МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
**«Санкт-петербургский государственный технологический институт
(технический университет)»
(СПбГТИ(ТУ))**

Информационные технологии в управлении персоналом
Учебное пособие для курсовой работы

Квалификация (степень) выпускника
Бакалавр

Форма обучения
заочная

Факультет **экономики и менеджмента**

Санкт-Петербург
2012

Введение

Владение полной и достоверной информацией дает возможность рационально управлять и правильно принимать управленческие решения в любой сфере человеческой деятельности. Особенно это актуально для экономической деятельности.

Современное общество живет в период, характеризующийся небывалым ростом объема информационных потоков. Рыночные отношения предъявляют повышенные требования к достоверности, актуальности и полноте информации.

Развитие вычислительной техники и интегрированных сетей связи, появление современных информационных технологий привело к бурному подъему индустрии переработки информации. Одним из важнейших достижений в этой области, безусловно, являются современные информационные системы управления предприятием и информационные системы для решения других экономических задач.

Экономические информационные системы и системы управления предприятием предназначены для накопления, хранения и выдачи информации по мере ее необходимости в управленческой деятельности, а также для решения конкретных экономических задач. Основой современных информационных систем являются базы данных или базы знаний.

1 Общие сведения

Банк данных разрабатывается и используется для накопления, поиска, обработки и выдачи информации о характеристиках объектов и составляет основу информационной системы.

Банк данных состоит из трех частей: база данных, СУБД (система управления базой данных) и программное пользовательское приложение, которое решает конкретные задачи пользователя и предоставляет ему информацию из базы данных в различных ракурсах.

База данных – это набор данных информационной системы, который строится и работа, с которым обеспечивается специальным пакетом программ, называемым СУБД (система управления базой данных). База данных может содержать, например, адреса клиентов фирмы или наименования товаров. Структура, по которой эта информация записывается в память компьютера, и управление этими данными, например их сортировка, поиск и т.д. обеспечивается с помощью комплекса программ обслуживания, называемого СУБД.

Таким комплексом программ является, например **Microsoft Access.** В **Microsoft Access** данные подготавливаются и идентифицируются в табличной форме, термин "*таблица*" используется как набор данных об объектах одного типа (одного информационного объекта).

СУБД – это комплекс стандартных программ, с помощью которых создают структуру базы данных, наполняют эту базу конкретными данными, и строят программное пользовательское приложение для управления ими и решения конкретных экономических задач.

Программное пользовательское приложение – это комплекс пользовательских программ, созданных на основе конкретных СУБД и решающий конкретные задачи пользователя по предоставлению ему информации из базы данных в различных ракурсах.

2 Анализ предметной области

В информационных системах, создавая **банк данных,** пользователь описывает (моделирует) часть реального мира называемого **предметной областью.**

Сведения о реальном мире представляются в **банках данных** посредством определенного набора понятий. В таком качестве выступают характеристики информационных объектов предметной области, отношения и связи между ними.

Наименьшей единицей описания данных предметной области является **Attribute** (атрибут) –, т.е. определенная, минимальная, желательно неделимая часть информации о некотором информационном объекте (отношении).

Relation (отношение или информационный объект) – определяет принадлежность элементов описания (атрибутов) к одному и тому же описываемому объекту предметной области.

Совокупность элементов описания (атрибутов) объединенных отношением принадлежности к одному и тому же описываемому объекту предметной области называется **записью базы данных.**

Связь – это способ, которым связана информация из различных информационных объектов (отношений) в информационной системе управления.

Основной целью анализа предметной области является выделение основных информационных объектов (отношений) и установление связей между ними.

2.1 Некоторые сведения о фирме

Фирма, для которой мы попытаемся построить информационную систему, занимается торгово-закупочной деятельностью, то в качестве основных информационных объектов мы выберем следующие: **«Кадры», «Клиенты», «Склад», «Поставщики», «Заказы».**

Вся информация о сотрудниках нашей фирмы будет соответствовать информационному отношению **«Кадры»**. Определим атрибуты этого отношения. Какую информацию о сотрудниках мы хотим хранить в нашей информационной системе. Вероятно это фамилии, имена и отчества сотрудников. Информация об их образовании, дате рождения и поступления к нам на фирму, их адрес, паспортные данные. Информации о наличии у них детей, для правильного начисления налогов. К тому же, мы должны знать, в каком отделе у нас работает тот или иной сотрудник, и какую должность он занимает.

Разрабатывая информационную систему управления для нашего предприятия, договоримся, что на данном предприятии будут работать следующие сотрудники: директор фирмы, финансовый директор (главный бухгалтер), бухгалтер (2 штатные единицы), секретарь, уборщица.

Так как эта фирма - торгово-закупочная, то мы будем закупать товары, следовательно, необходим начальник отдела снабжения и торговые агенты (2 штатные единицы). Отвечать за реализацию товаров будет отдел сбыта, который включает начальника отдела сбыта и торговых представителей (3 штатные единицы).

Приобретенные товары будут храниться на складе. На складе нужен кладовщик (2 штатные единицы), а для случая, когда транспортировку осуществляет наша фирма, и водители (2 штатные единицы). Товары надо разгружать и загружать на машины для

перевозки, поэтому предусмотрим 2 штатные единицы грузчиков, а также введем должность начальника административно-хозяйственного отдела.

Структура нашего предприятия предусматривает наличие следующих отделов: администрация (директор и секретарь); бухгалтерия (финансовый директор и бухгалтеры); отдел сбыта (начальник отдела сбыта и торговые представители); отдел снабжения (начальник отдела снабжения и торговые агенты); административно-хозяйственный отдел (начальник отдела, кладовщики, водители грузчики и уборщица).

Товары наша фирма будет закупать у различных поставщиков, информацию о которых нам тоже необходимо знать. Поэтому для хранения этой информации сконструируем отношение **«Поставщики»**. Очевидно, что нам необходимо хранить информацию о названии и статусе предприятия поставщика, его адрес, факс и номер телефона, а также его банковские реквизиты.

Для хранения информации о том, какой товар и в каком количестве храниться на нашем складе, а также от какого поставщика и по какой накладной поступил этот товар, сконструируем отношение **«Склад»**.

Для того, чтобы знать кому мы продаем наш товар, выделим следующий информационный объект **«Клиенты»**. Структура информации в этом отношении будет аналогична структуре информационного объекта **«Поставщики».**

Так как информационные объекты **«Поставщики»** и **«Клиенты»** идентичны по своей структуре, то в дальнейшем при создании таблиц, одну из них можно продублировать и потом внести в нее необходимые изменения.

Ну, а для того, чтобы знать, какие клиенты у нас делают конкретные заказы, когда они оформлены и как скоро будут выполнены, а кроме того кто из сотрудников отвечает за их выполнение, необходим информационный объект **«Заказы».**

2.2 Взаимосвязи между отношениями

Взаимосвязи между информационными объектами предметной области типизированы и бывают следующих видов:

1. *один к одному* - одна запись одного из отношений может быть связана только с одной записью в другом отношении.

2. *один ко многим* - одна запись одного отношения взаимосвязана со многими записями другого отношения.

3. *многие ко многим* - одна и та же запись может входить в отношения со многими другими записями в различных отношениях.

Мы провели анализ предметной области и выделили следующие информационные объекты – отношения: **«Кадры»** – сведения о сотрудниках фирмы, **«Клиенты»** и **«Поставщики»** – сведения о фирмах закупающих у нас и поставляющих нам товары, **«Склад»** – информация о товарах на складе и **«Заказы»** – информация о торговой деятельности нашей фирмы.

Определим, как же связана информация этих информационных объектов.

Поставщики поставляют на наш склад различные товары. При этом один поставщик может поставлять товары разных наименований. Один поставщик привозит на наш склад много наименований товаров. Связь между информационными объектами «**Поставщики»** и «**Склад» - «один ко многим**».

Клиенты покупают товары нашей фирмы, делая заказы. При этом мы будем стараться вести дела таким образом, чтобы каждый клиент сделал в нашей фирме не один заказ, а, купив наши товары однажды, стремился иметь с нами дело и дальше. Будем

рассылать в фирмы клиентов информационные листы о наличии у нас новых товаров. То есть будем считать, что один клиент - это много заказов. Связь между информационными объектами «**Клиенты**» и «**Заказы**» – **«один ко многим».**

Делая заказы, клиенты покупают товары. При этом на один заказ они могут накупить товары разных наименований. Например, если наша фирма продает рыбные консервы, то магазин, являющийся клиентом нашей фирмы, может купить несколько коробок со шпротами, несколько коробок с сайрой, несколько коробок лосося и т.д. То есть на один заказ идет много товаров. В то же время мы покупаем на склад сразу много товара и распродаем его частями по различным заказам. Товар одного наименования может быть частями распродан по нескольким заказам. Например, мы закупили 100 ящиков шпрот и продали 10 из них на один заказ, 15 на другой и т.д. Таким образом, много товаров мы продаем по многим заказам. Связь между информационными объектами «**Склад**» и «**Заказы**» **- «многие ко многим».**

Заказы наших клиентов должны обслуживать сотрудники отдела сбыта. При этом естественно, что один сотрудник обслуживает много заказов. Таким образом, связь между информационными объектами «**Кадры**» и «**Заказы**» – **«один ко многим**».

Мы проанализировали нашу предметную область, выделили основные информационные объекты – отношения, установили связи между ними.

3 Построение информационной модели

В основе большинства используемых методов проектирования информационных экономических систем лежит понятие **информационной модели экономики** – некоторого целенаправленного формализованного отображения существующей системы экономической информации (предметной области) с определением связей, характеризующих систему управления и управляемый объект. Исследование и разработка моделей для создания и использования информационных систем актуальны по следующим соображениям. Информационная модель обеспечивает формализованное представление исследуемых элементов (данных) и их взаимосвязи.

Построение информационных моделей полностью отвечает современным представлениям о разработке баз данных информационных систем, поскольку обеспечивает необходимый этап – построение моделей концептуального уровня базы данных, а также наполнение базы данных необходимой информацией.

Информационно-логическая (инфологическая) – это некоторое формализованное, заданное в явном виде отображение конкретной предметной области. Она отражает объекты экономической информационной системы и связи между ними, но не зависит от методов представления данных в конкретной СУБД.

Концептуальная модель - модель описывает предметную область в виде информационных конструкций, некоторой совокупности информационных объектов, формируя логическую структуру данных предметной области в форме, соответствующей конкретной СУБД, но без указания деталей, связанных с организацией хранения данных и доступа к ним.

Концептуализм – это направление философии, отрицающее реальное существование общих понятий, независимых от единичных вещей, но признающее существование в уме концептов как особых форм познания действительности.

Концептуальный – относящийся к понятиям, выражающий понятия или относящийся к концепции.

Концепция – это определенный способ понимания трактовки каких-либо явлений.

Процесс отображения информации о некоторой предметной области на определенную систему управления базой данных - называется разработкой логической структуры базы данных.

Построение информационной модели базы данных основано на современном подходе к обработке информации, состоящем в том, что структуры данных обладают относительной устойчивостью. Действительно, типы информационных объектов предприятия, для управления которых создается информационная система, если и изменяются во времени, то очень редко. А это означает, что структура данных достаточно стабильна. Поэтому возможно построение информационной базы с постоянной структурой и изменяемыми значениями данных.

Каноническая структура информационной базы, отображающая в структурированном виде, информационную модель предметной области позволяет сформировать логические записи, их элементы и взаимосвязи между ними.

Следующей задачей проектирования информационной системы управления является задача выбора СУБД для организации данных нашей предметной области в памяти компьютера.

В настоящие время многие известные фирмы, занимающиеся разработкой программных продуктов, предлагают на рынок объектно-ориентированные СУБД.

Самыми распространенными являются **Microsoft Access**, **SQL Server, Delphi, Oracle, Informix.**

Мы для своей работы выберем СУБД **Microsoft Access**.

Microsoft Access это функционально полная реляционная объектно-ориентированная СУБД.

В ней предусмотрены все необходимые средства для определения и обработки данных, а так же для управления ими при работе с большими объемами информации.

Microsoft Access является частью пакета прикладных программ **Microsoft Office,** одного из самых распространенных программных продуктов на современном этапе использования программного обеспечения.

Кроме того СУБД **Microsoft Access** обладает следующими достоинствами:

- нет ограничения на объем памяти;
- осуществляется контроль над вводом информации;
- имеется возможность связей DDE и OLE;
- можно задать форматы представления информации для вывода на экран или печать;
- возможность запомнить информацию, представленную в виде звука или рисунка;
- воспринимает множество самых разнообразных данных организованных в другой СУБД, как и более ранних разработок под DOS: Paradox, Btrive Fox Pro, DBASEI – IVплюс+, так и самых современныхMS SQL Server, Oracle, Informix;
- возможен импорт и экспорт данных других приложений.

Чаще всего в крупных корпорациях **Microsoft Access** используется в общей информационной системе как пользовательская среда для обработки данных.

Соответствующим образом подготовленные менеджеры могут применять **Microsoft Access** для создания собственных запросов, диаграмм и отчетов, а в информационных отделах **Microsoft Access** используется как средство разработки конечного пользовательского интерфейса приложения для обработки информации.

Учитывая все выше изложенное, выберем СУБД **Microsoft Access** для построения нашей информационной системы управления.

СУБД **Microsoft Access** – это объектно-ориентированная СУБД, построенная на основе реляционной модели организации данных. Реляционные модели были первоначально сформулированы доктором фирмы IBM Э.Ф.Коддом в 1970 году. Эти идеи оказали широкое влияние на технологию Банков Данных во всех аспектах, а также на технологию разработки искусственного интеллекта (Баз Знаний) и обработку текстов на естественных языках.

Математическим термином для обозначения таблицы в реляционных СУБД является **отношение** (Relation).

При работе с реляционными моделями используется как математическая терминология, так и неформальная, исторически принятая в сфере обработки данных (см. таблицу 1).

Таблица 1 – Терминология СУБД

Формальная терминология	Неформальная терминология
Отношение	Таблица
Картеж	Запись
Атрибут	Столбец или Поле

Создавая базу данных в СУБД, построенных по реляционным моделям, мы имеем дело с тремя аспектами работы с данными. Со структурой данных, с целостностью данных и с манипулированием данных.

Структура данных это логическая организация данных в базе данных.

Под целостностью данных понимается безошибочность и точность информации хранящейся в базе данных.

Манипулирование данных это действия, совершаемые над данными в базе данных.

Эти три аспекта и отражают основные процедуры процесса накопления данных (хранение, актуализация, извлечение).

В СУБД, построенных по реляционной модели, записи базы данных хранятся в таблицах, каждая из которых содержит данные об одном информационном объекте (отношении).

Все значения данных в базах данных, построенных на основе реляционных СУБД, являются атомарными, т.е. в каждой таблице на пересечении строки и столбца всегда имеется в точности одно значение данных и никогда не бывает множество значений.

Полное информационное содержание базы данных представляется в виде явных значений данных. Рассмотрим несколько способов проектирования базы данных для экономической информационной системы.

3.1 Нормализация данных

В реляционной теории нет универсальных рецептов для проектирования надежной и эффективной в использовании базы данных. Разработчик имеет право выбирать различные инструменты и методы проектирования. Некоторые полагаются исключительно на интуицию и здравый смысл, другие используют различные вспомогательные средства, порой довольно изощренные. Однако при всем разнообразии подходов все же есть некоторые каноны, нарушение которых весьма отрицательно скажется как при проектировании базы данных, так и при ее эксплуатации. Так, например, весьма актуальной является проблема нормализации баз данных. Пренебрежение нормализацией делает структуру базы данных запутанной, а саму базу – ненадежной в работе. Однако перед тем как приступить к ее изучению, нам придется разобраться с некоторыми важными понятиями реляционной теории. Одно из них – понятие ключа.

3.1.1 Понятие ключа

Мы уже знаем, что отношение состоит из кортежей (или, более простыми словами, таблица состоит из строк, хотя это не вполне корректно). Естественно, что функции системы управления базами данных не исчерпываются простым накоплением данных; мы хотим также манипулировать содержимым БД. Для этого необходимо иметь возможность каким-то образом отличать кортежи друг от друга.

Пусть R-отношение с атрибутами A_1, A_2, …, A_n. Говорят, что множество атрибутов К = (A_i, Aj,…$A_к$) отношения R является первичным ключом отношения R, тогда и только тогда, когда удовлетворяются два независимых от времени условия: уникальность и минимальность.

Первое условие (уникальность) указывает на то, что в произвольно заданный момент времени никакие два различных картежа отношения R не имеют одного и того же значения К = (A_i, Aj,…,$A_к$).

Второе условие (минимальность) говорит о том, что ни один из атрибутов в К=(A_i, Aj,…, $A_к$) не может быть исключен из него (из К) без нарушения условия уникальности.

Каждое отношение обладает по крайне мере одним возможным ключом поскольку, по меньшей мере, комбинация всех атрибутов удовлетворяет условиям уникальности.

Один произвольно выбранный возможный ключ для данного отношения называется его **первичным ключом**, а остальные возможные ключи называются **альтернативными.**

Помимо первичных и альтернативных ключей идентифицирующих данное отношение есть еще и внешний ключ.

В общем случае **внешний ключ** это атрибут или комбинация атрибутов одного отношения R'', значение которого обязательно должно совпадать со значением первичного ключа некоторого другого отношения R', причем внешние первичные ключи должны быть определены на одних и тех же **доменах**. Внешние ключи в неявном виде связывают отношения.

Неформальным языком: ключом отношения называется набор атрибутов отношения, однозначно определяющий кортеж. Или еще проще: ключ – это набор столбцов таблицы, значения которых уникально определяют строку.

3.1.2 Домены

Наименьшей единицей данных в реляционной модели является отдельное значение данных. Такие значения рассматриваются как атомарные и неразложимые.

Доменом называют множество подобных значений одного и того же типа данных. Домены представляют собой пулы значений, из которых берутся фактические значения, появляющиеся в атрибутах.

Смысл доменов: если значения двух атрибутов берутся из одного домена, то их можно сравнивать, соединять и объединять, если из разных, то любые действия над ними лишены смысла.

Домены по своей природе являются понятием концептуальным и могут храниться или не храниться в базе данных как фактическое множество значений.

Домены специфицируются как часть определения базы данных, поэтому определение каждого атрибута должно включать ссылку на соответствующий домен, во избежание двусмысленности.

Отношения на доменах $Д_1$, $Д_2$,…, $Д_n$ состоит из заголовка и тела.

В свою очередь:

Заголовок состоит из такого фиксированного множества атрибутов, что существует взаимно однозначное соответствие между атрибутами A_1, A_2, …, A_n и определяющими их доменами $Д_i$(i=1,n).

Тело состоит из меняющегося во времени множества картежей (записей), где картеж состоит из множества пар атрибутов – значений (A_i : V_i) (i=1,n). Причем по одной такой паре для каждого атрибута A_i в заголовке.

Для любой заданной пары атрибут – значение (A_i : V_i), V_i является значением из единственного домена, с которым связан атрибут A_i.

Таким образом, все отношения соответствуют приведенному определению отношения (Relation).

Строго говоря, изображая отношение в виде таблицы, мы просто используем удобный способ представления.

Таблица и отношение в действительности не одно и тоже.

При изображении таблицы мы упорядочили расположение столбцов-атрибутов и строк – картежей, хотя отношение – это математическое множество, а множество в математике не обладает каким – либо упорядочением.

Значение n – число атрибутов в отношении – называется **степенью отношения.**

Отношение степени 1 – называется унарным;

отношение степени 2 – называется бинарным;

отношение степени 3 – называется тринарным;

отношение степени n – называется эн-арным.

3.1.3 Функциональные зависимости

Перед тем, как приступить к изучению понятия – нормализации, нам предстоит познакомиться с весьма важным понятием функциональной зависимости (ФЗ).

Как неоднократно упоминалось ранее, хранение данных в реляционных базах данных преследует две цели: понизить избыточность данных и повысить их достоверность. Для этого необходимо иметь возможность накладывать некоторые ограничения, как на сами хранимые данные, так и на взаимосвязи и взаимозависимости между ними, т.к. в противном случае вместо базы данных мы получим просто свалку, лишенную всякой структуры. Обычно эти ограничения вытекают из анализа предметной области, которая моделируется нашей информационной системой. Одним из средств формализации информации, полученной в результате такого анализа, являются зависимости между данными.

Существует несколько разновидностей зависимостей, которые рассматриваются реляционной теорией: F-зависимости, MV-зависимости и J-зависимости. В настоящий момент наибольший интерес среди них для нас представляют функциональные зависимости, или F-зависимости.

Рассмотрим следующий пример. Допустим, у нас имеется таблица, в которую внесены данные о заказах. Каждая строка таблицы Заказы представляет собой сведения об одном заказе и указывает, какой торговый представитель отвечает за данный заказ в данный день.

Предположим, что не каждое сочетание данных является допустимым. На совокупность данных накладывается ряд ограничений:

- каждый заказ имеет определенную дату выполнения.
- данный торговый представитель в данный день отвечает только за один заказ.
- для данного заказа назначается только один торговый представитель.

Рассмотрев исходные данные с учетом этих ограничений, мы можем сделать ряд выводов, которые показаны в таблице 2.

Таблица 2 – Исходные данные

Торговый представитель	Номер заказа	Дата выполнения заказа
Иванов	1	16 мая 2005 г.
Петров	2	16 мая 2005 г.
Сидоров	3	16 мая 2005 г.
Петров	4	17 мая 2005 г.
Сидоров	5	17 мая 2005 г.
Иванов	6	17 мая 2005 г.

– ДАТА ВЫПОЛНЕНИЯ ЗАКАЗА функционально зависит от НОМЕРА ЗАКАЗА.

– ЗАКАЗ функционально зависит от {ТОРГОВОГО ПРЕДСТАВИТЕЛЯ и ДАТЫ ВЫПОЛНЕНИЯ}.

– ТОРГОВЫЙ ПРЕДСТАВИТЕЛЬ функционально зависит от {ЗАКАЗА и ДАТЫ).

