бизнес-логики происходит с использованием формата XML, а средой передачи данных служат либо локальная сеть (Интранет), либо глобальная сеть (Интернет).

В ADO.NET для манипуляции с данными могут использоваться команды, реализованные в виде SQL-запросов или хранимых процедур (DataCommand).

Если требуется выполнить более чем одну операцию с данными, например, получить некоторый набор данных, а затем скорректировать его, - то необходимо выполнить последовательность команд. Каждая команда выполняется отдельно, последовательно одна за другой. Между выполняемыми командами соединение с базой отсутствует.

Очень часто, в приложении нужно обработать не одну запись, а их набор: список сотрудников, перечень продаж и т.п. Как правило, в приложениях требуется одновременная работа с более чем одной таблицей: сотрудники и все их продажи; продажа и ее элементы, т.е. с набором связанных данных. Причем для удобства пользователя данные требуется группировать и сортировать то по одному, то по другому признаку. При этом нерационально каждый раз возвращаться к исходной базе данных и заново считывать данные. Более практично работать с некой временной "вырезкой" информации, хранящейся в оперативной памяти компьютера.

Эту роль выполняет набор данных - DataSet, который представляет собой своеобразный кэш записей, извлеченных из базового источника. DataSet может состоять из одной или более таблиц, он имеет дело с копиями таблиц из базы данных источника. Кроме того, в данном объекте могут содержаться связи между таблицами и некоторые ограничения на выбираемые данные:

Данные в DataSet - это некий уменьшенный вариант основной базы данных. Тем не менее, вы можете работать с такой "вырезкой" точно так же, как и с реальной базой.

Вы можете выполнить операции коррекции непосредственно в DataSet, а потом все внесенные изменения будут переданы в основную базу данных.

Важно отметить то, что DataSet - пассивный контейнер для данных, который обеспечивает только их хранение. Что же нужно поместить в этот контейнер, определяется в другом объекте - адаптере данных DataAdapter. В адаптере данных содержатся одна или более команд, которые определяют, какую информацию нужно поместить в таблицы объекта DataSet, по каким правилам нужно синхронизировать информацию в конкретной таблице DataSet и соответствующей таблицей основной базы данных и т. п.

Адаптер данных обычно содержит четыре команды SELECT (выбрать), INSERT (добавить), UPDATE (изменить), DELETE (удалить).

Набор данных DataSet - "независимая" копия фрагмента базы данных, расположенная на компьютере пользователя. Причем в этой копии могут быть не отражены те изменения, которые могли внести в основную базу данных другие пользователи. Если требуется увидеть самые последние изменения, сделанные другими пользователями, то необходимо обновить DataSet.

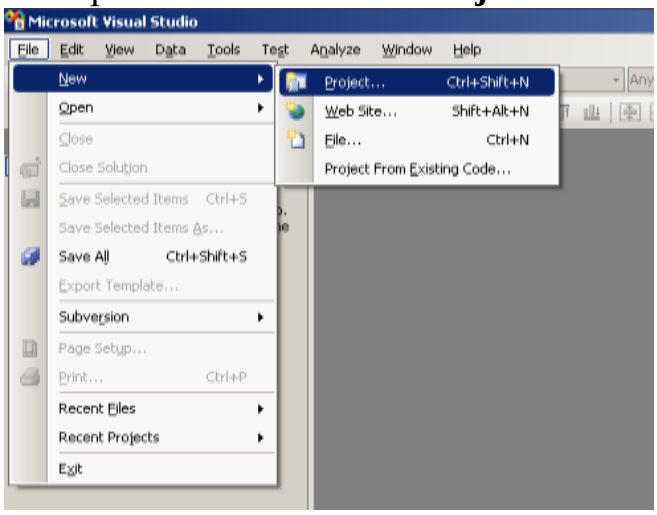
4 Практические задания

4.1 Создание приложения для работы с данными кафе

Создайте на диске отдельную папку для файлов дальнейшей программы. Например, C:\LabsBD\

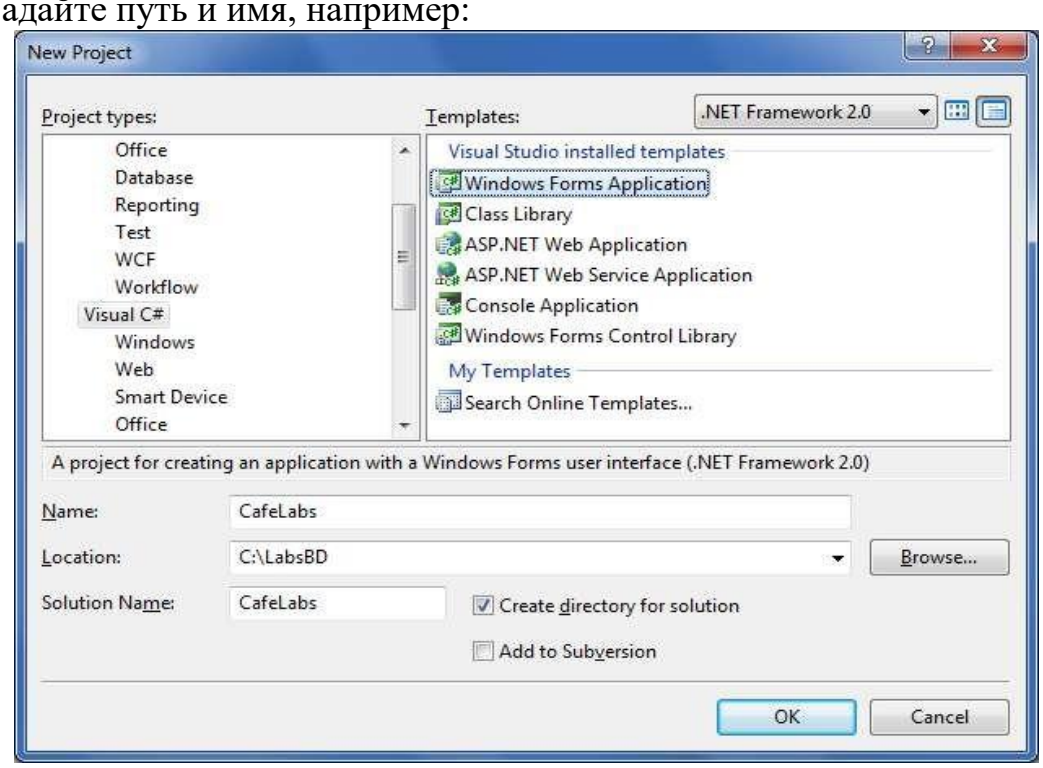
Запустите **Microsoft Visual Studio**.

Выберете меню **File\New\Project**.



Выберете проект на Vusal C#.

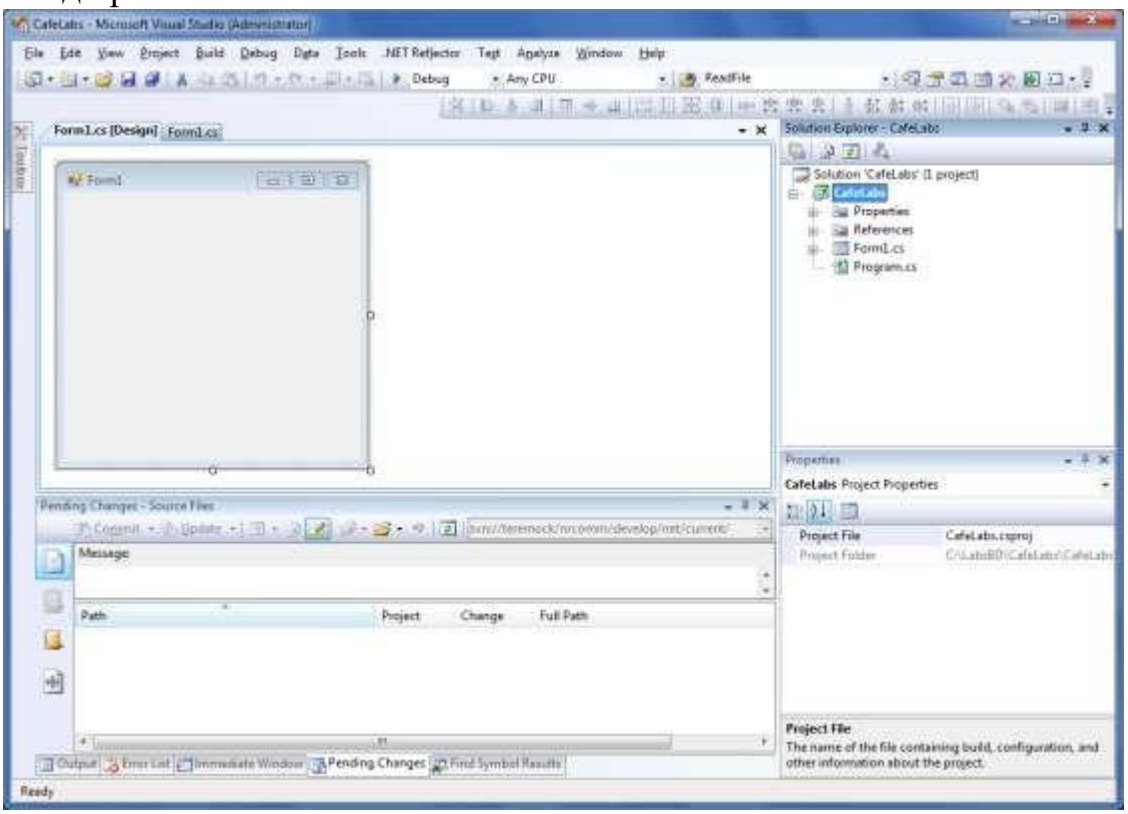
Задайте путь и имя, например:



Нажмите **ОК**.

Установленный флаг **Create directory for solution** приведет к тому, что для файлов проекта будет создана отдельная папка, с соответствующим именем.

Вид проекта в Visual Studio:



При помощи дизайнера графических форм можно добавлять в приложение различные элементы управления и он будет автоматически генерировать код для этих элементов (по умолчанию файл с главной формой приложения называется Form1.cs).

В Visual Studio классы Windows-форм формируются в двух файлах: Form1.cs и Form1.Designer.cs. В файле Form1.Designer.cs присутствует код, сгенерированный дизайнером, а файле Form1.cs - присутствует код инициализации класса и пользовательские члены класса (поля, свойства, методы, события и др.)

Для разметки окон приложения в соответствии с требованиями пользователя необходимо изменить свойства класса Forms1. Это можно сделать с помощью дизайнера окон (**Form Designer**), путем изменения свойств в окне Свойства (**Properties**) или в коде программы.

Наш проект будет состоять из нескольких форм.

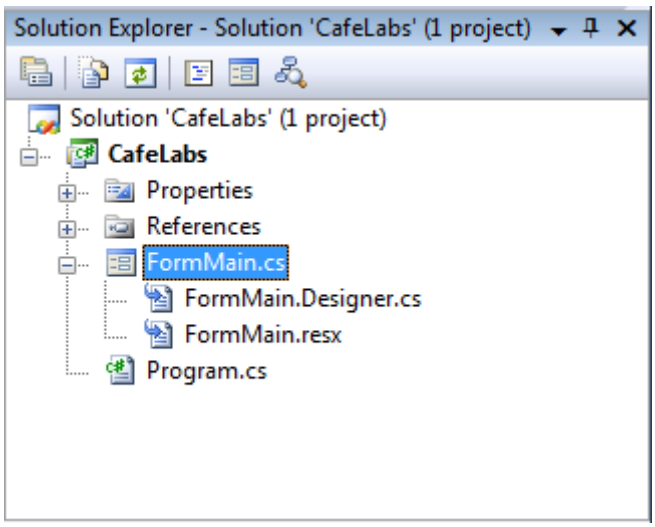
4.2. Подготовка главной формы

Данная форма отображается после запуска программы.

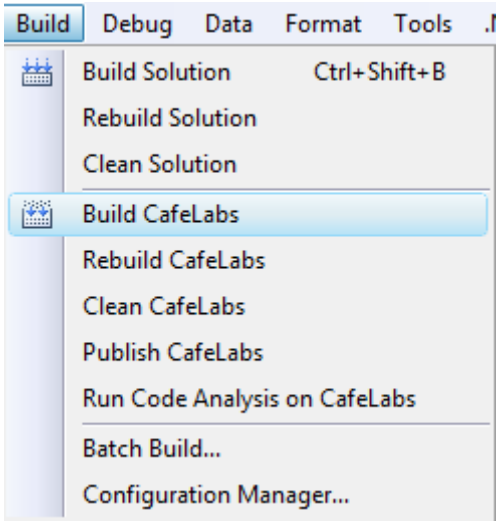
На вкладке **Properties** (Свойства) для **Form1** измените значение в поле **Text** (Заголовок) на **Программа для кафе**. При этом на форме изменится заголовок окна.

На вкладке **Properties** (Свойства) измените для **Form1** значение в свойстве **Name** (Имя) на **FormMain**.

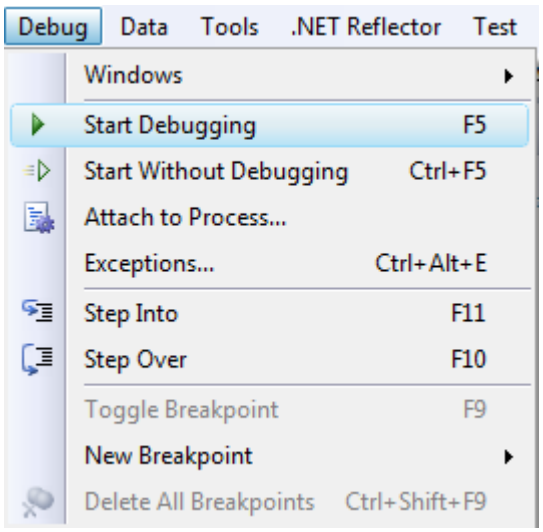
В окне проекта измените имя файла для главной формы на **FormMain.cs**:



Откомпилируйте приложение, выбрав из главного меню раздел **Build** действие **Build CafeLabs:**

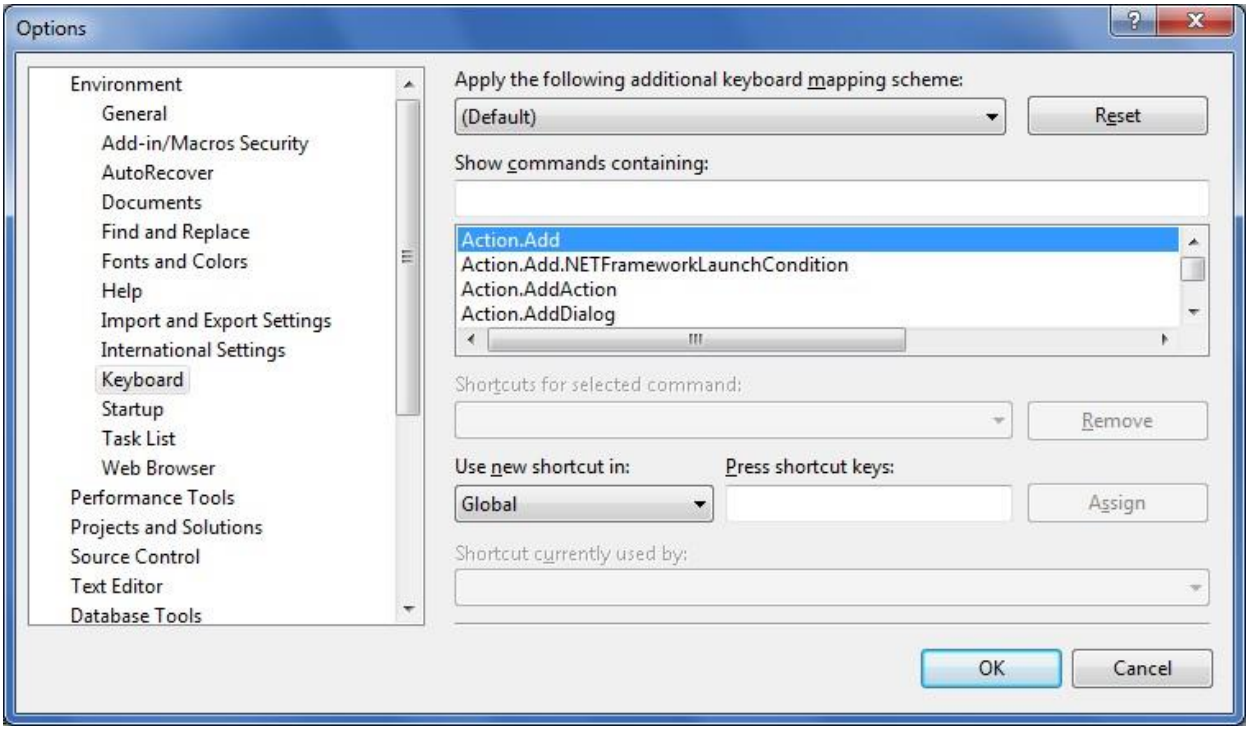


Нажмите **F5**. Посмотрите, как работает созданное приложение.



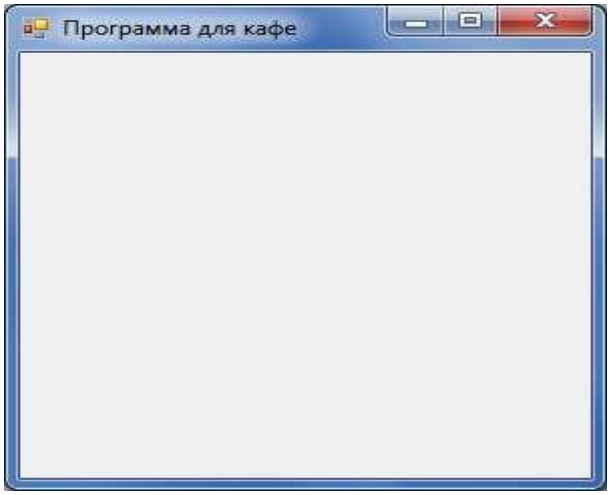
Примечание:

Описание горячих клавиш дается согласно настройкам по умолчанию в Visual Studio:



Если Вы их в настройках меняете, то они могут быть другими.

Вид формы запущенного приложения имеет вид:



Закройте запущенную программу. Перейдите Visual Studio.

4.2.1 Создание главного меню

Для создания главного меню используется компонент **MenuStrip** с вкладки **ToolBox**.

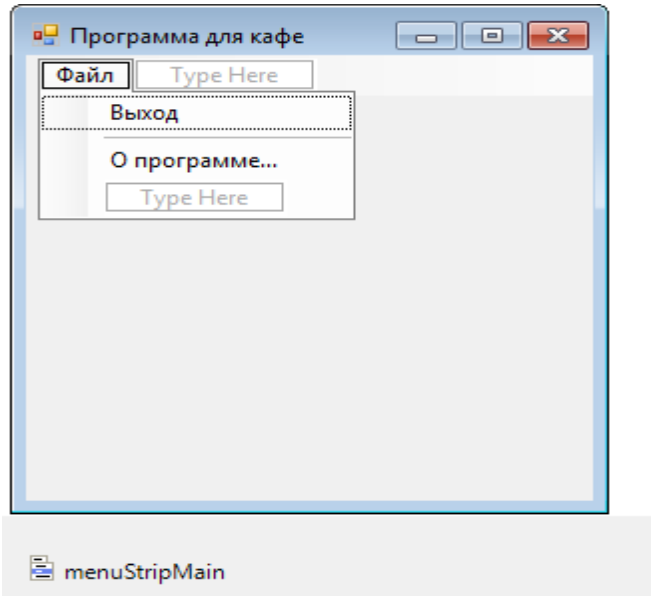
Это невизуальный компонент при запуске приложения. При запуске приложения пользователи видят не сам компонент, а меню, сгенерированное им.

Создание главного меню на главной форме.

Добавим в главную форму элемент для меню. Для этого откройте вкладку **ToolBox** и сначала щелкнете мышью на элементе **MenuStrip** вкладки, а затем щелкнем мышью на форме.

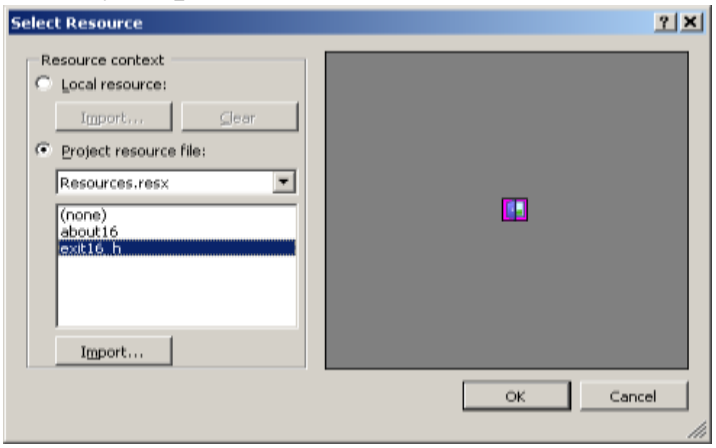
Свойство **Name** для него укажите в **menuStripMain**

В дизайнере меню укажите пункты через свойство **Text** соответствующего пункта:

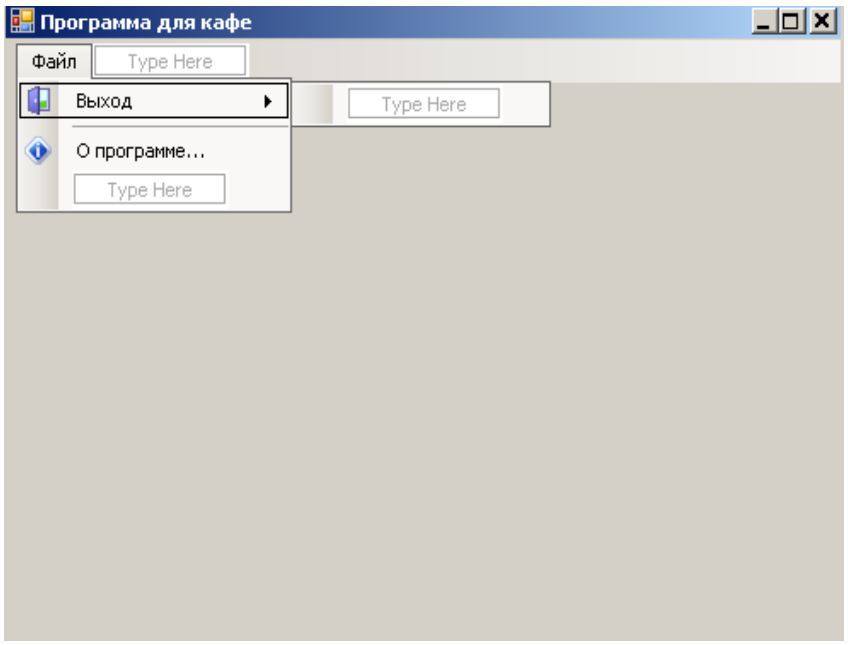


Для данных пунктов задайте пиктограммы через свойство **Image**.

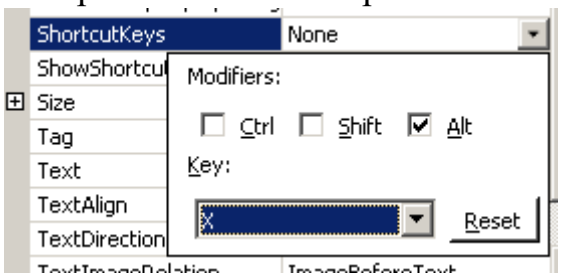
Сначала добавьте понравившиеся пиктограммы через **Project resource file** и кнопку **Import**:

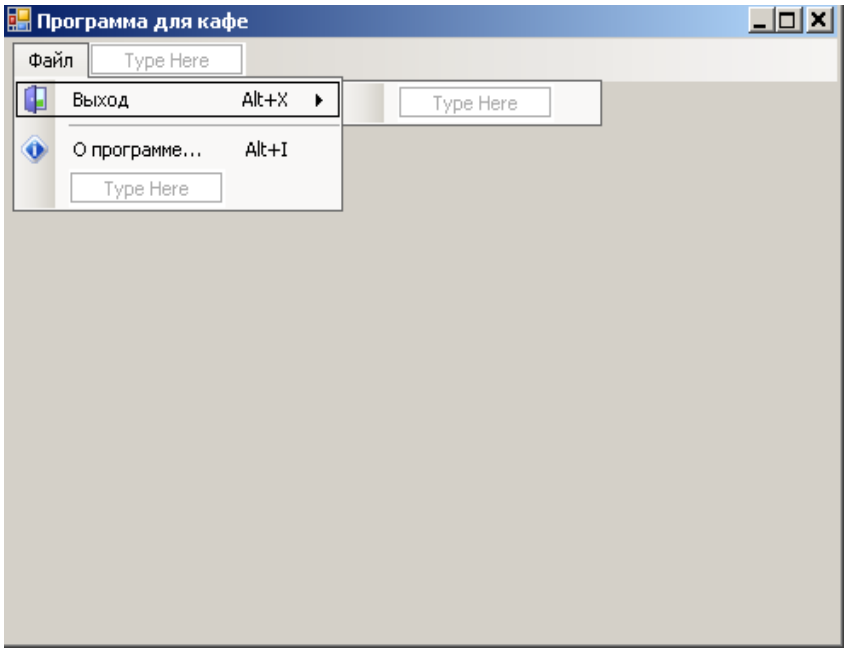


Затем укажите выбранную пиктограмму соответствующему пункту меню:



и горячие клавиши через свойство **ShotcutKeys**, например:





Запрограммируем данные пункты меню.

Нажмите два раза мышкой на пункте меню **Выход** и в обработчике события **Click** (клик мышью) пропишите, вызов метода **Close()**.

```
private void выходToolStripMenuItem_Click(object sender, EventArgs e)
```

```
        {
            Close();
        }
```

Примечание:

Рассмотрим код данного листинга.

Первая строка является объявлением метода, сгенерированным Visual Studio на языке C#.

Первое слово **private** определяет видимость метода, как внутренний, т.е. видимый только для членов класса Main_Form.

Второе слово **void** определяет тип данных возвращаемого значения (результата). Ключевое слово **void** перед именем функции, или в качестве аргумента функции (в скобках в конце строки), означает отсутствие соответствующего элемента.

Третье слово **выходToolStripMenuItem_Click** обозначает имя метода. За именем следует список передаваемых аргументов, заключенный в круглые скобки. Круглые скобки нужно использовать всегда, даже когда у функции нет параметров.

Тело функции заключается в фигурные скобки **{}**

Последняя строка – это вызов метода Close(), позволяющего закрыть форму нашего приложения (в данном случае главную форму):

Close();

C# чувствителен к регистру. В именах функций и переменных заглавные (прописные) буквы должны использоваться точно так же, как в их объявлениях.

Две косые черты, или слэши (//) - все, что следует до конца строки после двух идущих подряд косых черт (без пробелов, табуляций и т.п. между ними) рассматривается компилятором как комментарий и игнорируется.

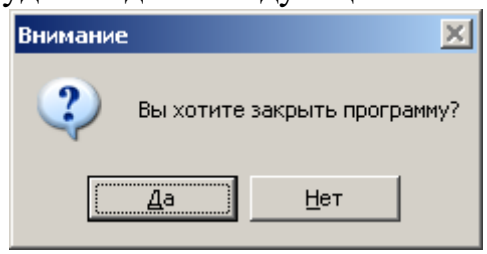
При добавлении в код нескольких строк комментариев используются символы в начале комментария /*, а в конце комментария */.

Данный метод **Close()** может вызываться не только в данном меню **Выход (выходToolStripMenuItem)**, но и когда пользователь нажмет на форме.

Запрограммируем для данного метода в событие (вкладка **Events**) **FormClosing** у **FormMain** вызов диалога, подтверждающего закрытие программы:

```
private void FormMain_FormClosing(object sender, FormClosingEventArgs e)
        {
            e.Cancel = MessageBox.Show("Вы хотите закрыть программу?",
"Внимание",      MessageBoxButtons.YesNo,      MessageBoxIcon.Question)      !=
DialogResult.Yes;
        }
```

В результате, выбирая пункт меню **Выход** или закрывая программу по ×, будет выдано следующее сообщение:



При нажатии Да, программа будет закрыта.

Примечание:

Для метода **MessageBox.Show()**

Перечисление MessageBoxButtons	
Константа	**Кнопки, отображаемые в окне MessageBox**
OK	OK
OKCancel	OK, Cancel
YesNo	Yes, No
YesNoCancel	Yes, No, Cancel
RetryCancel	Retry, Cancel
AbortRetryIgnore	Abort, Retry, Ignore

Перечисление MessageBoxIcon	
Константа	**Значок**
Asterisk, Information	
Error, Stop	
Exclamation, Warning	
Question	
None	Значок не отображается

Если в окне диалогового окна **MessageBox** имеется несколько кнопок, то для того, что бы определить, какую кнопку щелкнул пользователь, программа должна проанализировать значение, возвращенное методом **MessageBox.Show()**. Метод **MessageBox.Show()** может вернуть одно из нескольких значений перечисления **DialogResult**:

Перечисление DialoqResult	
Константа	**Кнопка, при щелчке которой возвращается эта константа**
Abort	Abort
Cancel	Cancel
Ignore	Ignore
No	No

None	Модальное диалоговое окно продолжает работать
OK	OK
Retry	Retry
Yes	Yes

Продолжаем программировать пункты меню.

Пункт **О программе…**:

```
private void оПрограммеToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("(C)ТУСУР,КИБЭВС,ФИО,группа,2015", "О программе",
MessageBoxButtons.OK, MessageBoxIcon.Information);
         }
```

Примечание:

Укажите свои ФИО и группу.

Примечание:

Согласно «Гражданскому кодексу Российской Федерации (часть четвертая)» к охраняемым результатам интеллектуальной деятельности и средствам индивидуализации (статья 1225), которым предоставляется правовая охрана (интеллектуальной собственностью), относят программы для электронных вычислительных машин (**программы для ЭВМ), базы данных.**

К объектам авторских прав относятся программы для ЭВМ, которые охраняются как литературные произведения (статья 1259).

Согласно статье 1271, правообладатель для оповещения о принадлежащем ему исключительном праве на произведение вправе использовать **знак охраны авторского права, который состоит из следующих элементов:**

- латинской буквы "С" в окружности или скобках;

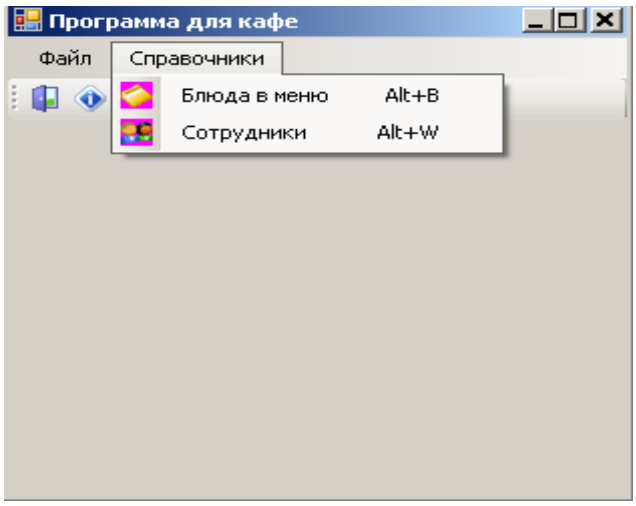
- имени или наименования правообладателя;

- года первого опубликования произведения.

Таким образом, знак охраны авторского права в данном случае может иметь вид:

(С)ТУСУР, КИБЭВС, ФИО, группа, 2015.

По аналогии определите пункты меню для справочников блюд, сотрудников:



4.2.2 Создание контекстного (локального, всплывающего) меню

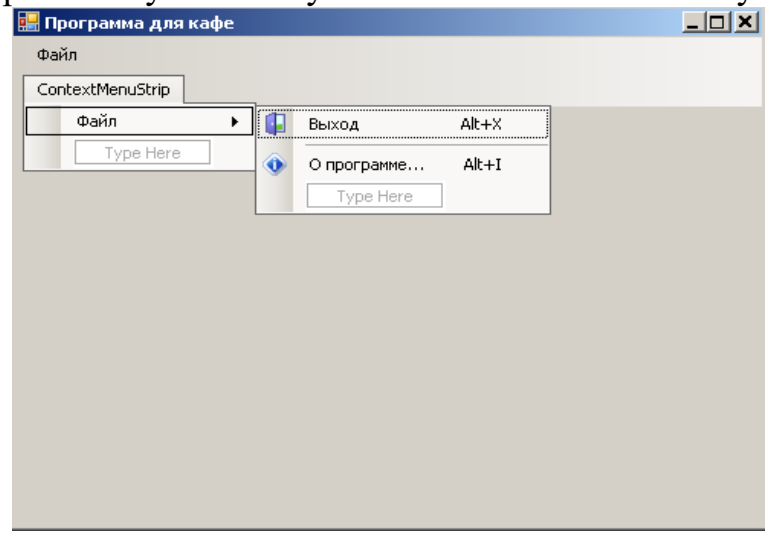
Контекстное (локальное, всплывающее) меню обычно появляется на экране при нажатии правой кнопки мышки на визуальных компонентах.

Для контекстного меню используется элемент **ContextMenuStrip** с закладки **ToolBox**.

Добавим в главную форму элемент для контекстного меню. Для этого откройте вкладку **ToolBox** и сначала щелкнете мышью на элементе **ContextMenuStrip** вкладки, а затем щелкнем мышью на форме.

Свойство **Name** для него укажите в **contextMenuStripMain**.

В дизайнере меню укажите пункты аналогично главному меню:



Для обработчиков данных пунктов меню через событие **Click** укажите соответственно обработчики:

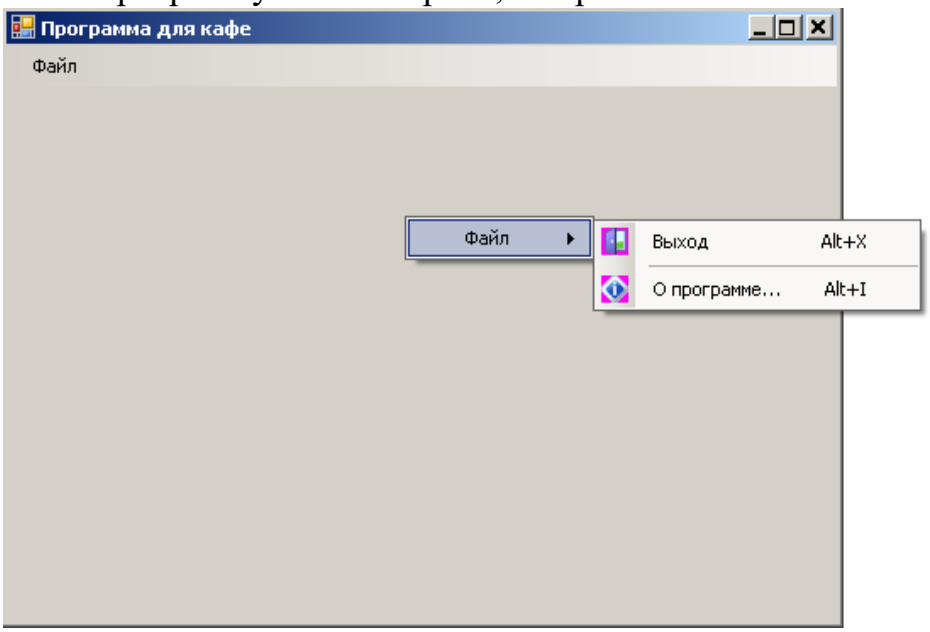
выходToolStripMenuItem_Click

оПрограммеToolStripMenuItem_Click

чтобы дважды данные действия не программировать.

Чтобы контекстное меню появлялось при нажатии правой кнопки мыши на форме, укажите в свойстве **contextMenuStrip** формы **FormMain** созданное **contextMenuStripMain**.

Запустите программу и посмотрите, как работает контекстное меню.



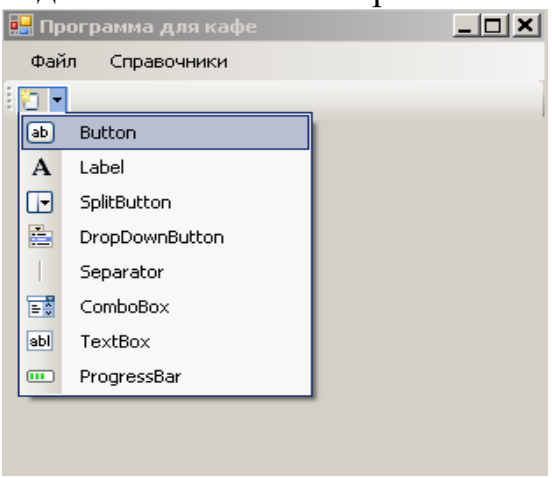
Аналогично проделайте для справочников блюд, сотрудников.

4.2.3 Создание панели инструментов

Для панели инструментов используется элемент **ToolStrip** с закладки **ToolBox**.

Разместите элемент **ToolStrip** на форму. Укажите для него свойство **Name** в **toolStripMain**.

В дизайне панели выберете **Button**:

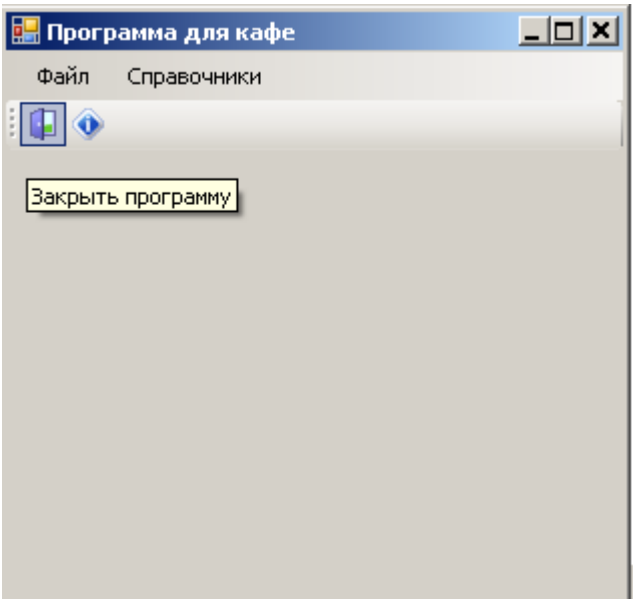


Для нее укажите аналогичную пиктограмму для выхода и всплывающую подсказку (через свойство **ToolTipText**).

В событии **Click** укажите для данной кнопки соответственно:

выходToolStripMenuItem_Click

Аналогично создайте кнопку для **О программе**.



Аналогично проделайте для справочников блюд и сотрудников.

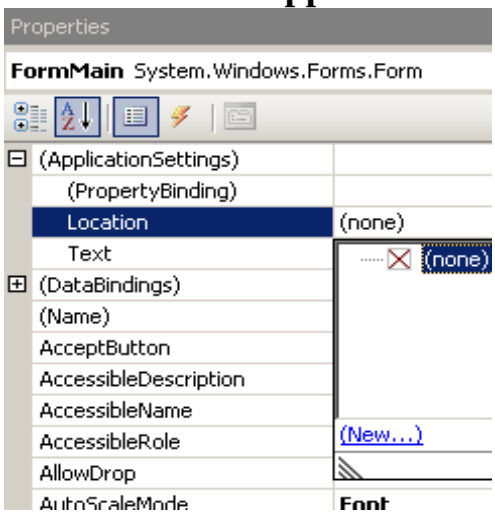
Запустите программу. Посмотрите, как отрабатывают созданные обработчики.

Закройте программу. Вернитесь в Visual Studio.

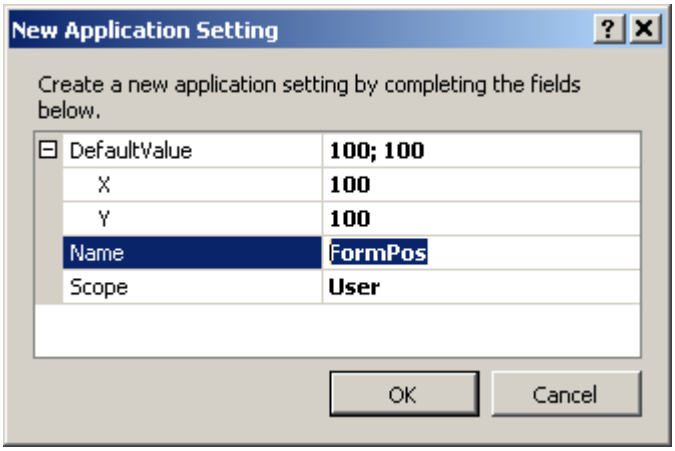
Примечание:

Существует возможность автоматического сохранения состояния большинства свойств элементов управления в конфигурационном файле приложения.

Например, для того, чтобы форма запущенного приложения сохранила свое положение на экране в конфигурационном файле и восстановила при следующем запуске свое положение на экране, необходимо изменить свойство **Location** в **ApplicationSettings** формы **FormMain**:

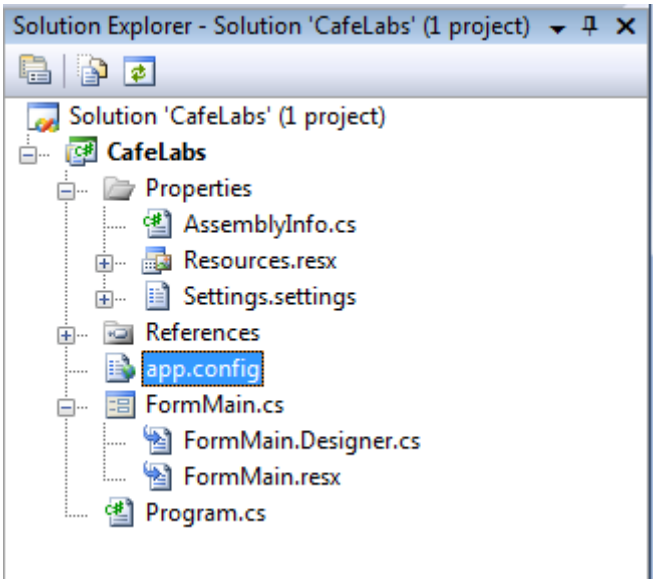


Нажмите **New** и укажите в появившемся диалоговом окне название данной настройки (**FormPos**), которая будет использована в конфигурационном файле, и значения позиции формы по умолчанию:



Нажмите ОК.

Теперь в составе проекта появился конфигурационный файл **app.config**



В проект при этом нужно добавить следующую строку кода в событии закрытия главной формы приложения:

```
private void FormMain_FormClosed(object sender, FormClosedEventArgs e)
        {
            Settings.Default.Save();
         }
```

При этом необходимо добавить пространство имен**:**

```
using CafeLabs.Properties;
```

В процессе работы приложения изменяемые настройки программы для которых указано **Scope** в **User** будут сохранены в конфигурационный файл, расположенный в домашней папке пользователя. Свойста и параметры имеющие признак **Scope** равный **Application** храняться в файле **CafeLabs.exe.config** расположенном рядом с исполняемым файлом. В данном примере в папке:
C:\LabsBD\CafeLabs\CafeLabs\bin\Debug\CafeLabs.exe.config

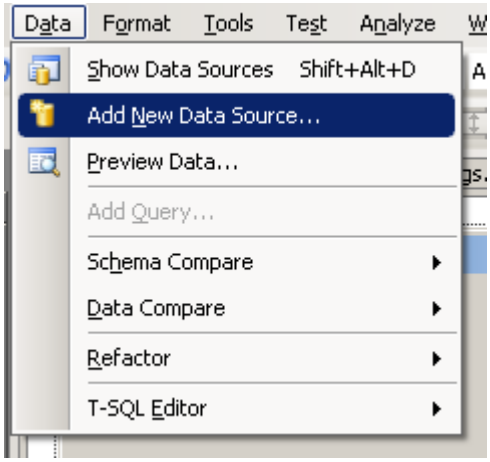
Запустите приложение. Проверьте настроенную возможность главной формы запоминать свое положение на экране.

4.3. Подключение к БД

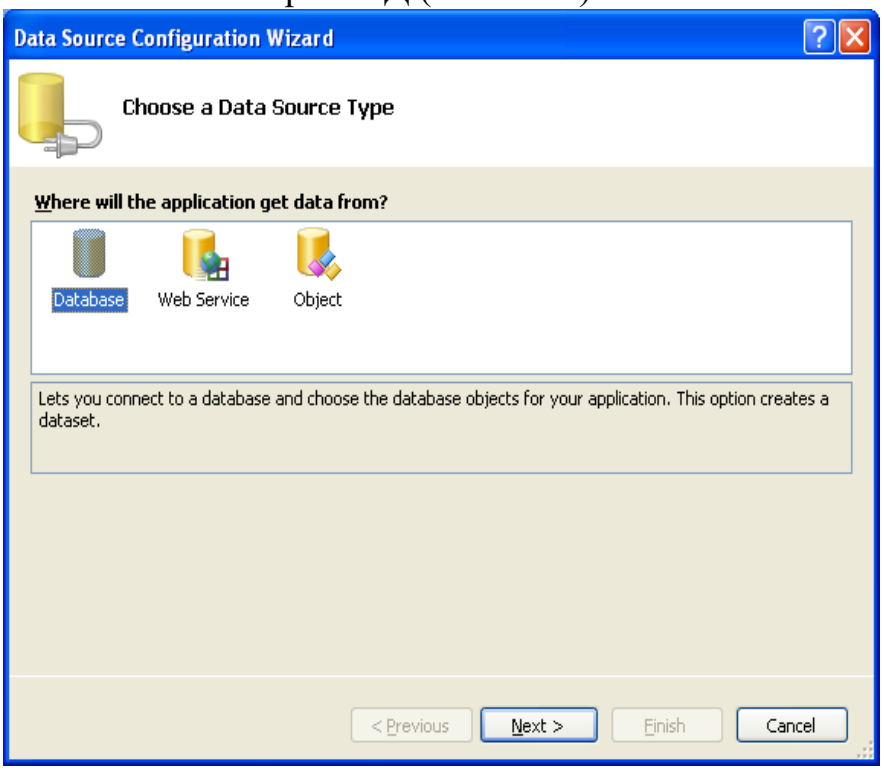
Программа для кафе должна позволять работать с данными базы данных.

Для этого необходимо подключиться к базе данных в приложении.

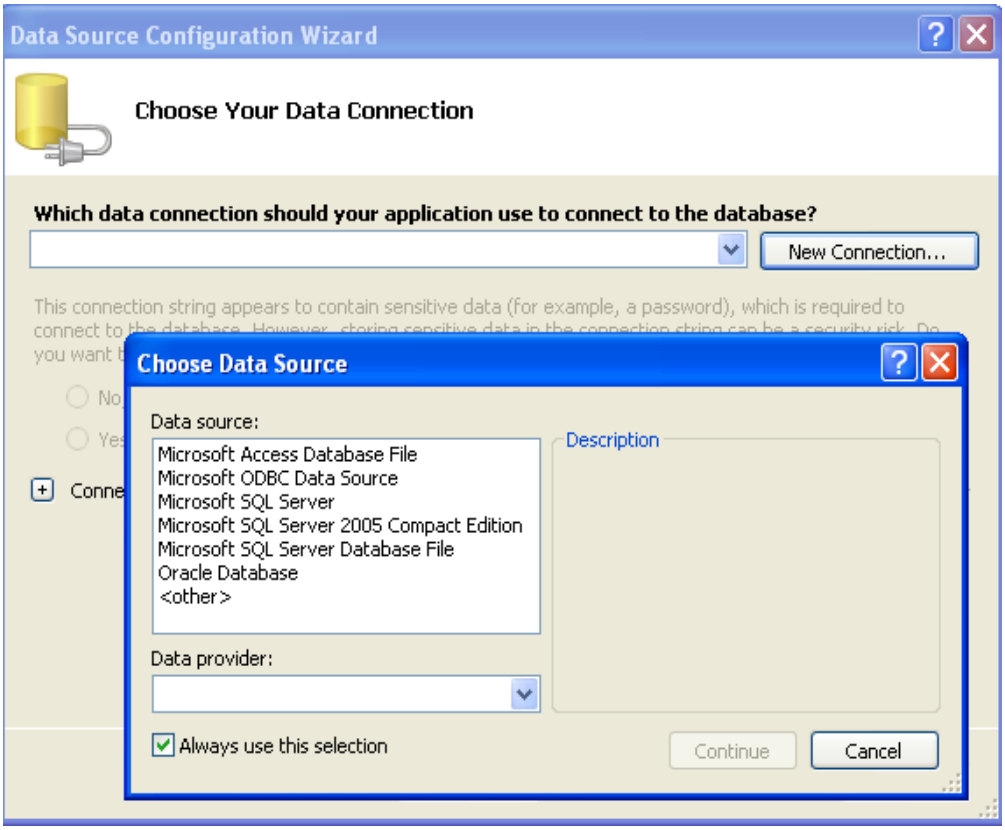
Выберете меню **Data\Add New Data Source:**



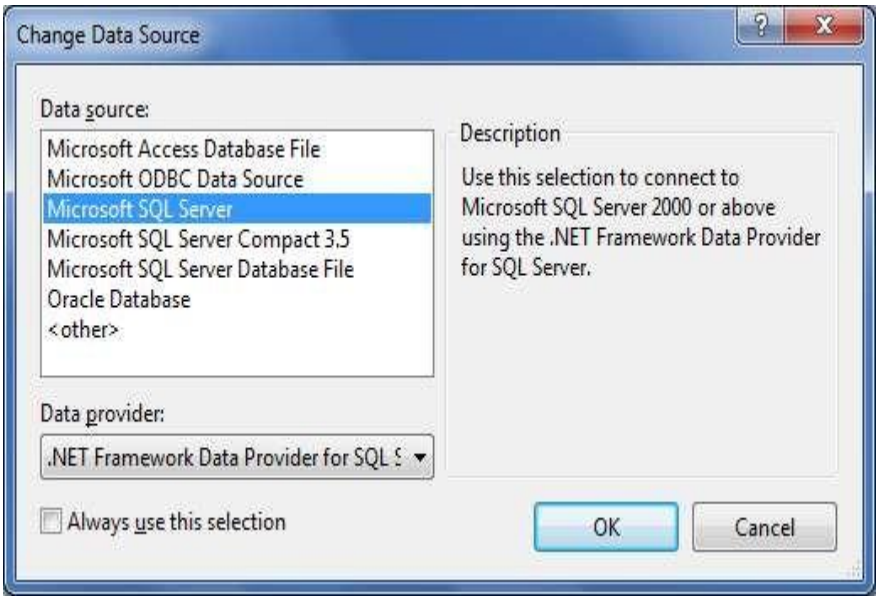
В диалоговом окне выберете БД (**Database**) и нажмите **Next**:



Выберете Новое соединение (**New connection**), после этого откроется диалоговое окно, в котором необходимо выбрать источник данных.



Для БД под управлением СУБД Microsoft SQL Server выберете **Microsoft SQL Server**:

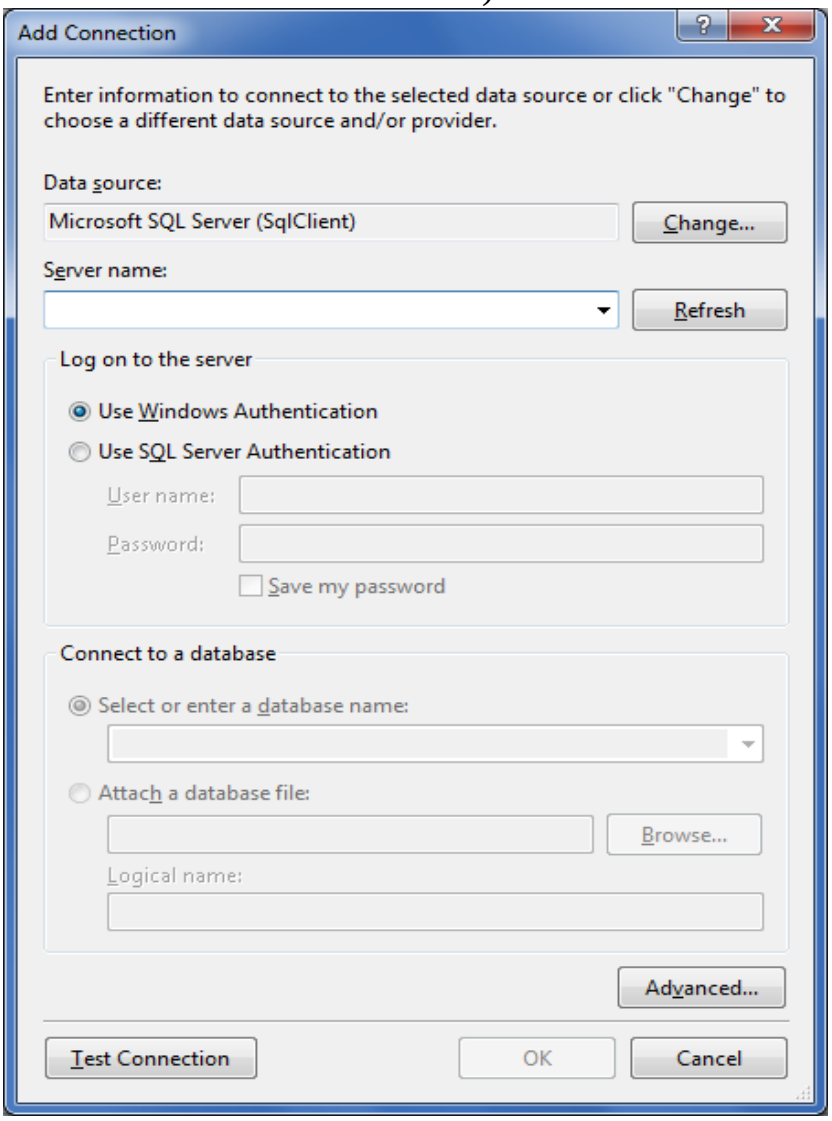


Если у Вас другая СУБД, то выберете необходимую СУБД.

Нажмите кнопку **ОК** для продолжения формирования источника данных.

Для СУБД Microsoft SQL Server:

Далее укажите имя сервера (**Server name**), режим необходимой аутентификации (**Log on to the server**) (см. лабораторную работу №1), имя базы данных (**Select or enter a database name**):

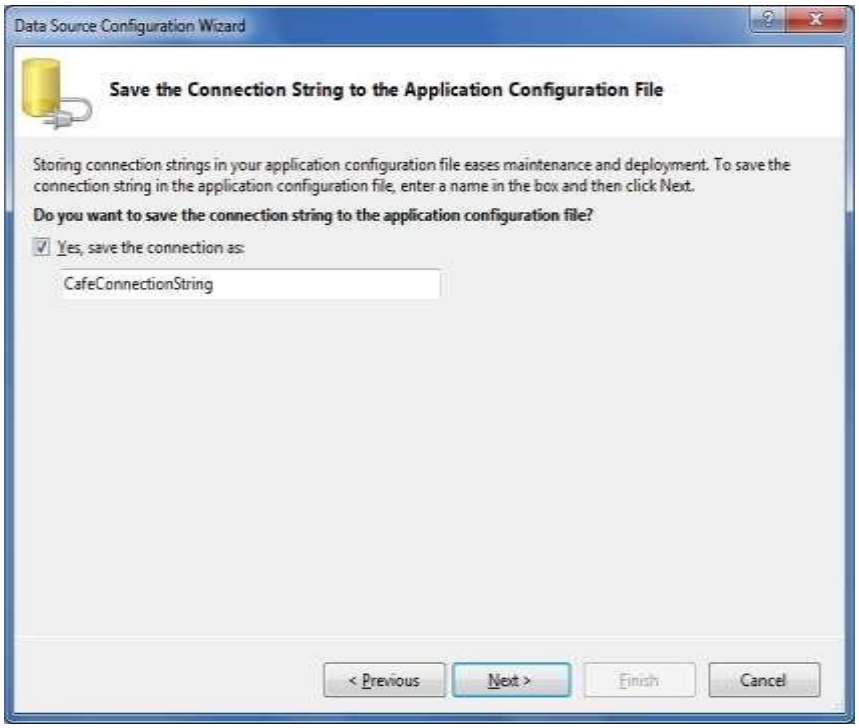


Можете нажать **Test Connection** для проверки соединения с БД.
Если соединение прошло успешно, то выдается сообщение:

Затем нажмите кнопку **ОК** и продолжите формировать источник данных.

Нажмите кнопку **Next**.

Укажите необходимость создания конфигурационного файла, так как путь к БД может меняться и чтобы не перекомпилировать программу при смене пути к БД, достаточно поменять путь в конфигурационном файле.

