БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
Кафедра системного анализа и компьютерного моделирования

В. В. Скакун, И. С. Эйсмонт

МОДЕЛИ ДАННЫХ И СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Методические указания к лабораторным работам

Для студентов специальностей 1-98 01 01 «Компьютерная безопасность», 1-31 03 07 «Прикладная информатика»

МИНСК
2020

УДК 004.65(076.5)
ББК 32.973-018.2я7
С42

Рекомендовано советом факультета
радиофизики и компьютерных технологий
26 мая 2020 г., протокол № 8

Р е ц е н з е н т
кандидат физико-математических наук,
доцент *Н. Н. Яцков*

Скакун, В. В.

С42 Модели данных и системы управления базами данных : метод. указания к лабораторным работам / В. В. Скакун, И. С. Эйсмонт. – Минск : БГУ, 2020. – 101 с.

Методические указания предназначены для проведения лабораторных работ по курсу «Модели данных и системы управления базами данных». Содержится учебный материал по инфологическому и даталогическому проектированию реляционных баз данных, написанию запросов к базам данных, а также разработки баз данных средствами современных СУБД.

Предназначено для студентов факультета радиофизики и компьютерных технологий БГУ.

УДК 004.65(076.5)
ББК 32.973-018.2я7

ВВЕДЕНИЕ

Настоящее издание является учебно-методическим пособием для выполнения лабораторного практикума по курсу «Модели данных и СУБД», читаемого для студентов факультета радиофизики и компьютерных технологий. Цель данного пособия состоит в обучении студентов практическими навыками по моделированию предметной области и разработке реляционных баз данных средствами современных СУБД. В рамках практикума студентам предоставляется возможность овладеть навыками разработки клиент-серверной базы данных, начиная с инфологического моделирования предметной области и заканчивая программной реализацией полноценного интерфейса доступа к разработанной базе данных.

Выполняя задания к лабораторным работам, студенты знакомятся с принципами моделирования предметной области, навыками построения схем данных в стандарте функциональных диаграмм IDEF1X, навыками создания БД в СУБД SQL Server, получают развернутые знания по написанию запросов на языке SQL и овладевают навыками по созданию интерфейса БД в СУБД MS Access.

Приведенные в издании теоретические сведения содержат достаточный объем информации, позволяющий выполнять задания лабораторных работ, не обращаясь к другим литературным источникам. Задания к лабораторным работам не имеют привычных вариантов, вариабельность задач определяется свободным выбором студентами предметной области, гарантирующим уникальность получаемых на выходе разработок. В то же время для каждой лабораторной строго регламентирован перечень минимальных требований.

САМОСТОЯТЕЛЬНАЯ РАБОТА № 1

РАЗРАБОТКА СХЕМЫ ДАННЫХ

Цель работы. Проведение инфологического моделирования предметной области и разработки схемы данных.

Краткие сведения

Инфологическое моделирование. Этап инфологического моделирования заключается в разработке концептуальной модели предметной области, представляющей объекты и их взаимосвязи без указания способов их физического хранения. Инфологическое проектирование связано прежде всего с попыткой представления семантики предметной области в модели БД. Усилия на этом этапе должны быть направлены на структуризацию данных и выявление взаимосвязей между ними. Результатом проектирования является схема данных. Этот процесс можно разбить еще на несколько под-этапов:

Уточнение задачи. Перед началом работы над конкретным приложением разработчик должен явно представлять то, что он будет разрабатывать. В случаях, когда разрабатывается небольшая персональная БД, такие представления могут быть достаточно полными. В других случаях, когда разрабатывается большая БД под заказ, таких представлений или может быть очень мало, или они будут поверхностными. Сразу начинать разработку с определения таблиц и связей между ними будет явно рановато. Такой подход в большинстве случаев приведет к полной переделке приложения. Поэтому следует затратить некоторое время на составление списка всех основных задач, которые должны решаться этим приложением, включая и те, которые могут возникнуть в будущем.

Уточнение последовательности выполнения задач. Чтобы приложение работало логично и удобно, лучше всего объединить основные задачи в группы и затем упорядочить задачи каждой группы так, чтобы они располагались в порядке их выполнения. Группировка и графическое представление последовательности их выполнения поможет определить естественный порядок выполнения задач.

Анализ данных. После определения списка задач необходимо для каждой задачи составить полный перечень данных, требуемых для ее решения. После анализа данных можно приступать к разработке концептуальной модели, т. е. к выделению объектов, их атрибутов и связей между объ-

ектами. Для разработки концептуальной модели чаще всего используется модель «сущность-связь».

Даталогическое моделирование. Этап даталогического моделирования заключается в трансляции концептуальной модели в логическую модель, реализуемую некоторой СУБД. Для этого следует руководствоваться требованиями, налагаемым конкретной логической моделью данных, например реляционной. Нереализуемые виды атрибутов преобразуются в простые, а связи высоких порядков разбиваются на поддерживаемые виды связей.

Виды связей. В общем случае связи можно классифицировать на унарные, бинарные, тернарные и связи более высоких порядков. Унарная связь позволяет отражать иерархию отношений подчиненности. Наибольший интерес представляют бинарные связи, так как связи более высоких порядков редко встречаются, труднореализуемы, нелегки для понимания и всегда могут быть разбиты на совокупность бинарных связей.

Первый тип связи – «один к одному» (one to one, 1:1). Эта связь означает, что одному экземпляру объекта А может соответствовать один и только один экземпляр связанного с ним объекта В. Напомним, количество экземпляров объекта, участвующих в связи, называется показателем кардинальности. На практике такой вид связи реализуют редко, так как почти всегда два связываемых объекта можно объединить в один. В качестве примера использования такой связи может послужить связь между персональной и служебной характеристикой работника некоторой фирмы. В реляционной БД связь «один к одному» устанавливается тогда, когда в связи участвуют ключевые поля с обеих сторон.

Второй тип связи – «один ко многим» (one to many, 1:M). Это означает, что одному экземпляру объекта А соответствует несколько (много) экземпляров объекта В. Например: сведения о клиентах – записи о покупках; сведения о товаре – записи о наличии его на складе. Один клиент может сделать много покупок, а один и тот же товар может быть завезен многократно. Вид связи «многие к одному» является обратимым к виду «один ко многим», поэтому на практике обычно говорят о связи «один ко многим». Технология соединения таблиц в реляционной БД связью «один ко многим» следующая. Вначале определяется подчиненная (зависимая) таблица и в нее добавляется внешний ключ. Затем устанавливается связь между введенным внешним ключом подчиненной таблицы и первичным ключом базовой (главной) таблицы.

Следует подчеркнуть, что не следует пытаться «заставить» реляционную СУБД создать требуемый для вас вид связи. Вид связи назначается самой СУБД посредством анализа полей, участвующих в связи. Если, с

одной стороны, в связи участвует первичный ключ, то показатель кардинальности на его стороне будет тождественно равен единице, так как повторение значений в первичном ключе исключено. Если же в связи участвует внешний ключ, то он может принимать много значений. Примечание: если на внешний ключ дополнительно наложено ограничение уникальности, то вид связи получится «один к одному», так как возможность повторения значений во внешнем ключе также будет исключена.

Четвертый тип связи – «многие ко многим» (many to number, N:M). Это означает, что одному экземпляру объекта А соответствует несколько (много) экземпляров объекта В, и наоборот, каждому экземпляру объекта В может соответствовать несколько экземпляров объекта А. Например, сведения об авторах и написанных ими книгах или отношение между клиентами и товарами. Книга может быть написана несколькими авторами и, наоборот, один автор может написать много книг. Каждый товар может быть куплен несколькими клиентами и, наоборот, один клиент может купить несколько товаров и т.д.

В реляционной модели такая связь может быть реализована с помощью дополнительной таблицы, содержащей ключевые поля обеих таблиц, участвующих в связи (являющимися для нее внешними ключами). Эту вспомогательную соединяющую таблицу можно дополнить другими полями. В примере ниже соединяющая таблица превращается в полноценную таблицу хранения заказов.

При определении типа связи следует быть очень внимательным, поскольку лучше перестраховаться и выбрать более универсальный тип связи (вместо связи «один к одному» выбрать связь типа «один ко многим», взамен связи «один ко многим» выбрать связь «многие ко многим»), поскольку в противном случае реализовать «недосмотренную» связь впоследствии будет невозможно. Стоит также удержаться от соблазна всегда использовать наиболее универсальный тип связи «многие ко многим», поскольку, как мы видели из примера выше, реализация данного типа свя-

зи связана с рядом трудностей (ввод дополнительной связующей таблицы).

Зависимости между атрибутами. Наряду со связями между объектами следует выделять также зависимости между атрибутами объектов. Зависимости бывают *функциональные*, *транзитивные* и *многозначные*. Понятие функциональной зависимости является базовым, так как на ее основе формулируются определения других видов зависимостей. Атрибут B функционально зависит от атрибута A, если каждому значению атрибута A соответствует одно и только одно значение атрибута B. То есть если нам известно значение атрибута A (его иначе называют детерминантом), то мы однозначно можем определить значение атрибута B. Однако данному значению атрибута B может соответствовать несколько различных значений атрибута A. Математически функциональная зависимость B от A обозначается записью $A \to B$. Необходимо отметить, что A и B могут быть составными, т. е. состоять из двух и более атрибутов.

Функциональная зависимость может быть полной либо частичной. Частичной зависимостью (частичной функциональной зависимостью) называется зависимость атрибута B от части составного атрибута A. Полная функциональная зависимость определяется зависимостью атрибута B от всего составного атрибута A.

Атрибут C зависит от атрибута A транзитивно $A \to\to C$ (существует транзитивная зависимость), если для атрибутов A, B, C выполняются условия $A \to B$ и $B \to C$, но обратная зависимость отсутствует. Например: ФИО $\to$ Должность $\to$ Оклад. Поскольку оклад формируется по тарифной сетке, зависящей от должности, и может меняться по указам правительства, он не напрямую определятся знанием ФИО.

Между атрибутами может иметь место и многозначная зависимость. Атрибут B многозначно зависит от атрибута A, если некоторому значению A соответствует множество значений B. Многозначные зависимости могут образовывать связи «один ко многим», «многие к одному» или «многие ко многим», обозначаемые соответственно A «– B, A –» B и A «–» B. Например, информация о студентах и сдаваемыми ими предметами хранится совместно. Тогда один студент обязан сдать несколько экзаменов, а один предмет обязаны сдать вообще все студенты, обучающиеся в одной группе при условии, что все они обучаются по одной специальности (многие ко многим). Различают тривиальные и нетривиальные многозначные зависимости. *Тривиальной* называется многозначная зависимость X –» Y, для которой $Y \subset X$ или $Y \cup X = R$, где R – рассматриваемое множество данных. Если хотя бы одно из двух этих условий не выполняется (т.е. Y не

является подмножеством X или $Y \cup X$ состоит не из всех атрибутов R), то такая многозначная зависимость называется нетривиальной. Пример со студентами и изучаемыми ими предметами является примером нетривиальной многозначной зависимости.

Проектирование концептуальной модели можно провести на основе анализа существующих зависимостей между атрибутами. Понятно, что все атрибуты, характеризующиеся полной функциональной зависимостью от ключевого атрибута, конечно, будут атрибутами объекта одного типа. Те атрибуты, которые характеризуются транзитивной зависимостью от ключевого атрибута, необходимо выделить и сформировать отдельные объекты. Многозначные зависимости также требуют дополнительного разбиения по объектам. Практически процесс формирования объектов на основе анализа зависимостей между атрибутами необходимо продолжать до тех пор, пока все неключевые атрибуты будут характеризоваться только полной функциональной зависимостью от соответствующих ключевых атрибутов (детерминантов). На первом же шаге проектирования можно значительно сократить множество анализируемых атрибутов, исключив тривиальные зависимости, т. е. те зависимости, которые не могут не выполняться. Атрибуты, связанные тривиальной зависимостью, например фамилия имя и отчество, необходимо всегда рассматривать вместе.

Нормализация данных в реляционной модели. Один и тот же набор данных в реляционной модели можно представить различными способами. Процесс нормализации данных позволяет решить вопрос о наиболее эффективной их структуре, обладающей минимальной избыточностью.

Теория нормализации оперирует с пятью нормальными формами и одной промежуточной, уточняющей третью. На практике обычно руководствуются первыми тремя нормальными формами. Процесс проектирования БД с использованием нормальных форм является итерационным и заключается в последовательном применении правил нормальных форм от низшей к высшей. Каждая следующая нормальная форма уменьшает избыточность данных и сохраняет свойства всех предыдущих форм. Перевод отношения в следующую нормальную форму осуществляется методом «декомпозиции без потерь», т. е. разбиением исходной таблицы на несколько связанных. Такая декомпозиция должна обеспечить равенство результатов выборок из исходного отношения и выборок, основанных на совокупности полученных отношений, т. е. говоря математическим языком, должно соблюдаться следующее правило: $\bigcup_{i=1}^{m} R_i = R,$ где

$R = \{A_1, A_2, \text{K}, A_n\}$ – исходное отношение, а $D = \{R_1, R_2, \text{K}, R_m\}$ – декомпозиция, т. е. множество нормализованных отношений.

Первая нормальная форма требует соответствия исходной таблицы требованиям, предъявляемым к отношениям, т. е.:

1) таблица не должна иметь повторяющихся записей;

2) все атрибуты должны быть простыми (скалярными).

Первое требование легко решается введением ключевого поля, однозначно определяющего остальные поля таблицы. Ключевым полем может быть как некоторый атрибут или совокупность атрибутов объекта, обладающих условием уникальности, например, код товара, код клиента и дата покупки, так и искусственно введенный идентификатор (табельный номер). Второе требование постулирует, чтобы в каждой ячейке было представлено только одно значение, а не массив или перечисление. Составные поля должны быть разложены на простые, а многозначные – вынесены в отдельные таблицы.

Таблица находится во второй нормальной форме, если:

1) она удовлетворяет условиям первой нормальной формы;

2) любое поле, не входящее в ключ, должно однозначно и полно идентифицироваться значением первичного ключа. Если первичный ключ является составным, то остальные поля должны зависеть от его полного выражения, а не от части (неключевые атрибуты должны характеризоваться полной функциональной зависимостью от первичного ключа).

Таблица находится в третьей нормальной форме, если:

1) она удовлетворяет условиям второй нормальной формы;

2) ни одно из полей таблицы, не входящих в ключ, не должно идентифицироваться с помощью другого поля, не входящего в ключ (иначе, отношение не должно иметь неключевых атрибутов, которые бы находились в транзитивной зависимости от первичного ключа).

Сведение таблицы к третьей нормальной форме предполагает ее разделение с целью помещения в отдельную таблицу (или несколько таблиц) столбцов, которые не зависят напрямую от выражения первичного ключа. В результате такого разбиения каждое из полей, не входящих в первичный ключ, должно оказаться независимым от какого-либо другого неключевого поля. Существует третья усиленная нормальная форма - форма Бойса – Кодда (BCNF), дополнительно постулирующая отсутствие зависимости атрибутов составного ключа от не ключевых атрибутов.

Четвертая нормальная форма устраняет нетривиальные многозначные зависимости. Отношение находится в четвертой нормальной форме, если оно находится в BCNF и в нем отсутствуют нетривиальные многозначные зависимости. Пятая (проективно-соединительная) нормальная

форма основывается на концепции устранения зависимостей объединения, возникающих после декомпозиции при устранении многозначных зависимостей.

Методология IDEF1X. Для построения реляционной схемы данных наиболее часто используют методологию IDEF1X. Методология IDEF1X – один из подходов к семантическому моделированию данных, основанный на модели сущность-связь и позволяющий построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. Является одним из стандартов моделирования сложных систем, предложенных в семействе методологий IDEF (Integrated DEFinition). Всего было предложено 15 стандартов, IDEF0 – IDEF14. Для справки, IDEF0 – это методология функционального моделирования, где изучаемая система предстает в виде набора взаимосвязанных функциональных блоков. IDEF1X (IDEF1 eXtended) является расширением стандарта IDEF1. В настоящее время в рамках этой методологии предложено несколько вариантов графического отображения объектов (нотаций), в большей степени отличающихся по виду представления связей между таблицами.

В IDEF1X объекты реального мира представляются сущностями, аналогично модели сущность-связь. Каждая сущность имеет имя и некоторый набор атрибутов. Независимые сущности изображаются прямоугольником с прямыми углами, а зависимые – прямоугольником с закруглёнными углами. Ключевые атрибуты записываются вверху списка атрибутов и отделяются от неключевых атрибутов линией. Необязательные к заполнению атрибуты помечаются символом (O – optional, не обязательный к заполнению). Сущности связываются посредством связей. Сами сущности могут быть зависимыми и независимыми, а связи – идентифицирующими (между независимой и зависимой сущностями) или неидентифицирующими (дочерняя сущность остается независимой). В случае идентифицирующей связи дочерняя сущность идентифицируется с помощью первичного ключа родительской сущности, который мигрирует и в атрибуты первичного ключа дочерней сущности (внешний ключ является частью первичного ключа дочерней сущности). В неидентифицирующей связи внешний ключ, связывающий главную сущность с подчиненной, не входит в состав первичного ключа подчинённой сущности.

Связи в стандартной нотации изображаются прямой линией с точкой на конце. Мощность связи (какое количество экземпляров сущности-потомка может существовать для сущности-родителя) указывается рядом. Идентифицирующая связь изображается сплошной линией, неидентифицирующая – пунктирной. Наиболее характерными типами связей между сущностями являются:

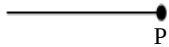
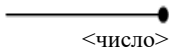
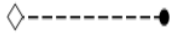
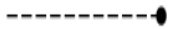
- связи типа «часть–целое», (например, «состоит из», «включает»);
- классифицирующие связи (например, «тип – подтип», «множество – элемент», «общее – частное» и т. п.);
- производственные связи (например, «начальник–подчиненный»);
- функциональные связи, определяемые обычно глаголами «производит», «влияет», «зависит от», «вычисляется по» и т. п.

Связь характеризуется следующим набором параметров:

- именем – указывается в виде глагола и определяет семантику (смысл) связи;
- мощность (кратность, кардинальность): один-к-одному (1:1), один-ко-многим (1:M) и многие-ко-многим (N:M).
- типом: идентифицирующая и неидентифицирующая;
- обязательностью: обязательная (при вводе нового экземпляра в дочернюю сущность заполнение атрибутов внешнего ключа обязательно) и необязательная. Необязательность обозначается ромбиком.

Внешний вид связи на диаграммах IDEF1X указывает на ее мощность, тип и обязательность. Типы связей представлены в таблице 1.

Таблица 1. Типы связей в методологии IDEF1X

Внешний вид	**Тип и обязательность связи**	**Мощность связи справа**
	Обязательная, идентифицирующая	1
	Обязательная, идентифицирующая	0 .. ∞
 Z	Обязательная, идентифицирующая	0 или 1
 P	Обязательная, идентифицирующая	1 .. ∞
 <число>	Обязательная, идентифицирующая	<число>
	Необязательная, неидентифицирующая	0 .. ∞
	Обязательная, неидентифицирующая	0 .. ∞

Пример IDEF1X диаграммы, разработанной для некоторого торгового предприятия, приведен на рис. С1.1. Связи клиентов и товаров с заказами являются идентифицирующие, а товаров с категориями – неиденти-

фицирующими. Код категории объявлен необязательным к заполнению полем.

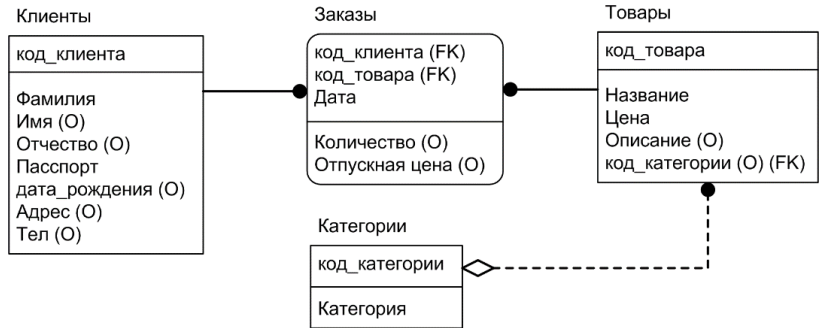


Рис. С1.1. Пример IDEF1X диаграммы в стандартной нотации.

Существует несколько нотаций отображения связей между сущностями. Кроме стандартной, широкое применения получила нотация «воронья лапка», в английском «crow's feet». Мощность связи отображается как показано на рис. 2, пример IDEF1X диаграммы приведен на рис. 3.

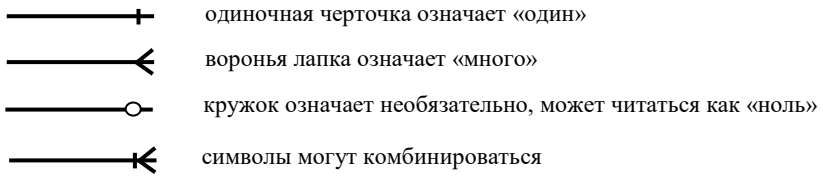


Рис. С1.2. Мощность связи в нотации crow's feet

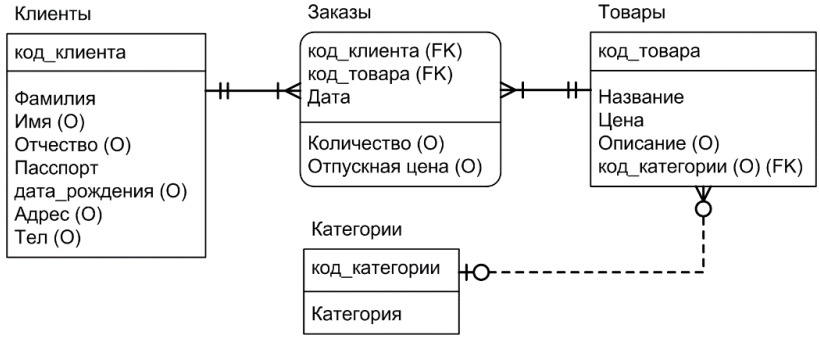


Рис. 3. Пример IDEF1X диаграммы с нотацией crow's feet

При разработке реляционной модели с использованием методологии IDEF1X должны быть выполнены следующие этапы (или требования):

1. Устранение элементов, не отвечающих принятой модели данных:
   - устранение связей N:M (путем ввода промежуточной сущности);
   - устранение связей, имеющих атрибуты (связи с атрибутами должны быть преобразованы в отдельные сущности);
   - устранение сложных связей (тернарных и выше);
   - устранение унарных связей (путем ввода копии сущности и таким образом преобразования унарной связи и бинарную);
   - устранение многозначных атрибутов (путем введения новой сущности и связи 1:N);
   - устранение избыточных связей.
2. Проверка модели с помощью правил нормализации;
3. Определение требований поддержки целостности данных.

Методология IDEF1X нашла широкое применение в автоматизированных системах разработки приложений, так называемых CASE (Computer Aided Software Engineering) систем. Автоматизированные системы разработки приложений представляют собой программные средства, поддерживающие процессы создания и сопровождения информационных систем, такие как анализ и формулировка требований, проектирование приложений, генерация кода, создание развертываемых систем (инсталляторов), тестирование, управление конфигурацией и проектом. Основная цель CASE систем состоит в том, чтобы отделить процесс проектирования программного обеспечения от его кодирования и последующих этапов разработки (тестирование, документирование и т. д.), а также автоматизировать весь процесс создания программных систем.

Примерами CASE систем могут служить ERWin (Logic Works), Rational Rose (Rational Software), PowerDesigner (Sybase), DataBase Designer, Developer/Designer 2000 (Oracle Corp.), MySQL Workbench и Microsoft Visio. Интерфейс последней представлен на Рис. 4.

Как видно из рисунка (левая панель), система предоставляет набор графических примитивов (сущность, связь, категория) и набор средств для определения первичных и внешних ключей, вида связи и т.д., позволяющих создавать сложные схемы за сравнительно небольшое время.