Итак, если значения кортежа на некотором множестве атрибутов единственным образом определяют значения на другом множестве атрибутов, говорят, что имеет место функциональная зависимость или, короче, F-зависимость. Понятие функциональной зависимости тесно связано с понятием ключа.

3.1.4 Целостность данных

Как уже неоднократно упоминалось в данном цикле, база данных – это не просто хранилище фактов (с этой задачей способны справиться и незатейливые плоские файлы). При проектировании баз данных упор в первую очередь делается на достоверность и непротиворечивость хранимых данных, причем эти свойства не должны утрачиваться в процессе работы с данными, т.е. после многочисленных изменений, удалений и дополнений данных по отношению к первоначальному состоянию БД. Для поддержания БД в устойчивом состоянии используется ряд механизмов, которые получили обобщенное название средств поддержки целостности. Эти механизмы применяются как статически (на этапе проектирования БД), так и динамически (в процессе работы с БД). Приведение структуры БД в соответствие этим ограничениям – это и есть нормализация.

В целом суть этих ограничений весьма проста: каждый факт, хранимый в БД, должен храниться один-единственный раз, поскольку дублирование может привести (и на практике непременно приводит, как только проект приобретает реальную сложность) к несогласованности между копиями одной и той же информации. Следует избегать любых неоднозначностей, а также избыточности хранимой информации. С этими требованиями трудно не согласиться, выглядят они вполне разумно. Но трудно следовать им на практике.

Для устранения всяческих неопределенностей и неоднозначностей традиционно используется формальный язык математики. Воспользуемся им и мы, тем более что для реляционных БД, построенных на солидном математическом фундаменте, формализмы достаточно хорошо проработаны.

Целостность реляционной модели определяется двумя общими правилами:

– **целостность по сущности.** Здесь не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе базового отношения принимал неопределенные значения. **Базовым отношением** называют независимое именованное отношение. Базовые отношения соответствуют сущностям в реальном мире, а следовательно отличимы, т.е. имеют уникальную идентификацию. В реальной модели функцию уникальной идентификации выполняют первичные ключи, таким образом, ситуация, когда первичный ключ принимает неопределенное значение, является противоречивой, и говорит о том, что некоторая сущность не обладает индивидуальностью. Отсюда и название – целостность по сущности;

– **целостность по ссылкам**. Если базовое отношение R'' включает некоторый внешний ключ FK соответствующий некоторому первичному ключу PK какого-либо базового отношения R', то каждое значение FK в R'' должно быть равно значению PK в некотором кортеже R', либо быть полностью неопределенным.

3.1.5 Формы нормализации

Под нормализацией отношения подразумевается процесс приведения отношения к одной из так называемых нормальных форм (или в дальнейшем НФ).

Итак, наша цель – приведение отношений к НФ. Следует заметить, что в процессе нормализации постоянно встречается ситуация, когда отношение приходится разложить на несколько других отношений. Поэтому более корректно было бы говорить о нормализации не отдельных отношений, а всей их совокупности в БД. Однако в примерах для простоты будем по возможности иметь дело с отдельными отношениями, если это не приведет к неясностям. Нормализация реальных баз данных – гораздо более трудоемкий процесс.

Впрочем, не все обстоит так плохо. Нормализация может не представлять такую уж проблему, если БД проектируется сразу по определенным канонам. Другими словами, можно сначала сделать БД как попало, а потом нормализовать ее, или же с самого начала строить ее по правилам, чтобы в дальнейшем не пришлось переделывать. Сейчас мы пойдем первым путем, т.е. возьмем в качестве примеров никуда не годные БД и попытаемся привести их в порядок. При этом будем иметь в виду, что при профессиональном подходе к проектированию БД используется одна из хорошо проработанных технологий.

Итак, что же представляет собой процесс нормализации? Фактически это не что иное, как последовательное преобразование исходной БД к НФ, при этом каждая следующая НФ обязательно включает в себя предыдущую (что, собственно, и позволяет разбить процесс на этапы и производить его однократно, не возвращаясь к предыдущим этапам). Всего в реляционной теории насчитывается шесть НФ: 1-я НФ (обычно обозначается также 1НФ); 2НФ; 3НФ; НФ Бойса-Кодда (НФБК); 4НФ;5НФ.

Рассмотрим подробнее три первые НФ.

3.1.6 Первая нормальная форма

Схема отношения R находится в 1НФ, если значения в dom(A) являются атомарными для каждого атрибута A в R. Другими словами, каждый атрибут отношения должен хранить одно-единственное значение и не являться ни списком, ни множеством значений.

Атомарными предполагает, что в каждой таблице на пересечении строки и столбца всегда имеется в точности одно значение данных, и никогда не бывает множества значений;

Следует заметить, что, несмотря на внешнюю строгость данного определения, однозначно определить понятие атомарности зачастую оказывается довольно затруднительно, если заранее неизвестны семантика атрибута и его роль в обработке хранимых данных. Атрибут, который является атомарным в одном приложении, может оказаться составным в другом.

Простейший пример: в нашей базе данных в таблице КАДРЫ, хранящей личные сведения о сотрудниках, имеется атрибут «домашний-адрес», в котором адрес хранится в формате: улица, дом[/корпус], [квартира] (следуя общепринятой нотации, здесь в квадратных скобках указаны опциональные фрагменты адреса, которые могут отсутствовать). В данном случае адрес хранится в виде единой текстовой строки, поскольку маловероятно, чтобы потребовалось выбрать сотрудников, скажем, по номеру

квартиры. Таким образом, в таком контексте адрес является атомарным понятием, и его деление на составные части не имеет смысла, т.к. только внесет в базу данных излишнюю громоздкость. Однако тот же адрес для приложения, предназначенного для сортировки почты в почтовом отделении, атомарным не является, поскольку желательно сгруппировать конверты в отдельные стопки по улицам, так как каждую улицу обслуживает свой почтальон. Кроме того, с целью оптимизации перемещений почтальона в пределах улицы, каждую стопку желательно отсортировать по номерам домов, чтобы сделать возможным разнести почту за один проход по улице без возврата. Итак, очевидно, что в отрыве от контекста затруднительно дать строгое определение атомарности, за исключением самых простых случаев (например, домен натуральных чисел). В большинстве случаев необходимо располагать сведениями о предметной области, а также о том, каким образом планируется обрабатывать хранимые в базе данных данные.

Приведение отношения к 1НФ – довольно простая операция. Мы должны просмотреть схему отношения и разделить составные атрибуты на различные строки/столбцы. Возможно, эту операцию придется повторить несколько раз до тех пор, пока каждый из атрибутов не станет атомарным (с учетом сказанного в предыдущем абзаце).

В БД имеется таблица поставщиков, в которой хранятся следующие сведения, представленные в таблице 3.

Таблица 3 – Сведения о поставщиках

Название фирмы	Город	Адрес	Контактные лица
ОАО Поршневой з-д	Владимир	Ул. 2-я Кольцевая, 17	Иванов И.И., зам. дир., тел (3254)76-15-95 Петров П.П., нач. отд. сбыта, тел (3254)76-15-35
ООО «Вымпел»	Курск	Ул. Гоголя, 25	Сидоров С.С., директор, тел. (7634)66-65-38
ЧП «Альфа»	Владимир	Ул. Пушкинская, 37	Васильев В.В., директор, тел (3254)74-57-45

Очевидно, что в данном случае атрибут «контактные лица» не является атомарным, поскольку в нем попадаются списки из нескольких лиц. Разделим эти кортежи таким образом, чтобы каждый кортеж содержал данные только об одном лице (таблица 4).

Таблица 4 – Сведения о поставщиках с разбивкой по контактным лицам

Название фирмы	Город	Адрес	Контактные лица
ОАО Поршневой з-д	Владимир	Ул. 2-я Кольцевая, 17	Иванов И.И., зам. дир., тел (3254)76-15-95
ОАО Поршневой з-д	Владимир	Ул. 2-я Кольцевая, 17	Петров П.П., нач. отд. сбыта, тел (3254)76-15-35
ООО «Вымпел»	Курск	Ул. Гоголя, 25	Сидоров С.С., директор, тел. (7634)66-65-38
ЧП «Альфа»	Владимир	Ул. Пушкинская, 37	Васильев В.В., директор, тел (3254)74-57-45

Несколько лучше, хотя при ближайшем рассмотрении оказывается, что атрибут «контактные лица» снова может быть назван атомарным лишь с натяжкой, поскольку содержит разнородные данные, хотя и об одном лице. К тому же нам может быть потребуется выбирать список фирм имеющих одинаковый статус.

Разобьем наше отношение еще на несколько атрибутов. В частности выделим в отдельный атрибут статус предприятия, а также в отдельные атрибуты ФИО, должность и телефон контактного лица (таблица 5).

Таблица 5 – Сведения о поставщиках с учетом атомарности

Статус фирмы	Название фирмы	Город	Адрес	ФИО	Должн. конт. лица	Телеф. конт. лица
ОАО	Поршневой з-д	Владимир	ул. 2-я Кольцевая, 17	Иванов И.И.	зам. дир	(3254) 761595
ОАО	Поршневой з-д	Владимир	ул. 2-я Кольцевая, 17	Петров П.П.	нач. отд сбыта	(3254) 761535
ООО	«Вымпел»	Курск	ул. Гоголя, 25	Сидоров С.С.	директор	(7634) 666538
ЧП	«Альфа»	Владимир	ул. Пушкинская, 37	Васильев В.В.	директор	(3254) 745745

Теперь можем считать, что каждое значение каждого из атрибутов нашего отношения является атомарным. Следовательно, отношение находится в 1НФ.

3.1.7 Вторая нормальная форма

Теперь наше отношение выглядит несколько корректнее. Однако даже беглый взгляд на него говорит о том, что оно пока еще далеко от совершенства. Очевидно, что повторяющиеся значения, которых немало в предыдущей таблице, являются потенциальным источником проблем.

Во-первых, при вводе их значений легко ошибиться. Например, достаточно изменить всего одну букву в графе «Адрес», и формально это будет уже совершенно

другой адрес, не имеющий ничего общего с первым. Найти подобные опечатки в объемной таблице – задача не простая.

Во-вторых, название улицы может измениться, хоть это происходит и не так часто (тем не менее, пренебрегать такой возможностью не стоит).

В-третьих, нельзя пренебрегать и эффективностью хранения информации. Из нашей таблицы можно как минимум дважды узнать адрес поршневого завода, хотя нам хватило бы и одного раза.

Таким образом, борьба с избыточностью данных выгодна со всех сторон – как со стороны повышения удобства пользования данными и поддержания их в целостном виде, так и со стороны эффективности обработки и хранения данных, аппаратными средствами сервера баз данных. Именно на устранение этой избыточности и направлена дальнейшая нормализация отношений.

Однако перед тем, как дать определение 2НФ, нам необходимо познакомиться с понятием полной и частичной функциональной зависимостей. Схема отношения R находится во 2НФ относительно множества функциональных зависимостей F, если она находится в 1НФ и каждый неключевой атрибут полностью зависит от каждого ключа для R. Другими словами, отношение находится во 2НФ, если оно находится в 1НФ, и при этом все неключевые атрибуты зависят только от ключа целиком, а не от какой-то его части.

Вернемся к нашей таблице, которая представляет пример нашего отношения, приведенного к 1НФ. Предположим, что при постановке задачи заказчик сообщил нам, что в пределах каждого города наименование предприятия является уникальным, но в разных городах названия могут совпадать. Таким образом, предприятие характеризуется составным ключом «Наим+Город». Посмотрим на наше отношение с точки зрения того, что мы только что узнали про 2НФ.

Очевидно, что телефонный код города зависит исключительно от самого города и никак не связан с названием предприятия. Отсюда и один из источников избыточных данных – сколько раз в таблице встречается строка, содержащая сведения о контактном лице какого-либо предприятия, находящегося в данном городе, столько раз и повторяется информация о коде города. Чтобы устранить эту избыточность, нам придется разбить наше отношение на несколько (в данном случае – 2, хотя для более сложных отношений их может быть и гораздо больше) (см. таблицы 6 и 7).

Таблица 6 – Первая часть отношения «Поставщики»

Статус фирмы	Название фирмы	Город	Адрес	ФИО	Должн. конт. лица	Телеф. конт. лица
ОАО	Поршневой з-д	Владимир	ул. 2-я Кольцевая, 17	Иванов И.И.	зам. дир	761595
ОАО	Поршневой з-д	Владимир	ул. 2-я Кольцевая, 17	Петров П.П.	нач. отд сбыта	761535
ООО	«Вымпел»	Курск	ул. Гоголя, 25	Сидоров С.С.	директор	666538
ЧП	«Альфа»	Владимир	ул. Пушкинская, 37	Васильев В.В.	директор	745745

Таблица 7 – Вторая часть отношения «Поставщики»

Название города	Код города
Владимир	3254
Курск	7634

Итак, мы избавились от частичной зависимости атрибута «Код города» от составного ключа, переместив коды городов в отдельное отношение с ключом «Город». Таким образом, теперь мы получили два отношения, каждое из которых находится во 2НФ.

Поскольку в данном случае речь идет о частичной зависимости атрибутов от ключа, то все вышесказанное относится исключительно к отношениям с составным ключом. Отношение с простым, или атомарным, ключом (состоящим из единственного атрибута), приведенное к 1НФ, находится во 2НФ по определению и в данном этапе нормализации не нуждается.

Несмотря на предпринятые усилия, таблица «Поставщики» определенно все еще содержит изрядную избыточность – достаточно взглянуть на повторяющиеся значения в столбцах. Значит, процесс нормализации еще не завершен, и пора переходить к его следующей стадии.

3.1.8 Третья нормальная форма

Для того чтобы формально определить 3НФ, нам придется предварительно познакомиться с понятием транзитивной зависимости атрибутов, от которой мы и попытаемся избавиться на этом этапе.

Обозначим: R – схема отношения, X – подмножество R, A – атрибут в R, F – множество функциональных зависимостей. A называется транзитивно зависимым от X в R, если существует такое Y, являющееся подмножеством R, что: X->Y; ~(Y->X); Y->A; ~ (A Є XY)

Теперь можно дать собственно определение 3НФ: Схема отношения R находится в 3НФ относительно множества функциональных зависимостей F, если она находится в 1НФ и ни один из непервичных атрибутов в R не является транзитивно зависимым от ключа для R.

То есть, проще говоря, чтобы привести отношение к 3НФ, необходимо устранить функциональные зависимости между неключевыми атрибутами отношения. Другими словами, факты, хранимые в таблице, должны зависеть только от ключа.

В реляционной теории имеется лемма, которая гласит, что любая схема отношения, находящаяся в 3НФ относительно F, находится в 2НФ относительно F.

В нашем случае присутствует функциональная зависимость между атрибутами «Ф.И.О.», «Должность» и «Тел». Очевидно, что на предприятии некий человек занимает определенную должность и располагает определенным рабочим телефоном. Чтобы избавиться от данной функциональной зависимости, проведем декомпозицию таблицы на две таблицы «Поставщики» на две таблицы. Первая (таблица 8) из них хранит факты, относящиеся непосредственно к самому предприятию: Вторая (таблица 9) хранит факты, относящиеся к конкретному лицу, исполняющему некоторые обязанности на данном предприятии:

Данные в таблицах 8 и 9 представляет собой базу данных о «Поставщиках» приведенную к 3НФ.

Таблица 8 – Первая часть декомпозиции

Статус фирмы	Название фирмы	Город	Адрес
ОАО	Поршневой з-д	Владимир	Ул. 2-я Кольцевая, 17
ОАО	Поршневой з-д	Владимир	Ул. 2-я Кольцевая, 17
ООО	«Вымпел»	Курск	Ул. Гоголя, 25
ЧП	«Альфа»	Владимир	Ул. Пушкинская, 37

Таблица 9 – Вторая часть декомпозиции

Название фирмы	Город	ФИО	Должн. конт. лица	Телеф. конт. лица
Поршневой з-д	Владимир	Иванов И.И.	зам. дир	761595
Поршневой з-д	Владимир	Петров П.П..	нач. отд сбыта	761535
«Вымпел»	Курск	Сидоров С.С.	директор	666538
«Альфа»	Владимир	Васильев В.В.	директор	745745

3.2 Реальный подход к проектированию

Надеюсь, у Вас сложилось определенное представление о процессе нормализации отношений и о его роли при проектировании баз данных. Должна заметить, что для простоты изложения я выбрала пример базы данных хранящей информацию только о поставщиках и специально предварительно их «испортила», чтобы в дальнейшем подвергнуть нормализации. На практике дело обычно обстоит не так.

Во-первых, кроме совсем уж зеленых новичков, вряд ли кому придет в голову начинать проектирование базы данных с таблиц, нарушающих 1НФ (возможно, за исключением случаев, когда нужно занести в БД данные, подготовленные табличным процессором вроде Excel).

С составными ключами на практике тоже доводится встречаться не так уж часто. Обычно составной ключ подменяют так называемым суррогатным ключом, т.е. ключом искусственного происхождения, который обычно генерируется самой СУБД, которая и гарантирует его уникальность. Составной ключ в этом случае становится альтернативным ключом, а суррогатный – первичным. Исключением, пожалуй, является ситуация с идентифицирующими зависимостями, когда имеет место миграция первичного ключа родительской сущности в составной первичный ключ дочерней. В нашем случае это ключ отношения **«Заказ-товар».**

Таким образом, на практике приводить отношения к 1НФ и 2НФ приходится не столь уж часто. Скорее вполне достаточно оказывается проглядеть отношения, чтобы убедиться, что они не нарушают эти нормальные формы.

С 3НФ ситуация несколько сложнее, так как транзитивная зависимость не всегда так явно бросается в глаза. Поэтому нарушение 3НФ – наиболее частая проблема, с которой приходится иметь дело. Впрочем, нормализация – это фактически исправление огрехов,

допущенных при проектировании БД. Как известно, частенько оказывается проще не допускать этих огрехов с самого начала, нежели спроектировать БД кое-как, а потом оптимизировать (чем мы, собственно, и занимались на протяжении всего пункта 3.1.). Неоценимую помощь в проектировании баз данных оказывают соответствующие методологии их предлагается большое количество. Но отложим теорию и спроектируем небольшую, но вполне реальную базу данных для решения нашей задачи.

Так как фирма, для которой мы попытаемся построить информационную систему управления, занимается торгово-закупочной деятельностью, то в качестве основных информационных объектов (отношений) мы выберем следующие: **«Кадры», «Клиенты», «Склад», «Поставщики», «Заказы».**

Вся информация о сотрудниках нашей фирмы будет соответствовать информационному отношению **«Кадры».**

Товары наша фирма будет закупать у различных поставщиков, информацию о которых нам тоже необходимо знать. Поэтому для хранения этой информации сконструируем отношение **«Поставщики»**. Очевидно, что нам необходимо хранить информацию о названии и статусе предприятия поставщика, его адрес, факс и номер телефона, а также его банковские реквизиты.

Для хранения информации о том, какой товар и в каком количестве храниться на нашем складе, а также от какого поставщика и по какой накладной поступил этот товар, сконструируем отношение **«Склад»**.

Для того, чтобы знать кому мы продаем наш товар, выделим следующий информационный объект **«Клиенты»**. Структура информации в этом отношении будет аналогична структуре информационного объекта **«Поставщики».**

Ну, а для того, чтобы знать, какие клиенты у нас делают конкретные заказы, когда они оформлены и как скоро будут выполнены, а кроме того кто из сотрудников отвечает за их выполнение, необходим информационный объект **«Заказы».**

Определим, как же связана информация этих информационных объектов (отношений).

Поставщики поставляют на наш склад различные товары. При этом один поставщик может поставлять товары разных наименований. Один поставщик привозит на наш склад много наименований товаров. Связь между информационными объектами «**Поставщики»** и «**Склад» - «один ко многим**».

Клиенты покупают товары нашей фирмы, делая заказы. При этом мы будем стараться вести дела таким образом, чтобы каждый клиент сделал в нашей фирме не один заказ, а, купив наши товары однажды, стремился иметь с нами дело и дальше. Будем рассылать в фирмы клиентов информационные листы о наличии у нас новых товаров. То есть будем считать, что один клиент - это много заказов. Связь между информационными объектами «**Клиенты»** и «**Заказы» – «один ко многим».**

Заказы наших клиентов должны обслуживать сотрудники отдела сбыта. При этом естественно, что один сотрудник обслуживает много заказов. Таким образом, связь между информационными объектами «**Кадры»** и «**Заказы» – «один ко многим**».

Делая заказы, клиенты покупают товары. При этом на один заказ они могут накупить товары разных наименований. Например, если наша фирма продает рыбные консервы, то магазин, являющийся клиентом нашей фирмы, может купить несколько коробок со шпротами, несколько коробок с сайрой, несколько коробок лосося и т.д. То есть на один заказ идет много товаров. В то же время мы покупаем на склад сразу много товара и распродаем его частями по различным заказам. Товар одного наименования может быть частями распродан по нескольким заказам. Например, мы закупили 100 ящиков шпрот и продали 10 из них на один заказ, 15 на другой и т.д. Таким образом, много товаров мы

продаем по многим заказам. Связь между информационными объектами «**Склад»** и «**Заказы» - «многие ко многим».**

Мы проанализировали нашу предметную область, выделили основные информационные объекты – отношения, установили связи между ними.

Кроме того, следует помнить, что в базах данных, построенных на основе реляционной СУБД, не допускается связь между отношениями «многие ко многим». Поэтому вводятся специальные таблицы – связки, которые разбивают связь «многие ко многим» на связи «один ко многим» и «многие к одному». В нашем случае следует добавить таблицу **«Заказ-товар**», в которой будут содержаться данные о том, на какие конкретные заказы, какие конкретные товары, в каком количестве проданы. Этот метод представления данных единственный имеющийся в распоряжении реляционной СУБД, т.к. реляционная СУБД не имеет указателей.

Информационная модель данных для предприятия, занимающегося торгово-закупочной деятельностью, показана на рисунке 1.