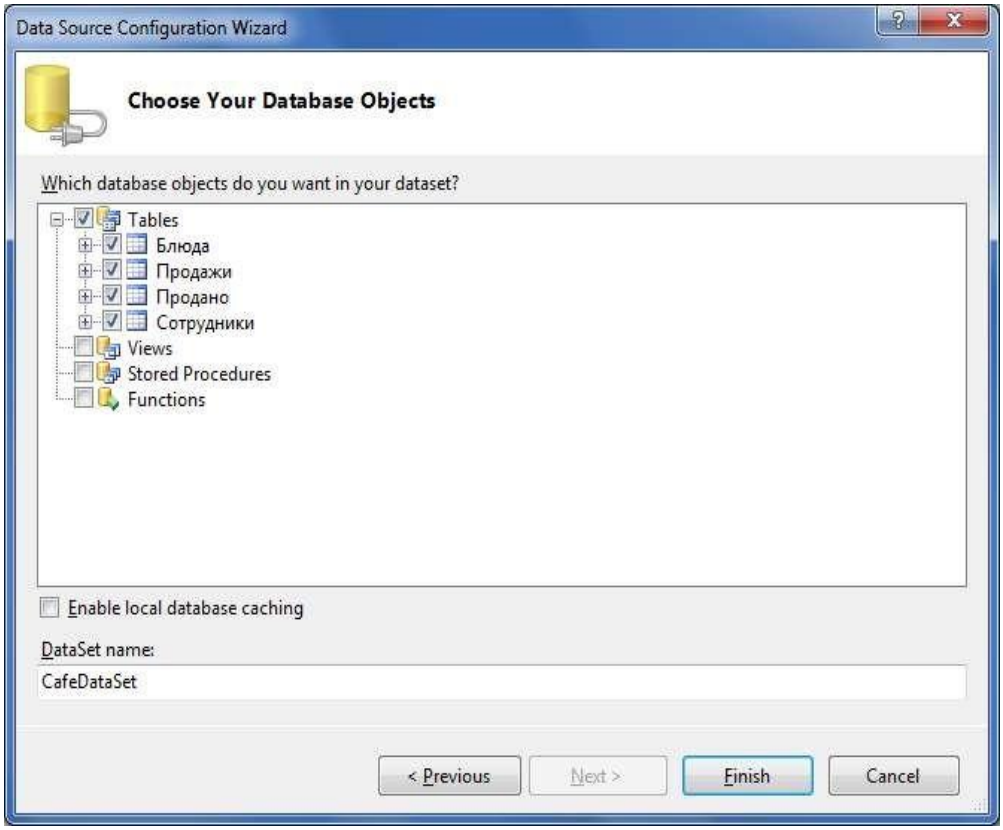


Нажмите **Next**.

Далее укажите объекты, которые будут помещены в создаваемый набор данных (**DataSet**).

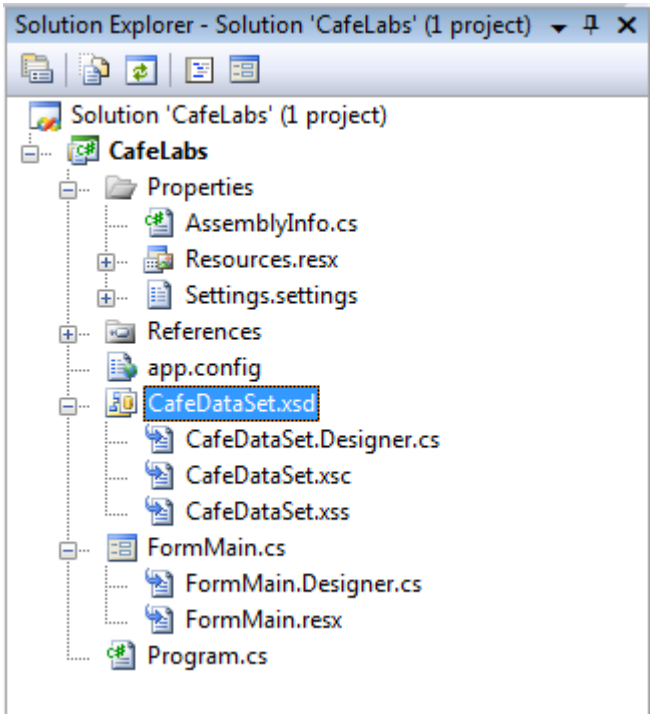
Наша программа будет работать со всеми таблицами БД, которые и добавим в набор данных.

Если нужны еще объекты БД, кроме таблиц, то необходимо указать и их.



Нажмите кнопку **Finish**.

В результате будет создан класс **CafeDataSet**, который добавляется в решение проекта:



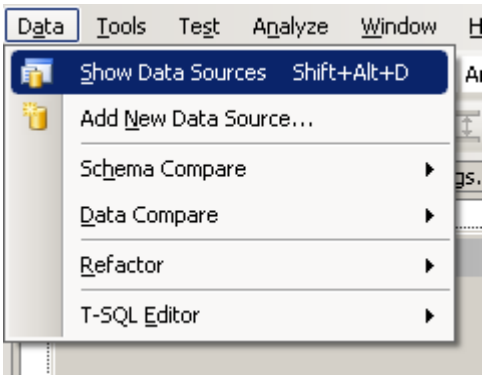
Примечание:

Конфигурационный файл находится в папке, где .exe файл и имеет имя (в данном примере):

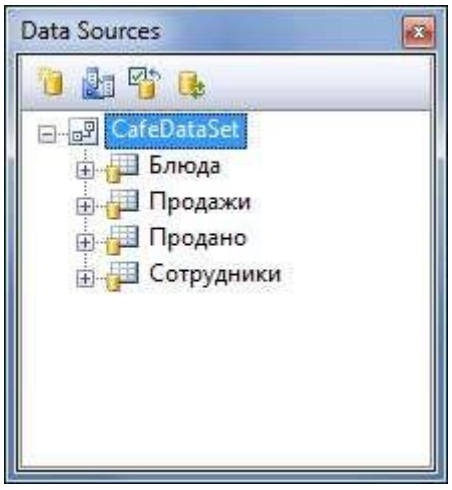
C:\LabsBD\CafeLabs\CafeLabs\bin\Debug\CafeLabs.exe.config

Таким образом, нами сформирован набор данных **DataSet**, являющийся источником данных (**DataSource**) в приложении.

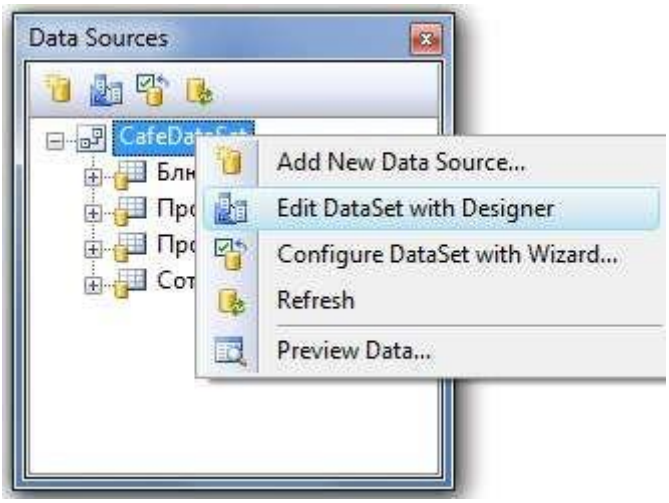
Выберете меню **Data**, а затем **Show Data Sources:**



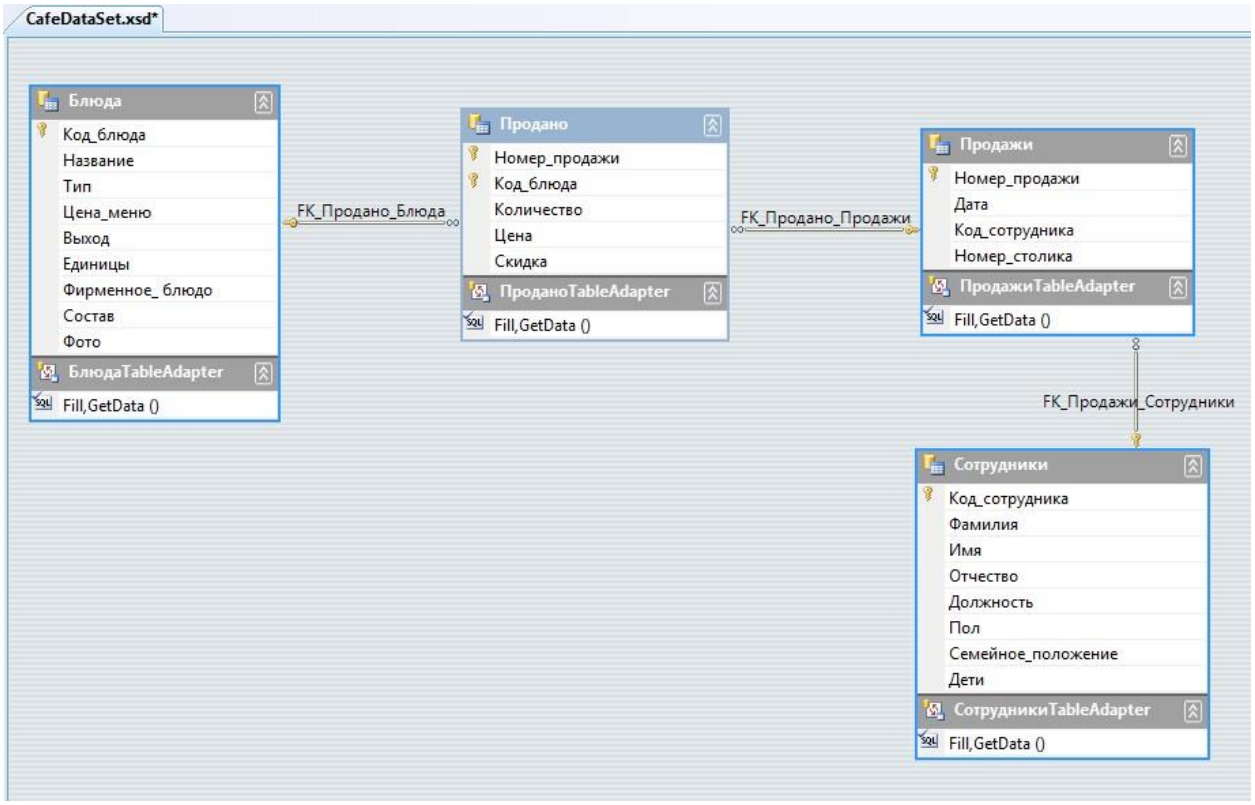
Для просмотра сформированного источника данных:



Если нажать правую кнопку мыши на сформированном наборе данных **CafeDataSet** и выбрать **Edit Dataset with Designer**:



То Вы сможете просмотреть дизайнер сформированного набора данных (или нажмите 2 раза мышкой на наборе данных **CafeDataSet.xsd**):



Как видно сформированы классы таблиц, адаптеры и методы **Fill** и **GetData**:

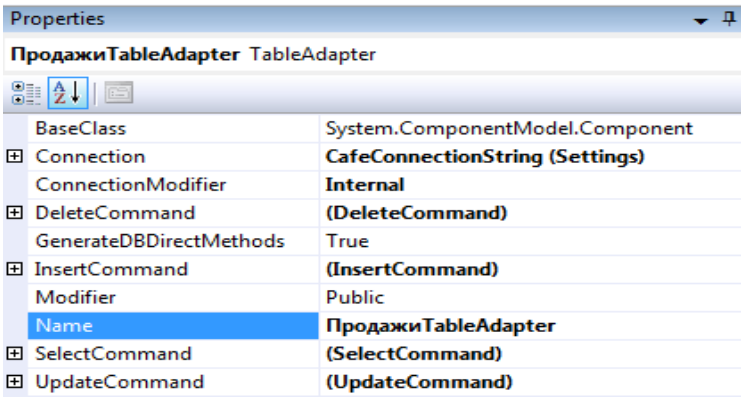
Блюда, Сотрудники, Продажи, Продано типа **DataTable**;

БлюдаTableAdapter, СотрудникиTableAdapter, ПродажиTableAdapter, ПроданоTableAdapter типа **DataAdapter**.

Как было сказано ранее, объекты типа **DataAdapter** позволяют проводить основные операции в наборе данных соответствующих объектов БД.

TableAdapter позволяет работать с данными в наборе данных соответствующих таблиц БД через соответствующие свойства.

Например, для **ПродажиTableAdapter**:



1) **SelectCommand** — выбрать данные.

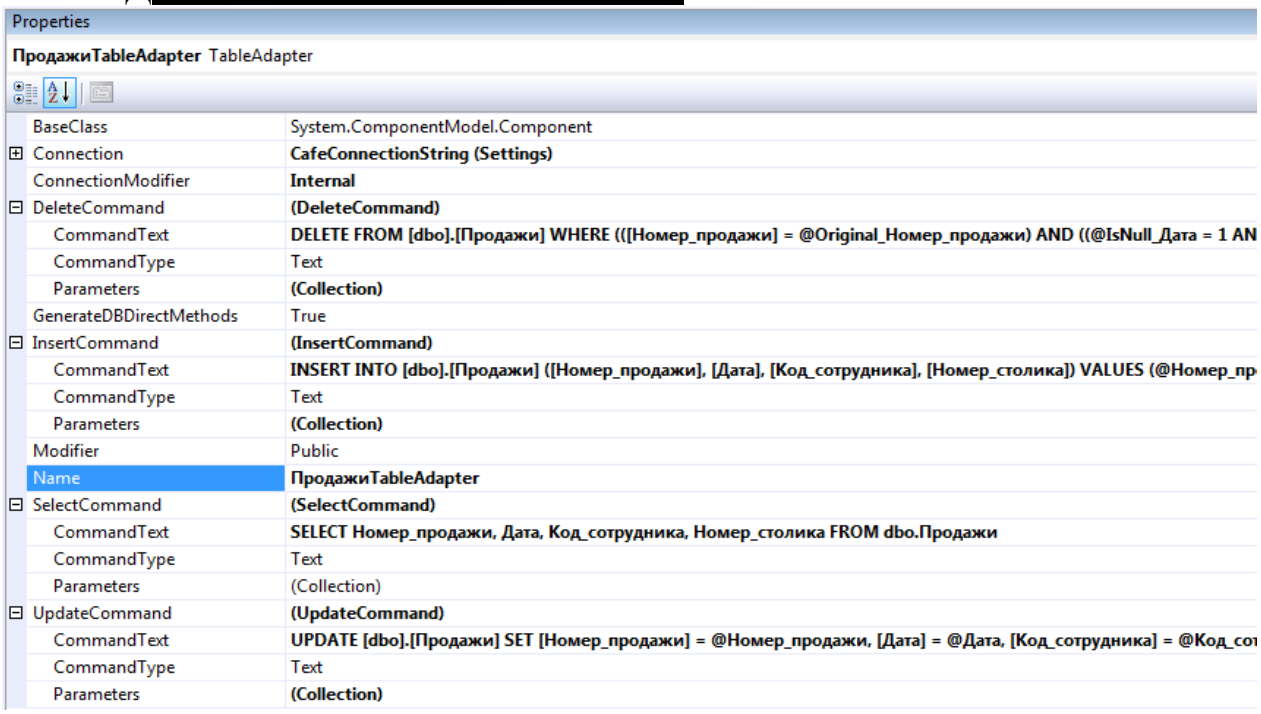
Метод **Fill** — заполняет данными набор данных соответствующих таблиц БД.

Он реализует выборку данных из соответствующих таблиц БД.

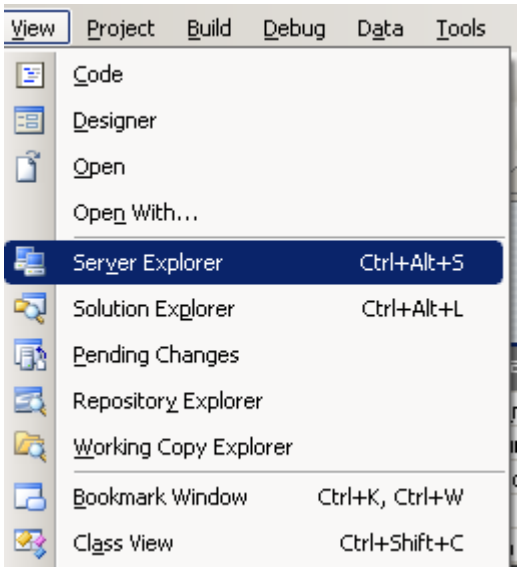
2) **InsertCommand** – добавить данные.
3) **DeleteCommand** — удалить данные.
4) **UpdateCommand** — изменить данные.

Например, для Продажи:

Для СУБД Microsoft SQL Server:



Открыв окно **Server Explorer** через меню **View**:



можно просмотреть сформированное соединение к БД.

Примечание:

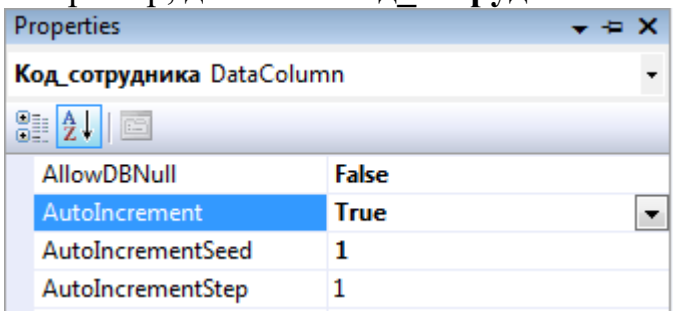
При формировании класса **CafeDataSet** необходимо учесть то, что первичные ключи таблиц Сотрудники, Продажи являются суррогатными и автоматически формируются (ключ со свойством автоинкремент) источником данных.

Это можно обеспечить, задав для ключевых колонок соответствующих таблиц следующие свойства:

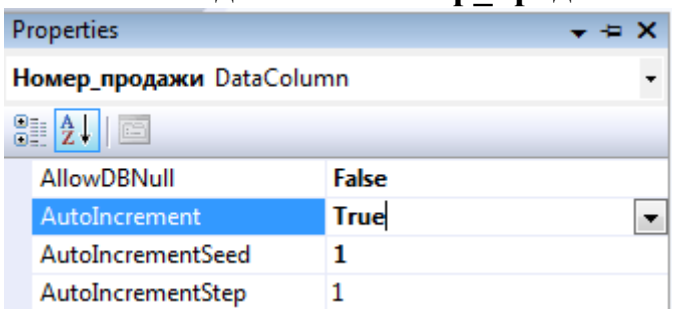
AutoIncrement, AutoIncrementSeed, AutoIncrementStep.

Столбец со свойством **AutoIncrement** равным **true** генерирует последовательность значений, начинающуюся со значения **AutoIncrementSeed** и имеющую шаг **AutoIncrementStep**.

Например, для поля **Код_сотрудника** таблицы **Сотрудники**:



По аналогии для поля **Номер_продажи** в **Продажи**:

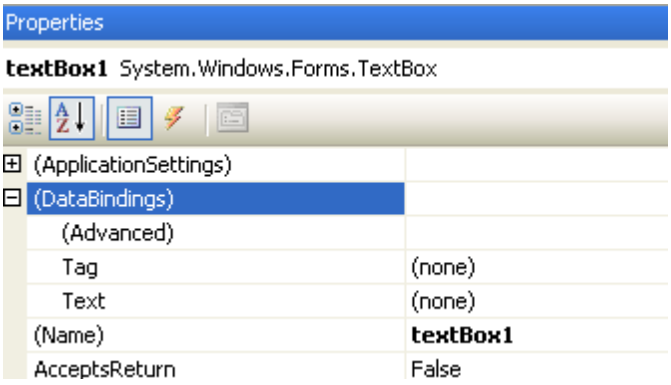


БД и ее объекты подключены в нашем приложении. Перейдем к визуальному оформлению БД в приложении.

4.4 Визуальное оформление БД в приложении

4.4.1 Основные особенности

Многие объекты с палитры **Toolbox** могут работать с данными БД через свойство **DataBindings**, например, для поля/строки ввода (**TextBox**):



По мере выполнения лабораторных работ будем добавлять особенности работы с данными объектами.

Привязывать элементы пользовательского интерфейса можно не только к таблично представленным данным. Практически любая структура данных может выступить в роли их источника – обычные объекты, массивы и т.д.

Для отображения информации из БД на элементах управления Windows Forms необходимо осуществить их заполнение из соответствующих столбцов и строк таблиц DataSet. Это можно сделать путем написания специального кода, который предназначен для заполнения соответствующих элементов управления или использовать механизм привязки данных к пользовательским интерфейсам .NET.

Windows Forms позволяют отображать данные путем привязки их элементов управления к источникам данных. Привязка данных обычно используется для отображения результатов поиска, детализации сводок, генерации отчетов и ввода данных.

Существует два типа привязки данных: простая и сложная. В случае простой привязки данных элемент управления привязывается к отдельному элементу данных. Простая привязка, в основном, используется с такими элементами управления, как поле ввода (**TextBox**), надпись (**Lable**) и др. При

сложной привязке данных элемент управления привязывается более чем к одному элементу данных - обычно к одному или нескольким столбцам в нескольких строках результирующего набора строк. К элементам управления, поддерживающим сложную привязку данных, относятся список (**ListBox**), выпадающий список (**ComboBox**), список данных (**DataList**) и табличный элемент (**DataGridView**).

В технологии привязки элементов управления Windows Forms используется абстрактный класс BindingManagerBase. Данный класс синхронизирует все элементы управления Windows Forms (т. е. объекты Binding), привязанные к одному источнику данных, позволяя отображать информацию об объекте в источнике данных - например, о строке в локальной таблице.

Класс BindingContext используется для создания экземпляра объекта BindingManagerBase, и возвращается при этом объект CurrencyManager или PropertyManager в зависимости от типа источника данных.

Компонент BindingSource

Компонент BindingSource является универсальным связующим звеном в Windows Forms. Он существенно облегчает задачу привязки данных.

Компонент BindingSource можно рассматривать как типизированную коллекцию элементов, при этом тип источника данных может быть задан различными способами.

Компонент BindingSource достаточно сложен и предназначен для решения различных задач. Он может применяться как для простой, так и для сложной привязки данных, в качестве одиночного или списочного источника данных, поддерживает фильтрацию, уведомления при изменении списка объектов, транзакционное добавление новых элементов, расширенную поддержку во время разработки и многое другое.

BindingSource представляет собой промежуточный слой между источником данных и соответствующим элементом управления (controlom), к нему привязанным.

Допустим, у нас есть Label, TextBox и ComboBox, привязанные к таблице (DataTable). Предположим, что для обновления информации был создан и заполнен данными из БД новый DataTable. Встает несложная, в общем то, задача – сменить у всех трех controlов источник данных. Теперь можно изменить свойства DataSource/DataMember единственного BindingSource, а не у всех controlов.

Покажем, как, например, можно отобразить данные БД в приложении. На примере БД для кафе.

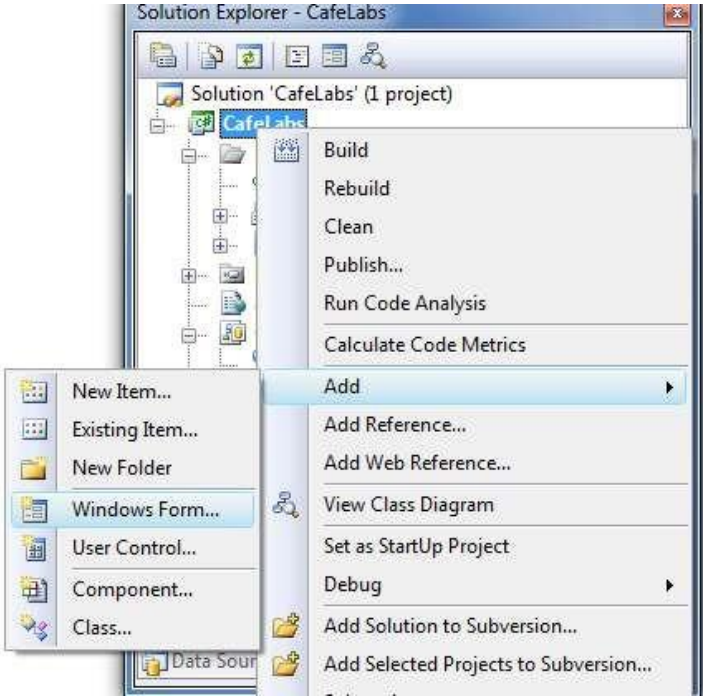
4.4.2 Отображение данных в виде Details

Отобразим данные по блюдам в приложении.

Будем показывать информацию по блюдам в отдельной форме.

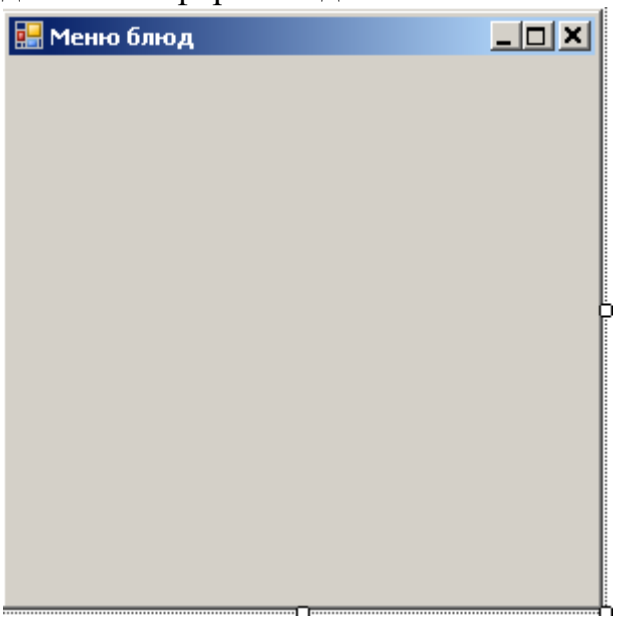
Добавим новую форму в проект для отображения данных по блюдам в меню.

Для этого нажмите на проекте правую кнопку мыши и выберете пункт **Add\Windows Forms**:



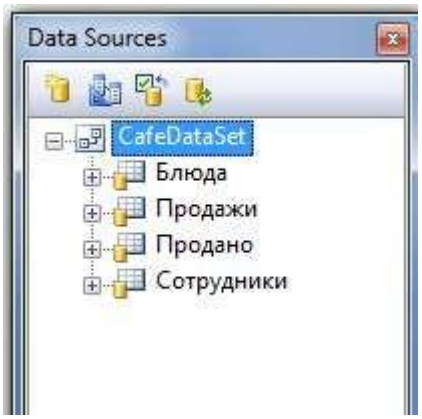
Задайте для новой формы имя **FormDishesList**.

Свойство **Text** для новой формы задайте в **Меню блюд**.

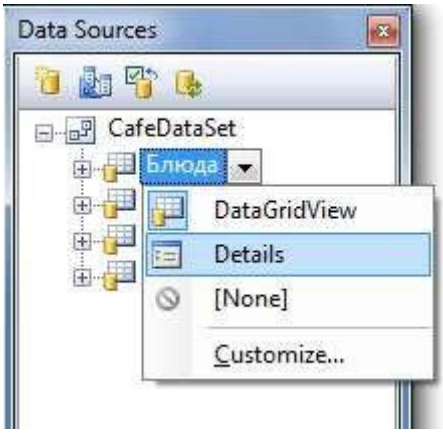


На данной форме будем отображать данные по списку блюд в меню.

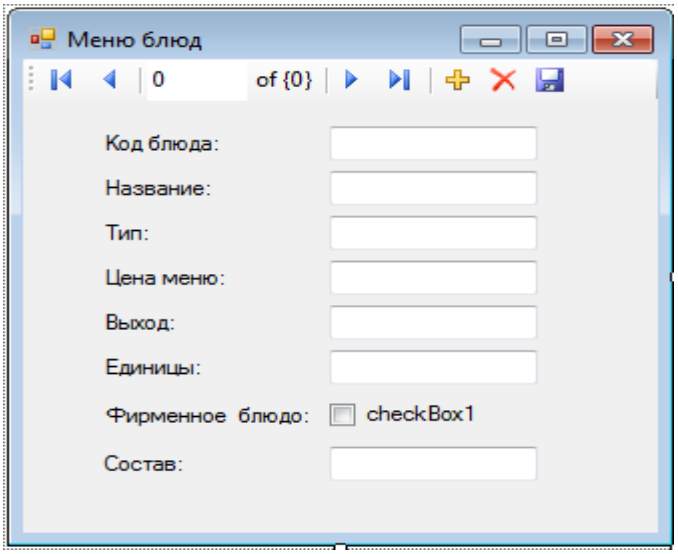
Откройте источник данных:



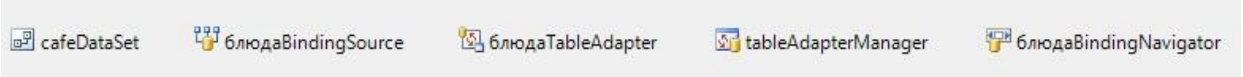
Выберете таблицу **Блюда** и форму представления данных **Details** (каждое поле в отдельном компоненте):



И удерживая левую кнопку мыши, положите на форму для меню блюд (**FormDishesList)**. В результате получится форма:

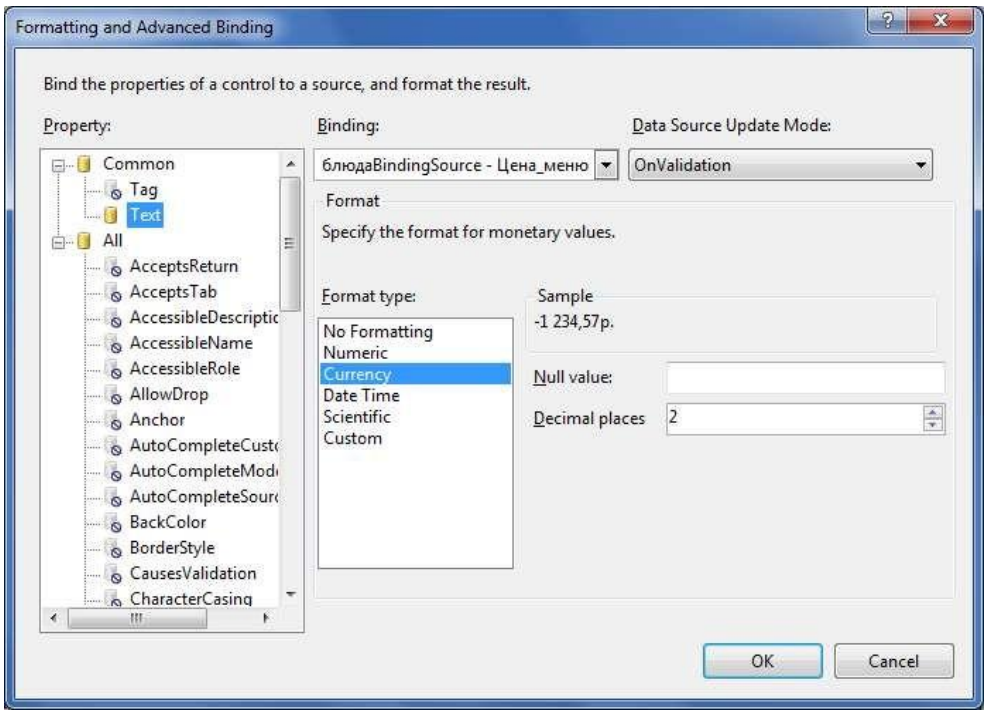


И создадутся объекты:



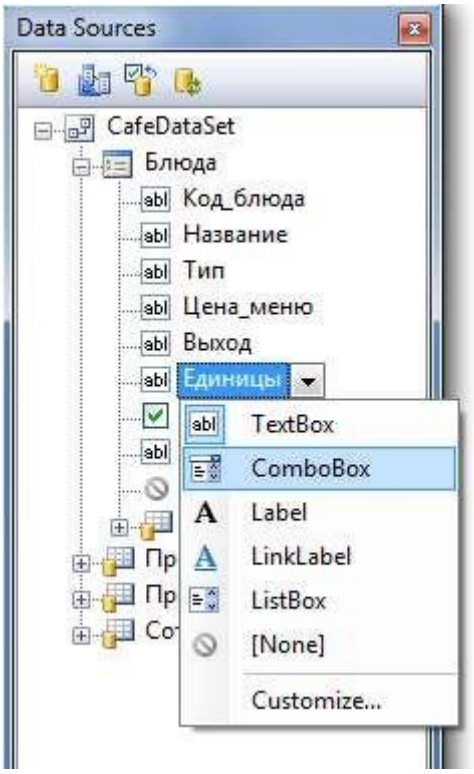
Продолжим настраивать данную форму.

Для данной БД **Цена_меню** должна быть выражена в денежном формате. Для данной настройки укажите в свойстве **Advanced** в **DataBindings** у **цена_менюTextBox**:

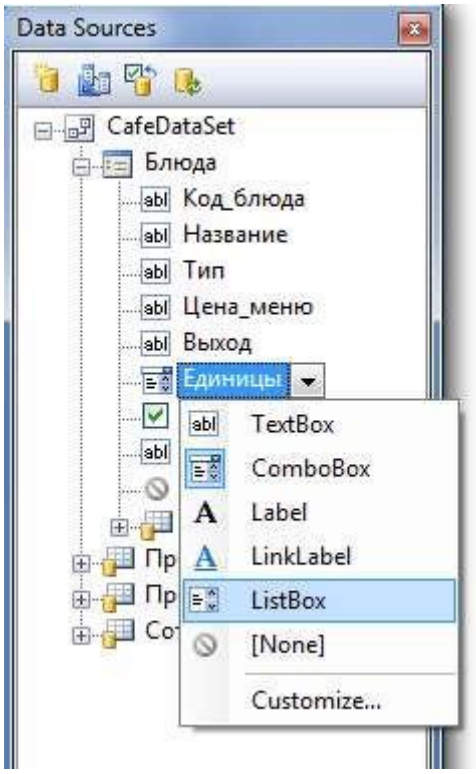


Для поля **Единицы** допустимы значения "гр." или "мл." или "шт." Для удобства пользователей положим или ComboBox (выпадающий список) или ListBox (список).

Для этого укажите в источнике данных для **Единицы**:

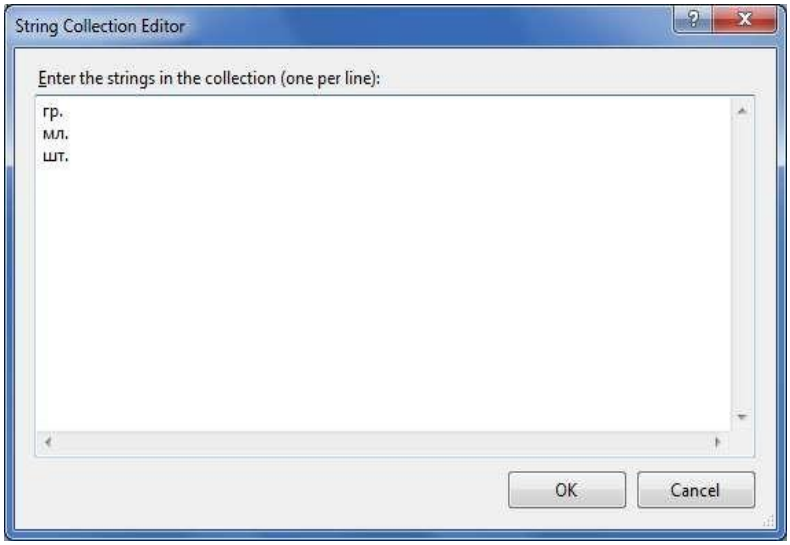


Или



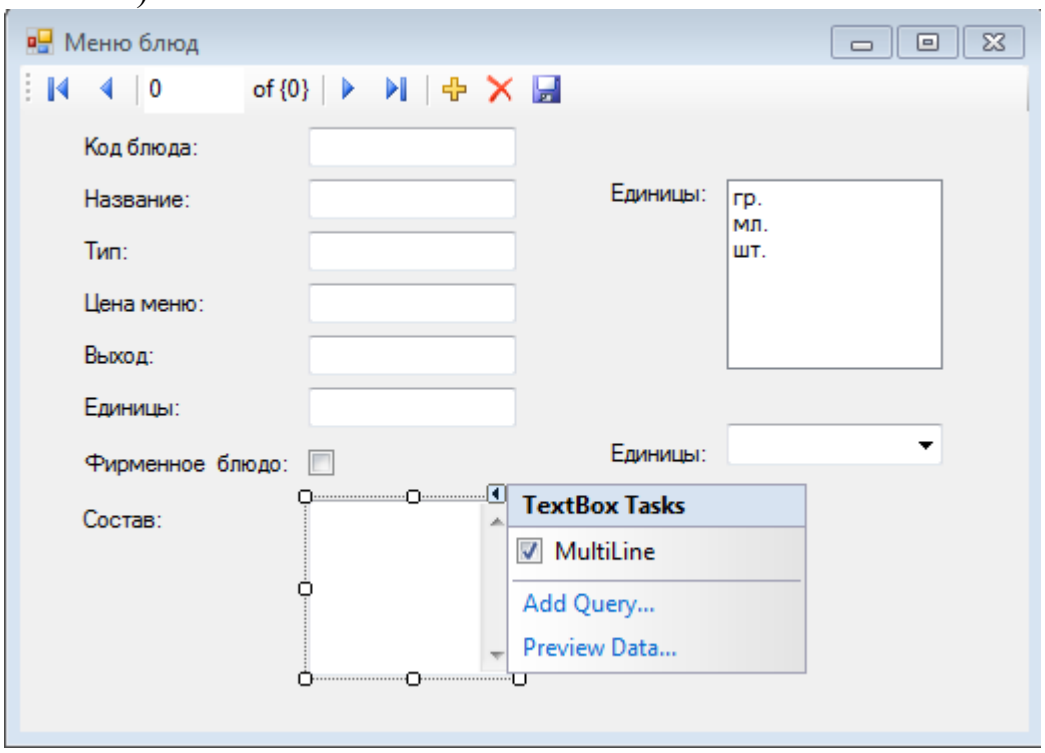
В данном примере для демонстрации разместим оба элемента.

В свойстве **Items** для них укажите:

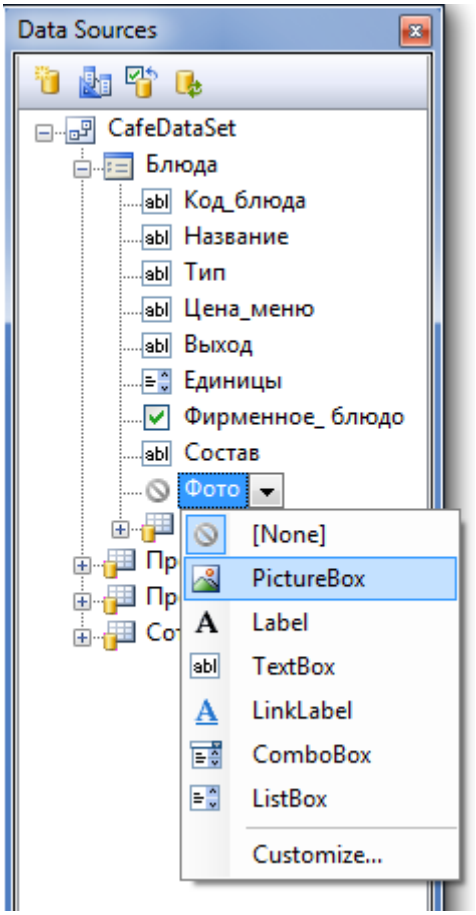


В свойстве **Text** у **фирменное_блюдоCheckBox** укажите пустую строку.

Поле **Состав** может хранить длинный текст. Чтобы отобразить значения в нем несколькими строками укажите свойство **MiltiLine** элемента **составTextBox** и растяните его на форме для удобства. Также для удобства установите свойство **ScrollBar** (скролинг) например, в **Vertical** (вертикальный).

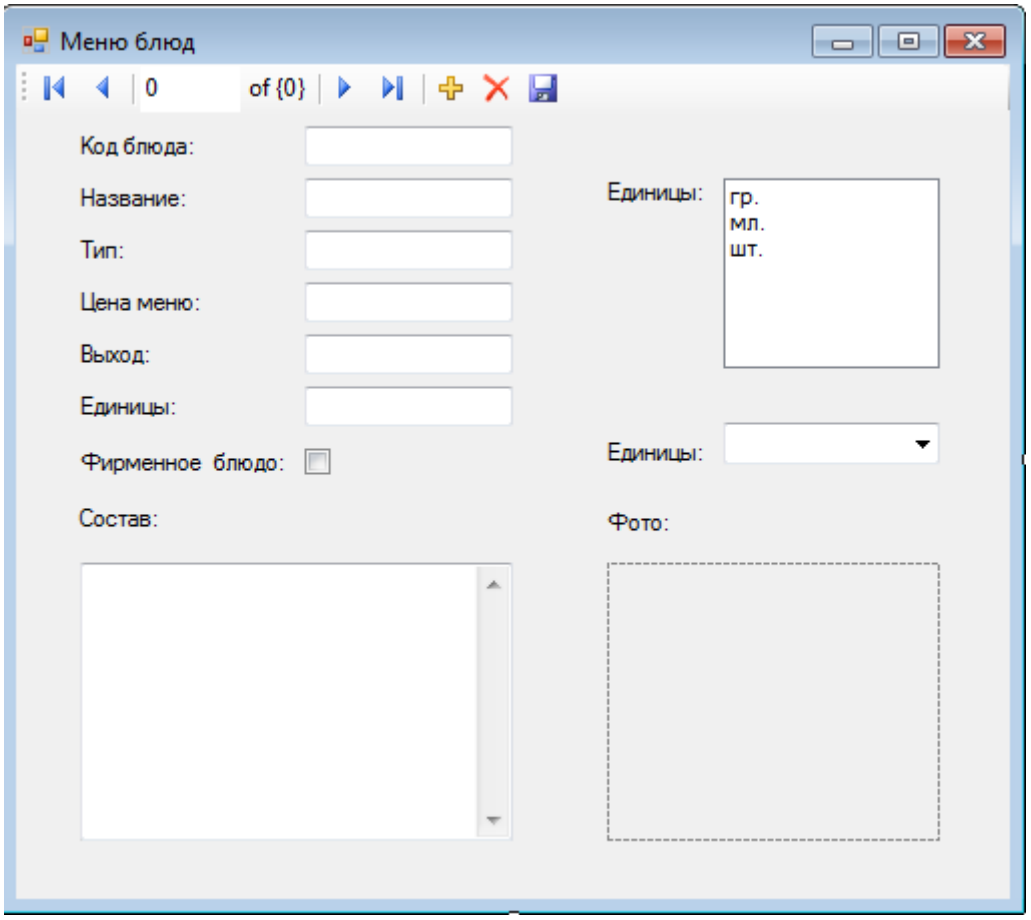


Поле **Фото** может хранить фотографии готового блюда. Для их отображения укажите для поля Фото в наборе данных тип элемента **Picture Box:**



И разместите его на форме.

Вид формы имеет вид:



Для **блюдаBindingNavigator** измените на русский язык текст всплывающих подсказок через свойство **Text** для выбранного элемента навигатора. Например, для кнопки с изображение дискеты **блюдаBindingNavigatorSaveItem** укажите свойство **Text** в **Сохранить**.

И далее для остальных кнопок навигатора согласно их смыслу:

	Первая запись, в начало
	Предыдущая запись, назад
	Следующая запись, вперед
	Последняя запись, в конец
	Добавить запись
	Удалить запись
	Сохранить

Для кнопки **bindingNavigatorCountItem** укажите свойство **Text** в **из {0}**

Примечание:

Обратите внимание, что все элементы управления на форме работают через **блюдаBindingSource** (свойство **DataBindings** элементов).

Покажем через главную форму данную форму для блюд.

Перейдите на главную форму и в событии **Click** для пункта меню **Блюда в меню** пропишите:

```
private void блюдаВМенюToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FormDishesList fd = new FormDishesList();
            fd.Show();

         }
```

Запустите программу и посмотрите, как можно работать с данными через данную форму.

Примечание:

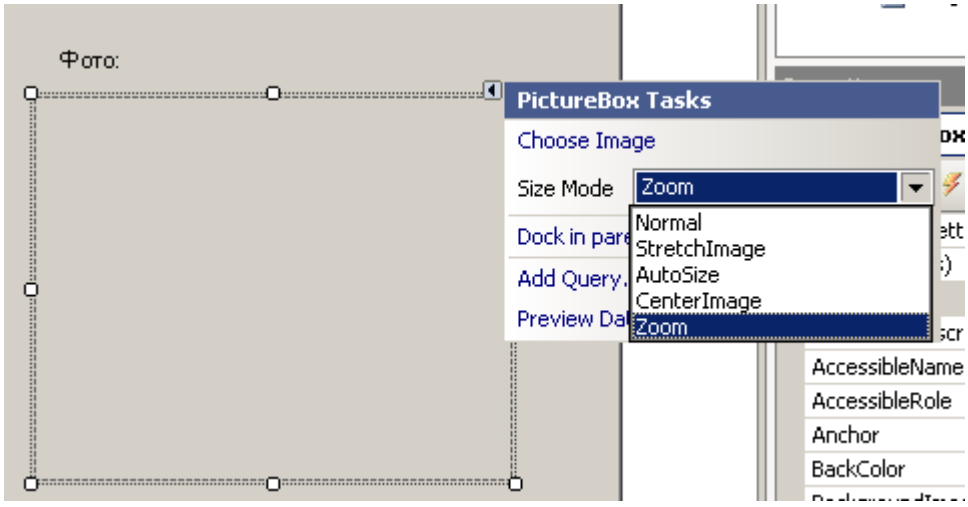
Не забывайте нажимать кнопку Сохранить для сохранения данных.

Закройте исполняемый файл. Перейдите на форму **FormDishesList**.

4.4.3 Работа с графическим полем

В нашей БД есть поле **Фото** для хранения фотографий блюд.

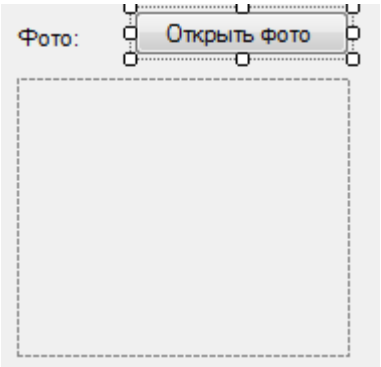
Чтобы фотография помещалась в размер **фотоPictureBox** для него укажите:



Для удобства загрузки фотографии блюда из файла разместите рядом с **фотоPictureBox** с палитры **Toolbox** компонент **button** для которого укажите свойства:

- **Name** в **buttonOpenPhoto**;

- **Text** в **Открыть фото**.



Для загрузки фото из файла положите на форму с палитры Toolbox диалог **openFileDialog1**. Задайте ему свойство **Name** в **openFileDialogPhoto.**

Запрограммируем нажатие кнопки **buttonOpenPhoto** для загрузки фото в **фотоPictureBox**:

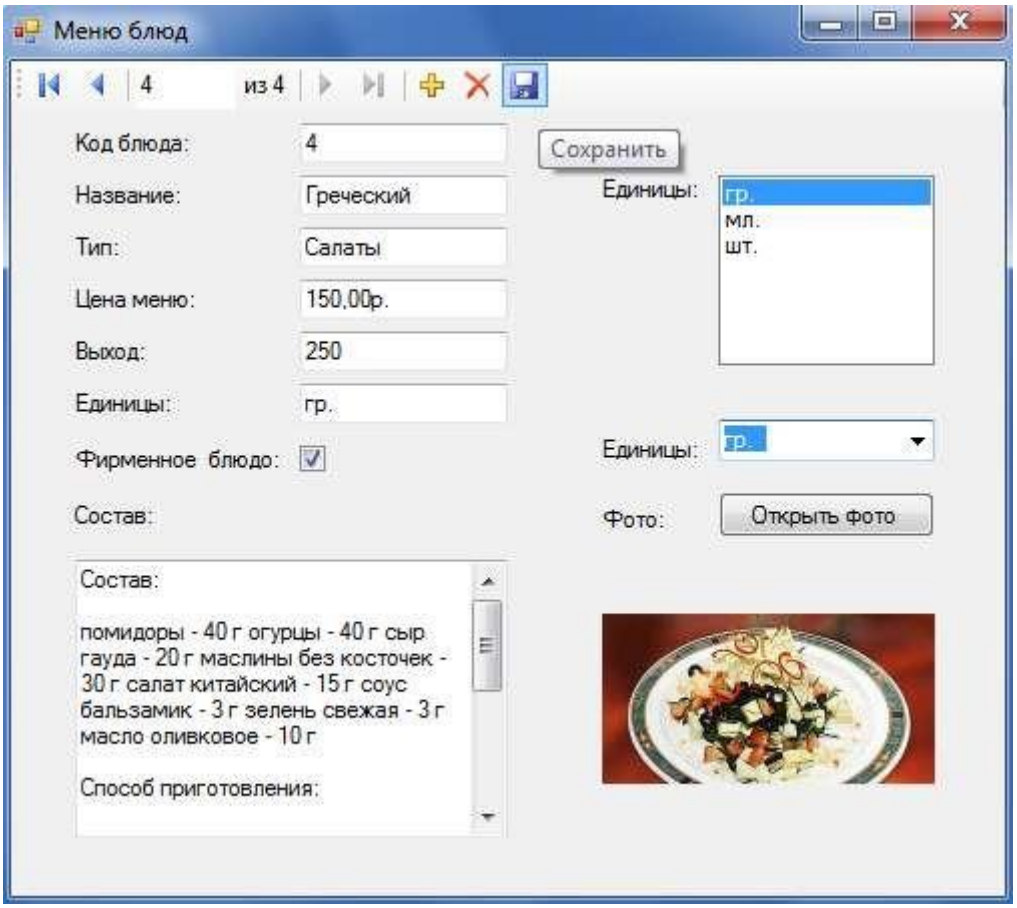
```
string fileImage = "";
        private void buttonOpenPhoto_Click(object sender, EventArgs e)
        {
            openFileDialogPhoto.Title = "Укажите файл для фото";
            if (openFileDialogPhoto.ShowDialog() == DialogResult.OK)
            {
                fileImage = openFileDialogPhoto.FileName;
                фотоPictureBox.Image = new
Bitmap(openFileDialogPhoto.FileName);

            }
            else fileImage = "";

         }
```

Или с конструкцией **try…catch**:

```
string fileImage = "";
        private void buttonOpenPhoto_Click(object sender, EventArgs e)
        {
            openFileDialogPhoto.Title = "Укажите файл для фото";
            if (openFileDialogPhoto.ShowDialog() == DialogResult.OK)
            {
                fileImage = openFileDialogPhoto.FileName;
                  try
                  {

                    фотоPictureBox.Image = new
Bitmap(openFileDialogPhoto.FileName);
                  }
                  catch
                  {
                    MessageBox.Show("Выбран не тот формат файла", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                  }

            }
            else fileImage = "";

         }
```

Конструктор класса **Bitmap** позволяет загрузить файлы следующих форматов: **BMP, GIF, EXIG, JPG, PNG, TIFF**.

Запустите программу, посмотрите, как можно работать с данными через данную форму:



4.4.4 Шаблон проектирования «Одиночка» (англ. Singleton)

Примечание:
Для отображения формы используются два основных метода:
метод Show() – возможен переход между формами;
ShowDialog() – открывает форму в модальном режиме, т.е. невозможно переключиться на другую форму, пока форма, показываемая по ShowDialog() открыта.

Перейдите на главную форму и посмотрите событие **Click** для пункта меню **Блюда в меню**:

```
private void блюдаВМенюToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
    FormDishesList fd = new FormDishesList();
    fd.Show();

}
```

Запустите программу и посмотрите, как можно работать с данными через данную форму.

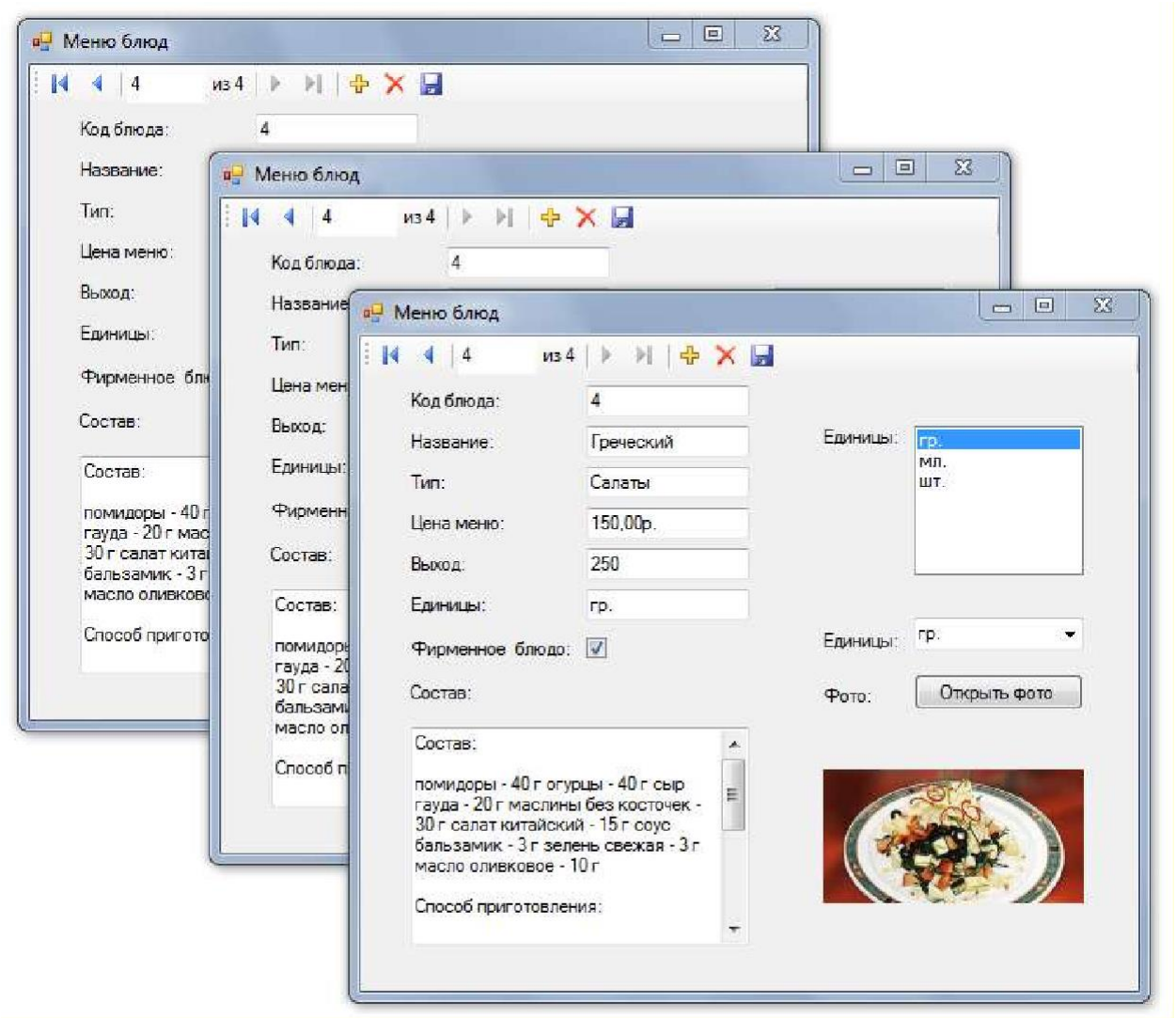
Обратите внимание, что в данный момент нами используется метод **Show()** и возможно переходить от открытой формы для блюд к главной форме и наоборот.

Внимание:

выберете в запущенной программе еще раз данный пункт меню **Блюда в меню**, не закрывая уже открытую форму **Блюда в меню**.

Посмотрите, что создалась еще одна такая же форма.

Таким образом, пользователь может накопить достаточное количество отрытых одних и тех же форм.



Закройте программу и вернитесь в Visual Studio.

Пропишите в обработчике для данного обработчика вместо метода **Show()** метод **ShowDialog():**

```
private void блюдаВМенюToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FormDishesList fd = new FormDishesList();
            fd.ShowDialog();
         }
```

Запустите программу.

Посмотрите, как можно работать с данными через данную форму.

Внимание:

обратите внимание, что сейчас нельзя перейти в главную форму, пока форма для меню блюд не будет закрыта.

Однако часто пользователям требуется открыть несколько форм одновременно с возможностью переключения между ними. Например, в нашем случае посмотреть по коду блюда его название, тип и другую информацию, не закрывая при этом форму при оформлении продажи.