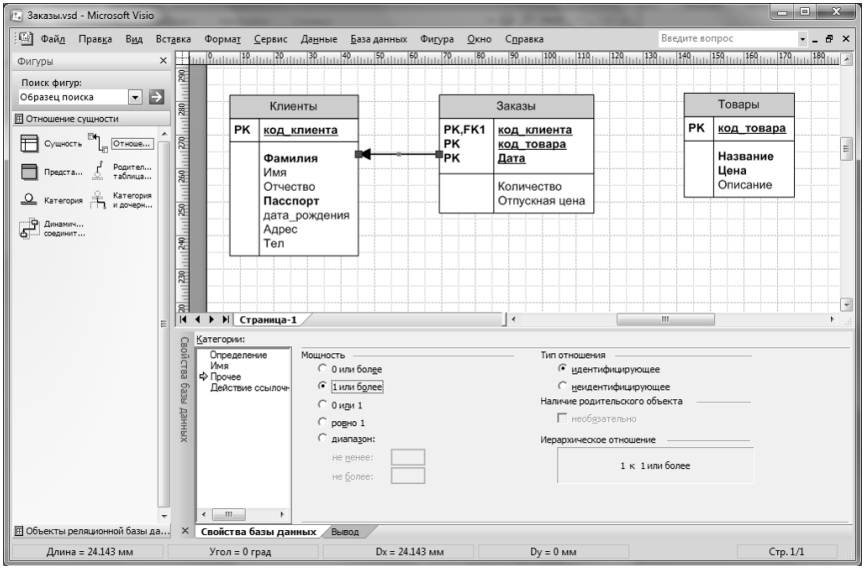


Рис. С1.4. Интерфейс CASE системы Microsoft Visio

Задания

1. Определить цель БД и задачи, которые должна решать БД;
2. Провести анализ данных требуемых для решения поставленных задач;
3. Разработать схему данных (реляционная модель). Требование – минимум три таблицы, связанные отношениями один ко многим. Также требуется определить по крайней мере один атрибут, допускающий форматированный ввод данных, например, номер паспорта, телефона или типа подобное;
4. Провести нормализацию данных. Учесть требования первых трех нормальных форм;
5. Разработать IDEF1X диаграмму с помощью CASE (например MS Visio, MySQL Workbench и другие) системы;
6. Результаты оформить в виде отчета, где должны быть представлены: цели и задачи, решаемые БД; конечная схема данных в стандарте IDEF1X.

Форма отчета. Формой отчета является файл со схемой данных выбранной тематики, реализующая задания, представленные в разделе «Задания».

ЛАБОРАТОРНАЯ РАБОТА № 1

РАЗРАБОТКА ТАБЛИЦ В СУБД SQL SERVER

Цель работы. Практическое освоение разработки таблиц БД в СУБД MS SQL Server.

Краткие сведения

Разработка таблиц в SQL Server Management Studio. Разработку БД в СУБД SQL Server можно выполнить с помощью графического редактора SQL Management Studio (лучше всего использовать версию 2017, т.к. в более поздних отсутствует построитель схемы данных). После запуска Management Studio первым делом нужно установить соединение с сервером. Для этого служит окно Connect to Server (рис.1.1). Имя сервера задается в виде Имя компьютера\имя экземпляра сервера. Примечание: если используется полная, не Express, версия сервера, то следует в поле Server name указывать только имя компьютера. По умолчанию имя экземпляра для экспресс-версии равно SQLEXPRESS. Обычно используется режим аутентификации Windows. Если есть необходимость использовать смешанный режим аутентификации, то дополнительно указывается имя пользователя и пароль.

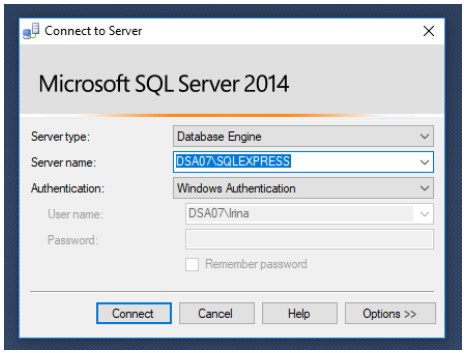


Рис. 1.1. Окно соединения с сервером

После установления соединения с сервером можно приступить к созданию базы данных и определения таблиц. Создать новую базу данных можно с помощью вызова контекстного меню для объекта Database и выбора пункта меню New Database. Название БД можно дать любое, но луч-

ше состоящее из латинских букв, цифр, символа подчеркивания и без пробелов (рис. 1.2).

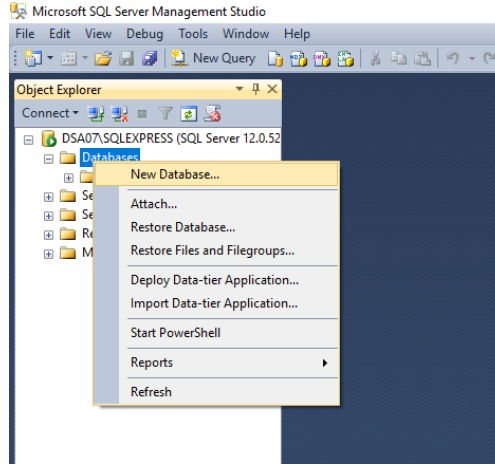


Рис. 1.2. Создание БД

Когда база данных создана, можно приступать к созданию таблиц, (см. рис. 1.3). Имена таблиц могут быть набраны как в латинской кодировке, так и в русской. Допустимо наличие пробелов в названии таблиц, но в этом случае при обращении к ним придется заключать их в квадратные скобки [].

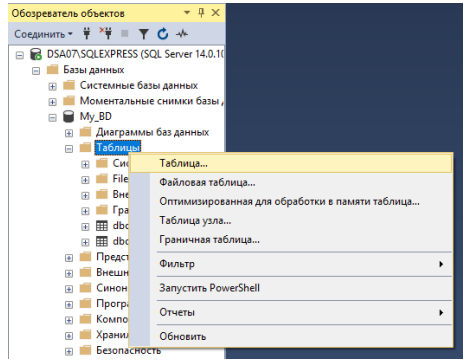


Рис. 1.3. Создание таблиц

Примечание! Если система не позволяет сохранить изменения после редактирования структуры таблицы, выдавая следующую ошибку (см.

рис. 1.4), то необходимо зайти в меню *Сервис -> Параметры -> Конструкторы* и снять отметку с пункта «Запретить сохранение изменений, требующих повторного создания таблицы» (рис. 1.5).

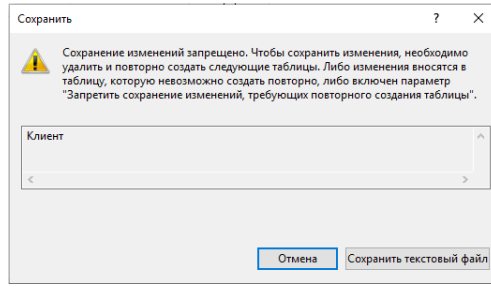


Рис. 1.4. Сообщение о невозможности сохранить таблицу

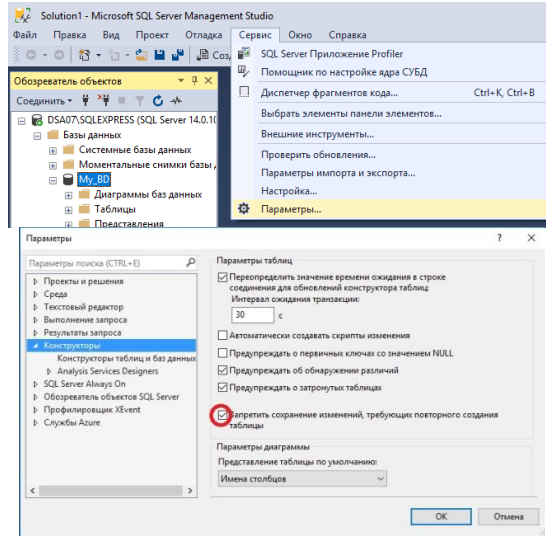


Рис. 1.5. Настройки среды для устранения ошибки при сохранении таблиц

Данный запрет введен для повышения безопасности работы с уже введенными в эксплуатацию БД, но при разработке новой БД является излишним. Эту настройку достаточно сделать один раз в самом начале работы над базой данных.

Режимы работы с таблицами. В контекстном меню для таблиц есть несколько режимов их открытия (см. рис. 1.6).

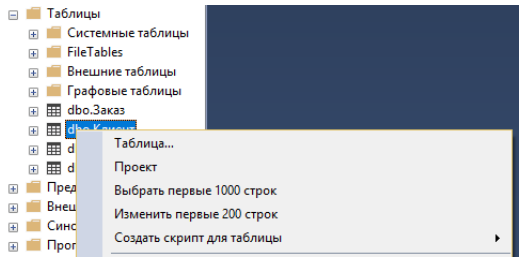


Рис. 1.6. Режимы открытия таблиц

Режим *Таблица* создает новую таблицу независимо от того, для какой таблицы было вызвано контекстное меню.

Режим *Проект* открывает таблицу в режиме разработки. В данном режиме появляется возможность задать названия полей, типы данных, определить обязательные поля, условия на значение, правила ссылочной целостности, значения по умолчанию, создать индексы и т.д.) (рис. 1.7).

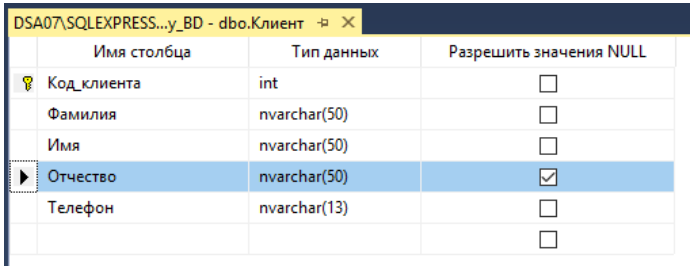


Рис. 1.7. Режим проект

Режим *Выбрать первые 1000 строк* позволяет выбрать первые 1000 записей БД. В верхней половине окне можно увидеть запрос выборки, а в нижней – сам результат выборки (см. рис. 1.8). Если этот запрос видоизменить и выполнить, то соответственно поменяются результат выборки. Этот режим удобен, когда в БД содержится много записей, а разработчик хочет просмотреть их, не создавая новое окно для ввода запроса.

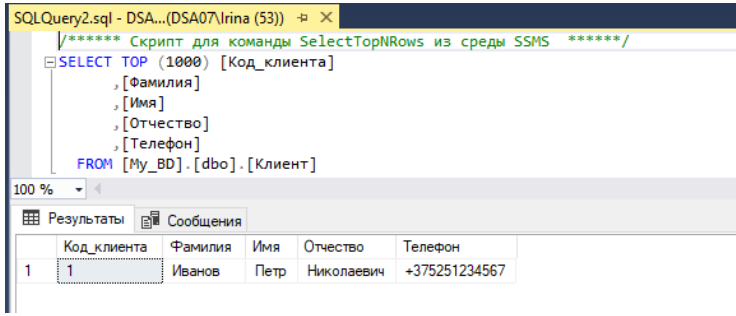


Рис. 1.8. Режим Выбрать первые 1000 строк

Примечание: этот режим можно использовать для автоматической генерации запроса выборки для данной таблицы.

Режим *Изменить первые 200 строк* позволяет вводить данные в БД и производить редактирование записей (см. рис.1.9). Разработчику он необходим для тестирования проверочных ограничений на их корректность. Запись не будет сохранена, пока не будут выполнены все ограничители целостности.

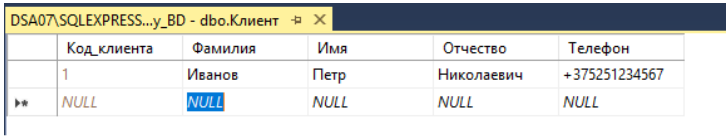


Рис. 1.9. Режим Изменить первые 200 строк

Пункт контекстного меню *Создать скрипт* позволяет создавать шаблоны для некоторых видов запросов.

Определение обязательных полей. В каждой таблице желательно иметь одно или несколько обязательных для заполнения полей, имеющих важное значение в семантическом смысле. Это гарантирует целостность записи, поскольку пользователь не сможет сохранить пустую запись, не имеющую никакого смысла.

Для задания поля обязательным к заполнению следует убрать птичку в столбце *Разрешить значения NULL* в режиме *Проекта* (рис. 1.10). В данном примере поле Отчество необязательное, а все остальные обязательные к заполнению.

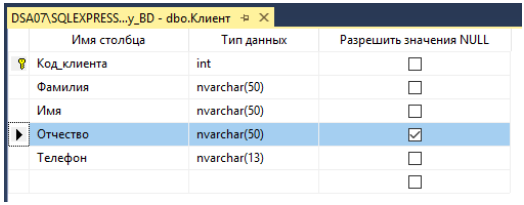


Рис. 1.10. Выбор обязательных полей

Задание первичного ключа. В каждой таблице должен быть определен первичный ключ. Как правило, это поле типа int с соответствующим названием (Код_… или id). При необходимости можно использовать и другие уникальные поля в качестве ключа. Для задания первичного ключа можно выделить поле, вызвать контекстное меню для него и выбрать «Задать первичный ключ». В селекторном столбце указатель поля поменяется с  на  . Это будет свидетельствовать о том, что данное поле является первичным ключом (рис. 1.11).

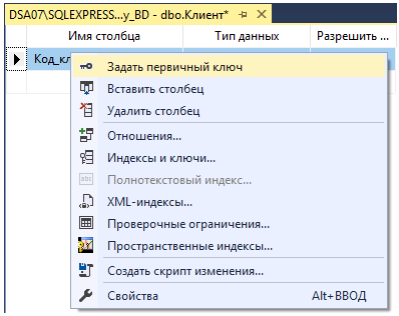


Рис. 1.11. Задание первичного ключа

Чтобы первичный ключ автоматически получал наращиваемое на единицу значение (стал «счетчиком») при вставке новой записи нужно в свойствах столбца установить Спецификацию идентификатора – «Да» и выставить соответствующие значения в поля Начальное значение и Шаг (см. рис. 1.12).

В некоторых таблицах приходится определять составной первичный ключ, который состоит из двух и более атрибутов. В этом случае необходимо выделить несколько полей и только потом в Контекстном меню выбрать «Задать первичный ключ». Значок  появится сразу возле всех выделенных полей (см. рис. 1.13).

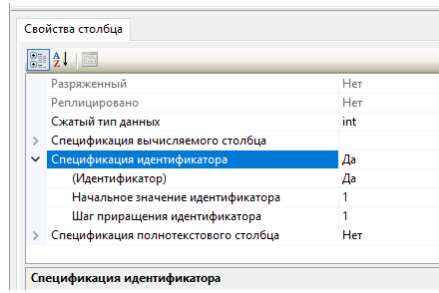


Рис. 1.12. Создание поля типа «счетчика»

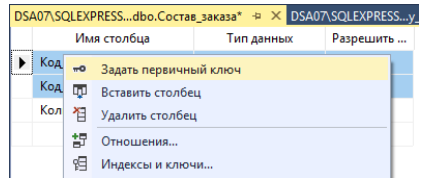


Рис. 1.13. Задание составного первичного ключа

Условия на значение. Во многих случаях ограничения на ввод только по типу данных являются недостаточными. В этом случае можно задать специфические ограничения на значение поля, выбрав в контекстном меню пункт *Проверочные ограничения* и затем *Добавить* (рис. 1.14).

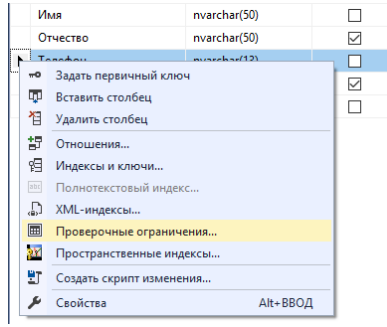


Рис. 1.14. Задание проверочных ограничений

В появившемся окне в поле *Выражение* (см. рис. 1.15) нужно ввести проверочное выражение, используя синтаксис построения условий во фразе WHERE SQL запросов (смотри методические указания для следующей лабораторной работы).

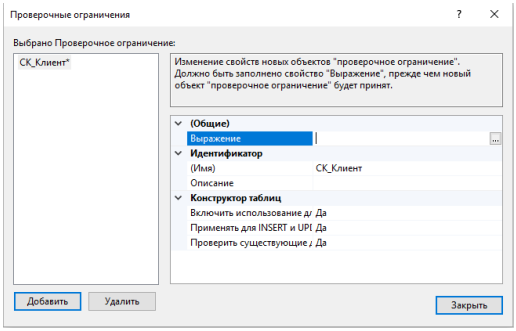


Рис. 1.15. Проверочные ограничения

Имя ограничения лучше всего поменять на более понятное разработчику. *Примечание!* Меню Проверочные ограничения относятся ко всем полям таблицы, поэтому в выражении нужно указывать имя поля, к которому относится данное ограничение. Например, проверочное ограничение для поля Телефон может выглядеть так (рис. 1.16):

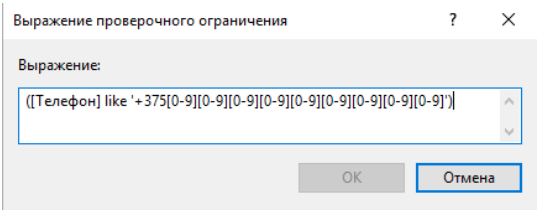


Рис. 1.16. Пример проверочного ограничения

Это означает, что в номере телефона 13 символов, при этом знак + и цифры 375 будут везде одинаковыми, а остальные 9 цифр могут изменяться от 0 до 9.

Если пользователь допустит ошибку при вводе данных в это поле или при их изменении, то появится предупреждение об ошибке и данные не будут сохранены до тех пор, пока пользователь не введет правильное значение (рис. 1.17).

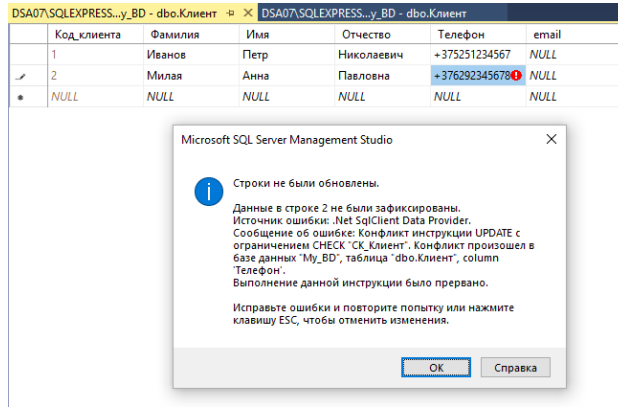


Рис. 1.17. Проверка работы проверочных ограничений

Значение по умолчанию. Для некоторых полей можно задавать значение по умолчанию, которое будет отображаться, если пользователь ничего не введет в данное поле. Это можно сделать в *Свойствах столбца*) в пункте *Значение по умолчанию или привязка* (рис. 1.18). Например, для поля Количество уместно задать значение по умолчанию 1 (рис.1.18).

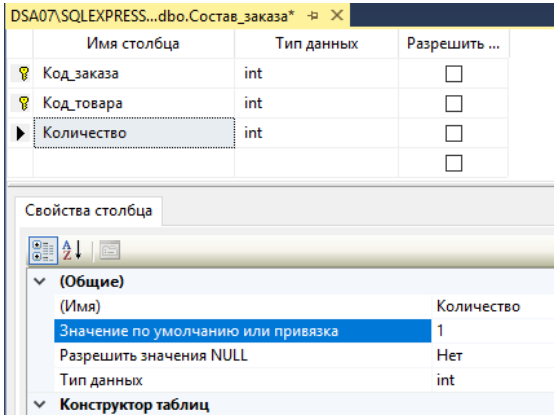


Рис. 1.18. Задание значения по умолчанию

Задание связей. Создание связей между таблицами производится путем определения ограничений ссылочной целостности (foreign key). Графическое представление схемы данных возможно с помощью объекта *Диаграмма*. Фактически каждая связь схеме данных является отражением

соответствующего ограничения ссылочной целостности. Данные ограничения можно задать с помощью языка SQL, с помощью соответствующего диалогового окна, доступного по вызову пункта меню Отношения, и графически в самом окне диаграммы данных. Для создания диаграммы можно вызвать контекстное меню *Создать диаграмму базы данных* (рис.1.19).

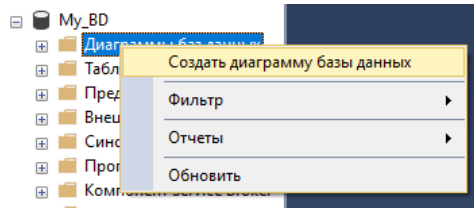


Рис. 1.19. Меню Создание диаграммы БД

Далее в появившимся окне нужно выбрать из представленного списка уже заранее созданные таблицы БД и нажать кнопку ОК. В окне диаграммы БД создание связей возможно с помощью графического перетаскивания полей. Захватываем мышью поле первичного ключа в одной таблице и перемещаем его в поле внешнего ключа другой таблицы (см. рис. 1.20).

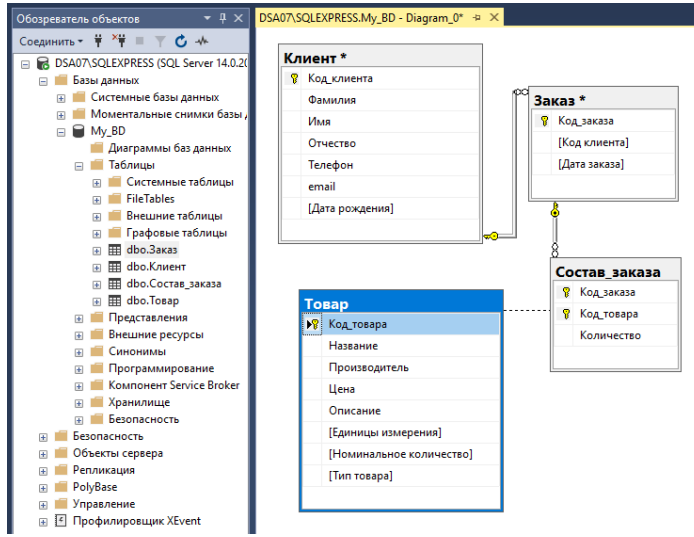


Рис. 1.20. Создание связей

Далее в появившимся диалоговом окне убеждаемся в правильности выбора полей и завершаем создание связи нажатием кнопки ОК (рис.1.21). Связь отображается линией, соединяющей две таблицы.

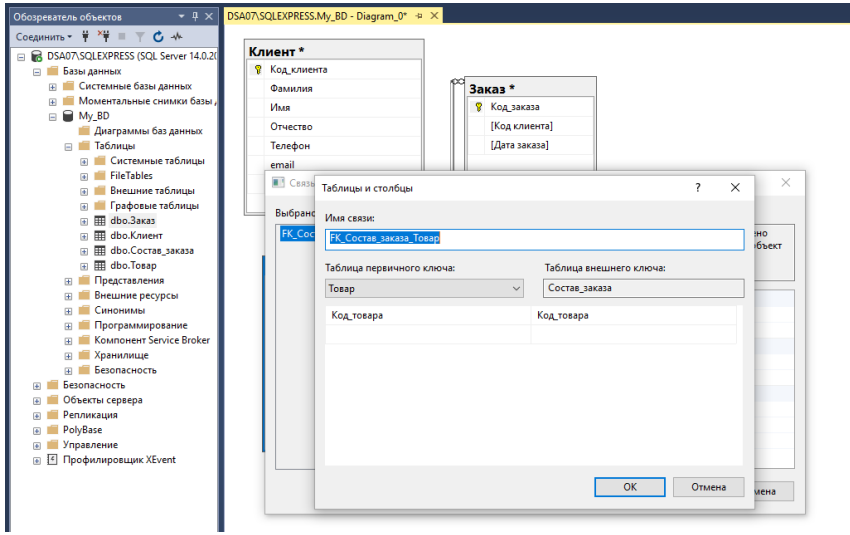


Рис. 1.21. Построитель связей

После сохранения диаграммы ее можно найти в узле Диаграммы баз данных (рис.1.22).