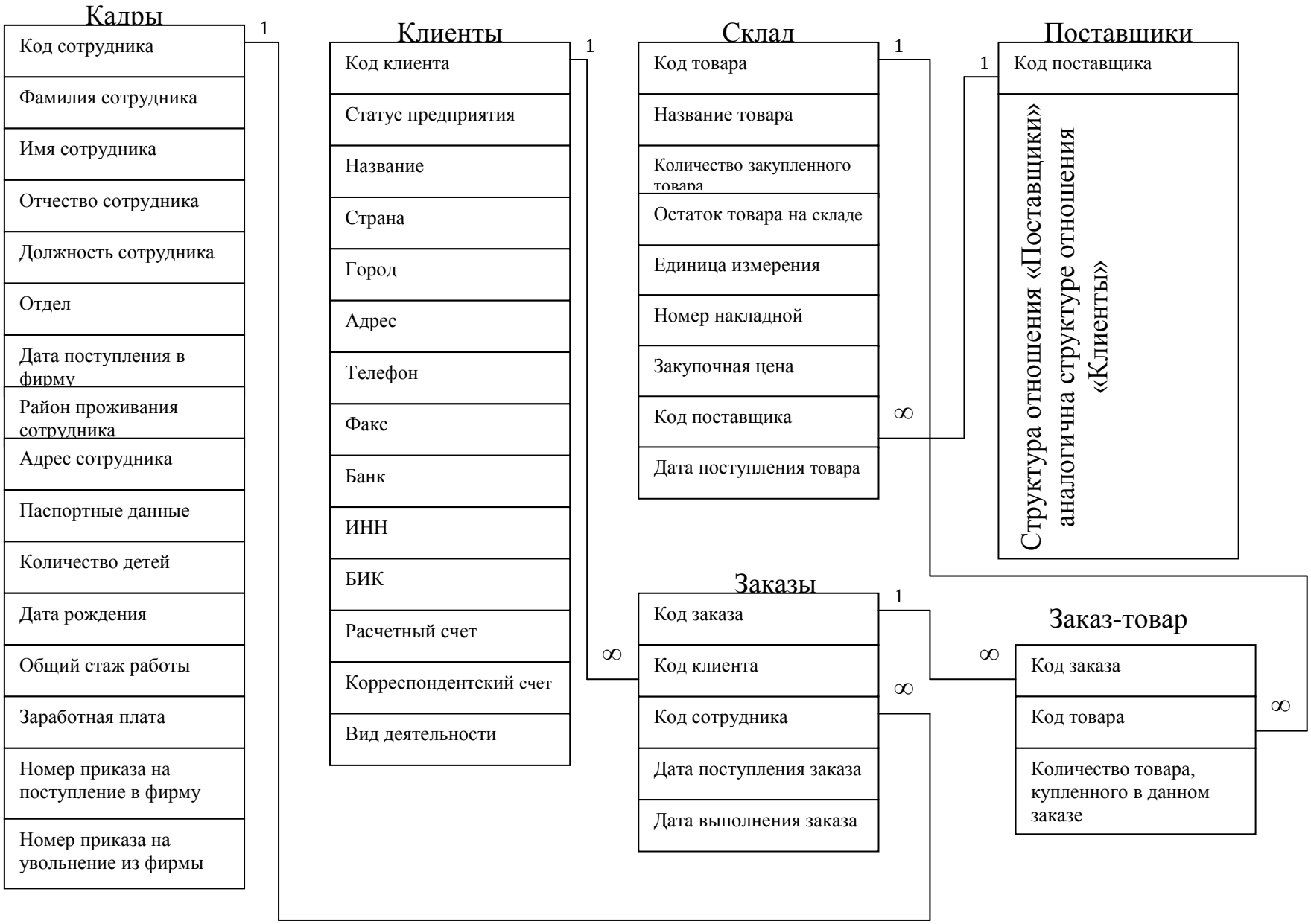

Рисунок 1 - Информационная модель данных исследуемой предметной области

4 Физическое проектирование базы данных (конструирование таблиц)

Решая задачу физического проектирования базы данных, пользователь осуществляет выбор рациональной структуры хранения данных и методов доступа к ним, исходя из набора методов и средств, которые предоставляются ему СУБД, на основе которой эта база данных строится.

Под физической структурой (организацией) базы данных понимается физическое представление данных и расположение их на запоминающих устройствах. Если логическая структура базы данных - структура для пользователя, то физическая - структура базы данных для ЭВМ. Физическая структура определяет, тип и свойства данных, которые будут записаны в память компьютера.

По сути дела физическое проектирование базы данных подразумевает конструирование таблиц в СУБД. Для решения стоящей перед нами задачи, разработка системы управления предприятием, занимающимся торгово-закупочной деятельностью, мы будем конструировать таблицы в СУБД **Microsoft Access.**

СУБД **Microsoft Access** представляет собой систему управления базами данных, в состав которой входят таблицы, запросы, формы, отчеты, макросы и модули как самостоятельные объекты, хранящиеся в *общем файле базы данных* на жестком диске или любом другом носителе данных. Благодаря этому создание связанных объектов и проверка ссылочной целостности данных значительно облегчена.

При проектировании готового к использованию приложения разработчику приходится проделать огромную работу по созданию отдельных его компонентов, таких как создание таблиц, экранных форм, запросов, отчетов, макросов и модулей. Для автоматизации этого процесса в состав пакета **Microsoft Access** включен ряд специализированных программ, решающих подобные задачи. Одно направление этих программ получило название **Конструктор**, а другое - **Мастер.**

Конструктор предоставляет в распоряжение пользователя ряд инструментальных средств, с помощью которых можно быстро и просто составить требуемую конструкцию: таблицу, форму, запрос, отчет, макрос и модуль.

Мастер также помогает проектировать, но осуществляет это другим способом. Присваивая программе такое название, разработчики хотели подчеркнуть, что создание нового объекта с помощью программы **Мастер,** происходит без приложения особых усилий со стороны пользователя. **Мастер** во время работы задает пользователю ряд вопросов, на которые он должен ответить, и на основе полученных ответов строит вполне законченный объект.

Создание базы данных в СУБД **Microsoft Access** начинается с определения имени файла с базой данных (стандартное имя db1.mdb) и места на внешне носителе информации, в котором этот файл будет записан. Студенты нашего факультета должны создать файлы со своим именем, на диске D:, своих рабочих станций, в папках с номером своей группы.

База данных заполняется информацией, структурированной в соответствии с предписаниями разработчика. Созданную в **Microsoft Access** базу данных можно наполнять объектами различного типа - таблицами, формами, отчетами и т.д. и выполнять операции с ними. Таблицы являются основой для записи информации, содержащейся в базе данных. Без таблицы нельзя спроектировать форму, на базе таблиц составляются

запросы и отчеты. При проектировании базы данных следует определить, какая именно информация должна входить в базу данных и в каких таблицах она будет располагаться.

Каждая строка таблицы содержит одну **запись** данных. Совокупность элементов описания (полей), объединенных отношением принадлежности к одному и тому же описываемому объекту предметной области, называется **записью базы данных.**

Количество полей в **записи**, и их тип определяет пользователь при проектировании базы.

Приступим к созданию таблиц в СУБД **Microsoft Access** для нашей информационной системы управления. Вид окна **Microsoft Access** в режиме конструирования таблиц показан на рисунках 2 и3.

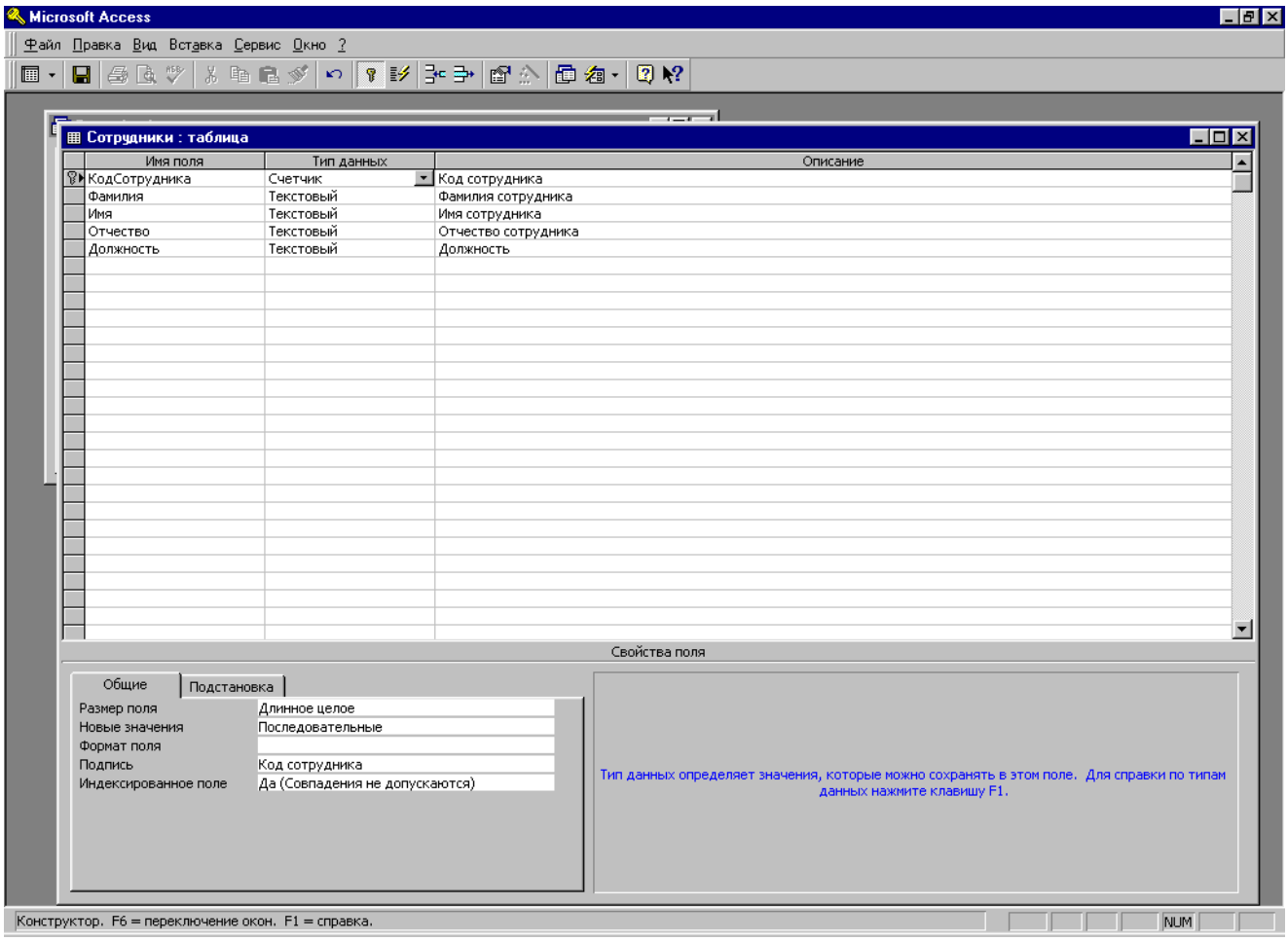

Рисунок 2 - Вид окна Microsoft Access в режиме конструирования таблиц (общие свойства поля)

Как видно из рисунка окно диалога для конструирования таблиц в СУБД **Microsoft Access** разбито на две части. В строках верхней части окна пользователь определяет имя поля, тип данных для записи содержимого этого поля в ЭВМ и может дать подробное словесное описание содержимого каждого поля.

В нижней части окна задаются свойства (характеристики) полей, составляющих запись базы, т.е. вводится спецификация записи. В правой нижней части окна **Access** выдает справочную информацию о допустимых на каждом этапе проектирования действиях.

Символом ►отмечена активная строка таблицы – строка, в которую происходит ввод информации. Переход к следующей ячейке строки осуществляется нажатием клавиши **Tab**, либо щелчком мыши в соответствующей ячейке.

Информация для заполнения колонок **Поле** и **Описание** вводится с клавиатуры.

Каждая строка спецификации определяет характеристики одного поля записи. В колонке **Имя поля** задается имя поля. Оно может иметь длину до 64 символов и может содержать буквы кириллического алфавита, пробелы и специальные символы, за исключением точек, восклицательных знаков и угловых скобок. Единственным ограничением является запрет на наличие в одной таблице двух полей с одинаковыми именами. Последнее ограничение связано с тем, что **Microsoft Access** идентифицирует поле по его имени.

При заполнении колонки **Тип данных**, в правой части столбца появляется кнопка раскрывающегося списка ▼, щелкнув по которой, вы можете выбрать из списка доступных типов необходимый вам **тип данных**. Разумеется, что, выбирая **тип данных,** необходимо проанализировать конкретные данные, которые будут записываться в поле таблицы. Например, если мы будем записывать фамилии сотрудников, то понятно, что тип данных выберем **«текстовый»**, а если закупочную цену товара, то тип данных **«денежный»** и т.п.

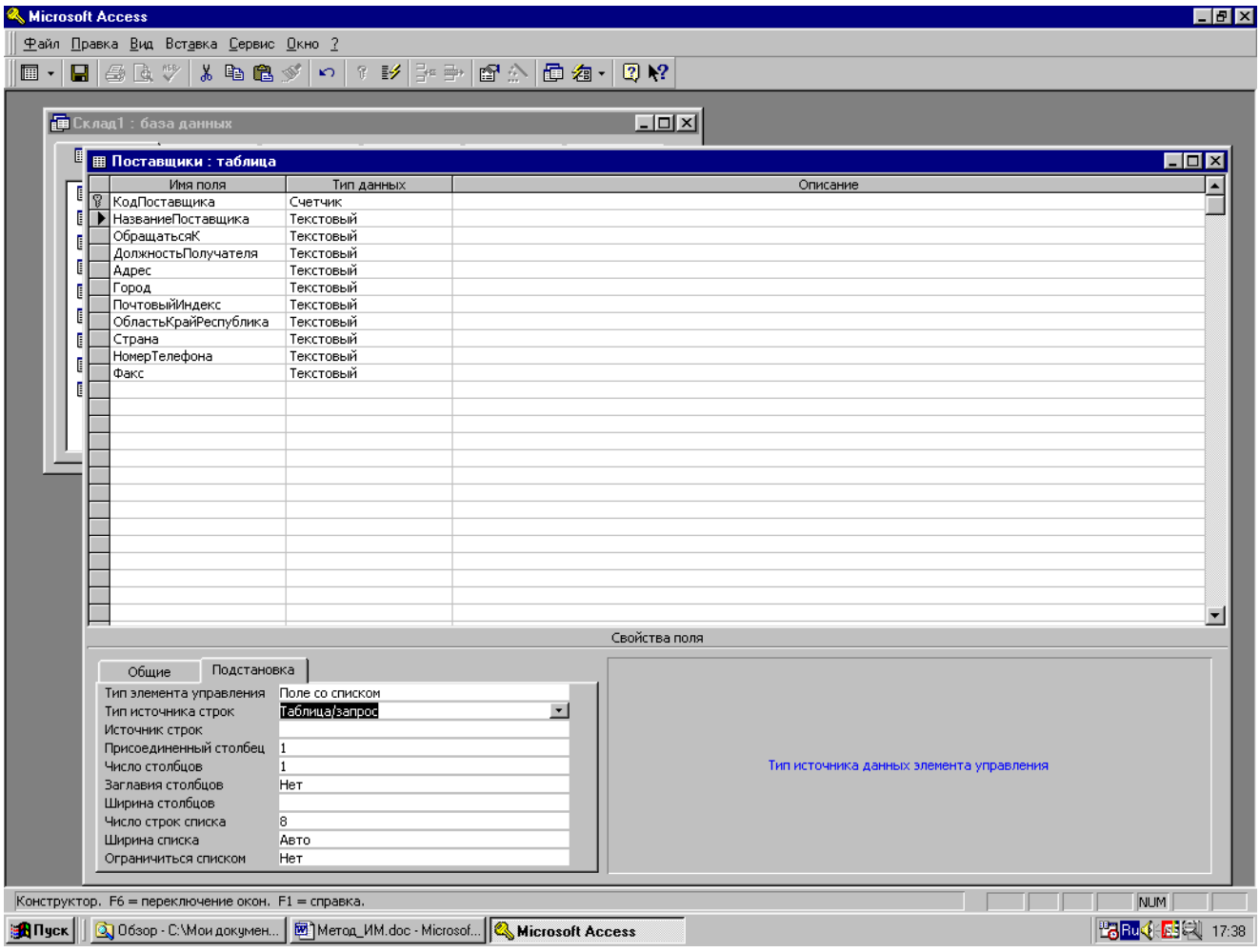

Рисунок 3 - Вид окна Microsoft Access в режиме конструирования таблиц (подстановка для поля)

Список типов данных показан в таблице 10.

Доступность свойств полей в зависимости от типов данных показана в таблице11.

Свойство **«Размер поля»** определяет максимальный размер данных, которые могут сохраняться в полях с типом данных **«Текстовый» и «Числовой».**

Свойство **«Формат поля»** задается для представления данных при выводе на экран. Символы формата определены для разных типов данных в таблицах Help. При организации формата отдельные значения можно выводить различными цветами. Название цвета записывается в квадратных скобках и обязательно используется с другими символами формата.

Свойство **«Маска ввода»** используется, когда вводимые в поля данные имеют предварительно определенную структуру.

Свойство **«Маска ввода»** выполняет следующие функции:

— ограничивает ввод за счет установленной фиксированной длины и типа символа;

— автоматически при вводе вставляет фиксированные символы;

— выполняет простые действия над вводимыми данными (например: преобразует вводимые символы к верхнему регистру).

Таблица 10 - Список типов данных

Тип данных	Использование	Размер
Текстовый	Для алфавитно-цифровых данных	До 255 байт
MEMO	Для алфавитно-цифровых данных	До 64 Кб
Числовой	Числовые данные (данные, над которыми производятся математические вычисления)	1, 2, 4 или 8 байтов
Дата/время	Используется для даты/времени	8 байтов
Денежный	Данные о денежных суммах, хранятся в памяти компьютера с 4 знаками после запятой	8 байтов
Счетчик	Уникальное длинное целое, генерируемое Access при создании каждой новой записи. Генерация может проводиться методом случайных чисел или методом инкремента (+1)	4 байта
Логический	Для записи данных, в которых допускается только 2 значения	1 бит
Объект OLE	Картинки, диаграммы и другие графические объекты, созданные в приложениях Windows	До 1 Гб

Таблица 11 - Доступность свойств полей в зависимости от типов данных

Свойства \ Тип данных	текстовый	МЕМО	числовой	Дата/время	денежный	счетчик	логический	OLE
Размер поля	+		+					
Формат	+	+	+	+	+	+	+	
Маска ввода	+		+	+	+			
Число десятичных знаков			+		+			
Подпись поля	+	+	+	+	+	+	+	+
Значение по умолчанию	+	+	+	+	+		+	
Условие на значение	+	+	+	+	+		+	
Сообщение об ошибке	+	+	+	+	+		+	
Обязательное поле	+	+	+	+	+		+	+
Пустые строки	+	+						
Индексированное поле	+		+	+	+	+	+	

Создавая маску ввода, соблюдают следующие правила:

— маска ввода состоит из трех частей, разделенных точкой с запятой;

— первая часть маски состоит из символов маски и постоянных символов и является обязательной (символы маски см. в таблице Help);

— вторая и третья часть маски не являются обязательными, их можно пропускать;

— вторая часть маски указывает: требуется ли сохранять постоянные символы маски в поле. Если постоянные символы должны быть включены, то вторая часть маски содержит «0», если сохраняются только символы, вводимые пользователем, то – «1»;

— если вторая часть маски опускается, а третья часть имеет место быть, то первая и третья часть разделяются двумя точками с запятой;

— третьим компонентом маски ввода является символ-указатель заполняемых при вводе позиций. По умолчанию для этой цели используется символ «подчеркивание».

Маску ввода можно создавать самим, а можно воспользоваться мастером по созданию масок. Как и любой другой мастер, он предлагает на выбор набор стандартных масок (для ввода даты и пароля).

Свойство **«Число десятичных знаков».** Устанавливается для определения выводимого количества десятичных знаков. Следует иметь ввиду, что при выборе типа данных и формата – «денежный» автоматически не устанавливается число знаков = 2, поэтому если требуется учитывать копейки, то необходимо заменить значение Auto на значение «2».

Свойство **«Подпись поля».** Данное свойство выражается именем, фразой или описанием, которое используется по умолчанию в качестве заголовка столбца или управляющей метки поля. Благодаря этому свойству можно использовать короткие имена

полей, что очень важно, так как имена полей используются при написании программных модулей.

Свойство **«Значение по умолчанию».** Это значение будет присвоено полю каждый раз при добавлении новой записи. Значение по умолчанию используется для автоматического ввода значений в поле. Следует помнить, что Access вставляет значение по умолчанию только один раз для каждой новой записи в тот момент, когда она впервые добавляется в таблицу. При удалении содержимого поля Access оставляет поле пустым, второй раз значение по умолчанию вставлено не будет.

Свойство **«Условие на значение».** Это условие используется для контроля вводимой в поле информации. Оно оценивает содержание вводимого поля, а количество и тип вводимых символов контролируется маской ввода. Условие на значение задается выражением, которое в общем случае состоит из операторов сравнения и значений, используемых для сравнения операндов. Если выражение не содержит оператора сравнения, то Access будет использовать оператор «равно». Полный список операторов сравнения показан в таблице 12.

В операторе LIKE можно использовать следующие символы шаблона:

? - заменяет любой произвольный символ, но один;

* - заменяет любое количество произвольных символов;

- заменяет одну любую цифру;

[] – определяет диапазон символов;

! – определяет исключение из диапазона.

Например: если надо выбрать товары, название которых начинается на букву «А», то пишем LIKE (А*).

Свойство **«Сообщение об ошибке»** - сообщение, выведенное в окне диалога при ошибочном вводе данных пользователем, т. е., если не соблюдено условие на значение или маска ввода. Если сообщение непонятно пользователю, то он не сможет правильно вводить значение, не поймет, как исправить свою ошибку. Например: для поля «номер телефона» в таблицах «клиенты» и «поставщики» можно задать сообщение об ошибке: «если код города состоит менее чем из 6 цифр, то сначала вводим пробелы, а затем цифры кода».