Получается, что в этом случае лучше использовать метод **Show()**. Но, если пользователь несколько раз выберет соответствующий пункт меню, то несколько одинаковых форм и откроется. Чтобы этого избежать и при выборе этого же пункта переходить к этой же открытой форме можно использовать **шаблон проектирования «Одиночка» (**англ. **Singleton).**

Шаблон проектирования (паттерн, англ. **design pattern**) - это многократно применяемая архитектурная конструкция, предоставляющая решение общей проблемы проектирования в рамках конкретного контекста и описывающая значимость этого решения. Паттерн - это описание или образец для того, как решить задачу, таким образом, чтобы это можно было использовать в различных ситуациях.

Шаблон проектирования «Одиночка» (анг. **Singleton) -** это типичное решение гарантирует, наличие только одного экземпляра класса, и предоставляет ссылку на этот единственный экземпляр (объект).

Реализуем шаблон проектирования «Одиночка» (англ. Singleton) на примере формы FormDishesList.

Для этого в коде формы **FormDishesList** пропишите:

```
private static FormDishesList f;

        public static FormDishesList fd
        {
            get
            {
                if (f == null || f.IsDisposed) f = new FormDishesList();
                return f;
            }
        }
```

Как видно из данного листинга, теперь мы не на пункте меню для блюд каждый раз создаем **fd =new FormDishesList()**, а только в том случае, если она была пустой (**null**) или освобождена в памяти (**IsDisposed**).

Перейдите на главную форму и для сооотвествующего пункта меню для блюд укажите тогда новый код:

```
private void блюдаВМенюToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FormDishesList.fd.Show();
         }
```

Запустите программу и посмотрите полученную возможность.

Закройте программу и вернитесь в Visual Studio.

Чтобы при выборе пункта меню для блюд каждый раз возвращать фокус на уже открытую форму (повторно мы ее не создаем) объявим свой метод **ShowForm()** на форме **FormDishesList:**

Для этого в коде формы **FormDishesList** пропишите:

```
public void ShowForm()
    {
        Show();
        Activate();
     }
```

Тогда на главной форме для пункта меню для блюд пропишите новый код:

```
private void блюдаВМенюToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FormDishesList.fd.ShowForm();
         }
```

Запустите программу и посмотрите появившеюся возможность.

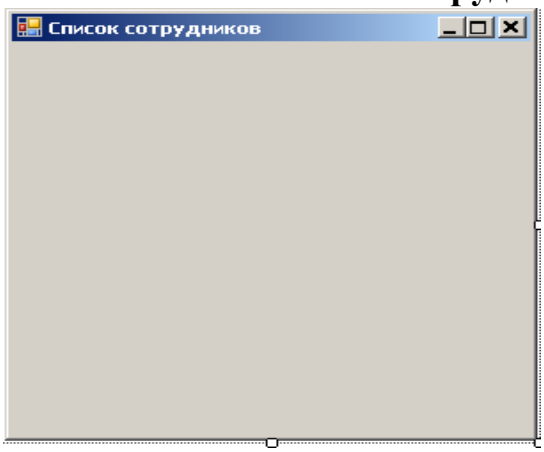
Внимание:

посмотрите, что сейчас возможен переход между формами, в случае повторного выбора пункта меню для блюд будет показана и сфокусирована одна форма для блюд.

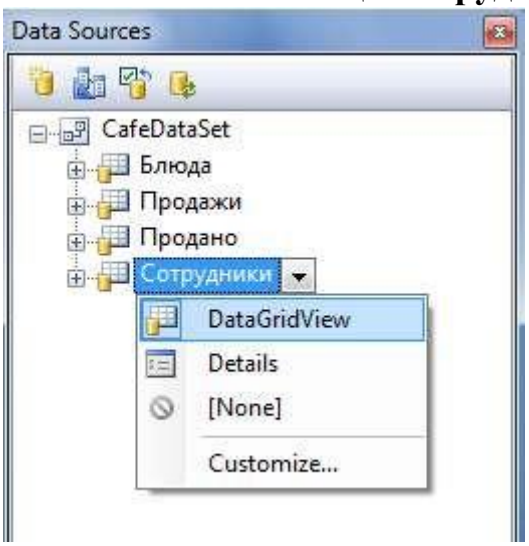
4.4.5 Представление данных в DataGridView

Добавим еще форму в проект **для отображения данных по сотрудникам FormWorkesList**.

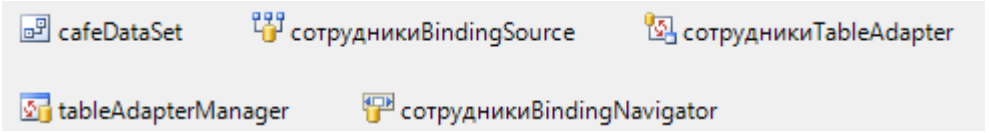
Укажите для нее свойство **Text** в **Список сотрудников**.



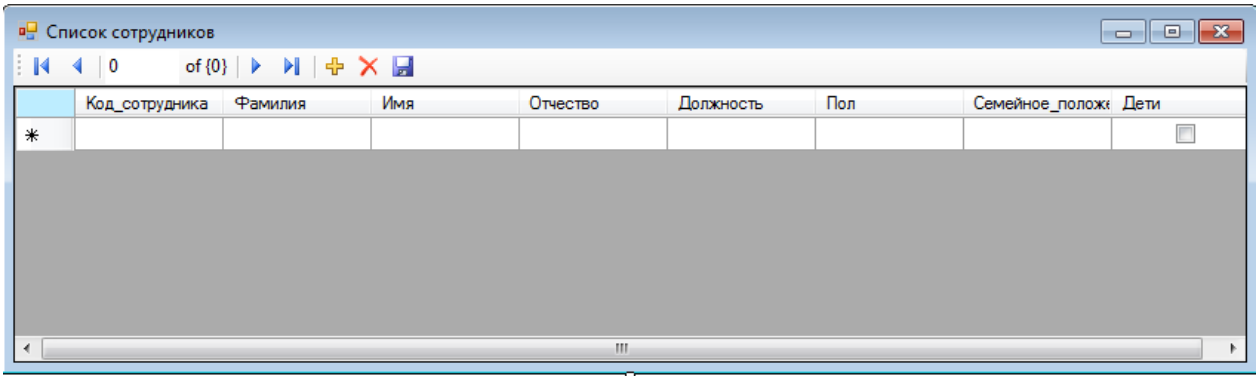
На нее разметите **DataGridView** по таблице **Сотрудники**:



При этом создадутся:



Укажите для **сотрудникиDataGridView** свойство **Dock** в **Fill** (всю область). Вид формы имеет вид:



Для **сотрудникиBindingNavigator** измените на русский язык текст всплывающих подсказок через свойство **Text** для выбранного элемента навигатора. Например, для кнопки с изображение дискеты **сотрудникиBindingNavigatorSaveItem** укажите свойство **Text** в **Сохранить**.

И далее для остальных кнопок навигатора согласно их смыслу.

Для кнопки **bindingNavigatorCountItem** укажите свойство **Text** в **из {0}**

Реализация шаблона проектирования «Одиночка» для формы FormWorkesList:

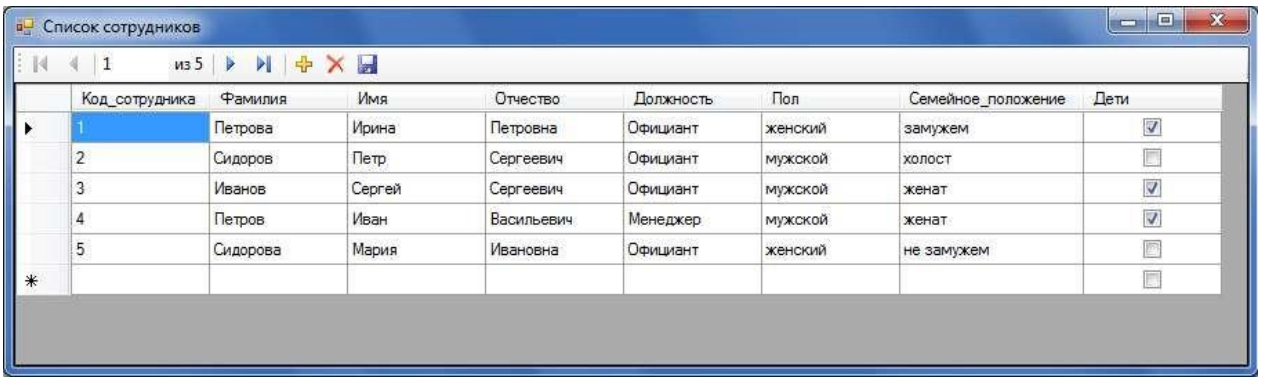
Для формы Сотрудники пропишите в коде **FormWorkesList:**

```
private static FormWorkesList f;

        public static FormWorkesList fw
        {
            get
            {
                if (f == null || f.IsDisposed) f = new FormWorkesList();
                return f;
            }
        }

        public void ShowForm()
        {
            Show();
            Activate();
        }
```

А в коде главной формы для пункта меню Сотрудники пропишите код:

```
private void сотрудникиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FormWorkesList.fw.ShowForm();
        }
```

В результате при выборе данного пункта в программе будет показана форма для сотрудников:



Код_сотрудника	Фамилия	Имя	Отчество	Должность	Пол	Семейное_положение	Дети
1	Петрова	Ирина	Петровна	Официант	женский	замужем	☑
2	Сидоров	Петр	Сергеевич	Официант	мужской	холост	☐
3	Иванов	Сергей	Сергеевич	Официант	мужской	женат	☑
4	Петров	Иван	Васильевич	Менеджер	мужской	женат	☑
5	Сидорова	Мария	Ивановна	Официант	женский	не замужем	☐
*							☐

Если поместить новый **DataGridView** на форму и не производить никаких настроек, то **DataGridView будет:**

- Автоматически показывать заголовки колонок и заголовки строк. И те, и другие остаются видимыми при любом скроллинге.

- Ставить на одном из заголовков строк маркер (черный треугольничек) текущей строки .

- Выбирать целую строку, если пользователь щелкнет по заголовку строки.

- Выбирать сразу несколько строк , если щелчок по заголовку строки производится с зажатым Ctrl или Shift. При этом текущая строка (помеченная треугольничком) всегда будет единственной.

- Удалять все выбранные строки по нажатию на Delete.

- Отображать ячейку, имеющую фокус ввода, особым образом.

- Если пользователь выполнит двойной щелчок по разделителю колонок, будет произведена автоподборка ширины левой колонки.

- Если в методе Main приложения был вызван метод EnableVisualStyles, будет применяться стиль Windows XP, выбранный в настройках рабочего стола.

Помимо этого **DataGridView будет поддерживать редактирование содержимого:**

- Если пользователь выполнит двойной щелчок по ячейке (или нажмет на ней F2), данная ячейка будет переведена в режим редактирования.

- Если пользователь изменит хотя бы один символ в редактируемой ячейке, на заголовке соответствующей строки появится спецсимвол (пишущий карандаш), и будет отображаться до тех пор, пока фокус ввода не покинет редактируемую ячейку, или пока пользователь не нажмет Esc. Последнее действие восстановит то значение ячейки, которое она содержала до входа в режим редактирования.

- Если пользователь прокрутит control вниз до последней строчки, будет отображена дополнительная, специальная строчка для внесения новой записи. Такая строчка всегда помечена символом звездочки на заголовке.

Когда пользователь любым способом перемещается в эту строку, DataGridView добавляет новую запись со значениями по умолчанию. Если

фокус ввода находится в этой строке, и пользователь нажимает Esc, новая запись пропадает, а фокус ввода перемещается на строчку выше.

Если DataGridView привязан через свойство DataSource **к источнику данных**, то по умолчанию выполняется следующее.

- Каждая колонка, получаемая из источника данных, вызовет добавление соответствующей колонки в controle.
- Названия колонок источника отобразятся в заголовках колонок.
- Если пользователь щелкнет по заголовку колонки, строки будут автоматически отсортированы.

Излишне говорить, что практически все из перечисленного выше может быть разрешено /запрещено /настроено.

Привязка данных в DataGridView может быть проведена несколькими способами:

- Основной – отображение данных из внешних коллекций (например, ListView, DataTable).
- Специальный режим отображения свободных (unbound) данных, то есть данные хранятся в самом controle.
- Виртуальный (Virtual mode). В нем control посылает событие, при поступлении которого прикладной код возвращает некоторые данные. Так как данные при этом не обязаны где-то храниться, виртуальный режим может оперировать большим количеством строк без каких-либо проблем с производительностью или нехваткой памяти.

В основном DataGridView будет работать в первом режиме, так как в большинстве случаев данные будут поступать из БД, при этом копируясь в промежуточные коллекции, например, DataTable.

Чаще всего данные попадают в DataGridView из подключенного источника данных. При этом встроенный механизм Windows Forms Data Binding автоматически заполняет каждую ячейку значением из соответствующей ячейки источника. Но, как вы уже знаете, DataGridView поддерживает также специальный режим отображения «свободных» (не привязанных, unbound) данных. Кроме того, поддерживается комбинированный режим одновременного отображения связанных и свободных данных.

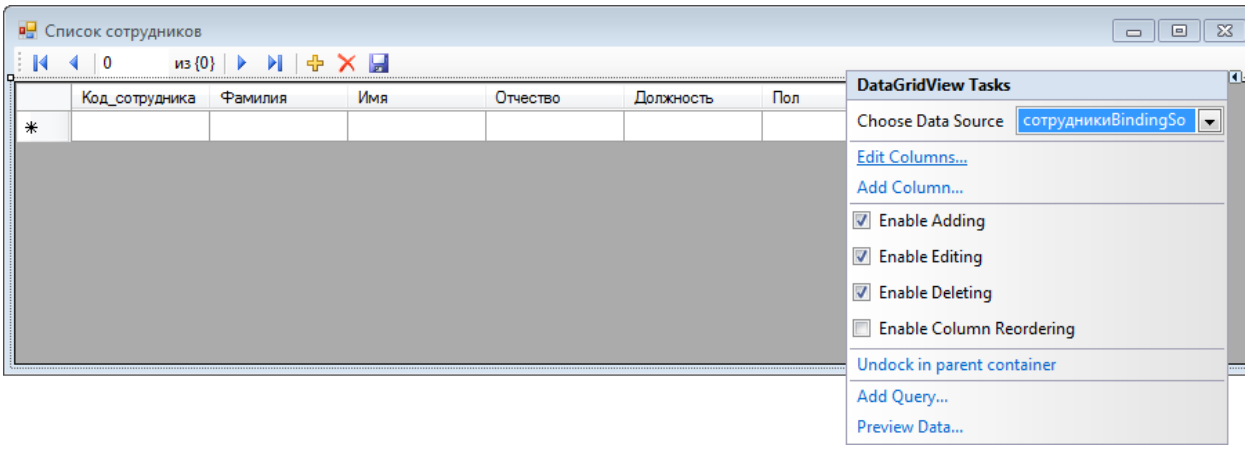
Свободными данными называются данные, заносимые в ячейки вручную. Связанными данными называются данные, заносимые в ячейки автоматически из привязанного источника данных. Будут ли данные считаться связанными или свободными, определяется на уровне колонок. Колонка становится связанной, если в ее свойство DataGridViewColumn.DataPropertyName (типа string) заносится название колонки или свойства объекта из источника данных.

Колонка считается свободной, если упомянутое свойство становится равным пустой строке.

Если свойство AutoGenerateColumns выставлено в true, DataGridView не только генерирует новый список колонок при подключении источника данных, но и привязывает эти колонки к колонкам или свойствам источника данных. Однако это не означает, что нельзя управлять привязкой колонок после их автоматического формирования.

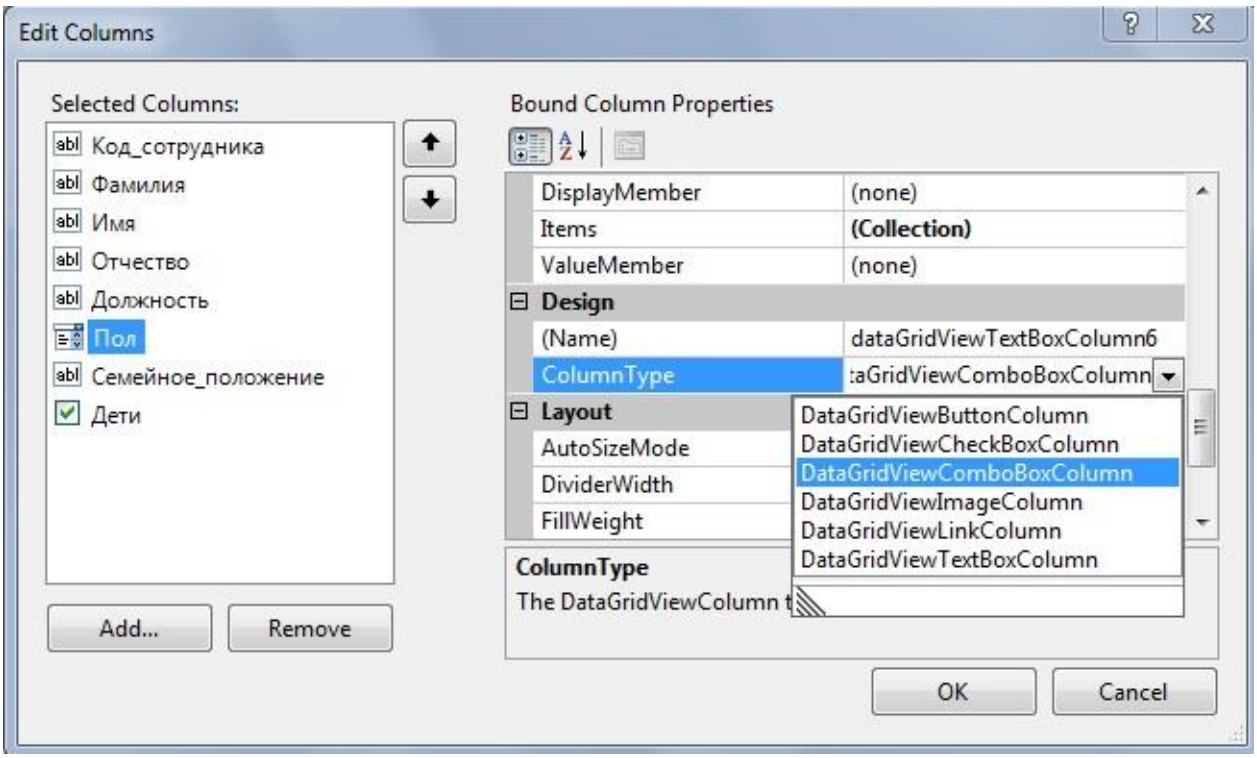
Настроим несколько колонок в сотрудникиDataGridView на форме FormWorkesList.

Нажмите на **сотрудникиDataGridView** команду **Edit Columns:**

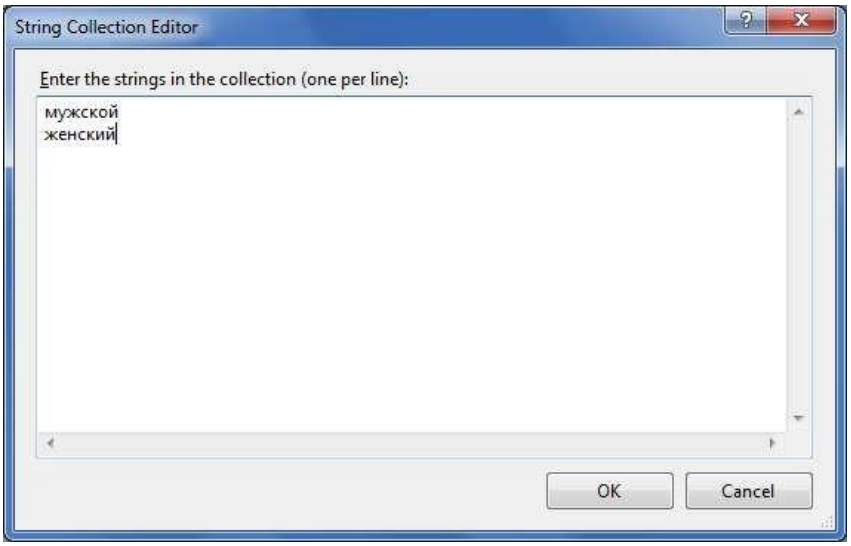


В поле **Пол,** согласно наложенным ограничениям, допустимы значения 'мужской' или 'женский'. Сделаем для удобства работы с этим полем выпадающий список из данных значений (**DataGridViewComboBoxColumn**).

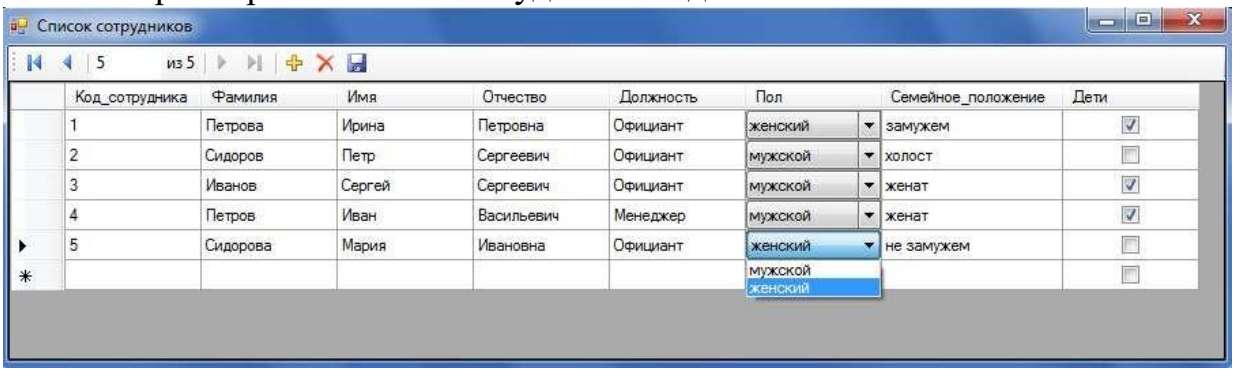
Для поля **Пол** укажите тип колонки:



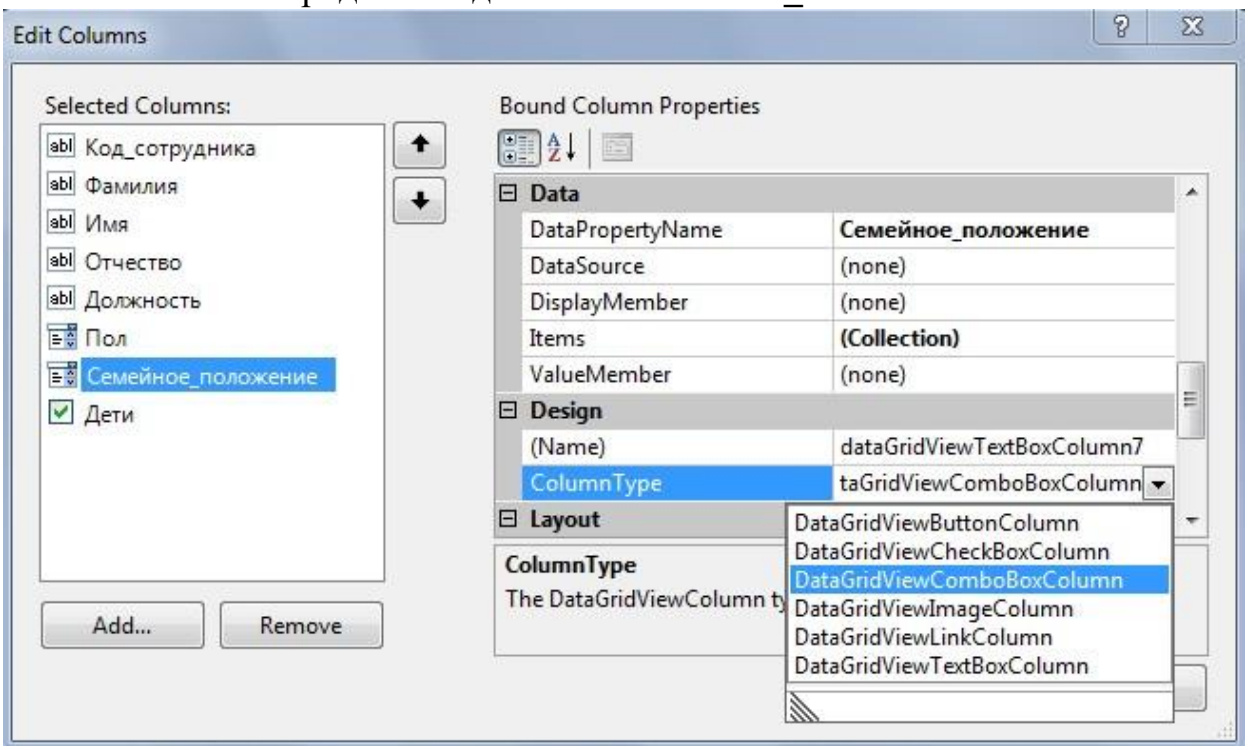
В свойстве **Items** укажите:



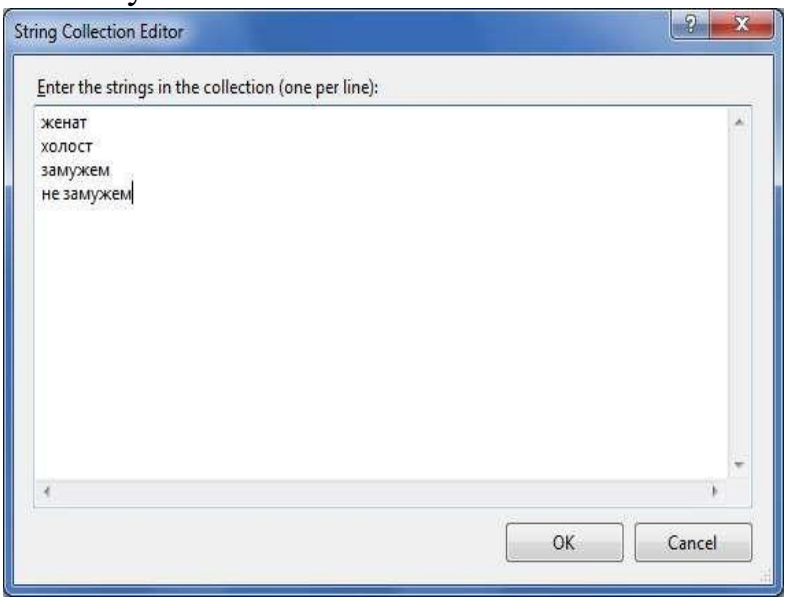
Теперь в приложении это будет выглядеть:



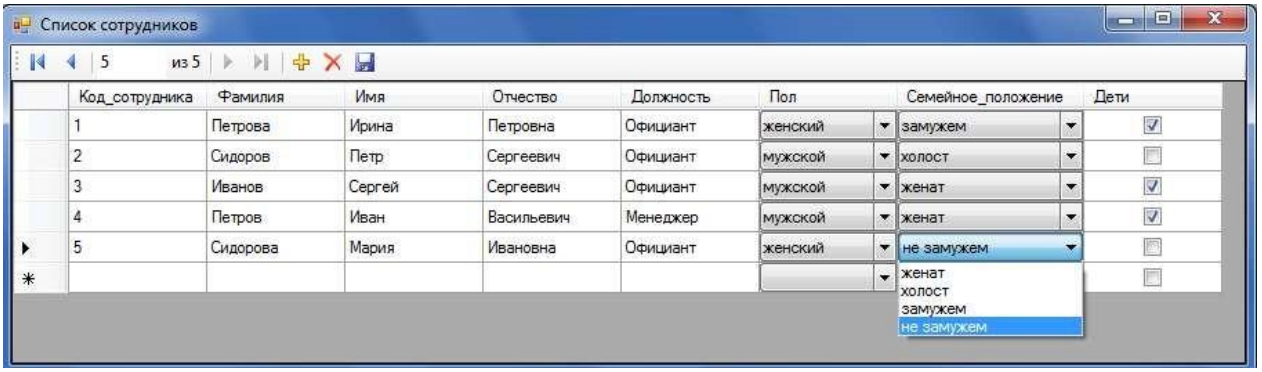
Аналогично проделайте для поля **Семейное_положение**:



В свойстве **Items** укажите:



Вид формы в приложении:



Примечание:

Обратите внимание, что для поля **Дети** из-за выбранного типа данных автоматически задан тип для колонки элемента управления флажок (**DataGridViewCheckBoxColumn**).

Раскрасим сотрудникиDataGridView по признаку пола сотрудника.

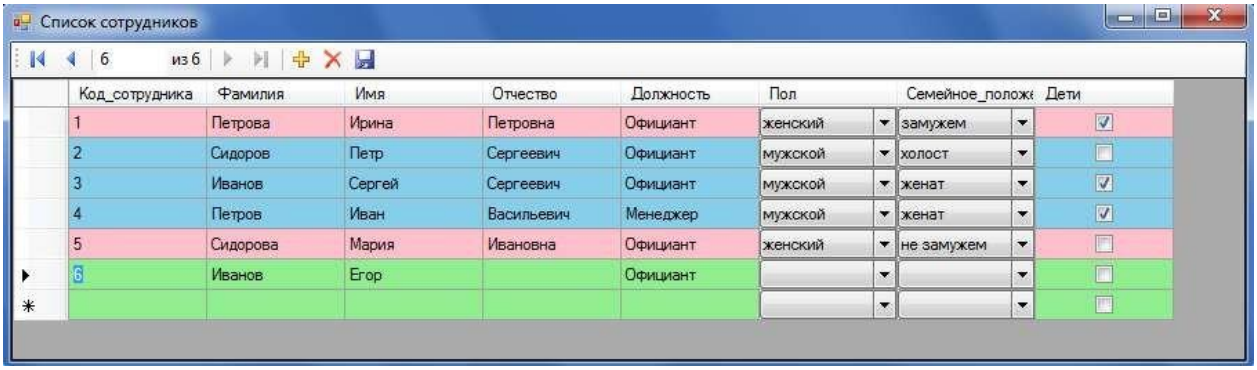
Примечание:

Укажите для дальнейшей работы свойство **Name** для колонки **Пол** в **dataGridViewTextBoxColumnPol**.

Для этого пропишем в событии **CellFormatting** у **сотрудникиDataGridView:**

```
private void сотрудникиDataGridView_CellFormatting(object sender,
DataGridViewCellFormattingEventArgs e)
        {
            if
((сотрудникиDataGridView.Rows[e.RowIndex].Cells["dataGridViewTextBoxColumnPol
"].Value == null) ||
(сотрудникиDataGridView.Rows[e.RowIndex].Cells["dataGridViewTextBoxColumnPol"
].Value.ToString() == ""))
                e.CellStyle.BackColor = Color.LightGreen;
            else
            {
                if
(сотрудникиDataGridView.Rows[e.RowIndex].Cells["dataGridViewTextBoxColumnPol"
].Value.ToString() == "мужской")
                    e.CellStyle.BackColor = Color.SkyBlue;
                else
                    e.CellStyle.BackColor = Color.Pink;

            }

        }
```

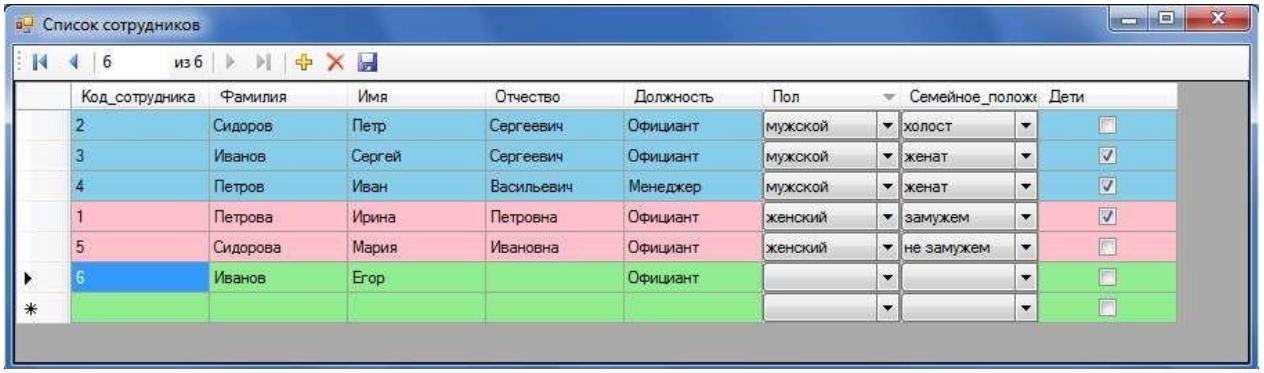
В результата получим раскрашенный **сотрудникиDataGridView**:



Примечание:

В нашей БД к полям Отчество, Пол, Семейное_положение, Дети не были предъявлены требования обязательного заполнения, поэтому такая ситуация, как на рисунке, вполне возможна в данной БД.

Результат можно отсортировать по полу, нажав по заголовку колонки **Пол**:



По аналогии можно настраивать данные в приложении и из других объектов БД.

Примечание:

Следует заметить, что в лабораторном практикуме для удобства объяснения материала и понимания предметной области используются идентификаторы с использованием кириллицы. Например, **сотрудникиBindingSource**, **сотрудникиTableAdapter**, **сотрудникиDataGridView**, **сотрудникиBindingNavigator** и др.

Чтобы сократить переключения раскладки клавиатуры при написании исходного текста программы, избежать проблем с возможным переносом программы на другие платформы, не поддерживающие кириллицу в качестве

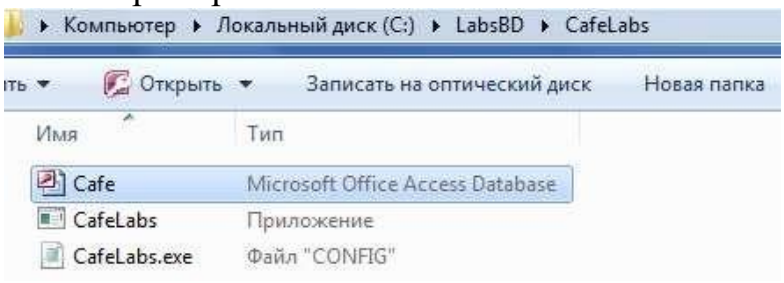
идентификаторов, или при предоставлении исходного кода не русскоязычным разработчикам, предпочтительно использовать латинский/английский алфавит.

4.5 Изменение пути к БД через конфигурационный файл

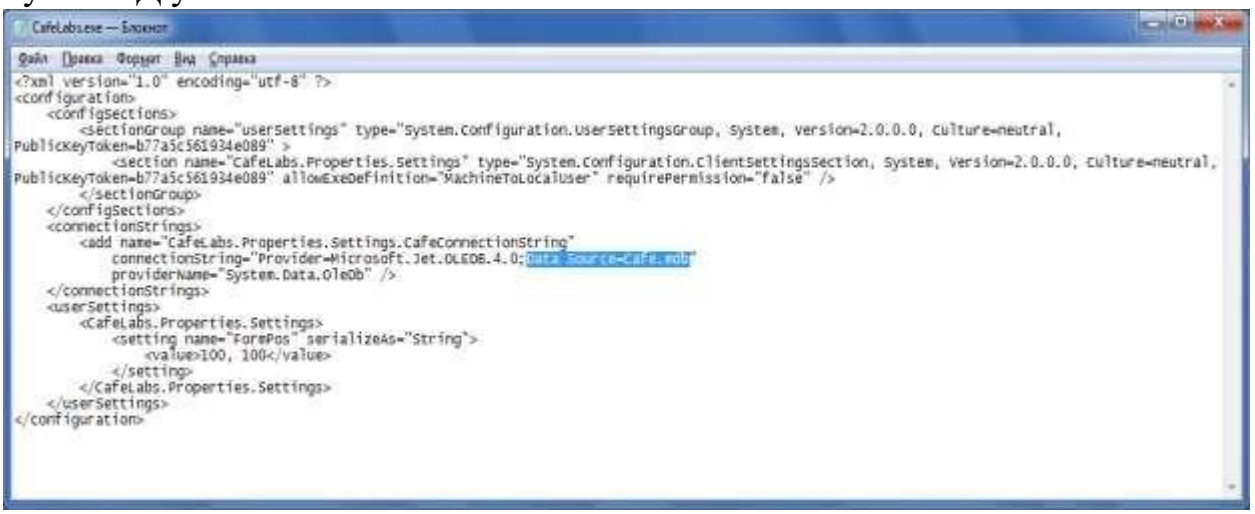
Как было описано ранее (раздел 4.3), мы установили опцию создавать конфигурационный файл, в который занести свойство **ConnectionString** (строка соединения с БД).

Покажем, как менять путь к БД через данный конфигурационный файл.

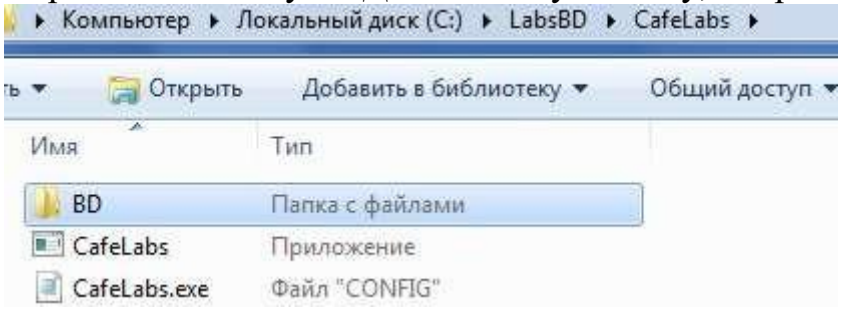
Пусть сейчас, например, файл БД находится в папке рядом с .exe файлом. Например:

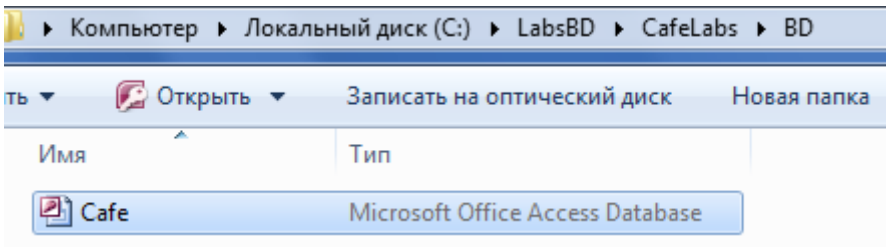


В данном случае в конфигурационном файле CafeLabs.exe.config для пути к БД указано:



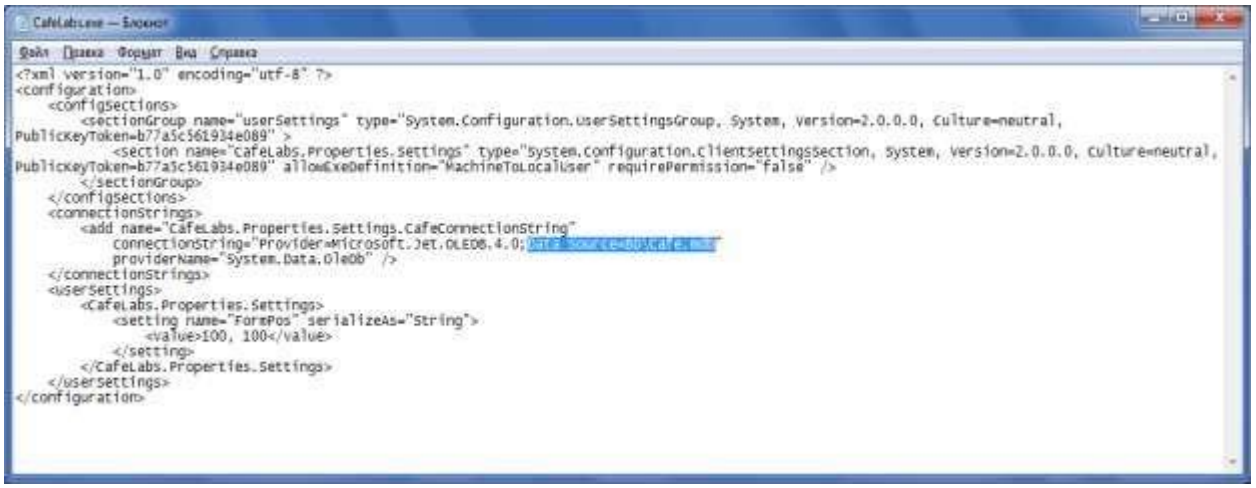
Переместим данную БД в отдельную папку, например, BD:





В данном случае путь к БД изменился. После запуска программы для работы с данными будет выдано сообщение, что не удается найти файл БД в папке, где .exe файл.

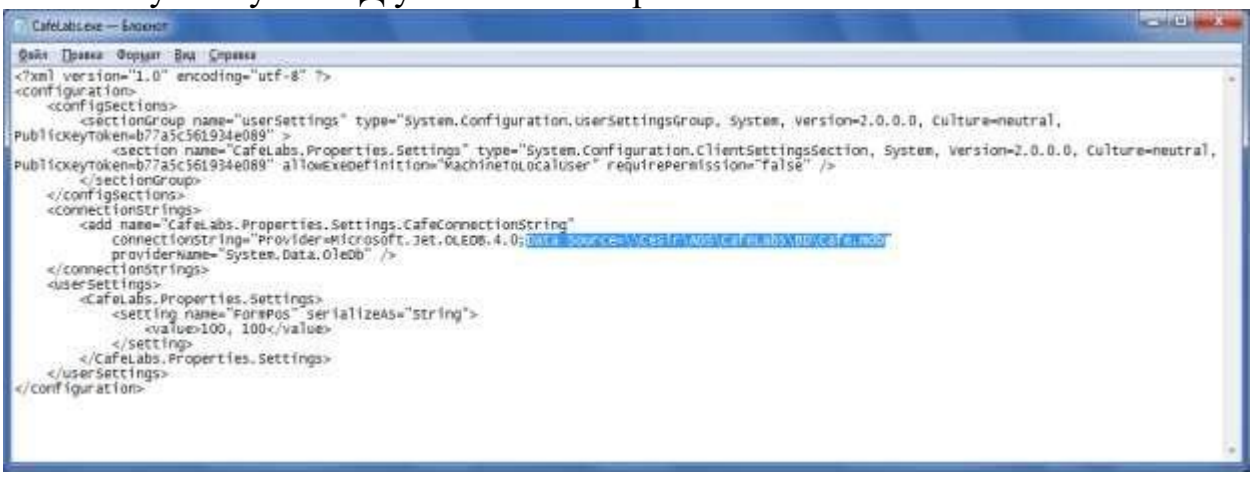
Для устранения данной ошибки откройте конфигурационный файл **CafeLabs.exe.config** и измените в нем путь к БД. В нашем случае – в папке BD рядом с .exe файлом:



В результате, программа по работе с данными вновь будет работоспособной без перекомпиляции приложения.

Примечание:

Если БД установлена на одном компьютере, а приложение на другом, то в этом случае путь к БД указывается через синтаксис:



---

5 Задание на лабораторную работу №2

5.1 Изучить теоретический и практический материал лабораторной работы №2.

5.2 Для своей БД, созданной в лабораторной работе №1, необходимо выполнить:

5.2.1 создать главную форму проекта с наличием главного меню, контекстного меню, панели инструментов;

5.2.2 настроить, чтобы главная форма сохраняла свое положение при повторном запуске программы;

5.2.3 провести подключение к БД в приложении;

5.24 сформировать конфигурационный файл для изменения пути к БД в приложении;

5.2.5 провести визуальное оформление данных БД в приложении. Обязательно:

- наличие нескольких форм в проекте;
- использование различных элементов управления Label, TextBox, ComboBox (и/или ListBox), CheckBox, PictureBox, DataGridView, BindingNavigator и др.

Если в БД нет данных для таких объектов, то изменить структуру БД.

5.2.6 провести модификацию настроек DataGridView (типы колонок обязательно);

5.2.7 настройте для удобства BindingNavigator (всплывающие подсказки обязательно;

5.2.8 предоставить работу с графическим полем (в том числе загрузку фотографии из файла);

5.2.9 реализация шаблона проектирования «Одиночка» (англ. Singleton);

5.2.10 протестируйте работу полученного приложения.

6 Оформление отчёта к лабораторной работе

Оформить отчёт со следующим содержанием:

6.1 Титульный лист.
6.2 Цель работы.
6.3 Постановка задачи.
6.4 Краткая теория и ход выполнения заданий.
6.5 Описание результатов.
Приведите скриншоты полученной программы и листинг программы.
6.6 Заключение (выводы).

Отчет должен быть оформлен согласно требованиям ОС ТУСУР.

7 Контрольные вопросы

7.1 Что такое ADO.NET? Ее особенности.
7.2 Модель доступа к данным в ADO.NET.
7.3 Приведите примеры типов System.Data.
7.4 Что такое DataSet? Его особенности.
7.5 Особенности DataAdapter.
7.6 Как создать главное меню, контекстное меню, панель инструментов?
7.7 Как провести подключение БД в приложении Visual Studio?
7.8 Для чего нужен компонент BindingSource? Его особенности.
7.9 Что характерно для отображения данных в виде Details?
7.10 Что характерно для представления данных в данных в DataGridView?
7.11 Опишите управляющие кнопки BindingNavigator.
7.12 Особенности DataGridView.
7.13 Как в DataGridView провести сортировку данных?
7.14 Что характерно для шаблона проектирования «Одиночка» (англ. Singleton)?

ЛАБОРАТОРНАЯ РАБОТА № 3 СОЗДАНИЕ ПРИЛОЖЕНИЯ В MICROSOFT VISUAL STUDIO ДЛЯ РАБОТЫ СО СВЯЗАННЫМИ ТАБЛИЦАМИ БАЗЫ ДАННЫХ

Цель работы:

Изучение основных особенностей создания приложения для работы со связанными таблицами базы данных в Microsoft Visual Studio для своей предметной области (на примере базы данных для кафе).

1 Введение

Разрабатываемое приложение предназначено для работы с базой данных для кафе (см. лабораторную работу №1, лабораторную работу №2).

Структура БД для кафе в рассматриваемом примере:

В лабораторной работе №2 было создано приложение для работы с данными в родительских таблицах:

- **Блюда**;
- **Сотрудники**.

В данных таблицах нет внешних ключей (FK), они являются родительскими таблицами:

- таблица **Блюда** связана с таблицей **Продано** связью 1:M;
- таблица **Сотрудники** связана с таблицей **Продажи** связью 1:M;
- таблица **Продажи** связана с таблицей **Продано** связью 1:M.

Примечание:

Таблица **Продано** появилась в БД из-за существования связи M:M между **Блюда** и **Продажи**.

В БД для кафе была установлена **ссылочная целостность, для которой характерно:**

- первичный ключ (PK) таблицы должен содержать уникальные

непустые значения для данной таблицы;

- внешний ключ (FK) потомка должен содержать только те значения, которые уже имеются среди значений PRIMARY KEY родителя (или колонок UNIQUE родителя);

- нельзя удалить экземпляр родителя, имеющий хотя бы один связанный с ним экземпляр потомка. Чтобы удалить экземпляр родителя, имеющего связанные потомки, необходимо сначала удалить связанные записи в потомке, а потом удалить экземпляр родителя с ними связанного. ИЛИ Можно установить механизм **каскадного удаления,** при котором удаляется экземпляр родителя вместе со связанными экземплярами потомка;

- нельзя изменить значение в РК (UNIQUE) родителя, имеющем хотя бы один связанный с ним экземпляр потомка. Можно установить механизм **каскадного обновления,** при котором изменяется РК (UNIQUE) родителя и значения FK связанных экземпляров потомка.

Ограничение FOREIGN KEY может содержать пустые значения (значения NULL). Кроме того, тип данных и размер колонки или колонок внешнего ключа должны совпадать со ссылочной колонкой или колонками.

Для связи между таблицами **Сотрудники** и **Продажи** механизм **каскадного обновления связанных записей** не установлен, так как Код_сотрудника в **Сотрудники** является типом данных счетчик, а его нельзя редактировать.

Для связи между таблицами **Сотрудники** и **Продажи механизм каскадного удаления связанных записей** не установлен, чтобы при удалении данных о сотруднике, связанные с ним продажи автоматически не удалялись. Чтобы удалить сотрудника необходимо сначала удалить все его продажи, а затем сведения по самому сотруднику.

Для связи между таблицами **Продажи** и **Продано** механизм **каскадного обновления связанных записей** не установлен, так как Номер_продажи в **Продажи** является типом данных счетчик, а его нельзя редактировать.

Для связи между таблицами **Продажи** и **Продано установлен механизм каскадного удаления связанных записей**, чтобы при удалении данных о продаже, связанные с ней блюда в **Продано** автоматически удалялись.

Для связи между таблицами **Блюда** и **Продано установлен** механизм **каскадного обновления связанных записей**, так как Код_блюда в **Блюда** может поменяться (например, введение новой формы меню, а блюдо неизменно). При этом необходимо, чтобы связанные с данным блюдом продажи автоматически среагировали на смену значения в Код_блюда.

Механизм каскадного удаления связанных записей для связи между таблицами **Блюда** и **Продано** не установлен, чтобы при удалении данных о блюде, связанные с ним продажи в **Продано**, автоматически не удалялись.

2 Практические задания

Работа продолжается в проекте, созданном при выполнении лабораторной работы №2.

2.1 Связь 1:М между Продажи и Продано в приложении

Как было сказано ранее, в БД кафе таблица **Продажи** связана с таблицей **Продано** связью 1:М.

Покажем, как можно, например, реализовать данную связь в приложении.

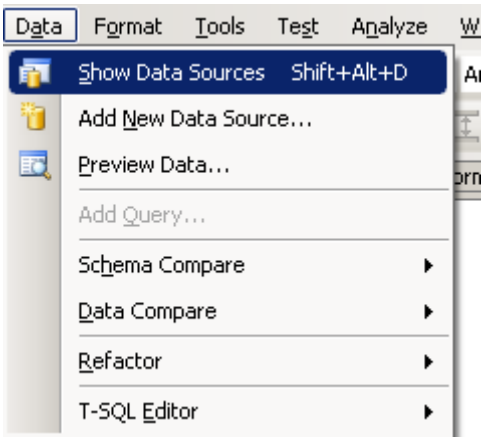
Создайте для работы с данными продаж отдельную форму.

Для этого нажмите на проекте правую кнопку мыши и выберете пункт **Add\Windows Forms**.

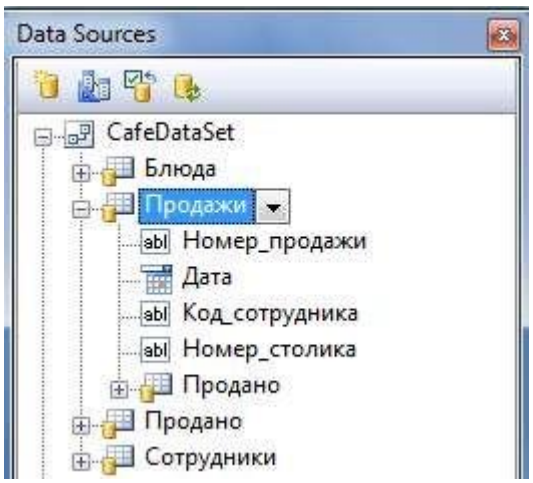
Задайте для новой формы имя **FormSalesList**.

Свойство **Text** для новой формы задайте в **Продажи**.

Откройте источник данных **DataSource**, выбрав меню **Data\Show Data Source**:

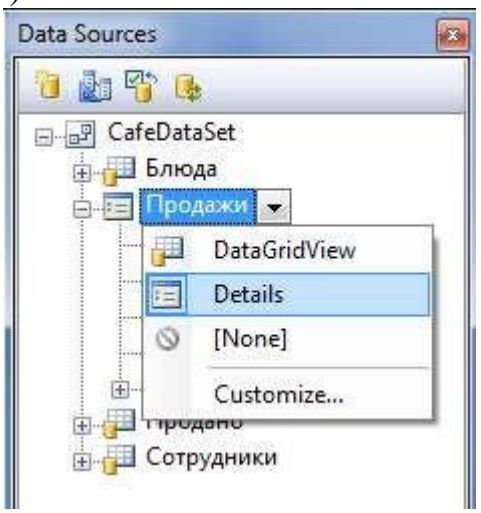


В открытом источнике данных выберете таблицу **Продажи**:

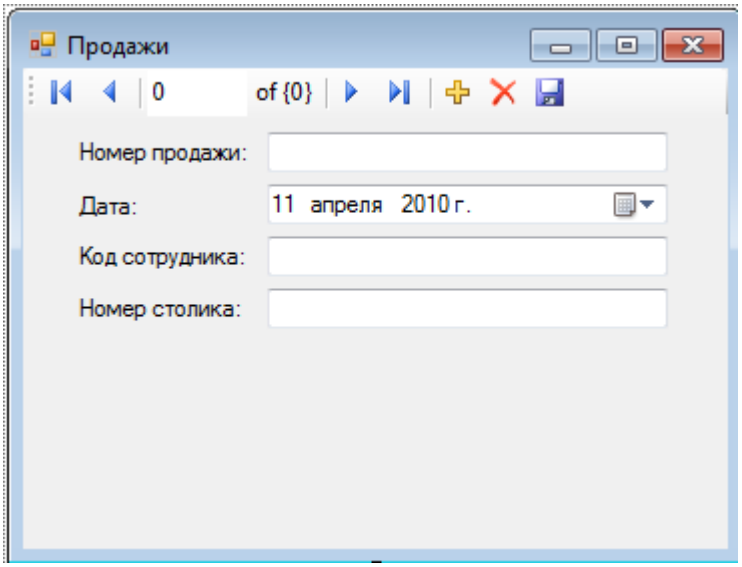


Обратите внимание, что для таблицы **Продажи** при нажатии + открывается не только список колонок данной таблицы, но и связанная с ней таблица **Продано**.

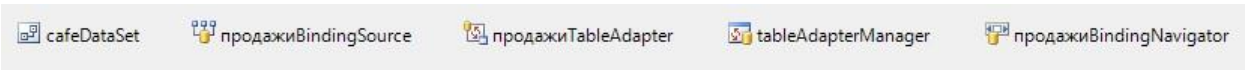
Выберете для продаж форму представления данных **Details** (каждое поле в отдельном компоненте):



И удерживая левую кнопку мыши, положите на форму. В результате получится форма:



И создадутся объекты:

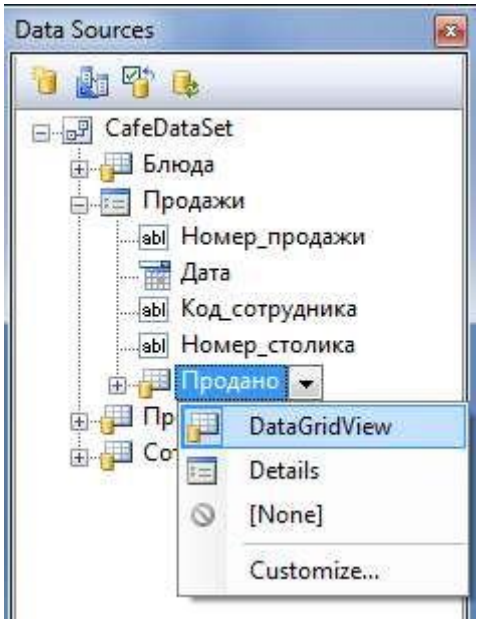


Разместите с панели инструментов **Toolbox** компонент **groupBox** для **Продано**.

Для **groupBox** укажите:

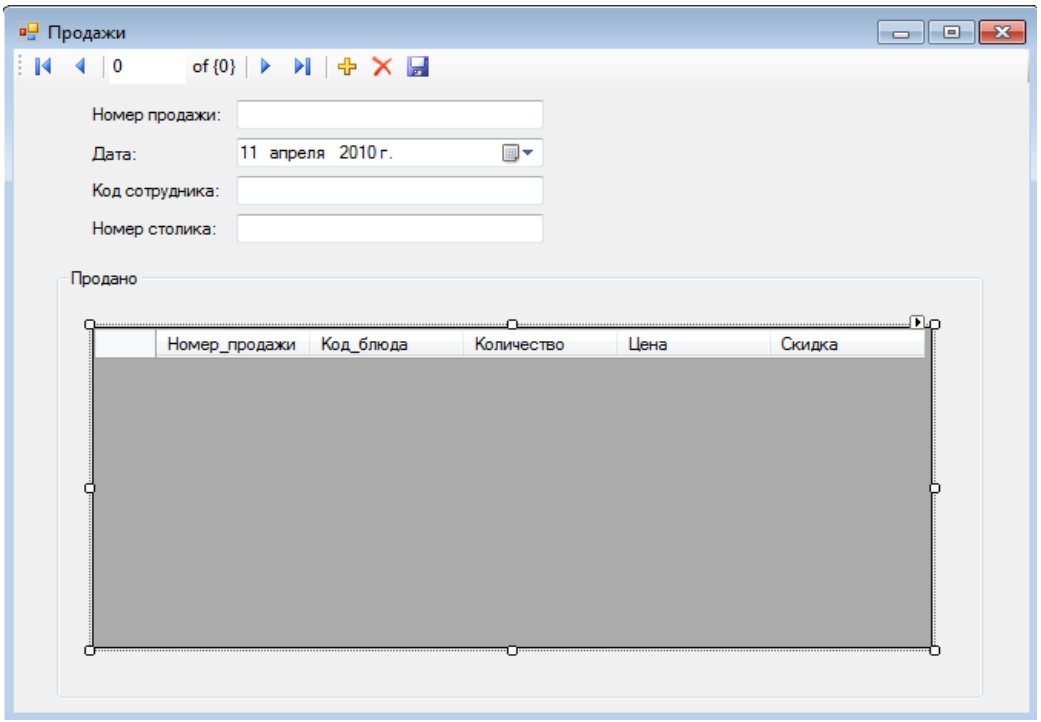
- свойство **Name** в **groupBoxDishSale**;
- свойство **Text** в **Продано**.