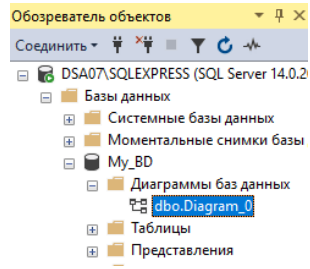


Рис. 1.22. Диаграмма в дереве объектов БД

Создание индексов. Индексы различают на первичные и вторичные. Первичный индекс определяется для первичного ключа в процессе его создания. Для остальных полей индексы нужно создавать самому. Составной индекс определяется для нескольких полей. Для создания индекса открываем нужную таблицу в режиме проекта и в контекстном меню вы-

бираем Индексы и ключи. В появившемся окне нажимаем на кнопку Добавить. Если этого не сделать, то изменения будут вноситься в уже имеющийся индекс (как правило в первичный), а не создаваться новый вторичный. Для нового индекса нужно определить поля, по которым будет выполняться индексирование, и уникальность (если необходимо). Задание полей для индексирования происходит в отдельном окне. В нем можно выбрать одно или несколько полей таблицы, а также указывать их порядок сортировки (или оставить по умолчанию). В созданном индексе также можно изменить название, а можно оставить и предложенное по умолчанию.

Задания

1. Разработать физическую модель данных, т.е. указать типы и размеры полей, а также необходимые ограничения целостности;
   Создать таблицы в СУБД MS SQL Server. Определить обязательные и ключевые поля. Задать значения по умолчанию. Определить условия на значение, задать связи между таблицами (требования ссылочной целостности). Задать каскадное удаление и обновление, где необходимо. Требования:
   - определить условие на значение для всех числовых полей (кроме первичных и внешних ключей);
   - определить условие на значение для текстового поля используя функцию **Like**, например, потребовать ввод символа @ в поле email;
   - для каждой таблицы определить по крайней мере одно **обязательное поле.**
2. Разработать индексы
   - для каждой из таблиц, кроме вспомогательных (справочников) и хранящих небольшой объем данных (до 100 записей), определить минимум один вторичный индекс;
   - определить минимум один составной индекс;
3. Ввести данные;
4. Научиться применять поиск, сортировку и фильтрацию данных средствами MS SQL Management Studio.

Форма отчета. Формой отчета являются разработанная база данных выбранной тематики, реализующая задания, представленные в разделе «Задания».

ЛАБОРАТОРНАЯ РАБОТА № 2

РАЗРАБОТКА ЗАПРОСОВ

Цель работы. Практическое освоение языка SQL.

Краткие сведения

Создание запросов. Создание запросов в SQL Management Studio может осуществляться двумя способами: с помощью написания запроса в текстовом (рис. 2.1) и визуальном редакторе запросов (рис. 2.3).

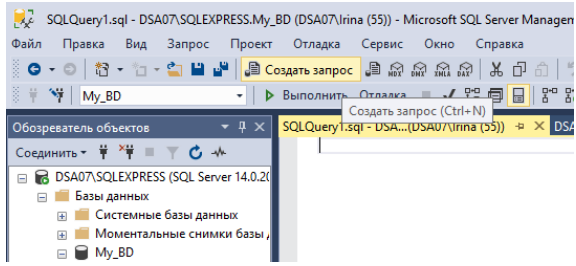


Рис. 2.1. Создание запроса

Для выполнения запроса можно нажать на кнопку *Выполнить* или, если в редакторе запросов присутствует много запросов, выделить нужный и тоже нажать кнопку *Выполнить* (рис. 2.2). Результат выполнения запроса отобразится в нижней части редактора. Если запрос не может быть выполнен, то в данном месте будет отображаться текст ошибки.

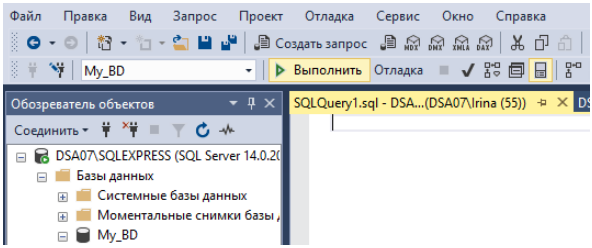


Рис. 2.2. Выполнение запроса

Для создания запроса в визуальном построителе нужно в окне редактора запросов в контекстном меню выбрать пункт *Создать запрос в редакторе* (рис. 2.3).

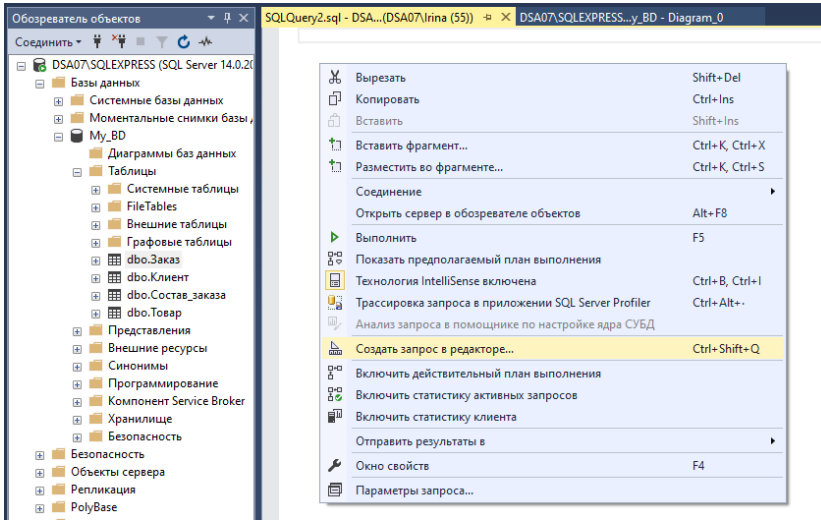


Рис. 2.3. Создание запроса с помощью редактора

Далее нужно указать, какие таблицы и какие поля будут участвовать в запросе и нажать ОК (рис. 2.4). Текст запроса на языке SQL появится в верхней половине окне. Его можно будет выполнить точно так же, как и обычный (написанный вручную) запрос, а также отредактировать по желанию, как вручную, так и с помощью редактора.

SQL. Запросы манипулирования данными

Запросы выборки. При изложении синтаксиса запросов будем придерживаться диалекта, принятого в СУБД SQL Server. Основу DML инструкций составляет инструкция SELECT:

```
SELECT [ALL|DISTINCT|DISTINCTROW] <список полей>
FROM <список таблиц>
[WHERE <условие отбора>]
[<Таблица1> INNER|RIGHT|LEFT JOIN <Таблица2>
ON <Таблица1>.<Поле1> = <Таблица2>.<Поле2>]
[GROUP BY <список полей>] [HAVING <условие отбора>]
[ORDER BY <спецификация> [,<спецификация>] ...]
```

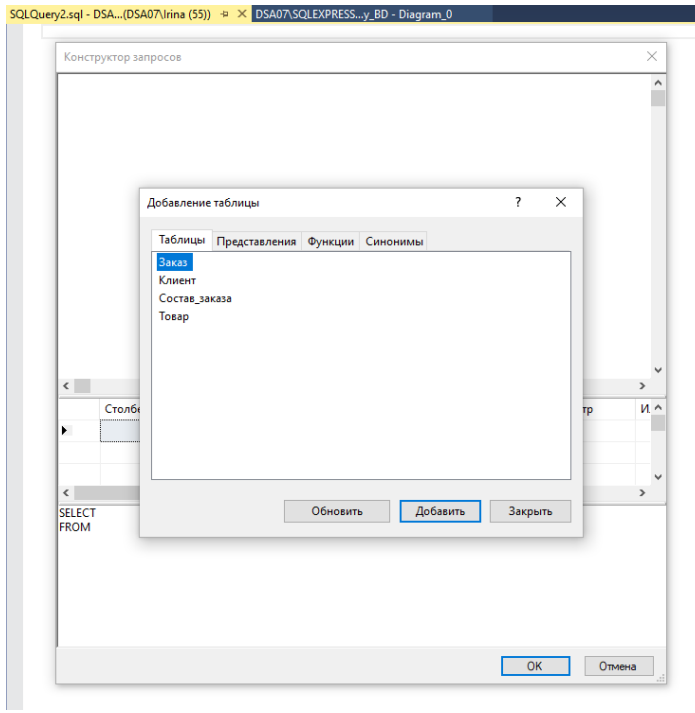


Рис. 2.4.Создание запроса с помощью редактора: выбор таблиц и полей

Инструкция SELECT позволяет производить выборку и вычисления над данными из одной или нескольких таблиц. Результатом выполнения инструкции является ответная таблица. Простейшая форма инструкции SELECT включает операторы SELECT и FROM. Оператор SELECT определяет поля, подлежащие выводу в выходной набор, а оператор FROM определяет имена таблиц, включенных в запрос. Имена полей и таблиц отделяются запятыми. Регистр не учитывается. Если в запросе участвуют несколько таблиц, то для исключения двусмысленности имена полей следует записывать в полной форме: <таблица>.<поле> (например, Клиенты.Адресс). Для повышения эффективности вначале указываются меньшие таблицы. Если имена полей и таблиц включают пробелы, то их необходимо заключать в квадратные скобки. Список полей следует задавать в той последовательности, в которой они должны быть представлены в выходном наборе. Например:

```
SELECT Фамилия, Имя, Отчество FROM Клиенты;
```

Для выбора всех полей применяется операция *:

```
SELECT * FROM Клиенты;
```

Список данных может содержать имена полей, участвующих в запросе, а также выражения над полями или строковые константы. В выражениях (вычисляемых полях) могут принимать участие имена полей, множество встроенных функций, константы, круглые скобки и следующие операторы и функции:

Арифметические операторы: + - / * % (% выч. остаток от деления)

Операторы сравнения: = <> > < >= <= !< !> !=

(восклицательный знак вызывает инверсию результата, оператор + также используется для конкатенации строк)

Побитовые операторы: & | ^.

CONCAT – выполняет конкатенацию строк. `CONCAT( hd, ' Gb' );`

POWER – возведение в степень. `POWER( float_expression , y );`

ISNULL – замена пустого значения

```
ISNULL(expression, replacement_value)
ISNULL(количество, 1)
```

NULLIF – возвращает значение NULL, если два указанных выражения равны. Если выражения не равны, функция NULLIF возвращает первое выражение: `NULLIF(expression1, expression2)`.

Именам полей и таблиц можно назначать альтернативные имена (псевдонимы). Псевдонимы записываются через ключевое слово AS (<имя таблицы> или <имя поля> AS <псевдоним>) или просто через пробел. Например:

```
SELECT [Фамилия] + ' ' + [Имя] + ' ' + [Отчество] AS Полное_имя, 'возраст', DateDiff(yyyy, [Дата рождения], GetDate()) AS Возраст FROM Клиенты;
```

В данном примере псевдонимы использовались для задания имен вычисляемым полям. В противном случае имя вычисляемого поля имело бы вид: Выражение1 и т.д. В большинстве случаев псевдонимы используются для сокращения набора длинных имен. Особенно это эффективно для замены длинных имен таблиц, поскольку в многотабличных запросах приходится включать имена таблиц в описание каждого поля.

```
SELECT cl.Фамилия, cl.Имя FROM Клиенты As cl;
SELECT Clients.Фамилия Surname, Clients.Имя [First name] FROM Клиенты Clients;
```

SQL Server имеет строгое соблюдения типов. Соответственно, выполнить прямую конкатенацию строки и числа нельзя. Преобразование типов выполняется с помощью следующих функций:

```
CAST(expression AS data_type)
```

```
CONVERT(data_type[(length)], expression [, style])
Str(число, длина, точность)
```

Например: `Str(123, 5, 2)` `POWER(X, CAST(2.0 as int))`

```
CAST(Продукты.вес AS varchar(10)) + ' кг' AS вес_продукта
```

`CONCAT(Продукты.вес, ' кг')` (преобразование не требуется)

В примерах выше «Продукты.вес» – поле таблицы.

Если в запрос включены не все поля некоторой таблицы, то выходной набор может содержать одинаковые строки. Предикаты ALL, DISTINCT (записываются сразу после оператора SELECT) служат для управления выводом повторяющихся строк. По умолчанию принимается предикат ALL, т.е. в ответную таблицу включаются все строки, в том числе и повторяющиеся. Предикат DISTINCT исключает повторяющиеся записи. Предикат TOP возвращает определенное число записей, находящихся в начале или в конце диапазона, описанного с помощью фразы ORDER BY. Можно указывать или абсолютное число записей или определенный процент, например: `SELECT TOP 5 PERCENT Person.Name FROM Person; SELECT TOP 10 Person.Name FROM Person;`

Оператор WHERE необязателен. Он задает условия, которым должны удовлетворять записи в результирующей таблице. Выражение <условие отбора> является логическим. Его элементами могут быть имена столбцов, операции сравнения <, <=, >, >=, =, <>, арифметические операции, логические операторы (NOT, AND, OR, XOR), скобки, функции IN, BETWEEN, LIKE, IS (NOT) NULL и множество встроенных функций. Строки заключаются в одиночные кавычки (зависит от СУБД) Для создания литерала типа Дата/Время из строки требуется явное приведение типа с помощью оператора CONVERT.

Функция IN проверяет на равенство любому значению из списка:
`[Город] IN ('Минск', 'Москва', 'Киев').`

Функция BETWEEN задает диапазон значений. Границы диапазона разделяются оператором And: `BETWEEN 50 And 100.`

Функция LIKE проверяет на соответствие заданному шаблону символов. В качестве символов шаблона используются:

% – любое число произвольных символов.

_ – один произвольный символ.

[] – диапазон допустимых символов. К примеру, [А - Я], [3 - 9]. Если наоборот необходимо исключить эти символы, то перед ними ставится !: [!А - Я]. Например: `LIKE '%чай%', LIKE 'MP223[34][4-6]__'.`

Приведем примеры запросов на применение фразы WHERE в SQL Server:

```
SELECT Фамилия FROM Клиенты WHERE Город = 'Минск';
```

```
SELECT * FROM Клиенты WHERE Отчество IS NULL;
SELECT * FROM Клиенты WHERE Паспорт LIKE 'MP223[34][4-6]__';
```

Запрос может быть основан на нескольких таблицах. Простое включение полей из нескольких таблиц даст простой перебор всех их возможных комбинаций (декартово произведение). Для двух таблиц общее число записей будет равно произведению числа записей в первой и второй таблицах. Но так как реляционная база данных практически всегда состоит из таблиц, связанных между собой посредством совпадающих значений полей, участвующих в связи, то для правильного соединения данных необходимо включить в запрос явное определение соответствующих связей. Связи можно задать с помощью двух способов: с помощью оператора INNER|RIGHT|LEFT JOIN и с помощью дополнительного условия выборки после оператора WHERE. Причем в SQL соединени данных можно произвести даже по неравенству, т.е. поддерживаются операции сравнения =, <>, <, <= ,> , >=. Оператор INNER|RIGHT|LEFT JOIN является необязательной частью инструкции SELECT и оформляется как часть оператора FROM.

```
SELECT Товары.[Наименование товара], Заказы.Дата, Заказы.[Полная цена] FROM Товары INNER JOIN Заказы ON Товары.Код_товара = Заказы.Код_товара WHERE Товары.Цена > 100;
```

Этот же запрос можно записать следующим образом (второй способ задания связи не рекомендуется, так как он нагружает фразу WHERE дополнительным отбором записей):

```
SELECT Товары.[Наименование товара], Заказы.Дата, Заказы.[Полная цена] FROM Товары, Заказы
WHERE Товары.Код_товара = Заказы.Код_товара AND Товары.Цена > 100;
```

Для большинства многотабличных запросов набор записей формируется на основе совпадающих значений полей, участвующих в связи, т.е. с помощью внутреннего соединения (INNER JOIN). Для двух таблиц, связанных отношением «один ко многим», из главной таблицы будут отобраны только те записи, которые имеют связанные записи в подчиненной таблице. Допустим, мы имеем две таблицы, «Клиенты» и «Заказы». С помощью внутреннего соединения мы получим сведения по клиентам, имеющим хотя бы один заказ. Но предположим, что мы хотим получить информацию по всем клиентам и посмотреть, кто заказывал что-либо, а кто нет. Ответ на такой вопрос позволяют дать запросы с внешним соединением (LEFT|RIGHT [OUTER] JOIN). Тогда в запрос будут включены все записи с таблицы «Клиенты» и те записи с таблицы «Заказы», которые имеют связанные записи в главной таблице. Соответственно, такое соединение вернет совпадающие по условию соединения данные из обеих таб-

лиц и еще дополнит выборку оставшимися данными из внешней таблицы, которые по условию не подходят, заполнив недостающие данные значением NULL. Выбор LEFT либо RIGHT зависит от того, с какой стороны от слова JOIN находится та таблица, из которой необходимо отобрать все записи, например:

```
SELECT Фамилия, Имя, [Дата заказа], Цена
FROM Клиенты LEFT JOIN Заказы
ON Клиенты.код_клиента = Заказы.код_клиента;
```

Оператор ORDER BY задает порядок сортировки результирующего множества. Обычно он замыкает инструкцию SELECT. Каждая <спецификация> сортировки представляет собой пару вида: <имя поля> [ASC/DESC]. Большинство СУБД требуют, чтобы поле, участвующее в сортировке, также присутствовало и в выходном наборе.

```
SELECT Наименование FROM Товары ORDER BY Наименование;
```

Запросы с группировкой. Иногда интерес представляет не каждая строка таблицы в отдельности, а итоговые значения по группам данных. Например, может понадобиться общая сумма продаж для клиентов, проживающих в определенном районе или интересно знать средний объем продаж по месяцам, чтобы выяснить тенденции сбыта. Получить ответы на такие вопросы можно с помощью итоговых запросов (запросов с группировкой). Оператор GROUP BY позволяет выделять группы в результирующем множестве записей. Группой являются записи с совпадающими значениями в полях, перечисленных за оператором GROUP BY. Оператор перегруппирует таблицу таким образом, чтобы в каждой группе все строки имели одно и тоже значение поля. Инструкция SELECT затем применяется уже к группам перекомпонованной таблицы. Каждое выражение во фразе SELECT должно принимать единственное значение для группы, т.е. оно может быть либо самим полем, либо арифметическим выражением, включающем это поле, либо константой, либо агрегатной функцией, возвращающей единственное значение для группы. В запросах с группировкой необходимо тщательно следить за включение полей во фразу SELECT, так как в противном случае можно не получить желаемого результата. Если во фразу SELECT будет помещено хотя бы одно поле, которое не является единственным для группы, например ключевое поле подчиненной таблицы, то создание групп будет прервано, так как в результате каждая строка запроса будет уникальна.

В результате запроса только с группировкой по некоторому полю получится таблица, содержащая по одной записи для каждой группы. Группирование записей само по себе ничего не дает. Обычно производятся

вычисления для групп. Для этой цели имеется ряд групповых (иначе агрегатных) функций:

- SUM – вычисляет сумму всех значений заданного поля в каждой группе;
- AVG – вычисляет среднее арифметическое всех значений заданного поля в каждой группе;
- STDEV – вычисляет стандартное отклонение всех значений заданного поля в каждой группе;
- VAR – вычисляет дисперсию всех значений заданного поля в каждой группе;
- COUNT – вычисляет число записей, для которых значение заданного поля отлично от NULL. Для подсчета всех записей необходимо использовать операцию *: Count(*);
- MIN – возвращает минимальное значение заданного поля в каждой группе;
- MAX – возвращает максимальное значение заданного поля в каждой группе;
- FIRST – возвращает первое значение заданного поля в каждой группе;
- LAST – возвращает последнее значение заданного поля в каждой группе.

Групповые функции могут применяться сами по себе, без выполнения группировки. Тогда группой будет считаться все отобранные оператором WHERE записи. Например, запрос:

```
SELECT Count(*) FROM Заказы
```

подсчитает количество записей в таблице заказы, а запрос

```
SELECT Sum([Отпускная цена] + [Транс издержки]) As Полная_цена
FROM Заказы WHERE Город = 'Минск';
```

подсчитает полную сумму цен по всем заказам, сделанным из Минска. Следующий запрос выполнит группировку по полю Фамилия, подсчитает сумму цен по всем заказам, сделанным из Минска, и выведет ее, переименовав в Стоимость, вместе с фамилией клиента:

```
SELECT Клиенты.Фамилия, Sum(Заказы.Цена) AS Стоимость,
Count(Заказы.КодЗаказа) AS Количество FROM Клиенты INNER JOIN
Заказы ON Клиенты.КодКлиента = Заказы.КодКлиента WHERE Клиен-
ты.Город = 'Минск'
```

Оператор HAVING действует совместно с оператором GROUP BY и используется для выборки записей с применением агрегатных функций. Он выполняет те же функции, что и WHERE, но уже в рамках выходного набора. Оператор HAVING устанавливает, какие записи, сгруппированные посредством GROUP BY, должны отображаться и участвовать в группо-

вых операциях. Правила записи < условия отбора > аналогичны правилам формирования <условия отбора> оператора WHERE. Следующий запрос отбирает коды товаров, покупаемых более чем одним покупателем:

```
SELECT код_товара FROM Заказы GROUP BY код_товара
HAVING Count(*) > 1;
```

Отметим, что применение агрегатных функций неприемлемо во фразе WHERE, где отбор записей производится на уровне отдельных строк. Попытка написать `WHERE Count(*) > 1` приведет к ошибке.

Вложенные запросы. Инструкции SELECT могут многократно вкладываться друг в друга. Вложенная инструкция SELECT записывается после оператора WHERE и служит для отбора записей основного запроса. SQL выполняет вложенный подзапрос и затем сравнивает каждую строку основного запроса с результатом вложенного. Вложенные запросы записываются внутри скобок. Например:

```
SELECT Фамилия, Имя FROM Клиенты
WHERE Кредит < (SELECT AVG(Кредит) FROM Клиенты);
```

Если подчиненный запрос возвращает набор записей, то вместо операторов сравнения следует использовать функции ALL, SOME, ANY, EXISTS, IN. Квантор существования EXISTS обычно записывается следующим образом EXISTS(SELECT * FROM …). Перед ним можно поставить NOT для инверсии результата. Выражение EXISTS считается истинным тогда и только тогда, когда результат вычисления подзапроса является непустым множеством. В запросах данного типа необходимо явно определять связь между таблицами в виде условия отбора фразы WHERE вложенного запроса. В качестве примера выберем фамилии покупателей, которым был продан компьютер «Pentium II 350»:

```
SELECT Фамилия FROM Клиенты WHERE EXISTS(SELECT * FROM Заказы
WHERE Заказы.код_клиента = Клиенты.код_клиента And [Наименова-
ние товара] = "Pentium II 350");
```

Функци SOME и ANY (синонимы) используются для отбора в главном запросе записей, которые удовлетворяют сравнению с *некоторыми* (хотя бы одной) записями, отобранными подчиненным запросом. Например, этот запрос отбирает все товары, цена которых больше, чем цена любого товара (т.е. самого недорогого), проданного со скидкой 25%:

```
SELECT * FROM Товары WHERE Цена > ANY(SELECT Цена FROM
Заказано WHERE Скидка >= 0.25);
```

Функция ALL используются для отбора в главном запросе записей, которые удовлетворяют сравнению со всеми записями, отобранными подчиненным запросом. Следующий запрос отбирает все товары, цена кото-

рых больше, чем цена всех товаров (т.е. самого дорогого), проданных со скидкой 25%:

```
SELECT Марка FROM Товары WHERE Цена > ALL(SELECT Цена FROM За-
казано WHERE Скидка >= 0.25);
```

Если в запросе, представленном выше, указать оператор сравнения «=» (Цена = ALL(…)) то основной запрос выведет записи только в том случае, если подчиненный запрос отберет товары с совершенно одинаковой ценой. Если подчиненный запрос вернет хотя бы одну запись с другой ценой, то основной запрос не выведет ни одной строки, так как условие цена = ALL(…) не будет выполнено. Функция IN используются для отбора в главном запросе только тех записей, которые содержат значения, совпадающие с одним из отобранных подчиненным запросом. Запрос отбирает все товары, цена которых совпадает с ценой товаров, проданных со скидкой 25%:

```
SELECT Марка FROM Товары WHERE Цена IN(SELECT Цена FROM Зака-
зано WHERE Скидка >= 0.25);
```

Запросы действий. С помощью запросов действий пользователь может изменять или переносить данные в таблицах, обновлять, добавлять или удалять группы записей, а также создавать новые таблицы.