Свойство **«Обязательное поле»** - это свойство заставляет пользователя ввести данные, и не сохранит запись до тех пор, пока значение в это поле не будет введено.

Таблица 12 - Операторы сравнения

операторы	**назначение**
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
=	Равно
<>	Не равно
OR (или) AND (и)	Для использования нескольких сравнений, разделенных логическими операторами
IN	Проверяет на равенство любому значению из списка. Операндом является список, заключенный в круглые скобки. IN («начальник отдела сбыта», «торговый агент», «торговый менеджер».)
BETWEEN	Проверяет, что значение поля находится внутри заданного диапазона, при этом верхняя и нижняя граница диапазона фактически разделяются логическим оператором AND. BETWEEN 50 AND 100 BETWEEN #01.10.03#AND#14.11.03#
LIKE	Проверяет соответствие текстового или MEMO поля заданному шаблону символов.

Свойство **«Пустые строки»** - если мы задаем значение «пустые строки» - да, то определяется возможность хранить в базе данных значение **Null** для обозначения в этом поле неизвестного значения.

Следует помнить:

— пустые значения **Null** не могут быть равны никаким другим значениям, даже также пустым;

— пустое значение **Null** нельзя объединить или по нему связывать;

— пустые значения **Null** не учитываются в групповых функциях, например, таких как «sum».

Для того, чтобы избежать ошибок можно задавать значение текстового и MEMO полей, равного пустой строке «пробел». И это означает, что значение поля не известно, но оно не пустое. Такие значения можно объединять, по ним можно связывать, и они равны друг другу. Ни в коем случае нельзя путать пустые строки и пустые значения.

В том случае, когда мы точно знаем список значений в конструируемом нами поле, мы можем построить для ввода данных в это поле элемент управления **«поле со списком».** В этом списке мы перечислим все известные нам значения этого поля, и лицу, заполняющему таблицу данными, не надо будет каждый раз набирать значение этого поля с клавиатуры. Ему достаточно будет выбрать значение из поля со списком.

Для организации «поля со списком» переходят на корешок «Подстановка» (см. рисунок 3) и в строке «тип элемента управления» выбирают значение «поле со списком». На вкладке **«Подстановка»** появится список дополнительных свойств, которые можно определить, конструируя элемент управления.

В строке **«тип источника строк»,** определяют источник данных для поля со списком. Типом источника строк может быть таблица или запрос, список значений или список полей.

В строке **«источник строк»** перечислим значения, которыми будет заполняться поле, для которого конструируется элемент управления. Значения поля вписываются в соответствии с правилами Microsoft Access (если данные текстовые, то в кавычках, если числовые, то без них, если даты, то в символах # и т.д.) и отделяются друг от друга точкой с запятой.

Свойство **«присоединенный столбец»** определяет значение, какого столбца будет значением элемента управления, в том случае, когда элемент управления состоит из нескольких столбцов.

Свойство **«число столбцов»** определяет число столбцов, выводящихся в поле со списком.

Свойство **«заглавия столбцов»** используют для вывода строки заголовков столбцов в элементе управления «поле со списком». Это свойство желательно определять, когда «поле со списком» имеет несколько столбцов.

Свойство **«ширина столбцов»** содержит значение, указывающее ширину каждого столбца в дюймах или в сантиметрах, в зависимости от системы измерений (американской или метрической), указанной в поле «Система единиц» вкладки «Числа» диалогового окна «Язык и стандарты панели управления Windows». По умолчанию устанавливается либо 1 дюйм, то есть 2,54 сантиметра. Значение свойства «ширина столбцов» должно лежать в диапазоне от 0 до 22 дюймов (55,87 см) для каждого столбца «поля со списком». Пустое значение ширины столбца задается, если перед разделителем списка в соответствующей позиции не указано числовое значение. В этом случае Microsoft Access автоматически задает стандартное значение ширины столбца, зависящее от числа столбцов и ширины поля со списком.

Свойство **«число строк списка»** задает максимальное число строк, выводящихся в раскрывающемся списке элемента управления «поле со списком». Значением свойства «число строк списка» является целое число, определяющее максимальное число выводящихся строк. По умолчанию задается значение 8. Диапазон значений свойства число строк списка от 1 до 255. Если полное число строк списка превышает указанное в свойстве число строк списка, раскрывающийся список выводится с вертикальной полосой прокрутки.

Свойство **«ширина списка»** задает ширину раскрывающегося списка в поле со списком. Оставим его значение в положении «Авто».

Свойство **«ограничиться списком»** определяется двумя значениями: «да» и «нет». Значение «да» определяет, что введенное значение, не совпадающее с элементом списка, не принимается. Значение «нет»**,** допускает ввод любого текста, удовлетворяющего свойству «условие на значение» и не превышающее размер данного поля, определенный свойством «размер поля».

Сконструируем таблицы данных для нашей системы управления.

4.1 Конструирование таблиц для информационной системы управления предприятием

Начнем создание таблиц для хранения информации нашей системы управления с создания таблицы **«Кадры»**, которая будет содержать сведения о сотрудниках нашей фирмы.

Рассмотрим, как конструируются поля таблицы. Как было показано в информационной модели данных, первое поле таблицы **«Кадры»** - это поле **«Код сотрудника»**. В это поле мы будем записывать индивидуальный числовой код сотрудников нашей фирмы. При этом не имеет значения, какой конкретный код имеет тот или иной сотрудник, но важно, чтобы два сотрудника не имели одинаковых кодов. Выбирая тип данных для этого поля **«счетчик»** мы решим эту задачу.

В самой первой строке в колонке **Имя поля** напишем *Код сотрудника*. Тип данных для него выберем – **Счетчик**. Описание – *Код сотрудника фирмы*. После того, как мы выбрали тип данных, СУБД определила, какие свойства должны быть определены для данного поля. Мы видим, что по умолчанию определен размер поля, как **длинное целое –** это означает, что для хранения данных, в этом поле выделено 4 байта (диапазон чисел, которые могут быть записаны в это поле от-2 147 483 648 до 2 147 483 647). Способ генерации новых значений для поля с типом данных **счетчик** определен по умолчанию, как **последовательный**. Т.е. каждое следующее число будет больше предыдущего на единицу. Формат вывода данных для этого поля можно выбрать **основной** из списка форматов. В сущности нас мало интересует формат вывода данных для этого поля. Мы практически не будем работать с данными этого поля в формах. Свойство **подпись** – определяет подпись этого поля в форме. Договоримся, что во всех полях, конструируемых нами таблиц, мы всегда будем определять это свойство. Его содержимое может совпадать с описанием поля. Чтобы запретить ввод в поле повторяющихся значений, определим наше поле как индексированное, (совпадения не допускаются). Поле код сотрудника будет в этой таблице у нас ключевым полем, содержащим уникальные для каждой записи значения. Чтобы сделать поле **ключевым** нужно в верхней части окна конструирования таблицы выделить строку этого поля и нажать кнопку  .

Как уже упоминалось выше, в базах данных, построенных в реляционных СУБД, понятие таблица идентично понятию отношение. Строго говоря, это не совсем так. При изображении таблицы мы упорядочили расположение столбцов - атрибутов и строк – картежей, хотя отношение – это математическое множество, а множество в математике не обладает каким – либо упорядочением. Поскольку отношение это множество, а множества по определению не содержат совпадающих элементов, поэтому никакие два картежа одного отношения не могут в произвольно заданный момент времени быть дубликатами друг друга. Отношение должно иметь **первичный ключ**, идентифицирующий это отношение.

Помимо первичных ключей, идентифицирующих данное отношение, могут задаваться еще и внешние ключи.

В общем случае **внешний ключ** это атрибут или комбинация атрибутов одного отношения, значение которого обязательно должно совпадать со значением первичного ключа некоторого другого отношения. Внешние ключи в неявном виде связывают отношения.

Приступим к конструированию следующего поля таблицы **Кадры** *фамилия сотрудника*. **Имя поля** можно задать одним словом - *фамилия,* а вот в описание этого

поля и в свойство **подпись** запишем подробно - *фамилия сотрудника фирмы*. Тип данных для этого поля выберем **текстовый**, т.к. совершенно ясно, что фамилии сотрудников – это текстовые данные. **Размер поля** определим, анализируя данные, которые мы будем записывать в это поле. **Размер поля** должен быть таким, чтобы *фамилия сотрудника*, состоящая из максимального количества букв, могла быть записана в это поле. Вряд ли можно встретить фамилию, в которой будет больше 15 букв, поэтому определим размер этого поля 15.

Формат для вывода данных этого поля можно не задавать. А вот **маску ввода** желательно определить. Прежде чем мы это сделаем, давайте выясним, что это такое маска и зачем она нужна. С помощью **свойства маска** ввода можно облегчить пользователю ввод данных в элемент управления поле редактирования, а также предотвратить некоторые ошибки ввода данных в таблицы. Например, если мы вводим в память компьютера дату, скажем 14.09.02, то должны набрать с клавиатуры не только цифры этой даты, но и точки, которые повторяются при вводе любой даты. С помощью маски ввода можно организовать ввод текстовых констант (точек) во время ввода данных автоматически. Но мы собираемся вводить маску ввода не для даты, а для поля *фамилия сотрудника*. Зачем же в этом поле нужна маска? Представьте себе такую ситуацию: оператор вводит данные в созданную нами систему управления и, вводя фамилию, нечаянно нажимает не клавишу Shift для ввода заглавной буквы, а клавишу Caps Lock, которая включает режим Caps Lock. В результате фамилия введена неправильно, и данные придется вводить сначала. Маской ввода мы можем исправить эту ошибку пользователя автоматически. К тому же, фамилии в поле фамилия сотрудника должны быть вписаны, начиная с первой позиции в этом поле. За этим тоже будет следить составленная нами маска ввода.

Маска ввода, которую определяет пользователь, создавая информационную систему в СУБД **Microsoft Access,** состоит из трех частей, разделенных точкой с запятой. Первая часть состоит из символов маски, используемых для ее определения в СУБД **Microsoft Access**. **Таблицу символов маски** смотрите в справочнике **Microsoft Access**. Вторая часть определяет режим занесения в таблицу строковых констант, добавляемых к символам, вводящимся пользователем. Введенный в данный компонент символ 0 указывает, что постоянные символы (например, точки в дате, скобки и дефисы в маске ввода телефонных номеров) сохраняются вместе с введенными пользователем символами; значение 1 или пустое значение данного компонента указывает, что сохраняются только символы, введенные пользователем. Третья часть определяет символ, используемый для изображения пустых позиций в маске ввода, в которые помещаются вводящиеся пользователем символы. В этом компоненте можно указать любой символ ANSI; но удобнее работать, если это будет пробел, его необходимо заключить в кавычки “ “.

Маска ввода для поля **фамилия сотрудника** будет выглядеть следующим образом: **>L<CCCCCCCCCCCCCC;;” “**. Как следует из **таблицы символов маски**, символ маски > преобразует все символы, стоящие за ним, к верхнему регистру, а нам надо, чтобы первая буква фамилии была заглавная. Символ маски **L** требует, чтобы в первой позиции набираемого текста обязательно была буква, а не пробел и не цифра. Символ маски < преобразует все символы, стоящие за ним, к нижнему регистру, что нам и требуется, т.к. в фамилии только первая буква заглавная. Остальные четырнадцать символов маски, определенные английской буквой **С**, позволят в оставшиеся позиции поля фамилия вводить как буквы, так и пробелы. Вторая часть в созданной нами маске отсутствует, т.к. в водимых в это поле данных нет строковых констант. Поэтому мы ее опускаем и ставим в маске подряд две точки с запятой. Последнюю, третью часть маски, определяем пробелом.

Заполняя строку свойств**, значение по умолчанию** будем помнить о том, что таблица кадры будет в дальнейшем заполняться нами с учетом вакантных должностей. Т.е. обязательным полем в таблице **кадры** у нас будет поле *должность*, а в поле *фамилия* может быть и **пустая строка**, если эта должность вакантная. Поставим **значение по умолчанию** “ “, т.е. определим, что если должность вакантная, то в поле фамилия стоит “ “.

Свойства условие на значение и сообщение об ошибке нам при конструировании этого поля заполнять не надо. (Договоримся, что в дальнейшем, разбирая особенности конструирования полей таблиц, мы будем останавливаться только на тех свойствах, значение которых надо обязательно определить.)

В строке **обязательное поле,** определим значение - **нет**, как мы договорились ранее, это поле необязательное. А в строке **пустые строки** значение **да.**

В строке **индексированное поле** определим значение **да (совпадения допускаются**). Индекс, создаваемый по какому-либо полю, ускоряет выполнение запросов, в которых используются индексированные поля, и операции сортировки и группировки. Например, по полю "Фамилия" часто выполняется поиск в таблице "Кадры", следует создать индекс для этого поля. А так как в одной организации могут работать, например два Ивановых, то и совпадение данных в этих полях допускается.

Поля *имя сотрудника* и *отчество сотрудника* сконструируйте сами, по аналогии с полем *фамилия сотрудника***.**

Для поля *должность сотрудника,* также как и для полей, *фамилия, имя* и *отчество сотрудника* выбираем **тип данных - текстовый**. А вот **размер поля** мы можем задать точно, так как точно знаем штатное расписание нашей фирмы. Перечень должностей сотрудников, которые будут у нас работать описан в пункте 2.1. методических указаний. Там же дан список отделов. Самое длинное данное, которое мы должны записать в поле *должность* – это должность начальника административно-хозяйственного отдела. Размер этой должности 47 символов. Таким образом, **размер поля** для поля *должность* **47.**

Свойства **формат поля, маска ввода, значение по умолчанию, условие на значение и сообщение об ошибке** для этого поля заполнять не будем. Установим в положение **да** свойство **обязательное поле**. Как мы договорились ранее это поле должно быть заполнено, даже если должность вакантна. А свойство **пустые строки,** определим значением, **нет**. Индексировать данные этого поля нам тоже совсем незачем, поэтому свойство, **индексированное поле** определим значением, **нет**. Так как мы точно знаем перечень должностей на нашем предприятии, мы можем построить для ввода данных в это поле элемент управление **поле со списком**. В этом списке мы перечислим все должности нашего предприятия, и лицу, заполняющему таблицу данными, не надо будет каждый раз вписывать должность. Ему достаточно будет выбрать название должности из поля со списком. Для организации **поля со списком** перейдем на корешок **Подстановка** и в строке **тип элемента управления** выберем значение **поле со списком**. На вкладке **Подстановка** выведутся все дополнительные свойства, необходимые для определения конфигурации элемента управления. В строке **тип источника строк**, определяющей источник данных для поля со списком, выберем значение **список значений**. Источником данных в этом случае будет являться список значений, указанный в свойстве **источник строк**. Поэтому в строке источник строк перечислим должности сотрудников нашего предприятия, написав их в кавычках, и отделив друг от друга точкой с запятой, как этого требуют правила **Microsoft Access**. Свойство **присоединенный столбец** определяет значение, какого столбца становится значением элемента управления. Определение этого свойства имеет значение, когда **поле со списком** состоит из нескольких столбцов. У нас оно состоит из одного столбца, поэтому нам определять его не надо, и мы оставляем его

значение по умолчанию, то есть **1**. Свойство **число столбцов** определяет число столбцов, выводящихся в поле со списком. У нас оно равно **1**. Свойство **заглавия столбцов** используют для вывода строки заголовков столбцов в **полях со списком**. В данном случае это совсем не обязательно. Это свойство желательно определять, когда **поле со списком** имеет несколько столбцов. Установим значение этого свойства, **нет**. Свойство **ширина столбцов** содержит значение, указывающее ширину каждого столбца в дюймах или в сантиметрах, в зависимости от системы измерений (американской или метрической), указанной в поле **Система единиц** вкладки **Числа** диалогового окна **Язык и стандарты панели управления Windows**. По умолчанию устанавливается либо 1 дюйм, либо 2,54 сантиметра. Значение свойства **ширина столбцов** должно лежать в диапазоне от 0 до 22 дюймов (55,87 см) для каждого столбца поля со списком. Пустое значение ширины столбца задается, если перед разделителем списка в соответствующей позиции не указано числовое значение. В этом случае **Microsoft Access** автоматически задает стандартное значение ширины столбца, зависящее от числа столбцов и ширины поля со списком. Оставим значение свойства **ширина столбцов** пустым. Свойство **число строк списка** задает максимальное число строк, выводящихся в раскрывающемся списке **поля со списком**. Значением свойства **число строк списка** является целое число, определяющее максимальное число выводящихся строк. По умолчанию задается значение 8. Воспользуемся им при конструировании поля *должность*. Диапазон значений свойства **число строк списка** от 1 до 255. Если полное число строк списка превышает указанное в свойстве **число строк списка**, раскрывающийся список выводится с вертикальной полосой прокрутки. Свойство **ширина списка** задает ширину раскрывающегося списка в поле со списком. Оставим его значение в положении **Авто**. Свойство **ограничиться списком** определяется двумя значениями: **да** и **нет**. Значение **да** определяет, что введенное значение, не совпадающее с элементом списка, не принимается. Значение **нет,** допускает ввод любого текста, удовлетворяющего свойству **условие на значение** и не превышающее размер данного поля, определенный свойством **размер поля.**

Поле *отдел* в таблице **кадры** конструируется аналогично сконструированному полю *должность***.**

Сконструируем поле *дата поступления* на работу в нашу фирму. Так как данные этого поля будут записаны в память компьютера в виде календарных дат, то выбираем **тип данных** для этого поля **дата/время**. Определение свойств этого поля начнем с определения его формата. Свойство **формат поля** позволяет указать использование встроенных или специальных числовых форматов для полей даты/времени. Давайте рассмотрим следующий пример. Дату 14.09.02 можно записать просто цифрами, разделенными точками, можно полностью 14 сентября 2002 года, можно сокращенно 14-сен-02 и т. д. Значение свойства **формат поля** для даты определяет, в каком виде будет выведена на экран монитора дата. Установим значение этого свойства **длинный формат даты**. Как уже упоминалось выше, чтобы облегчить ввод данных в память компьютера можно сконструировать **маску ввода**. Определим ее следующим образом: **99\.99\.99;0;" "**. Символы 9 в маске определяют, что в данных вместо этих символов могут быть цифры или пробел. Символ \ указывает, что следующий символ следует воспринимать как постоянный (в нашем случае этот символ точка). Точки с запятой разделяют три части маски. Значение ноль во второй части указывает на то, что строковые константы (в нашем случае точки) будут записаны в память компьютера вместе с данными. А третья часть маски, как уже указывалось выше, определяет, что символы вводимых данных будут введены в поле управления вместо символа пробел. Остальные свойства для поля дата поступления можно не определять. Согласиться с предлагаемыми СУБД **Microsoft Access**

по умолчанию. Обратите внимание, что мы не определяли свойство **размер поля** для поля *дата поступления*. Это связано с тем, что в СУБД **Microsoft Access** для **типа данных дата/время** определен стандартный **размер поля 8 байтов**. Поэтому, определяя **маску ввода** для этого поля, мы не могли во второй ее части поставить значение **1**. В этом случае строковые константы точки не были бы запомнены в память компьютера, и размер вводимого данного был бы только 6 байтов, что недопустимо для полей имеющих **тип данных дата/время.**

Подумайте, как сконструировать поле *дата рождения* сотрудника и чем свойства поля *дата рождения* сотрудника отличаются от свойств поля *дата поступления* сотрудника на работу.

Поле *район проживания*. Мы выделяем район проживания сотрудника из его адреса в отдельное поле, учитывая требования налоговой инспекции. Конструируя это поле, мы естественно выберем **тип данных текстовый** и построим элемент управления для ввода информации **поле со списком**. Размер поля определим по максимальному значению символов в данных списка.

Поле *адрес сотрудника*. Это поле с **текстовым типом данных, размер** которого определяется приблизительно, например, **70 символов**. Попробуйте сами построить **маску ввода** для этого поля.

Поле *паспортные данные*. Также как и поле, *адрес сотрудника* - это поле с **текстовым типом данных**, размер которого определяется приблизительно. Так как данные, которые будут записываться в это поле, требуют больше места в памяти, чем данные поля *адрес сотрудника*, то и **размер** этого поля будет больше, например, **100 символов**. Обязательно постройте **маску ввода** для этого поля, внимательно изучив данные вашего паспорта.

Поле *количество детей*. Ясно, данные этого поля целые числа, для записи которых вполне хватит одного байта (напоминаю, что в один байт можно записать целые числа из диапазона от –128 до +127). Поэтому **тип данных** для этого поля **числовой, размером** в **один байт.**

Поле *заработная плата*. Очевидно, что **тип данных** для этого поля **денежный.** Выбрав его, построим **формат** этого **поля** для вывода числовых данных. Давайте договоримся, что не все сотрудники нашей фирмы будут получать фиксированные оклады, часть из них будут работать сдельно. Определим для этого поля свойство **значение по умолчанию 0,00** рублей и будем считать, что если в это поле не проставлен оклад данного сотрудника, то оплата его труда сдельная (т.е. если значение данных в поле 0,00 рублей, то оплата труда сдельная). Построим специальный формат для этого поля. Специальные числовые форматы могут включать в себя от одного до четырех разделов, отделенных друг от друга точкой с запятой. Каждый формат содержит спецификацию для различных типов числовых данных. Первый раздел включает формат положительных чисел, второй - формат отрицательных, третий - формат нулевых значений, а четвертый - формат пустых (Null) значений. Уделим небольшое внимание значению Null. Это значение, используемое для представления отсутствующих или неизвестных данных. При работе с этим значение в поле следует помнить, что в отличие от значения ноль, значения Null не равны. Построим следующий **формат поля ####0,00" рублей";;[Зеленый]"оплата труда сдельная"**. В этом случае если в поле заработная плата будет проставлено значение оклада сотрудника, то оно будет выводиться на экран следующим образом, например, 1250,00 рублей, а если сотрудник получает сдельную оплату, то зеленым цветом будет выводиться строка «оплата труда сдельная».