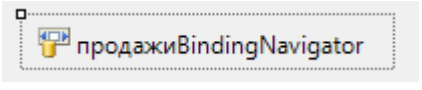
Выберете для таблицы **Продано** в **Продажи** (чтобы показывались данные по проданным блюдам для выбранной пользователем продажи) форму представления данных в **DataGridView**:



И перенесите, удерживая левую кнопку мыши, на **groupBoxDishSale**.

В результате получится форма:

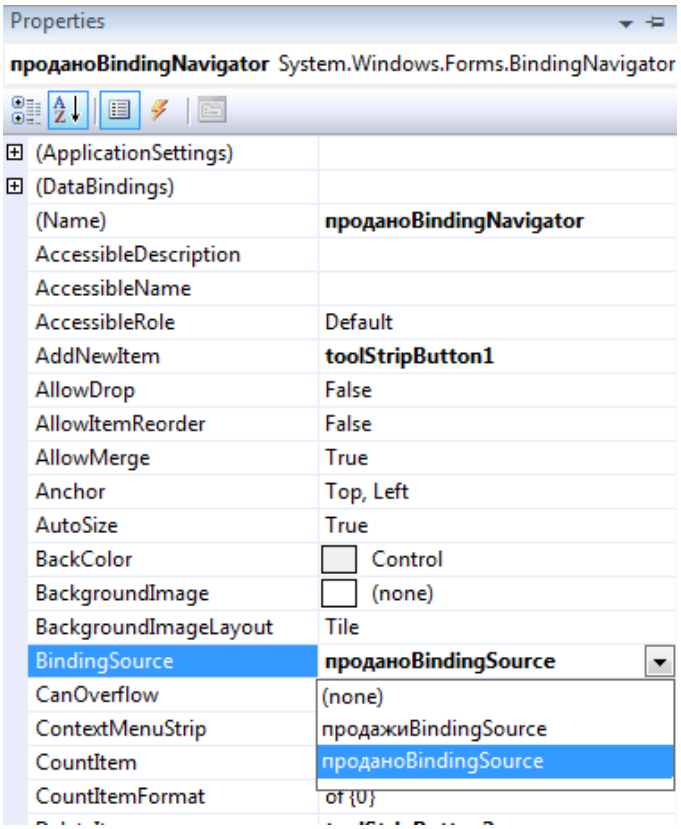


Обратите внимание, что **BindingNavigator** для **Продано** не был размещен. Добавим его сами. Для этого, например, выделите **продажиBindingNavigator** 

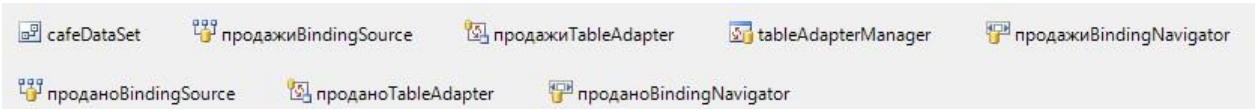
и нажмите на нем

правую кнопку мыши и выберете команду копировать (**Copy**). Затем на **groupBoxDishSale** нажмите правую кнопку мыши и выберете команду вставить (**Paste**). В результате будет добавлен **bindingNavigator1**. Для **bindingNavigator1** укажите:

- свойство **Name** в **проданоBindingNavigator**;
- в свойстве **BindingSource** укажите **проданоBindingSource** для привязки к **Продано**:



В итоге получаем объекты:



Настройте размещение **проданоBindingNavigator** и **проданоDataGridView** в **groupBoxDishSale** через свойство **Doc**:

- для **проданоBindingNavigator** свойство **Doc** в **Top** (верх);
- для **проданоSaleDataGridView** свойство **Doc** в **Fill** (вся область).

Настройте более удобный вид для пользователей **проданоBindingNavigator** и **продажиBindingNavigator** (см. лабораторную работу №2).

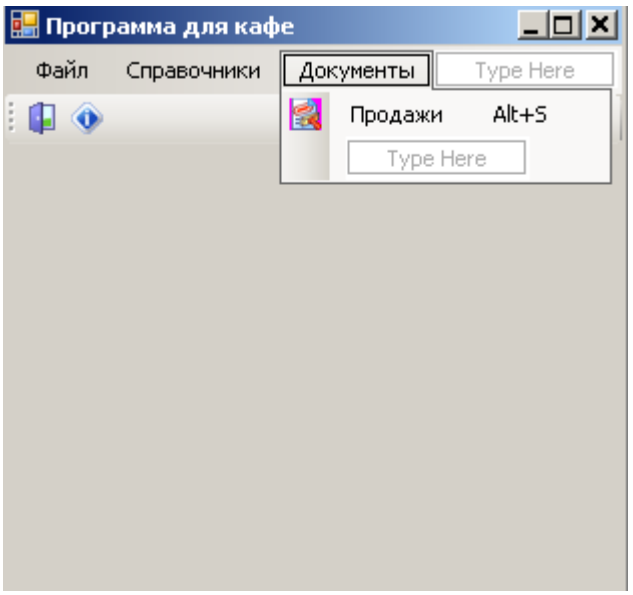
Задайте для кнопки сохранения данных в **проданоBindingNavigator** свойство **Name** в **проданоBindingNavigatorSaveItem.**

Запрограммируйте кнопку для сохранения данных для **проданоBindingNavigatorSaveItem** (по аналогии с **продажиBindingNavigatorSaveItem):**

```
private void проданоBindingNavigatorSaveItem_Click(object sender, EventArgs
e)
        {
            this.Validate();
            this.проданоBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.cafeDataSet);
        }
```

Покажем форму для продаж из нашей главной формы.

Для этого перейдите на главную форму и оформите для Продаж специальные пункты меню и панели инструментов (см. лабораторную работу №2). Например, в главном меню:



Реализуем шаблон проектирования **«одиночка» » (англ. Singleton)** (см. лабораторную работу №2) для данной формы.

Для этого в коде формы FormSalesList пропишите:

```
static FormSalesList f;
        static public FormSalesList fs
        {
            get
            {
                if (f == null || f.IsDisposed) f = new FormSalesList();
                return f;
            }
        }


        public void ShowForm()
        {
```

```
            Show();
            Activate();
        }
```

Тогда на главной форме для пункта меню Продажи пропишите код:

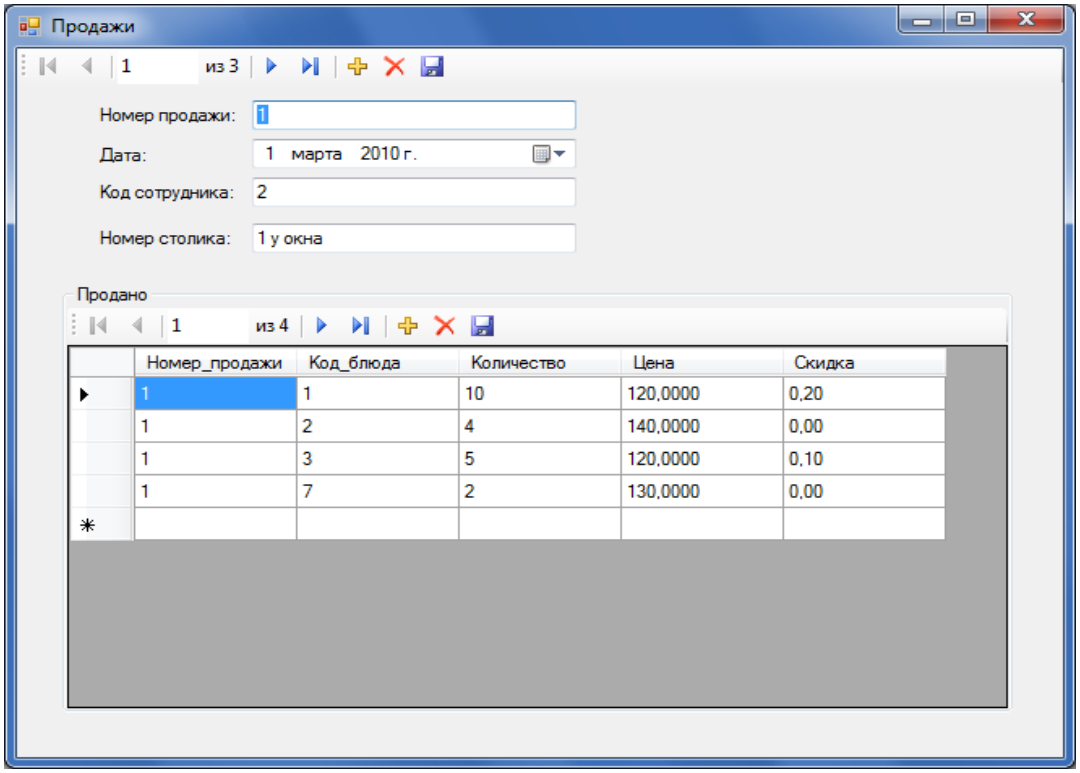
```
private void продажиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FormSalesList.fs.ShowForm();
        }
```

Запустите программу и посмотрите появившеюся возможность.

Примечание:

посмотрите, что возможен переход между формами. В случае повторного выбора пункта меню Продажи будет показана и сфокусирована одна форма для Продаж.

Запустите программу и посмотрите, как отображается данная форма, как работать с данными в данной форме. Обратите внимание, что показываются проданные блюда именно выбранной продажи:



Примечание:

Чтобы при добавлении новой продажи, дата продажи (**cafeDataSet.Продажи.Columns["Дата"]**) была присвоена по умолчанию (**DefaultValue**) текущей дате (**DateTime.Now**) допишите в обработчике

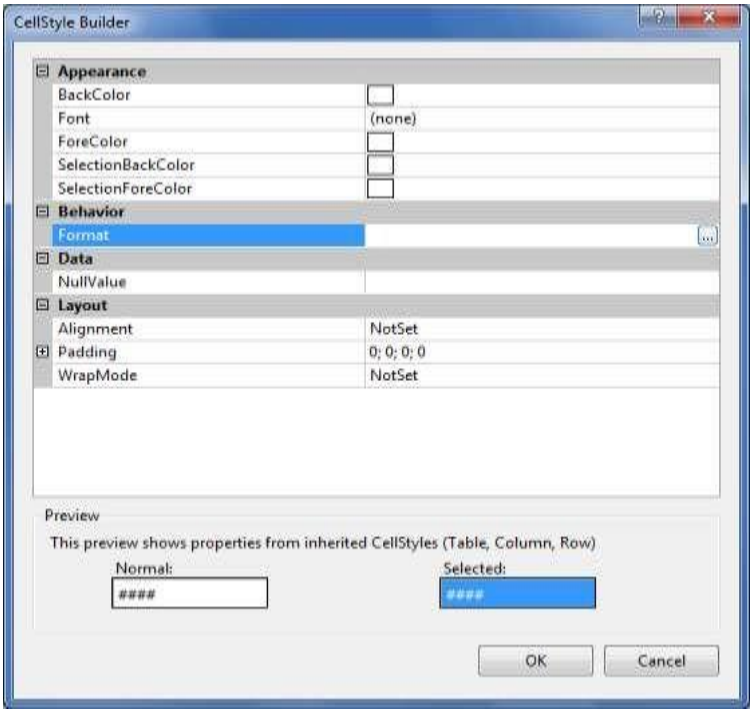
```
private void FormSalesList_Load(object sender, EventArgs e)
```

события на загрузку формы **FormSalesList** следующий код:

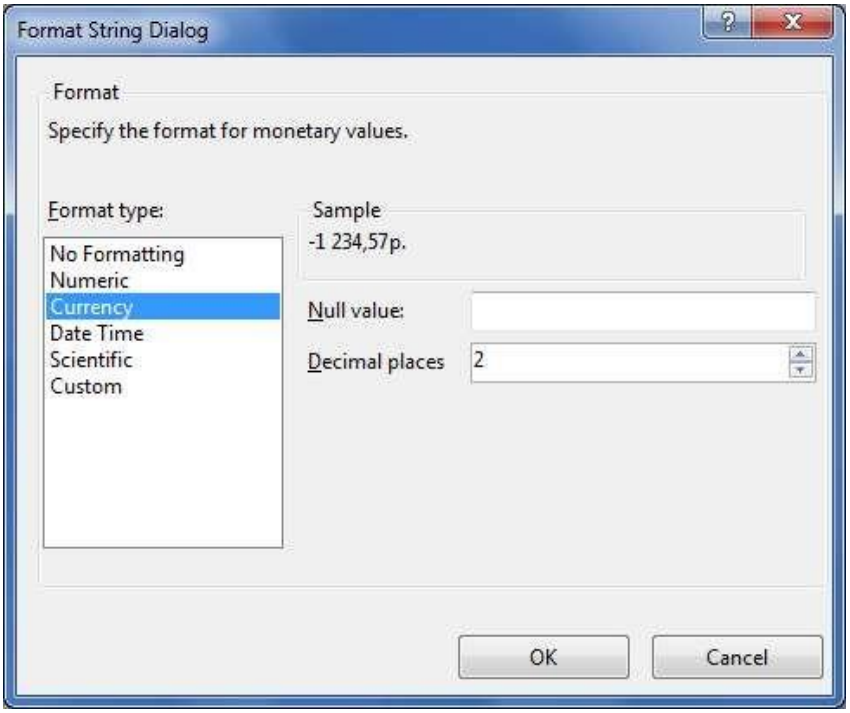
```
cafeDataSet.Продажи.Columns["Дата"].DefaultValue = DateTime.Now;
```

Для данной БД **Цена** должна быть выражена в денежном формате с точностью в 2 знака.

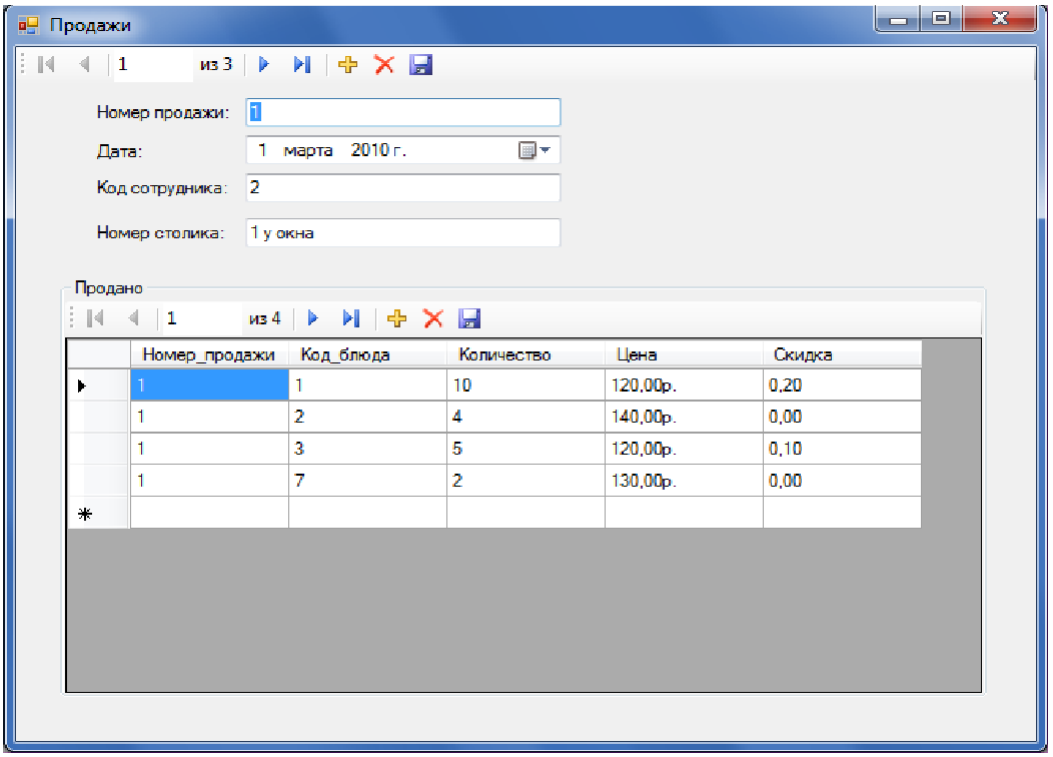
Чтобы это настроить зайдите в редактор колонок **проданоDataGridView.** Для колонки **Цена** зайдите в свойство **DefaultCellStyle.** В котором определите через свойство **Format:**



тип денежный (**Currency**) с точностью в 2 знака:



Запустите программу. Посмотрите настроенную возможность.



2.2 Обработка исключений в приложении

Ошибки, вызываемые компьютерной программой, могут быть разделены на три основных типа:

- **Design-time** (во время проектирования) - возникают при написании кода, который не удовлетворяет правилам того языка, на котором он пишется.

- **Runtime** (во время выполнения) - возникают при работе с программой, когда проводится попытка выполнить недопустимое действие, например, нарушение целостности данных, доступ к данным, которых не существуют, отсутствие прав доступа и др.

- **Logic** (логические) - не оказывают никакого влияния на работоспособность приложения. Наличие в программе логических ошибок означает лишь то, что могут быть неверные результаты от работы программы. Например, ошибки в вычислениях и др.

Чаще всего при работе с программой пользователи совершают **Runtime** ошибки.

Runtime ошибки вызывают **исключения (exception)**.

Исключение (exception) – это реакция на ошибку, сгенерированную приложением.

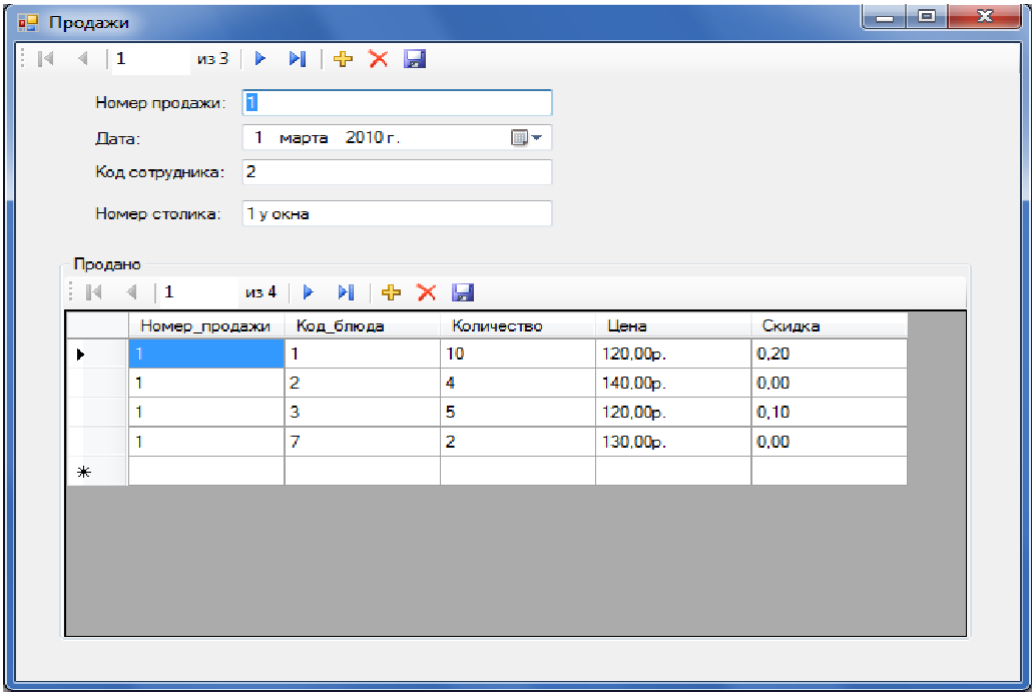
При возникновении исключения появляется диалог, сообщающий об ошибке с дополнительной информацией. Этот диалог дает возможность пользователю продолжить выполнение программы.

Обработчик ошибок – это блок кода, который может обнаружить исключения и выполнить необходимые действия для исправления этой ошибки.

Чтобы их обработать, можно использовать различные конструкции, например, конструкцию **try…catch**

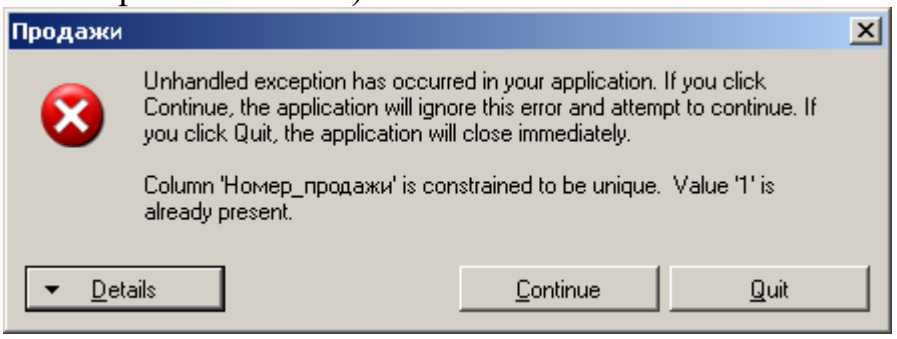
Если в блоке **try** генерируется исключение, то программа автоматически переходит в блок **catch**. Если никаких исключений не генерируется, то программа пропускает блок **catch**.

Покажем обработку исключений на примере работы с формой Продажи:

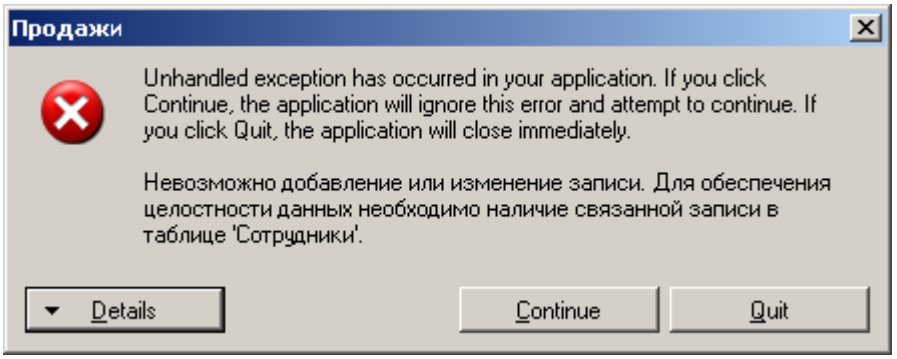


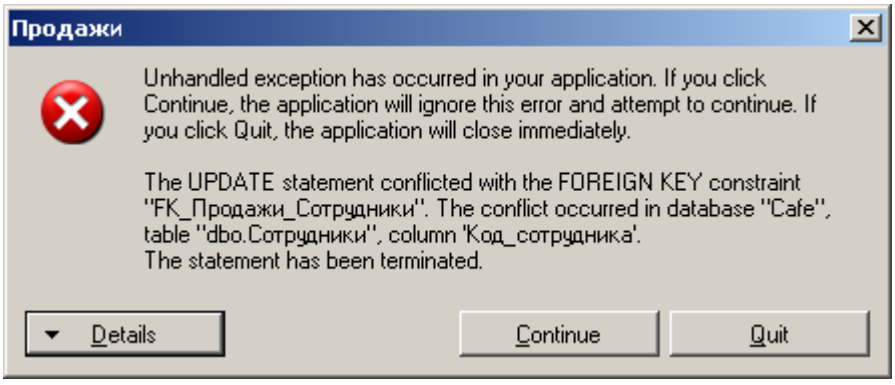
При работе с данными на данной форме могут возникнуть следующие исключения, например:

- **в таблице Продажи указан повторяющийся Номер_продажи** (в нашем случае – первичный ключ):

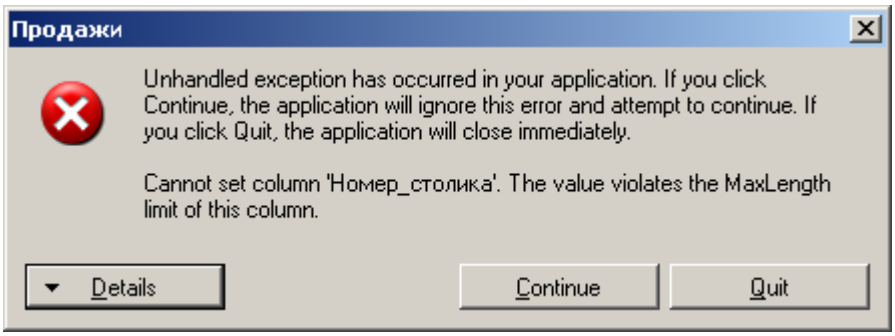


- **в таблице Продажи указан Код_сотрудника, которого в БД (в родительской таблице Сотрудники) не существует** (нами ранее была установлена ссылочная целостность):

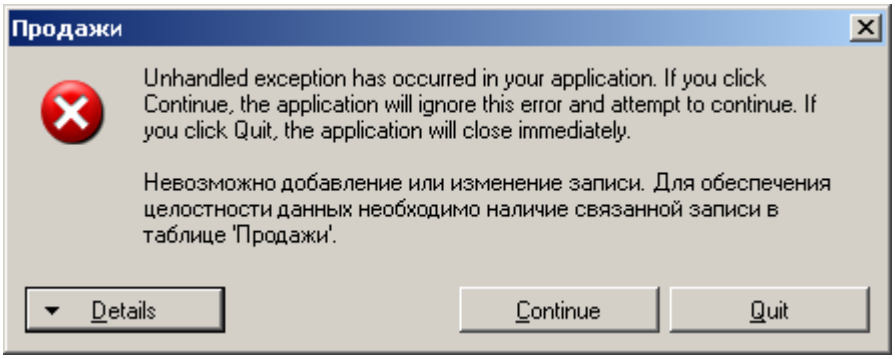


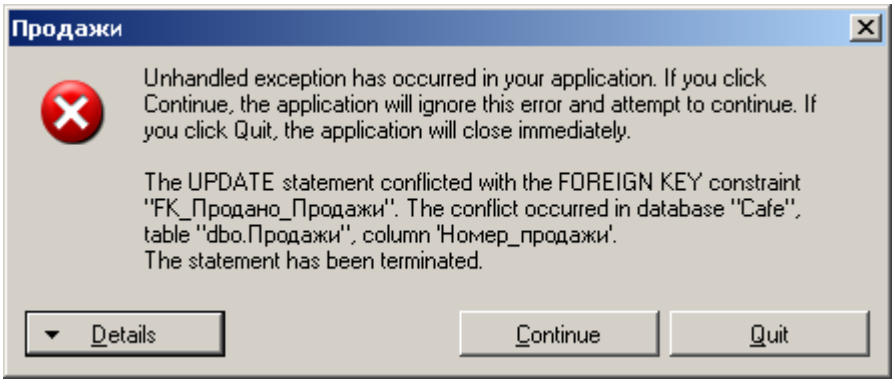


- в таблице Продажи превышено количество символов для ввода номера столика:

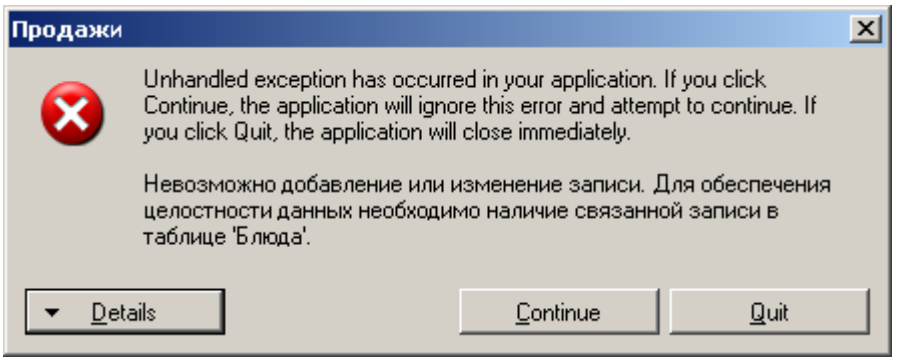


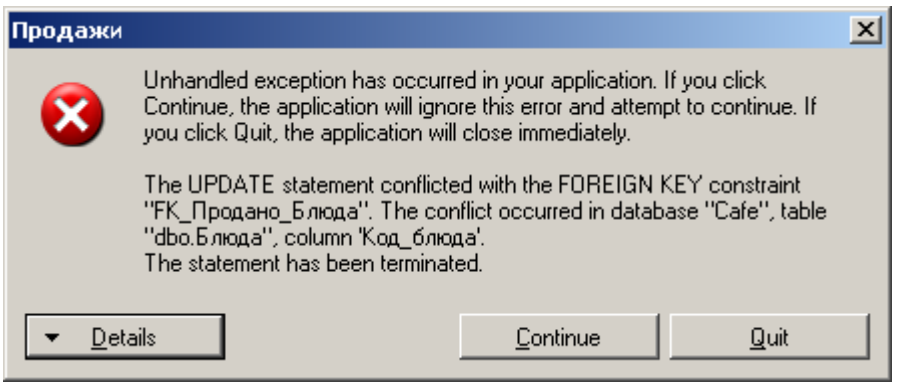
- в таблице Продано указан Номер_продажи, не существующий в БД (в родительской таблице Продажи) (нами ранее была установлена ссылочная целостность):



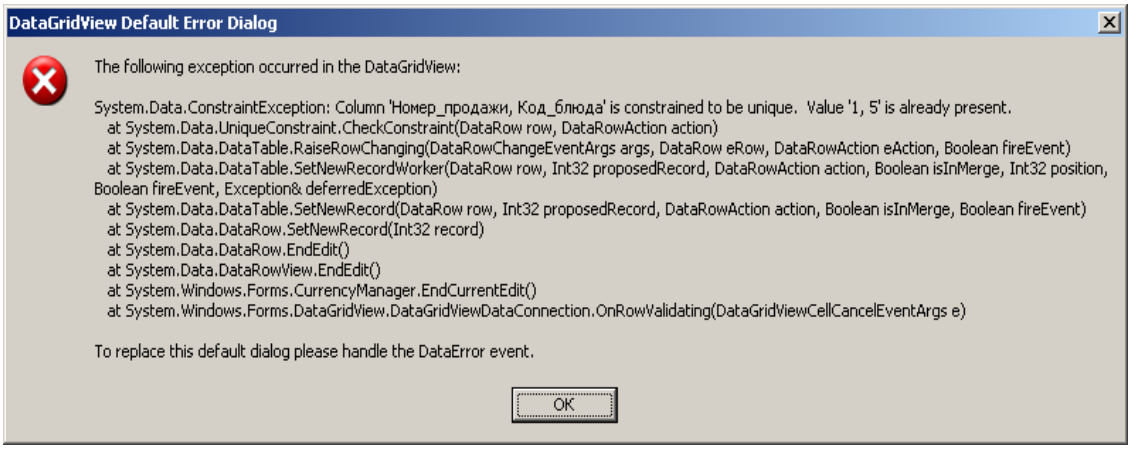


- в таблице Продано указан Код_блюда, не существующий в БД (в родительской таблице Блюда) (нами ранее была установлена ссылочная целостность):

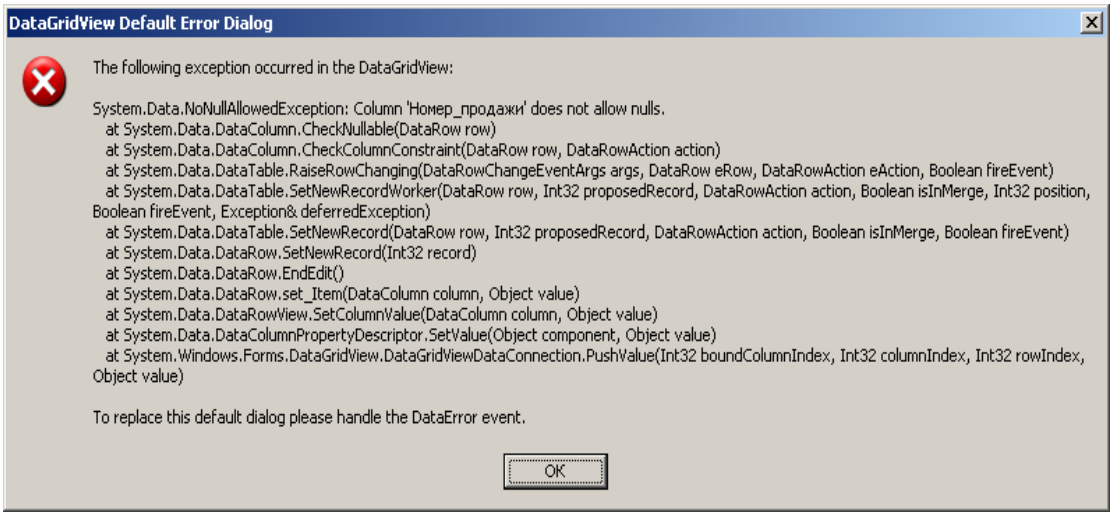




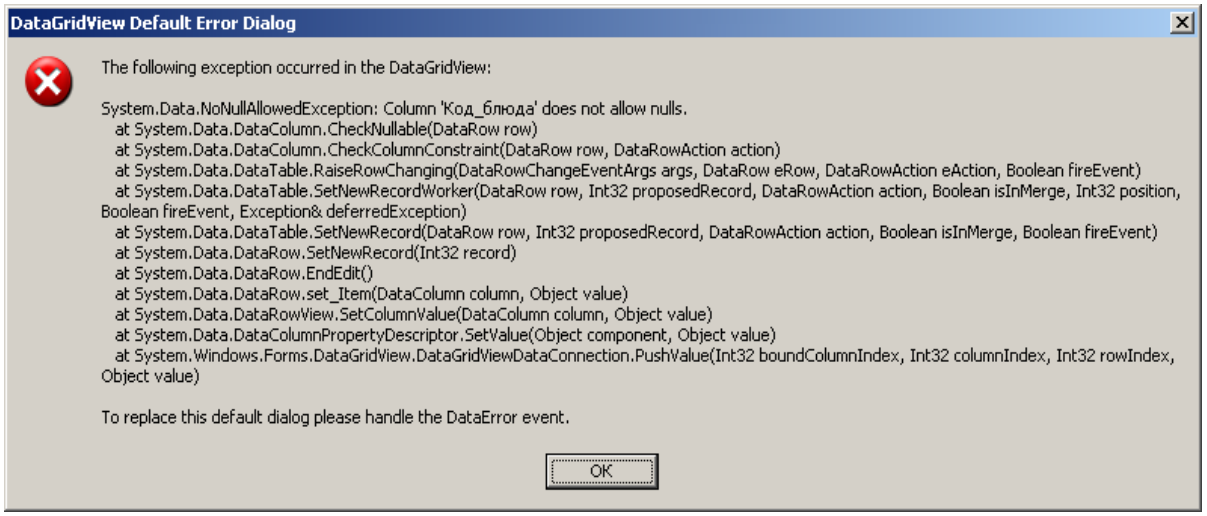
- в таблице Продано повторился первичный ключ (в нашем случае составной первичный ключ Номер_продажи и Код_блюда):



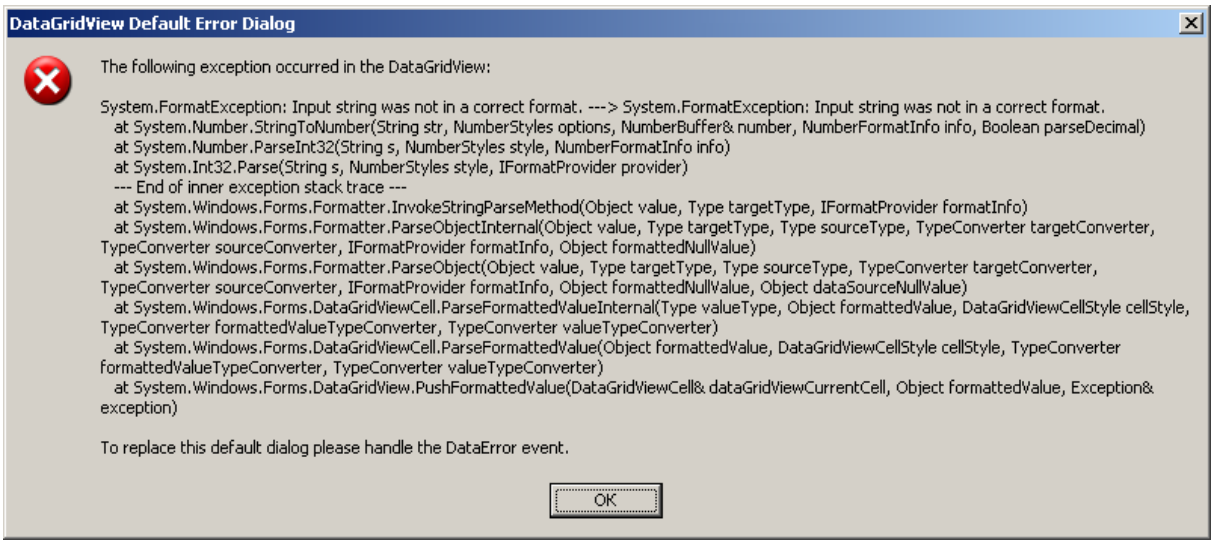
- в таблице Продано не указан Номер_продажи:



- в таблице Продано не указан Код_блюда:



- задан не тот тип данных в проданоDataGridView для целочисленных колонок Номер_продажи, Код_блюда, Количество:



- И другие исключения согласно наложенным ограничениям в БД (см. лабораторную работу №1).

Рассмотрим как, например, можно реализовать обработку исключений при оформлении данных в таблице **Продажи** (в случае указания одинаковых номеров продаж, в продаже указан Код_сотрудника, которого в БД не существует, превышено количество символов для ввода номера столика).

Для этого в коде **FormSalesList** замените **блок сохранения**:

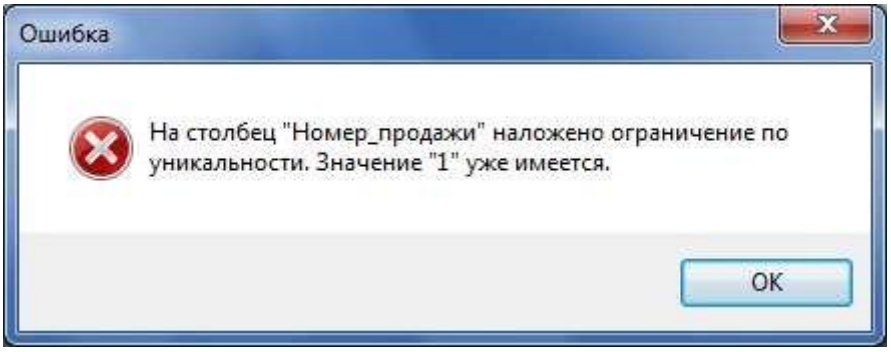
```
private void продажиBindingNavigatorSaveItem_Click(object sender, EventArgs
e)
        {
            this.Validate();
            this.продажиBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.cafeDataSet);

        }
```

На блок с конструкцией **try…catch**:

```
private void продажиBindingNavigatorSaveItem_Click(object sender, EventArgs
e)
        {
            try
            {
                  this.Validate();
                  this.продажиBindingSource.EndEdit();
                  this.tableAdapterManager.UpdateAll(this.cafeDataSet);
            }
            catch (Exception err)
            {
                  MessageBox.Show(err.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }

         }
```

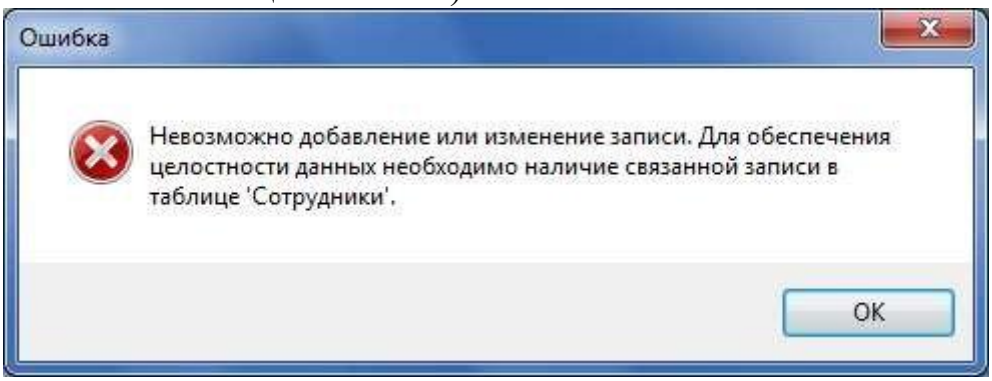
Теперь будет такая обработка исключений:

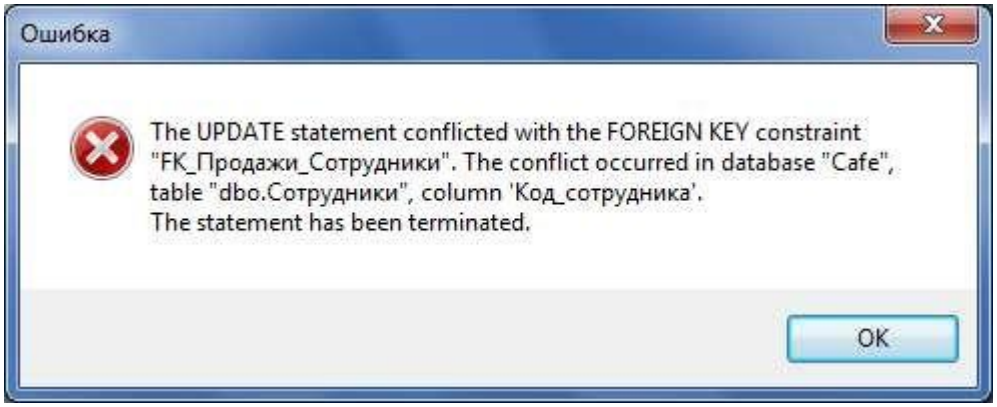
- в таблице Продажи указан повторяющийся Номер_продажи (в нашем случае – первичный ключ) (для **Microsoft Access, Microsoft SQL Server**):



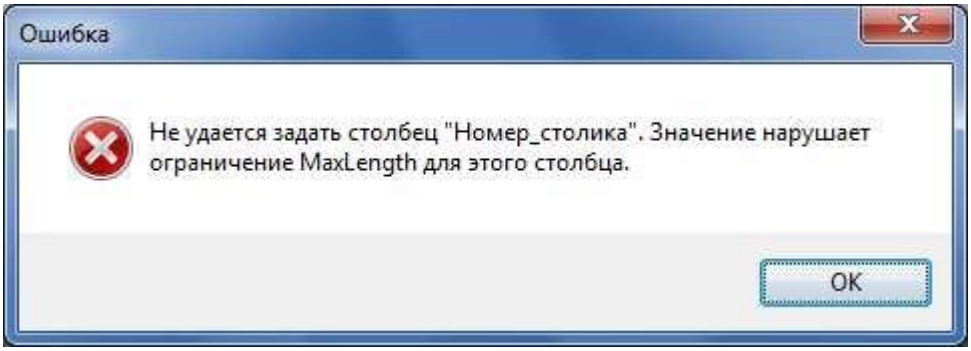
или

- в таблице Продажи указан Код_сотрудника, которого в БД (в родительской таблице Сотрудники) не существует (нами ранее была установлена ссылочная целостность):

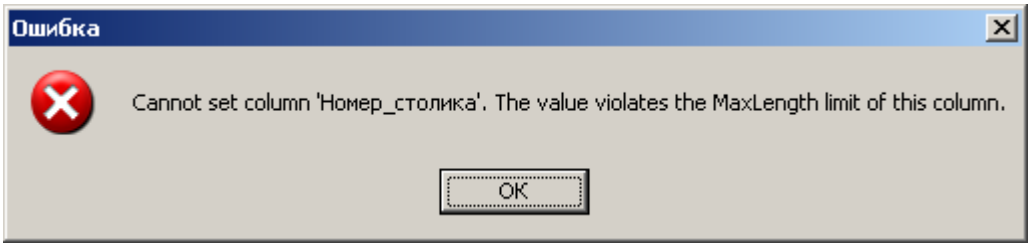




- в таблице Продажи превышено количество символов для ввода номера столика:



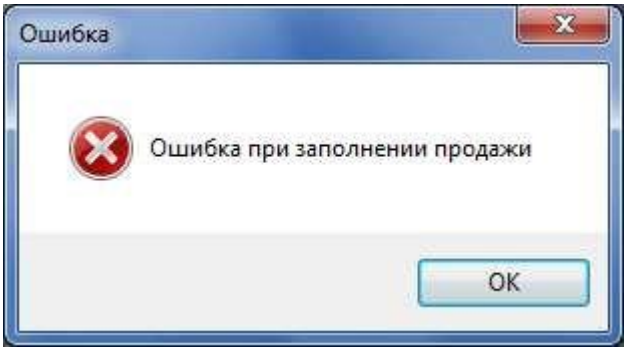
или



Или можно задать свой текст обработки исключения:

```
private void продажиBindingNavigatorSaveItem_Click(object sender, EventArgs
e)
        {
            try
            {
                  this.Validate();
                  this.продажиBindingSource.EndEdit();
                  this.tableAdapterManager.UpdateAll(this.cafeDataSet);
            }
            catch (Exception err)
            {
                  MessageBox.Show("Ошибка при заполнении продажи", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

        }
```

В этом случае:



По аналогии можно обрабатывать и другие исключения.

2.3 Поиск, фильтрация данных

Для удобства пользователей в приложении необходимо предусматривать поиск/фильтрацию необходимых данных.

Поиск и фильтрацию данных можно осуществить различными способами. Например, написание SQL-запросов (см. лабораторную работу №4), использование метода Find (BindingSource.Find), свойства Filter (BindingSource.Filter) и др.

Рассмотрим как, например, можно реализовать поиск/фильтрацию с использованием метода **Find,** свойства **Filter.**

2.3.1 Метод BindingSource.Find

Данный метод позволяет найти нужный элемент в источнике данных.
В данном методе возможны варианты:

2.3.1.1 Find(PropertyDescriptor, Object) - поиск индекса элемента по дескриптору и ключу.

Синтаксис:

public virtual int Find(PropertyDescriptor *prop*, Object *key*)

Где параметры:

***prop* (PropertyDescriptor):** тип System.ComponentModel.PropertyDescriptor, определяет дескриптор для поиска;

***key* (Object)** : тип System.Object - искомое значение (ключ).

Возвращаемое значение: тип System .Int32 - индекс найденного элемента (начинается с нуля). Значение -1, если элемент не найден.

Этот метод обычно используется, чтобы найти первую строку, в которой значение поля, заданного дескриптором, равно значению параметра ключа.

Например:

```
int indexPos = сотрудникиBindingSource.Find("Фамилия", textBox1.Text);
if (indexPos > -1)
    сотрудникиBindingSource.Position = indexPos;
else
    MessageBox.Show("Таких сотрудников нет");
```

2.3.1.2 Find(String, Object) - поиск индекса элемента по имени и значению.

Синтаксис:

public int Find(string *propertyName*, Object *key*)

Где параметры:

propertyName: тип System.String - имя свойства для поиска;

key: тип System.Object - искомое значение (ключ).

Возвращаемое значение: тип System .Int32 - индекс найденного элемента (начинается с нуля). Значение -1, если элемент не найден.

Например:

```
int indexPos = сотрудникиBindingSource.Find("Фамилия", "Петров");
сотрудникиBindingSource.Position = indexPos;
```

2.3.2 Свойство BindingSource.Filter

Данное свойство задает выражение, используемое для фильтрации строк.

Тип для данного свойства: **System .String -** строка, которая определяет, как строки необходимо фильтровать.

Для формирования значение фильтра необходимо указать имя столбца, затем оператор и значения для фильтра.

Если основной источник данных DataSet, DataTable или DataView, можно указать логические выражения с помощью синтаксиса свойства DataColumn .Expression.

Например:

сотрудникиBindingSource.Filter = "Фамилия = 'Петров'"

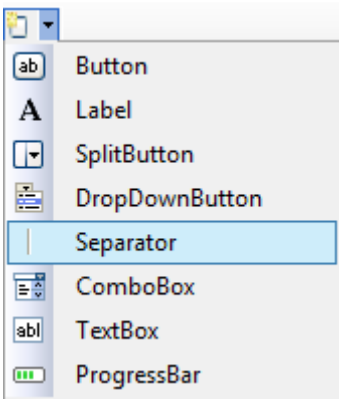
2.3.3 Реализация поиска и фильтрации данных

Реализуем, например, поиск/фильтрацию данных необходимого сотрудника. Данная возможность пригодится пользователям, работающим с данными по кадровому персоналу кафе, а также пользователям, оформляющим продажи для поиска необходимо кода сотрудника.

Будем осуществлять поиск/фильтр заданного в строке ввода значения по выбранной пользователем в **сотрудникиDataGridView** колонке.

Для этого перейдите на форму **FormWorkesList**.

На панели **сотрудникиBindingNavigator** рядом с кнопкой **Сохранить** (**сотрудникиBindingNavigatorSaveItem**) разместите разделитель (**Separator**) через **Add ToolStripButton**:



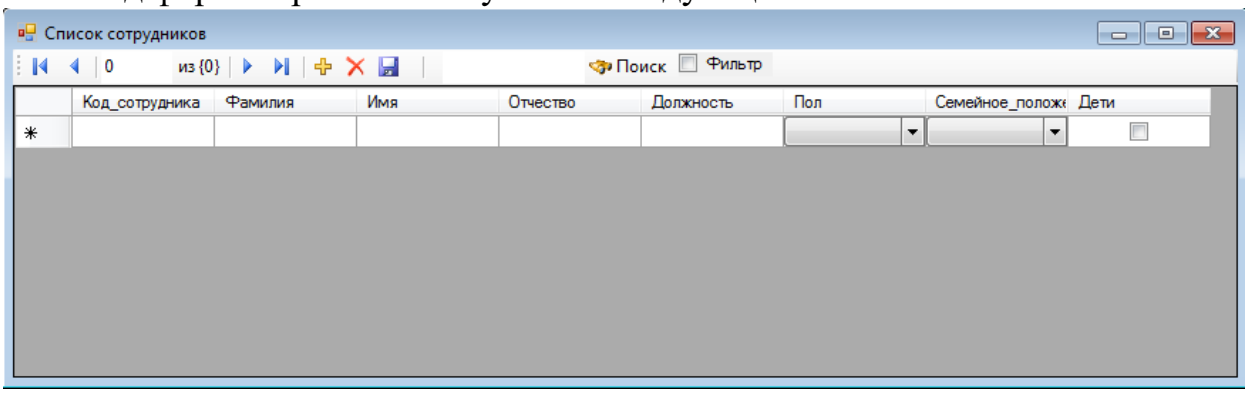
Или через **Edit Items:**

Рядом строку ввода **TextBox** для которой укажите:
- свойство **Name** в **toolStripTextBoxFind**.

Рядом с ней кнопку **Button** для поиска для которой укажите:
- свойство **Name** в **toolStripButtonFind**;
- свойство **DisplayStyle** в **ImageAndText**;
- свойство **Text** в **Поиск**;
- свойство T**extAlign** в **MiddleRight;**
- свойство **Image** в соответствующую картинку;
- свойство **ImageAlign** в **MiddleLeft**.

Рядом с ней компонент **CheckBox** с **Toolbox** для которого укажите:
- свойство **Name** в **checkBoxFind**;
- свойство **Text** в **Фильтр**.

Вид формы при этом получается следующий:



Как было сказано ранее, будем осуществлять поиск заданного в строке ввода **toolStripTextBoxFind** значения по выбранной пользователем в **сотрудникиDataGridView** колонке.

Пропишем обработчик для отбора выбранной пользователем колонки в **сотрудникиDataGridView**.

Для этого в коде **FormWorkesList** пропишите:

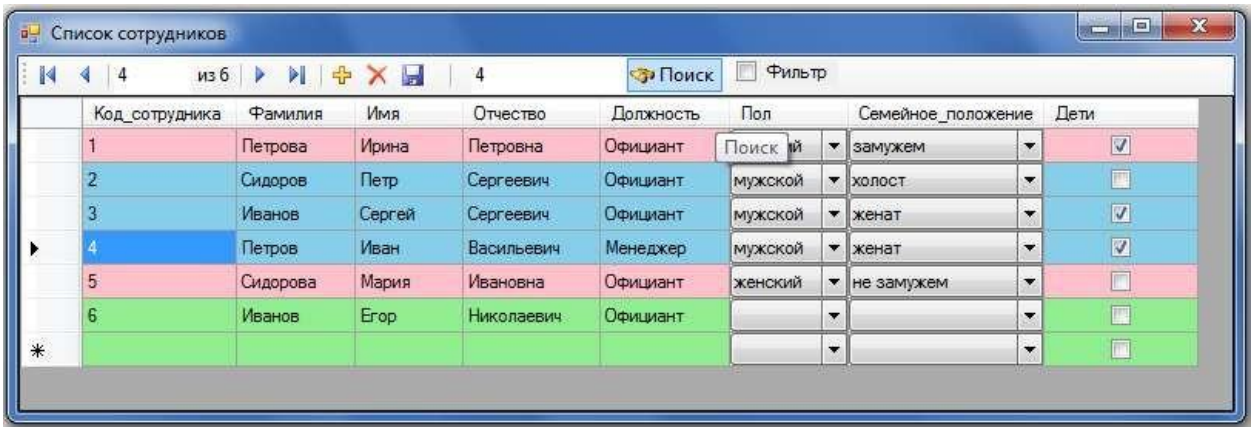
```
string GetSelectedFieldName()
        {
            return
сотрудникиDataGridView.Columns[сотрудникиDataGridView.CurrentCell.ColumnIndex
].DataPropertyName;
        }
```

На обработчике события **Click** кнопки **Поиск (toolStripButtonFind)** пропишите:

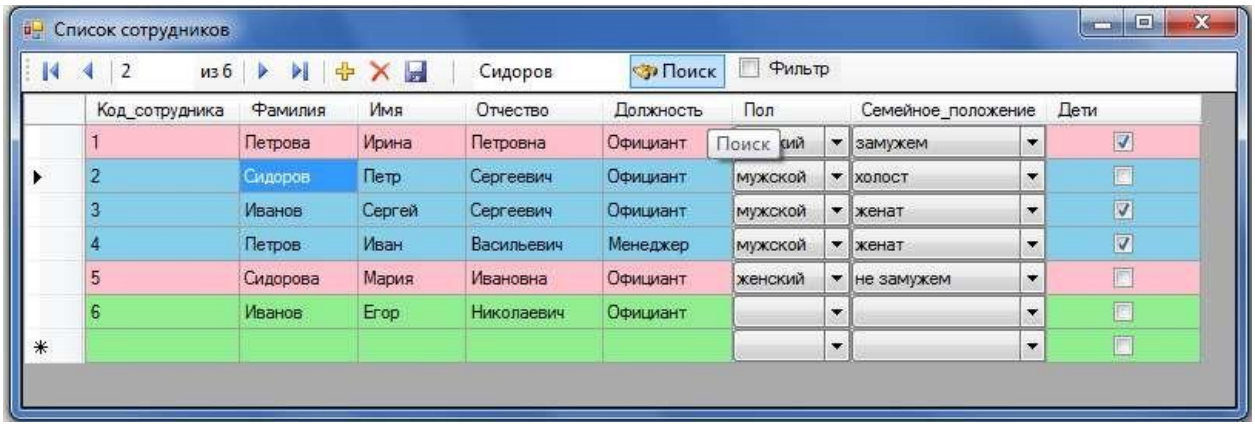
```
private void toolStripButtonFind_Click(object sender, EventArgs e)
        {
            if (toolStripTextBoxFind.Text == "")
            {
                MessageBox.Show("Вы ничего не задали", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                return;
            }

            int indexPos;

            try
            {
                indexPos =
сотрудникиBindingSource.Find(GetSelectedFieldName(),
toolStripTextBoxFind.Text);
            }
            catch (Exception err)
            {
                MessageBox.Show("Ошибка поиска \n" + err.Message);
                return;
            }

            if (indexPos > -1)
                  сотрудникиBindingSource.Position = indexPos;
              else
            {
                MessageBox.Show("Таких сотрудников нет", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                сотрудникиBindingSource.Position = 0;
            }
        }
```

Запустите программу. Посмотрите, как отрабатывает поиск данных, например:

Поиск по колонке Код_сотрудника:



Поиск по колонке Фамилия:



Закройте программу. Вернитесь в проект.