С помощью запроса на обновление можно изменить группу записей, отобранных по заданному критерию. Инструкция запроса на обновление имеет формат вида:

```
UPDATE <имя таблицы>
SET <имя поля> = {<выражение> , NULL } [, SET <имя поля > =
{<выражение> , NULL } ... ]
[WHERE <условие отбора>]
```

Новые значения полей в записях могут быть пустыми (NULL), либо вычисляться в соответствии с арифметическим выражением. Правила записи арифметических и логических выражений аналогичны соответствующим правилам для вычисляемых полей. Например,

```
UPDATE Товары
SET Наименование = “Pentium III 800”, Цена = Цена + 250
WHERE Наименование = “Pentium II 350”;
```

С помощью запроса на добавление записи одной таблицы (все или отобранные запросом) можно добавить в конец другой таблицы. Этот запрос также позволяет добавить в таблицу всего одну запись, состоящую из литералов. Соответственно запрос на добавление имеет форматы двух видов:

```
INSERT INTO <имя таблицы> [(<список полей>)] VALUES (<список
значений>) и
```

```
INSERT INTO <имя таблицы> [(<список полей>)] <инструкция SELECT>
```

В первом формате оператор INSERT предназначен для ввода одной записи, состоящей из литералов или выражений. Порядок перечисления полей должен соответствовать порядку значений, перечисленных в списке значений оператора VALUES. При явном перечислении можно опускать некоторые поля. Если список полей опущен, то в списке значений должны быть перечислены все значения в порядке следования полей таблицы. Во втором формате оператор INSERT предназначен для добавления записей, отобранных из другой таблицы с помощью инструкции SELECT. Здесь также необходимо обеспечить соответствие полей (типов и размеров полей или, по крайней мере, возможность полноценной конвертации данных), перечисленных как после оператора INSERT INTO, так и после SELECT. Например,

```
INSERT INTO Товары(Наименование, Цена) VALUES («Pentium II 233», 450);
INSERT INTO Клиенты(Фамилия, Имя, Отчество) SELECT Фамилия, Имя, Отчество FROM Поставщики WHERE Город = “Минск”;
```

С помощью запроса на удаление можно сразу удалить группу записей, удовлетворяющих определенному критерию. Этот запрос особенно эффективен при удалении большого числа записей. С помощью этого запроса можно явно удалить записи только из одной таблицы. Но если было определено каскадное удаление, то будут также удалены связанные записи из подчиненных таблиц. Запрос на удаление имеет формат вида:

```
DELETE FROM <имя таблицы> [WHERE <условие отбора>]
```

Если необязательный оператор WHERE опущен, т. е. условие отбора удаляемых записей отсутствует, удалению подлежат все записи таблицы. Например, запрос удаляет все записи, где Наименование начинается с Celeron:

```
DELETE FROM Товары WHERE Наименование LIKE “Celeron%”;
```

С помощью этого запроса можно создать новую таблицу на основе уже существующей и перенести в нее все или удовлетворяющие некоторому критерию записи. Новая таблица будет иметь ту же структуру. Обычно этот запрос применяется для создания архивных копий таблиц. Запрос на создать новой таблицы имеет формат вида:

```
SELECT <список полей> INTO <имя новой таблицы>
FROM <имя таблицы> [WHERE <условие отбора>].
```

В новую таблицу будут перенесены поля, перечисленные в списке полей. Например:

```
SELECT Клиенты.Фамилия, Клиенты.Имя, Клиенты.Отчество INTO Со-
вершеннолетние
FROM Клиенты
WHERE Year(getdate())-Year(Клиенты.[Дата рождения])>18;
```

Внимание: при выполнении запросов действий нужно быть предельно внимательным. Запросы действий получили свое название вследствие того, что они производят действие. СУБД не предупреждает пользователя о том, что предпринимается попытка удалить или обновить скажем 1000 записей. Записи просто удаляются и клиентскому приложению (в роли которого выступает Management Studio) приходит лишь отчет об успешности или неуспешности выполненной операции и затронутом количестве строк. На самом деле записи не удаляются, место, которое они занимали, помечаются как свободное. Но это не говорит о том, что удалённые записи можно вернуть. Не имея резервной копии или журнала выполнения транзакций это сделать практически невозможно.

В заключение описания запросов выборки рассмотрим несколько операторов, реализующих множественно-теоретические операции реляционной алгебры. Оператор UNION позволяет объединить выходные наборы нескольких инструкций SELECT в одну результирующую таблицу. Этот запрос является аналогом операции объединения отношений $R = A \cup B$ в реляционной алгебре. При объединении записи, возвращаемые второй и последующими инструкциями SELECT, будут дописываться в конец первой, причем из результата выборки будут всегда исключаться повторяющиеся записи. Для включения всех записей после оператора UNION нужно указать предикат ALL. В выходных наборах всех инструкций SELECT должно быть одинаковое количество полей с одинаковыми характеристиками. В крайнем случае некоторую инструкцию SELECT можно дополнить строковыми константами или вычисляемыми полями. В качестве заголовков столбцов для выходного набора будут приниматься имена полей первой инструкции. Для выполнения сортировки результирующего набора данных объединение может дополняться оператором ORDER BY, который записывается в конце последней инструкции SELECT.

```
SELECT [Фамилия] + ' ' + [Имя] + ' ' + [Отчество] As Название,
Город FROM Клиенты
UNION
SELECT Поставщик, Город FROM Поставщики
ORDER BY Название, Город;
```

Кроме UNION в SQL Server включены еще две операции реляционной алгебры: исключения – EXCEPT (аналог операции разности $R = A - B$. Отношение *R* включает записи, присутствующие в *A* и отсутствующие в *B*) и пересечения – INTERSECT (аналог операции пересече-

ния $R = A \cap B$. Отношение *R* включает записи, присутствующие как в *A*, так и в *B*). Как и для объединения, операнды (таблицы) этих операций должны содержать соответствующий по типам набор полей.

Операторы определения данных. Оператор создания таблицы имеет формат вида:

```
CREATE TABLE <имя таблицы> (<имя поля> <тип данных> [NOT NULL]
[,<имя поля> <тип данных> [NOT NULL]] ... )
```

Обязательными операндами оператора являются имя создаваемой таблицы и имя хотя бы одного поля с указанием типа данных. При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Конструкция NOT NULL требует, чтобы в этом столбце должно быть определено значение. Например:

```
CREATE TABLE Товары(Код INT PRIMARY KEY IDENTITY(1,1), Тип
CHAR(8), Наименование VARCHAR(20) NOT NULL, Цена DECIMAL(8,2) );
```

SQL поддерживает пять скалярных типов данных: символьный (строковый), битовый, численный, даты/время и интервал. Основные типы данных SQL Server перечислены ниже:

char(n) nchar(n)	Текстовые значения фиксированной длины в n символов (n байт). Если введено меньше символов, строка дополняется пробелами до n символов. nchar(n) – для хранения символов в Unicode (2n байт).
varchar(n) nvarchar(n)	Текстовые значения до n символов (до 4000). Сохраняется сколько введено. nvarchar(n) – для хранения символов в Unicode.
text (ntext)	Текст или битовый массив (*оставлен для совместимости*). Следует использовать varchar(max) – до 2 Gb.
bit	Логические значения (одно поле 1 байт, 8 полей 1 байт, 16 полей – 2 байта, …)
tinyint	Целые числа от 0 до 255 (1 байт)
smallint	Целые числа от -32 768 до 32 767 (2 байта)
int	Целые числа от -2 147 483 648 до 2 147 483 647 (4 байт)
bigint	Целые числа от -2^{63} до 2^{63-1} (8 байт)
smallmoney money	Денежные значения, используемые в математических расчетах с точностью до 15 знаков в целой и до 4 знаков в дробной части (4 и 8 байт соответственно)
float(n)	Числа с плавающей точкой (4-8 байт). n – от 1 до 53.

	float(24) соответствует типу float; float(53) соответствует типу double
real	Числа с плавающей точкой (4 байт)
date	Значения даты с точностью до дня от 1 до 9999 года (3 байта)
smalldatetime datetime	smalldatetime – значения даты/время от 01.01.1900 до 06.06.2079 (4 байта) . Точность 1 мин. smalldatetime – значения даты/время от 1 января 1753 года по 31 декабря 9999 года (8 байт). Точность 0.0033 секунды
time(n)	Значения времени, точность от 100 нс, 3 - 5 байт.
image	Массив байтов от 0 до 2 147 483 647 байт
binary varbinary	Двоичные данные фиксированной и переменной длины размером в *n* байт, где *n* — значение от 1 до 8000 (*n* байт).

Оператор изменения структуры таблицы имеет формат вида:

```
ALTER TABLE <имя таблицы>({ADD, MODIFY, DROP} <имя поля> [<тип данных>] [NOT NULL],...)
```

Изменение структуры таблицы может состоять в добавлении (ADD), изменении (MODIFY) или удалении (DROP) одного или нескольких столбцов таблицы. Правила записи инструкции ALTER TABLE такие же, как и инструкции CREATE TABLE, разве что при удалении столбца указывать тип данных не требуется. Для примера добавим одно поле:

```
ALTER TABLE Товары(ADD Категория VARCHAR(20));
```

Оператор CONSTRAINTS в составе инструкции ALTER TABLE позволяет задать ограничители целостности. Фраза PRIMARY KEY определяет первичный ключ таблицы. Фраза UNIQUE позволяет определить альтернативные (потенциальные) ключи. Фраза FOREIGN KEY … REFERENCES используется для задания ссылочной целостности (определения связей между таблицами). Можно дополнительно определить каскадное удаление и каскадное обновление. Для выполнения ссылочной целостности SQL определяет четыре вида действий: CASCADE, SET NULL, SET DEFAULT, NO ACTION (используется по умолчанию). Фраза CHECK служит для задания дополнительных условий для значений полей таблицы, например:

```
ALTER TABLE Заказы ADD CONSTRAINT cfk foreign key(client_id) references clients(id)
```

(Результатом действия данного запроса является связь между таблицами в диаграмме данных. Можно вначале определить ссылочную целостность между таблицами, а только затем вызвать построитель диаграмм, чтобы убедиться в правильности на-

значенных связей. Удаление связи на диаграмме эквивалентно удалению заданного правила ссылочной целостности.)

```
ALTER TABLE Заказы ADD CONSTRAINT cprice check(price between 0 and 100000)
ALTER TABLE Заказы ADD CONSTRAINT cdate default(GetDate()) for date
```

Оператор удаления таблицы имеет формат вида:

```
DROP TABLE <имя таблицы>.
```

Инструкция создания индекса имеет формат вида:

```
CREATE [UNIQUE] INDEX <имя индекса> ON <имя таблицы> (<имя поля> [ASC | DESC] [,<имя поля> [ASC | DESC]... ) [WITH {PRIMARY | DISALLOW NULL | IGNORE NULL}]
```

Инструкция позволяет создать индекс для одного или нескольких столбцов заданной таблицы. Для одной таблицы можно создать несколько индексов. Например:

```
CREATE INDEX AscDate ON Клиенты([День рождения])
CREATE INDEX AscName ON Клиенты(Фамилия, Имя, Отчество)
```

Индекс с требованием уникальности индексируемых значений создается с префиксом UNIQUE, например:

```
CREATE UNIQUE INDEX passportasc ON clients(passport)
```

Инструкция удаления индекса имеет формат вида:

```
DROP INDEX <имя индекса>    DROP INDEX passportasc
```

На основе запроса выборки можно построить представление. В SQL представление является виртуальной таблицей, построенной на основе данных одной или нескольких таблиц, т. е. запрос выборки может быть многотабличным. По одним и тем же таблицам можно построить несколько представлений. Представление само по себе не содержит данных, а извлекает их из таблиц в момент обращения к нему.

Инструкция создания представления имеет следующий формат:

```
CREATE VIEW <имя представления>
[(<имя поля> [,<имя поля> ]... )] AS <инструкция SELECT>
```

Если имена столбцов в представлении не указываются, то будут использоваться имена столбцов из запроса, описываемого соответствующим оператором SELECT. Пример инструкции создания представления показан ниже:

```
CREATE VIEW заказы_клиентов (клиент, товар, цена, дата) AS SELECT Фамилия + ' ' + Имя + ' ' + Отчество, Товар, Цена, Дата FROM Клиенты INNER JOIN Заказы ON Клиенты.код_клиента = Заказы.код_клиента WHERE Город = 'Минск' ORDER BY Фамилия + ' ' + Имя + ' ' + Отчество;
```

Инструкция удаления представления имеет формат вида:

```
DROP VIEW <имя представления>.
```

Задания

1. Написать запросы в СУБД MS SQL Server. Первые 3 запроса из списка можно написать с помощью визуального построителя запросов, приветствуется независимо продублировать (возможно на других данных) и на языке SQL с целью изучения последнего. Остальные запросы написать на языке SQL:
   - многотабличный запрос выборки с сортировкой и отбором данных (INNER JOIN, WHERE, ORDER BY),
   - запрос с применением вычисляемых полей (примечание: не применять агрегатные функции), например, вычислить возраст, полную цену, и типа подобное,
   - запрос выборки с внешним объединением двух отношений (LEFT|RIGHT JOIN),
   - запрос группировкой, вычислением итогов и отбором данных (GROUP BY, HAVING),
   - запрос на добавление (INSERT INTO),
   - запрос на удаление,
   - запрос на обновление,
   - запрос на создание новой таблицы на основе существующей,
   - запрос на объединение (UNION), где применимо (только в SQL),
   - вложенный запрос на SQL (вложение во фразе WHERE);
   - запрос на создание новой таблицы,
   - запрос на создание индекса,
   - запрос на создание представления, объединяющего данные двух таблиц;
2. Изучить SQL на примере созданных запросов;
3. По мере необходимости написать и другие запросы, требуемые для решения конкретных задач разрабатываемой базы данных.

Форма отчета. Формой отчета являются база данных выбранной тематики и листинг всех выполненных запросов, реализующая задания, представленные в разделе «Задания».

ЛАБОРАТОРНАЯ РАБОТА № 3

РАЗРАБОТКА ФОРМ СРЕДСТВАМИ MS ACCESS

Цель работы. Практическое освоение разработки форм в MS Access как средство создания интерфейса доступа к СУБД SQL Server.

Краткие сведения

СУБД MS Access может служить не только в качестве системы управления некоторой локальной БД, но и как прекрасное средства создания интерфейса доступа к высокоразвитым СУБД, например, SQL Server, Oracle и др. В такой связке получаем возможность создавать БД, имеющие клиент-серверную архитектуру, где Access будет реализовывать клиентскую часть. Таким образом открывается превосходная возможность объединить достоинства высокоразвитых клиент-серверных СУБД и достоинства Access, такие как легкость генерации экранных форм и превосходные возможности создания и легкой модификации отчетов. СУБД MS Access отлично подходит и как средство изучения технологии БД, так как позволяется сконцентрироваться на проблематике БД, а не изучению многочисленных фреймворков и библиотек доступа, требуемых для создания полноценного графического интерфейса к БД. Доступ к данным в такой связке реализуется посредством технологии ODBC. ODBC позволяет полностью абстрагироваться от деталей конкретной СУБД.

Настройка соединения с SQL Server. Настройка доступа к конкретной БД осуществляется посредством администратора данных ODBC. Для доступа к удаленному серверу с использованием технологи ODBC требуется создать источник данных Data Source Name (DSN). Для этого необходимо открыть администратор источников данных ODBC (см. рис. 3.1) и нажать кнопку Добавить.

В открывшемся окне потребуется ввести название источника и имя сервера. В заключение потребуется выбрать БД из списка доступных на сервере. Если на компьютере установлен SQL Server Express, то имя сервера состоит из имени компьютера и имени экземпляра сервера (по умолчанию SQLEXPRESS), разделенным символом \, например, LAB48-PC\SQLEXPRESS.

На локальной машине по умолчанию соединение устанавливается через протокол общей памяти (протокол Shared Memory). Соединение с удаленным сервером можно настроить и по протоколам TCP/IP и Named Pipes. Для этого соответствующие протоколы должны быть разрешены в

диспетчере конфигурации SQL Server. Дополнительно требуется разрешение доступа по соответствующим протоколам в брандмауэре (замечание, разрешения добавляются автоматически при установке SQL Server).

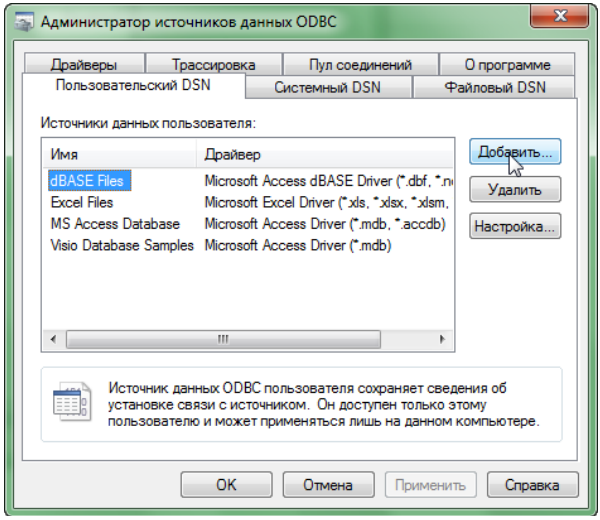


Рис. 3.1. Добавление источника данных

Если в диспетчере конфигурации SQL Server созданы псевдонимы (aliases) с явным указанием определенного протокола, чаще всего TCP/IP, то производится попытка соединения по указанному протоколу. Если соединение никак не удается установить (в сообщении об ошибке можно найти название протокола), то можно просто удалить псевдонимы и обращаться к серверу по его полному имени (имя компьютера \ имя экземпляра).

Для включения протокола TCP/IP выполните следующие шаги.

1. Запустите диспетчер конфигурации SQL Server (Диспетчер конфигурации можно найти в Управление компьютером/ Службы и приложения) (рис. 3.2).
2. Разверните узел Сетевая конфигурация SQL Server(рис. 3.2).
3. Выберите Протоколы для MSSQLSERVER (рис. 3.2).
4. Щелкните правой кнопкой мыши пункт TCP/IP и выберите Включить (рис. 3.3 а)).
5. Дополнительно проверьте включены ли требуемые клиентские протоколы (рис. 3.3 б)).
6. Выберите Службы SQL Server.

7. Щелкните правой кнопкой мыши SQL Server и выберите команду Перезапустить.

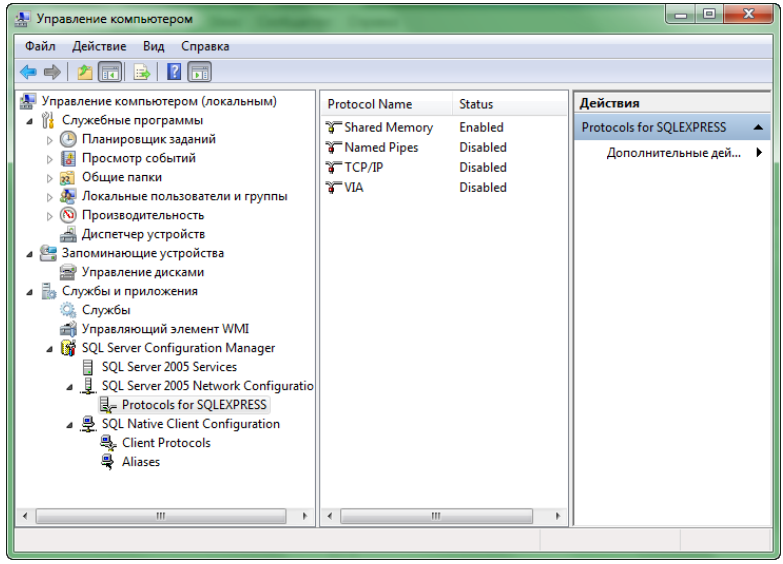
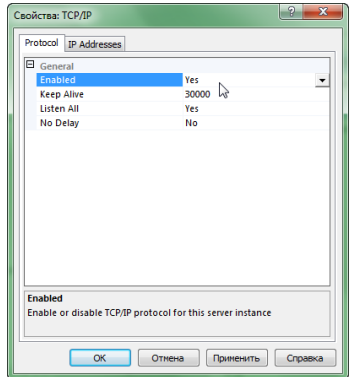
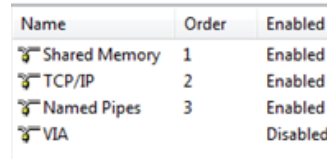


Рис. 3.2. Управление компьютером





а) б)

Рис. 3.3. а) Свойства TCP/IP б) Проверка их активности

Разработка форм. Для создания форм Microsoft Access предоставляет следующие возможности (рис. 3.4).

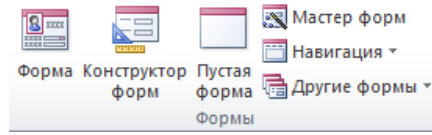


Рис. 3.4. Формы

Для автоматического создания формы, открываем таблицу, для которой эта форма создается, и нажимаем кнопку *Форма*. Результатом выполнения этого действия является полностью работоспособная форма со всеми полями этой таблицы и заголовком (рис. 3.5).

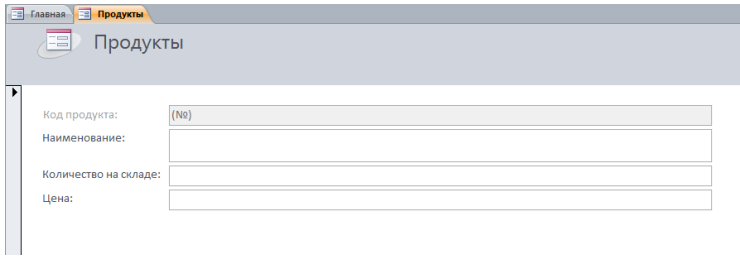


Рис. 3.5. Форма для таблицы

Пункт меню *Конструктор форм* позволяет разработчику создать форму от начала до конца. При выборе этой команды открывается дизайнер (конструктор) форм следующего вида (рис. 3.6), в правой части которого расположено окно свойств.

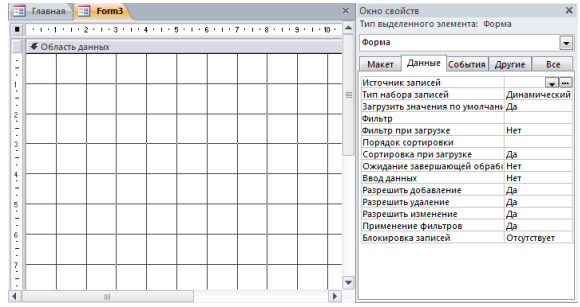


Рис. 3.6. Форма в режиме конструктора

Кнопка *Пустая форма* позволяет также разработчику создать форму «с нуля». Разница в действиях состоит только в том, в каком режиме открывается созданная форма. Создание форм с помощью этих трех команд

предполагает последующее сохранение созданной формы. При попытке ее закрыть появляется диалоговое окно с предложением сохранить форму.

Для создания формы с помощью *Мастера форм* необходимо пройти следующие шаги.

Шаг 1. Выбираем необходимую таблицу и поля и нажимаем кнопку *Далее* (рис 3.7). Если ничего не выбрано, то появляется диалоговое окно с требованием выбрать поля.