Сконструируйте сами поля *номер приказа на поступление* и *номер приказа на увольнение*. Учтите, что данные поля *номер приказа на увольнение* содержат дату этого приказа. Не забудьте построить маску ввода для этих полей.

Если дома в вашем распоряжении имеется сканер, то добавьте еще одно поле *фотография сотрудника*. Очень жаль, но имеющаяся в лаборатории техника не позволяет нам построить это поле в рамках нашей лабораторной работы. **Тип данных** этого поля - **поле объекта OLE**. Задавая такой тип данных, пользователь может заполнять это поле связанными или внедренными в таблицу **Microsoft Access** объектами OLE. Так, например, можно в поле объекта OLE внедрять файлы с фотографиями сотрудников.

Позволю себе напомнить вам, что такое OLE**. OLE** – это сокращение от **Object Linking and Embedding** по-русски «связь и внедрение объектов», термин, который используется для работы с двумя типами данных в одном месте. Когда вы связываете два документа или внедряете информацию из одного документа в другой, Windows хранит информацию о связи двух документов. Эта технология позволяет нам редактировать информацию, в одном документе с гарантией, что эти изменения окажут воздействие на все документы, которые с ним связаны.

Попробуем разобраться, в чем различие между **вставкой, связью и внедрением**? Данные, **вставляемые** вами, добавляются к документу в формате, который этот документ понимает. Если вы хотите добавить к документу **внедренный объект**, необходимо использовать технологию OLE. В случае внедрения объекта, второй документ хранится рядом с первым, так что они никогда не смешиваются. Когда внедряете один тип данных в файл, содержащий другой тип данных, они не смешиваются, поэтому вы можете редактировать их отдельно. Несмотря на то, что данные хранятся вместе и выглядят как одно целое, их типы в таком документе остаются разными.

В том случае, когда необходимо использовать два типа данных вместе, но хранить их отдельно, объекты **связывают**. Это может понадобиться, например, тогда, когда вы вставляете картинку с логотипом вашей компании во все документы фирмы. Если внедрять отдельную копию логотипа во все документы, будет много работы, если логотип компании измениться. Необходимо будет найти каждый документ и изменить в нем внедренный логотип. Лучше всего держать логотип в одном файле и включать в каждый файл инструкции о том, как найти **связанный объект**. В этом случае логотип может храниться на любом компьютере в сети вашей компании. **Связь** всегда содержит файл (например, с картинкой) и инструкции о том, как найти оригинал.

И тот и другой методы имеют свои области применения. Все зависит от формы и назначения документа. Внедряя объекты, мы избавляемся от необходимости поддерживать и обслуживать связи, но при этом можем получать файлы огромных размеров, с которыми трудно оперировать. Связывая объекты, мы резко уменьшаем размеры файлов и значительно повышаем производительность компьютера, но вынуждены следить за тем, чтобы все связанные объекты хранились строго в тех папках, в которые они были помещены в момент создания связи. С принципами связывания и внедрения объектов непосредственно соприкасается **принцип совместного использования объектов** в корпоративных вычислительных системах.

Кроме того, можно было бы в таблицу **кадры** добавить также поле *биография сотрудника*, в которое подробно записывать информацию о личных качествах сотрудников и событиях их биографии. Понятно, что это информация текстовая. Но, к сожалению, для такого поля мы не можем определить **тип данных текстовый**, так как **размер поля** для этого типа данных ограничен (**255 байтов**). Поэтому мы выберем **тип данных MEMO.** MEMO - это та называемый длинный текст. В поле, имеющем **тип данных MEMO** можно записать до **65535 символов.**

Закончив конструирование таблицы **кадры**, не забудьте дать правильное имя таблице. **Microsoft Access** предлагает свои имена таблицам (таблица1, таблица2 и т.д.), замените стандартное имя на имя **Кадры**. Имена объектов в СУБД **Microsoft Access** могут быть длиной до 64 символов, в том числе и символов кириллицы, цифр и специальных символов, кроме восклицательного знака точки и угловых скобок.

Следующая таблица, которую нам предстоит сконструировать – это таблица **Клиенты**. Первое поле этой таблицы – это поле *код клиента*. Для этой таблицы оно **ключевое**. Его конструирование аналогично конструированию поля **код сотрудника** в таблице **кадры.**

Далее очевидно, что поле *статус предприятия*, требует сконструировать для ввода в него информации элемент управления **поле со списком**, так как статус работающих в нашей стране предприятий фиксирован и регламентирован законом.

В поле *название предприятия* вряд ли нам удастся построить маску ввода, так как мы это делали в поле *фамилия сотрудника*. Названия предприятий очень не однородны. Некоторые предприятия выбирают для своих названий аббревиатуру, некоторые определяют в нескольких словах, поэтому, задав **тип данных текстовый** для этого поля, просто приблизительно зададим его **размер.**

Созданная нами фирма должна иметь клиентов не только в нашей стране, но и за рубежом. Так как мы знаем название некоторых стран, в которые мы будем импортировать товары с нашего склада, создадим для поля *страна клиента* элемент управления **поле со списком**. Также поступим и, конструируя поле *город клиента***.**

Мы разбили адрес клиента на мелкие атрибуты, выделив в отдельные поля страну и город. Это поможет нам в дальнейшем в нашей информационной системе предоставлять информацию управляющему персоналу для более успешного решения задач маркетинга и менеджмента.

Поле *адрес клиента* конструируем аналогично полю адрес сотрудника. Конструируя поля - *телефон клиента* и *факс клиента*, не забудьте создать **маску ввода.**

Видимо, для того чтобы знать, к кому конкретно обращаться в фирме, можно сконструировать поле *имя и должность контактного лица***.**

Конструируя группу полей для записи банковских реквизитов фирмы, обратим внимание на следующее. В поле *банк* вполне можно задать элемент управления **поле со списком**, в котором перечислить, наиболее популярные банки. А при конструировании полей *ИНН, БИК, расчетный счет, корреспондентский счет и вид деятельности***,** необходимо помнить, что данные в этих полях будут записываться цифрами. Но хотя мы и будем записывать цифры в эти поля, эти данные для нас не числа. Мы не будем производить над ними математические действия, мы будем только записывать их в документы, то есть использовать цифры как набор символов. Поэтому **тип данных** для этих полей – **текстовый**, **размер поля** определяется количеством цифр, которые необходимо записать в поле таблицы, а так как в данных не должны появляться буквы, то мы можем создать для этих полей **маску ввода.**

Так как структуры таблиц **клиенты** и **поставщики** совпадают, то, создав таблицу - **клиенты**, очень легко создать таблицу **поставщики**. Для этого необходимо скопировать в буфер обмена таблицу **клиенты**, и вставить ее затем с именем **поставщики**. Затем открыть копию таблицы клиенты с именем **поставщики** и внимательно откорректировать ее в соответствии с названием.

Следующая таблица, необходимая нам для работы, это таблица **склад**. **Ключевое** поле этой таблицы **код товара**. Его конструирование не отличается от конструирования полей *код сотрудника и код клиента*. Поле *название товара* конструируется аналогично полю *название фирмы клиента***.**

А вот выбирая **тип данных** для поля - *количество товара*, надо решить какие товары будет продавать наша фирма. Если мы будем продавать штучный товар, например, косметику, то нас устроит **тип данных числовой, размер поля целое**. Если же скажем, мы продаем строительные материалы, то нам необходимо выбрать **тип данных числовой,** но **размер поля установить с плавающей точкой**, т.к. к строительным материалам можно отнести, например, линолеум, продаваемый в кусках, длина которых может быть не целое число метров. Причем в этом случае **число десятичных знаков** после запятой мы должны будем установить **два**. Если же мы хотим торговать пищевыми продуктами: мясом, рыбой и так далее, вес которых измеряется в килограммах и граммах, то мы должны, выбрав **тип данных числовой**, и установив **размер поля с плавающей запятой,** определить **число десятичных знаков** после запятой **три.**

При этом следует учесть, что поля *количество товара* и *остаток товара на складе*, пока товар со склада не продан, имеют одинаковые значения. Мы обязательно это учтем, и, заполняя данными таблицу **склад**, построим форму для заполнения данных о полученных на склад товарах таким образом, что по заполнению поля *количество купленного товара*, поле *остаток товара на складе* будет заполняться автоматически. Поэтому поле *остаток товара на складе* конструируется также как и поле - *количество купленного на склад товара*. Однако следует учесть, что данные в поле *остаток товара* будут автоматически изменяться по мере оформления заказов. И хотя, разрабатывая форму, по которой сотрудник нашей фирмы будет формировать заказ клиента, мы постараемся показать ему точное значение этого поля, но никто не застрахован от ошибок. Вполне возможна ситуация, когда продано товара больше, чем имеется на складе. В этом случае значением поля *остаток* будет отрицательное число. Необходимо, например, цветом числа показать это пользователю. К тому же если весь товар продан, и поле *остаток* имеет значение ноль, мы можем выводить на экран в форму, и на печать в отчеты строку текста «товара нет». Для этого необходимо определить **формат поля** *остаток*. Учитывая определенные выше требования, он будет выглядеть, например, так: **######0,00;[Красный]" –"######0,00;[Синий]"Товара нет".**

Поле - *единица измерения* необходимо в таблице склад. Закупая товары, и зная количество закупленного товара, мы должны понимать, в каких единицах мы это количество считаем: в штуках, коробках, галлонах, флаконах, кусках и т.д. Так, как нам точно известно, какими товарами будет торговать наша фирма, мы можем, определив **текстовый тип данных** и **размер** этого **поля**, сконструировать также для его заполнения **элемент управления поле со списком.**

Конструируя поле – *номер накладной*, необходимо учитывать какие номера накладных используются при работе с документами в данной фирме. Будем считать, что это набор символов из пяти цифр. Определим **тип данных** для этого поля **текстовый**, а **размер** ему зададим **5.**

Для поля *закупочная цена* выберем **тип данных денежный. Формат поля** оставим стандартный **денежный**, этот формат предлагает нам СУБД **Microsoft Access** по умолчанию. В этом случае на экран в формы и в бумажные отчеты будет выводиться значение закупочной цены и при этом: группы из трех разрядов будут отделены друг от друга пробелами; свойство **число десятичных знаков** по умолчанию получит значение **2;** а после числового значения выведется символ «р.», показывающий, что цена в рублях.

При конструировании поля *код поставщика* не будем забывать, что это поле мы определили как **внешний** ключ, для дальнейшей связи данных в таблицах **склад** и **поставщики**. В таблице **поставщики** мы задали **первичный ключ код поставщика. Тип данных** этого поля **счетчик**. Как уже говорилось, в поле с **типом данных - счетчик** мы будем получать числовые значения – целые числа **размером длинное целое число.**

Понятно, что, определяя тип данных для поля код поставщика, мы не можем взять тип данных - счетчик. Во-первых, если мы определяем тип данных счетчик, то в этом поле будут автоматически проставляться числовые значения либо методом случайных чисел, либо методом инкремента (каждое последующее больше предыдущего на единицу). Нас это не устраивает, нам в этом поле необходимо иметь значение, соответствующее коду поставщика, конкретного товара, в записи о котором этот код поставщика записан. Во-вторых, мы уже задали тип данных счетчик для поля код товара, а по правилам **Microsoft Access** в таблицах может быть только одно поле с таким типом данных.

Поэтому мы определим **тип данных** для поля *код поставщика* **числовой**, а **размер поля длинное целое**, т.к. данные этого поля связывают таблицы **склад** и **поставщики** и должны иметь одинаковый размер.

Поле *дата поступления заказа* в таблице **склад** конструируется аналогично полю *дата поступления на работу* в таблице **кадры**. Не забудьте определить свойство этого поля **значение по умолчанию**, задав **значением по умолчанию текущую дату** из памяти компьютера. Как уже говорилось выше, это можно сделать, поставив в строку этого свойства вызов встроенной функции СУБД **Microsoft Access Date().**

Нам осталось сконструировать две таблицы - **заказы** и **заказ-товар**. Данные в этих таблицах помогут связать все данные нашей информационной системы управления в единое целое.

Первое поле таблицы **заказы** – это поле *код заказа*. Очевидно, что это поле **первичный ключ** отношения **заказы** и будет конструироваться аналогично полям *код сотрудника, код товара, код клиента и код поставщика*.

Поля *код клиента* и *код сотрудника* это **внешние ключи**, помогающие связать данные таблиц **кадры** и **клиенты** с данными таблицы **заказы**. Они конструируются аналогично полю *код поставщика* в таблице **склад.**

Поля *дата поступления заказа* и *дата выполнения заказа* конструируются аналогично полям из таблиц **кадры** – *дата поступления на работу* и *дата рождения* и полю *дата поступления товара на склад* из таблицы **склад.**

Последняя таблица, которую нам осталось сконструировать это таблица **заказ-товар**. Она связывает данные таблиц **склад** и **заказы** и разбивает связь **многие ко многим** между этими отношениями на связи **один ко многим, многие к одному.**

Поля таблицы заказ-товар *код товара и код заказа* это **внешние ключи** помогающие связать данные таблиц **склад** и **заказы** с данными таблицы **заказ-товар.** Они конструируются аналогично полям *код поставщика* в таблице **склад** и *код клиента* и *код сотрудника* в таблице **заказы.**

Поле *количество товара купленного в данном заказе* должно конструироваться аналогично полю *количество закупленного товара на склад*, в таблице **склад** т.к. не исключена ситуация, что мы всю закупленную партию товара продадим на один заказ, значит, данные в этих полях вполне могут быть одинаковыми.

Встает вопрос, как определить ключевое поле для таблицы заказ-товар? По определению первичным ключом отношения, например R, с атрибутами A_i, A_j,……A_k, будет являться K=(A_i, A_j,……A_k) тогда и только тогда, когда удовлетворяются два независимых от времени условия: уникальность и минимальность. Первое условие (уникальность) указывает на то, что в произвольно заданный момент времени никакие два различных картежа отношения R не имеют одного и того же значения K=(A_i, Aj,…,$A_к$). Второе условие (минимальность) говорит о том, что ни один из атрибутов в K=(A_i, Aj,…, $A_к$) не может быть исключен из него (из К) без нарушения условия уникальности.

В таблице **заказ-товар** первичным ключом, удовлетворяющим требованиям уникальности и минимальности, будет так называемый **составной ключ**, состоящий из

атрибутов (полей) *код товара и код заказа*. Для его определения необходимо выделить одновременно поля *код товара и код заказа* и нажать на панели инструментов кнопку, определяющую ключевое поле.

Во время конструирования таблиц можно переходить из режима конструирования таблицы в режим заполнения таблицы данными. Это может потребоваться, чтобы проверить правильно ли сконструирована **маска ввода** или правильно ли работает **элемент управления поле со списком**. Но следует помнить, что заполнять таблицы данными при конструировании не желательно. Для ввода данных в дальнейшем будут разработаны формы. Поэтому, введя запись для проверки работы таблицы в режиме заполнения, ее нужно затем удалить. После того, как вы сконструировали все таблицы, необходимо построить **схему данных**. При построении **схемы данных,** на этом этапе работы с информационной управляющей системой, таблицы не должны содержать данных.

5 Построение схемы данных

В реляционных СУБД пользователь может описать связи между таблицами. **Microsoft Access** учитывает эти связи при поиске взаимосвязанных данных во время обработки запросов, форм и отчетов, базирующихся на нескольких таблицах.

Для установки связи между таблицами в окне базы данных следует выбрать команду **Схема данных** из меню **Правка** или кнопку схема данных на панели инструментов  . Откроется окно, в котором будут представлены таблицы базы данных с указанием их связей, если они были созданы ранее. В противном случае появиться окно диалога **Добавление таблицы.** В этом окне пользователь видит список таблиц базы данных и может добавить их в схему. Если вы хотите добавить новую таблицу к уже имеющейся схеме данных, то можно выбрать команду **Добавить таблицу** из меню **Связи** или нажать кнопку  на панели инструментов. В схему данных можно добавлять не только таблицы, но и запросы, поэтому окно диалога **Добавить таблицу** имеет три корешка **таблицы** (со списком таблиц), **запросы** (со списком запросов) и **таблицы и запросы** (с общим списком таблиц и запросов).

Создавая связи между таблицами, с помощью мыши перетащите поле, которое следует использовать для установления связи (обычно это ключевое поле, выделенное полужирным шрифтом), из списка полей главной таблицы на соответствующее поле подчиненной таблицы. Связи создаются от **первичных ключей к внешним ключам.**

В появляющемся при создании связи окне диалога **Связи** необходимо, во-первых, проверить правильно ли вы провели связь (это видно по именам полей в основной и связанной таблице), во-вторых, проверить правильно ли **Microsoft Access** определил тип отношения для вашей связи (один к одному или один ко многим), в-третьих, правильно выбрать параметры связи. Установленная связь отобразится на схеме линией, соединяющей связанные поля из таблиц. При этом будет отображен тип связи (один к одному или один - ко – многим). Вид окна диалога связи показан на рисунке 4.

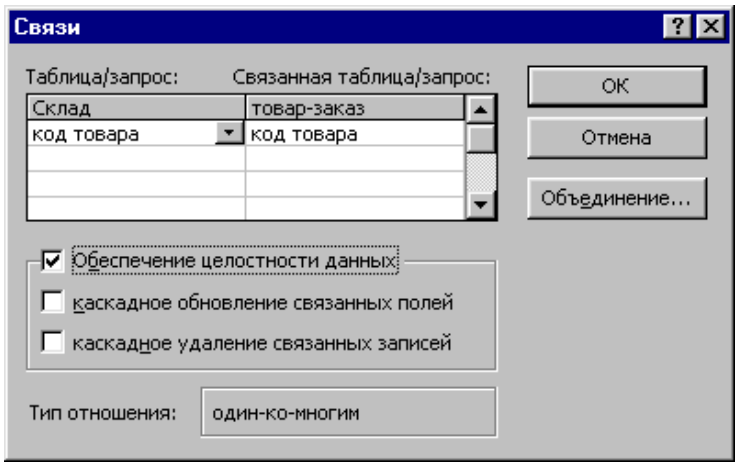

Рисунок - 4 Окно диалога «Связи»

Две опции **Каскадное обновление связанных полей** и **Каскадное удаление связанных записей**, которые доступны только при включенном индикаторе **Обеспечение**

целостности данных, обеспечивают каскадное обновление полей и каскадное удаление связанных записей в связанных таблицах базы данных.

Для связей, для которых определена **целостность данных**, пользователь имеет возможность указать, следует ли автоматически выполнять для связанных записей операции **каскадного обновления и каскадного удаления**. Если *включить* данные параметры, *станут возможными* **операции удаления и обновления**, которые в противном случае *запрещены* **условиями целостности данных**. Чтобы обеспечить **целостность данных** при **удалении записей** или изменении значения ключевого поля в главной таблице, автоматически вносятся необходимые изменения в связанные таблицы.

Если при определении связи установить флажок **Каскадное обновление связанных полей**, любое изменение значения в ключевом поле главной таблицы приведет к автоматическому обновлению соответствующих значений во всех связанных записях. Например, при изменении кода клиента в таблице «Клиенты» будет автоматически обновлено поле «Код Клиента» во всех записях таблицы «Заказы» для заказов каждого клиента, поэтому целостность данных не будет нарушена. **Microsoft Access** выполнит **каскадное обновление** без вывода предупреждающих сообщений.

Если в главной таблице ключевым полем является поле счетчика, как в нашей информационной системе, то установление флажка **Каскадное обновление связанных полей** не приведет к каким-либо результатам, так как изменить значение поля счетчика невозможно.

Если при определении связи установить флажок **Каскадное удаление связанных записей**, любое *удаление записи в главной таблице* приведет к *автоматическому удалению связанных записей в подчиненной таблице*. Например, при удалении из таблицы «Клиенты» записи конкретного клиента будут автоматически удалены все связанные записи в таблице «Заказы» (а также записи в таблице «Заказ-товар», связанные с записями в таблице «Заказы»). Если записи удаляются из формы или таблицы при установленном флажке **Каскадное удаление связанных записей, Microsoft Access** выводит предупреждение о возможности удаления связанных записей. Если же записи удаляются с помощью запроса на удаление записей, то удаление осуществляется автоматически без вывода предупреждения.

6 Разработка форм

6.1 Общие сведения о формах

Для более удобной работы с данными, которые в информационных системах, построенных на основе реляционных СУБД, хранятся в таблицах, разрабатываются формы. Рассмотрим основные направления для применения разработанных форм:

— **для ввода данных** в таблицы, создаются формы, предназначенные только для ввода в базу новых данных или значений, помогающих автоматизировать выполнение пользовательского приложения, решающего конкретные пользовательские задачи.

— **формы, которые в удобном для пользователя ракурсе выводят данные из таблиц** на экран и позволяют эти данные редактировать. При этом Microsoft Access позволяет сделать все данные в форме или некоторую их часть доступными только для чтения, и таким способом не допустить их корректировку и обеспечить их надежное хранение. В одну форму на один экран можно помещать данные из разных таблиц, связанных между собой. Также в формах можно производить необходимые вычисления, используя значения, выводимых в форму полей, и выводить полученные значения на экран в свободных элементах формы, несвязанных с полями таблиц. Кроме того, в формах в зависимости от значений в одних полях значения в других полях можно сделать невидимыми. Все перечисленные возможности позволяют сделать форму гибким инструментом для работы с данными в таблицах информационной системы.

— с помощью форм в Microsoft Access имеется возможность **управлять ходом выполнения пользовательского приложения** и, таким образом, обеспечить удобную работу пользователя, создав пользовательский интерфейс. То есть можно создавать формы, при работе с которыми пользователь сможет легко понять последовательность действий, которую он должен выполнить, чтобы решить стоящую перед ним задачу. Кроме того, разработка таких форм позволяет автоматизировать работу информационной системы управления, задав последовательность выполнения действий для решения конкретных задач. В таких формах создаются специальные элементы управления (в основном кнопки), с помощью которых можно: открывать другие формы; выполнять запросы или команды меню; фильтровать выводимые на экран данные; устанавливать значения в записях и формах; выводить на экран специальные пользовательские меню; распечатывать отчеты. Имеется возможность спроектировать форму таким образом, чтобы макросы или модули и функции языка программирования Visual Basic for Application (VBA) запускались автоматически в ответ на определенные события.