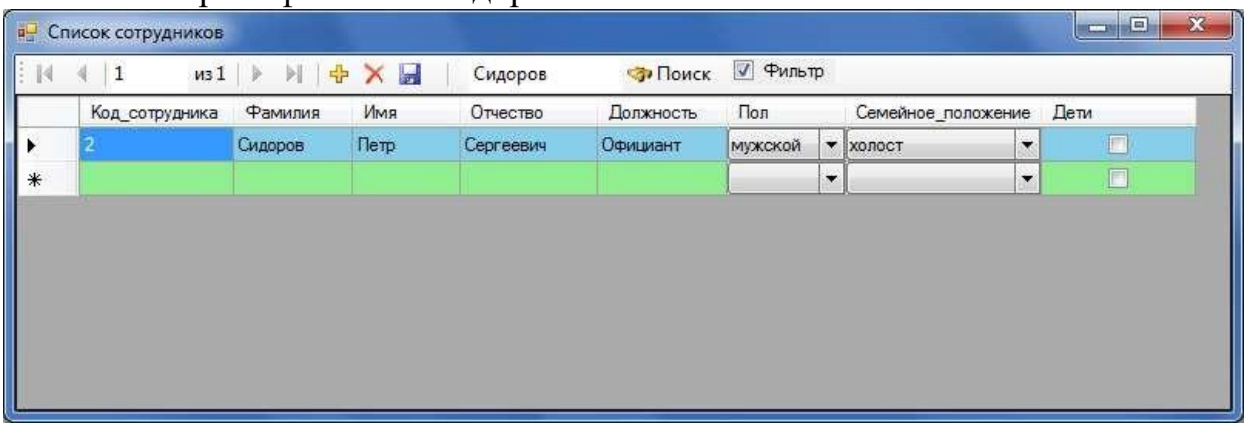
На обработчике **checkBoxFind** для **фильтрации данных** пропишите:

```
private void checkBoxFind_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxFind.Checked)
            {
                if (toolStripTextBoxFind.Text == "")
                    MessageBox.Show("Вы ничего не задали", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                else
                    try
                    {
                        сотрудникиBindingSource.Filter =
GetSelectedFieldName() + "='" + toolStripTextBoxFind.Text + "'";

                    }
                    catch (Exception err)
                    {
                        MessageBox.Show("Ошибка фильтрации \n" +
err.Message);
                    }
            }
            else
                сотрудникиBindingSource.Filter = "";

            if (сотрудникиBindingSource.Count == 0)
            {
                MessageBox.Show("Нет таких");
```

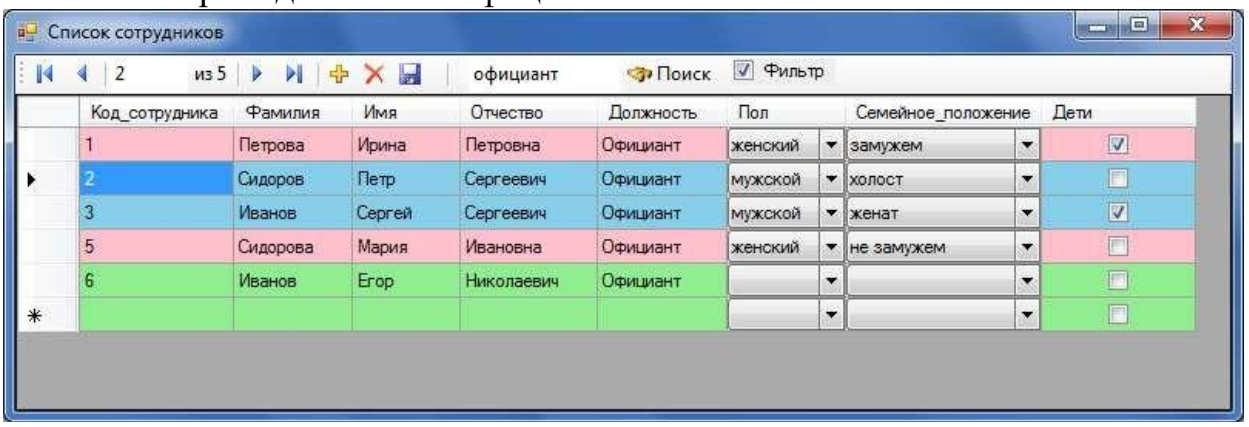
```
            сотрудникиBindingSource.Filter = "";
            checkBoxFind.Checked = false;
        }

    }
```

Запустите программу. Посмотрите, как отрабатывает фильтрация данных, например:

Фильтр по фамилии Сидоров:



Фильтр по должности официант:



Закройте программу. Вернитесь в проект.

Для удобства пользователей по аналогии реализуются поиск/фильтрация данных по блюдам, продажам.

2.4 Связь 1:M между Сотрудники и Продажи в приложении

Как было сказано ранее, в БД кафе таблица **Сотрудники** связана с таблицей **Продажи** связью 1:M.

Рассмотрим как можно, например, реализовать данную связь в приложении.

При работе с продажами не всегда удобно помнить код необходимого сотрудника.

Кроме того, следует отметить, что нами была установлена ссылочная целостность в БД, т.е. должен быть указан код существующего в БД сотрудника, иначе будет ошибка при работе с данными (см. раздел 2.2).

В нашем случае, чтобы посмотреть код необходимого сотрудника необходимо перейти на форму Сотрудники и посмотреть для него код.

Данную ситуацию можно изменить. Например, можно организовать вызов формы с сотрудниками из формы Продажи.

Для этого перейдите на форму Продажи (**FormSalesList)**.

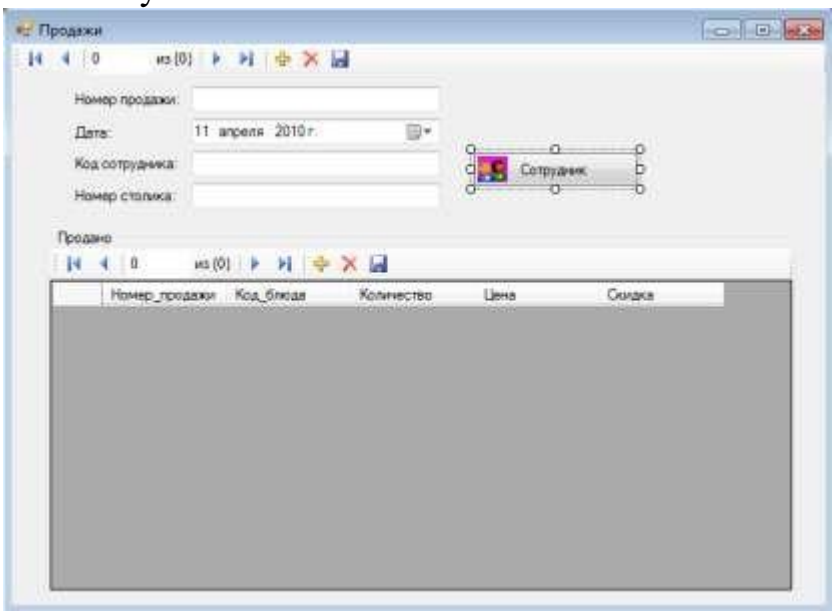
Рядом с **код_сотрудникаTextBox** разместите с панели инструментов **Toolbox** кнопку **button**.

Укажите для **button**:

- свойство **Name** в **buttonWorker**;
- свойство **Text** в **Сотрудник**.

При желании данной кнопке можно указать необходимую картинку через свойство **Image**. А через свойство **ImageAlign** можно указать в каком месте данную картинку на этой кнопке выводить, например, для слева по центру укажите **MiddleLeft**.

В результате получаем:



По нажатию на данную кнопку **buttonWorker** (**Сотрудник**) будем показывать форму с данными по сотрудникам. В данной форме пользователь выбирает необходимого ему сотрудника и его данные (код сотрудника) заносятся в соответствующую продажу.

Следует напомнить, что таблицы **Сотрудники** и **Продажи** связаны связью 1:М по колонке Код_сотрудника.

Когда пользователь будет выбирать сотрудника в форме Сотрудники, необходимо запомнить код выбранного сотрудника и передать его в

соответствующее поле таблицы **Продажи** Код_сотрудника (реализовать связь между **Сотрудники** и **Продажи** (1:M) по **Код_сотрудника**).

Перейдите на форму Сотрудники (**FormWorkesList**).

Чтобы пользователю было удобней выбирать сотрудника, добавьте на форме **FormWorkesList** на панели **сотрудникиBindingNavigator** кнопку **Button** для которой укажите:

- свойство **Name** в **toolStripButtonOK**;
- свойство **DisplayStyle** в **ImageAndText**;
- свойство **Text** в **Выбрать**;
- свойство **TextAlign** в **MiddleRight**;
- свойство **Image** в соответствующую картинку;
- свойство **ImageAlign** в **MiddleLeft**;
- в событии **Click** укажите:

```
private void toolStripButtonOK_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.OK;
        }
```

Примечание:

Если Вами была добавлена кнопка **Button** с панели инструментов **Toolbox**, то для нее можно непосредственно через **Properties** свойство **DialogResult** указать в OK.

Реализуем следующую логику:

если пользователь нажал кнопку **Выбрать** (**toolStripButtonOK**), то запоминаем код выбранного сотрудника. Если он закрыл форму, нажав , то код сотрудника не запоминаем, считая, что пользователь передумал выбирать сотрудника.

Кнопку **Выбрать** (**toolStripButtonOK**) пользователь видит только когда открывает список сотрудников из формы Продажи.

Реализуем придуманную логику.

В коде формы **FormWorkesList** пропишите:

```
//переменная для текущего(выбранного) кода сотрудника
        int idCurrent = -1;

        public int ShowSelectForm(int id)
        {
```

```
            toolStripButtonOK.Visible = true;
            idCurrent = id;
            if (ShowDialog() == DialogResult.OK)
                return
(int)((DataRowView)сотрудникиBindingSource.Current)["код_сотрудника"];
            else
                return -1;
        }
```

Перейдите на форму **FormSalesList.**

В коде **FormSalesList** пропишите обработчик нажатия кнопки **Сотрудник (buttonWorker)** :

```
private void buttonWorker_Click(object sender, EventArgs e)
        {
            int id = -1;
            if
(((DataRowView)продажиBindingSource.Current)["код_сотрудника"].ToString() !=
"")
            {
                id =
(int)(((DataRowView)продажиBindingSource.Current)["код_сотрудника"]);
            }
            id = FormWorkesList.fw.ShowSelectForm(id);
            if (id >= 0)
            {
                MessageBox.Show(id.ToString());
                ((DataRowView)продажиBindingSource.Current)["код_сотрудника"]
= id;
                продажиBindingSource.EndEdit();
            }

        }
```

Запустите программу. Посмотрите полученную возможность.

Обратите внимание, что при открытии формы Сотрудники по кнопке **buttonWorker** курсор уснавливается на первую запись, а не на текущего сотрудника в данной продаже.

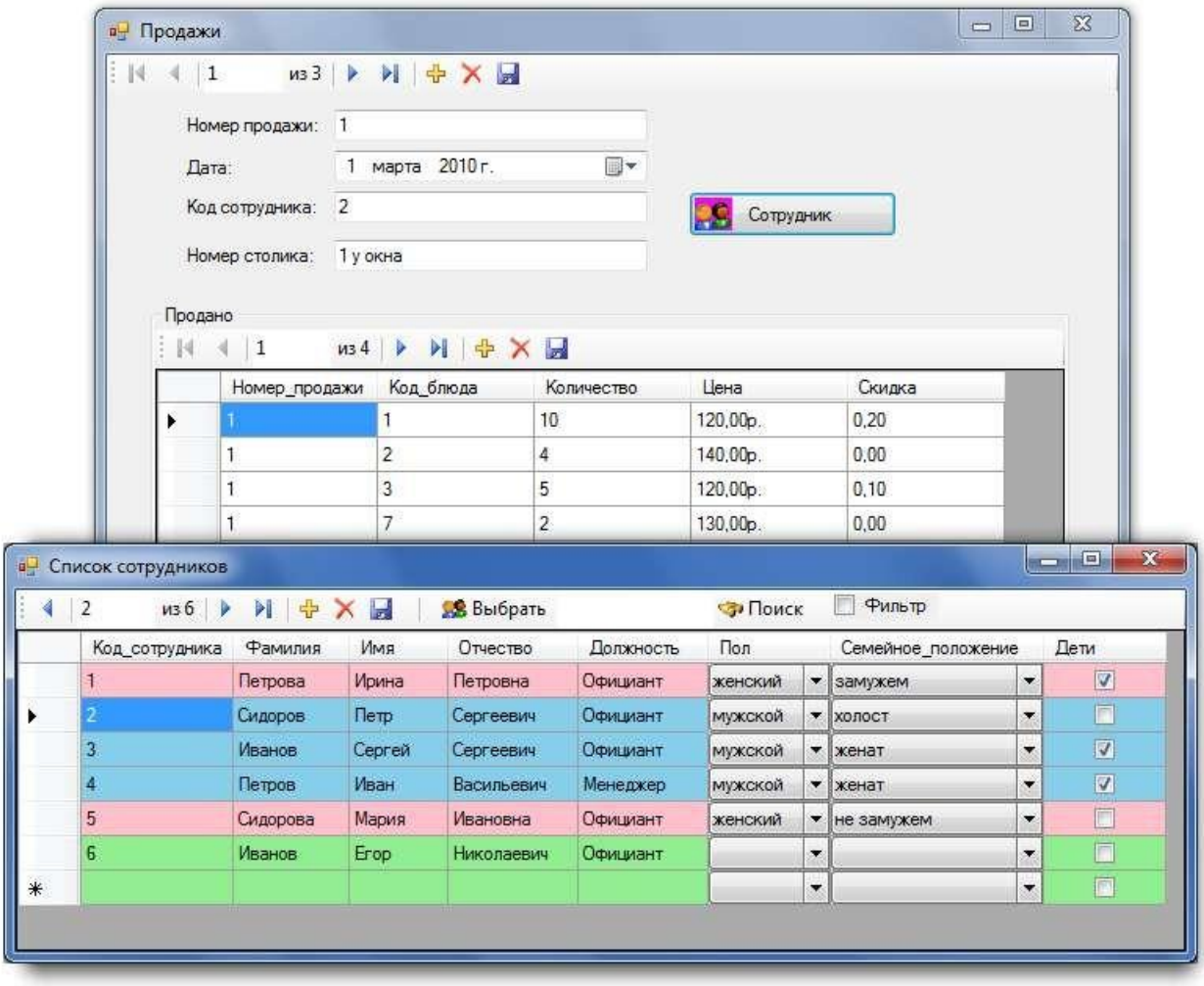
Вернитесь в проект, чтобы эту ситуацию изменить.

Перейдите на форму **FormWorkesList.**

В событии **Shown** для нее пропишите установку позиции на текущего сотрудника в продаже:

```
private void FormWorkesList_Shown(object sender, EventArgs e)
        {
            сотрудникиBindingSource.Position =
сотрудникиBindingSource.Find("код_сотрудника", idCurrent);
        }
```

Запустите программу. Посмотрите, что при открытии формы Сотрудники курсор устанавливается на заданного сотрудника в Продажах:



При открытии же справочника сотрудников не из формы для продаж, в сотрудниках не нужно устанавливать курсор на текущего в продаже сотрудника, а также необходимо спрятать кнопку **Выбрать** (**toolStripButtonOK**).

Для этого измените код для метода **ShowForm()** на форме **FormWorkesList**:

```
public void ShowForm()
        {
            toolStripButtonOK.Visible = false;
            сотрудникиBindingSource.Position = 0;
            Show();
            Activate();
        }
```

Теперь позиция будет на первой записи при открытии справочника сотрудников не через форму Продажи, например, через пункты меню.

Примечание:

Помимо формы можно использовать и другие способы взятия информации из таблицы **Сотрудники** в **Продажи**, например, использование **ListBox, ComboBox** и др.

2.5 Создание вычисляемых колонок

Таблицы могут содержать колонки, значения которых получаются расчетным путем. Например, в записях элементов **Продано** имеет смысл вычислять колонку **Стоимость** (произведение количества на цену с учетом скидки). Вычисляемые колонки могут также содержаться в дочерних записях.

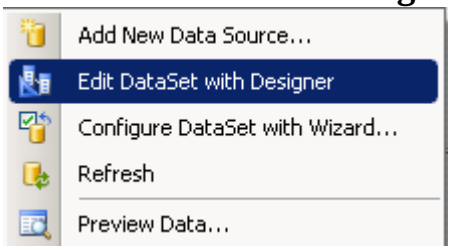
Для создания вычисляемых колонок используются выражения (**Expression**).

Синтаксис выражений состоит из стандартных арифметических операций, булевых и строковых операторов, а также констант. Вы можете сослаться на значения некоторых данных, используя имена колонок и таблиц (как в SQL-запросах), включать агрегирующие функции.

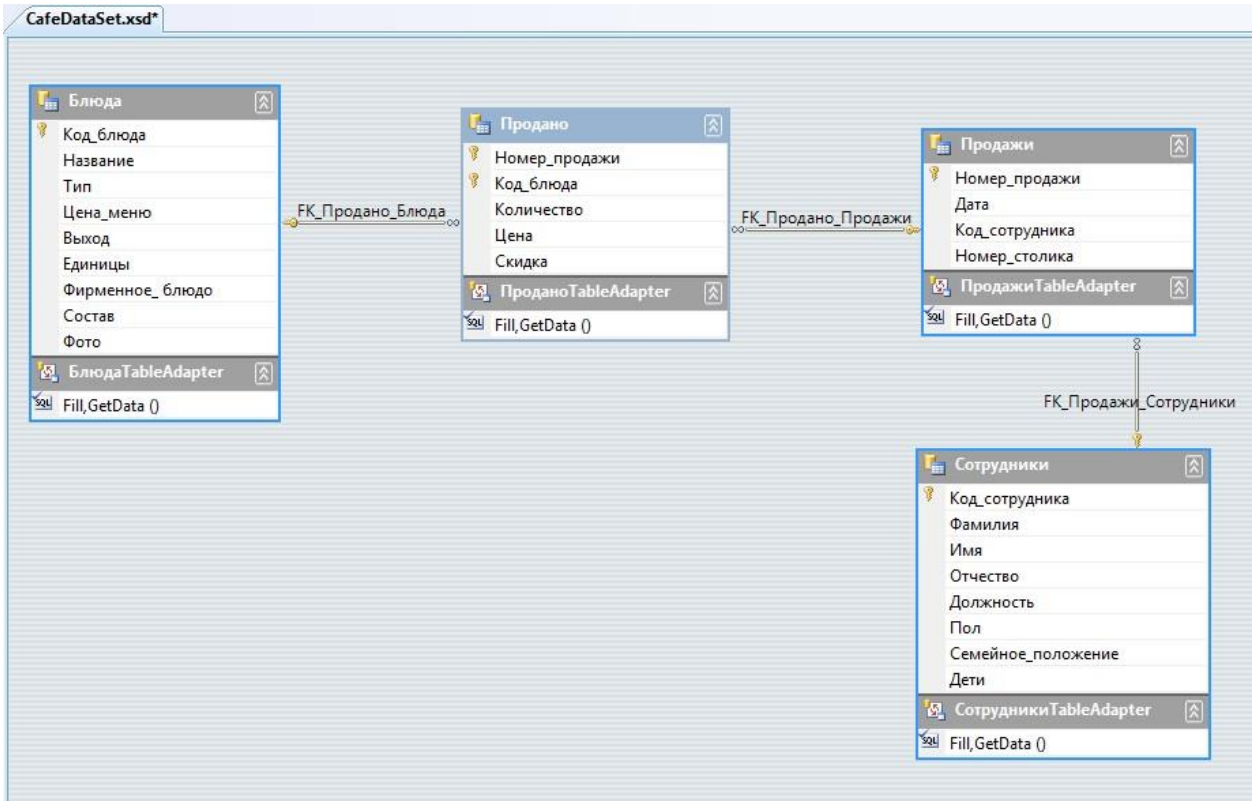
Вычисляемые колонки можно создавать и через непосредственно SQL запрос (см. лабораторную работу №4).

Для создания вычисляемых колонок в наборе данных необходимо выполнить следующие шаги.

Если схема еще не открыта в дизайнере, щелкните правой кнопкой на наборе данных и выберете **Edit DataSet with Designer**:

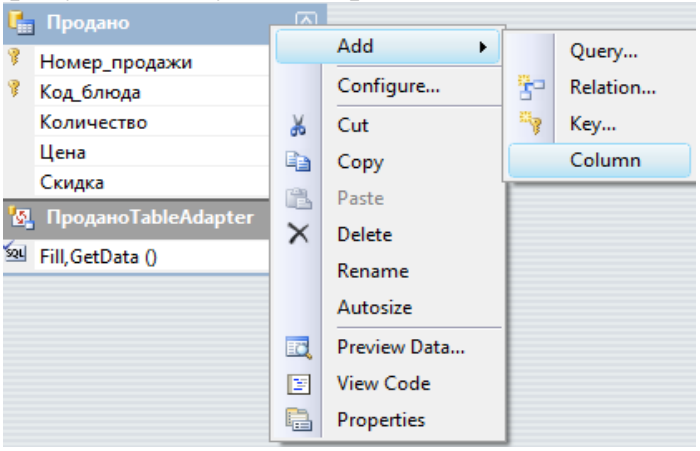


Или два раза щелкните мышью на XML-схеме (xsd-файл) в **Solution Explorer**. В нашем случае на файле **CafeDataSet.xsd**.

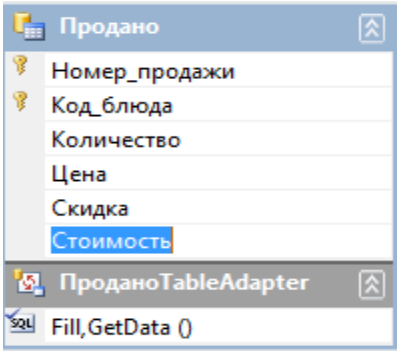


В сетке, представляющей таблицу, с которой вы хотите работать, добавьте новую колонку, выполняя следующие шаги:

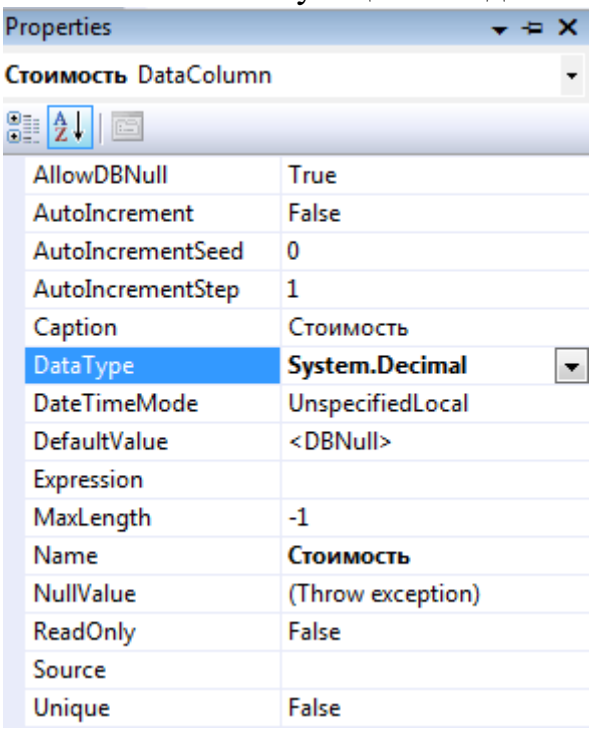
- нажмите правую кнопку и выберете **Add\Column**:



- введите имя колонки, например, **Стоимость**;

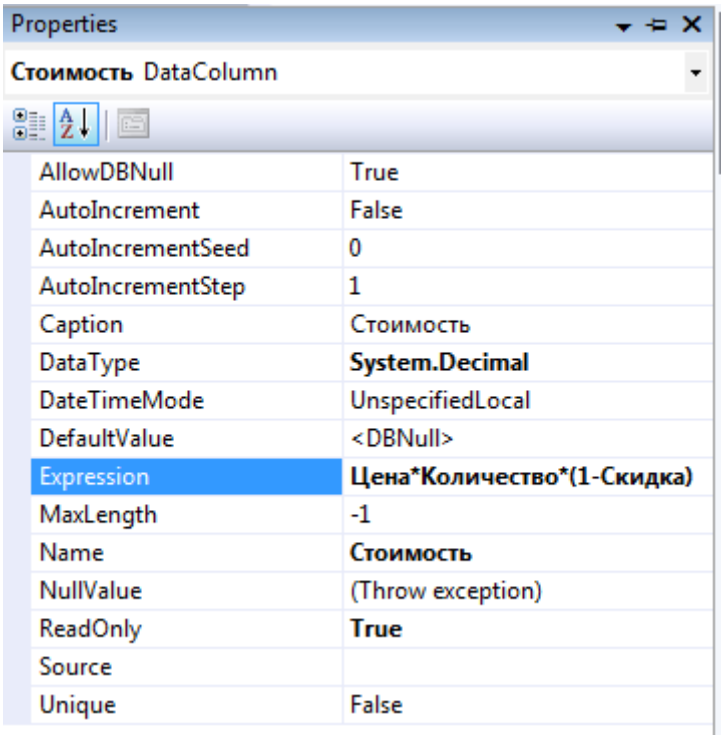


- в свойствах укажите соответствующий тип данных:



- оставаясь на добавленной колонке в сетке таблицы, перейдите в окно **Properties** и найдите там свойство **Expression**. В данной строке введите значение вычисляемого выражения, например, в данном случае **Стоимость вычисляется с учетом скидки**:

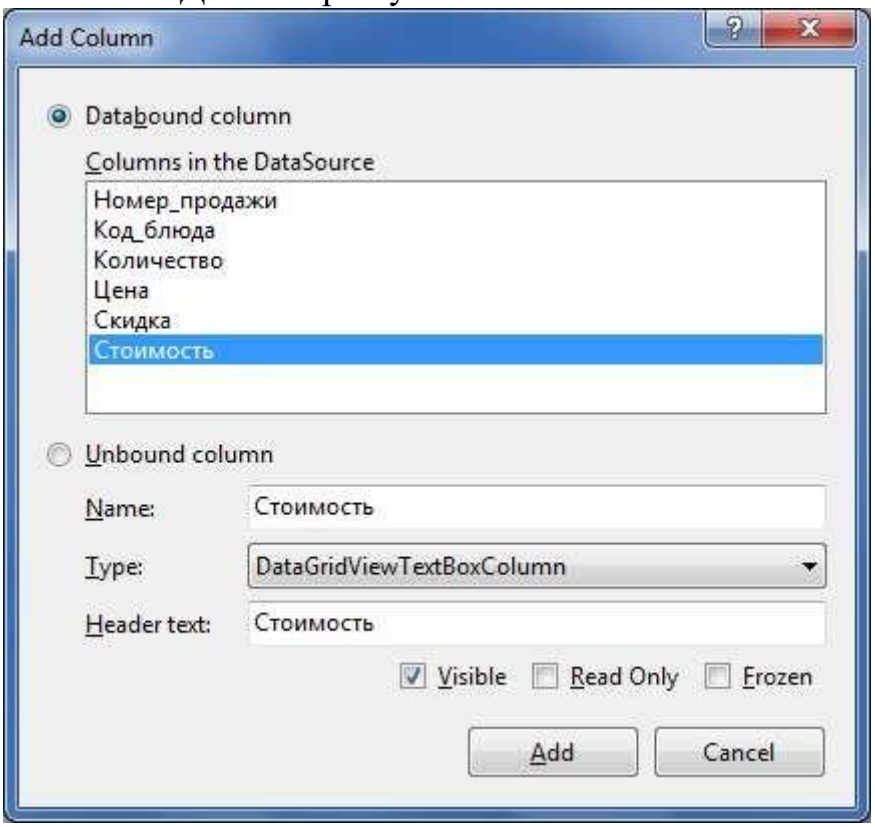
Цена*Количество*(1-Скидка)



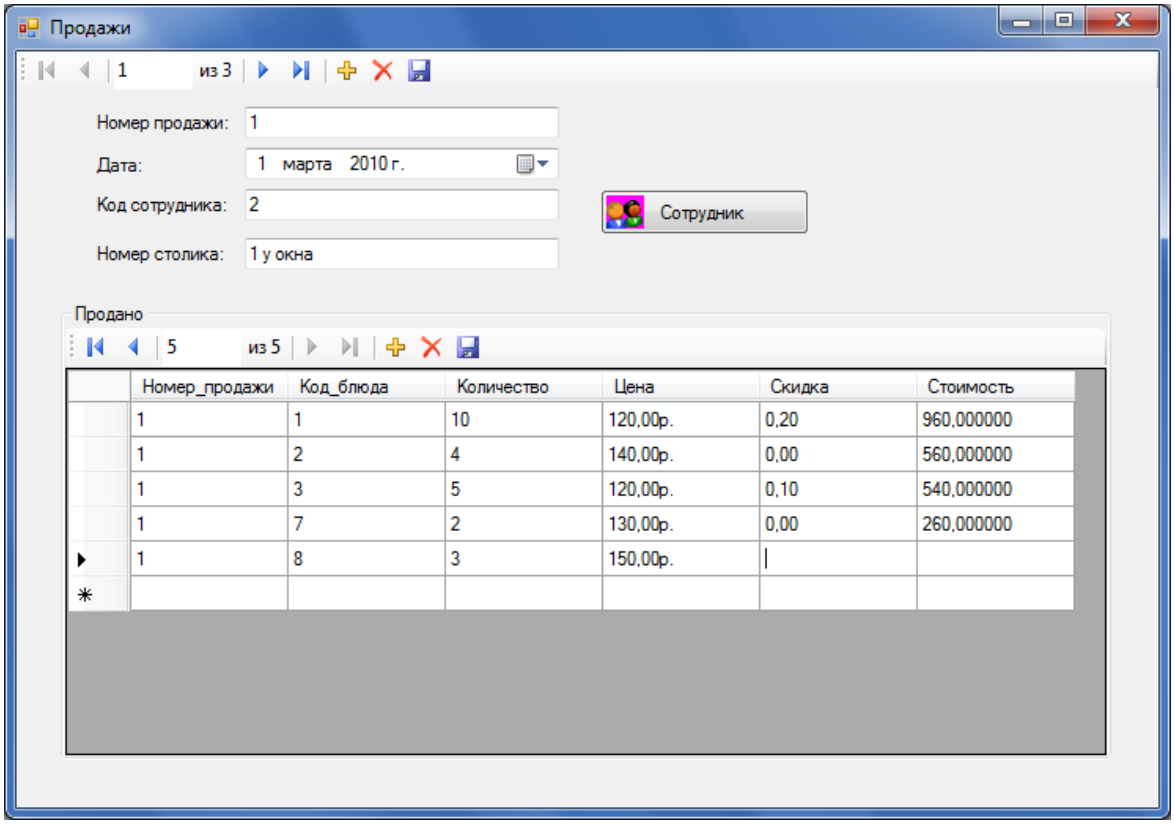
Перейдите на форму **FormSalesList.**

Теперь настроим **проданоDataGridView**, чтобы он отобразил новое поле **Стоимость**.

Для этого нажмите правую кнопку на **проданоSaleDataGridView** и укажите **Add Column**. Для которой укажите поле **Стоимость**:



Настройте размеры формы, чтобы можно было увидеть данную колонку в приложении. Запустите программу и посмотрите настроенную возможность.

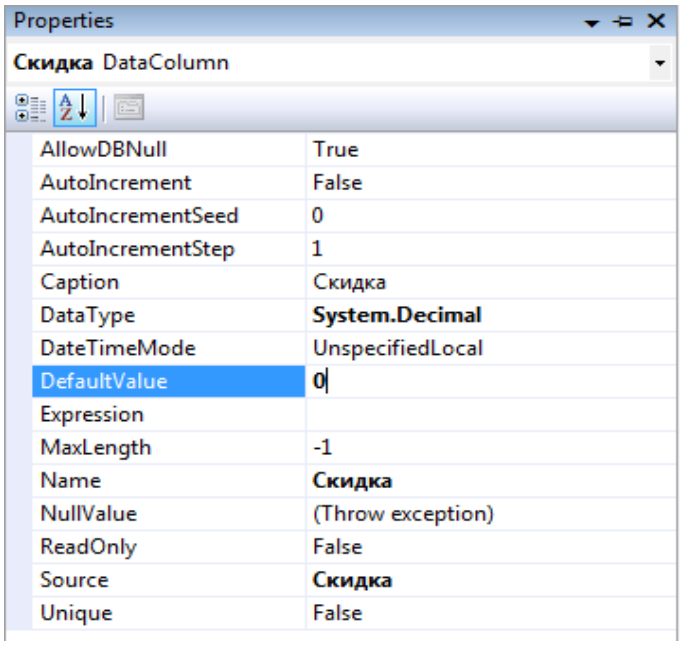


Номер_продажи	Код_блюда	Количество	Цена	Скидка	Стоимость
1	1	10	120,00р.	0,20	960,000000
1	2	4	140,00р.	0,00	560,000000
1	3	5	120,00р.	0,10	540,000000
1	7	2	130,00р.	0,00	260,000000
1	8	3	150,00р.		

Обратите внимание, что если скидка не указана, т.е. содержит Null значение, то стоимость не вычисляется.

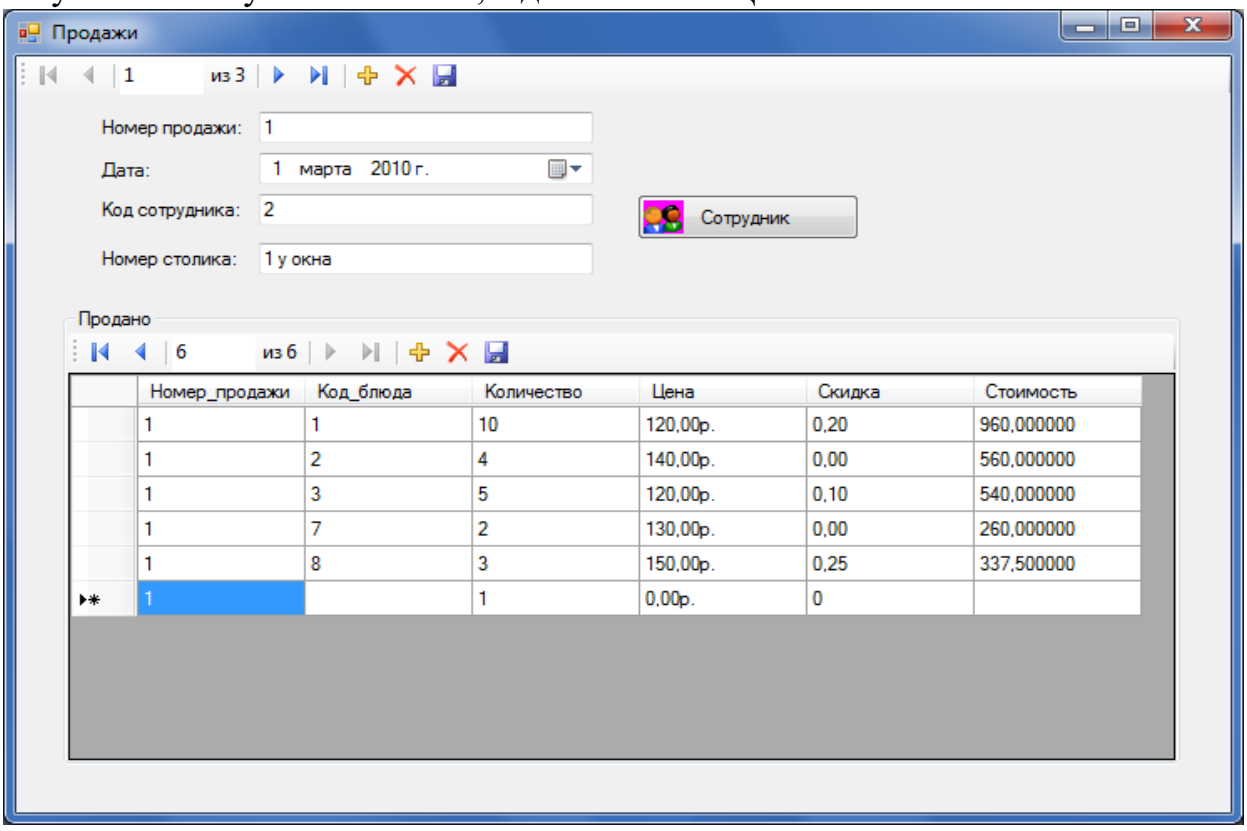
Чтобы этого избежать, необходимо для колонки **Скидка** установить значение по умолчанию 0.

Для этого в дизайнера набора данных укажите для поля **Скидка** значение по умолчанию 0:



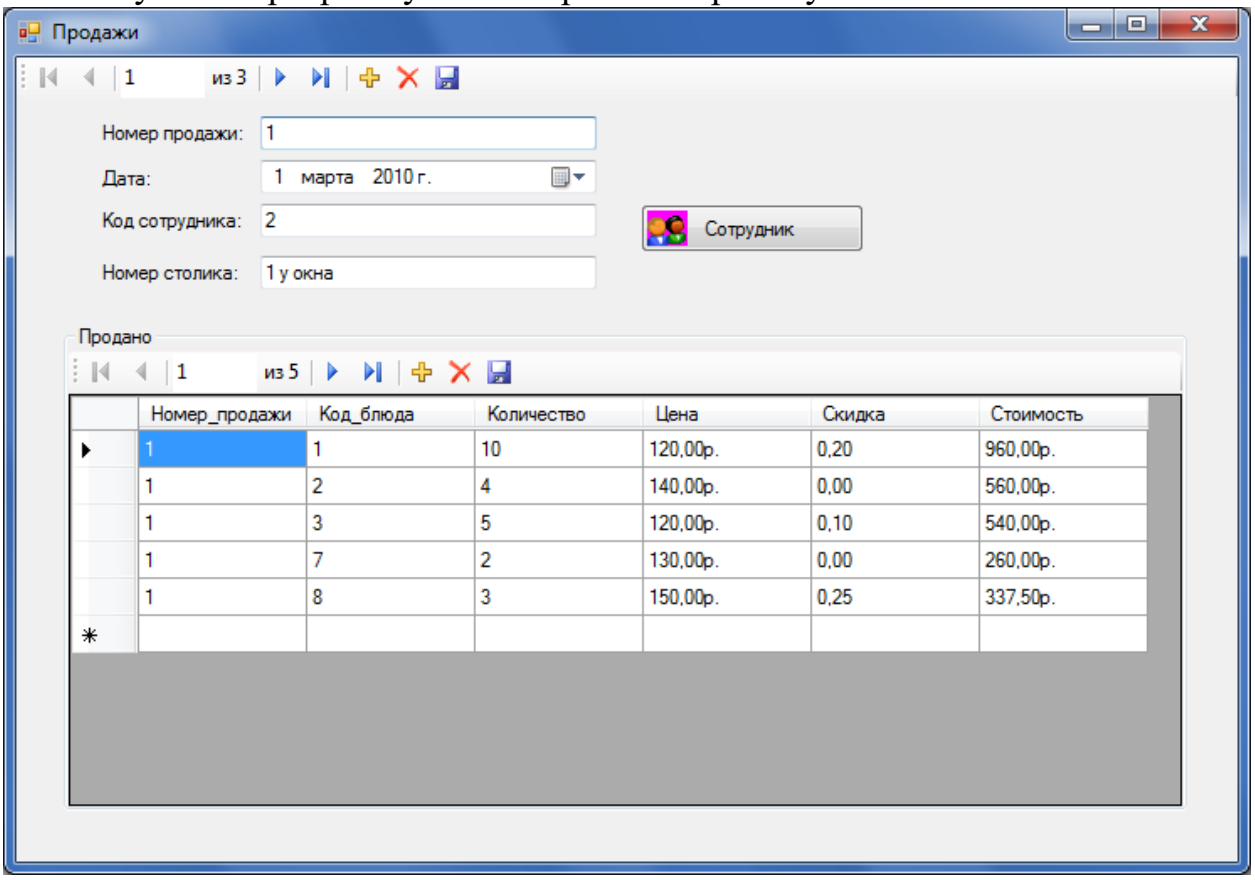
Теперь при работе с данными значение по умолчанию для скидки равно 0.

По аналогии можно для удобства для колонки **Количество** значение по умолчанию установить в 1, а для колонки **Цена** в 0.



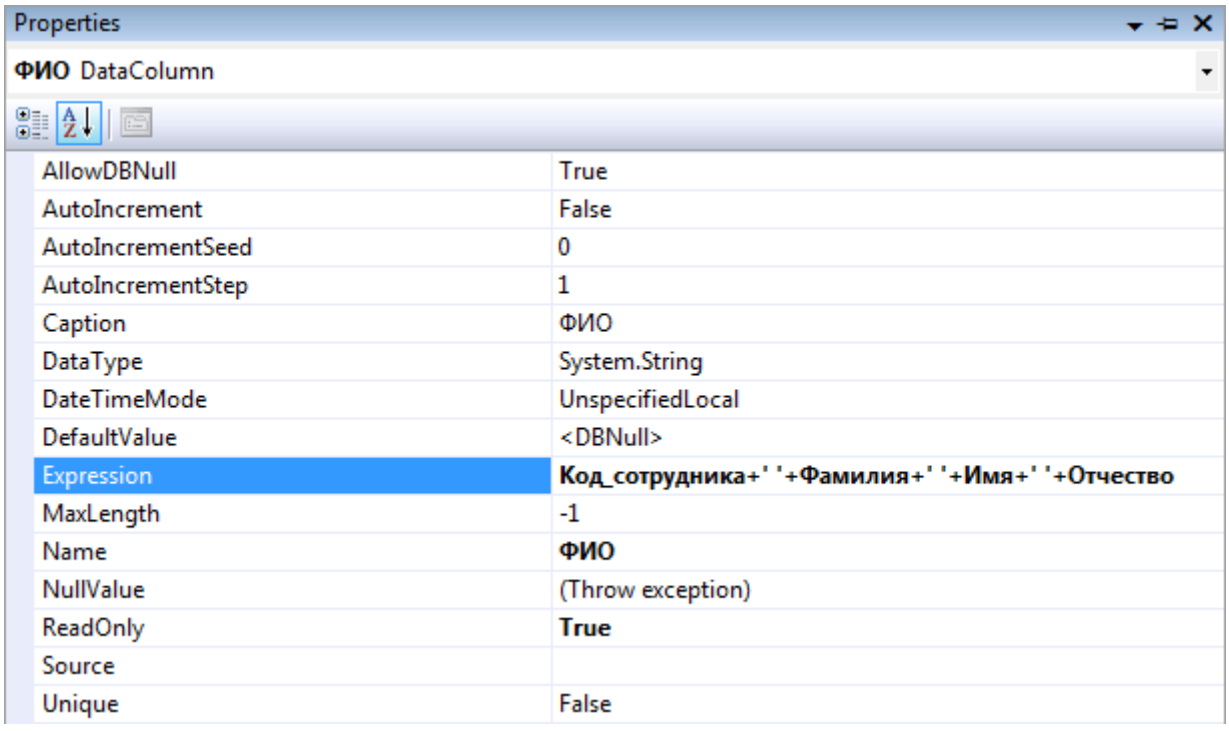
По аналогии с колонкой **Цена** укажите для колонки **Стоимость** денежный формат с точностью в 2 знака (см. раздел 2.1).

Запустите программу. Посмотрите настроенную возможность.

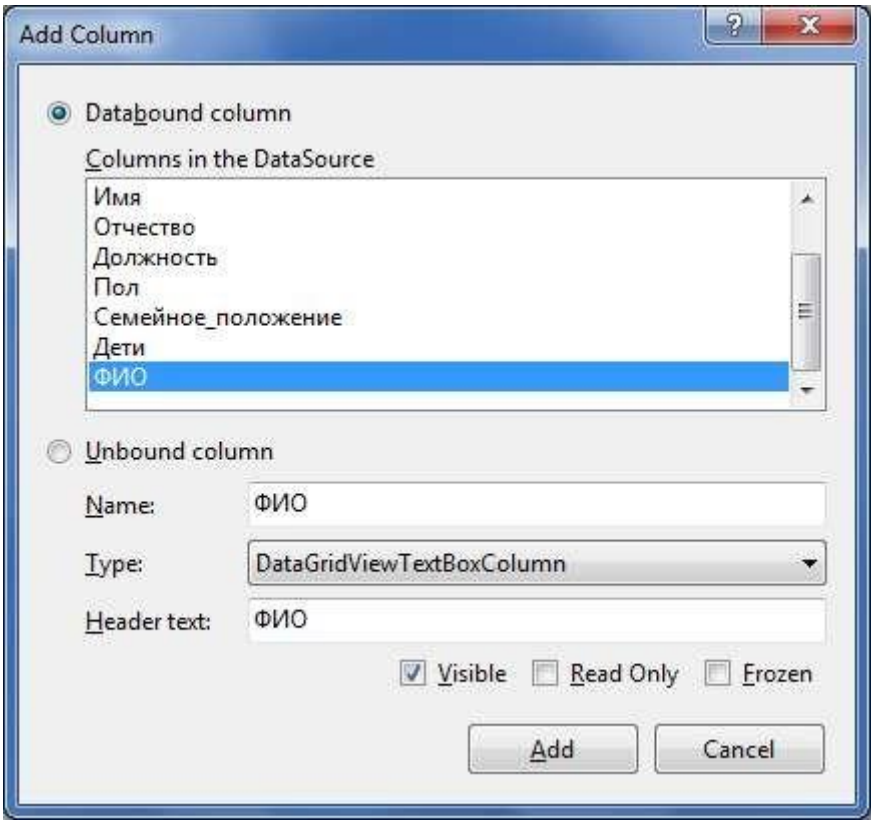


Номер_продажи	Код_блюда	Количество	Цена	Скидка	Стоимость
1	1	10	120,00р.	0,20	960,00р.
1	2	4	140,00р.	0,00	560,00р.
1	3	5	120,00р.	0,10	540,00р.
1	7	2	130,00р.	0,00	260,00р.
1	8	3	150,00р.	0,25	337,50р.

Аналогично создайте вычисляемую колонку **ФИО** строкового типа в таблице **Сотрудники:**



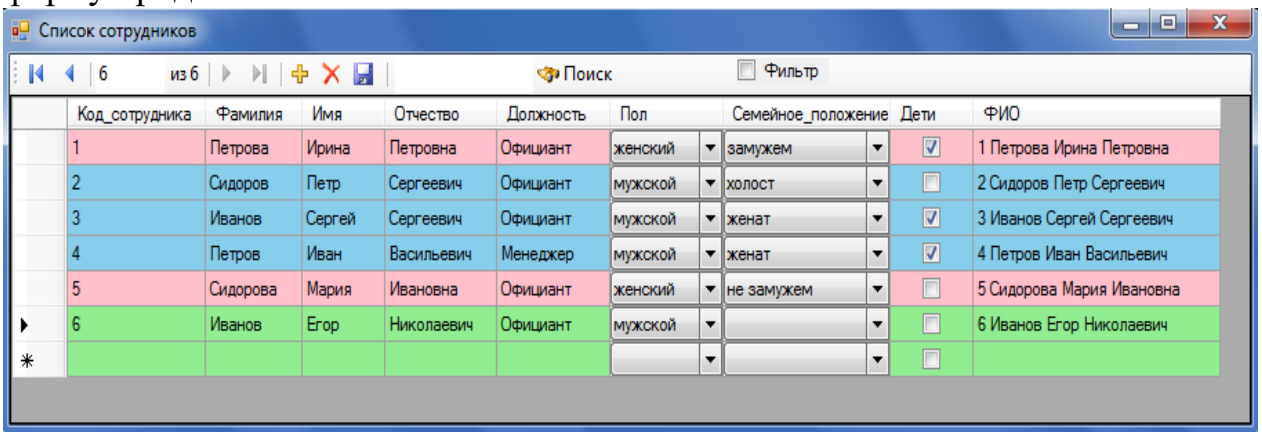
По аналогии укажите данное поле в **сотрудникиDataGridView**:



Настройте, чтобы в приложении была видна полученная колонка.

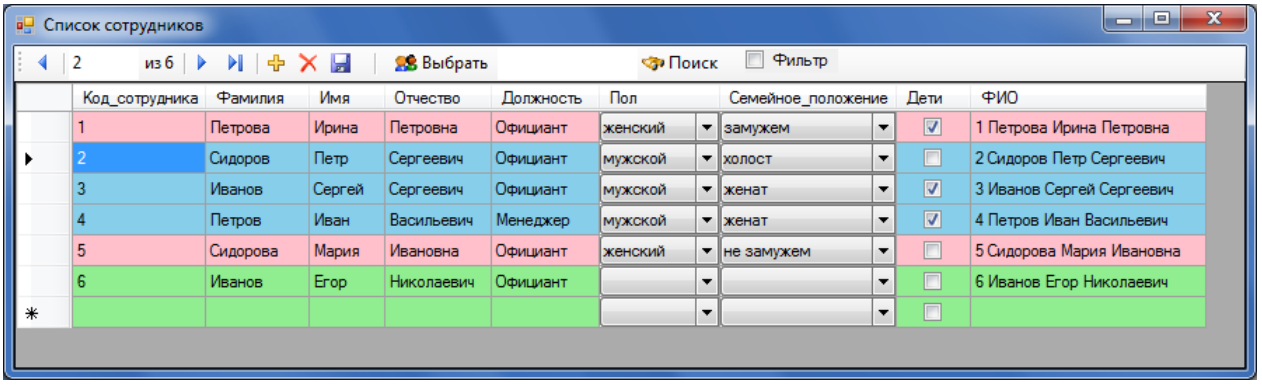
Запустите программу, посмотрите отработку вычисляемого поля ФИО:

На рисунке показана форма списка сотрудников, открытая не через форму продаж:



Код_сотрудника	Фамилия	Имя	Отчество	Должность	Пол	Семейное_положение	Дети	ФИО
1	Петрова	Ирина	Петровна	Официант	женский	замужем	☑	1 Петрова Ирина Петровна
2	Сидоров	Петр	Сергеевич	Официант	мужской	холост	☐	2 Сидоров Петр Сергеевич
3	Иванов	Сергей	Сергеевич	Официант	мужской	женат	☑	3 Иванов Сергей Сергеевич
4	Петров	Иван	Васильевич	Менеджер	мужской	женат	☑	4 Петров Иван Васильевич
5	Сидорова	Мария	Ивановна	Официант	женский	не замужем	☐	5 Сидорова Мария Ивановна
6	Иванов	Егор	Николаевич	Официант	мужской		☐	6 Иванов Егор Николаевич

Если через форму продаж, то форма с выделенным текущим сотрудником в продаже и с кнопкой Выбрать:



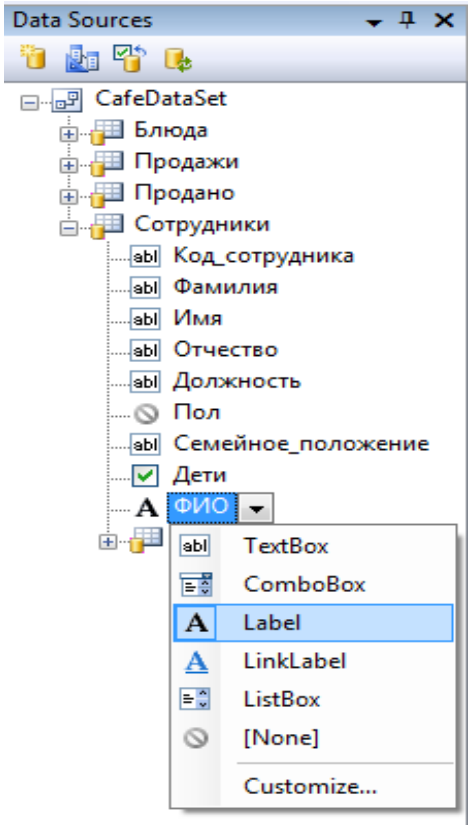
По аналогии создаются и другие вычисляемые поля.

2.6 Создание подстановочных полей (lookup)

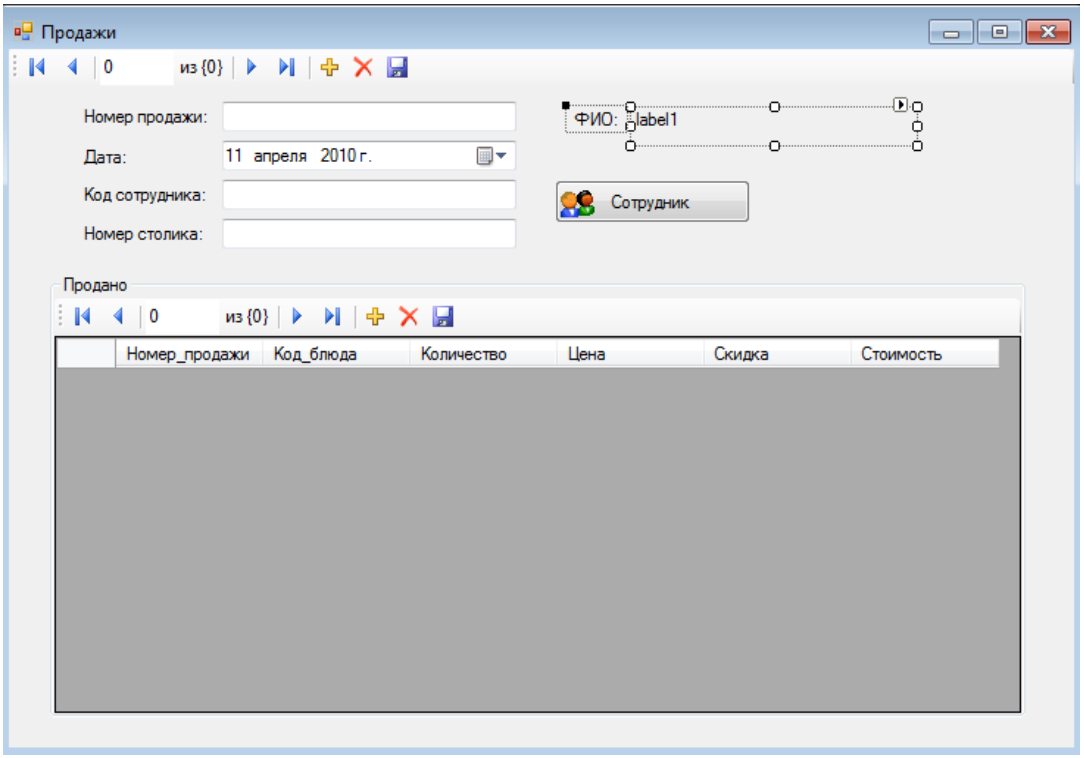
Теперь у нас есть вычисляемое поле ФИО. Можно для удобства пользователей сделать на форме Продажи показ не только кода сотрудника, но и его ФИО с кодом.