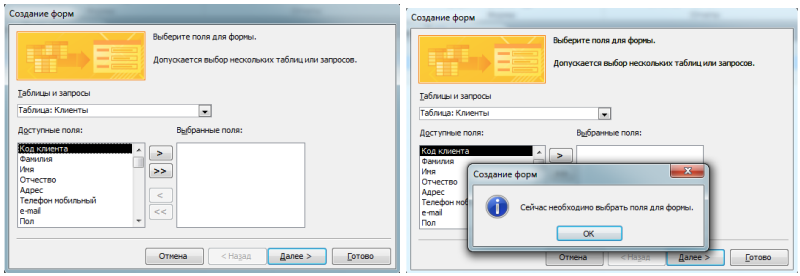


Рис. 3.7. Создание форм с помощью Мастера форм

Шаг 2. Выбор внешнего вида формы. Варианты: в один столбец, ленточный, табличный, выровненный. На рисунке 3.8 показаны образцы. Снова нажимаем кнопку *Далее*.

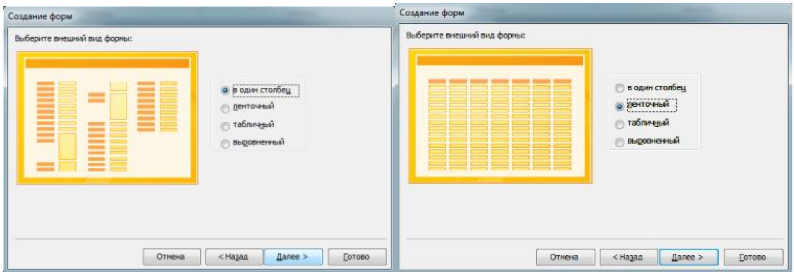


Рис. 3.8. Варианты вида формы

Шаг 3. Задание имени формы и выбор дальнейших действий с формой (рис. 3.9). После этого нажимаем кнопку *Готово*.

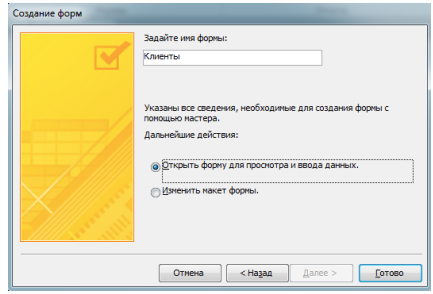
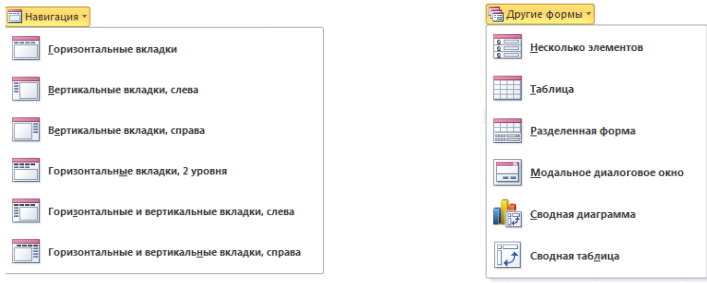


Рис. 3.9. Создание форм с помощью Мастера форм (Шаг 3)

Пункт меню *Навигация* позволяет создавать форму с вкладками разного вида (рис. 3.10 а)). Пункт меню *Другие формы* позволяет создать формы в виде таблицы, диаграммы и т.д. (рис. 3.10 б)).



а) б)

Рис. 3.10. Создание форм с помощью пунктов меню а) Навигация и б) Другие формы

Создание форм в режиме конструктора. При создании формы в режиме *Конструктор* в меню открываются новые инструменты – *Инструменты конструктора форм* (рис. 3.11). Они включают в себя три вкладки: *Конструктор*, *Упорядочить* и *Формат*.

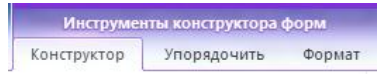


Рис. 3.11. Инструменты конструктора форм

Основная вкладка для работы с формами – *Конструктор* (рис. 3.12).

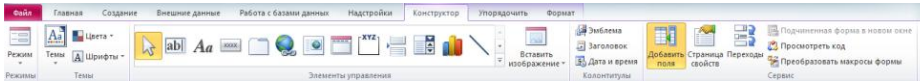
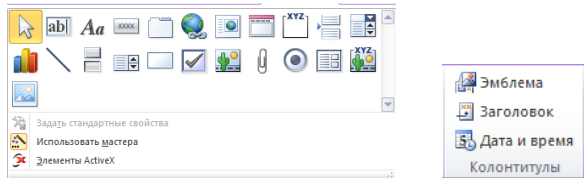


Рис. 3.12. Вкладка Конструктор

Для добавления каких-нибудь элементов управления, например надписей и полей на форму в конструкторе имеется пункт меню *Элементы управления* (рис. 3.13 а)). Также на форму можно добавить заголовок формы и примечание формы. Заголовок обычно состоит из картинки-эмблемы, названия формы и текущей даты. С помощью пункта меню *Колонтитулы* в форму можно также добавить колонтитулы (рис. 3.13 б)).



а) б)

Рис. 3.13. а) Элементы управления и б) Колонтитулы

На вкладке *Упорядочить* существует полезная функция упорядочивания полей на форме (рис. 3.14). Она позволяет привязать поля друг к другу и/или поле к его подписи и форматировать их размер.

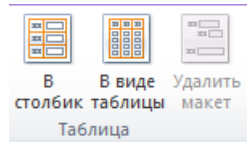


Рис. 3.14. Упорядочивание полей

Важно! Если планируется создать форму для просмотра или редактирования записей некоторой таблицы, то нужно не забыть привязать форму к нужной таблице. Для этого в *Окне свойств* формы на вкладке *Данные* в пункте *Источник записей* выбираем нужную таблицу из выпадающего списка (рис. 3.15). Источником данных для формы может служить также запрос выборки. В этом случае инструкция запроса (на языке SQL) напрямую вводится в данном свойстве. Создать запрос можно и с помощью конструктора запросов, открываемого при нажатии на кнопку с изображением трех точек.

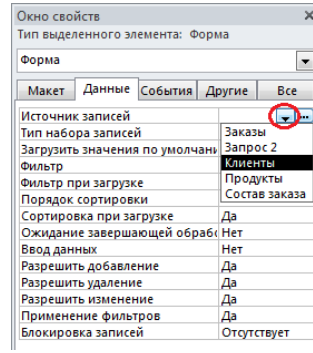


Рис. 3.15. Привязка формы к таблице

Если при создании формы в режиме *Конструктор* окно *Окно свойств* самой формы не открылось, то существует два варианта это сделать.

Вариант 1. Правой кнопкой мыши нажимаем на квадратик в левом верхнем углу формы и выбираем *Свойства* или щелкаем правой кнопкой мыши в любом месте формы и выбираем *Свойство формы* (рис. 3.16).

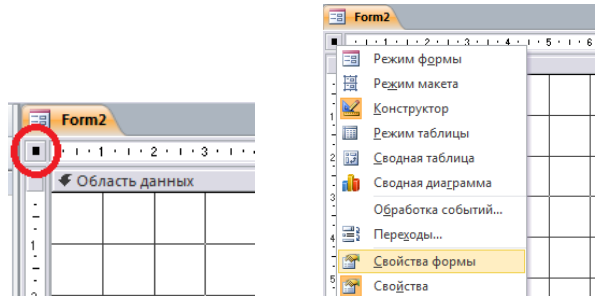


Рис. 3.16. Выбор Свойств формы

Вариант 2. Нажимаем на кнопку *Страница свойств* в меню *Сервис* на вкладке *Конструктор* (рис. 3.17).

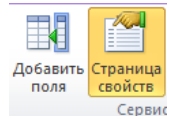


Рис. 3.17. Кнопки Страница свойств и Добавить поля в меню Сервис

Как только форма привязывается к источнику данных, можно приступить к добавлению полей. Для добавления поля можно выбрать мышью требуемый элемент управления в панели инструментов и затем щелкнуть левой клавишей мыши на форме в том месте где планируется расположить поле. *Примечание.* Вместе с полем производится также вставка надписи (слева), поэтому следует оставлять для ее некоторое место. Затем остается привязать вставленный элемент управления к полю источника данных, задав его свойство *Данные* (выбрав поле в выпадающем списке аналогично привязке самой формы к источнику данных). Чтобы упростить процесс добавления полей источника данных на форму, существует кнопка *Добавить поля* в пункте меню *Сервис* на вкладке *Конструктор* (рис. 3.17). После нажатия на нее открывается окно *Список полей* со всеми доступными полями в привязанном источнике данных (рис. 3.18).

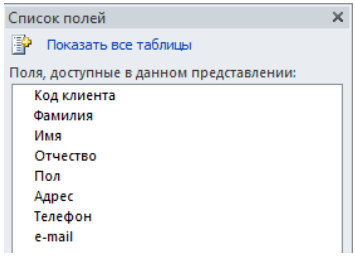


Рис. 3.18. Список полей

Если нажать на поле двойным кликом, то оно отобразится на форме в разделе *Область данных*. Также можно использовать перетаскивание мышкой, что более удобно.

Замена отображения внешних ключей на отображение соответствующих им данных из базовых таблиц (с помощью комбинированных списков). Формы являются средством ввода-вывода данных. Для пользователя их использование должно быть интуитивно понятно. Удобным способом отображения и ввода малоинформативных значений внешних ключей является подстановка соответствующих им данных из базовых таблиц. Пользователь видит информативные данные, например, фамилию клиента, а не значение его первичного ключа, а в данную таблицу в качестве внешнего ключа заносится первичный ключ выбранной записи. Эту задачу можно решить с помощью комбинированных списков.

Выбираем из меню *Элементы управления* элемент *Поле со списком* (рис. 3.19 а)) и добавляем на форму в *Область данных,* помня о том, что

перед элементом есть подпись к нему, на которую нужно оставить соответствующее место (рис. 3.19 б)).

а) б)

Рис. 3.19. Поле со списком

Добавленный элемент и подпись к нему можно упорядочить с помощью меню *Упорядочить -> В столбик* (рис. 3.14), тогда подпись никогда не будет закрываться полем. Если при создании поля со списком режим мастера не был включен, то после его добавления на форму следует задать его базовые свойства. Вид *Окна свойств* для *Поля со списком* представлен на рисунке 3.20.

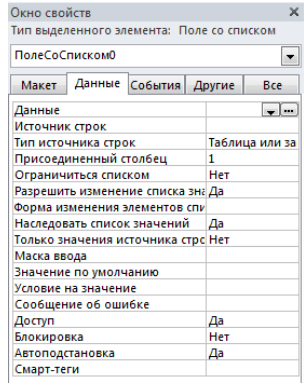


Рис. 3.20. Окно свойств для поля со списком

Для обеспечения работоспособности выпадающего списка обязательно задаем следующие свойства:

- На вкладке *Данные*:
  - ✓ *Данные* – выбираем название поля внешнего ключа, т.е. производим привязку к внешнему ключу таблицы. Например, запрос следующего вида выбирает из таблицы «Клиенты» 4 поля: первичный ключ таблицы и поля Фамилия, Имя, Отчество, записывая 3 последних в одно поле ФИО (примечание, в Access используется свой диалект языка SQL: конкатенация строк

производится с помощью оператора &, а строки заключаются в двойные кавычки):

```
SELECT Клиенты.[Код клиента], Клиенты.Фамилия & "  " & Клиенты.Имя & "  " & Клиенты.Отчество AS ФИО FROM Клиенты;
```

- ✓ *Источник строк* – нажимаем на  , открывается окно конструктора запросов. В нем создаем запрос выборки первичного ключа и по крайней мере одного из информативных полей базовой таблицы.
- ✓ *Ограничиться списком* – меняем на «Да».
- ✓ *Присоединенный столбец* – указываем номер поля, содержащего значения первичного ключа, в данном случае 1.

➢ На вкладке *Макет*:

- ✓ *Число столбцов* – для данного примера 2: один на первичный ключ и один на отображаемое поле ФИО.
- ✓ *Ширина столбцов* – первый столбец скрываем, устанавливая его ширину в **0 см**, далее через точку с запятой указываем нужную ширину второго столбца, например: **0см;5см**.

Остальные свойства задаются по желанию разработчика, например желательно задать название элемента управления. Результат создания *Поля со списком* отображен на рисунке 3.21. Итак, на форме «Заказы» вместо поля «Код клиента» пользователь видит поле «Клиент». Он заполняет его, выбирая из выпадающего списка ФИО клиента. Однако при этом в таблице «Заказы» заполняется поле «Код клиента» (внешний ключ) значением первичного ключа выбранной записи из таблицы «Клиенты».

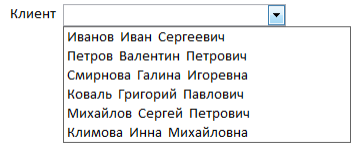


Рис. 3.21. Готовое поле со списком

Создание составной формы (master-detail). Для одновременного отображения и редактирования записей из двух таблиц, связанных отношением 1 - ∞, наиболее удобной является составная форма (master-detail form). Тогда записи из таблицы со стороны 1 будут отображаться в основной форме, а из таблицы со стороны ∞, имеющие совпадающие значения во внешнем ключе, – в подчиненной. Подчиненная форма располагается внутри основной (является элементом управления *Подчиненная форма*  для основной формы) и обычно представлена в виде таблицы или ленточ-

ной формы. Создать такую форму можно с помощью *Конструктора форм*.

Пример составной формы представлен на рисунке 3.22. Такая форма позволяет просматривать и обновлять информацию о клиенте и обо всех сделанных им заказах. При переходах от записи к записи в основной форме происходит автоматическая фильтрация записей подчиненной формы. Значения внешнего ключа подчиненной таблицы (в нашем случае «Код клиента» таблицы «Заказы») также назначаются автоматически при добавлении записей. Соответственно, вначале производится ввод нового клиента, а затем его заказов.

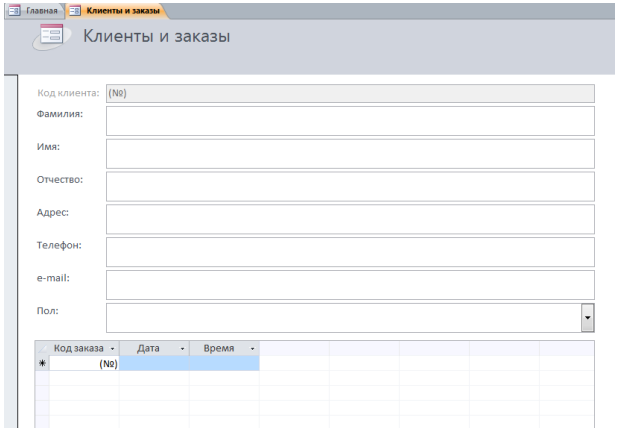


Рис. 3.22. Составная форма.

Для создания составной формы вначале создаем простую форму для подчиненной таблицы. Потом в *Окне свойств* этой формы на вкладке *Макет* меняем свойство *Режим по умолчанию* с *Простой формы* на *Режим таблицы* и сохраняем полученную форму. В подчиненной форме следует удалить элемент управления, привязанный ко внешнему ключу, поскольку он задаётся автоматически. Остальные внешние ключи (если есть) заменяются на выпадающие списки, подставляющие информативные данные из связанных таблиц. После этого создаем основную форму в обычном режиме и затем из меню *Элементы управления* добавляем в нее элемент *Подчиненная форма/отчет*. В *Окне свойств* подчиненной формы на вкладке *Данные* заполняем три свойства:

- ✓ *Объект-источник* – имя подчиненной формы;
- ✓ *Основные поля* – имя первичного ключа из основной таблицы;

- ✓ *Подчиненные поля* – имя связанного с основной таблицей внешнего ключа из подчиненной таблицы.

Если делать это с помощью *Мастера*, то достаточно во всплывающем окне выбрать уже готовую подчиненную форму и нажать *Готово*.

Создание вычисляемого поля. Для создания вычисляемого поля необходимо выполнить следующие действия:

1. Добавить на форму элемент *Поле*  из меню *Конструктор -> Элементы управления*.

   В *Поле* написать «=» и произвести ввод выражения (завершать ввод выражения точкой с запятой не требуется). Например, рассчитаем возраст клиента:

   ```
   =DateDiff("yyyy";[Дата рождения];Date())
   ```

   Это выражение можно записать в виде `=Year(Date())-Year([Дата рождения])`, но первое выражение будет работать всегда, а второе – в зависимости от текущей даты может приводить к ошибкам.

   В составных формах часто возникает необходимость вычислять итоговые значения для подчиненного набора записей, например количество заказов или общая сумма покупки. Для этого используются функции DCount(), DSum(), DLookup(). Рассмотрим использование данных функций. Выражение

   ```
   =DCount("[Код заказа]";"Заказы"; "[Код клиента]=" & IIf(IsNull([Код клиента]);0;[Код клиента]))
   ```

вычислит количество заказов клиента. Функция IIF(условие, выражение1, выражение2) в приведенном выше примере позволяет заменить пустое значение на 0. Выражение

```
=DSum("[Стоимость]";"[Расчет стоимости]"; "[Код клиента]=" & IIf(IsNull([Код клиента]);0;[Код клиента]))
```

рассчитает суммарную стоимость всех заказов клиента. Функция DLookup() используется для вывода значения поля, не принадлежащего источнику записей. Например, в форме «Товары» можно вывести сведения о поставщике, хранящиеся в связанной отношением $\infty - 1$ таблице «Поставщики»:

```
=DLookUp("[Должность]";"[Поставщики]";"[КодПоставщика]=" & [КодПоставщика])
```

Здесь условие отбора "[КодПоставщика]=" & [КодПоставщика] реализовано с помощью конкатенации условия [КодПоставщика]= и значения, возвращаемого одноименным элементом управления формы «Товары» (где КодПоставщика – это поле таблицы «Поставщики»).

Вычисление с помощью агрегатных функций. Итоговые вычисления производятся с помощью агрегатных функций, имеющих тот же смысл и написание, как в запросах. Аргументами их могут быть только имена полей. Множеством данных для подобных вычислений будут все записи, отображаемые в форме. Размещать такие поля с вычислениями следует в области примечаний формы. Например, `= Sum([Цена])` будет вычислять суммарную цену всех имеющихся в таблице «Товары» товаров, а `= Count([Код заказа])` вычислит сколько всего заказов в таблице «Заказы».

Задания

1. Создать DSN базы данных с помощью ODBC администратора.
2. Реализовать в MS Access подключение к удаленной базе данных посредством технологии ODBC.
3. Создать схему данных в MS Access, дублирующую схему подключающей БД.
4. Разработать формы:
   - **простые формы** для каждой из таблиц,
   - минимум одну с**оставную форму (master-detail)** и многостраничную (набор вкладок) формы. Требования: разрабатывать формы в режиме **конструктора**, применить разнообразное форматирование и оформление данных, использовать все возможные элементы управления (кнопки, переключатели, списки, поля со списком и т.д.).
5. **Заменить отображение внешних ключей** на отображение соответствующих им данных из базовых таблиц (с помощью комбинированных списков);
6. По крайней мере, в одной форме определить вычисляемое поле;
7. Определить пользовательский формат данных и маску ввода для поля, допускающего форматированных ввод данных
8. По мере необходимости разработать формы, требуемые для решения задач базы данных.

Форма отчета. Формой отчета являются база данных выбранной тематики и проект MS Access, реализующий задания, представленные в разделе «Задания».

ЛАБОРАТОРНАЯ РАБОТА № 4

РАЗРАБОТКА ОТЧЕТОВ СРЕДСТВАМИ MS ACCESS

Цель работы. Практическое освоение разработки отчетов средствами MS Access.

Краткие сведения

Отчеты являются наилучшим средством для предоставления данных в виде печатного документа и сочетают в себе возможности форматированного вывода данных с группированным представлением большого количества записей из нескольких таблиц. Так как число записей и число полей может быть велико, то отчеты могут выводится на нескольких страницах. Кроме вывода самих данных отчеты позволяют производить разнообразные вычисления и расчет итоговых значений. Отчет является автоматически генерируемым законченным документом и его нельзя редактировать. Единственно, что возможно, так это просмотреть его в режиме предварительного просмотра перед печатью либо внедрить в состав другого документа, например, в документ MS Word или MS Excel.

Отчеты и формы имеют много общего. Однако, в отличие от форм, отчеты не предназначены для ввода и правки данных в таблицах и в них нет смысла размещать такие элементы управления, как кнопки, поля со списком, группы и т.п. Отчеты обладают следующими возможностями:

- имеют широкие возможности для форматирования текста;
- имеют широкие возможности для группировки, а также для вычисления промежуточных и общих итогов. В отчете каждая группа и итоги по ней представлены отдельно, и можно определить до 10 уровней группировки;
- можно производить сложные вычисления не только внутри некоторой группы, но и по нескольким группам одновременно;
- позволяют легко создать такие документы, как счета, почтовые наклейки и т.д.

Создавать отчеты в Access можно с помощью автоотчетов, мастера и с помощью конструктора (рис.4.1). Наиболее удобно создавать формы с помощью автоотчетов. Для этого необходимо только указать базовую таблицу или запрос и вид отчета. Мастер позволяет создавать более сложные отчеты с группировкой и расчетом итогов. Но основным средством разработки отчетов является конструктор.

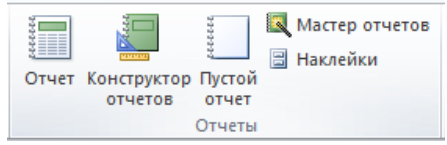


Рис. 4.1. Кнопки создания отчетов на вкладке Создание.

Отчет, как и форма, имеет область заголовка, примечания и область данных. Дополнительно к ним (как для любого печатного документа) можно определить верхний и нижний колонтитулы. Заголовок и примечание отчета, верхний и нижний колонтитулы не являются обязательными и их можно добавить или удалить по желанию с помощью уменьшения их размера до нуля. Но, в отличие от форм, число областей не является фиксированным и зависит от данных, размещаемых в отчете. Некоторые области могут повторяться многократно, отражая соответствующую информацию из базовой таблицы (запроса), а некоторые будут представлены в единственном числе. К примеру, печать области данных повторяется для каждой записи базового набора, а заголовок отчета печатается всего один раз на первой странице. Рассмотрим подробнее, из каких областей состоит отчет, и какие данные можно в них размещать:

1. Область заголовка и область примечания отчета. Эти области располагаются соответственно один раз в начале и конце отчета. Обычно в область заголовка отчета включают название отчета, дату его создания и логотип предприятия. В области примечания помещают расчет итоговых результатов по данным всего отчета и некоторую служебную информацию. Если отчет многостраничный, то эта информация будет размещена на последней странице.
2. Верхний и нижний колонтитулы. Эти области предназначены для вывода информации, повторяющейся для каждой страницы. Например, там можно поместить номер страницы, шапку для вывода некоторой табличной информации и т.п.
3. Область данных. В области данных размещается вывод основных данных. Вертикальные размеры этой области должны быть минимальными (фактически по размеру элементов управления, отвечающих за вывод данных), поскольку область данных повторяется для каждой записи базового набора. В свою очередь область данных также может иметь сложный вид, так как обычно в отчете выводится группированная информация. Тогда для каждой группы можно дополнительно задать свои заголовок и примечание.

4. Область заголовка и область примечания группы. В этих областях помещается та информация, которая является общей для группы. Формирование данных областей производится одновременно с заданием группировки. Например, здесь можно выводить заголовок группы (поле, по которому производится группировка и выбирается обычно в качестве заголовка группы). В области примечания обычно размещается расчет итогов (так называемых промежуточных итогов) по каждой группе.

Так как отчеты содержат большое количество записей, то для удобства просмотра они должны быть определенным образом сгруппированы и упорядочены. Если отчет основан на запросе, то он игнорирует все условия сортировки и группировки, заданные для него. Группировка и сортировка данных задается прямо в отчете с помощью окна *Группировка, сортировка и итоги*, встроенного внизу отчета (рис. 4.2).