— также можно разработать **форму, содержащую некоторое сообщение**. С помощью таких сообщений формы могут предоставлять пользователю вспомогательную информацию о работе приложения. Информировать его о выполняемых в данный момент действиях, или предупредить о последствиях следующего шага пользователя, или вывести сообщение об ошибке.

— кроме того, хоть в Microsoft Access **для печати информации** и необходимых документов предусмотрены отчеты, можно также распечатать информацию в форме. Имеется даже возможность определить первый набор операций для вывода формы на экран, а другой на печать.

Обобщая все вышеизложенное можно сделать вывод о том, что **главная задача форм создать удобный пользовательский интерфейс**. Организовать максимально удобную работу пользователя с информационной системой.

Формы в Microsoft Access бывают следующих видов:

— **простая форма** - в один столбец (автоформа в столбец) в форму выводятся данные одной записи в таблице – это так называемый карточный интерфейс, т.е. данные одной записи, представлены в виде карточки (карточка клиента, карточка сотрудника, карточка товара, поступающего на склад).

— **многостраничная форма** – вариант простой формы, выполняется, когда поля из одной записи не помещаются на одну страницу, в этом случае при работе с формой пользователь вынужден использовать вертикальную линию прокрутки, или клавиши PgUp/PgDn, для того чтобы работать с полями в форме. Построение таких форм не желательно. Пользователю удобнее представлять информацию в формах, которые помещаются на один экран. Для организации таких форм можно использовать, например, элементы управления «корешки вкладок».

— **ленточная форма (автоформа ленточная)** - используется для просмотра списка записей, так называемый табличный интерфейс. В этой форме отформатированные записи выводятся на экран одна за другой, то есть в форму выводится информация о многих записях в таблице. Эту форму лучше строить для записей с небольшим количеством полей или использовать в такой форме горизонтальную линию прокрутки.

— **табличная форма (автоформа табличная)** – вариант табличного интерфейса, при котором в форму выводятся не отформатированные записи.

— **сложные формы (формы «один ко многим»)** - формы создаваемые на основе запроса из двух и более связанных таблиц. В этих формах информация из одной таблицы (связь со стороны «один») выводится в основной (главной) форме, а информация из других таблиц (связь со стороны «многие») выводится в другие подчиненные формы.

— **подчиненная форма** - внедряется в основную форму и содержит информацию из другой таблицы или запроса (связь со стороны «многие»). Подчиненная форма должна быть связана с основной таблицей через какое-нибудь поле.

— **всплывающая форма** - специальное окно диалога, которое служит для вывода сообщений для пользователя или для ввода данных пользователем, прежде чем приложение Microsoft Access продолжит свою работу. Всплывающая форма располагается поверх других открытых форм, даже если активной является другая форма. Всплывающая форма может быть немодальной или модальной.

— **модальная форма** – окно диалога, которое остается поверх других окон. Фокус ввода невозможно перевести в другие окна, пока модальная форма не будет закрыта или свернута. Если всплывающая форма является немодальной, пользователь имеет возможность получить доступ к другим объектам и командам меню, пока форма открыта, а если модальной – то нет.

Всплывающие модальные и немодальные формы используются для создания наиболее удобного пользовательского интерфейса и позволяют обеспечить защиту системы от некорректной работы пользователя.

В Microsoft Access имеется возможность создания форм, как в режиме конструктора, так и с помощью мастера. При разработке информационной системы следует использовать оба метода. Формы в Microsoft Access состоят из трех основных частей: **области заголовка, области данных и области примечания.**

Вся информация в формах содержится **в элементах управления**. Элементы управления могут быть **связанными и несвязанными**. **Связанные элементы**

управления вводят/выводят данные из базовых таблиц или запросов. **Несвязанные элементы управления** показывают данные, вычисляемые непосредственно в форме, или текст, какие-либо числа, дату или время, номер страницы.

При конструировании форм для связанных элементов тип элемента управления конструируется таким, каким он был задан при конструировании таблицы. При этом поле помещается в область данных формы прямым перетаскиванием его названия из списка полей. В том случае, когда необходимо задать в форме тип элемента управления, отличный от того, который был задан при конструировании таблицы, можно его построить с помощью **«панели элементов».**

Элементы управления **можно перемещать**, для этого элемент управления выделяют, а затем добиваются **вида указателя мыши в виде ладони**. Перемещать элемент управления можно **отдельно от связанной с ним надписи, в этом случае добиваются вида указателя мыши в виде ладони с выдвинутым указательным пальцем**, маркер для такого перемещения в верхнем левом углу элемента управления.

При конструировании формы необходимо учитывать следующее:

— информацию, **которую вводят** в базу данных или вставляют из таблицы базы данных, **размещают в области данных;**

— информацию или элементы управления, которые **не меняются от записи к записи**, добавляют **либо в область заголовка, либо область примечания**;

— все данные в форме **желательно располагать на одном экране**, по возможности убрать в форме горизонтальные и вертикальные линии прокрутки, для этого можно использовать, например, элемент управления – «корешки вкладок»;

— **порядок перемещения курсора** в формах для ввода данных **по нажатию на клавиши Tab или Enter**, должен быть последовательным (иначе с такой формой пользователю будет трудно работать и увеличиться количество ошибок ввода).

Разрабатывая форму, пользователь **определяет свойства ее объектов**. Свойства можно определять **для всей формы, для областей заголовка, данных и примечания, отдельно для каждого элемента управления и даже отдельно для надписи в элементе управления.**

Свойства бывают:

— **свойства макета -** определяют внешний вид формы;

— **свойства данных** – определяют с какими данными мы работаем в форме;

— **свойства событий** – определяют обработку событий в процессе работы пользователя с формой;

— **другие** – некоторые свойства, не входящие в вышеперечисленные категории.

Разработаем основные формы нашей системы.

6.2 Конструирование формы «Кадры»

Форма «Кадры» предназначена для ввода в базу данных информации о сотрудниках нашей фирмы.

В окне базы данных выбираем корешок «формы», по нажатию на кнопку «создать» появляется окно диалога, в котором отмечаем, что форма будет создана конструктором, и **выбираем таблицу «Кадры»**, для которой хотим сконструировать форму.

Так как форма «Кадры» как и любая другая форма для ввода данных состоит из трех основных частей: области заголовка, области данных и области примечания, **то необходимо включить переключатель в выпадающем меню из пункта верхнего меню**

«вид» - «заголовок/примечание». Затем необходимо проверить **наличие на экране панели элементов**. **На панели элементов обязательно должна быть нажата кнопка мастера**. Также необходимо **проверить, что на экран выведен список полей таблицы**, в которую будут вводиться данные.

В области заголовка создается надпись – **«Личная карточка сотрудника».** Для создания надписи на панели элементов выбираем кнопку «надпись» и определяем область, в которой будет надпись. Затем с клавиатуры вписываем необходимый текст. Также **в область заголовка формы поместим эмблему** нашей фирмы. Для этого на панели элементов выберем кнопку «свободная рамка объекта», определим область, в которой будет наша эмблема, и в открывшемся окне диалога покажем путь к файлу, в котором находится рисунок с эмблемой. Можно воспользоваться любым файлом с рисунком из имеющихся на компьютере, или нарисовать свою эмблему заранее, воспользовавшись графическим редактором.

В область данных прямым перетаскиванием из списка полей перетащить поля «**Фамилия сотрудника», «Имя сотрудника» и «Отчество сотрудника».** Обратите внимание, что в форме поле состоит из двух частей: области для ввода/вывода данных и подписи поля. Желательно расположить подписи полей над областью для ввода/вывода (см. п.6.1).

Остальные данные в форме расположим на вкладках. Это позволит нам избежать работы пользователя с полосами прокрутки. Для того, чтобы создать вкладку на панели элементов выбираем кнопку «вкладка» и в области данных рисуем вкладку. По умолчанию вкладки состоят из двух корешков. При необходимости можно добавлять вкладки (пункт верхнего меню «вставка» - «вкладка»). Вкладкам дадим имена, изменяя свойства каждой вкладки (см. п.6.1). Одну из вкладок назовем «Личные данные», другую «Служебные данные». Затем в область данных на вкладки перетаскиваем поля из списка полей.

В область примечания добавим кнопки, которые помогут пользователю при работе с формой. Создадим следующие кнопки: «на первую запись»; «на последнюю запись»; «на запись вперед»; «на запись назад»; «поиск записи»; «добавление записи»; «удаление записи»; «ввод данных закончен». Кнопки создаем, воспользовавшись на панели элементов инструментом «кнопка». При этом первые семь кнопок создаем с помощью мастера, а для создания последней мастер отключим и просто нарисуем кнопку, действие для нее будет задано в дальнейшем, когда нами будут созданы макросы, позволяющие создать гибкий пользовательский интерфейс нашей системы.

Изменяя свойства элементов формы (см. п.6.1) раскрасьте ее. Подберите разные цвета для области заголовка, области данных и области примечаний. Для надписей и ввода данных подберите удобный и красивый шрифт. Раскрасьте буквы и области полей. Задайте рамки объектам и т.д. Внимательно изучите «свойства макета» объектов и постарайтесь создать красивую форму. Измените размеры элементов управления таким образом, чтобы вся информация в них была видна. **Кнопки желательно создавать одинакового размера. Удобнее, когда на кнопке надпись, определяющая ее действие, а не картинка.**

Примерный вид формы «Кадры» показан на рисунке 5.

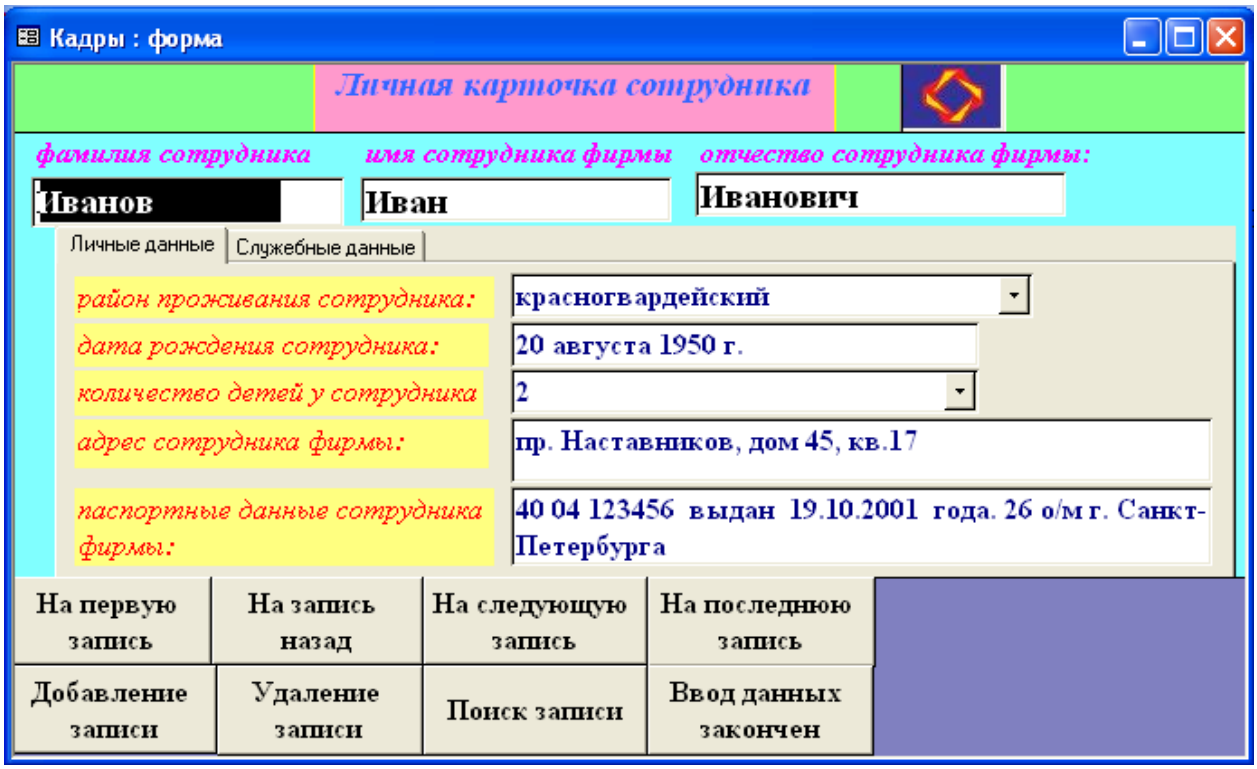

Рисунок 5 – Форма «Кадры»

После того, как работы по раскрашиванию формы закончены, **задайте правильный порядок перехода между полями, при нажатии на клавиши «Tab» и «Enter».** Это можно выполнить, вызвав окно диалога последовательность перехода из контекстного меню области данных. Вид окна диалога показан на рисунке 6.

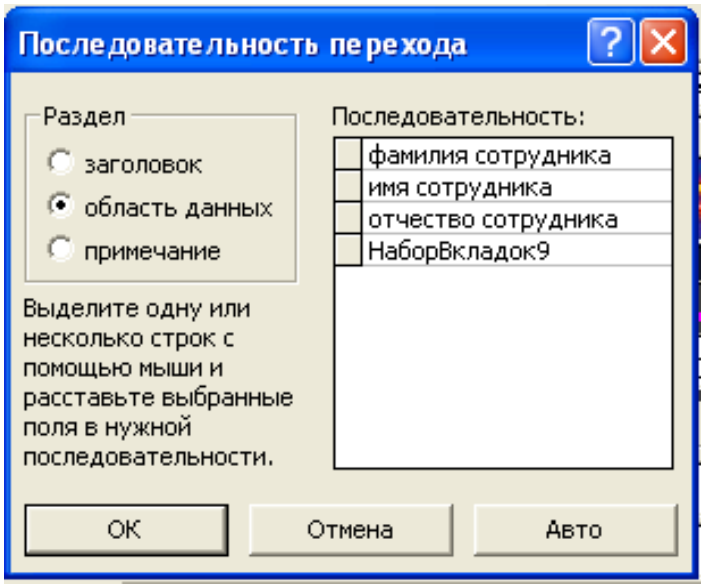

Рисунок 6 – Окно диалога «Последовательность перехода»

Изменяя свойства всей формы, **уберите «Область выделения записи» и «Кнопки перехода по записям». Задайте для нее «Выравнивание по центру экрана».** После

того, как выровняете границы формы, **задайте тип границы – «тонкая»,** в этом случае границы формы нельзя будет изменить. Конструируя форму кадры, постарайтесь максимально изучить свойства макета для различных элементов формы. Это позволит в дальнейшем создавать красивые формы, с которыми будет приятно работать пользователям.

6.3 Создание форм «Клиенты» и «Поставщики»

Начнем с создания формы «Клиенты». Форму будем создавать мастером. Отвечая на вопросы мастера, следует помнить, что форма предназначена для ввода данных о клиентах и это **простая форма в один столбец**. При выборе полей, которые выводятся в форму, не берем поле «код клиента», так тип данных для этого поля «счетчик» и значит, данные этого поля будут вводиться автоматически.

После того, как мастер создал форму, **перейдем в режим конструктора и придадим ей вид удобный для работы с ней пользователя.**

Информацию в области данных можно разместить произвольно, но при этом должны быть соблюдены следующие требования:

— вся информация должна размещаться на одном экране;

— порядок перехода между полями в форме должен иметь смысл;

— кнопки в области примечания должны быть такие же, как в форме «Кадры»;

— в области заголовка должна быть надпись – «Информация о клиентах» и эмблема.

Когда создана форма «Клиенты» ее можно скопировать в буфер и вставить с именем «Поставщики». Затем открыть в режиме конструктора и в окне диалога «свойства» всей формы, на корешке «данные», **изменить свойство «источник данных». Заменить таблицу «Клиенты», на таблицу «Поставщики».** В заголовке формы изменить надпись на надпись «**Информация о поставщиках**».

Форму «Поставщики» оформить в другой цветовой гамме.

6.4 Заполнение базы данных

Для создания следующих форм нам необходимо, чтобы в таблицах нашей базы данных были данные, приступим к заполнению базы.

По форме «Кадры» заполним таблицу «Кадры» в соответствии со штатным расписанием (см. п. 2.1), оставляя вакантными следующие должности: бухгалтер – одна штатная единица, торговые агенты – 2, торговые представители – 1, грузчик – 1 (при заполнении таблицы «Кадры» вакантными должностями, в записи должно быть заполнено только два поля – «отдел» и «должность сотрудника»).

Заполним таблицы «Клиенты» и «Поставщики» по формам. В каждую таблицу внесем по пять записей.

6.5 Создание формы «Склад»

Форма строится мастером по таблице «Склад».

В форме должны быть реализованы следующие действия:

1. после того как пользователь заполняет (обновляет) значение поля «количество товара, поступившего на склад», автоматически должно устанавливаться такое же значение в поле «остаток». Это действие определяется тем, что по создаваемой форме будут вводиться данные о поступающих в фирму товарах, а на момент поступления товара на склад остаток равен поступающему количеству, так как еще ничего не продали.

2. для ввода данных о поставщиках построим элемент управления «поле со списком» на основании запроса «список поставщиков». Это необходимо, так как в таблицу «Склад» мы должны заполнять код поставщика, от которого мы получаем товар. В документе (в накладной), по которому мы вводим данные, написаны статус и название фирмы, от которой получен товар. Невозможно запомнить номера поставщиков. Поэтому мы должны в форме «склад» создать такой элемент управления, чтобы мы выбирали из списка поставщиков, поставщика поставившего товар по статусу и названию фирмы, а в таблицу «Склад» при этом вставился бы код поставщика, соответствующий фирме поставщика с таким названием.

3. ввод данных о названиях товаров должен осуществляться с помощью элемента управления «поле со списком», при этом этот элемент должен работать с автоподстановкой. Наша фирма продает товары определенного профиля. Естественно, что названия товаров повторяются. Поэтому мы должны избавить пользователя нашей информационной системы от ввода названий товаров, если они были уже единожды введены. Элемент управления должен работать с автоподстановкой, то есть, как только в поле введена первая буква из названия товара, компьютер должен предложить пользователю название товара на эту букву, которое он может корректировать или взять полностью. Если поступают товары новых наименований, то пользователь вводит их, и они добавляются в элемент управления «поле со списком». Очевидно, что хотя на склад поступают несколько раз товары одного наименования, в элементе управления «поле со списком» каждое название должно присутствовать только один раз и все названия товаров желательно отсортировать по алфавиту.

Решение первой задачи.

Открываем форму «Склад» в режиме конструктора.

Для поля «закупленное количество» вызываем окно диалога «Свойства», выбираем корешок «свойства событий», строка «после обновления» (After Update).

Напротив этой строки нажимаем кнопку «мастера», в появившемся окне диалога мастер предлагает нам построить: либо выражение, либо макрос, либо программу. Выбираем программу. Попадаем в окно для построения программных модулей обслуживающих формы. Нам необходимо записать лишь один оператор языка программирования. Оператор присваивания. Присвоить значению поля «остаток товара на складе» значение поля «количество закупленного товара». Это можно выполнить в ручную вписав: **[Остаток товара на складе] = [Количество закупленного товара].** Можно воспользоваться средствами MS Access и выбрать имена полей из списка Complete Word. В этом случае запись оператора присваивания будет выглядеть так: **Остаток_товара_на_складе = Количество_закупленного_товара**. В любом случае, созданная программа будет автоматически присваивать полю «остаток товара на складе» значение из поля «количество закупленного товара» как только его обновят, то есть впишут в него какое-то значение.

Решение второй задачи

Для ввода данных о поставщиках построим элемент управления «поле со списком» на основании запроса «список поставщиков».

В СУБД Access проектирование запросов реализуется двумя способами:

— запрос проектируется на основе бланка QBE (Query By Example). Пользователь при таком проектировании задает отдельные параметры запроса в окне проектирования с использованием подсказок и образцов;

— запросы, проектируемые на основе структурированного языка запросов SQL (Structured Query Language). Формируя такой запрос, пользователь применяет инструкции и функции языка, выстраивая некоторое описание.

Типы запросов легко транслируются из одного вида в другой, поэтому СУБД Access безразлично каким образом строится запрос.

Строим простой запрос-выборку «список поставщиков» на основе бланка QBE.

С помощью запросов пользователь может выбирать из базы данных информацию, которая его интересует в данный момент времени.

Результат запроса представляет собой таблицу, которая называется Dynaset – динамический набор данных. В эту таблицу включают поля, выбранные из основных таблиц, записи которых удовлетворяют критериям отбора в запрос.

Dynaset при каждом выполнении запроса строится заново на основе свежих табличных данных.

Попадаем в бланк QBE. Каждый столбец бланка QBE относится к одному полю, с которым будем работать в данном запросе. Он может быть просто полем одной из таблиц, вычисляемым полем или итоговым полем.

В Access можно задавать вычисления над любыми полями таблицы и сделать вычисляемое значение новым полем в наборе записей. Для этого можно использовать множество встроенных в Access функций, а также создавать поля с использованием арифметических операций над полями таблицы. Кроме того, можно создавать новые текстовые поля, как результат конкатенации текстовых полей и символьных констант.

В выражении можно использовать следующие операторы – таблица 13.

Таблица 13 – операторы вычисления

оператор	действие
+	Сложение
-	Вычитание
*	Умножение;
/	Деление;
\	Округляет два арифметических выражения до целых значений и делит первое число на второе, результат округляется до целого;
^	Возводит первое арифметическое выражение в степень, заданную вторым выражением;
MOD	Округляет оба арифметических выражения до целых значений, делит первое число на второе и возвращает в качестве результата остаток;
&	Оператор конкатенации. Создает текстовую строку как результат присоединения второй строки или текстовой константы к концу первой. Если один из операторов является числом, , Access перед проведением конкатенации преобразует его в строку символов.