Перейдите на форму Продажи (**FormSalesList).**

Выберете из источника данных DataSources на форму данное поле из таблицы **Сотрудники** в виде метки (**Label**):

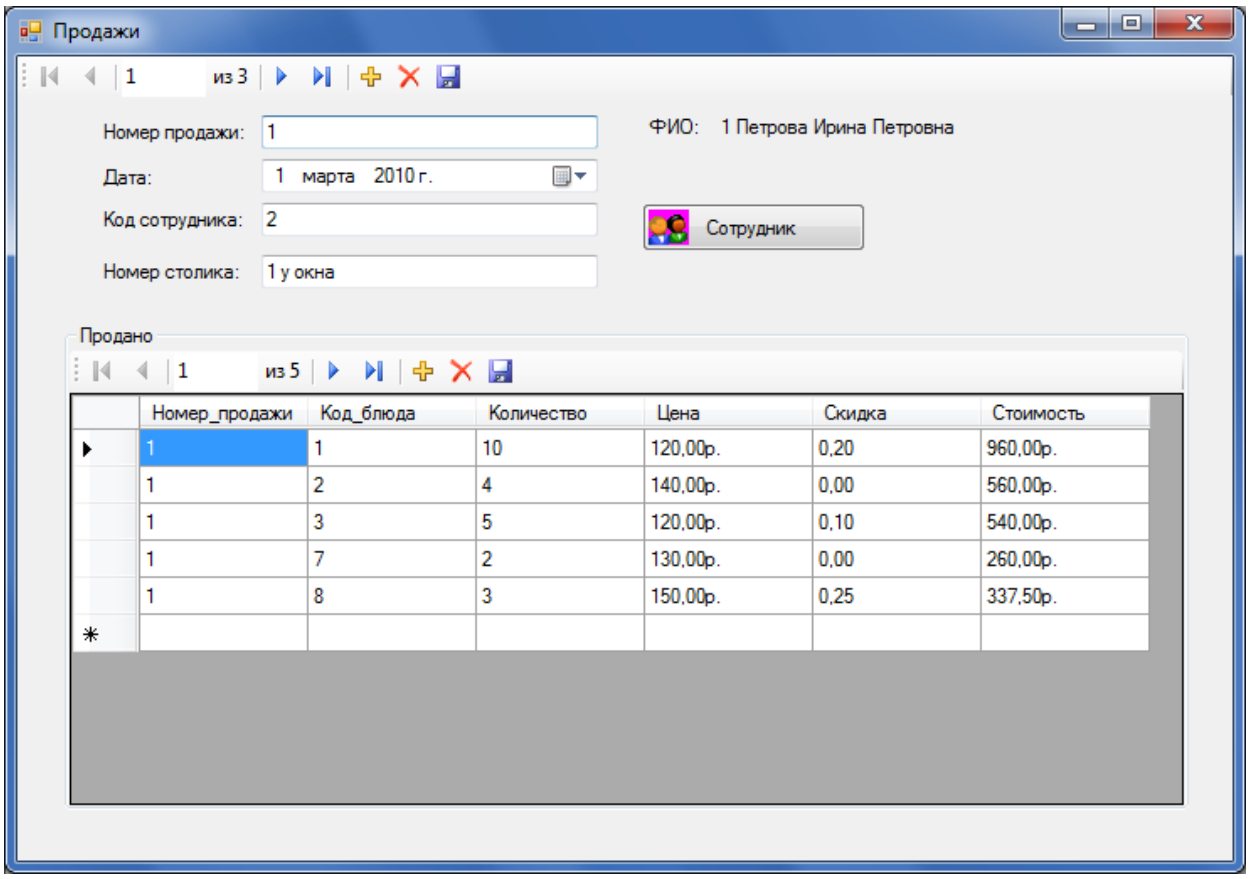


И переместите на форму Продажи (**FormSalesList)**:



Запустите программу.

Как Вы видите, значения данных сотрудника в текущей продаже не соответствует тому значении, что показывает ФИО из таблицы **Сотрудники**.



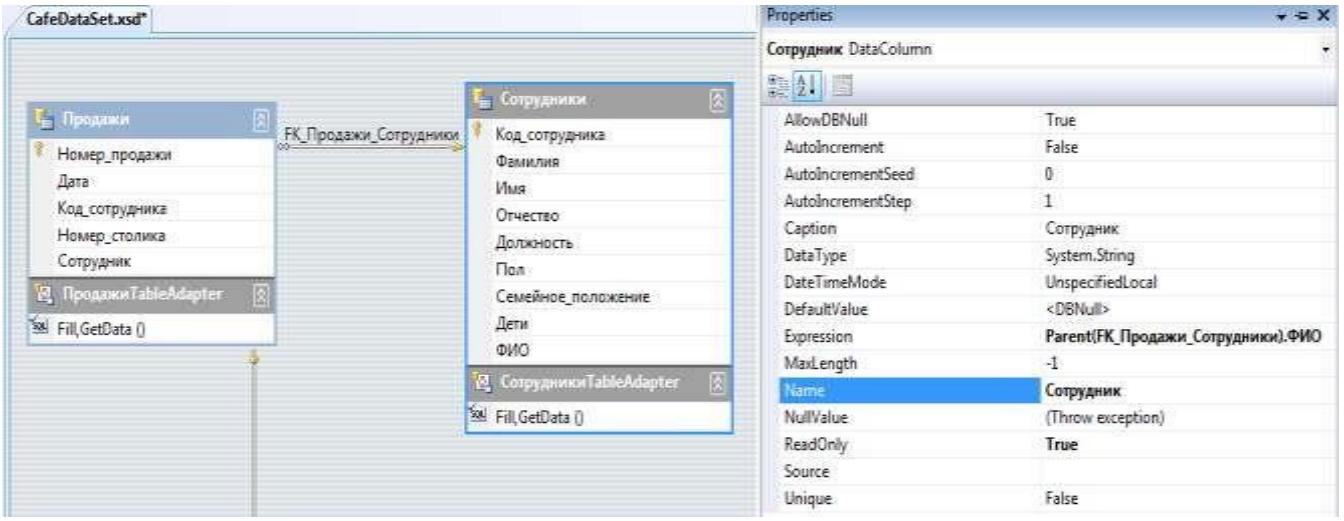
Данную ситуацию можно решить несколькими способами.

Например, проводить поиск для сотрудника в таблице **Продажи** и сотрудника в таблице **Сотрудники** (см. раздел 2.3). Можно создать подстановочные поля (lookup) и др.

Рассмотрим, как создавать подстановочные поля (**lookup**) на примере.

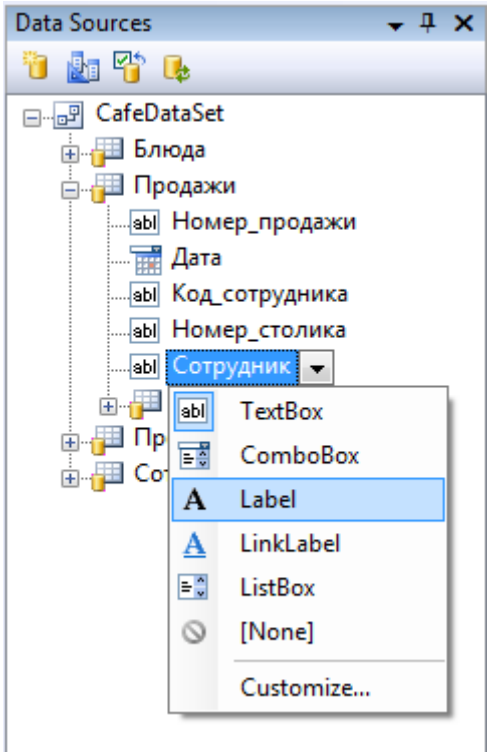
Создадим в таблице **Продажи** новое поле **Сотрудник**, которое будет брать свои значения из таблицы **Сотрудники** из поля ФИО.

Для этого поля **Сотрудник** в свойстве **Expression** укажите:
Parent(FK_Продажи_Сотрудники).ФИО

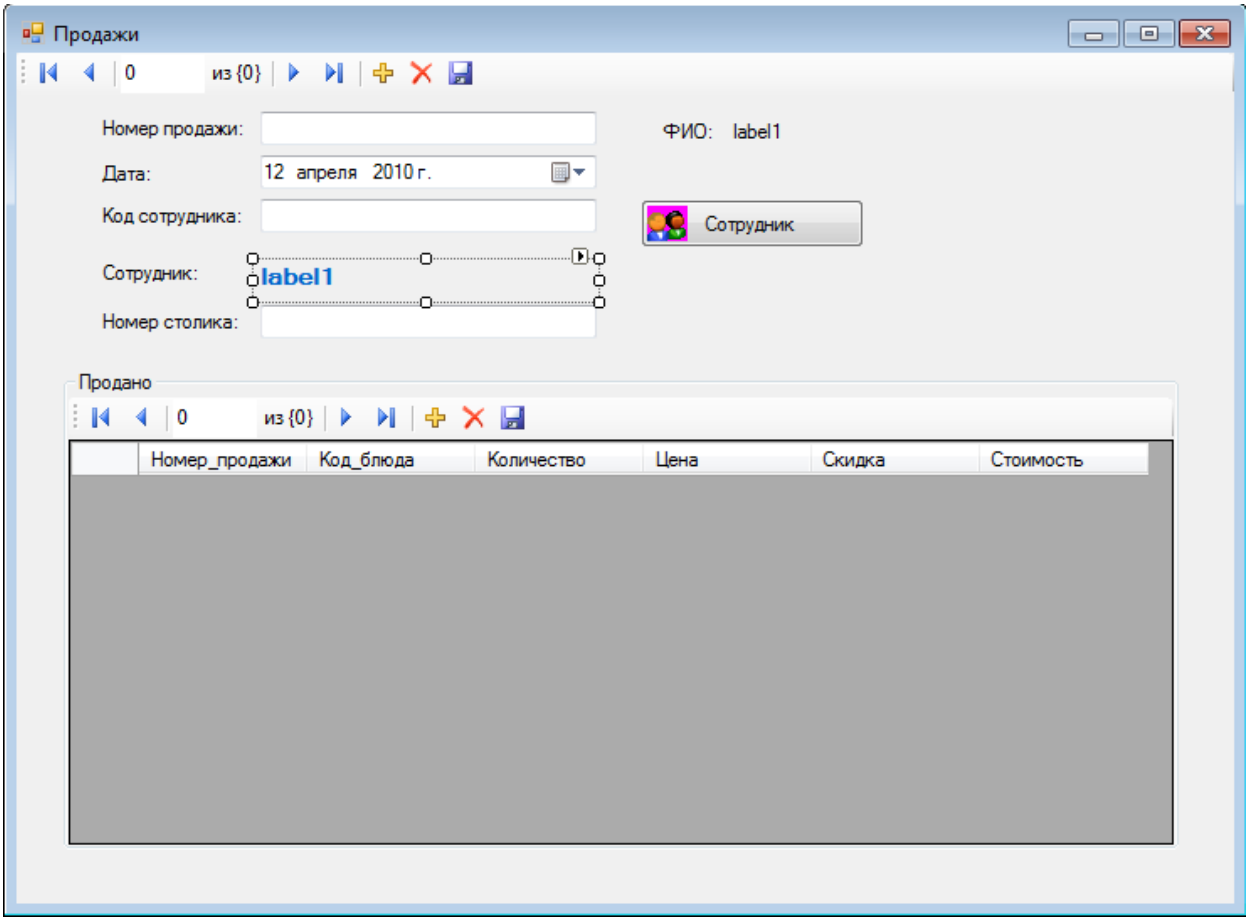


Где **Parent(родитель)** указывает принадлежность поля ФИО к родительской таблице **Сотрудники** через связь **FK_Сотрудники_Продажи**.

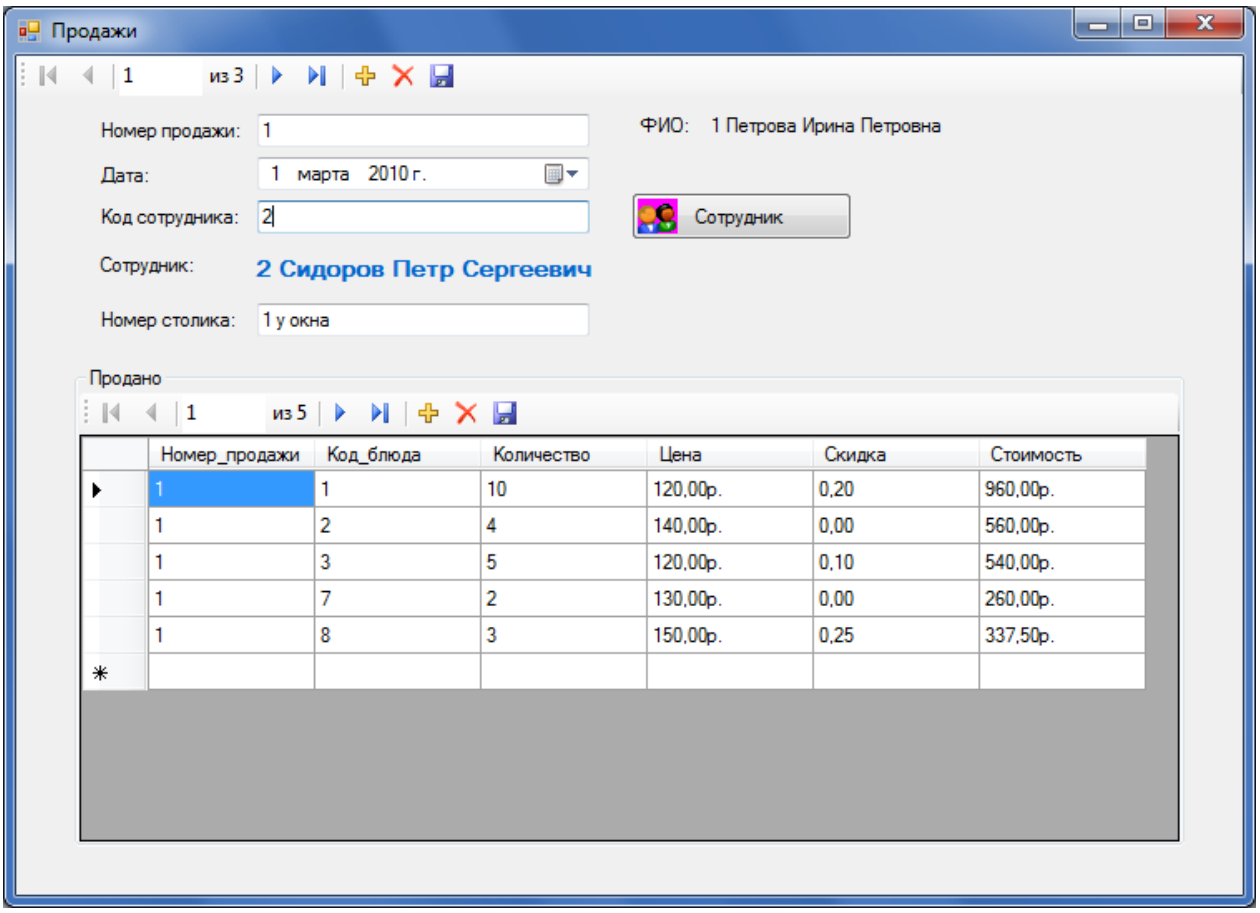
Затем перейдите на форму **FormSalesList** и положите для уточнения под полем Код сотрудника метку с его ФИО:



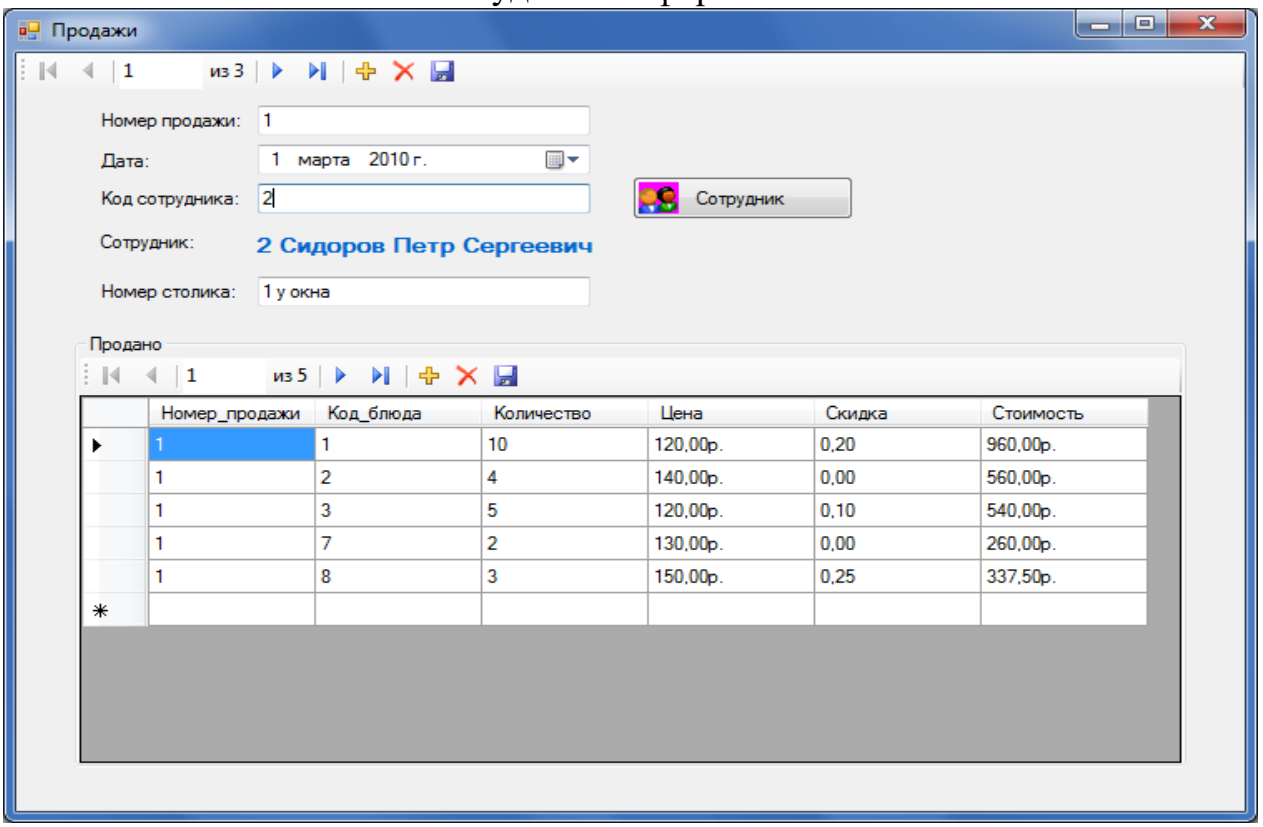
И переложите на форму. Для удобства настройте цвет и стиль шрифта:



Запустите программу.
Посмотрите настроенную возможность:



А ФИО с меткой можно удалить с формы:



Примечание:

Когда пользователь работает в форме Продажи и открывает из нее форму Сотрудники, то он всегда может не только выбрать уже имеющегося сотрудника, но и добавить нового сотрудника, отредактировать данные по существующим сотрудникам, при желании удалить.

Чтобы подстановочное поле могло брать обновленные записи из таблицы сотрудников **Сотрудники,** необходимо дописать в обработчике **buttonWorker_Click** выделенную строку:

```
private void buttonWorker_Click(object sender, EventArgs e)
        {
            int id = -1;
            if
(((DataRowView)продажиBindingSource.Current)["код_сотрудника"].ToString() !=
"")
            {
                id =
(int)(((DataRowView)продажиBindingSource.Current)["код_сотрудника"]);
            }
            id = FormWorkesList.fw.ShowSelectForm(id);
            if (id >= 0)
            {
                MessageBox.Show(id.ToString());
                ((DataRowView)продажиBindingSource.Current)["код_сотрудника"]
= id;
                продажиBindingSource.EndEdit();
                сотрудникиTableAdapter.Fill(this.cafeDataSet.Сотрудники);
            }

        }
```

По аналогии создаются и другие подстановочные поля (lookup).

2.7 Связь M:M между Блюда и Продажи в приложении

Связь M:M между **Блюда** и **Продажи** в реляционных БД реализуется через связи 1:M между **Продажи** и **Продано** и 1:M между **Блюда** и **Продано.**

Реализация 1:M между **Продажи** и **Продано** выполнена в разделе 2.1.

Добавим реализацию 1:M между **Блюда** и **Продано**.

Перейдите на форму **FormSalesList**.

Обратите внимание, что при оформлении блюд в продажах необходимо помнить их код блюда, что не всегда является удобным.

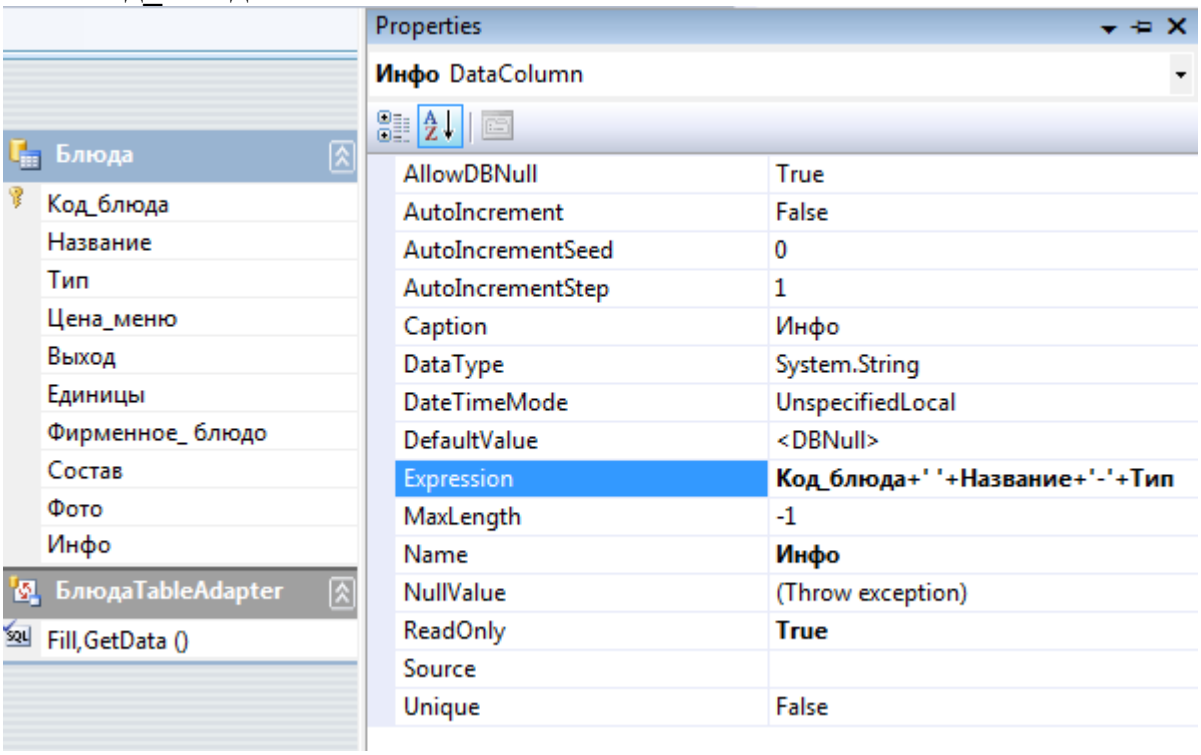
Настроим более удобный способ указания блюд в продажах для пользователя.

Наиболее быстрый способ - это настройка подстановочного поля в **Продано,** которое по заданному коду блюда будет брать значения из таблицы **Блюда**, например, его название и тип.

Сформируем такую колонку.

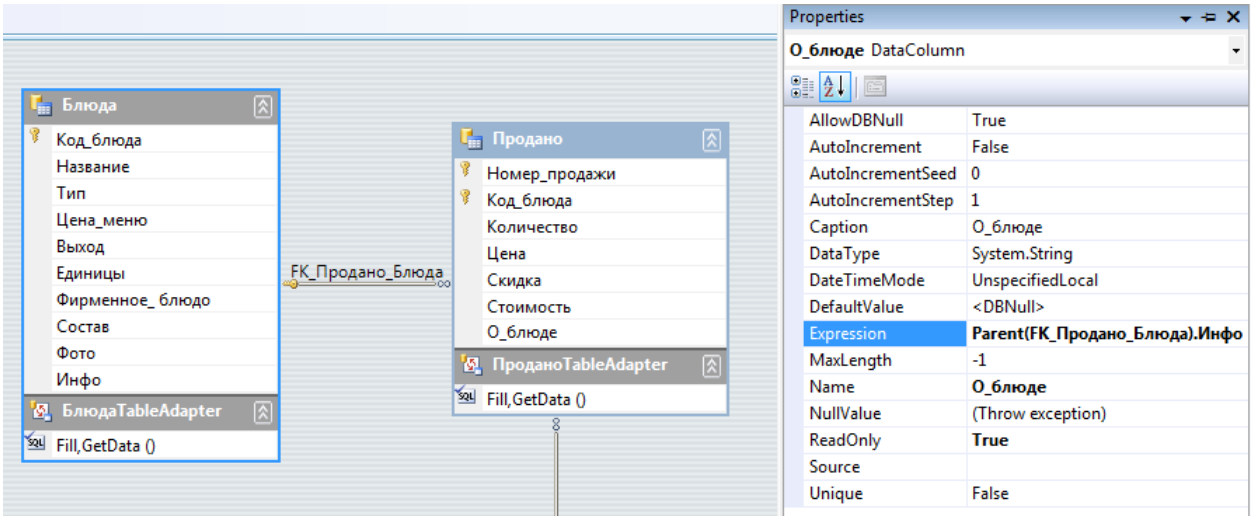
Для этого необходимо создать новую колонку **Инфо** строкового типа в таблице **Блюда**. Для нее укажите в свойстве **Expression**:

Код_блюда+' '+Название+'-'+Тип



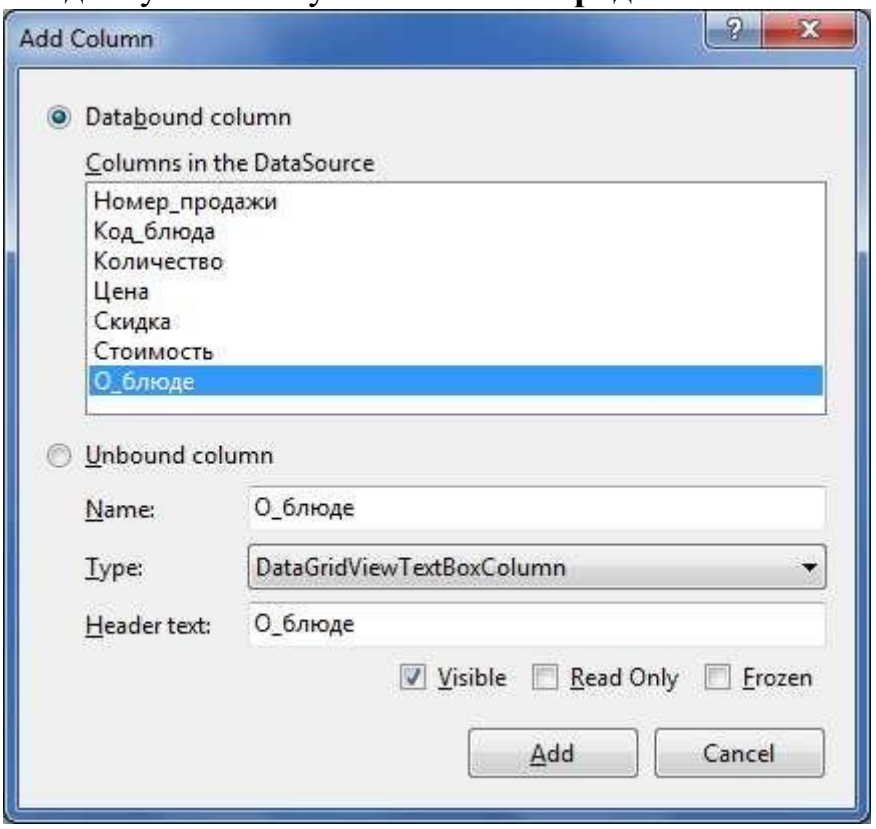
Создайте в **Продано** новую колонку **О_блюде** строкового типа. Для него укажите в свойстве **Expression**:

Parent(FK_Продано_Блюда).Инфо

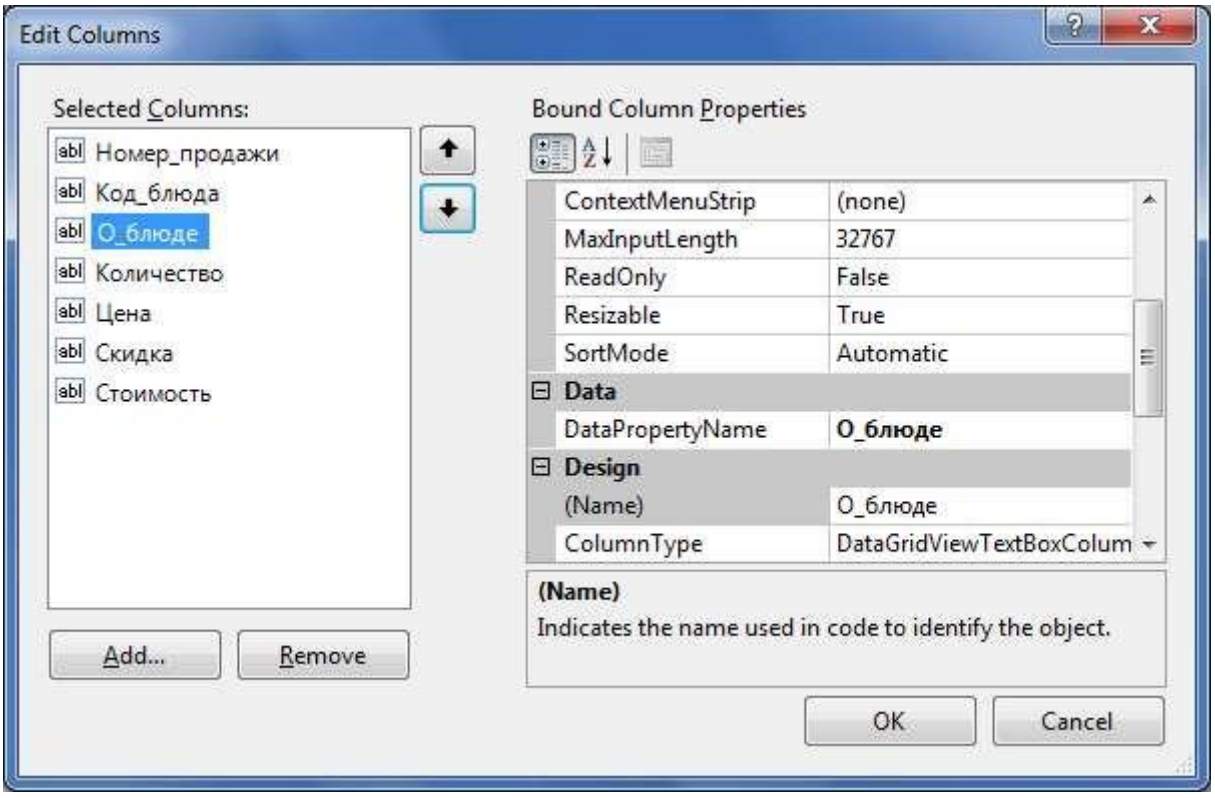


В данном случае для родителя (**Parent**) необходимо указать имя связи **FK_Продано_Блюда.**

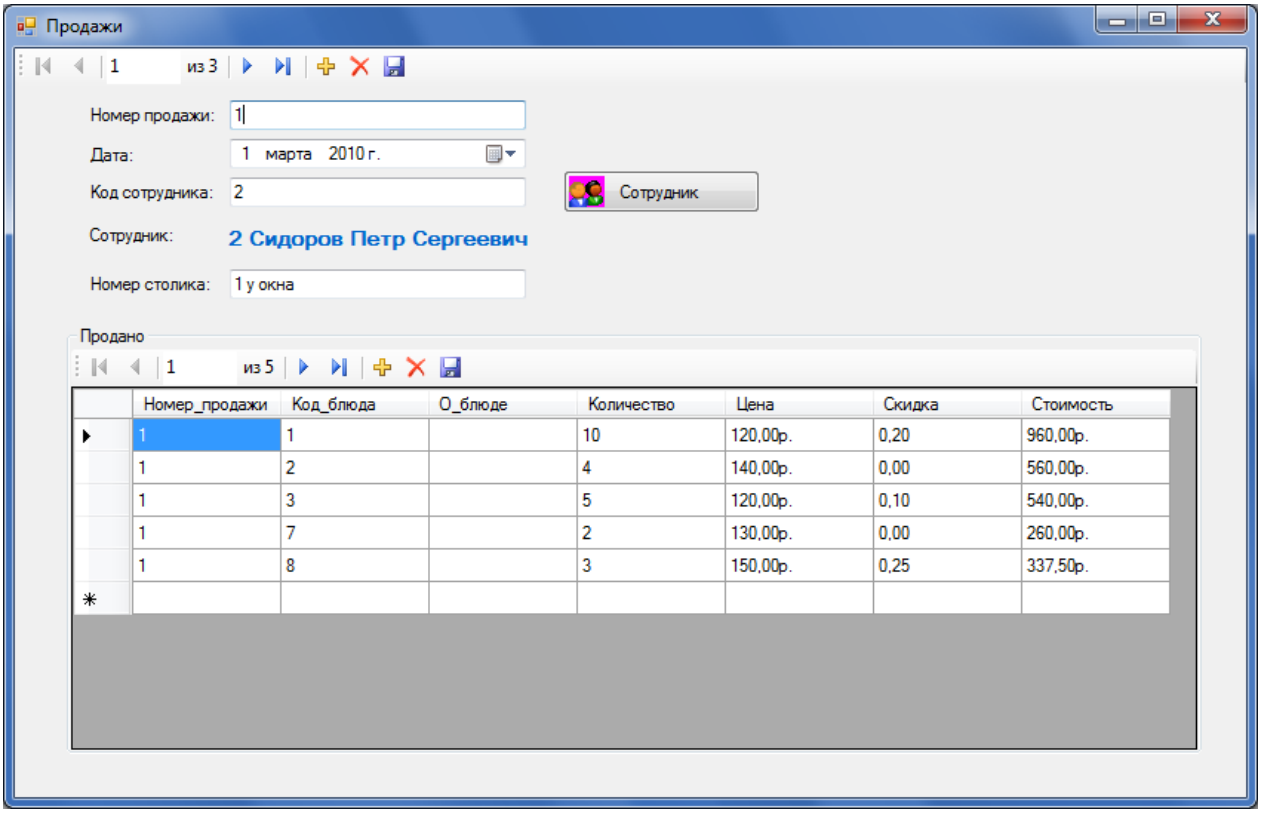
Добавьте данную колонку к колонкам в **проданоDataGridView**:



Переместите в колонках **проданоDataGridView** колонку **O_блюде** рядом с **Код_блюда**, выбирая соответствующие стрелки:

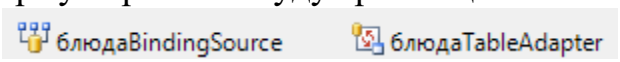


Запустите программу. Посмотрите добавленную возможность:

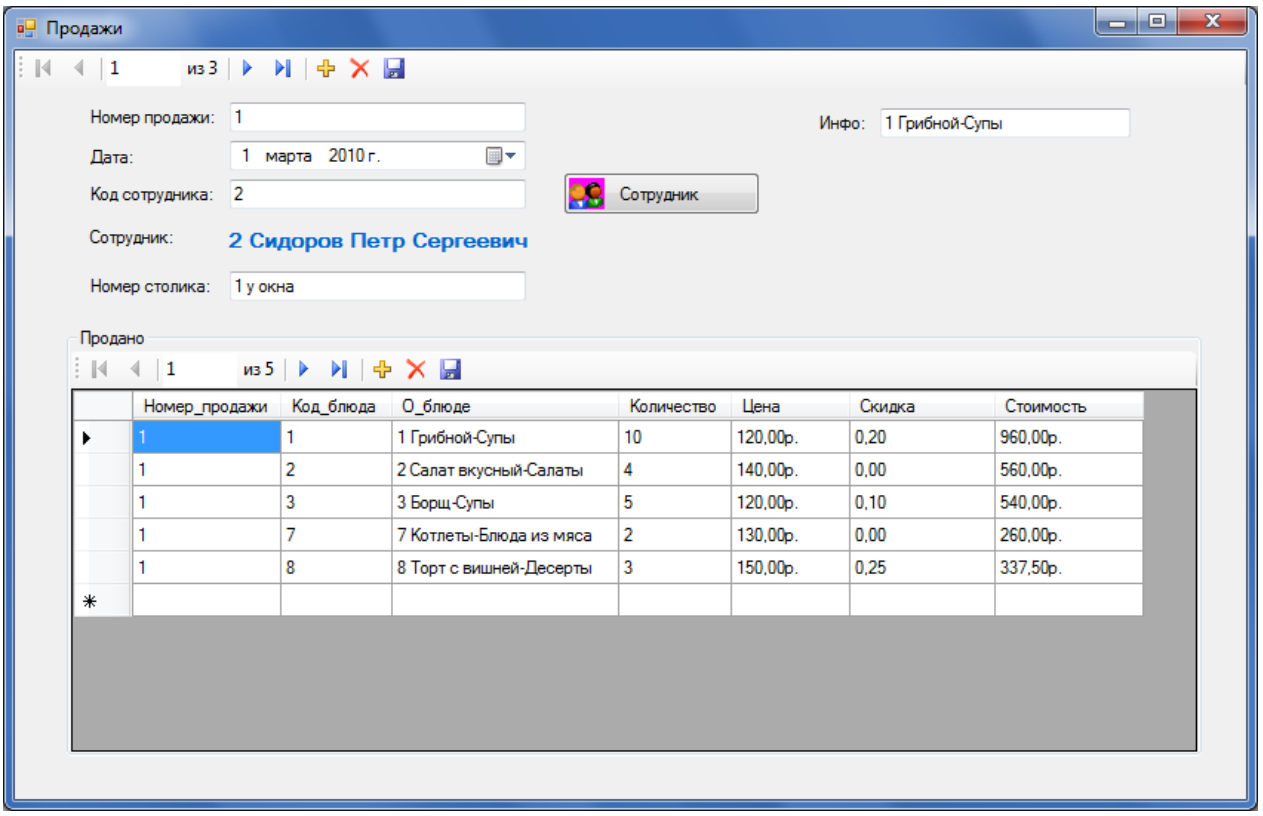


Как видите, значения подстановочного поля **О_блюде** не отображаются, так как набор данных для блюд на данной форме не сформирован.

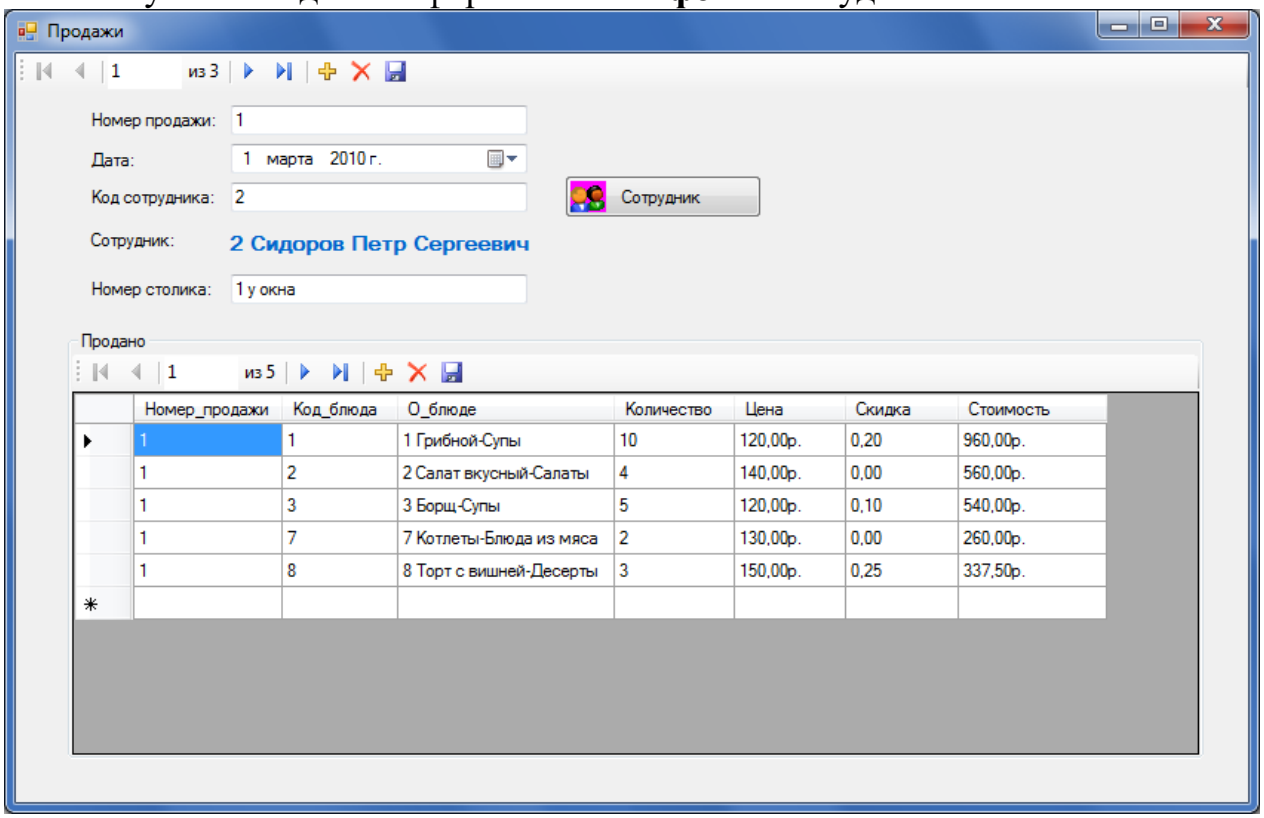
Для его формирования можно, например, добавить поле **Инфо** из **Блюда** на форму. При этом будут размещены:



Теперь запустите приложение.



Ненужное на данной форме поле **Инфо** можно удалить:

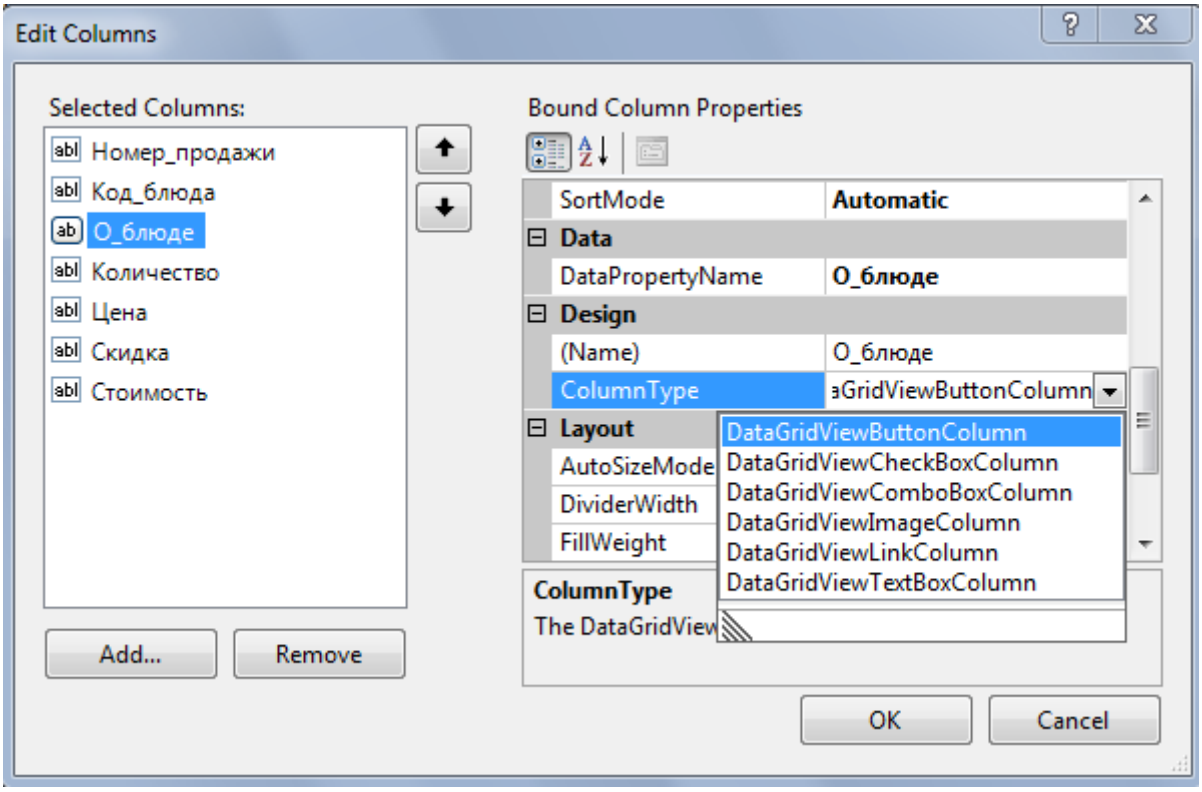


Продолжим оформлять форму с продажами.

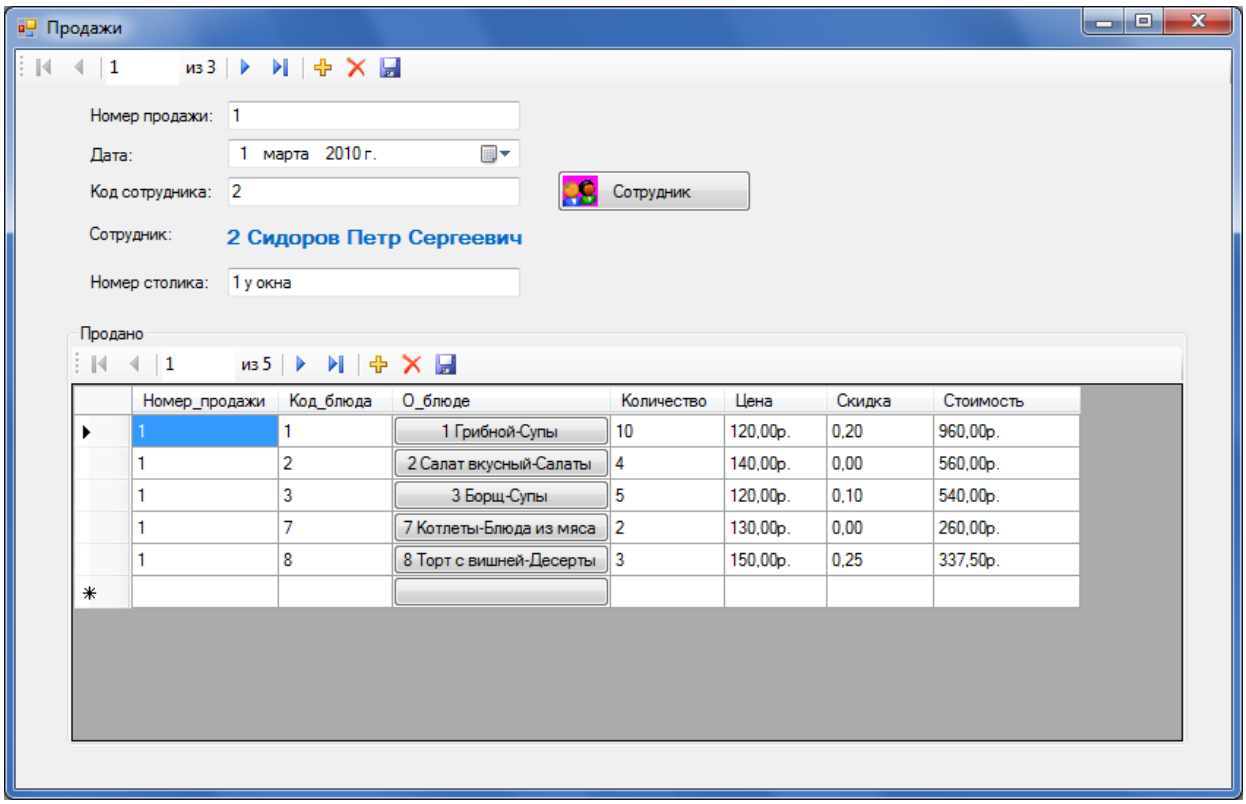
Будем показывать меню блюд по нажатию кнопки в **проданоDataGridView**. Для этого воспользуемся уже добавленной колонкой

О_блюде (или создайте отдельную колонку).

Укажите для данной колонки тип кнопки (**DataGridViewButtonColumn**):



Запустите программу. Посмотрите, какие изменения произошли в **проданоDataGridView**:



Закройте программу. Вернитесь в проект.

Запрограммируем данную кнопку на показ формы с меню блюд по аналогии, как было реализовано для **Сотрудники** и **Продажи** (см. раздел 2.4).

Для удобства заполнения продаж, будем для выбранного блюда указывать его цену из меню блюд при оформлении продажи.

Для этого укажите в коде формы для блюд (**FormDishesList)**:

```
int idCurrent = -1;

public int ShowSelectForm(int idDish, out decimal price)
        {
            idCurrent = idDish;
            ShowDialog();
            price =
(decimal)((DataRowView)блюдаBindingSource.Current)["цена_меню"];
            return
(int)((DataRowView)блюдаBindingSource.Current)["код_блюда"];
        }
```

В коде **FormDishesList** пропишите:

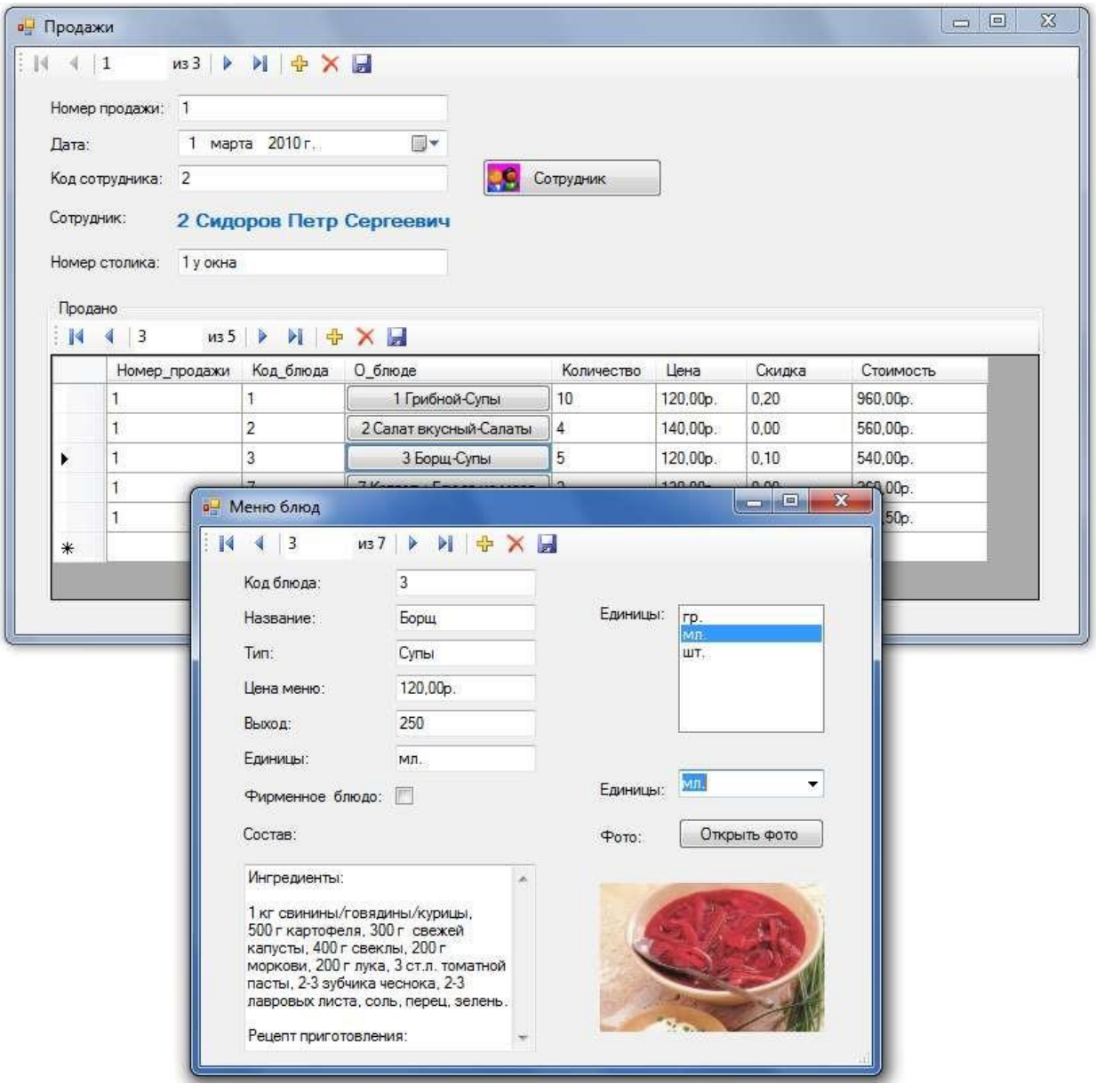
```
public void ShowForm()
        {
            блюдаBindingSource.Position = 0;
            Show();
            Activate();
        }


private void FormDishesList_Shown(object sender, EventArgs e)
        {
            блюдаBindingSource.Position =
блюдаBindingSource.Find("код_блюда", idCurrent);
        }
```

В коде **FormSalesList** пропишите:

```
string GetSelectedFieldName()
        {
            return
проданоDataGridView.Columns[проданоDataGridView.CurrentCell.ColumnIndex].Data
PropertyName;
        }


private void проданоDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
        {
            if (String.Compare(GetSelectedFieldName(), "О_блюде", false) ==
0)
            {
                decimal price = 0;
                int idDishCurrent = -1;
```

```
int.TryParse((((DataRowView)проданоBindingSource.Current)["код_блюда"]).ToStr
ing(), out idDishCurrent);
                int idDish = FormDishesList.fd.ShowSelectForm(idDishCurrent,
out price);
                MessageBox.Show("Код блюда=" + idDish.ToString());
                ((DataRowView)проданоBindingSource.Current)["код_блюда"] =
idDish;
                ((DataRowView)проданоBindingSource.Current)["цена"] = price;
                проданоBindingSource.EndEdit();
                проданоTableAdapter.Update(this.cafeDataSet);
                блюдаTableAdapter.Fill(this.cafeDataSet.Блюда);
            }

        }
```

Запустите программу.
Посмотрите полученную возможность:



Примечание:

Кнопка **Выбрать** в данном примере для формы блюд не была использована. Выбор блюда происходит по закрытию формы.

3 Задание на лабораторную работу №3

3.1 Изучить материал лабораторной работы №3.

3.2 Для своей БД необходимо выполнить:

3.2.1 создать форму/формы проекта для работы с данными по связанным таблицам. Обязательно 1:M на одной форме, M:M через 1:M и M:1 на одной форме;

3.2.2 реализовать шаблон проектирования «Одиночка» (англ. Singleton);

3.2.3 обработать исключения в программе, возникающие при некорректной работе с данными (не менее 2-3 разных ошибок);

3.2.4 реализовать поиск, фильтрацию данных;

3.2.5 создать вычисляемые колонки (не менее 1-2 разных);

3.2.6 создать подстановочные колонки (не менее 1-2 разных);

3.2.7 использовать различные элементы управления Label, TextBox, Button, DataGridView, BindingNavigator и др.;

3.2.8 настроить для удобства BindingNavigator (всплывающие подсказки обязательно);

3.2.9 протестировать работу полученного приложения.

4 Оформление отчета к лабораторной работе

Оформить отчёт со следующим содержанием:

4.1 Титульный лист.
4.2 Цель работы.
4.3 Постановка задачи.
4.4 Краткая теория и ход выполнения заданий.
4.5 Описание результатов.
Приведите скриншоты полученной программы и листинг программы.
4.6 Заключение (выводы).

Отчет должен быть оформлен согласно требованиям ОС ТУСУР.

5 Контрольные вопросы

5.1 Что характерно для ссылочной целостности?
5.2 Что подразумевает каскадное удаление и каскадное обновление?
5.3 Как можно оформить в приложении работу с данными по связанным таблицам БД?
5.4 Какие ошибки могут вызываться компьютерной программой?
5.5 Что такое исключение? При помощи какой конструкции можно обрабатывать исключения, возникающие при некорректной работе с данными?
5.6 Что характерно для конструкции try…catch?
5.7 Как можно реализовать поиск, фильтрацию данных?
5.8 Особенности метода BindingSource.Find.
5.9 Особенности свойства BindingSource.Filter.
5.10 Как можно создать вычисляемые колонки?
5.11 Как можно создать подстановочные колонки? Что характерно для подстановочных колонок?
5.12 Каковы особенности работы со связанными данными в приложении?

ЛАБОРАТОРНАЯ РАБОТА № 4 SQL ОПЕРАТОРЫ ДЛЯ БД СУБД MICROSOFT SQL SERVER В ПРИЛОЖЕНИИ MICROSOFT VISUAL STUDIO

Цель работы:

Изучение основных особенностей формирования запросов в приложении Microsoft Visual Studio для своей предметной области (на примере базы данных для кафе).

1 Краткие сведения об SQL (Structured Query Language)

SQL содержит операторы определения данных, операторы манипулирования данными, операторы, предназначенные для администрирования БД и др.

SQL является международным стандартом, признанным такими организациями по стандартизации, как Международная организация по стандартизации (International Organization for Standardization, **ISO**), Американский национальный институт стандартов (American National Standards Institute, **ANSI**), который входит в **ISO** и Международную электротехнической комиссию (МЭК, International Electrotechnical Commission, **IEC**).

Современные реляционные СУБД содержат в своем составе SQL, соответствующий одному из стандартов. Разработчики СУБД при реализации языка SQL могут вносить в него расширения (дополнительно к базовому набору команд стандарта SQL).

СУБД Microsoft SQL Server в качестве языка запросов использует версию SQL, получившую название Transact-SQL (сокращённо T-SQL). Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями и особенностями.

В лабораторной работе рассматриваются SQL операторы манипулирования данными.