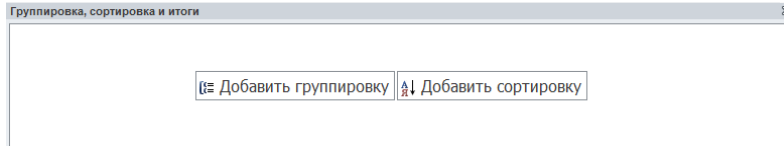


Рис.4.2. Окно Группировка, сортировка и итоги

Вызвать появление этого окна можно с помощью кнопки 

. В данном окне можно определить до 10 полей или выражений для группировки данных (то есть группировка может осуществляться по значению выражения, например, = [Фамилия] & " " & [Имя] & " " & [Отчество]). Первое поле или выражение определяет основную группу, последующие – подгруппы внутри группы предыдущего уровня. В этом же окне можно включить либо выключить вывод областей заголовка и примечания для каждой из групп. Дополнительно можно установить еще другие свойства: *начиная с А* или *От старых к новым* или *От минимального к максимальному* в зависимости от вида поля, *По всему значению* и *Не удерживать группу на одной странице*. Свойство *Не удерживать группу на одной странице* устанавливается в *Удерживать группу на одной странице*, если есть желание, чтобы записи, относящиеся к одной группе, размещались на одной странице. Есть возможность выделять данные в новую группу не по полному изменению значения поля, участвовавшего в группировке, а по смене, скажем, его части, или когда значения поля попадают в некоторый диапазон. Вид и задаваемый размер диапазона зависят от типа поля. Для текстовых значений можно указать, чтобы Access начинал новую группу

при изменении первого или нескольких начальных символов значения поля, а для числовых или типа *Дата/время* задать интервал значений. Например, 10. Тогда будут формироваться следующие группы: от -20 до –11, от -10 до –1, от 0 до 9, от 10 до 20 и т.д. Для поля типа *Дата/время* можно устанавливать следующие интервалы: *По годам, По кварталам, По месяцам, По неделям, По часам*, и *По минуте* и дополнительно указать соответственно количество лет, кварталов, месяцев и т.д. в каждой группе.

Создание отчетов в режиме конструктора почти ничем не отличается от создания форм. Требуется только установить сортировку и группировку, включить области заголовков и примечаний и разместить требуемые поля (или другие элементы управления) в соответствующих областях отчета. При этом необходимо помнить, что имеет смысл использовать только те элементы управления, которые можно вывести на печать, и что вывод данных будет определяться областью размещения соответствующих элементов управления. Например, одно и тоже вычисляемое поле может выполнять совершенно разные действия. Поле со свойством *Данные* =Count([Код заказа]) будет подсчитывать количество заказов для каждого клиента, если оно помещено в область заголовка или примечаний для группы *Клиенты*, и общее количество заказов, если будет помещено в области заголовка/примечаний всего отчета.

При создании отчета важно не только определить данные, которые должны быть представлены в отчете, но и нужным образом оформить отчет как документ: правильно разбить его на страницы, пронумеровать страницы, выделить наиболее важные данные. Если не все нужные данные помещаются на одной странице, Access позволяет вставить принудительный разрыв страницы. Текущая дата вводится с помощью функций Now() или Date(). Номер страницы представляется с помощью системных переменных Page и Pages. Например: "Страница" & [Page] & "из" & [Pages]. В результате получите: Страница 5 из 25. Вставить вывод даты и номера страницы можно также с помощью команд меню.

Для вычисления промежуточных (по группе) итогов необходимо разместить вычисляемое несвязанное поле в области примечаний группы. Например, Sum([Количество])*[Цена]) в качестве значения «данные» для вычисляемого поля. То же поле, но размещенное в области примечаний отчета, уже будет вычислять общий итог по всему отчету. Для вычисляемых полей в отчетах появляется новое интересное свойство – *Сумма с накоплением*. Если поле размещено в области данных и это свойство установлено в *Для группы*, то Access будет суммировать данные в каждой записи и сбрасывать итоговое значение в 0 после начала каждой группы. Если поле размещено в области примечаний группы, Access будет накап-

ливать итоговые суммы для всех групп этого уровня до тех пор, пока не встретится уровень группировки более высокого уровня. Таким образом, можно подсчитать количество купленных товаров по категориям и т.д. Если же свойство *Сумма с накоплением* установлено в *Для всего*, то Access будет суммировать данные по всем записям до конца отчета. Примером использования свойства *Сумма с накоплением* может служить нумерация строк в отчете. Если разместить несвязанное поле в области данных отчета и задать свойство *Данные* =1, а свойство *Сумма с накоплением* в *Для группы*, то можно пронумеровать к примеру заказы в пределах группы. Если установить свойство *Сумма с накоплением* в *Для всего*, то можно установить сквозную нумерацию.

Иногда требуется, чтобы определенные поля выводились в отчете в зависимости от их значения или от значения других элементов в отчете. В Access существует функция IIF(условие; выр 1; выр 2), возвращающая в зависимости от выполнения условия значения либо *выр 1*, либо *выр 2*. Далее, разумно не выводить поля с пустым значением. К тому же желательно, чтобы остальные поля при этом перемещались вверх, чтобы в отчете не оставалось пустого места. Для того чтобы поле «исчезало», когда его значение окажется пустой строкой, необходимо установить свойство *Сжатие* в *Да*. Свойство *Не выводить повторы* дополнительно позволяет не выводить элемент управления отчета, если значение этого элемента управления совпадает с его значением в предыдущей записи.

Как и формы, отчеты могут быть также составными, то есть иметь подчиненные отчеты. Использование подчиненных отчетов в ряде случаев позволяет просто упростить вывод связанных записей и задание группировки в отчете. Но в ряде случаев без подчиненных отчетов не обойтись. Например, требуется уровень вложенности более двух, представление итогов в виде отдельной таблицы наряду с выводом всех данных или главный отчет может быть простым контейнером для размещения нескольких подчиненных отчетов из несвязанных источников данных.

В качестве примера рассмотрим создание отчета, предоставляющего информацию по клиентам и сделанным ими заказам. Дополнительно потребуем от отчета вычисления промежуточных и общих итогов. Для этого в окне Г*руппировка, сортировка и итоги* создадим одну группу по ключевому полю таблицы *Клиенты* и включим отображение заголовка и примечания для группы. Далее разместим элементы управления *Надпись* и *Поле* как показано на рис.4.3 и определим для них соответствующие данные.

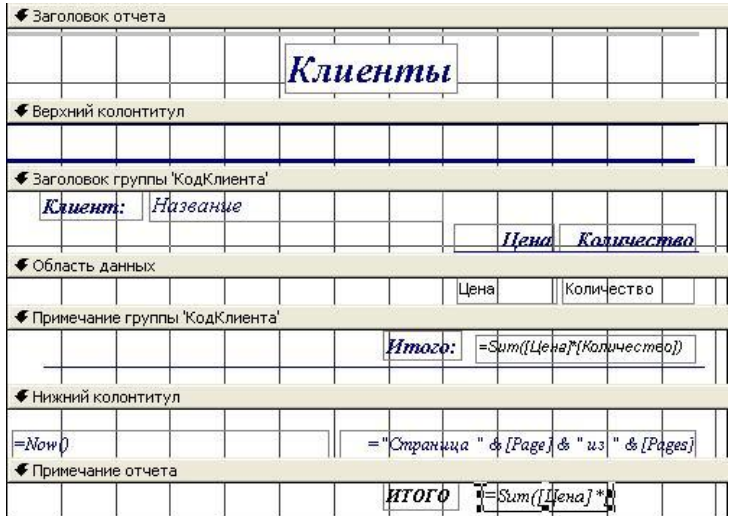


Рис.4.3. Отчет в режиме конструктора

Тот же отчет в режиме предварительного просмотра будет выглядеть следующим образом (рис.4.4)

Клиенты

Клиент: Сидоров Петр Петрович

Цена	Количество
215.00р.	20
100.00р.	6
466.00р.	2
Итого: 5 812.00р.	

Клиент: Иванов Иван Иванович

Цена	Количество
320.00р.	10
120.00р.	5
288.00р.	1
140.00р.	5
Итого: 4 988.60р.	

Рис. 4.4. Отчет в режиме предварительного просмотра

В отчете можно упорядочивать поля и надписи точно так же, как и в формах, на вкладке *Упорядочить*. Полезным свойством упорядочивания является *Упорядочить в столбик*, если его применить только к одному полю и его надписи. К этой паре потом можно будет прикреплять и другие поля и надписи. Так можно получить очень удобные конструкции.

Например, можно объединить поле и надпись Фамилия, а далее присоединить к ним поля Имя и Отчество и переименовать надпись Фамилия в Клиент (рис.4.5). Эту конструкцию можно перемещать целиком.

Рис. 4.5. Упорядочивание нескольких полей

Задания

1. Разработать отчеты:
   - простой отчет в виде таблицы,
   - **отчет с группировкой данных и вычислением промежуточных и общих итогов**.

   Требования: разрабатывать отчеты в режиме конструктора, применить разнообразное форматирование и оформление данных, включить отображение даты, номера страницы и количества страниц;
2. Разработать отчеты, необходимые для решения задач базы данных.

Форма отчета. Формой отчета являются база данных выбранной тематики и проект MS Access, реализующий задания, представленные в разделе «Задания».

ЛАБОРАТОРНАЯ РАБОТА № 5

АВТОМАТИЗАЦИЯ ПРИЛОЖЕНИЯ В MS ACCESS

Цель работы. Практическое освоение принципов автоматизации интерфейса базы данных на основе форм MS Access.

Краткие сведения

Создание удобного и интуитивно понятного интерфейса, предоставляющего доступ ко всем объектам БД и автоматизирующего выполнение основных задач, является неотъемлемой частью разработки любой БД, даже если не планировалось ее распространять. Интерфейс пользователя в Access реализуется с помощью форм, панелей инструментов и меню. Поскольку Access является СУБД и предоставляет полный набор операций, доступных через меню и встроенные панели инструментов, то создание пользовательского меню и панелей инструментов не является первоочередной задачей. Гораздо более важно научиться создавать удобные в работе и богатые функциональными возможностями формы. Если разрабатываемая БД небольшая и фактически предназначена для выполнения одной или небольшого количества взаимосвязанных задач, то достаточно создать одну главную форму, автоматически вызываемую при открытии БД, и ряд вспомогательных форм, открываемых обычно с помощью командных кнопок, размещенных на главной форме. Закрытие этой формы можно связать с закрытием всего приложения. Если же разрабатываемая БД большая и предназначена для выполнения ряда задач, то не обойтись без создания главной переключательной формы, которая автоматически вызывалась бы при открытии БД и оставалась открытой все время работы приложения. Создать такую форму можно самостоятельно в режиме конструктора или с помощью мастера создания переключательных форм, а задать ее автоматическое появление на экран наряду с заданием названия приложения, значка приложения и т.п. можно с помощью диалогового окна *Параметры запуска*, вызываемого по одноименной команде меню.

Автоматизация приложения в Access достигается за счет обработки событий, происходящих в формах и отчетах с помощью макросов и модулей. Макросы и программы, написанные на языке Visual Basic for Applications (VBA) и входящие в состав модулей, позволяют выполнить ряд запрограммированных действий и изменять состояние любого объекта БД. Из макросов и модулей можно ссылаться на любой объект, читать и менять его свойства.

Ссылки на объекты. При проведении вычислений и автоматизации форм и отчетов часто приходится ссылаться на саму форму/отчет или на какой-нибудь элемент управления в ней, чтобы считать или установить его свойства или значение. Все объекты Access располагаются внутри коллекций, поэтому доступ к некоторому объекту БД можно получить, вначале указав имя коллекции, а затем через восклицательный знак имя объекта в коллекции. Помимо операции "!" можно указать в скобках индекс или имя объекта. Индекс является порядковым номером открытой формы и начинается с нуля. Коллекции форм и отчетов являются глобальными и к объекту такой коллекции можно обращаться напрямую. Например: `Reports![Годовой отчет], Forms!Заказы` (иначе `Forms(0)` или `Forms("Заказы")`).

Для указания ссылки на свойство объекта используется операция ".", за которой следует имя свойства `Forms![Catalog Item].Visible`

Получив ссылку на объект или свойство, можно считать ее значение или, напротив, присвоить ей некоторое допустимое значение. Например:

```
frm = Forms![Catalog Item],
Forms![Catalog Item].Visible = False
```

По аналогии с коллекциями для основных типов объектов сами формы и отчеты становятся коллекциями имеющихся в них элементов управления. Синтаксис ссылки на конкретный элемент управления и на его свойства аналогичен вышеприведенному.

```
Forms![Заказы]![код_заказа] и
Forms![Заказы]![код_заказа].Visible
```

Несколько сложнее дело обстоит со ссылками на подчиненные формы или отчеты. Когда форма внедряется внутрь другой формы, то она будет являться элементом управления *Подчиненная форма* для основной формы (т.е. ссылка к самой подчиненной форме или к ее свойствам ничем не будет отличаться от ссылки, скажем, на элемент *Поле*).

```
Forms![Заказы]![Подчиненная Заказы].Visible
```

Одним из свойств элемента управления *Подчиненная форма* является свойство *Форма* (Form), которое позволяет ссылаться в свою очередь на подчиненную форму и на элементы управления в ней

```
Forms![Заказы]![Подчиненная Заказы].Form![Код_заказа]
```

Для ссылки на свойство элемента управления внутри подчиненной формы остается добавить еще точку и имя свойства. При работе с макросами и модулями необходимо иметь в виду, что доступ возможен только к элементам управления открытых форм или отчетов. Если же есть желание иметь доступ к элементам управления, но не видеть форму открытой, то

можно просто установить свойство (Visible) этой формы в False, т.е. скрыть ее.

Объекты в Access могут иметь свойство, используемое по умолчанию. Это свойство применяется в том случае, когда имя свойства в ссылке явно не указано. Например, у элемента управления *Поле* по умолчанию используется свойство *Значение* (Value), поэтому ссылка `Forms!Товары!Цена` позволяет получить доступ к значению, отображенному в текстовом поле Цена. Следовательно, можно просто присвоить некоторое значение элементу управления *Поле*, а вместе с ним тому полю базового набора, с которым этот элемент управления связан. Например, `Forms!Товары!Цена = 100` или `price = Forms!Товары!Цена`.

События Access. Событие – это распознаваемое изменение состояния любого объекта MS Access или операционной системы. Может представлять любое действие, инициируемое пользователем или самой системой. Такие действия, как открытие окна, состоят из нескольких последовательно происходящих событий. Поскольку автоматизация приложения происходит за счет обработки событий, то конечный результат будет зависеть не только от умения правильно обрабатывать отдельные события, но и от знания последовательности их выполнения. Большое количество действий, например, вставка либо удаление записи, имеют по два события для их обработки. Первое происходит непосредственно перед выполнением действия, второе – сразу после него. События, происходящие перед выполнением некоторого действия, можно отменять. Отмена события фактически отменяет выполнение самого действия. Определить, является ли событие отменяемым, можно по наличию в списке его аргументов параметра Cancel. Тогда для отмены события достаточно будет установить его в True. Здесь мы рассмотрим основные события (весь список можно найти в справочной системе), условно разделив их на следующие группы:

- *События данных*. Происходят при изменении данных в элементе управления или в форме и при переходе от записи к записи:
  - *Текущая запись*. Происходит при обращении к источнику данных формы/отчета. Возникает как при открытии формы, так и при переходе от записи к записи.
  - *Удаление.* Происходит непосредственно перед реальным удалением записи.
  - *До подтверждения Del*. Используется для отмены появления окна запроса на подтверждение удаления и самого удаления.
  - *После подтверждения Del*. Используется для проверки статуса удаления записи.

– *До вставки*. Происходит сразу перед вставкой записи (ввод первого символа в любое поле). Может использоваться для проверки, разрешена ли вставка записи.

– *После вставки*. Происходит сразу после вставки записи.

– *До обновления*. Возникает при обновлении данных как в самой записи, так и в элементе управления. Используется обычно для проверки условий целостности.

– *После обновления*. Возникает сразу после обновления данных в записи или элементе управления. При обработке события можно восстановить старые данные, сохраняемые в свойстве *Old Value* элемента управления.

– *Изменение*. Событие редактора. Возникает при вводе/удалении каждого символа.

– *Внесены изменения*. Позволяет проверить, были ли изменения в записи.

– *Отсутствие в списке*. Возникает в поле со списком при вводе значения, отсутствующего в списке.

- *События окна*. Происходят при открытии и закрытии форм и отчетов и при изменении их размеров:

– *Открытие*. Происходит непосредственно перед отображением первой записи.

– *Закрытие*. Происходит непосредственно перед тем, когда форма будет удалена из экрана.

– *Загрузка*. Происходит после события *Открытие* при загрузке данных в форму.

– *Выгрузка*. Происходит перед событием *Закрытие* при выгрузке данных.

– Изменение *размера*. Происходит при изменении размеров формы/отчета, а также при их открытии.

- *События фокуса*. Происходят при получении и потери фокуса и при активации/деактивации элемента управления или самой формы:

– Вход.

– Выход.

– Получение фокуса. Происходит, когда форма или элемент управления получает фокус, т.е. становится подсвеченной либо выделяется другим цветом.

– *Потеря фокуса.* Происходит, когда форма или элемент управления теряет фокус.

– *Включение.* Возникает, когда форма/отчет становится активной. Применяется для связывания появления панели инструментов или т.п. с появлением формы.

– *Отключение.* Возникает, когда активной становится другая форма/отчет.

- *События клавиатуры.* Происходят при нажатии на клавиши клавиатуры:

– *Клавиша вниз. Происходит при нажатии клавиши. Имеет два параметра KeyKode и Shift. По значениям этих параметров можно определить, что было нажато.*

– *Клавиша вверх. Происходит при отпускании клавиши. Имеет те же параметры.*

– *Нажатие клавиши. Генерируется непрерывно, когда клавиша нажимается и остается* нажатой.

- *События мыши.* Происходят при нажатии клавиш мыши или при ее перемещении:

– *Нажатие кнопки. Происходит при нажатии левой клавиши мыши. Это событие вызывается также автоматически при нажатии клавиши Enter для имеющей фокус кнопки или выборе элемента из списка.*

– *Двойное нажатие кнопки. Происходит после двойного нажатия клавиши мыши.*

– *Перемещение указателя.* Генерируется непрерывно, пока происходит перемещение мыши. Система определяет границы объектов и генерирует это событие поочередно для всех объектов, в которые попадает указатель мыши. Имеет параметры *Button, Shift, X и Y*. Параметры *Button и Shift* служат для определения состояния нажатых кнопок мыши и клавиш *Alt, Shift, Ctrl*.

– *Колесико мыши, Кнопка вниз и Кнопка вверх.* Дополнительные события мыши.

Если форма имеет хотя бы один доступный элемент управления, то при ее открытии возникает следующая последовательность событий: *Открытие* → *Загрузка* → *Изменение размера* → *Включение* → *Текущая запись* → *Вход* (э)→ *Получение фокуса* (э). Буква э в скобках означает, что событие относится к элементу управления, а не к форме. В следующих примерах буква *ф* в скобках будем обозначать событие, происходящее для самой формы, если это и так не ясно из контекста. Соответственно, при закрытии формы: *Выход* (э)→ *Потеря фокуса* (э) → *Выгрузка* → *Отключение* → *Закрытие*. При переключении между двумя открытыми формами

цепочка событий следующая: *Потеря фокуса* (э1) → *Отключение* (ф1) → *Включение* (ф2)→ *Вход* (э2)→ *Получение фокуса* (э2), а при переключении между двумя элементами управления одной формы: *Выход* (э1) → *Потеря фокуса* (э1) → *Вход* (э2)→ *Получение фокуса* (э2).

При изменении данных в элементе управления при каждом нажатии клавиши клавиатуры возникает следующая цепочка событий: *Клавиша вниз* → *Нажатие клавиши* → *Внесены изменения* → *Изменение* → *Клавиша вверх.* При последующем переходе к другому элементу управления дополнительно происходят: *До обновления* → *После обновления* → *Выход* → *Потеря фокуса*.

При добавлении записи в форму (производится ввод первого символа в поле пустой записи) цепочка событий следующая: *Текущая запись* → *Вход* (э)→ *Получение фокуса* (э) → *До вставки*→ *Изменение* (э) → *До обновления* (ф) → *После обновления* (ф) → *После вставки*. При удалении: *Удаление* → *До подтверждения Del* → *После подтверждения Del* → *Текущая запись* → *Вход* (э)→ *Получение фокуса* (э). Последние три события происходят вследствие того, что при удалении фокус переходит на следующую запись.

Макросы. Макрос представляет собой набор из одной или более макрокоманд, которые выполняются либо последовательно, либо в порядке, заданном определенными условиями. Каждая макрокоманда имеет собственное имя и один или несколько аргументов (простейшие макросы могут и не иметь аргументов). Использование макросов оправдано тем, что для их создания нет необходимости в серьезном изучении языка программирования. Использование макрокоманд интуитивно понятно, поскольку название макрокоманды фактически совпадает с выполняемым ей действием. Например, *ЗадатьЗначение*, *ОткрытьФорму*. Но программирование с помощью макросов имеет ряд серьезных ограничений. Набор макросов ограничен, макросы не могут возвращать значения, с их помощью невозможна обработка ошибок и т.д.

С помощью макросов можно выполнять следующие действия:

1. Открывать и закрывать любые объекты вашей БД в любом режиме. Для форм дополнительно можно ограничивать число выводимых записей по некоторому условию. Для отчетов можно запускать их на печать и экспортировать в формате TXT, RTF, XLS и HTML;
2. Выполнять запросы и задавать их параметры с помощью элементов управления любой открытой формы;

3. Запускать другие макросы и процедуры VBA, прерывать выполнение макросов и отменять событие, вызвавшее макрос, и даже выходить из приложения;

4. Устанавливать значения любых свойств любого элемента управления формы или отчета;

5. Осуществлять поиск записей в базовой таблице или запросе форм и отчетов и переходить к любой из них;

6. Переопределять либо создавать системное меню и выполнять любую команду любого меню Access;

7. Перемещать, изменять размеры, сворачивать, скрывать и т.д. формы и отчеты внутри рабочего окна Access. Можно также передавать фокус любой форме и любому элементу управления в ней;

8. Выводить на экран информационные сообщения либо выдавать звуковые сигналы;

9. Переименовывать, удалять, импортировать/экспортировать объекты и внешние данные;

10. Запускать другие приложения, осуществлять обмен информацией с помощью механизма DDE или буфера обмена. Можно даже передать последовательность нажатий клавиш в открытое приложение.

Макросы создаются с помощью конструктора макросов (смотри рис. 5.21**Ошибка! Источник ссылки не найден.**). Макросы можно также объединить в группы для уменьшения их общего числа и лучшей ориентации среди них. Для создания группы макросов необходимо вывести столбец *Имя макроса* в окне конструктора макросов с помощью соответствующей команды меню. Чтобы сослаться на макрос, входящий в группу, требуется указать имя группы макросов и через точку имя макроса. Можно задать условие на выполнение определенных макрокоманд. Для этого необходимо добавить столбец *Условие* в окно конструктора макросов с помощью соответствующей команды меню. Создание условий ничем не отличается от задания условий на значения полей таблиц.

Если условие действует на группу макрокоманд, то для того чтобы не повторять одно и то же условие для нескольких последовательно идущих макрокоманд, можно вместо условия поставить многоточие "…". Если условие принимает значение *Истина*, то Access запускает макрокоманду в этой строке и все макрокоманды в расположенных ниже строках, которые имеют многоточие вместо условия. Затем продолжается выполнение следующих макрокоманд внутри данного макроса, пока не встретится макрокоманда с другим условием, другой макрос или конец группы макросов. Если условие принимает значение *Ложь*, то Access игнорирует макроко-

манду данной строки и всех следующих за ней макрокоманд с многоточием и переходит к строке, содержащей новое условие или не содержащей условий вообще.