Создание запроса-выборки «Список поставщиков». Это запрос с одним вычисляемым полем. На примере построения этого запроса познакомимся с работой в бланке QBE. Рассмотрим строки бланка QBE.). Окно для построения запроса в режиме конструктора показано на рисунке 7.

Строка «поле» используется для выбора имен полей, которые должны присутствовать в наборе записей запроса, по которым надо провести сортировку или значение которых надо проверить. Имена полей, которые будут выводиться в запросах, можно изменить, также в этой строке можно создавать вычисляемые поля, используя различные выражения.

В нашем запросе в первый столбец на первую строку выбираем поле **«код поставщика»** прямым перетаскиванием.

Во втором столбце первой строки строим выражение построителем выражений:

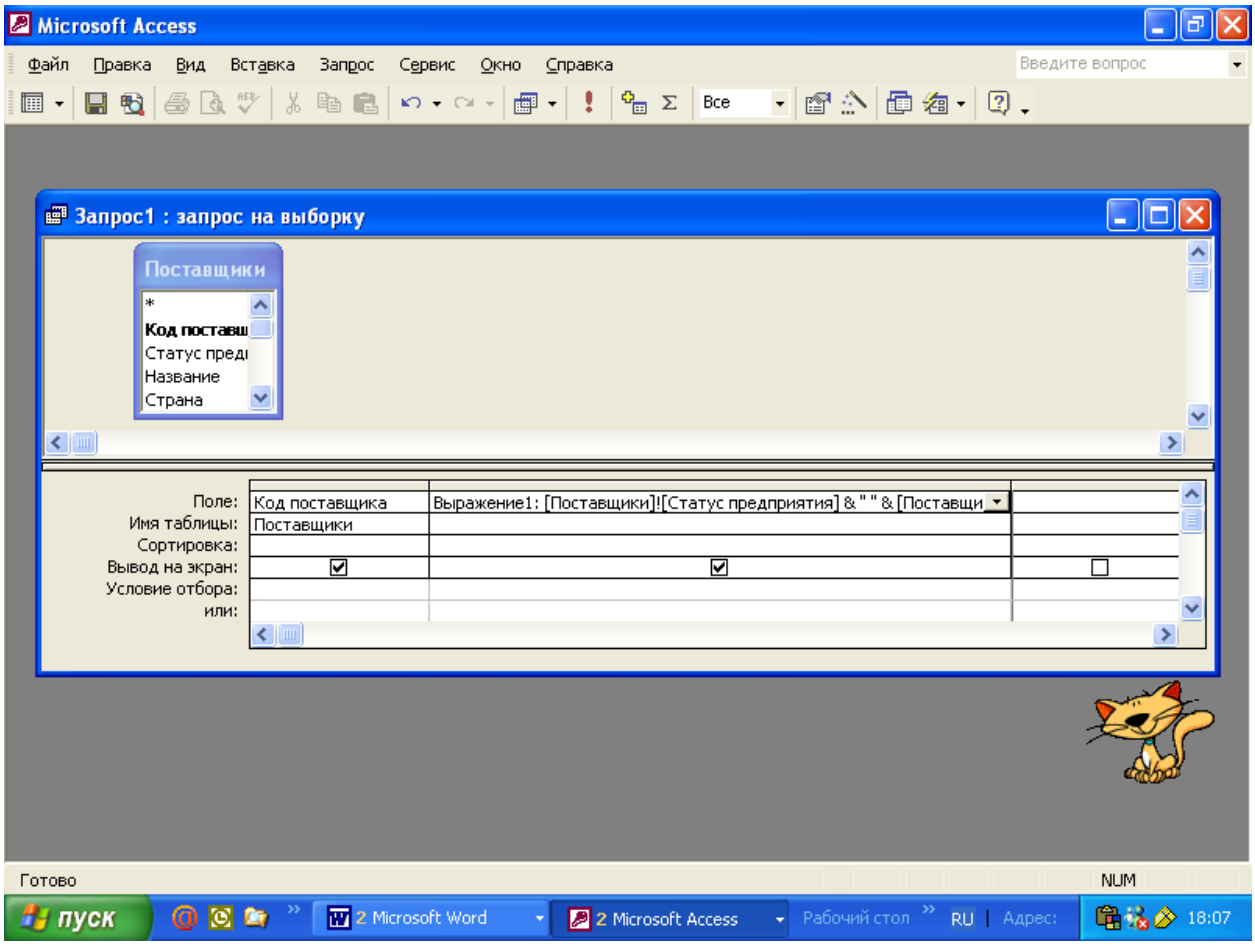

Рисунок 7 – Построение запроса в режиме конструктора

Окно построителя выражений состоит из двух частей:

верхняя часть – область для построения выражения и кнопки, необходимые пользователю при построении выражения;

нижняя часть – информация располагается в три столбца: в первом столбце объекты базы данных в виде иерархического дерева (для выбора нужного объекта необходимо открыть ветку, заменив плюс на минус,) во втором столбце список составляющих того объекта, который выделен в первой колонке, в третьем столбце список составляющих того объекта, который выделен во втором столбце.

Окно построителя выражений показано на рисунке 8.

Выражение1: [Поставщики]![Статус предприятия] & " " & [Поставщики]![Название]

Вторая строка «имя таблицы» - в эту строку заносятся имена таблиц, из которых выбираются имена полей, по которым строится запрос.

Третья строка бланка QBE – сортировка данных. Access выводит записи в том порядке, в каком они записаны в таблицу. Если записи необходимо сортировать, то в столбце этого поля на строке «сортировка» выбирается одно из значений списка по возрастанию/по убыванию.

Четвертая строка бланка QBE – вывод на экран. В этой строке устанавливается переключатель, определяющий – выводить на экран содержимое полей или нет. Часто в запрос требуется включить поля, по которым задают условия отбора, а содержимое этих полей на экран не выводится. В этом случае переключатель в положении выключен. В нашем случае переключатель - включен.

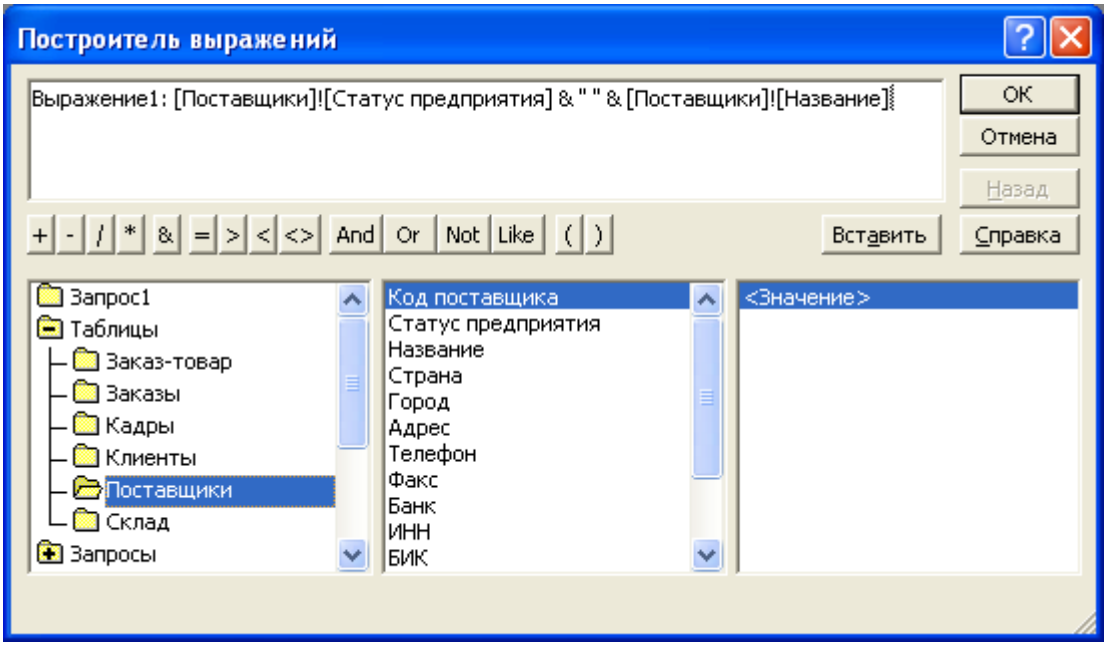

Рисунок 8 - Окно построителя выражений

Описание работы в бланке QBE со **строками – «условие» и «или»** смотри в построении запроса – «**список сотрудников, обслуживающих заказы**».

После того как заполнен бланк QBE для некоторых полей необходимо установить их свойства в запросе.

В общем случае поля, выводимые в наборе записей запроса, наследуют свойства, заданные для соответствующих полей таблицы.

В Access имеется возможность изменить значение этих свойств или задавать свойства для вычисляемых полей (для вычисляемых полей желательно определять хотя бы одно свойство – «подпись поля»).

Окно диалога «свойства» можно вызвать либо из контекстного меню, либо из выпадающего из пункта верхнего меню «вид». Имеется возможность задавать следующие свойства:

1. описание объекта. В это свойство заносится информация, которая выводится в строке состояния окна-запроса в режиме таблицы Dynaset или в формах, построенных по этому запросу, когда данное поле является текущим. В первых пяти запросах это свойство заполнять обязательно;

2. формат поля – представление данных на экране компьютера определяется также как в таблице;

3. маска ввода – определяет ввод данных в поле «запросы»;

4. подпись поля. Заголовок столбца таблицы или подпись поля в карточке в вычисляемых полях задавать обязательно.

В нашем запросе для вычисляемого поля задаем описание объекта (пример: статус и название фирмы-поставщика) и подпись поля (поставщик).

Создание элемента управления «поле со списком» для ввода «код поставщика» в форме «Склад». Последовательность действий:

1) открываем форму «склад» в режиме конструктора, выделяем поле «код поставщика» и удаляем его;

2) на панели элементов нажимаем кнопку «поле со списком» и, удерживая ее нажатой, выбираем из списка полей «код поставщика», перетаскиваем его в форму в то место, где будет создан новый элемент управления. На экране появится окно диалога «создание поля со списком»;

3) Access предлагает выбрать значение из таблицы или запроса или вводить самостоятельно список значений. В нашем случае из таблицы или запроса. И нажимаем кнопку «далее»;

4) в следующем окне выбираем запрос для поля со списком «список поставщиков».

5) в следующем окне диалога пользователь выбирает поля, которые будут присутствовать в элементе управления «поле со списком». При этом следует учесть, что когда список раскрывается пользователь может видеть несколько столбцов с данными в этом элементе управления, но когда значение выбрано в форме остается значение только из одного столбца. Для нас более наглядно, если в форме останется значение выражения в запросе, в котором написано имя и статус фирмы поставщика. Поэтому выбираем сначала выражение, а затем код поставщика.

6) в следующем окне пользователь может задать ширину столбцов для элемента управления. Если ширину столбца сделать нулевой, то поле не будет видно в элементе управления. Мы можем сделать ширину столбца «код поставщика» нулевой, так как нам не обязательно знать какой код соответствует какой из фирм поставщиков, но мы должны именно значение этого поля вставить в таблицу «Склад», так как по значению именно этого поля происходит связь между таблицами «Склад» и «Поставщики».

7) в следующем окне диалога следует выбрать столбец значение которого следует использовать в базе данных. В нашем случае это, конечно же, «код поставщика».

8) в следующем окне диалога следует выбрать запоминаем мы значение, выбранное в «поле со списком», чтобы затем произвести с ним какие-нибудь действия или сохраняем его в поле таблицы. Мы сохраняем в поле таблицы «Склад» «код поставщика».

9) в последнем окне диалога можно задать подпись, которую будет иметь создаваемый элемент управления. Мы называем наш элемент «Поставщик».

Решение третьей задачи.

Строим элемент управления «поле со списком» для ввода названий товара.

Сначала строим итоговый запрос «Список товаров». Итоговый запрос «Список товаров» в режиме конструктора показан на рисунке 9.

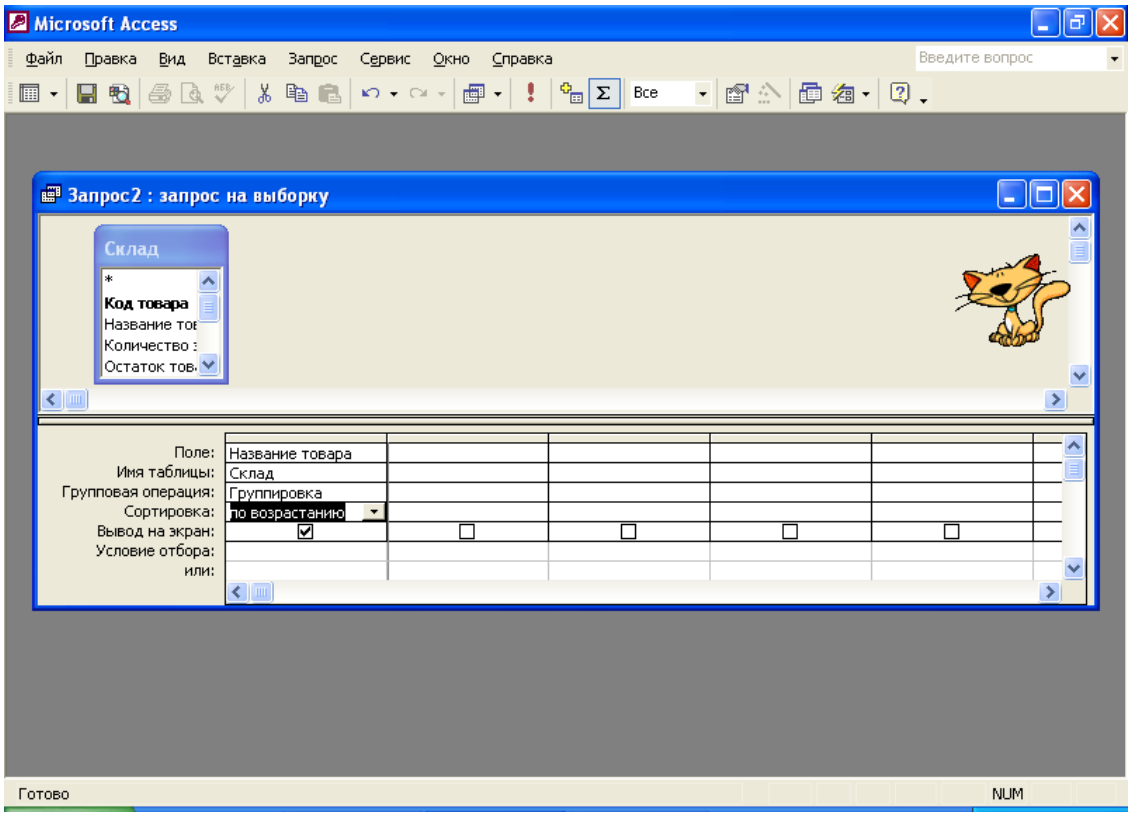

Рисунок 9 - Итоговый запрос «Список товаров» в режиме конструктора

Запрос строим на основе таблицы «Склад». Выбираем поле «Название товара» и добавляем строку групповая операция. В строке групповая операция выбираем действие «группировка». Если мы этого не сделаем, то названия товаров будут повторяться в элементе управления. Строка «групповая операция» добавляется кнопкой на панели инструментов. Так как названия товаров должны быть отсортированы, в строке сортировка выбираем значение «по возрастанию».

По созданному запросу строим в форме элемент управления «поле со списком» для ввода названий товаров. Строится аналогично тому, как был построен элемент управления для ввода «код поставщика». Окна диалога мастера такие же.

В итоговых запросах можно использовать следующие *групповые операции* – таблица 14.

Таблица 14 – Групповые операции

Групповая операция	Действие
SUM	Вычисляет сумму всех полей заданного поля в каждой группе. Можно использовать только для числовых или денежных полей.
Avg	Вычисляет среднее арифметическое всех значений данного поля в каждой группе. Функция может использоваться только для числовых или денежных полей, и не включает в вычисления значение Null.
Min	Возвращает наименьшее значение, найденное в этом поле внутри каждой группы. Для числовых полей возвращается наименьшее значение, для текстовых полей возвращается наименьшее символьное значение независимо от регистра. Access игнорирует значение Null.
Max	Возвращает наибольшее значение внутри поля в каждой группе. Для текстовых – наибольшее символьное значение независимо от регистра. Значение Null игнорируется.
Count	Возвращает число записей, в которых значение данного поля отличны от Null. Для того, чтобы подсчитать число записей в каждой группе с учетом значения Null, в строку «поле» вводится выражение: Count(*).
StDev	Подсчитывают статистическое стандартное отклонение для всех значений данного поля в каждой группе. Эту функцию можно применять только к числовым или денежным полям. Если в группе меньше двух строк, то Access возвращает значение Null.
Var	Подсчитывает статистическую дисперсию для всех значений данного поля по каждой группе. Эту функцию можно применять только к числовым или денежным полям. Если в группе меньше двух строк, то Access возвращает значение Null.
First **Last**	Возвращает первое значение поля в группе и последнее значение поля в группе.

6.6 Создание формы «Прием заказа»

Созданные нами ранее формы, позволяют заполнять данными таблицы «Кадры», «Клиенты», «Поставщики» и «Склад».

Внимание!!! Приступать к созданию формы «Прием заказа» следует только тогда, когда в эти таблицы внесены данные.

По форме «Прием заказа» мы должны будем заполнять данными таблицы «Заказы» и «Заказ-товар», фиксируя в этих таблицах данные о том:

— какая конкретная фирма-клиент делает заказ;

— когда он получен и должен быть выполнен;

— кто из сотрудников отвечает за его выполнение;

— какие конкретно товары, и в каком количестве проданы нами в оформляемый заказ.

Форма «Прием заказа» работает сразу с двумя таблицами и является сложной формой «один ко многим». В один заказ у нас может быть продано много товаров. Форма «Прием заказа» должна иметь вид, показанный на рисунке 10.

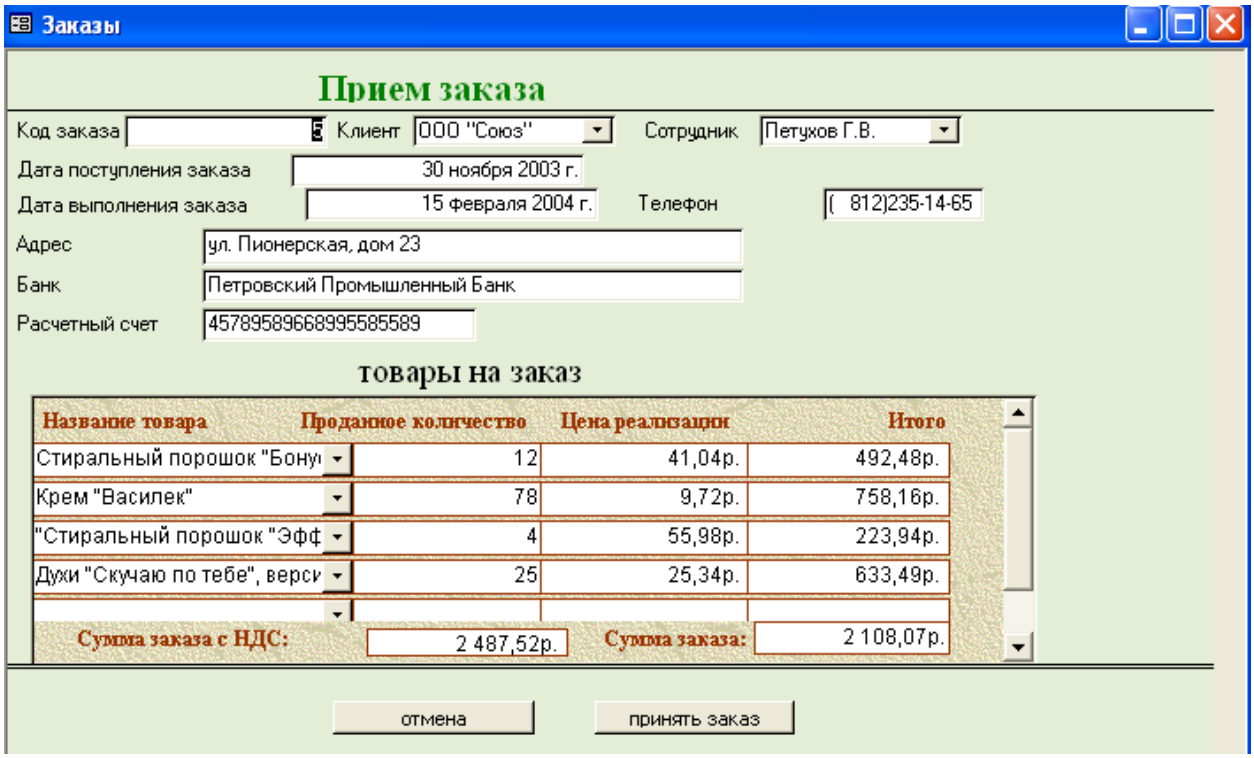

Рисунок 10 - Форма «Прием заказа»

Форма построена на основе двух форм:

— главной, в которой фиксируются данные о заказе и фирме, для которой этот заказ оформляется;

— подчиненной, в которой выбирается товар, продаваемый в этом заказе.

Внимание!!! В форме только две кнопки «Отмена» - эта кнопка выбирается пользователем в том случае, когда клиент передумал делать заказ товар в процессе его приема. Вторая «Принять заказ» - эта кнопка нажимается пользователем тогда, когда все данные о заказе внесены. При нажатии на эту кнопку значение поля «остаток» в таблице «Склад» должно быть уменьшено на количество товара, продаваемое в данном заказе.

Для удобства работы пользователя в форме должно быть реализовано следующее:

1. ввод данных о фирме-клиенте, для которой оформляется заказ, осуществляется с помощью элемента управления «поле со списком». При этом выбираем мы из списка, в котором видим статус и название фирмы-клиента, а после нашего выбора в таблицу «Заказы» попадает «код», выбранной пользователем фирмы-клиента. После того, как мы выбрали название фирмы, воспользовавшись элементом управления «поле со списком», автоматически заполняются поля с информацией об этой фирме.