Существует четыре основных типа запросов данных в SQL, которые относятся к так называемому **языку манипулирования данными** (Data Manipulation Language или DML):

- **SELECT** – выбрать строки;
- **INSERT** – добавить строки;
- **UPDATE** – изменить строки;
- **DELETE** – удалить строки.

1.1 Оператор выбора SELECT

Для выборки данных используется оператор SELECT. Его синтаксис приведен ниже.

SELECT
[ALL | DISTINCT] [<псевдоним>.]**<выражение>**[AS<колонка>]
[,[<псевдоним>.] <выражение>[AS<колонка>]...]
FROM <откуда берутся данные>
[WHERE <условие/условия отбора>]
[GROUP BY <колонка>[,<колонка>...]]
[HAVING <условие/условия отбора>]
[ORDER BY <колонка>[,<колонка>...]]

Здесь ALL означает, что в результирующую выборку будут включаться все записи, удовлетворяющие запросу. DISTINCT предотвращает повторный вывод одних и тех же записей.

<выражение> может содержать:

- поле базы данных;
- константу; указанное значение константы появится в каждом ряду результата запроса;
- выражение, которое может включать пользовательскую функцию.

Звёздочка «*» указывает, что в результат запроса надо включить все столбцы той или иной таблицы.

Необязательная фраза AS <колонка> специфицирует заголовок колонки в выводе результата запроса. Это полезно, когда имеются <выражения> или содержатся функции работы с полями и требуется присвоить колонке в результате соответствующее имя.

<колонка> может быть выражением.

Если не устраивают имена колонок данных по умолчанию, можно их обозначить по-своему, используя выражение:

[AS <Новое наименование колонки>]

Псевдоним БД, откуда берется нужное поле, может быть назначен в команде SELECT. Вне команды действия не имеет.

Каждой таблице, используемой в запросе, можно присвоить краткое имя (например, просто букву), а затем обращаться по псевдониму:

SELECT псевдоним1.колонка1, псевдоним2.колонка2, ...
FROM таблица1 псевдоним1 [, <таблица2 псевдоним2>…]

FROM задается всегда. Определяет имена всех баз данных, таблиц, запросов, которые принимают участие в запросе.

Отбор данных в выборке производится посредством ключевого слова **WHERE**:

WHERE <условие/условия отбора>

<условие отбора> специфицирует критерий, которому должны удовлетворять записи, чтобы попасть в результат действия запроса. Условия фильтрации можно соединять операторами AND, OR. Можно также использовать оператор NOT для реверса значения логического выражения. Также могут быть включены такие операторы, как =,= >, <, >=, <=, BETWEEN, LIKE, IN и др.

Примеры операторов:

<table>
<tr><th>Оператор</th><th>Описание</th><th>Пример</th></tr>
<tr><td>=</td><td>Равно</td><td>Цена =1000
записи с ценой равной 1000
Тип = ’Фрукты’
записи с типом фрукты</td></tr>
<tr><td><></td><td>Не равно</td><td>Количество <>500
записи с количеством не равным 500</td></tr>
<tr><td>></td><td>Больше</td><td>Цена >1000
записи с ценой больше 1000</td></tr>
<tr><td><</td><td>Меньше</td><td>Количество <10
записи в количестве менее 10</td></tr>
<tr><td>>=</td><td>Больше или равно</td><td>Цена >=1000
записи с ценой больше или равной 1000</td></tr>
<tr><td><=</td><td>Меньше или равно</td><td>Количество <=10
записи в количестве менее или равным 10</td></tr>
<tr><td>AND</td><td>Логическое И
Таблица истинности:
<table><tr><th>A</th><th>B</th><th>A AND B</th></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table></td><td>Цена>=100 AND Цена <=500
записи с ценой от 100 до 500 включительно</td></tr>
<tr><td>OR</td><td>Логическое ИЛИ
Таблица истинности:
<table><tr><th>A</th><th>B</th><th>A OR B</th></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table></td><td>Тип = ’Овощи’ OR Тип = ’Фрукты’
записи с типом фрукты или овощи</td></tr>
<tr><td>NOT</td><td>Реверс значения логического выражения
Таблица истинности:
<table><tr><th>A</th><th>NOT A</th></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table></td><td>Тип NOT ‘Фрукты’
типы кроме фруктов</td></tr>
</table>

LIKE	Для отбора данных по шаблону.	**Название LIKE '_a%’**

	Синтаксис: **<выражение>LIKE<шаблон>** При этом в шаблоне используются обозначения: _ - один символ, % - любое количество символов, [список] - любой символ из списка.	записи, в названии которых вторая буква а **Фамилия LIKE '[А,Б]%'** фамилии, начинающиеся на букву А или Б **Тип LIKE '[А-Д]%'** записи с типом, начинающимся на буквы от А до Д
IN	Позволяет выбрать значения из заданного списка. Синтаксис: **<выражение>IN(<выражение>,|; <выражение>,|;...)**	**Пол IN ('мужской'; 'женский')** записи с полом мужской или женский
BETWEEN	Проверяет, находится ли (или нет) выражение в заданном диапазоне. Синтаксис: **<выражение>BETWEEN <нижнее значение> AND <верхнее значение>**	**Цена BETWEEN 1000 AND 12000** записи с ценой от 1000 до 12000 включите
И др.		

Операторы условия =, >, <, >=, <=, !=, < > применяются к данным имеющим тип: числовой, символьный, дата.

Использование операции сцепления строк

Операция + соединяет два строковых значения, которые могут быть представлены выражениями:

<строковое выражение1> + <строковое выражение2>

Эту операцию можно использовать как после слова SELECT для указания возвращаемых значений, так и в предложении WHERE.

Например:

SELECT Фамилия + ' '+ Имя +' '+ Отчество AS ФИО

FROM Сотрудники

Агрегирующие (статистические) функции

Агрегирующая функция рассматривает множество строк таблицы и выдает только одно значение. Приведем список агрегирующих функций.

SUM([DISTINCT] выражение) — сумма (различных) значений в выражении.

AVG([DISTINCT] выражение) — средняя величина (различных) значений в выражении.

COUNT([DISTINCT] выражение) — количество (различных) непустых строк в выражении.

COUNT(*) — полное количество выбранных строк, включая пустые строки и дубликаты.

MAX(выражение) — максимальное значение в выражении.

MIN(выражение) — минимальное значение в выражении.

При использовании агрегирующих функций необходимо учитывать следующее:

- их можно записывать в списке выбора, в предложении HAVING, ORDER BY;
- нельзя применять в предложении WHERE без подзапросов;
- если определено предложение DISTINCT, то повторяющиеся значения аргумента в подсчете значения функции не участвуют;
- если аргумент содержит NULL значения, то функция COUNT всегда возвращает значение 0, а другие агрегирующие функции — NULL значение;
- функции SUM и AVG работают только с аргументами числовых типов;
- функции COUNT, MAX, MIN могут использоваться с аргументами различных типов (числовые, текстовые, дата и др.).

Группировка данных

Для группировки данных используется предложение **GROUP BY**, которое разделяет таблицу на наборы.

GROUP BY неразрывно связано с агрегирующими функциями и предложением HAVING.

Например:

```
SELECT AVG (Цена_меню) AS Средняя_цена
FROM Блюда

SELECT Тип, AVG (Цена_меню) AS Средняя_цена
FROM Блюда
GROUP BY Тип
```

В результате первого запроса получим среднюю стоимость блюд в меню. Это будет единственное значение. Во втором запросе блюда группируются по типу. Строки помещаются в одну группу тогда и только тогда, когда у них совпадает поле Тип. Затем фраза SELECT применяется к каждой группе, т.е. внутри каждой группы считается средняя цена.

Как правило, в большинстве реализаций SQL элемент из списка GROUP BY должен присутствовать в списке выбора SELECT.

Можно создавать группы внутри других групп.

Если выполнить предложение GROUP BY без агрегирующих функций, то оно будет напоминать предложение DISTINCT, т.е. разделять таблицу на

группы и из группы брать одно значение. Фактически предложение GROUP BY и агрегирующие функции были созданы друг для друга.

GROUP BY возможно применять вместе с предложением WHERE. В этом случае сначала находятся все строки, удовлетворяющие предложению WHERE, затем оставшиеся строки группируются в соответствии с предложением GROUP BY.

Условие, накладываемое на группировки, задается предложением HAVING. Предложение HAVING похоже на предложение WHERE, но HAVING работает не с отдельными записями таблицы, а с группами. Последовательность действий следующая. Сначала для всего запроса выполняется предложение WHERE, затем GROUP BY и уже к сгруппированным данным применяется условие, записанное в предложении HAVING.

Пусть необходимо группировать данные по типу блюд и при этом исключить из рассмотрения наборы, содержащие только одно блюдо:

```
SELECT Тип, COUNT(Код_блюда) AS Количество_блюд
FROM Блюда
GROUP BY Тип
HAVING COUNT(Код_блюда) > 1
```

Условия в предложении HAVING могут объединяться с помощью операторов AND, OR.

Сортировка результатов запроса

Упорядочивание/сортировка результатов запроса производится с использованием ключевого слова **ORDER BY**.

ORDER BY <колонка> [ASC/DESC][,<колонка>,...] — задает упорядочение по колонке, колонкам.

Фраза DESC вызывает упорядочение по убыванию значения соответствующей <колонки>. Фраза ASC специфицирует упорядочение по возрастанию и принимается по умолчанию.

Если указаны два или более столбцов, первый столбец будет использован для глобальной сортировки, второй столбец – для сортировки внутри группы, определяемой единым значением первого столбца, и т.д.

Количество уровней сортировки может быть любым. Порядок сортировки задается в предложении ORDER BY. Порядок следования столбцов при выводе задается списком выбора. В операторе SELECT список выбора может не совпадать со списком сортировки.

Ключевые слова ASC, DESC определяют направление сортировки: по возрастанию, по убыванию соответственно и действует только на один столбец.

При выводе возможна сортировка выражений.

Например:

```
SELECT Тип, COUNT(Код_блюда) AS Количество_блюд
```

FROM Блюда
GROUP BY Тип
HAVING COUNT(Код_блюда) > 1
ORDER BY Тип, COUNT(Код_блюда) DESC

Использование NULL значений

В том случае, когда информация неполна, неизвестна на настоящий момент времени, приходится иметь дело с так называемыми NULL значениями. При появлении NULL значений не действуют стандартные правила сравнения: одно NULL значение не равно другому NULL значению. NULL значения приводят к появлению трехзначной логики в логических выражениях.

Таблица истинностей для трехзначной логики приведена ниже.

a	b	NOT a	a AND b	a OR b
1	1	0	1	1
1	0	0	0	1
1	–	0	–	1
0	1	1	0	1
0	0	1	0	0
0	–	1	0	–
–	1	–	–	1
–	0	–	0	–
–	–	–	–	–

Здесь NULL значение показано прочерком (**–**).

При вычислениях строки с NULL значением столбца можно извлекать из таблицы с помощью специального условия IS [NOT] NULL.

Например, чтобы найти все блюда с указанным типом можно использовать запрос:

SELECT * FROM Блюда
WHERE Тип IS NOT NULL

Объединение/Соединение таблиц

Когда в запрос включатся несколько таблиц, для получения нужных результатов используются объединения. Объединение помогает вернуть из каждой включенной в запрос таблицы только нужные записи, на основании сведений о том, как эти таблицы связаны с другими таблицами в запросе.

Если фраза FROM определяет более одной таблицы или подзапроса, то все эти таблицы соединяются. По умолчанию объединенная таблица представляет собой **перекрестное соединение** (CROSS JOIN), называемое также **декартовым произведением**.

Например:

SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Продажи, Сотрудники

или

SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Продажи CROSS JOIN Сотрудники

В перекрестном объединении каждая строка из одной таблицы объединяется с каждой строкой из другой таблицы, в результате это приводит к тому, что называется полным объединением или декартовым произведением.

Например:

Пусть в таблице **Продажи** будут записи:

Номер_продажи	Дата	Код_сотрудника	Номер_столика
1	01.03.2010...	2	1 у окна
2	02.03.2010...	2	2 у окна
3	02.03.2010...	1	2 у стены
4	03.03.2010...	2	1 у окна

Пусть в таблице **Сотрудники** будут записи:

Код_сотрудника	Фамилия	Имя	Отчество	Должность	Пол	Семейное_положение	Дети
1	Петрова	Ирина	Петровна	Официант	женский	замужем	True
2	Сидоров	Петр	Сергеевич	Официант	мужской	холост	False
3	Иванов	Сергей	Сергеевич	Официант	мужской	женат	True
4	Петров	Иван	Васильевич	Менеджер	мужской	женат	True
5	Сидорова	Мария	Ивановна	Официант	женский	не замужем	False
6	Иванов	Егор	Николаевич	Официант	*NULL*	*NULL*	*NULL*

В результате запроса:

SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Продажи, Сотрудники

Получится:

Номер_продажи	Код_сотрудника	ФИО
1	1	Петрова Ирина Петровна
1	2	Сидоров Петр Сергеевич
1	3	Иванов Сергей Сергеевич
1	4	Петров Иван Васильевич
1	5	Сидорова Мария Ивановна
1	6	Иванов Егор Николаевич
2	1	Петрова Ирина Петровна
2	2	Сидоров Петр Сергеевич
2	3	Иванов Сергей Сергеевич
2	4	Петров Иван Васильевич
2	5	Сидорова Мария Ивановна
2	6	Иванов Егор Николаевич
3	1	Петрова Ирина Петровна
3	2	Сидоров Петр Сергеевич
3	3	Иванов Сергей Сергеевич
3	4	Петров Иван Васильевич
3	5	Сидорова Мария Ивановна
3	6	Иванов Егор Николаевич
4	1	Петрова Ирина Петровна
4	2	Сидоров Петр Сергеевич
4	3	Иванов Сергей Сергеевич
4	4	Петров Иван Васильевич
4	5	Сидорова Мария Ивановна
4	6	Иванов Егор Николаевич

Как видно из таблицы выборки, в ней находятся недостоверные данные. Например, продажу с номером 1 оформил 2 сотрудник, а в таблице выборки продажу номер 1 оформил каждый из сотрудников.

Чтобы этого избежать, необходимо провести объединение таблиц. В одном операторе SELECT может объединяться несколько таблиц. Объединение, как правило, проводится по ключевым столбцам,

Например:

```
SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Продажи, Сотрудники
WHERE Продажи.Код_сотрудника = Сотрудники.Код_сотрудника
```

В результате данного запроса получится выборка с достоверными данными:

Номер_продажи	Код_сотрудника	ФИО
1	2	Сидоров Петр Сергеевич
2	2	Сидоров Петр Сергеевич
3	1	Петрова Ирина Петровна
4	2	Сидоров Петр Сергеевич

Рассмотрим этапы обработки данного запроса. Сначала обрабатывается фраза FROM. Однако в этом случае, поскольку в команде заданы две таблицы, система создает декартово произведение строк этих таблиц. Если в предложении FROM указано более двух таблиц, то создается декартово произведение всех таблиц, перечисленных в команде. После создания большой реляционной таблицы система выполняет предложение WHERE. Каждая строка таблицы, созданной предложением FROM, проверяется на выполнение условия WHERE.

При проектировании структуры базы данных в таблицах нужно предусматривать столбцы, по которым в дальнейшем будет производиться объединение таблиц. Чаще всего для этой цели служат первичные и соответствующие им внешние ключи.

Внутреннее объединение (INNER JOIN) - объединяет записи из таблиц, если связующие поля этих таблиц содержат одинаковые значения.

Если же условие не выполняется, то тогда в результирующих набор данных ничего не добавляется.

Синтаксис:

FROM таблица1 INNER JOIN таблица2
ON таблица1.поле1оператор_сравнения таблица2.поле2

где:

- таблица1, таблица2 - имена таблиц, записи которых объединяются;
- поле1, поле2 - имена связываемых полей. Поля должны относиться к одному типу данных и содержать данные одного вида. Имена этих полей могут быть разными;
- оператор_сравнения - любой оператор сравнения: =, <, >, <=, >=,<>.

Например:

```
SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Сотрудники INNER JOIN Продажи ON Сотрудники.Код_сотрудника = Продажи.Код_сотрудника
```

В результате:

Номер_продажи	Код_сотрудника	ФИО
1	2	Сидоров Петр Сергеевич
2	2	Сидоров Петр Сергеевич
3	1	Петрова Ирина Петровна
4	2	Сидоров Петр Сергеевич

В INNER JOIN можно без WHERE:

SELECT Сотрудники.Код_сотрудника, Фамилия, Имя, Отчество, Должность, Номер_продажи
FROM Сотрудники INNER JOIN Продажи ON (Сотрудники.Код_сотрудника=Продажи.Код_сотрудника AND Фамилия='Сидоров' AND Должность='Официант')

Результат:

Код_сотрудника	Фамилия	Имя	Отчество	Должность	Номер_продажи
2	Сидоров	Петр	Сергеевич	Официант	1
2	Сидоров	Петр	Сергеевич	Официант	2
2	Сидоров	Петр	Сергеевич	Официант	4

Внешние объединения включают записи в результирующую таблицу выборку, даже если условие соединения не выполняется. В этом случае столбцы одной из таблиц возвращаются как обычно, а столбцы из другой таблицы содержат значения NULL. Это удобно для поиска пропущенных значений и нарушенных связей между таблицами.

Внешние объединения являются направленными, они могут быть левыми или правыми.

При **левом объединении (LEFT OUTER JOIN)** – в запрос включаются все строки из первой таблицы (левой) в предложении FROM инструкции SQL и те строки из другой таблицы (правой), в которых значения связанных полей совпадают, если таковые имеются, или NULL – если их нет.

При **правом объединении (RIGHT OUTER JOIN)** – в запрос включаются все строки из второй таблицы (правой) в предложении FROM инструкции SQL и те строки из другой таблицы, в которых значения связанных полей совпадают, если таковые имеются, или NULL – если их нет.

OUTER можно опускать, т.е.: **LEFT JOIN, RIGHT JOIN.**

Синтаксис:

```
FROM таблица1 [LEFT | RIGHT] JOIN таблица2
ON таблица1.поле1оператор_сравнения таблица2.поле2
```

Например:

SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Сотрудники LEFT JOIN Продажи ON Сотрудники.Код_сотрудника = Продажи.Код_сотрудника

Результат:

Номер_продажи	Код_сотрудника	ФИО
3	1	Петрова Ирина Петровна
1	2	Сидоров Петр Сергеевич
2	2	Сидоров Петр Сергеевич
4	2	Сидоров Петр Сергеевич
NULL	3	Иванов Сергей Сергеевич
NULL	4	Петров Иван Васильевич
NULL	5	Сидорова Мария Ивановна
NULL	6	Иванов Егор Николаевич

Что соответствует данным БД, так как сотрудники с кодами 3,4,5,6 продажи не оформляли.

```
SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Продажи RIGHT JOIN Сотрудники ON Продажи.Код_сотрудника = Сотрудники.Код_сотрудника
```

Результат:

Номер_продажи	Код_сотрудника	ФИО
3	1	Петрова Ирина Петровна
1	2	Сидоров Петр Сергеевич
2	2	Сидоров Петр Сергеевич
4	2	Сидоров Петр Сергеевич
NULL	3	Иванов Сергей Сергеевич
NULL	4	Петров Иван Васильевич
NULL	5	Сидорова Мария Ивановна
NULL	6	Иванов Егор Николаевич

UNION – объединение результатов выполнения нескольких операторов SELECT

Оператор UNION используется для одновременного просмотра всех данных, возвращаемых несколькими сходными запросами на выборку, в виде объединенного набора.

Оператор UNION позволяет объединить несколько инструкций SELECT в одну. Объединяемые инструкции SELECT должны иметь одинаковое число и порядок выходных полей с такими же или совместимыми типами данных. При выполнении запроса данные из каждого набора соответствующих полей объединяются в одно выходное поле, поэтому выходные данные запроса имеют столько же полей, сколько и каждая инструкция SELECT по отдельности.

Например:

```
SELECT Код_сотрудника, Фамилия, Имя, Отчество, Должность FROM Сотрудники
WHERE Должность ='Менеджер'
```

UNION
SELECT Код_сотрудника, Фамилия, Имя, Отчество, Должность FROM Сотрудники
WHERE Фамилия ='Сидоров'

Результат:

Код_сотрудника	Фамилия	Имя	Отчество	Должность
2	Сидоров	Петр	Сергеевич	Официант
4	Петров	Иван	Васильевич	Менеджер

1.2 Операторы изменения данных

1.2.1 Добавление записей/строк

Для добавления записей/строк используется оператор **INSERT**, который позволяет добавлять строки с помощью ключевого слова VALUES или с помощью оператора SELECT.

INSERT INTO имя_таблицы [(столбец1 [,столбец2] . . .)]
VALUES (константа1 [, константа2] . . .)

Применяя данный оператор, необходимо помнить, что вставляемые значения должны иметь тип соответствующий типу столбца, в который значение вставляется или же система должна уметь автоматически производить преобразование. Последовательности столбцов в таблицах должны быть согласованными. Если они не согласованы, операция INSERT либо не выполниться, либо выполниться не полностью. И при этом данные могут быть размещены в неверных столбцах.

При использовании оператора SELECT в операторе INSERT синтаксис несколько изменяется.

INSERT INTO имя_таблицы [(вставляемый список столбцов)]
SELECT список столбцов
FROM список таблиц
WHERE условия

Оператор SELECT позволяет взять данные из нескольких таблиц и вставить их в другую таблицу.

1.2.2 Изменение записей/строк

Изменение существующих данных производится с помощью оператора **UPDATE:**

```
UPDATE имя_таблицы
SET имя_столбца = выражение [,...]
[WHERE условие]
```

Если добавляемое значение имеет неверный тип или нарушаются другие ограничения на целостность, система обычно запрещает выполнение команды и выдает сообщение об ошибке. В предложении SET определяются столбцы и задаются их новые значения.

Предложение WHERE в операторе UPDATE определяет строки, которые будут изменяться. Предложение WHERE в операторе UPDATE аналогично предложению WHERE в операторе SELECT и так же может включать в себя подзапросы к одной или нескольким таблицам.

1.2.3 Удаление записей/строк

Удаление данных осуществляется с помощью оператора **DELETE:**

```
DELETE FROM имя_таблицы
WHERE условие
```

В предложении WHERE определяются строки подлежащие удалению. При отсутствии предложения WHERE из таблицы удалятся все строки, если не будет нарушено ограничение ссылочной целостности.

1.3 Подзапросы

Операторы SQL можно вкладывать друг в друга.

Подзапрос (подчиненный запрос) – это оператор SELECT, помещенный в другой оператор SELECT (в список выбора, в предложение WHERE и др.), INSERT, DELETE, UPDATE или в другой подзапрос.

Подзапрос заключен в круглые скобки. Подзапросы имеют две формы:

- **коррелированный подзапрос** (соотнесенный подзапрос) - внутренний запрос для своего выполнения должен получить данные из внешнего запроса. В коррелированном подзапросе внутренний запрос не может быть реализован немедленно: он ссылается на внешний запрос и выполняется поочередно для каждой строки во внешнем запросе;
- **некоррелированной подзапрос** - внутренний запрос выполняется независимо, передавая результаты во внешний запрос.

Например: Найти названия блюд в продаже номер 1:

Некоррелированный подзапрос:

```
SELECT Название
```

```
FROM Блюда
WHERE Код_блюда IN
        (SELECT код_блюда
        FROM Продано
        WHERE Номер_продажи =1)
```

Коррелированный подзапрос:

```
SELECT Название
FROM Блюда
WHERE 1 IN
        (SELECT Номер_продажи
        FROM Продано
        WHERE Код_блюда =Блюда.Код_блюда)
```

Составить перечень блюд с ценами выше средней:

```
SELECT * FROM Блюда
WHERE Цена_меню > (SELECT AVG(Цена_меню) FROM Блюда)
```

2 Особенности формирования запросов в Microsoft Visual Studio

Как уже было сказано ранее (см. лабораторную работу №2), при формировании наборов данных в Microsoft Visual Studio, а также для работы с данными выполняются соответствующие SQL операторы.

Данные SQL операторы необходимо определить в свойствах:

SelectCommand — выбрать данные;

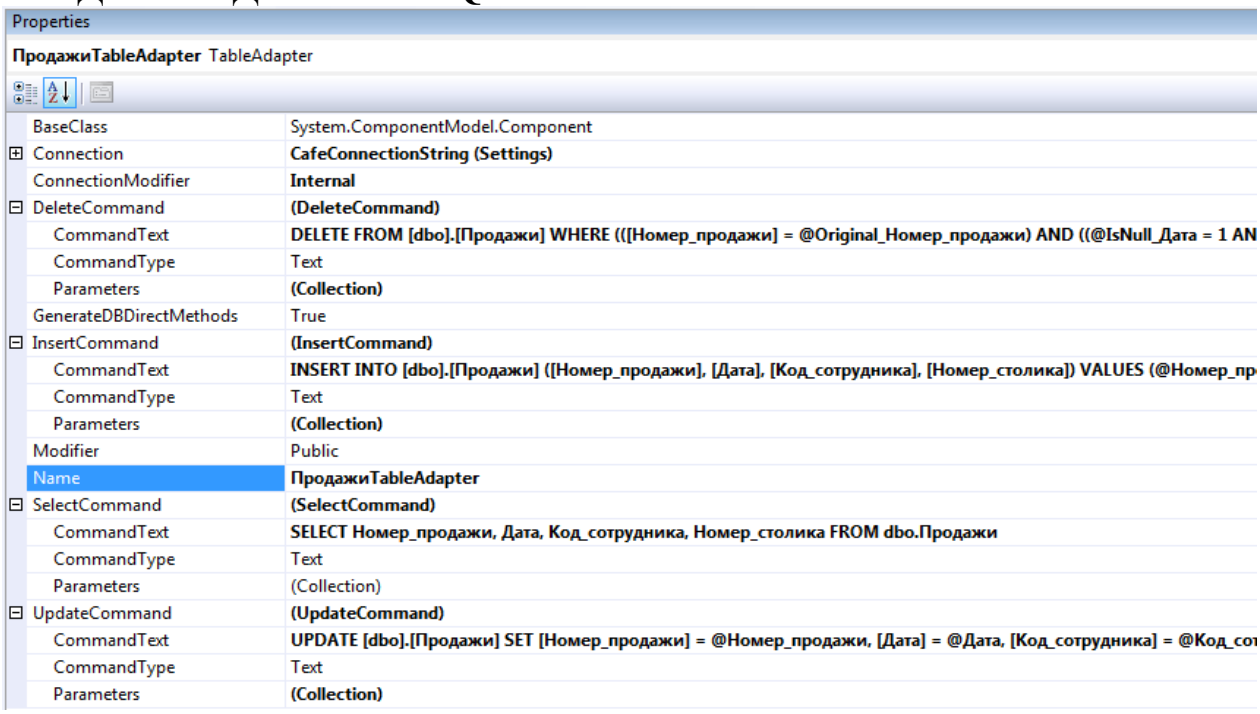
InsertCommand – добавить данные;

DeleteCommand — удалить данные;

UpdateCommand — изменить данные.

Например, для таблицы Продажи:

Для СУБД Microsoft SQL Server:



При задании SQL запросов необходимо определить в соответствующих свойствах:

- **CommandText** – сам текст SQL оператора;
- **CommandType** – тип;
- **Parameters** – параметры.

Таким образом, в разрабатываемом приложении нами уже используются SQL запросы.

Рассмотрим, как можно еще использовать запросы в приложении.

Статический запрос характерен тем, что описывающий его SQL-оператор не изменяется в процессе выполнения приложения.

Динамическим (параметрическим) является запрос, в SQL-операторе которого в процессе выполнения приложения могут изменяться отдельные его составляющие. В этом случае изменяемая часть оператора оформляется как **Параметры**.

Например, пусть в процессе выполнения приложения может быть выдан запрос: выдать все записи из таблицы Блюда для определенного типа блюд.

Например, для десертов:
SELECT * FROM Блюда
WHERE Тип = 'Десерты'

Для салатов:
SELECT * FROM Блюда
WHERE Тип = 'Салаты'

Также аналогичный запрос по другим типам блюд, которые могут задаваться пользователем. Чтобы каждый раз не писать множество статических запросов, необходимо реализовать параметрический запрос:

SELECT * FROM Блюда
WHERE Тип = @Type

Под параметром понимается имя, предваренное @
В дальнейшем можно задать нужные значения для параметров через свойство **Parameters** и его методы:
- например **для СУБД Microsoft SQL Server**:

```
SqlConnection connection = new
SqlConnection(Properties.Settings.Default.CafeConnectionString);
SqlCommand command = connection.CreateCommand();
command.CommandText = "SELECT * FROM Блюда WHERE Тип=@Type”;
command.Parameters.AddWithValue("@Type", textBoxType.Text);
```

или

```
command.Parameters.Add("@Type", SqlDbType.NVarChar).Value =
textBoxType.Text;
```

В динамических запросах параметры всегда заменяют значения, которые могут изменяться в процессе выполнения. Имена параметров произвольны и могут не совпадать со значениями полей таблицы, которым они обычно ставятся в соответствие.

Формируемые запросы - такие запросы, текст SQL-оператора которых формируется программно в процессе выполнения приложения. Действия по формированию такого запроса состоят в программном занесении в соответствующее свойство текста SQL-запроса, вид которого зависит от текущей ситуации и определяется рядом условий, а также в последующем его выполнении.

3 Практические задания

Работа продолжается в проекте, созданном при выполнении лабораторных работ №2, №3.

3.1 Создании формы для запросов

Создайте для запросов отдельную форму.

Для этого нажмите на проекте правую кнопку мыши и выберете пункт **Add\Windows Forms**.

Задайте для новой формы имя **FormSQL.**

Укажите для формы **FormSQL** свойство **Text** в **Запросы**.

3.2 Создание примеров запросов на выборку

С палитры компонент **Toolbox** разместите на форму **FormSQL** компонент **tabControl** – многостраничный контейнер. Укажите для него:

- свойство **Name** укажите в **tabControlSQL**;
- свойство **Dock** (расположение) в **Fill** (вся область **FormSQL**).

Для первой страницы **tabPage1** компонента **tabControlSQL** укажите:

- свойство **Name** укажите в **tabPagePrimer**;
- свойство **Text** в **Примеры запросов**.

Покажем на данной странице **tabPagePrimer**, пример использования запросов:

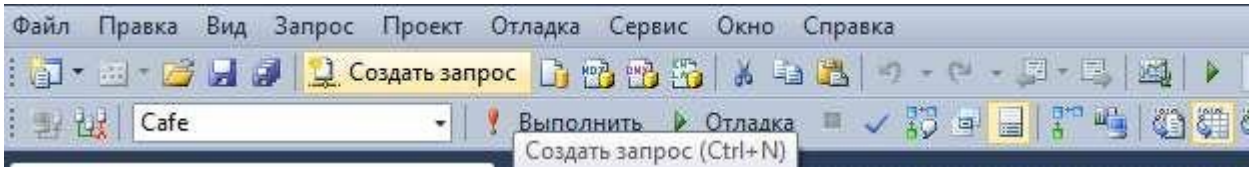
1) **для выборки из одной таблицы всех записей и полей** на примере таблицы **Сотрудники**:

```
SELECT * FROM Сотрудники
```

Примечание:

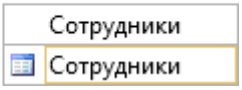
Перед реализацией запросов через интерфейс разрабатываемого приложения для кафе можно протестировать разработанные запросы по необходимым данным, например, в Microsoft SQL Server Management Studio.

Для этого, например, нажмите на панели инструментов **Создать запрос**:

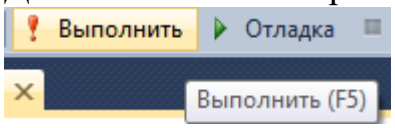


В появившемся окне напишите необходимый запрос, например:

```
SELECT * FROM Сотрудники
```



Для выполнения запроса нажмите, например, **Выполнить**:



И просмотрите результат запроса:

Код_сотрудника	Фамилия	Имя	Отчество	Должность	Пол	Семейное_положение	Дети
1	Петрова	Ирина	Петровна	Официант	женский	замужем	True
2	Сидоров	Петр	Сергеевич	Официант	мужской	холост	False
3	Иванов	Сергей	Сергеевич	Официант	мужской	женат	True
4	Петров	Иван	Васильевич	Менеджер	мужской	женат	True
5	Сидорова	Мария	Ивановна	Официант	женский	не замужем	False
6	Иванов	Егор	Николаевич	Официант	*NULL*	*NULL*	*NULL*

---

2) **для выборки определенных полей из определенной таблицы** на примере таблицы **Блюда**:

Для СУБД Microsoft SQL Server:

```
SELECT Код_блюда, Название+'-'+Тип AS Подробнее, Цена_меню AS [Текущая цена], 'Произойдет' AS Пояснение, Цена_меню*1.5 AS [Повышение цены на 50%]
FROM Блюда
```

Результат:

Код_блюда	Подробнее	Текущая цена	Пояснение	Повышение цены на 50%
1	Грибной-Супы	120,00	Произойдет	180.00000
2	Салат вкусный-Салаты	140,00	Произойдет	210.00000
3	Борщ-Супы	120,00	Произойдет	180.00000
4	Греческий-Салаты	150,00	Произойдет	225.00000
7	Котлеты-Блюда из мяса	130,00	Произойдет	195.00000
8	Торт с вишней-Десерты	150,00	Произойдет	225.00000
9	Коктель "Весна"-Напитки	200,00	Произойдет	300.00000

Чтобы цена после повышения была представлена с заданной точностью, можно использовать специальные функции приведения типов.

Например, **использование функции CAST.**

Иногда возникает потребность трактовать значение одного типа, как значение другого типа. Например, использовать числовое значение как символьную строку или наоборот. В этом случае применяют функцию

CAST (<значение> AS <тип данных>)

Функция CAST делает копию значения, преобразуя его к указанному типу данных. При этом не следует забывать о множестве типов данных, в которое может быть преобразовано значение.

Например:

SELECT Код_блюда, Название+'-'+Тип AS Подробнее, Цена_меню AS [Текущая цена], 'Произойдет' AS Пояснение, CAST(Цена_меню*1.5 AS decimal(16,2)) AS [Повышение цены на 50%]
FROM Блюда

Примечание:

В Microsoft SQL Server Management Studio:

```
SELECT Код_блюда, Название+'-'+Тип AS Подробнее,
        Цена_меню AS [Текущая цена], 'Произойдет' AS Пояснение,
        CAST(Цена_меню*1.5 AS decimal(16,2)) AS [Повышение цены на 50%]
FROM Блюда
```

Результат:

Код_блюда	Подробнее	Текущая цена	Пояснение	Повышение цены на 50%
1	Грибной-Супы	120,00	Произойдет	180.00
2	Салат вкусный-Салаты	140,00	Произойдет	210.00
3	Борщ-Супы	120,00	Произойдет	180.00
4	Греческий-Салаты	150,00	Произойдет	225.00
7	Котлеты-Блюда из мяса	130,00	Произойдет	195.00
8	Торт с вишней-Десерты	150,00	Произойдет	225.00
9	Коктель "Весна"-Напитки	200,00	Произойдет	300.00

---

3) **для выборки данных по продажам из двух таблиц БД Сотрудники** и **Продажи**. Данные таблицы связаны по Код_сотрудника:

SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Продажи, Сотрудники
WHERE Продажи.Код_сотрудника = Сотрудники.Код_сотрудника

или

SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Сотрудники INNER JOIN Продажи ON Сотрудники.Код_сотрудника = Продажи.Код_сотрудника

Примечание:

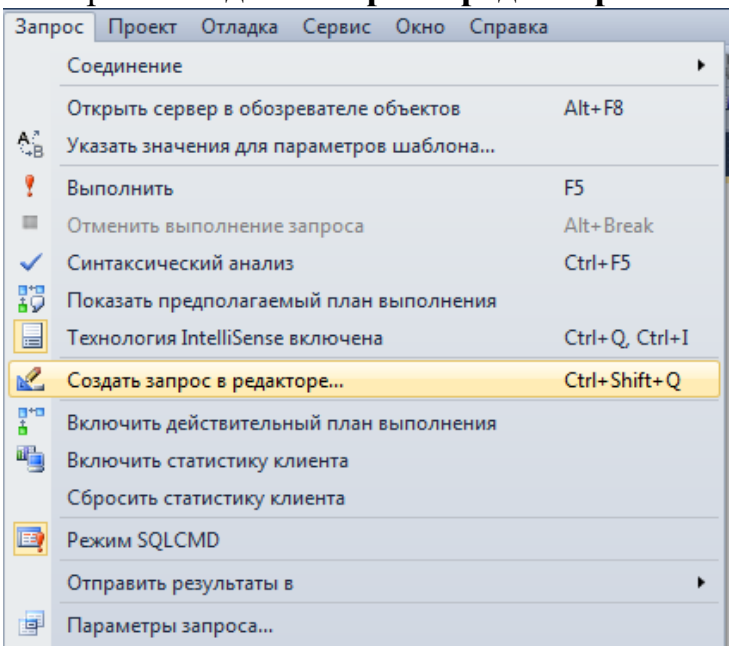
В Microsoft SQL Server Management Studio:

```
SELECT Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Сотрудники INNER JOIN Продажи ON Сотрудники.Код_сотрудника = Продажи.Код_сотрудника
```

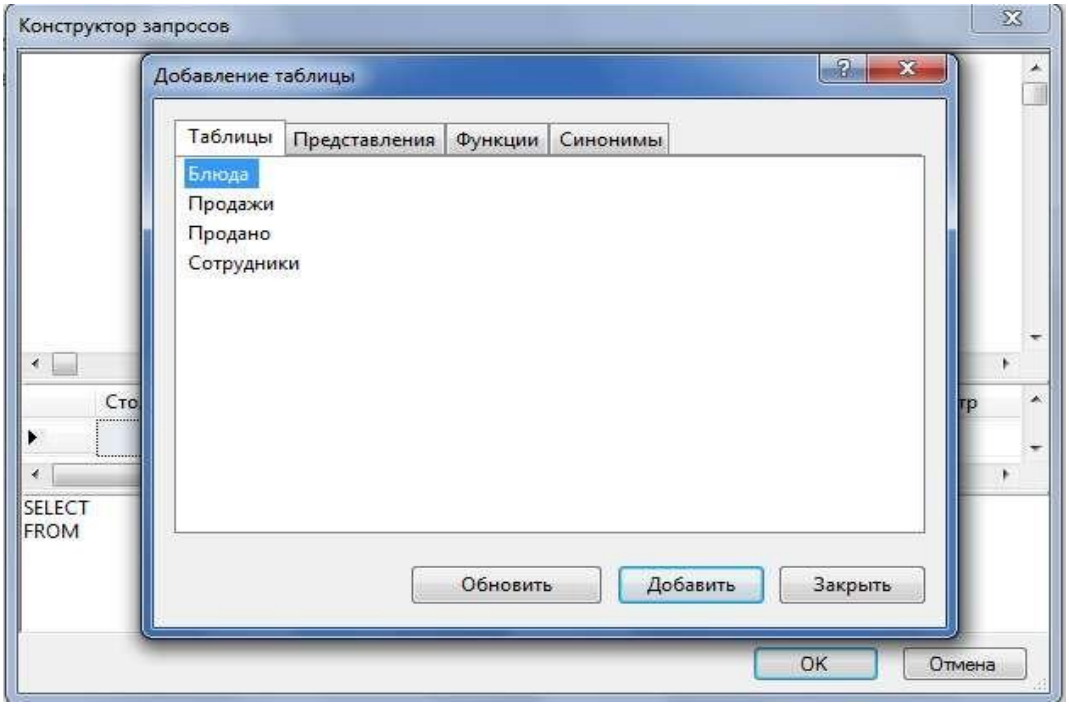
Результат:

Номер_продажи	Код_сотрудника	ФИО
1	2	Сидоров Петр Сергеевич
2	2	Сидоров Петр Сергеевич
3	1	Петрова Ирина Петровна
4	2	Сидоров Петр Сергеевич

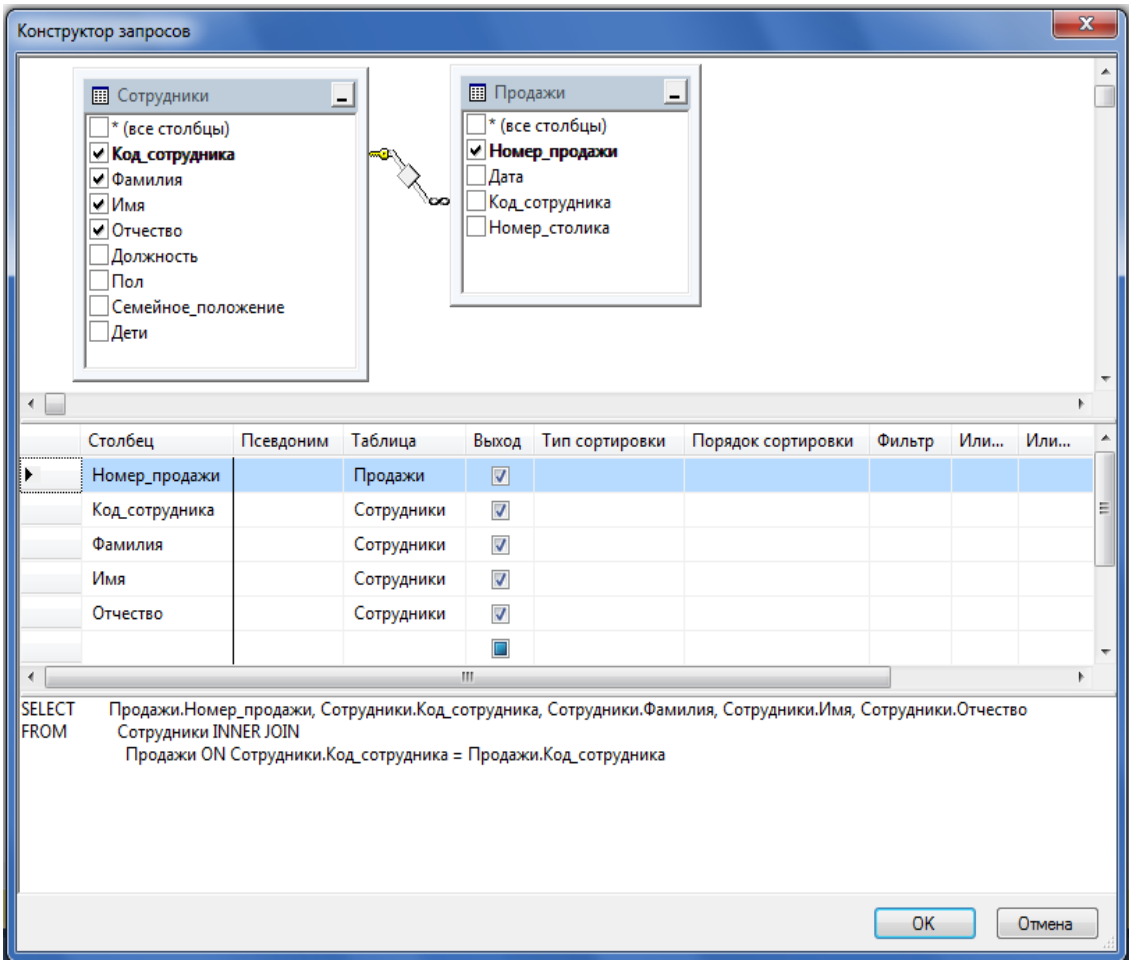
Для запросов можно использовать конструктор запросов. Для этого можно выбрать **Создать запрос в редакторе**:

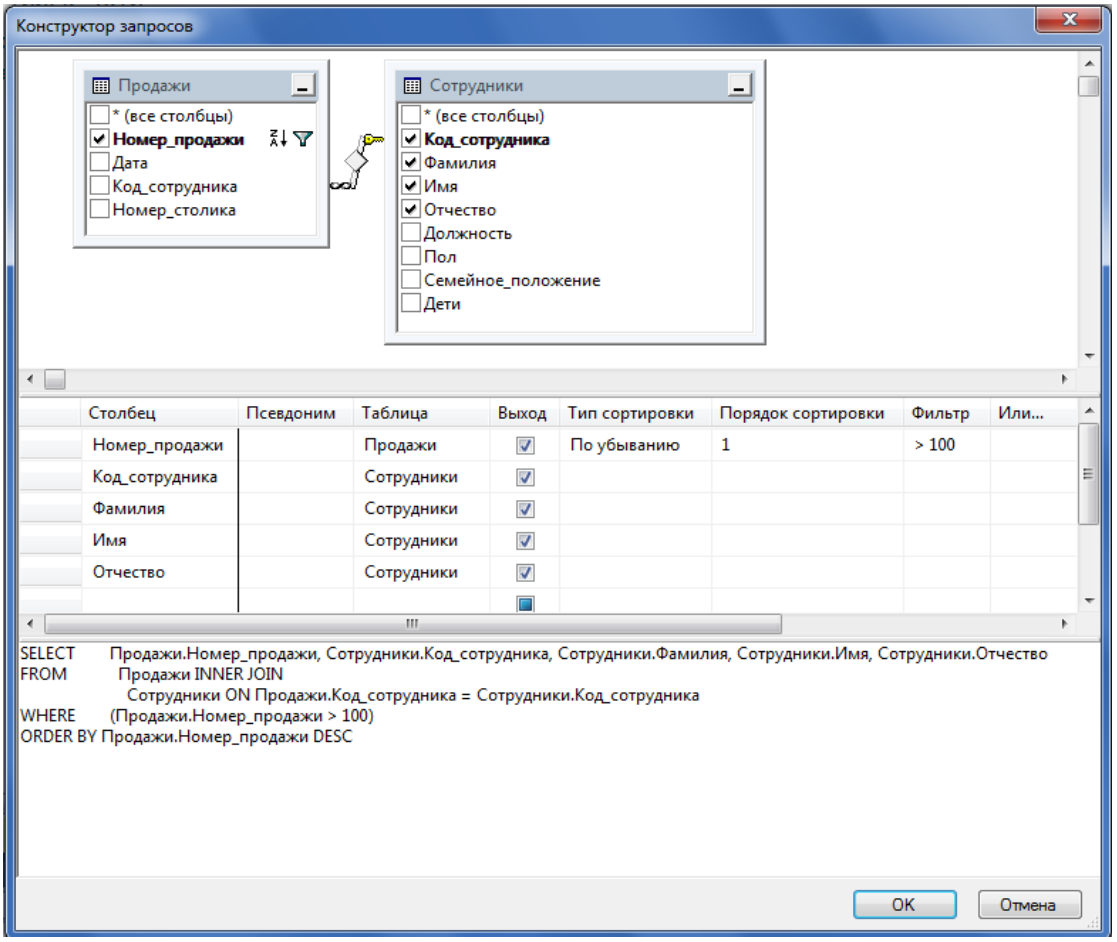


Выбрать необходимые таблицы:



Затем необходимые поля с необходимыми данными:





<u>Реализуем запросы через соответствующий интерфейс программы для кафе.</u>

Разместите на **tabPagePrimer** с палитры компонент **Toolbox** компонент **groupBox,** для которого укажите свойства:

- **Name** в **groupBoxSelect**;
- свойство **Text** в **Запросы по данным**;
- свойство **Dock** (расположение) в **Top**.

Разместите на **groupBoxSelect** с палитры компонент **Toolbox** компонент **radioButton** для первого запроса. Укажите для него свойства:

- **Name** в **radioButtonWorkers**;
- **Text** в **Сотрудники**.

Разместите на **groupBoxSelect** с палитры компонент **Toolbox** компонент **radioButton** для второго запроса. Укажите для него свойства:

- **Name** в **radioButtonDishes**;
- **Text** в **Блюда**.

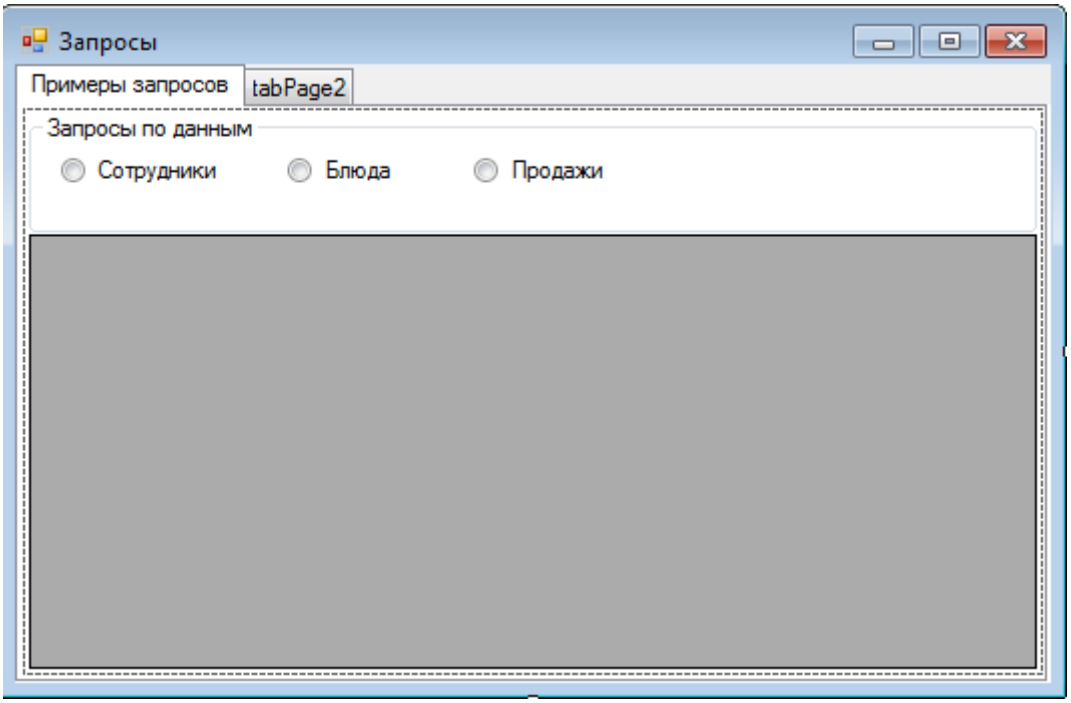
Разместите на **groupBoxSelect** с палитры компонент **Toolbox** компонент **radioButton** для третьего запроса. Укажите для него свойства:

- **Name** в **radioButtonSales**;
- **Text** в **Продажи**.

Под **groupBoxSelect** разместите для отображения данных запросов компонент **dataGridView** с палитры **Toolbox**. Для него укажите свойства:

- **Name** в **dataGridViewSelect**;
- **Dock** в **Fill** (вся область).

Станица имеет вид:



Запрограммируем кнопки Сотрудники, Блюда, Продажи.

Для СУБД Microsoft SQL Server:

Предварительно добавьте пространство имен:

```
using System.Data.SqlClient;
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
```

Для того, чтобы были доступны классы SqlClient для доступа к БД Microsoft SQL Server.

Организуем метод **FillDataGridView** для формирования набора данных по запросам в **dataGridView**. Входным параметром для него является строка с SQL запросом (**sqlSelect**), выходным набор данных:

```
//объявляем метод, на вход подаем строку запроса, а возвращаем объект DataTable
        DataTable FillDataGridView(string sqlSelect)
        {
            //Создаем объект connection класса SqlConnection для соединения с БД
            //CafeConnectionString - строка описания соединения с источником
данных

            SqlConnection connection = new
SqlConnection(Properties.Settings.Default.CafeConnectionString);
            //Создаем объект command для SQL команды
            SqlCommand command = connection.CreateCommand();
            //Заносим текст SQL запроса через параметр sqlSelect
            command.CommandText = sqlSelect;
            //Создаем объект adapter класса SqlDataAdapter
            SqlDataAdapter adapter = new SqlDataAdapter();
            //Задаем адаптеру нужную команду, в данном случае команду Select
            adapter.SelectCommand = command;
            //Создаем объект table для последующего отображения результата
запроса
            DataTable table = new DataTable();
            //заполним набор данных результатом запроса
            adapter.Fill(table);
            return table;
        }
```

По нажатию соответсвующих кнопок пропишите формирование строки запроса и вызов нашей функции **FillDataGridView**.

Для кнопки **Сотрудники** (**radioButtonWorkers**) пропишите:

```
private void radioButtonWorkers_CheckedChanged(object sender, EventArgs e)
        {
            dataGridViewSelect.DataSource = FillDataGridView("SELECT * FROM
Сотрудники");
        }
```

Для кнопки **Блюда** (**radioButtonDishes**) пропишите:

```
private void radioButtonDishes_CheckedChanged(object sender, EventArgs e)
        {
            dataGridViewSelect.DataSource = FillDataGridView(@"SELECT Код_блюда,
Название+'-'+Тип AS Подробнее, CAST(Цена_меню AS decimal(16,2)) AS [Текущая
цена], 'Произойдет' AS Пояснение, CAST(Цена_меню*1.5 AS decimal(16,2)) AS
[Повышение цены на 50%]  FROM Блюда");
        }
```

Для кнопки **Продажи** (**radioButtonSales**) пропишите:

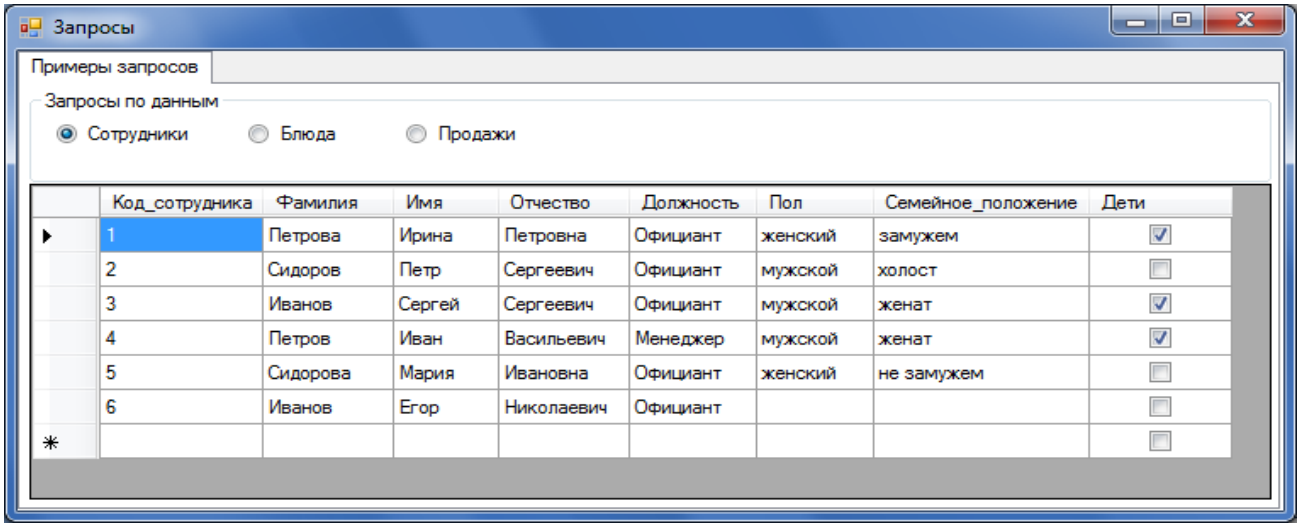
```
private void radioButtonSales_CheckedChanged(object sender, EventArgs e)
        {
            dataGridViewSelect.DataSource = FillDataGridView(@"SELECT
Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Продажи, Сотрудники WHERE  Продажи.Код_сотрудника =
Сотрудники.Код_сотрудника");
        }
```

или

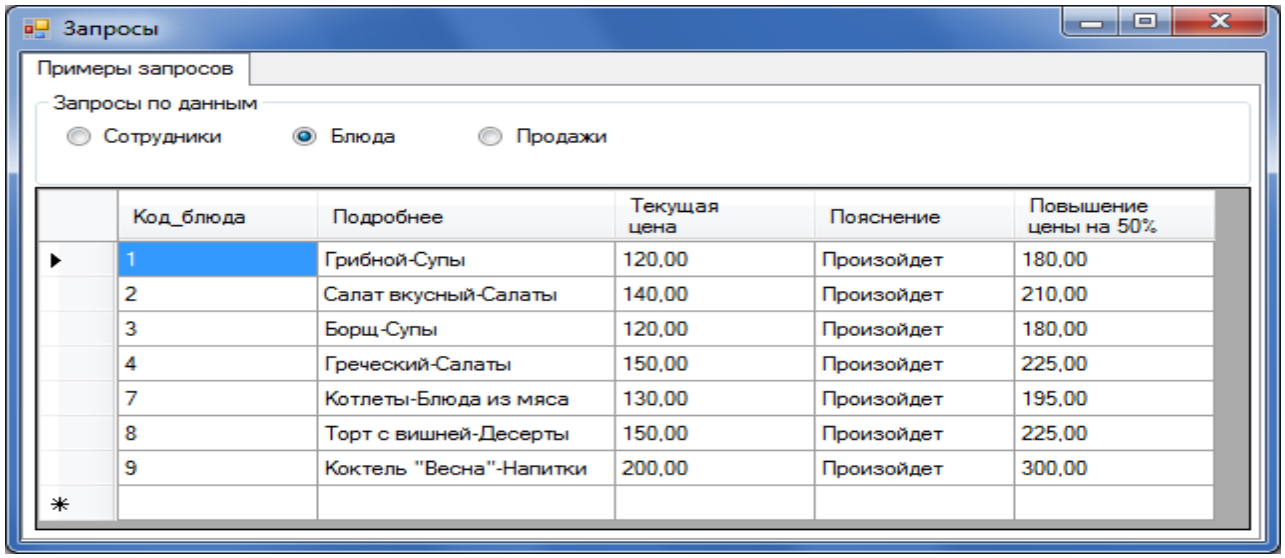
```
private void radioButtonSales_CheckedChanged(object sender, EventArgs e)
        {
            dataGridViewSelect.DataSource = FillDataGridView(@"SELECT
Номер_продажи, Сотрудники.Код_сотрудника, Фамилия+' '+Имя+' '+Отчество AS ФИО
FROM Сотрудники INNER JOIN Продажи ON Сотрудники.Код_сотрудника =
Продажи.Код_сотрудника");
        }
```

Запустите программу. Посмотрите, как отрабатывают полученные запросы.