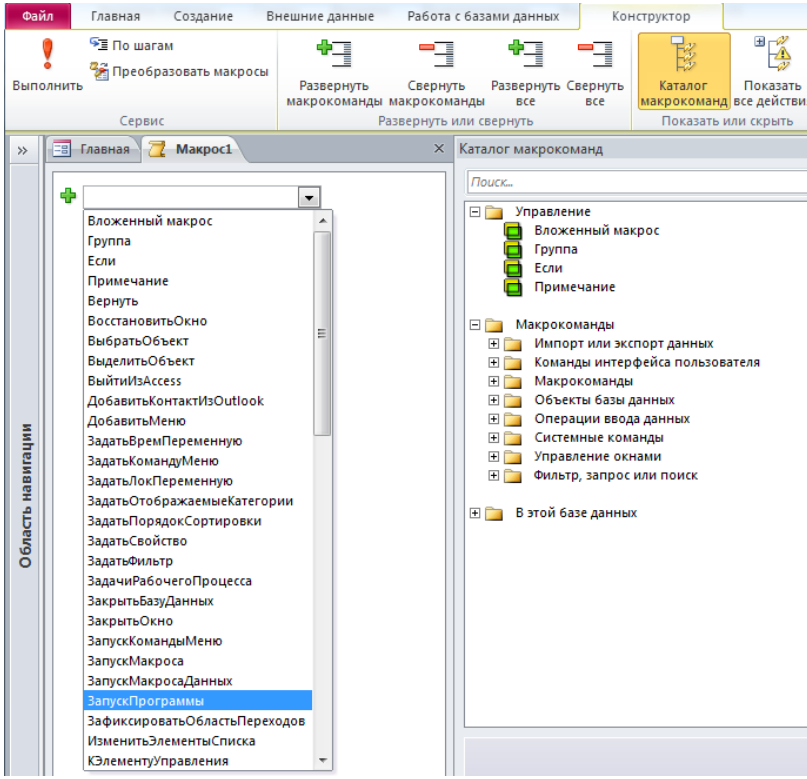


Рис. 5.1. Окно конструктора макросов

Для макросов невозможно задание альтернативного ветвления, поэтому для создания макроса с альтернативным ветвлением необходимо вначале определить группу макрокоманд с одним условием, а затем сразу же другую группу с альтернативным условием.

Макросы можно запустить на выполнение из другого макроса или процедуры VBA, но чаще всего макросы привязываются к определенным событиям в формах и отчетах и служат для их обработки. Для связи макроса с определенным событием необходимо задать имя макроса в строке обработки этого события в бланке свойств того объекта, событие которого хотите обработать. Почти все макрокоманды имеют аргументы, т.е. тре-

буют задания дополнительных данных, определяющих способ их выполнения. Например, с событием *Нажатие кнопки* элемента управления *Кнопка* можно связать макрос, содержащий макрокоманду *ОткрытьФорму*. Тогда после нажатия кнопки произойдет открытие формы, определенной в аргументе этой макрокоманды.

Существует зарезервированное имя макроса AutoExec для автоматического выполнения при открытии базы данных. Т.е. если присвоить это имя какому-либо макросу, то он начнет выполняться сразу же после запуска БД. Если же необходимо отключить его выполнение, то при открытии БД надо удерживать нажатой клавишу Shift. При одновременном задании параметров запуска вначале происходит обработка действий, определенных в окне *Параметры Запуска*, а затем макроса AutoExec.

Почти все макросы можно продублировать с помощью функций VBA.

Модули. Модули являются объектом базы данных, объединяющим одну или несколько процедур либо функций, написанных на языке Visual Basic for Applications (VBA). Краткое описание языка VBA будет дано в приложении. Модули следует использовать вместо макросов в следующих случаях:

- необходима обработка ошибок;
- необходимо возвращать и передавать параметры;
  необходимо создание общих функций;
- необходимо во время работы (динамически) создавать новые объекты БД;
- необходим вызов функций Windows или обмен данными через OLE;
- необходима функциональность, недостижимая с помощью макросов;
- необходимо производить действия более чем с одной БД одновременно.

Входящие в модуль процедуры/функции объединены общей областью декларации. В ней устанавливаются определения и правила, являющиеся общепринятыми для модуля, а также переменные, общие для всех процедур/функций модуля. Процедуры и функции могут иметь аргументы. Отличием функции от процедуры является то, что первая может в точку вызова вернуть некоторое значение, а вторая – нет. Функции и процедуры, записанные в модулях, доступных через закладку *Модули* окна базы данных (стандартные или общедоступные модули), являются доступными для всех объектов БД. Их можно использовать в следующих местах:

- во всех процедурах и других функциях вашей БД;
- в выражениях вычисляемых полей в формах и запросах;
- в выражениях, определяющих условия выполнения макрокоманд в макросах.

Кроме общедоступных модулей в каждой форме и отчете имеются свои локальные (иначе привязанные) модули. Программы локальных модулей загружаются динамически вместе с объектом, их содержащим. Процедуры в них являются локальными по отношению к данному объекту и используются преимущественно для обработки событий. Имя таких процедур состоит из имени объекта и имени события и не подлежит изменению. Если требуется обработка некоторого события, то проще всего создать процедуру его обработки из окна свойств того объекта, события которого необходимо обработать. Для этого надо нажать кнопку с многоточием в строке обрабатываемого события и выбрать построение программ. После этого автоматически будет создано тело процедуры, связанной с определенным событием.

При наборе команд в окне модуля Access проверяет правильность написания инструкций VBA. Перед первым использованием процедуры Access автоматически ее компилирует и сохраняет в псевдоисполняемом коде. Если некоторая форма содержит очень много кода внутри своего модуля, то рекомендуется большую часть его выделить в виде функций и поместить в отдельный стандартный модуль. Тогда загрузка самой базы будет происходить медленнее, но форма будет открываться значительно быстрее.

Создание главной (переключательной) формы. Главная форма обычно представляет собой простую форму с размещенными на ней кнопками доступа к другим объектам БД: формам, отчетам и запросам. Если БД небольшая и имеет всего несколько форм, то в качестве главной может быть выбрана уже имеющиеся форма, решающая главную задачу БД. Главная форма является так сказать точкой входа в интерфейс базы данных. Главную форму лучше всего сделать модальной и событие закрытия формы связать с выходом из приложения. Окно БД (область переходов), предоставляющее доступ ко всем объектам БД, необходимо скрыть от пользователя. Такого рода действия являются простейшим, до довольно эффективным средством защиты объектов БД от нежелательной модификации.

Пустую форму можно создать с помощью меню *Конструктор форм* (рис.5.2).

Рис. 5.2. Кнопка **«Конструктор форм»**

Для создания надписей воспользуемся элементом управления *Надпись* Aa в панели инструментов *Элементы управления* (рис. 5.3), а для создания кнопки доступа к другому объекту БД добавим элемент *Кнопка* xxxx.

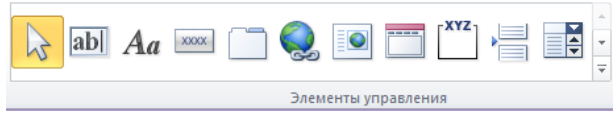
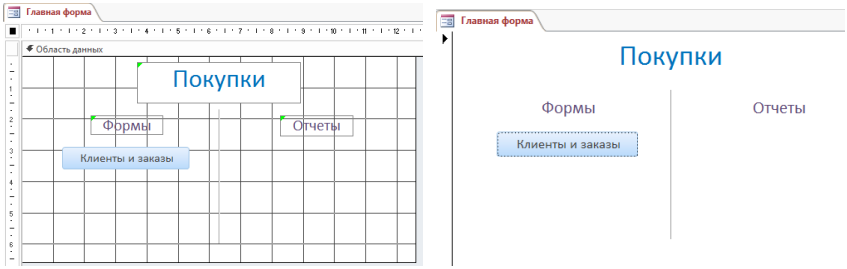


Рис. 5.3. Панель элементов управления разработки формы

Рассмотрим подробнее процесс создания главной (переключательной) формы в режиме конструктора. На рисунке 5.4. приведен примерный вид главной формы.



а) б)

Рис. 5.4. Главная (переключательная) форма: а) режим конструктора; б) режим формы

Мастер создания командных кнопок в последних версиях Access пользуется макросами для автоматизации действий. Предпочтительнее для обработки события нажатия кнопки написать процедуру на VBA, чтобы впоследствии иметь гораздо больше возможностей в программировании и иметь возможность полноценной отладки кода. Для этого о первом же окне мастера (рис. 5.5) нажимаем *Отмена*.

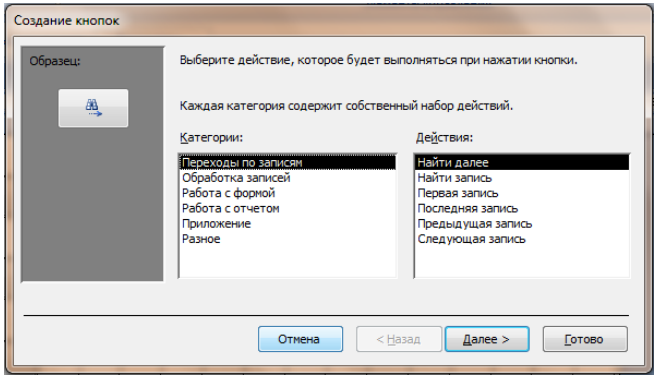
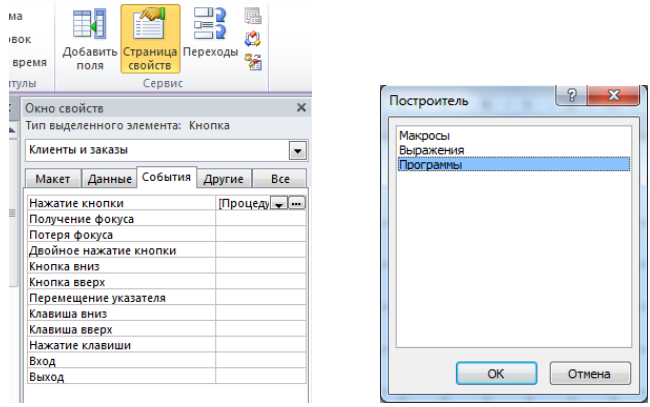


Рис. 5.5. Мастер создания кнопок

После этого заходим в *Окно свойств* элемента *Кнопка* (рис.5.6 а)). Удаляем в пункте меню *События -> Нажатие кнопки* встроенный макрос и нажимаем на  . В появившемся окне выбираем *Программы* и нажимаем ОК (рис. 5.6 б)).



а) б)

Рис. 5.6. а) Окно свойств элемента Кнопка и б) Окно Построителя

При желании можно совсем отключить помощника (*Мастера*). Для этого нужно нажать на кнопку *Использовать мастера*  (рис. 5.7) на панели инструментов.

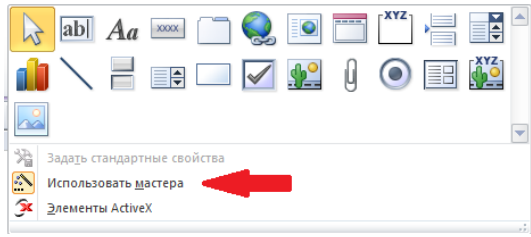


Рис. 5.7. Отключение Мастера

Заготовка обработчика нового события генерируется автоматически в виде:

```
Private Sub Клиенты_Click()
|
End Sub
```

Просмотреть код и скорректировать его по мере необходимости можно в окне редактора Microsoft Visual Basic (рис. 5.8).

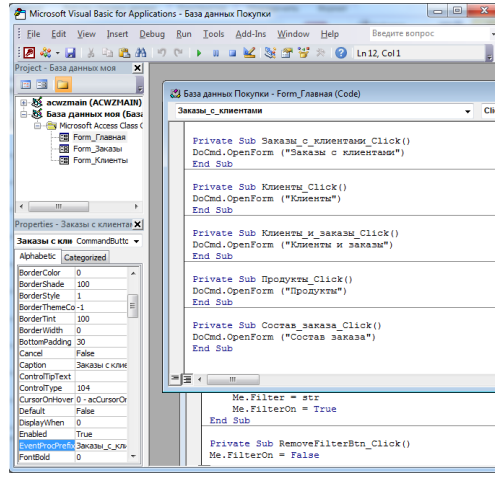


Рис. 5.8. Окно редактора Microsoft Visual Basic

Для того, чтоб кнопка *Клиенты и заказы* открывала форму с таким же названием, достаточно написать следующий код:

```
Private Sub Клиенты_и_заказы_Click()
DoCmd.OpenForm ("Клиенты и заказы")
End Sub
```

В заключение необходимо сохранить набранную программу в окне редактора VBA и перейти в основное окно Access. Теперь, при нажатии на кнопку будет открываться форма «Клиенты и заказы» (примечание, чтобы кнопка заработала, нужно включить активное содержимое).

Для просмотра отчета нужно вызвать метод OpenReport с параметром acViewPreview, в противном случае (со значением параметра по умолчанию) отчет будет сразу отправлен на печать:

```
Private Sub Кнопка0_Click()
DoCmd.OpenReport ("Отчет1"), acViewPreview
End Sub
```

Существует несколько вариантов параметра открытия отчета (на практике чаще всего применяются последние два) (рис.5.9):

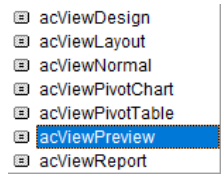


Рис. 5.9. Варианты параметра для открытия Отчета

Для открытия запросов можно использовать метод OpenQuery, например:

```
Private Sub Кнопка0_Click()
DoCmd.OpenQuery ("Запрос1")
End Sub
```

В заключение работы с главной формой полезно обработать событие выгрузки формы и вызвать метод закрытия БД DoCmd.CloseDatabase. Чтобы при закрытии спросить у пользователя, желает ли он действительно закончить сеанс работы с БД, можно написать следующий код в обработчике события:

```
Private Sub Form_Unload(Cancel As Integer)
If MsgBox("Закрыть БД?", vbYesNo) = vbYes Then
    DoCmd.CloseDatabase
End If
End Sub
```

Главная (переключательная) форма с кнопками, открывающими соответствующие элементы базы данных, может быть достаточным интерфейсом для работы с базой данных. Однако, при желании, интерфейс можно существенно расширить и усложнить.

Задание параметров запуска. Для того, чтобы пользователь не имел доступа к элементам БД, к которым разработчик не предоставил явного доступа, а также к режиму Конструктора всех созданных объектов, необходимо задать параметры запуска. Это можно сделать в меню *Файл -*>*Параметры -> Текущая база данных* (рис.5.10). Также в этом меню можно задать название БД, ее логотип, Главную (переключательную) форму, которая будет открываться при открытии приложения, настроить вид интерфейса открытия форм и задать другие параметры БД.

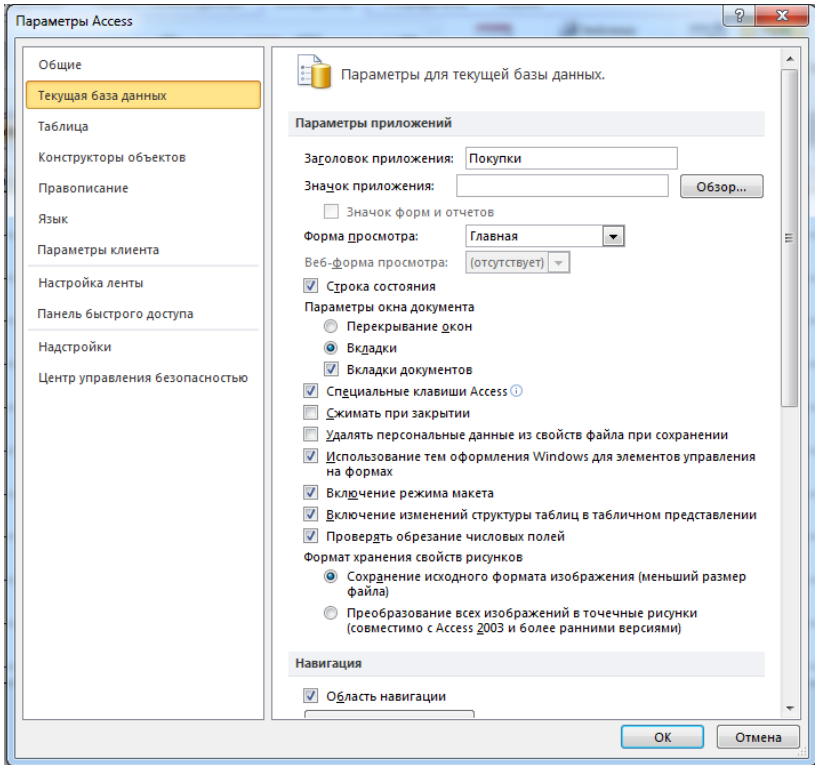


Рис. 5.10. Параметры Access

Итак, вводим *Заголовок приложения* и обязательно задаем *Форму просмотра* (главную форму, которая будет открываться при старте приложения). Чтобы защитить БД от модификаций пользователя, нужно убрать флажки со следующих пунктов:

- ✓ включение режима макета для этой базы данных;
- ✓ включение изменений структуры таблиц …;
- ✓ область переходов (Область навигации);
- ✓ полный набор меню Access;
- ✓ контекстное меню по умолчанию.

Параметры окна документа: *Вкладки* или *Перекрывание окон* можно настраивать по желанию.

Внимание. Защищать БД следует в самом конце разработки.

Реализация пользовательского поиска (с помощью комбинированного списка). Для создания пользовательского поиска с помощью комбинированного списка необходимо в область *Заголовка* формы добавить элемент управления *Поле со списком*  (рис. 5.11).

Рис. 5.11. Пример поля для поиска

Настраиваем это элемент почти так же, как и в случае замены внешнего ключа. Отличием будет только отсутствие привязки к полю таблицы. Для этого нужно зайти в окно «Свойства» и обязательно определить следующие свойства: на вкладке «Данные» – «Источник строк» и «Ограничиться списком», а на вкладке «Макет» – «Число столбцов» и «Ширина столбцов». В запрос свойства «Источник строк» добавляем запрос выборки первичного ключа таблицы и поля, по которому будет организовываться поиск (использование первичного ключа для поиска гарантирует переход к нужной записи). Остальные свойства поля со списком меняются по желанию разработчика. В свойствах *События -> После обновления* нажимаем на  и выбираем *Программы*. В окне VBA пишем следующий код и сохраняем его:

```
Private Sub FindCbx_AfterUpdate()
    Dim rst As Recordset
    Set rst = Me.RecordsetClone
    rst.FindFirst "[Код клиента] = " & FindCbx
    Me.Bookmark = rst.Bookmark
End Sub
```

В приведенном примере: FindCbx – имя поля со списком, Код клиента – имя ключевого поля таблицы.

Внимание. Не привязывайте поле со списком, предназначенное для поиска, к какому-либо полю таблицы (свойство *Данные* должно быть пустым) для исключения редактирования последнего (для текущей записи).

Элемент должен быть свободным от привязки, так как используется для поиска данных, а не редактирования данных.

Реализация пользовательского поиска данных (через поле ввода и кнопки). Создаем пользовательский поиск с помощью поля ввода и двух кнопок, который выглядит следующим образом (рис. 5.12):

Рис. 5.12. Пример поля и кнопок для поиска

Для этого добавляем в область *Заголовка* формы элементы *Поле* и два элемента *Кнопка*, при включенном помощнике отказываемся от его помощи (во всплывающем окне нажимаем *Отмена*) и переходим в окно *Свойств*. Для поля создаем обработчик события (*События -> Изменение*):

```
Private Sub InputFind_Change()
    If InputFind.Text = "" Then
        FindFirstBtn.Enabled = False
        FindNextBtn.Enabled = False
    Else
        FindFirstBtn.Enabled = True
    End If
End Sub
```

Создаем глобальную (уровня формы) переменную rst (в разделе описаний модуля):

```
Option Compare Database
Dim rst As Recordset
```

Для кнопок обрабатываем события нажатие кнопки и делаем их неактивными посредством задания в меню *Данные -> Доступ* значения *нет* (в новых версиях Access *Данные ->Включена ->нет*):

В приведенном примере: FindFirstBtn и FindNextBtn – имена кнопок, InputFind – имя поля, Имя – имя поля таблицы, по которому организуется поиск.

```
Private Sub FindFirstBtn_Click()
    Set rst = Me.RecordsetClone
    rst.FindFirst "[Имя] = '" & Me!InputFind & "'"
    If rst.NoMatch Then
        MsgBox "Запись не найдена"
        FindNextBtn.Enabled = False
    Else
        Me.Bookmark = rst.Bookmark
        FindNextBtn.Enabled = True
    End If
End Sub
```

```
Private Sub FindNextBtn_Click()
    rst.FindNext "[Имя] = '" & InputFind & "'"
    If rst.NoMatch Then
        MsgBox "Запись не найдена"
        DoCmd.GoToControl "InputFind"
        FindNextBtn.Enabled = False
    Else
        Me.Bookmark = rst.Bookmark
        FindNextBtn.Enabled = True
    End If
End Sub
```

Организация пользовательской фильтрации данных. Создаем пользовательскую фильтрацию данных с помощью поля ввода и двух кнопок (рис. 5.13):

Рис. 5.13. Пример пользовательской фильтрации

Для этого добавляем в *Заголовок* формы элементы *Поле* и *Кнопка*, при включенном помощнике отказываемся от его помощи (во всплывающем окне нажимаем *Отмена*) и переходим в окно *Свойств*. Для поля обрабатываем событие *Изменение*:

```
Private Sub InputFilter_Change()
 ApplyFilterBtn.Enabled = Not (InputFilter.Text = "")
End Sub
```

Для кнопок создаем обработчик события *Нажатие кнопки* и делаем кнопки неактивными, как было показано ранее:

```
Private Sub ApplyFilterBtn_Click()
    Dim str As String
    str = "[Пол] Like '*" & Me!InputFilter & "*'"
    Me.Filter = str
    Me.FilterOn = True
End Sub

Private Sub RemoveFilterBtn_Click()
Me.FilterOn = False
End Sub
```

В заключение создаем обработчик события *Текущая запись*:

```
Private Sub Form_Current()
     RemoveFilterBtn.Enabled = Me.FilterOn
End Sub
```

Обработка события *Текущая запись* позволит управлять доступом к кнопке отмены фильтрации не только по нажатию кнопки применить фильтр, а с помощью любых других способов отмены фильтрации, например, с помощью меню.

В приведенном примере: ApplyFilterBtn и RemoveFilterBtn – имена кнопок, InputFilter – имя поля, Пол – имя поля таблицы, по которому организуется фильтрация.

Связывание двух форм по отображению данных. Для связывания двух форм по отображению данных (т.е. с событием открытия некоторой формы связать открытие другой формы, при переходе от записи к записи в первой форме автоматически осуществлять переход к связанной записи во второй форме, при закрытии первой формы закрывать вторую форму) можно написать следующий код в обработчиках событий *Загрузка*, *Выгрузка* и *Текущая запись* главной (основной) формы:

```
Private Sub Form_Load()
    If Not CurrentProject.AllForms("Заказы").IsLoaded Then
        DoCmd.OpenForm "Заказы"
        Forms!Клиенты.SetFocus
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    If CurrentProject.AllForms("Заказы").IsLoaded Then
        If MsgBox("Закрыть связанную форму Заказы?", vbYesNo) = vbYes Then
            DoCmd.Close acForm, "Заказы"
         End If
    End If
End Sub
```

```
Private Sub FilterChildForm()
    If Me.NewRecord Then
        DoCmd.Close acForm, "Заказы"
        DoCmd.OpenForm "Заказы", , , , acFormAdd
        Forms!Клиенты.SetFocus
    Else
        Forms![Заказы].Filter = "[Код клиента] = " & Me.[Код клиента]
        Forms![Заказы].FilterOn = True
    End If
End Sub

Private Sub Form_Current()
     If CurrentProject.AllForms("Заказы").IsLoaded Then
        FilterChildForm
    End If
End Sub
```

В данном коде «Заказы» - зависимая форма. Для расположения двух форм рядом заходим в меню *Файл ->Параметры -> Текущая база данных* и в *Параметрах окна документа* выбираем *Перекрывание окон (*рис. *5.14).*

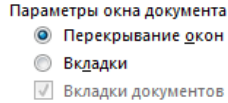


Рис. 5.14. Параметры для окон документа

Эту настройку тоже удобнее делать в конце разработки, т.к. работать с формами и другими элементами БД проще, когда они расположены как *Вкладки*, а не как *Окна*.