2. ввод данных о сотруднике, отвечающем за заказ, осуществляется с помощью элемента управления «поле со списком». При этом выбираем мы из списка, в котором

видим фамилию и инициалы сотрудника отдела сбыта, а после нашего выбора в таблицу «Заказы» попадает «код», выбранного нами сотрудника.

3. элемент управления «поле» для ввода данных о дате выполнения заказа должен работать с маской ввода и форматом вывода.

4. ввод данных о названиях товаров, покупаемых в заказе, осуществляется с помощью элемента управления «поле со списком». При этом выбираем мы из списка, в котором видим не только название товара, но и остаток этого товара на складе, а также цену, по которой реализуется этот товар и единицы измерения для продаж. После нашего выбора в таблицу «Заказ-товар» попадает «код товара», выбранного нами для продажи. Как правильно должен выглядеть список показано на рисунке 11.

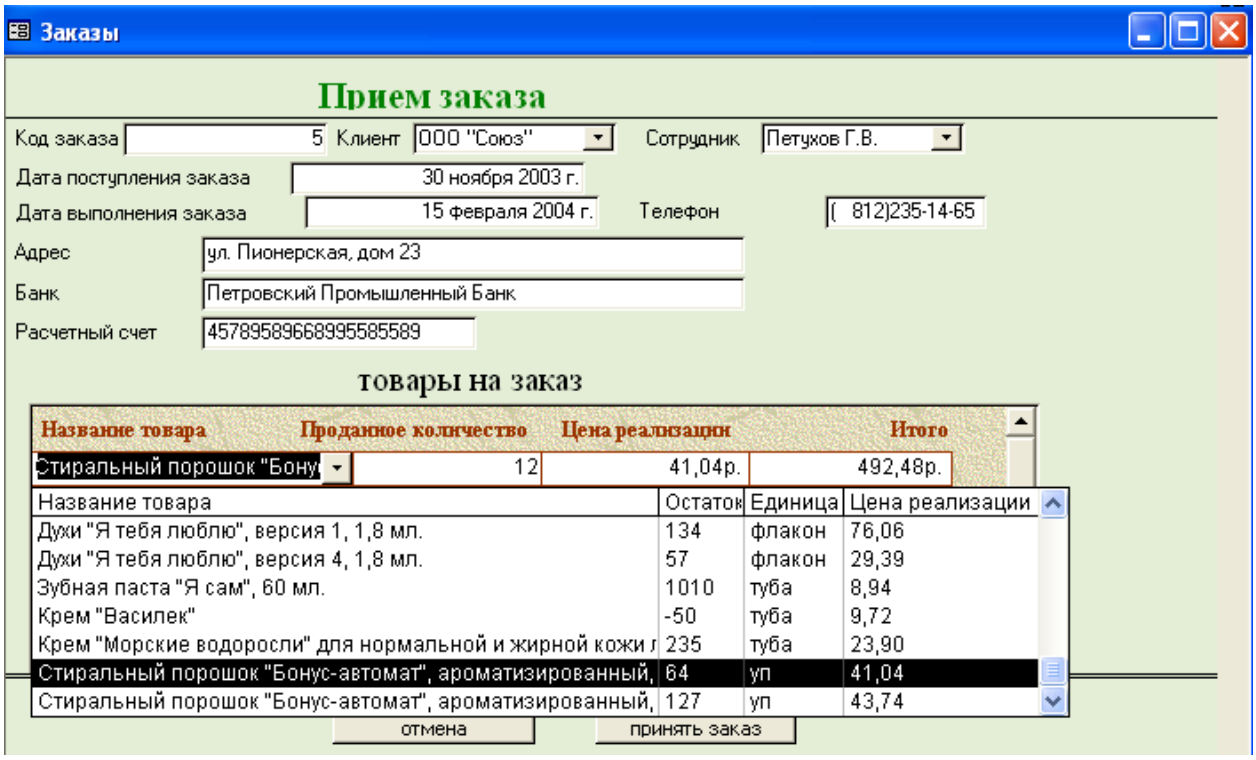

Рисунок 11 – Работа с формой «Прием заказа»

5. после заполнения поля «Проданное количество», автоматически должны заполняться поля «Цена реализации», «Итого», «Сумма заказа», «Сумма заказа с НДС».

6. по нажатию на кнопку «Принять заказ» - эта кнопка нажимается пользователем тогда, когда все данные о заказе и о товарах этого заказа внесены, значение поля «остаток» в таблице «Склад» для каждого товара, продаваемого в данном заказе, должно быть уменьшено на количество товара, продаваемое в данном заказе.

7. форма всегда должна открываться только на новую запись, иначе можно испортить данные о введенных в базу данных заказах.

Для реализации всего вышеперечисленного выполняем следующее:

Создаем запрос для главной формы «Прием заказа».

Так как в форме необходимо работать с данными из двух таблиц «Заказы» и «Клиенты», то и запрос строим на основе этих таблиц. Создание запроса в режиме конструктора показано на рисунке 12.

На рисунке трудно показать все выбранные поля, но из таблицы «Клиенты» надо выбрать все поля, участвующие в форме. Можно выбрать больше полей, чем в моем

примере и построить конкатенированное поле с адресом, в котором сконкатенированы «страна», «город» и «адрес».

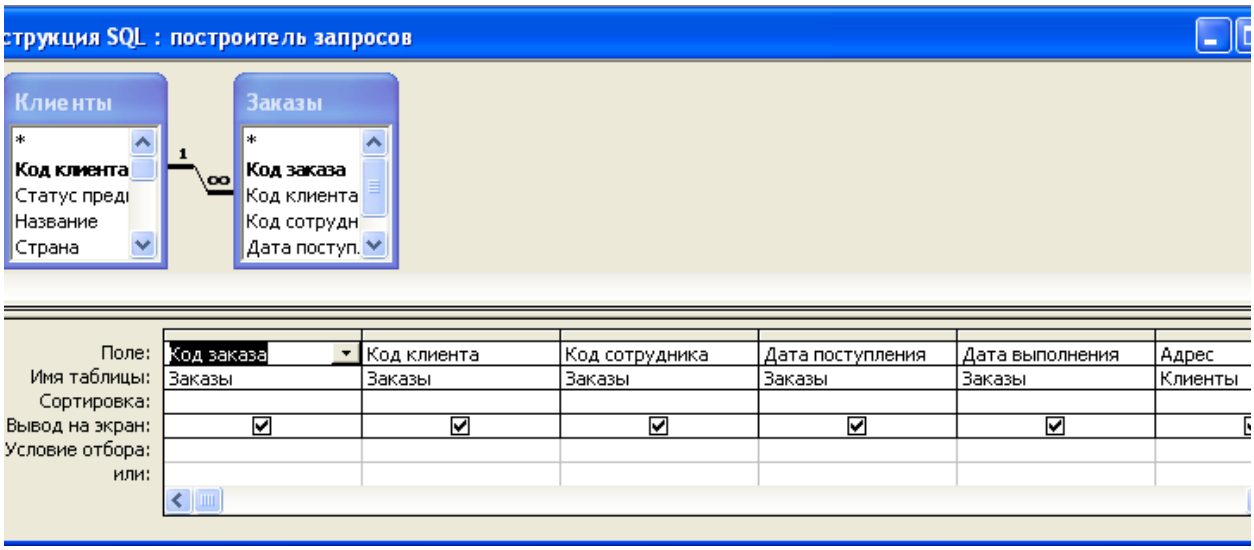

Рисунок 12 – Запрос для создания главной формы

По созданному запросу мастером создаем форму «Главная для формы «Прием заказов». Форму создаем в один столбец, а затем в режиме конструктора корректируем ее следующим образом:

— передвигаем поля в верхнюю часть формы, освобождая место для подчиненной формы;

— для полей «код клиента» и «код сотрудника» заменяем элементы управления «поле» на элементы управления «поле со списком».

Для создания элементов управления «поле со списком» создаем запросы «Список клиентов» и «Список сотрудников».

Запрос «Список клиентов» создается так же, как и запрос «Список поставщиков» для формы «Склад».

Запрос «Список сотрудников, обслуживающих заказы» в режиме конструктора показан на рисунке 13.

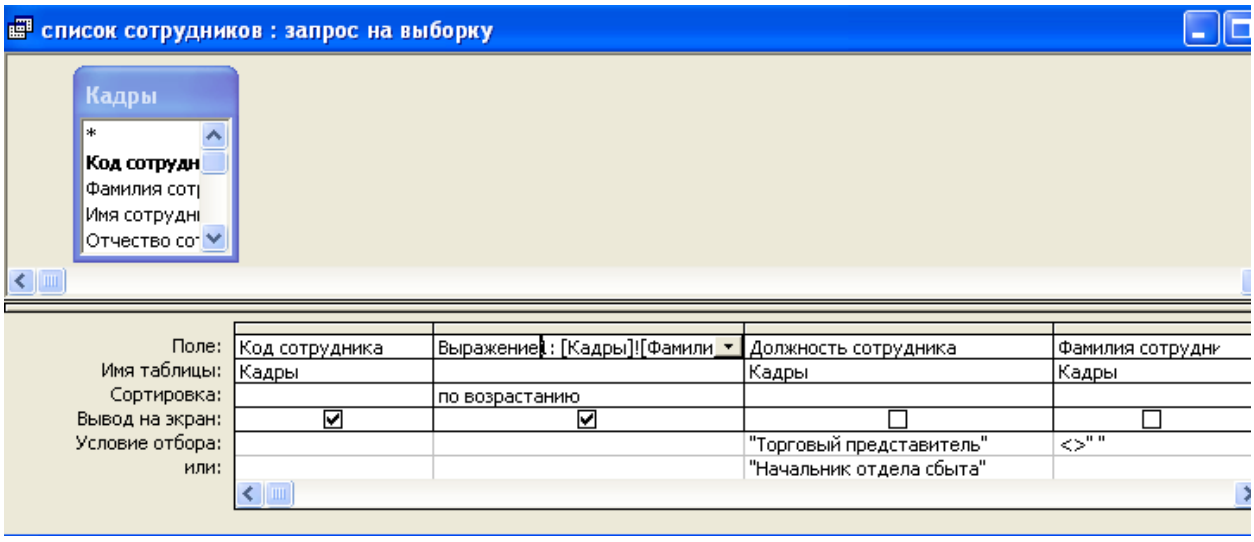

Рисунок 13 - Запрос «Список сотрудников, обслуживающих заказы»

Так как в списке должны быть только сотрудники «отдела сбыта» задаем условие отбора по названию должности. Выбираются только те сотрудники, которые работают в отделе сбыта. Можно построить запрос, в котором условие отбора задается по названию отдела, а не по названию должностей. Так как у нас есть вакантные должности, то, чтобы не выбирать «мертвых душ», а выбирать только оформленных сотрудников задаем для поля «Фамилия сотрудника» условие <> «_» (не равно пробелу). Для того, чтобы в списке фамилия сотрудника была написана с инициалами стоим выражение:

Выражение1: [Кадры]![Фамилия сотрудника] & " " & Left([Кадры]![Имя сотрудника];1) & "." & Left([Кадры]![Отчество сотрудника];1) & "."

Выражение строим с помощью построителя выражений и используем в нем функцию **Left.**

По созданным запросам строим элементы управления «поля со списками» также как делали это для поля «код поставщика» в форме «Склад».

Создаем запрос для подчиненной формы.

Так как в форме необходимо работать с данными из двух таблиц «Заказ-товар» и «Склад», то и запрос строим на основе этих таблиц. Создание запроса в режиме конструктора показано на рисунке 14.

Так как при работе с формой должны автоматически заполняться поля «Цена реализации» и «Итого». Строим построителем выражений в запросе два выражения:

Выражение1: [Склад]![Закупочная цена]*1,35

(Цена реализации больше, чем закупочная на 35%)

Выражение2: [Количество товара, купленного в данном заказе]*[Выражение1]

(поле «Итого»)

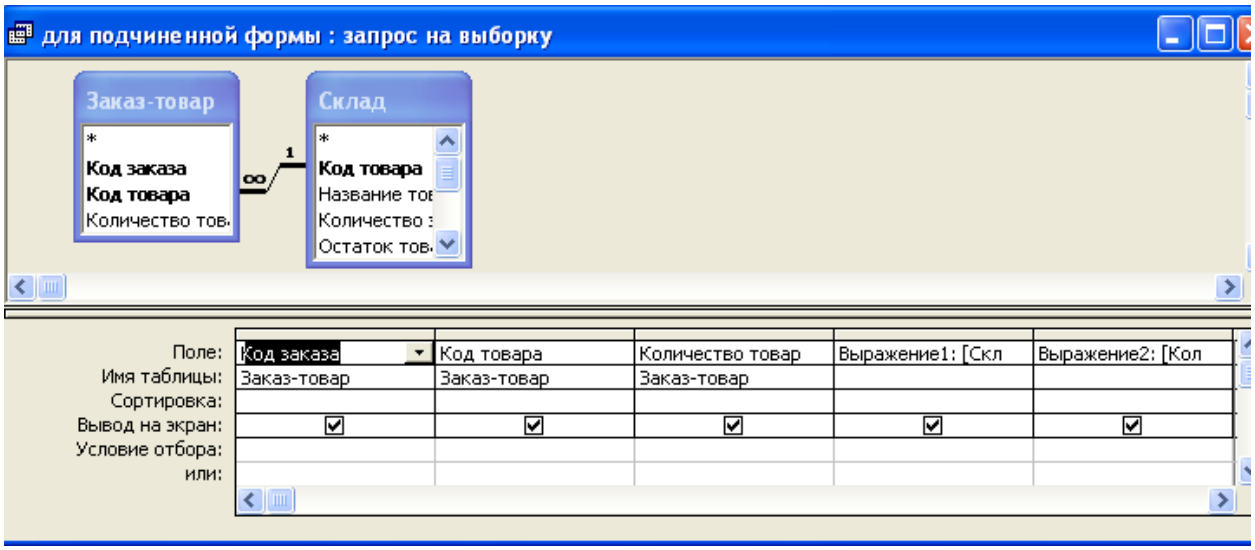

Рисунок 14 – Запрос для построения подчиненной формы

По созданному запросу стоим подчиненную форму. Строим мастером ленточную автоформу, затем корректируем ее.

Удаляем из формы поле «код заказа». Так как форма подчиненная, то будет работать в составе главной. А в главной форме уже есть «код заказа», таким образом, значение этого поля из главной формы и будет добавляться в таблицу «заказ-товар».

Для поля «код товара» заменяем элемент управления «поле» на элемент управления «поле со списком».

Для создания элемента управления «поле со списком» создаем запрос «Список товаров».

Запрос «Список товаров» в режиме конструктора показан на рисунке 15.

Мы не будем продавать товары, остаток которых на складе 0. Поэтому задаем условие отбора не равно 0. К тому же клиенты не должны знать закупочную цену, поэтому, чтобы сформировать цену реализации в списке строим выражение:

Выражение1: [Склад]![Закупочная цена]*1,35

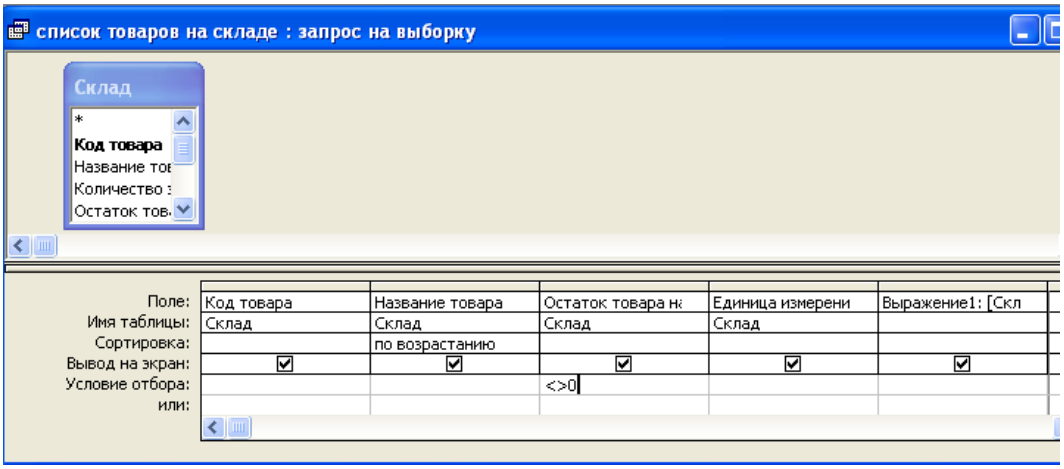

Рисунок 15 - Запрос «Список товаров» в режиме конструктора

На основе созданного запроса строим элемент управления «поле со списком» в подчиненной форме для ввода названий товара.

В области примечаний подчиненной формы создаем два свободных элемента управления поле для суммы заказа и суммы заказа с НДС.

Данные для этих элементов управления задаем построителем выражений.

=Sum([Выражение2]) (для поля «Сумма заказа»)

=[Поле12]*1,18 (для поля «Сумма заказа с НДС»)

Внимание!!! Номер поля зависит от того, какое оно по счету при разработке и у всех может быть разное. Это номер поля для свободного элемента «сумма заказа».

В главную форму добавим подчиненную и свяжем подчиненную форму и главную по полю код заказа.

Для работы кнопки «Принять заказ» **строим запрос действия для обновления поля остаток**. Запрос в режиме конструктора показан на рисунке 16.

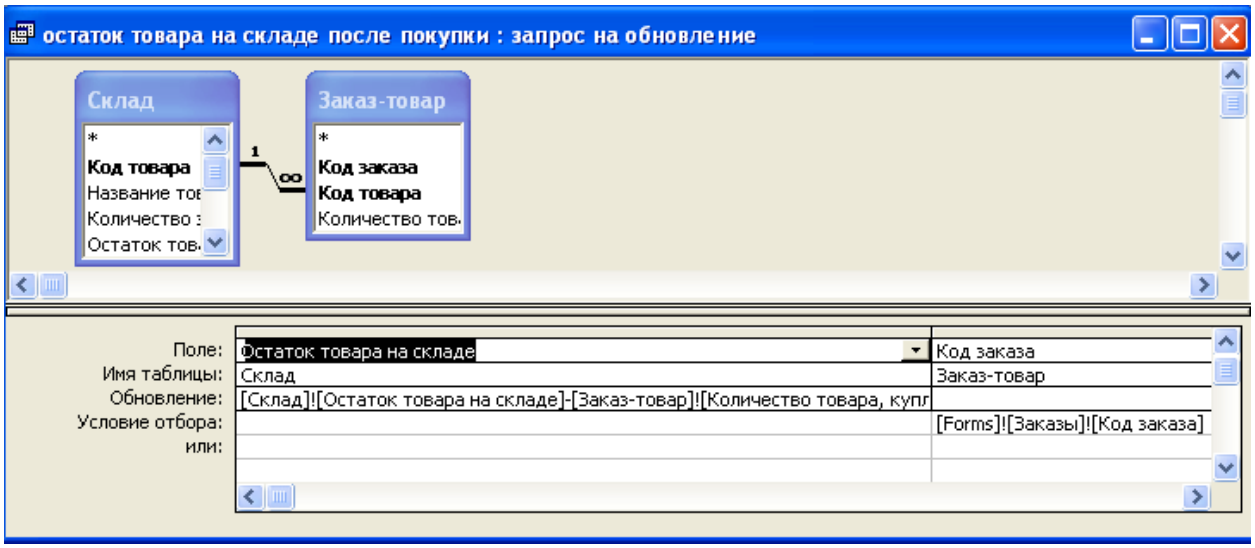

Рисунок 16 - Запрос действия для обновления поля остаток

Так как остатки должны обновляться только у тех товаров, которые куплены в оформляемом заказе, то условием отбора строим выражение:

[Forms]![Заказы]![Код заказа]

В поле обновления добавляем выражение, в котором от остатка отнимаем количество товара, проданное в данном заказе.

[Склад]![Остаток товара на складе]-[Заказ-товар]![Количество товара, купленного в данной заказе]

Созданный запрос должен работать только тогда, когда открыта форма «Прием заказа», иначе не выполнится условие отбора. Поэтому запрос запускается макросом.

Создание пользовательского интерфейса для разрабатываемой системы будет рассмотрено в дальнейшем.

Литература

1. Информатика и информационные технологии. Учебное пособие/ Под ред. Ю.Ю. Романова – М.: ЭКСМО, 2005.-544 с. – (Высшее экономическое образование).
2. Бугорский В.Н., Соколов Р.Е. Экономика и проектирование информационных систем. – СПб.: Питер, 1998.-340 с.
3. Введение в информационный бизнес/ Под ред. В.П. Тихомирова, А.В. Хорошилова. – М.: Финансы и статистика, 1996. – 239 с.
4. Вейскас Джон. Access 7.0 для Windows. – СПб.: Питер, 2003. – 850 с.
5. Винтер Рик. Microsoft Access 97: Справочник. – СПб.: Питер, 1999. – 411 с.
6. Грабауров В.А. Информационные технологии для менеджеров. – М.: Финансы и статистика, 2001. – 368 с.
7. Джонс Эдвард, Джонс Джарел. Access 97 Книга ответов. – СПб.: Питер, 1998. – 390 с.
8. Зегжда Д.П., Ивашко А.М. Как построить защищенную систему. – СПб.: НПО “Мир и семья”, 1997. - 290 с.
9. Карминский А.М., Нестеров П.В. Информатизация бизнеса. – М.: Финансы и статистика, 1997. – 415 с.
10. Мельников В. Защита информации в компьютерных системах. – М.: Финансы и статистика, 1997. – Москва «Финансы и статистика» 1997. – 364 с.
11. Морозов В.П., Тихомиров В.П., Хрусталев Е.Ю. Гипертексты в экономике. Информационная технология моделирования. – М.: Финансы и статистика, 1997. – 253 с.

Содержание