Например:



	Код_сотрудника	Фамилия	Имя	Отчество	Должность	Пол	Семейное_положение	Дети
▶	1	Петрова	Ирина	Петровна	Официант	женский	замужем	☑
	2	Сидоров	Петр	Сергеевич	Официант	мужской	холост	☐
	3	Иванов	Сергей	Сергеевич	Официант	мужской	женат	☑
	4	Петров	Иван	Васильевич	Менеджер	мужской	женат	☑
	5	Сидорова	Мария	Ивановна	Официант	женский	не замужем	☐
	6	Иванов	Егор	Николаевич	Официант			☐
*								☐



	Код_блюда	Подробнее	Текущая цена	Пояснение	Повышение цены на 50%
▶	1	Грибной-Супы	120,00	Произойдет	180,00
	2	Салат вкусный-Салаты	140,00	Произойдет	210,00
	3	Борщ-Супы	120,00	Произойдет	180,00
	4	Греческий-Салаты	150,00	Произойдет	225,00
	7	Котлеты-Блюда из мяса	130,00	Произойдет	195,00
	8	Торт с вишней-Десерты	150,00	Произойдет	225,00
	9	Коктель "Весна"-Напитки	200,00	Произойдет	300,00
*					

3.3 Полная запись оператора SELECT

Перейдите на страницу **tabPage2**.
Для нее укажите:
- свойство **Name** в **tabPageSelect;**
свойство **Text** в **Полная запись SELECT**.

На **tabPageSelect** с палитры **Toolbox** разместите компонент **groupBox** для которого укажите свойства:
- **Name** в **groupBoxFSelect;**
- **Text** в **Прибыль сотрудников**;
- **Dock** в **Top.**

Будем просматривать прибыль сотрудника, чья фамилия вводится в строке ввода.

Для этого разместите на **groupBoxFSelect** компоненты с палитры **Toolbox**:
- компонент **label** для которого укажите свойства:
  - **Name** в **labelWorker**;
  - **Text** в **Фамилия сотрудника**;
- рядом с **labelWorker** компонент **textbox,** для которого укажите свойство **Name** в **textBoxWorker**.

Для детализации прибыли сотрудников по продажам и типам разместите на **groupBoxFSelect** компонент **groupBox,** для которого укажите свойства:
- **Name** в **groupBoxDet**;
- **Text** в **Детализация прибыли сотрудников**.

На **groupBoxDet** разместите:
- компонент **radioButton** для детализации по продажам. Укажите для нее свойства:
  - **Name** в **radioButtonDet_Sales**;
  - **Text** в **Прибыль сотрудника по продажам**.
- компонент **radioButton** для детализации по типам. Укажите для нее свойства:
  - **Name** в **radioButtonDet_Type**;
  - **Text** в **Прибыль сотрудника по типам**.
- компонент **radioButton** для отсутствия детализации. Укажите для нее свойства:
  - **Name** в **radioButtonDet_NO**;
  - **Text** в **Нет детализации**.

Будем отбирать данные по прибыли более определенной суммы, вводимой пользователем с клавиатуры.

Для этого разместите на **groupBoxFSelect** компонент **checkBox**, для которого укажите свойства:

- **Name** в **checkBoxMore**;
- **Text** в **Выбрать прибыль более.**

Рядом с **checkBoxMore** разместите строку ввода для ввода необходимой суммы. Для этого с палитры компонент **Toolbox** разместите компонент **textbox**, для которого укажите свойство **Name** в **textBoxMore**.

Включим сортировку по убыванию прибыли. Для этого разместите на **groupBoxFSelect** компонент **checkBox**, для которого укажите свойства:

- **Name** в **checkBoxOrder**;
- **Text** в **Включить сортировку по убыванию прибыли**.

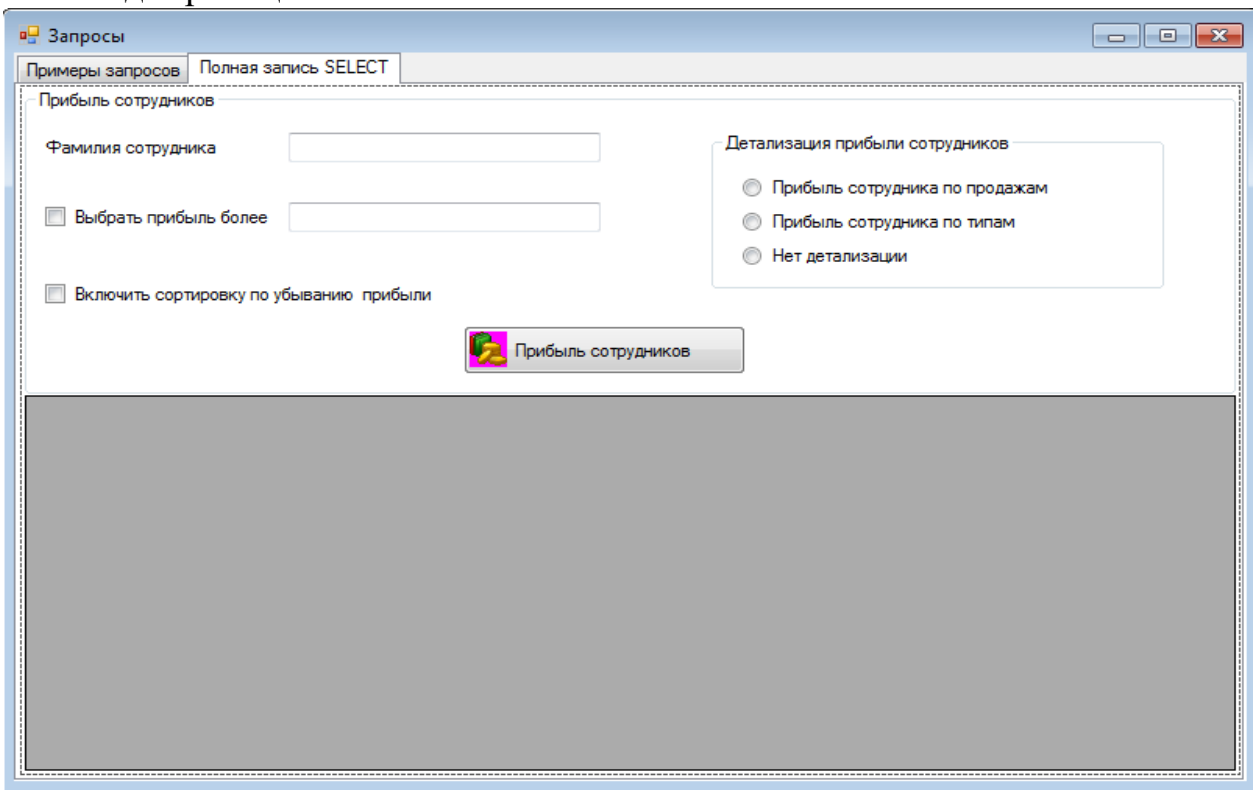
Выполнять запрос будем по нажатию кнопки. Для этого разместите на **groupBoxFSelect** с палитры компонент **Toolbox** компонент **button**, для которого укажите свойства:

- **Name** в **buttonF_select**;
- **Text** в **Прибыль сотрудников**;
- **Image** в соответствующую картинку.

Результат будем представлять в dataGridView. Разместите на **groupBoxFSelect** компонент **dataGridView**. Для него укажите свойства:

- **Dock** в **Fill**;
- **Name** в **dataGridViewFSelect**.

Вид страницы:



Запрограммируем кнопку **Прибыль сотрудников (buttonF_select)**.

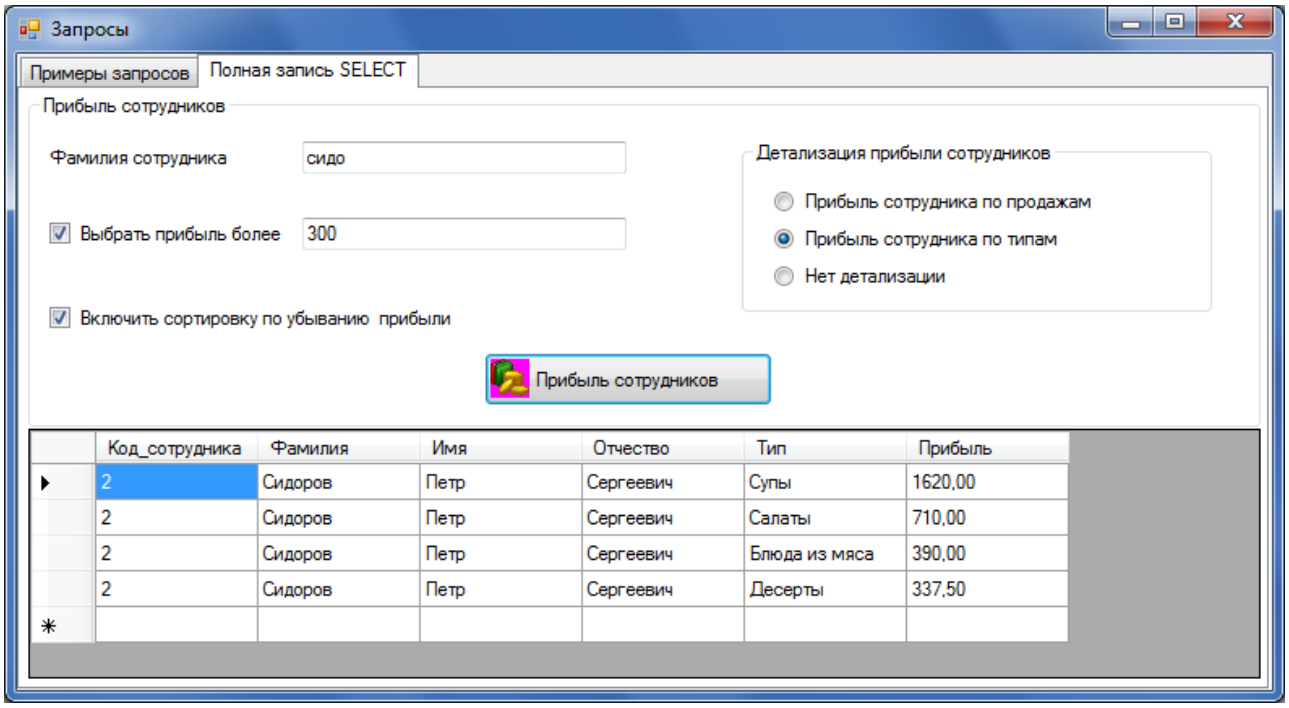
Для СУБД Microsoft SQL Server:

```
private void buttonF_select_Click(object sender, EventArgs e)
        {
            if (String.IsNullOrEmpty(textBoxWorker.Text))
            {
                MessageBox.Show("Обязательно укажите фамилию необходимого
сотрудника.\nДопустим ввод первых символов.", "Внимание", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }
            if (checkBoxMore.Checked && String.IsNullOrEmpty(textBoxMore.Text))
            {
                MessageBox.Show("Не указана прибыль в условии", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                checkBoxMore.Checked = false;
                return;
            }
            string sqlSelect = "";
            if (radioButtonDet_Sales.Checked)
                sqlSelect = @"SELECT w.Код_сотрудника, Фамилия, Имя, Отчество,
s.Номер_продажи,
                CAST(Sum(Количество*Цена*(1-Скидка)) AS decimal(16,2)) AS
Прибыль
                FROM Сотрудники w, Продажи s, Продано ds, Блюда d
                WHERE Фамилия LIKE @last_name  AND
                w.Код_сотрудника = s.Код_сотрудника AND
                s.Номер_продажи = ds.Номер_продажи AND
                ds.Код_блюда = d.Код_блюда
                GROUP BY w.Код_сотрудника, Фамилия, Имя, Отчество,
s.Номер_продажи";
            else
                if (radioButtonDet_Type.Checked)
                    sqlSelect = @"SELECT Сотрудники.Код_сотрудника, Фамилия,
Имя, Отчество, Тип,
                    CAST(Sum(Количество*Цена*(1-Скидка)) AS decimal(16,2)) AS
Прибыль
                    FROM Сотрудники INNER JOIN (Продажи INNER JOIN (Продано
INNER JOIN Блюда ON Продано.Код_блюда = Блюда.Код_блюда) ON
Продажи.Номер_продажи = Продано.Номер_продажи)
                    ON Сотрудники.Код_сотрудника = Продажи.Код_сотрудника
                    WHERE Фамилия LIKE @last_name
                    GROUP BY Сотрудники.Код_сотрудника, Фамилия, Имя, Отчество,
Тип";
                else
                    sqlSelect = @"SELECT Сотрудники.Код_сотрудника, Фамилия,
Имя, Отчество,
                    CAST(Sum(Количество*Цена*(1-Скидка)) AS decimal(16,2)) AS
Прибыль
                    FROM Сотрудники INNER JOIN Продажи ON
Сотрудники.Код_сотрудника=Продажи.Код_сотрудника
                    INNER JOIN Продано ON
Продажи.Номер_продажи=Продано.Номер_продажи
                    INNER JOIN Блюда ON Продано.Код_блюда=Блюда.Код_блюда AND
Фамилия LIKE @last_name
                    GROUP BY Сотрудники.Код_сотрудника, Фамилия, Имя, Отчество";

            if (checkBoxMore.Checked)
                sqlSelect += " HAVING Sum(Количество*Цена*(1-Скидка)) >@amount";
            if (checkBoxOrder.Checked)
```

```
                sqlSelect += " ORDER BY Sum(Количество*Цена*(1-Скидка)) desc";

            SqlConnection connection = new
SqlConnection(Properties.Settings.Default.CafeConnectionString);
            SqlCommand command = connection.CreateCommand();
            command.CommandText = sqlSelect;
            command.Parameters.AddWithValue("@last_name", textBoxWorker.Text +
"%");

            if (checkBoxMore.Checked)
                try
                {
                    command.Parameters.Add("@amount", SqlDbType.Money).Value =
Double.Parse(textBoxMore.Text);
                }
                catch
                {
                    MessageBox.Show("Прибыль в условии должна быть задана
числом", "ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    checkBoxMore.Checked = false;
                    return;
                }

            SqlDataAdapter adapter = new SqlDataAdapter();
            adapter.SelectCommand = command;
            DataTable table = new DataTable();
            adapter.Fill(table);
            dataGridViewFSelect.DataSource = table;
            if (table.Rows.Count == 0) MessageBox.Show("Нет значений!",
"Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

        }
```

Запустите программу. Посмотрите, как отрабатывают сформированные запросы.

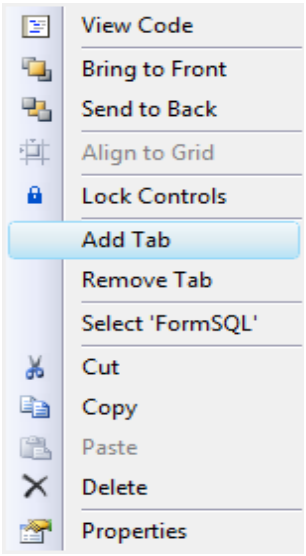
Например:



3.4 Подзапросы

Покажем пример использования подзапросов.

Добавьте новую станицу на **tabControlSQL**. Для это нажмите на **tabControlSQL** правую кнопку мыши и выберете **Add Tab**:



Задайте для новой страницы:
- свойство **Name** для новой страницы в **tabPageSubquery**;
- свойство **Text** в **Примеры подзапросов**.

Разместите на **tabPageSubquery** с палитры компонент **Toolbox** компонент **groupBox,** для которого укажите свойства:
- **Name** в **groupBoxSubquery**;
- свойство **Text** в **Подзапросы по данным**;
- свойство **Dock** (расположение) в **Top**.

Разместите на **groupBoxSubquery** с палитры компонент **Toolbox** компонент **radioButton** для первого подзапроса. Укажите для него свойства:
- **Name** в **radioButtonCorrelated**;
- **Text** в **Коррелированный подзапрос**.

Разместите на **groupBoxSubquery** с палитры компонент **Toolbox** компонент **radioButton** для второго запроса. Укажите для него свойства:
- **Name** в **radioButtonNoCorrelated**;
- **Text** в **Некоррелированный подзапрос**.

Будем просматривать подзапросы для продажи, чей номер вводится в строке ввода.

Для этого разместите на **groupBoxSubquery** компоненты с палитры **Toolbox**:

- компонент **label** для которого укажите свойства:
    - **Name** в **labelNumber**;
    - **Text** в **Номер продажи**;
- рядом с **labelNumber** компонент **textBox** для указания номера. Для него укажите свойство **Name** в **textBoxNumber.**

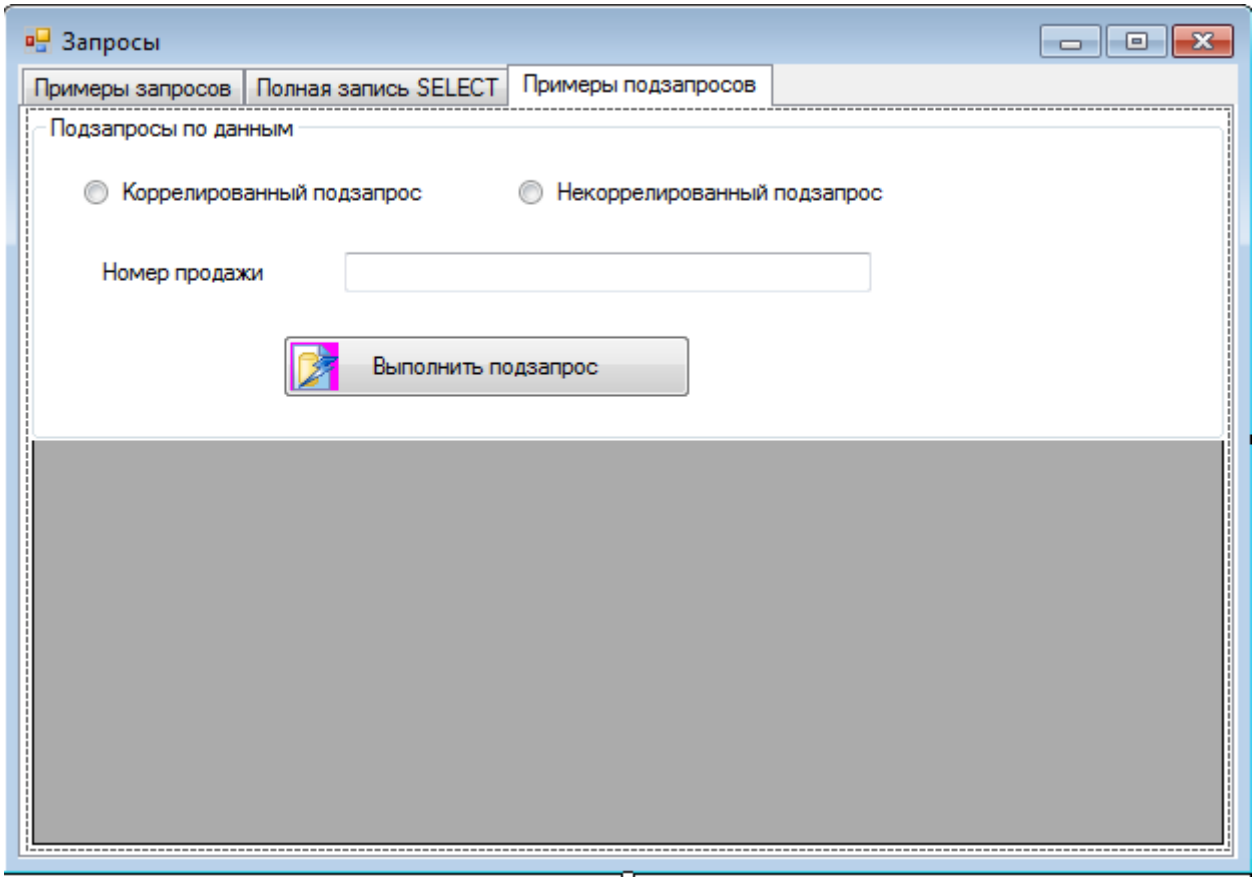
Разместите на **groupBoxSubquery** с палитры компонент **Toolbox** компонент **button** для запуска подзапросов. Укажите для него свойства:

- **Name** в **buttonSubquery**;
- **Text** в **Выполнить подзапрос**;
- **Image** в соответствующую картинку.

Под **groupBoxSubquery** разместите для отображения данных запросов компонент **dataGridView** с палитры **Toolbox**. Для него укажите свойства:

- **Name** в **dataGridViewSubquery**;
- **Dock** в **Fill** (вся область).

Вид страницы:



Запрограммируем кнопку **Выполнить подзапрос (buttonSubquery)**.

Для СУБД Microsoft SQL Server:

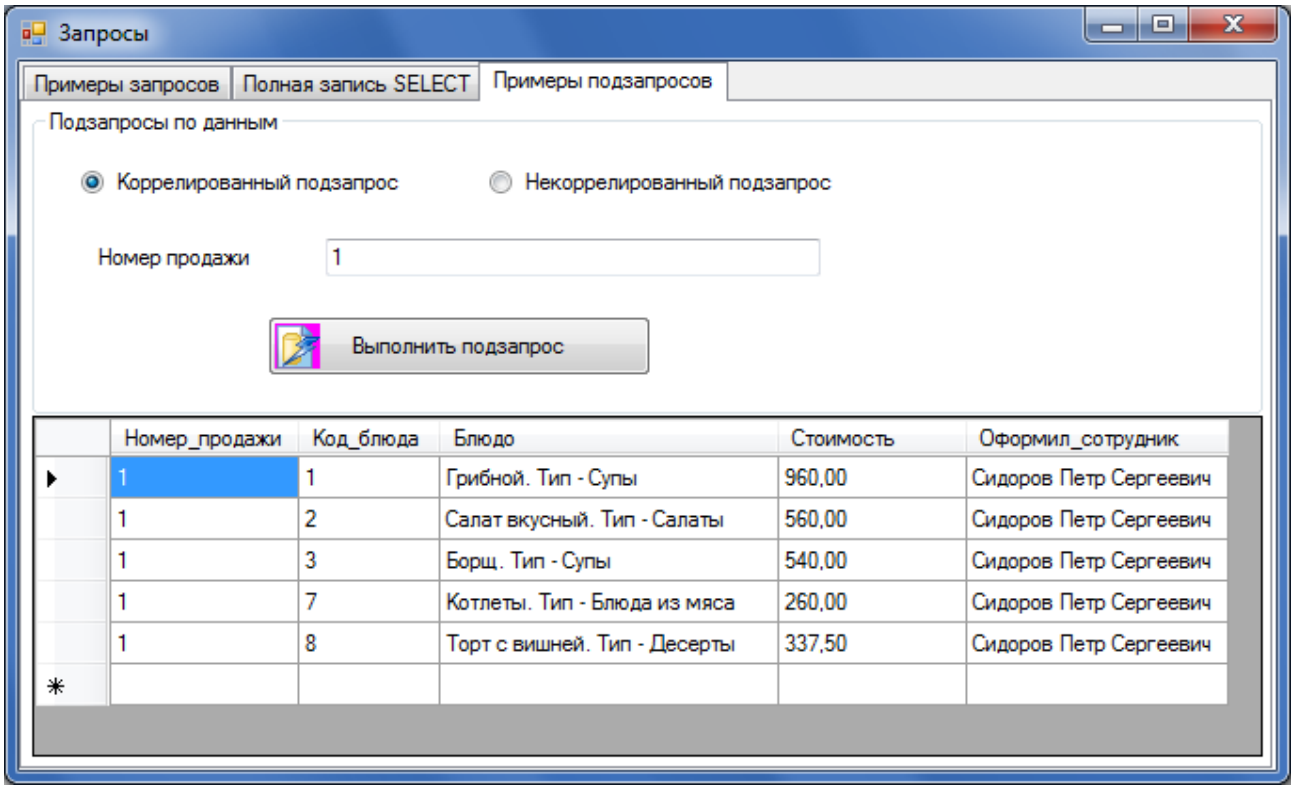
```
private void buttonSubquery_Click(object sender, EventArgs e)
```

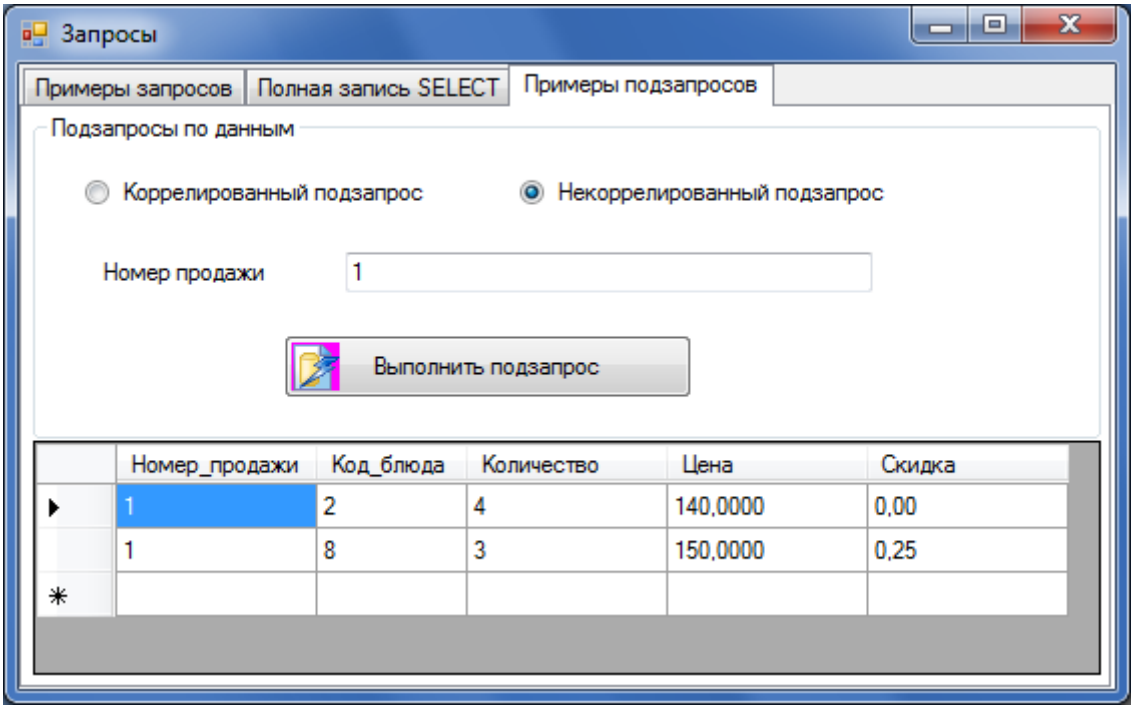
```
        {
            if (String.IsNullOrEmpty(textBoxNumber.Text))
            {
                MessageBox.Show("Обязательно укажите номер необходимой продажи",
"Внимание", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            string sqlSelect = "";

            if (radioButtonCorrelated.Checked)

                sqlSelect = @"SELECT Продано.Номер_продажи, Продано.Код_блюда,
                (SELECT Название+'. Тип - '+Тип FROM Блюда WHERE Код_блюда
=Продано.Код_блюда) AS Блюдо,
                CAST(Количество*Цена*(1-Скидка) AS decimal(16,2)) AS Стоимость,
                (SELECT Фамилия+' '+Имя+' '+Отчество FROM Сотрудники
                    WHERE Код_сотрудника IN (SELECT Код_сотрудника FROM Продажи
                    WHERE Номер_продажи=Продано.Номер_продажи)) AS
Оформил_сотрудник
                FROM Продано
                WHERE Номер_продажи = @number";

            else
                if (radioButtonNoCorrelated.Checked)
                    sqlSelect = @"SELECT * FROM Продано
                    WHERE Номер_продажи =@number AND Цена > (SELECT AVG(Цена)
FROM Продано)";
                else
                {
                    MessageBox.Show("Не выбрали вид подзапроса", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }

            SqlConnection connection = new
SqlConnection(Properties.Settings.Default.CafeConnectionString);
            SqlCommand command = connection.CreateCommand();
            command.CommandText = sqlSelect;
            try
            {
                command.Parameters.Add("@number", SqlDbType.Int).Value =
int.Parse(textBoxNumber.Text);
            }
            catch
            {
                MessageBox.Show("Номер продажи в условии должен быть задан
числом", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            SqlDataAdapter adapter = new SqlDataAdapter();
            adapter.SelectCommand = command;
            DataTable table = new DataTable();
            adapter.Fill(table);
            dataGridViewSubquery.DataSource = table;
            if (table.Rows.Count == 0) MessageBox.Show("Нет значений!",
"Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

        }
```

Запустите программу. Посмотрите, как отрабатывают сформированные

запросы.

Например:





3.5 Запросы изменения данных

Покажем пример использования запросов изменения данных (**INSERT, UPDATE, DELETE**) на примере таблицы **Блюда**.

Добавьте новую станицу на **tabControlSQL**. Для это нажмите на **tabControlSQL** правую кнопку мыши и выберете **Add Tab**.

Задайте для новой страницы:

- свойство **Name** в **tabPageDML**;
- свойство **Text** в **Запросы изменения данных**.

Перейдите на страницу **tabPageDML**.

Разместите с палитры компонент **Toolbox** компонент **groupBox**, для которого укажите свойства:

- **Name** в **groupBoxDML**;
- **Text** в **Операторы**;
- **Dock** в **Top**.

На **groupBoxDML** разместите с палитры компонент **Toolbox**:

- компонент **radioButton** для добавления. Укажите для нее свойства:
  - **Name** в **radioButtonInsert_dish**;
  - **Text** в **Добавить данные по блюду**;

- компонент **radioButton** для редактирования. Укажите для нее свойства:
  - **Name** в **radioButtonUpdate_dish**;
  - **Text** в **Изменить данные по блюду с заданным кодом**.

- компонент **radioButton** для удаления. Укажите для нее свойства:
  - **Name** в **radioButtonDelete_dish**;
  - **Text** в **Удалить данные по блюду с заданным кодом**.

Под **groupBoxDML** разместите с палитры **Toolbox** компонент **label**, для которого укажите свойства:

- **Name** в **labelId_dish**;
- **Text** в **Код блюда**;

- рядом с **labelId_dish** компонент **textbox**, для которого укажите свойство **Name** в **textBoxId_dish**.

Так как логика работы добавления, редактирования и удаления будет разной, то для удобства разместим дальнейшие компоненты на компонент панель. Для этого разместите с палитры **Toolbox** компонент **panel**, для которого укажите свойство **Name** в **panelDish**.

На панель **panelDish** разместите с палитры **Toolbox**:

- компонент **label**, для которого укажите свойства:
  - **Name** в **labelName_dish**;

- **Text** в **Название блюда**;

- рядом с **labelName_dish** компонент **textbox**, для которого укажите свойство **Name** в **textBoxName_dish**;

- компонент **label** для которого укажите свойства:
  - **Name** в **labelType_dish**;
  - **Text** в **Тип блюда**;

- рядом с **labelType_dish** компонент **textBox,** для которого укажите свойство **Name** в **textBoxType_dish**;

- компонент **label** для которого укажите свойства:
  - **Name** в **labelPrice_dish**;
  - **Text** в **Цена блюда**;

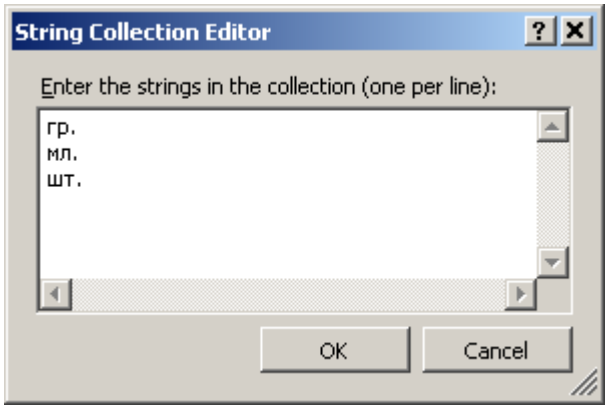
- рядом с **labelPrice_dish** компонент **textBox**, для которого укажите свойство **Name** в **textBoxPrice_dish**;

- компонент **label** для которого укажите свойства:
  - **Name** в **labelWeight_dish**;
  - **Text** в **Выход(вес) блюда**;

- рядом с **labelWeight_dish** компонент **textBox**, для которого укажите свойство **Name** в **textBoxWeight_dish**;

- компонент **label** для которого укажите свойства:
  - **Name** в **labelEd_dish**;
  - **Text** в **Единицы измерения**;

- рядом с **labelEd_dish** компонент **listBox**, для которого укажите свойства:
  - **Name** в **listBoxEd_dish**;
  - **Items** в:



- компонент **checkBox** для которого укажите свойства:
  - **Name** в **checkBoxFirm**;

- **Text** в **Фирменное блюдо**;

- компонент **label** для которого укажите свойства:
  - **Name** в **labelPhoto_dish**;
  - **Text** в **Фото блюда**;

- рядом с **labelPhoto_dish** компонент **pictureBox**, для которого укажите свойства:
  - **Name** в **pictureBoxPhoto_dish**;
  - **SizeMode** в **StretchImage**;

- компонент **label** для которого укажите свойства:
  - **Name** в **labelIng_dish**;
  - **Text** в **Состав блюда**;

- рядом с **labelIng_dish** компонент **textBox**, для которого укажите свойства:
  - **Name** в **textBoxIng_dish**;
  - **Multiline** в **True**;
  - **ScrollBars** в **Vertical** или в **Both**.

Для удобства загрузки фотографии блюда из файла разместите рядом с **pictureBoxPhoto_dish** компонент **button**, для которого укажите свойства:
- **Name** в **buttonOpenPhoto_dish**;
- **Text** в **Открыть фото блюда**.

Для загрузки фото из файла положите на форму диалог **openFileDialog**. Укажите для него свойство **Name** в **openFileDialogDish**.

Запрограммируем нажатие кнопки **buttonOpenPhoto_dish** для загрузки фото в **pictureBoxPhoto_dish**:

```
string fileImage = "";

        private void buttonOpenPhoto_dish_Click(object sender, EventArgs e)
        {
            openFileDialogDish.Title = "Укажите файл для фото";
            if (openFileDialogDish.ShowDialog() == DialogResult.OK)
            {
                fileImage = openFileDialogDish.FileName;
                try
                {
            pictureBoxPhoto_dish.Load(openFileDialogDish.FileName);
                }
                catch
                {
                    MessageBox.Show("Выбран не тот формат файла", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }
```

```
                    }
            else fileImage = "";
        }
```

Вызывать выполнение соответствующего запроса будем по нажатию кнопки **Выполнить запрос**.

Для этого разместите с палитры компонент **Toolbox** компонент **button**, для которого укажите свойства:

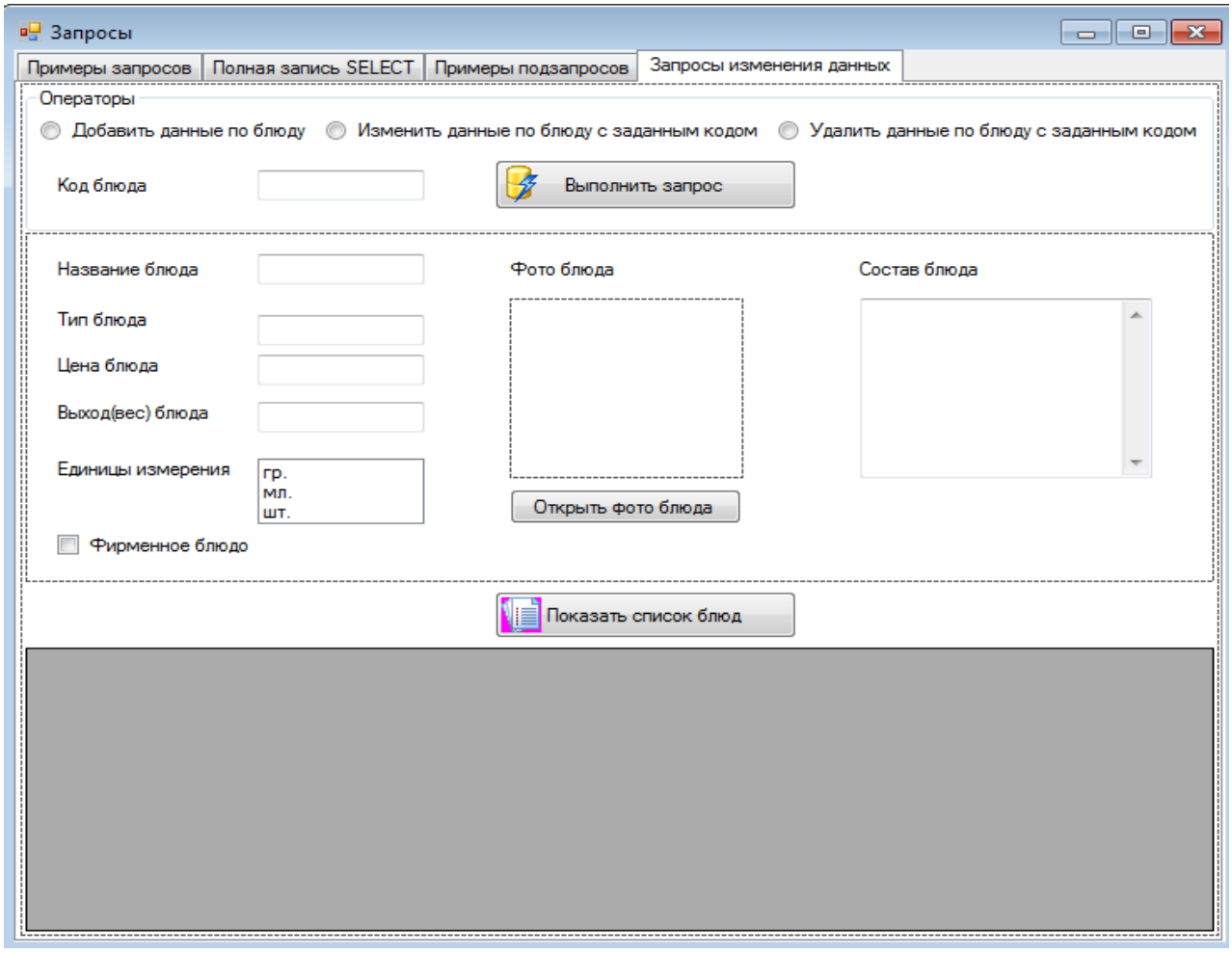
- **Name** в **buttonExecuteDML**;
- **Text** в **Выполнить запрос**;
- **Image** в соответствующую картинку.

Чтобы отобразить результаты выполнения запросов разместите на **tabPageDML** с палитры **ToolBox** компонент **dataGridView**, для которого свойство **Name** укажите в **dataGridViewDish**.

Для добавления данных в **dataGridViewDish** разместите кнопку **button** с палитры **ToolBox** для которой укажите свойства:

- **Name** в **buttonSelectDishes**;
- **Text** в **Показать список блюд**.

Таким образом, вид страницы имеет вид:



При добавлении данных в таблицу Блюда в соответствующие поля будут добавлены соответствующие значения из соответствующих компонент на форме.

Данные значения для ввода могут быть различны, следовательно, запрос будет не с заранее определенными значениями, а с параметрами, которые будут подставляться из соответствующих компонент на форме.

Запрос на добавление будет иметь вид:

INSERT INTO Блюда (Код_блюда, Название, Тип, Цена_меню, Выход, Единицы, Фирменное_блюдо, Фото, Состав)
VALUES (@id, @Name, @Type, @Price, @Weight, @Ed, @Firm, @Photo, @Sostav)"

Где параметры обозначены @.

Оформим данный запрос через соответствующий метод.

Колонки **Код_блюда, Название, Тип, Цена_меню** в данной БД обязательны для заполнения. Если их пользователь не укажет при добавлении нового блюда, то СУБД сгенерирует соответствующее исключение. Чтобы вместо этого сообщения выдать более понятное пользователю сообщение, необходимо отследить данную ситуацию в приложении, когда пользователь для нового блюда не указал Код_блюда, Название, Тип, Цену меню.

Следует отметить, что пользователь может по ошибке задать для колонок **Код_блюда, Цена** и **Вес** не числовые значения. В этом случае СУБД сгенерирует исключение. Чтобы вместо этого сообщения выдать понятное пользователю сообщение, необходимо отследить данную ситуацию в приложении.

Таким образом, получаем следующий метод для добавления данных о блюде **InsertDish()**:

Для СУБД Microsoft SQL Server:

```
void InsertDish()
        {
            if (String.IsNullOrEmpty(textBoxId_dish.Text) ||
(String.IsNullOrEmpty(textBoxName_dish.Text) ||
(String.IsNullOrEmpty(textBoxType_dish.Text) ||
(String.IsNullOrEmpty(textBoxPrice_dish.Text)))))
            {
                MessageBox.Show("Обязательно введите код блюда, название, тип и
цену блюда", "Внимание", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            int id;
            if (!int.TryParse(textBoxId_dish.Text, out id))
```

```
            {
                MessageBox.Show("Некоректное значение кода блюда!", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            double price = 0;
            if (!double.TryParse(textBoxPrice_dish.Text, out price))
            {
                MessageBox.Show("Некоректное значение цены!", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            double weight = 0;
            if ((!String.IsNullOrEmpty(textBoxWeight_dish.Text)) &&
(!double.TryParse(textBoxWeight_dish.Text, out weight)))
            {
                MessageBox.Show("Некоректное значение веса!", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            string sqlInsert = @"INSERT INTO Блюда (Код_блюда, Название, Тип,
Цена_меню, Выход, Единицы, [Фирменное_ блюдо], Фото, Состав)
                                 VALUES (@id, @Name, @Type, @Price, @Weight, @Ed,
@Firm, @Photo, @Sostav)";
            SqlConnection connection = new
SqlConnection(Properties.Settings.Default.CafeConnectionString);
            connection.Open();
            SqlCommand command = connection.CreateCommand();
            command.CommandText = sqlInsert;
            command.Parameters.AddWithValue("@id", id);
            command.Parameters.AddWithValue("@Name", textBoxName_dish.Text);
            //или другим способом, если необходимо явное указание типа данных
            command.Parameters.Add("@Type", SqlDbType.NVarChar).Value =
textBoxType_dish.Text;
            command.Parameters.AddWithValue("@Price", price);
            command.Parameters.AddWithValue("@Weight", weight);

            if (!String.IsNullOrEmpty(listBoxEd_dish.Text))
                command.Parameters.AddWithValue("@Ed", listBoxEd_dish.Text);
            else
                command.Parameters.AddWithValue("@Ed", DBNull.Value);

            command.Parameters.AddWithValue("@Firm", checkBoxFirm.Checked);

            if (!String.IsNullOrEmpty(fileImage))
                command.Parameters.AddWithValue("@Photo",
File.ReadAllBytes(fileImage));
            else
            {
                command.Parameters.Add("@Photo", SqlDbType.VarBinary);
                command.Parameters["@Photo"].Value = DBNull.Value;
            }

            command.Parameters.AddWithValue("@Sostav", textBoxIng_dish.Text);

            try
            {

                command.ExecuteNonQuery();
            }
```

```
catch(Exception err)
```

```
            {
                MessageBox.Show("Ошибка выполнения запроса.\n" + err.Message,
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            connection.Close();
            buttonSelectDishes_Click(this, EventArgs.Empty);
        }
```

Примечание:

Для того, чтобы загрузить фотографию из файла через `File.ReadAllBytes()` добавьте пространство имен:

```
using System.IO;
```

Покажем запрос на редактирование. Будем менять данные по блюду с указанным в соответствующей строке вводе коду блюда (**textBoxId_dish**).

Запрос будет иметь вид:

UPDATE Блюда
SET Название=@Name,
Тип=@Type,
Цена_меню=@Price,
Выход=@Weight,
Единицы=@Ed,
Фирменное_блюдо=@Firm,
Фото=@Photo,
Состав=@Sostav
WHERE Код_блюда=@Id_dish

Следует отметить, что при редактировании данных пользователь для заданного кода блюда может изменять значения не во всех полях (в том числе и полях обязательных для заполнения при добавлении записи о новом блюде).

При логике редактирования, не указанные в строках ввода пользователем значения, остаются в БД не изменяемыми. Меняются только те, что указаны пользователем в строках ввода. В этом случае необходимо в зависимости от заполненных данных формировать и сам текст оператора UPDATE (формируемый запрос с параметрами).

Оформим данный запрос через соответствующий метод **UpdateDish()**:

Для СУБД Microsoft SQL Server:

```
void UpdateDish()
        {
            if (String.IsNullOrEmpty(textBoxId_dish.Text))
            {
                MessageBox.Show("Обязательно укажите код блюда, для которого
будете менять данные", "Внимание", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }

            int id;
            if (!int.TryParse(textBoxId_dish.Text, out id))
            {
                MessageBox.Show("Некоректное значение кода блюда!", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            double price=0;
            if ((!String.IsNullOrEmpty(textBoxPrice_dish.Text)) &&
(!double.TryParse(textBoxPrice_dish.Text, out price)))
            {
                MessageBox.Show("Некоректное значение цены!", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            double weight=0;
            if ((!String.IsNullOrEmpty(textBoxWeight_dish.Text)) &&
(!double.TryParse(textBoxWeight_dish.Text, out weight)))
            {
                MessageBox.Show("Некоректное значение веса!", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            string sqlUpdate = "UPDATE Блюда SET {0} [Фирменное_ блюдо]=@Firm
WHERE Код_блюда=@Id_dish";
            SqlConnection connection = new
SqlConnection(Properties.Settings.Default.CafeConnectionString);
            connection.Open();
            SqlCommand command = connection.CreateCommand();

            string sqlValues = "";

            if (!String.IsNullOrEmpty(textBoxName_dish.Text))
                sqlValues += "Название=@Name,";
            if (!String.IsNullOrEmpty(textBoxType_dish.Text))
                sqlValues += "Тип=@Type,";
            if (!String.IsNullOrEmpty(textBoxPrice_dish.Text))
                sqlValues += "Цена_меню=@Price,";
            if (!String.IsNullOrEmpty(textBoxWeight_dish.Text))
                sqlValues += "Выход=@Weight,";
            if (!String.IsNullOrEmpty(listBoxEd_dish.Text))
                sqlValues += "Единицы=@Ed,";
            if (!String.IsNullOrEmpty(fileImage))
                sqlValues += "Фото=@Photo,";
            if (!String.IsNullOrEmpty(textBoxIng_dish.Text))
                sqlValues += "Состав=@Sostav,";

            command.CommandText = String.Format(sqlUpdate, sqlValues);

            if (!String.IsNullOrEmpty(textBoxName_dish.Text))
```

```
            command.Parameters.AddWithValue("@Name", textBoxName_dish.Text);
```

```
            //или другим способом, если необходимо явное указание типа данных
            if (!String.IsNullOrEmpty(textBoxType_dish.Text))
                command.Parameters.Add("@Type", SqlDbType.NVarChar).Value =
textBoxType_dish.Text;

            if (!String.IsNullOrEmpty(textBoxPrice_dish.Text))
                command.Parameters.AddWithValue("@Price", price);

            if (!String.IsNullOrEmpty(textBoxWeight_dish.Text))
                command.Parameters.AddWithValue("@Weight", weight);

            if (!String.IsNullOrEmpty(listBoxEd_dish.Text))
                command.Parameters.AddWithValue("@Ed", listBoxEd_dish.Text);

            if (!String.IsNullOrEmpty(fileImage))
                command.Parameters.AddWithValue("@Photo",
File.ReadAllBytes(fileImage));

            if (!String.IsNullOrEmpty(textBoxIng_dish.Text))
                command.Parameters.AddWithValue("@Sostav",
textBoxIng_dish.Text);

            command.Parameters.AddWithValue("@Firm", checkBoxFirm.Checked);

            command.Parameters.AddWithValue("@Id_dish", id);
            try
            {
                command.ExecuteNonQuery();
            }
            catch (Exception err)
            {
                MessageBox.Show("Ошибка выполнения запроса:\n" + err.Message,
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            connection.Close();
            buttonSelectDishes_Click(this, EventArgs.Empty);
        }
```

Примечание:

В данном методе **UpdateDish()** формирование строки запроса на обновление идет в зависимости от заполненных пользователем данных в соответствующих компонентах, для которых потом подставляются соответствующие параметры (**формируемый параметрический запрос**).

Сбор строки запроса на обновление происходит при помощи метода **String.Format**(), который в указанную строку **sqlUpdate** подставляет указанную строку **sqlValues** в параметр, определенный в **{0}**, в данном случае в раздел после **SET**:

UPDATE Блюда SET {0} [Фирменное_ блюдо]=@Firm WHERE Код_блюда=@Id_dish

Следует отметить, что **Фирменное_блюдо** указывается в данном запросе всегда, так как определяется по состоянию **checkBoxFirm.Checked**, а

Код_блюда необходим для указания того, какое именно блюдо будет редактироваться.

Если убрать поле **Фирменное_блюдо** из запроса, то при формировании запроса строка **sqlValues** всегда заканчивается символом «,» что с полем **Фирменное_блюдо** является корректным, а без него приведет к ошибке синтаксиса оператора **UPDATE**.

Чтобы исключить данную последнюю запятую из текста запроса (в случае отсутствии поля **Фирменное_блюдо** в запросе) можно использовать:

sqlValues = sqlValues.Substring(0, sqlValues.Length - 1);

Удаление данных по блюду будем проводить по заданному пользователем коду блюда в строке ввода **textBoxId_dish**.

Запрос имеет вид:

DELETE FROM Блюда WHERE Код_блюда=@Id_dish

Оформим данный запрос через соответствующий метод **DeleteDish():**

Для СУБД Microsoft SQL Server:

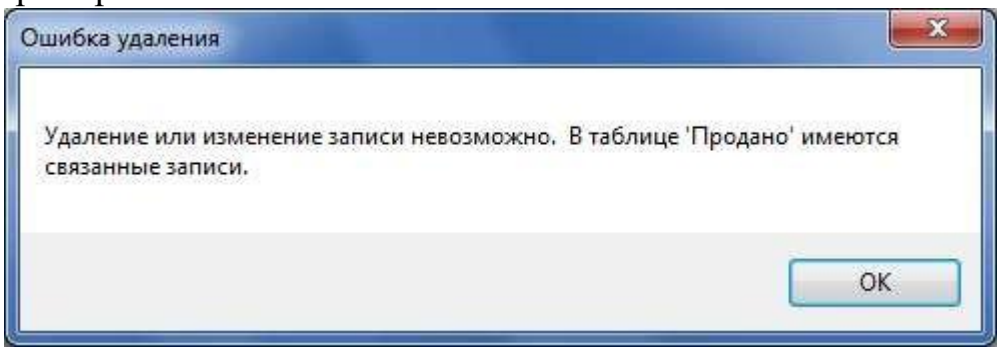
```
void DeleteDish()
        {
            if (String.IsNullOrEmpty(textBoxId_dish.Text))
            {
                MessageBox.Show("Обязательно укажите код блюда данные которого
необходимо удалить", "Внимание", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            int id;
            if (!int.TryParse(textBoxId_dish.Text, out id))
            {
                MessageBox.Show("Некоректное значение кода блюда!", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            string sqlDelete = @"DELETE FROM Блюда WHERE Код_блюда=@Id_dish";
            SqlConnection connection = new
SqlConnection(Properties.Settings.Default.CafeConnectionString);
            connection.Open();
            SqlCommand command = connection.CreateCommand();
            command.CommandText = sqlDelete;
            command.Parameters.AddWithValue("@Id_dish", id);
            try
            {
                command.ExecuteNonQuery();
            }
            catch (Exception err)
            {
                MessageBox.Show(err.Message, "Ошибка удаления");
```

```
            }
            connection.Close();
            buttonSelectDishes_Click(this, EventArgs.Empty);
        }
```

Примечание:

Так как у нас в БД установлена ссылочная целостность без механизма каскадного удаления для **Блюда** и **Продано**, то возможно возникновение исключения при удалении блюда, указанного ранее в **Продано**. Для обработки этого исключения используем блок **try… catch**

Например:



Так как при удалении сотрудника нужна только строка для ввода Кода блюда, то пропишите для кнопки **Удалить данные по блюду с заданным кодом** (**radioButtonDelete_dish**):

```
private void radioButtonDelete_dish_CheckedChanged(object sender, EventArgs e)
        {
            panelDish.Visible = !radioButtonDelete_dish.Checked;
        }
```

Запрограммируем кнопку **Показать список блюд** (**buttonSelectDishes**):

```
private void buttonSelectDishes_Click(object sender, EventArgs e)
        {
            dataGridViewDish.DataSource = FillDataGridView("SELECT * FROM
Блюда");
            DataGridViewImageColumn column =
(DataGridViewImageColumn)dataGridViewDish.Columns["Фото"];
            column.ImageLayout = DataGridViewImageCellLayout.Stretch;

        }
```

Примечание:

в **dataGridViewDish** масштабируется фото.

Для кнопки **Выполнить запрос** (**buttonExecuteDML**) пропишите:

```
private void buttonExecuteDML_Click(object sender, EventArgs e)
```

```
        {
            if (radioButtonInsert_dish.Checked)
                InsertDish();
            else
                if (radioButtonUpdate_dish.Checked)
                    UpdateDish();
                else
                    if (radioButtonDelete_dish.Checked)
                        DeleteDish();
                    else
                        MessageBox.Show("Вы не выбрали действие", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
```

Запустите приложение. Протестируйте работу добавления, редактирования, удаления.

4 Задание на лабораторную работу №4

4.1 Изучить материал лабораторной работы №4.
4.2 Для своей БД необходимо выполнить:
4.2.1 использование запросов на выборку. Обязательно:
- выборка из одной таблицы;
- выборка с условиями;
- выборка из нескольких таблиц;
- агрегирующие (статистические) функции;
- полная запись оператора SELECT(предложения SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY);
- запросы с параметрами с проверкой корректности. Параметры разных типов данных;

4.2.2 использование подзапросов;
4.2.3 использование операторов изменения данных. Обязательно:
- INSERT;
- UPDATE;
- DELETE;
- параметрические запросы с проверкой корректности. Параметры разных типов данных;

4.2.4 протестировать работу полученного приложения.

5 Оформление отчета к лабораторной работе

Оформить отчёт со следующим содержанием:
5.1 Титульный лист.
5.2 Цель работы.
5.3 Постановка задачи.
5.4 Краткая теория и ход выполнения заданий.
5.5 Описание результатов.
Приведите скриншоты полученной программы и листинг программы.
5.6 Заключение (выводы).

Отчет должен быть оформлен согласно требованиям ОС ТУСУР.

6 Контрольные вопросы

6.1 Запись оператора SELECT.
6.2 Приведите примеры операторов, использующихся в предложении WHERE.
6.3 Приведите примеры агрегирующих (статистических) функций, особенности их использования.

6.4 Как можно провести группировку данных в запросе?

6.5 Как можно провести упорядочивание/сортировку результатов запроса?

6.6 Особенности NULL значений в запросах.

6.7 Особенности перекрестного соединения таблиц.

6.8 Особенности внутреннего объединения таблиц.

6.9 Особенности левого объединения таблиц.

6.10 Особенности правого объединения таблиц.

6.11 Как провести объединение результатов выполнения нескольких операторов SELECT.

6.12 Запись оператора INSERT.

6.13 Запись оператора UPDATE.

6.14 Запись оператора DELETE.

6.15 Что такое подзапрос?

6.16 Особенности коррелированного и некоррелированного подзапроса.

6.17 Как можно задать текст SQL запроса в Microsoft Visual Studio?

6.18 Чем характерен динамический (параметрический) запрос?

6.19 Как можно задать нужные значения для параметров в Microsoft Visual Studio?