Задания

1. Создать главную (переключательную) форму;
2. На основе созданных форм разработать интерфейс приложения. Работа с базой данных должна производится только через созданный интерфейс (окно базы данных должно быть скрыто от пользователей);
3. Задать параметры запуска;
4. Автоматизировать работу приложения с помощью макросов и программирования на VBA. Требования: научиться динамически менять свойства объектов, научиться выполнять операции открытия и закрытия объектов, поиска и фильтрации;
5. Имплементировать пользовательский поиск (с помощью комбинированного списка);
6. Имплементировать пользовательскую фильтрацию данных (задать поле ввода и кнопку(и) включения/выключения фильтрации);

7. Имплементировать пользовательский поиск данных (задать поле ввода и кнопку(и) поиска);
8. Связать две формы по отображению данных, т.е. с событием открытия некоторой формы связать открытие другой формы, при переходе от записи к записи в первой форме автоматически осуществлять переход к связанной записи во второй форме, при закрытии первой формы закрывать вторую форму.

Форма отчета. Формой отчета являются база данных выбранной тематики и проект MS Access, реализующий задания, представленные в разделе «Задания»

ПРИЛОЖЕНИЕ

VISUAL BASIC FOR APPLICATIONS

Visual Basic for Applications (VBA) является интерпретируемым языком, т.е. его инструкции интерпретируются каждый раз при выполнении программы. Этот язык является общим для всех приложений MS Office; на нем можно писать с равным успехом программы как для Access, так и для Word либо Excel.

Объявление переменных. Переменные в Access можно объявлять явно и неявно. Имена переменных должны начинаться с буквы и не могут содержать пробелов или других знаков пунктуации, кроме знака подчеркивания. Также нельзя использовать зарезервированные слова. Число символов в имени не должно превышать 225. При неявном задании переменной ей автоматически присваивается тип Variant. Это универсальный тип данных. Переменная такого типа может принимать любые значения (этот тип по умолчанию имеют все данные, непосредственно полученные из таблиц и запросов) и дополнительно пустое значение *Null*. Проверить, содержит ли переменная какое-либо значение, можно с помощью функции *IsNull*().

Переменные определенных типов можно задавать либо с помощью суффиксов (определенного символа, записываемого сразу после имени переменной), либо с помощью явного объявления типа. Имеются следующие типы данных:

Byte	Currency @
Integer %	String $
Long &	Date
Single !	Boolean
Double #	Variant

Для явного описания переменной используется оператор Dim.

```
Dim <имя переменной> As <тип переменной>.
```

Например,
```
Dim i As Integer, j As Integer
Dim x As Double
```

Строки могут быть постоянной и переменной длины. Оператор `Dim <имя переменной> As String` объявляет строку переменной длины, а оператор `Dim <имя переменной> As String*<количество символов>` - фиксированной длины. Например:

```
Dim str1 As String              Dim str2 As String*20.
```

Можно ввести требование, чтобы все переменные были описаны явно. Для этого в разделе описаний этого модуля необходимо задать декларацию Option Explicit. Тогда Access будет автоматически обнаруживать ошибки в написании имен переменных и контролировать совпадение типов переменной и назначаемых ей значений.

Переменные имеют определенную область видимости. Существует 4 уровня видимости:

1. Локальный уровень или уровень процедуры. Переменные существуют только внутри процедуры, где они описаны и используются;
2. Уровень формы (отчета). Переменные описываются в разделе описаний модуля формы (отчета) и являются доступными только для процедур этой формы (отчета) в то время, когда форма (отчет) открыта;
3. Уровень модуля. Переменные описываются в разделе описаний стандартного модуля и доступны для всех процедур модуля при открытой БД;
4. Глобальный уровень. Переменная записывается в разделе описаний модуля при помощи инструкций Public или Global.

Если пытаться использовать неявно заданную переменную в то время, когда она недоступна, то Access взамен ее создаст новую переменную с тем же именем, что может привести к трудно обнаруживаемым логическим ошибкам. Это может служить дополнительным поводом для явного задания всех переменных.

Все переменные имеют определенное время жизни. Для локальных переменных время жизни определяется временем использования кода. Каждый раз при новом вызове процедуры переменной приписывается либо *Null*, либо 0, либо пустая строка в зависимости от типа. Для создания переменных со временем жизни, равным времени жизни приложения, используется инструкция Static. Тем не менее, статическая переменная не может быть использована в других процедурах. Изменяется лишь время ее жизни, а не область видимости. Если произойдет повторный вызов той же самой процедуры, в которой была описана статическая переменная, то эта переменная сохранит свое прежнее значение, которое она имела в момент завершения работы этой процедуры при предыдущем вызове. Все переменные в определенной процедуре будут статическими, если перед самой процедурой поставить инструкцию Static.

С помощью оператора Dim также можно определять многомерные массивы. Размерность и число элементов массива определяется внутри скобок после имени переменной. Например:

```
Dim myArray(20) As String*10
```

```
Dim myArray(5, 5) As Double или Dim myArray(3, 3, 3) As Double
```

Нумерация элементов массива начинается с 0, поэтому для первого примера будет зарезервирована память для массива, состоящего из 21 элемента, для второго – 6 на 6. При объявлении массива можно определять не только верхнюю границу массива, но и нижнюю. Например, Dim my Array(1 to 20) As String.

В VBA допускается использование динамических массивов. Для этого при описании переменной число элементов массива опускается, а затем применяется оператор ReDim для объявления реального размера массива. При этом все элементы массива обнуляются. Для сохранения значений элементов массива применяется инструкция Preserve.

```
Dim dynArray( ) As Double
ReDim Preserve dynArray (5, 2, 4)
```

Во время выполнения программы нижнюю и верхнюю границы массива можно определить с помощью функций LBound() и UBound() соответственно. Удалить массив из памяти можно с помощью оператора Erase.

Константы в VBA описываются с помощью ключевого слова Const. Вместе с объявлением переменной необходимо также произвести ее инициализацию: `Const pi As Double = 3.1415`

Кроме переменных вышеприведенных типов в VBA можно задавать объектные переменные, применяющиеся для хранения ссылок на объекты Access. Для каждой коллекции основных объектов в Access имеется соответствующий ей объектный тип, а также общий тип Object, принимающих ссылки на любые объекты. Например:

`Dim myObject As Object` – переменная любого объектного типа,

`Dim myControl As Control` – переменная типа элемента управления,

`Dim myForm As Form` – переменная типа формы.

Присваивать конкретные значения объектным переменным можно с помощью оператора Set.

```
Set myForm = Forms![Моя Форма].
```

Пользователь может также создавать свой тип данных на основе существующих. Пользовательский тип данных вводится между ключевыми словами `Type…End Type`.

```
Type tPhone
number As String*15
   type As String*10
End Type
```

Обращение к полю пользовательского типа производится через операцию “.”.

```
Dim tel As tPhone
```

```
tel.number = 3334455
tel.type = "Мобильный".
```

Заполнить объектные переменные, содержащие много свойств, а также переменные пользовательского типа данных можно с помощью оператора `With … End With.`

```
Dim myForm As Form
Set myForm = Forms![Имя формы]
With myForm
    .Top=1000
    .Left=1000
    .Width=5000
    . Height=4000
End With.
```

Создание комментариев. Любой текст, следующий за символом ‘, воспринимается как комментарий.

Процедуры и функции. Основными компонентами программ на VBA являются процедуры и функции. Они представляют собой фрагменты программного кода, заключенные между операторами Sub и End Sub или между Function и End Function. В общем случае процедуры и функции записываются следующим образом:

```
Sub <имя процедуры< [(<аргументы>)]
    <операторы>
End Sub
Function <имя функции> [(<аргументы>)] As <тип возвращаемого
значения>
     <операторы>
    <имя функции> = <возвращаемое значение>
End Function
```

Список аргументов разделяется запятыми. Функция отличается от процедуры тем, что ее имя выступает также в качестве переменной и используется для возвращения значения в точку вызова функции. Для вызова процедуры из другой процедуры или функции используется оператор Call. Вначале идет имя процедуры, а затем в скобках список фактических значений ее аргументов. Процедуры можно также вызывать просто по их имени без использования оператор Call. В этом случае список аргументов не заключается в скобки. Функции вызываются так же, как и процедуры, но гораздо чаще они вызываются по их имени с заключенным в скобки списком фактических значений аргументов в правой части оператора присваивания. Например:

```
Call mySub("Ландера", 4, i+1)
mySub "Ландера", 4, i+1
total_price = myFunc([Цена]*[Количество], [Доставка])
```

Внимание. При копировании текста программ из пособия или других источников, созданных не в редакторе VBA, возможно некорректное копирование одиночных и двойных кавычек. Если скопированный текст программы не работает, правильность написания кавычек в текстовых константах следует проверять в первую очередь.

Допускается два различных способа передачи переменных процедуре или функции: по ссылке или по значению. По умолчанию переменные передаются по ссылке (объявление ByRef). При изменении значения переменной внутри процедуры это изменение остается и при выходе из этой процедуры. Переменные можно передавать и по значению. Тогда изменение значения такой переменной не будет воздействовать на значение переменной, переданной в процедуру или функцию. Для этого используют объявление ByVal. В общем виде объявление аргументов производится следующим образом `ByVal/ByRef <имя аргумента> [( )] [As <тип>]`. Например:

```
Sub mySyb(srteet As String, ByVal building As Integer, ByRef
apt As Byte)
    Forms![Клиенты]![Адресс] = srteet & " " & building & ", "
& apt
End Sub
Function myFunc(full_price As Currency, shipment As Single) As
Currency
   myFunc = full_price * 1.2 + shipment
End Function
```

Для принудительного выхода из процедуры и функции используются соответственно Exit Sub и Exit Function.

Управление выполнением программы. Управление выполнением программы достигается за счет оператора безусловного перехода, операторов ветвления (условных операторов) и циклов. Условные операторы выполняют группу команд в зависимости от условия. К ним относится оператор If и Select Case. В простейшем виде оператор If можно записать в одну строку.

```
If <условие> Then <оператор>
Более сложный вариант использования оператора If приведен ни-
же:
If <условие> Then
    <операторы>
Else
     <операторы>
End If
```

Для задания выбора действий на основе проверки целой группы условий используется расширенный вариант записи оператора If.

```
If <условие> Then
    <операторы>
Else If <условие 2> Then
     <операторы>
Else
     <операторы>
End If
```

Если выбор действий зависит от различных значений одного и того же выражения, то вместо вложенного оператора If предпочтительнее использовать оператор Select Case … End Select. Этот оператор определяет, является ли выражение истинным, а также оценивает, заключены ли в определенных пределах значения этого выражения.

```
Select Case <имя переменной>
    Case <выражение 1> [ , <выражение 2>, …]
(Операторы, выполняемые, если значения переменной  удовлетворяют или выражение 1, или выражение 2, или …)
    [Case <выражение 3> To <выражение 4>]
(Операторы, выполняемые, если значение переменной находится в диапазоне, определяемом выражениями 3 и 4)
    [Case Is <выражение отношения>]
(Операторы, выполняемые, если значение переменной удовлетворяет выражению отношения.)
    [Case Else <операторы, выполняемые, если не было выполнено ни одно из вышеперечисленных условий>]
End Select
```

Любой строке в программе на VBA можно присвоить метку Метка:. Перейти на эту строку программы можно с помощью оператора безусловного перехода GoTo <метка>. Однако операторы GoTo нарушают стиль структурного программирования, и они применяются только для обработки ошибок в приложении.

Операторы цикла в VBA различаются на две основные группы: циклы с перечислением (For … Next) и циклы с условием (Do … Loop).

Оператор For…Next повторяет тело цикла заданное число раз. Он имеет следующий формат:

```
For <счетчик> = <начальное значение> To <конечное значение> [Step <приращение>]
    <операторы>
Next <счетчик>
```

Для выхода из цикла используется оператор Exit For.

Операторы циклов Do While … Loop, While … Wend являются синонимами. While … Wend оставлен для совместимости со старыми версиями. Эти операторы повторяют тело цикла, пока условие принимает значение True. Формат оператора Do следующий:

```
Do While <условие [=True]>
```

```
     <операторы>
Loop
```

Оператор Do Until … Loop выполняет тело цикла до тех пор, пока не будет выполнено условие. Его формат:

```
Do Until <условие [< > True]>
    <операторы>
Loop
```

Чтобы обеспечить выполнение операторов тела цикла по крайней мере один раз, можно использовать следующие варианты этих циклов.

```
Do
    <операторы>
Loop While <условие [=True]>
или
Do
     <операторы>
Loop Until <условие [=False]>
```

Для выхода из этих циклов используется оператор Exit Do.

Для построения итератора по всем элементам массива или коллекции используется оператор For Each

```
For Each <элемент> In <коллекция>
     <операторы>
Next <элемент>
```

где <элемент> – это переменная, используемая для ссылки на элементы семейства объектов. Следующий пример показывает, как заполнить выпадающий список элемента управления ПолеСоСписком1 названиями всех элементов управления, расположенных в форме:

```
Dim ctrl As Control
For Each ctrl In Form
   ПолеСоСписком1.AddItem ctrl.Name
Next ctrl
```

Обработка ошибок на этапе выполнения. Как тщательно не проверялся бы код, на этапе его выполнения неизбежно возникают ошибки. Попытка деления на ноль – это типичный пример ошибки времени выполнения. В Access обработка ошибок выполняется с помощью оператора On Error. Имеется три вида оператора On Error:

1. On Error GoTo Метка – осуществляет переход на строку с меткой (Метка) при возникновении произвольной ошибки на этапе выполнения. Обычная часть кода, обрабатывающая ошибки, помещается в корпус процедуры. После обработки ошибки можно либо вызвать повторение той части кода, где произошла ошибка, либо проигнорировать ошибку и продолжить выполнение последующих инструкций.

Для возвращения на строку с ошибкой используется ключевое слово Resume.

2. On Error Resume Next – игнорирует ошибку и продолжает выполнение последующих инструкций.
3. On Error GoTo 0 отключает обработку ошибок.

После обработки первой ошибки оператор On Error GoTo Метка продолжает выполняться для всех последующих ошибок, пока не закончится выполнение данной процедуры либо обработка ошибок не будет явно отключена оператором On Error GoTo 0. Если нет обработки какой-то ошибки или обработка выключена, то при возникновении необработанной ошибки приложение сразу же выдаст сообщение об ошибке и прекратит работу.

Если процедура обработки некоторого события создается с помощью мастера, то он автоматически дополняет процедуру кодом обработки ошибок.

```
Private Sub MyID_DblClick(Cancel As Integer)
On Error GoTo Err_MyID_DblClick
     Текст процедуры
Exit_MyID_DblClick:
Exit Sub
Err_MyID_DblClick:
MsgBox Err.Description
Resume Exit_MyID_DblClick
End Sub
```

Объект Err содержит информацию об ошибках выполнения и обычно используется вместе с оператором Select Case, чтобы определить, какое действие предпринять в зависимости от кода ошибки.

```
Select Case Err
    Case 58 To 76
        Call  FileError-  процедура  обработки  ошибок  работы  с
файлами
    Case 281 To 297
        Call DDEError  - процедура для обработки ошибок  DDE
    Case 340 To 344
        Call ArrayError - процедура ошибок массивов
End Select
```

Err=0 – отключение обработки необработанных ошибок.

Работа с объектами и коллекциями. В VBA можно работать с объектами и коллекциями Access, библиотекой доступа к данным DAO (Data Access Objects), библиотекой прямого доступа к данным баз данных ODBC (ODBC Direct), библиотекой доступа к данным ADO (ActiveX Data Objects) и другими библиотеками. В Access, DAO и ADO все объекты рас-

положены внутри коллекций и доступны в VBA. Каждый объект имеет свойства и методы. Все библиотеки организованы в виде иерархии объектов. Объекты имеют коллекции (семейства) подчиненных объектов и т.д. В Access имеется 8 базовых объектов:

1. Application – активное приложение;
2. Control – элемент управления;
3. DoCmd – объект вызова макрокоманд в VBA коде;
4. Form – открытая форма;
5. Module – объект, ссылающийся на стандартные модули;
6. Reference – объект, содержащий ссылки на объекты;
7. Report – открытый отчет;
8. Screen – ссылка на экран;

Их иерархия представлена на рис. П1.1. Названия семейств формируются путем возведения в множественное число названия соответствующего объекта. В свою очередь большинство объектов имеют присоединенные коллекции свойств (Properties), а формы и отчеты – коллекции разделов и т.д. Так как все объекты в Access хранятся внутри иерархически связанных коллекций, то доступ к объекту на нижней ступени иерархии можно получить, указав все имена коллекций, разделенных точкой, начиная от корневого объекта. Например,

```
Application.Forms("Заказы").Controls(0).Properties(0).
```

Большинство коллекций, к примеру, коллекции форм и отчетов, являются глобальными. Тогда к объекту этой коллекции можно обращаться напрямую: `Forms("Заказы")` или `Forms!Заказы`.

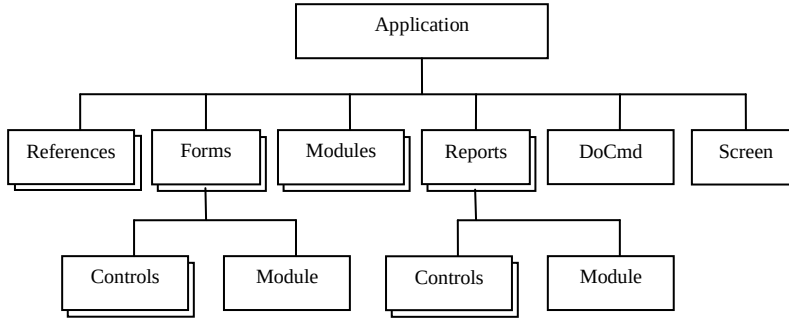


Рис. П1.1. Иерархия объектов Access

Поскольку библиотека DAO всегда поставляется с Access, рассмотрим ее структуру и основные методы более подробно. Объекты библиотеки DAO:

Database – открытая база данных;
DBEngine – ссылка на Microsoft Jet (ядро БД);
Error – объект ошибок;
Field – поле в таблицах, запросах, динамических наборах и т.д.;
Index – индекс;
Parameter – параметр запроса;
QueryDef – сохранённый запрос;
Recordset – динамический набор данных;
Relation – связь между таблицами;
TableDef – сохраненная таблица;
Workspace – активная сессия.

Их иерархия представлена на рис. П1.2. В программах на VBA имеется набор свойств объектов, которые возвращают ссылки на подчиненные объекты:

Me – ссылка на активную форму или отчет (доступна в присоединенном модуле);
ActiveControl – ссылка на активный элемент управления;
ActiveForm – ссылка на активную форму (доступна в объекте Screen);
ActiveReport – ссылка на активный отчет (доступна в объекте Screen);
Application – ссылка на открытое приложение;
Parent – ссылка на родительский объект, т.е. на коллекцию;
DBEngine – возвращает ссылку на Application.

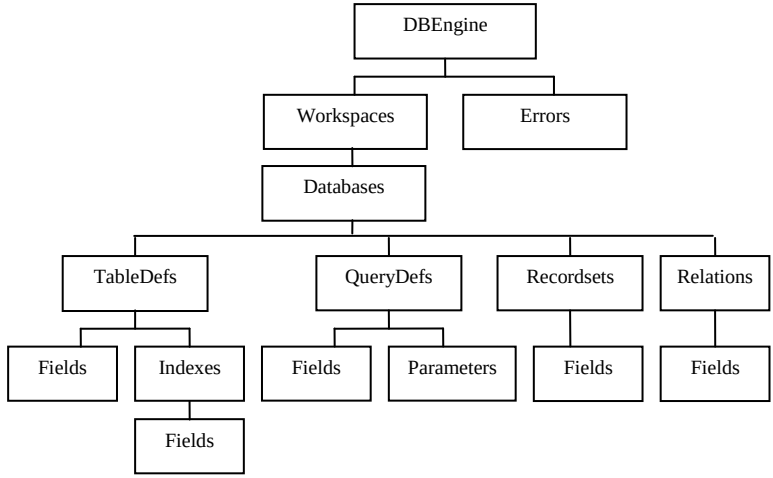


Рис. П1.2. Иерархия объектов DAO

Работа с записями и полями. Применить фильтрацию, изменить значение какого-либо поля в базовом наборе данных либо даже сменить базовый набор можно непосредственно с помощью свойств самой формы и элементов управления. Например:

```
Me![Полная цена] = Me![Цена] * Me![Количество]
Me.Filter = "[Адрес] Like '*" & Me![ПолеУсловияПоиска] & "*'"
Me.FilterOn = True
```

Если есть желание читать и менять данные в некотором поле базового набора, но не видеть его на экране, то достаточно поместить элемент управления *Поле* на форму и задать свойство Visible в False. Но прямой доступ к базовому набору в Access невозможен. Можно лишь создать динамическую копию базового набора и синхронизировать все производимые действия с ней с базовым набором самой формы. Для прямого доступа к записям и полям используется объект Recordset. Имеются четыре типа Recordset объектов – *table*, *dynaset*, *snapshot* и *forwarrd-only*:

1. *Table*: Может быть создан только на основе существующей или присоединенной таблицы. Предоставляет доступ ко всем методам и свойствам таблицы, а также к индексам, что дает намного более быстрый метод поиска (метод Seek);
2. *Dynaset* - может быть создан на основе таблицы или запроса. Позволяет обновлять данные в многотабличных запросах и в запросах к внешним БД. Обновление *Dynaset* объекта приводит к автоматическому обновлению всех участвующих в нем таблиц;
3. *Snapshot* - создает статическую копию и существует только в то время, когда он создан. Последующие изменения таблиц на него не воздействуют;
4. *Forward-only* создает статическую копию с просмотром только в прямом порядке.

Для создания объекта типа Recordset используется метод OpenRecordset:

```
Set variable = database.OpenRecodset (source [type, options, lockedits])
```

, или

```
Set variable = object.OpenRecodset ( [type, options, lockedits]),
```

где database – это переменная типа Database; object – переменная типа TableDef или QueryDef; source – ссылка на объект типа TableDef или QueryDef; type – тип динамического набора (может принимать следующие значения: dbOpenTable, dbOpenDynaset, dbOpenSnspshot dbOpenForwardOnly); options может принимать следующие значения: dbAppendOnly, dbReadOnly, dbForwardOnly, …; lockedits – аргумент, опре-

деляющий разрешение конфликтов в многопользовательских БД (может принимать следующие значения: DbReadOnly, dbPessimistic, dbOptimistic). Например:

```
Dim db As Database
Dim rst As Recordset
Set db = CurrentDb()
Set rst = db.OpenRecordset("Клиенты", dbOpenDynaset)
```

Открыть Recordset можно и основываясь на переменной типа формы (допустим только для форм, основанных на таблице или запросе) с помощью метода RecordsetClone. Метод RecordsetClone создает динамический набор, основываясь на свойстве *Источник данных* для формы

```
Dim rstOrders As Recordset
Set rstOrders = Forms![Заказы].RecordsetClone
или просто Me.RecordsetClone
```

Recordset можно создать, также основываясь на строке SQL

```
Set rst = db.OpenRecordset("SELECT * FROM Товары WHERE Цена >
1000", dbOpenDynaset, dbReadOnly)
```

После завершения работы с динамическим набором его необходимо закрыть. Существуют два общих способа закрытия объектов в VBA. Первый заключается в вызове метода Close, второй – в присвоении соответствующей объектной переменной значения Nothing. Например, rst.Close или Set rst = Nothing.

Recordset имеет текущее приложение – current position. Для синхронизации текущего положения динамического набора с текущей записью формы можно использовать свойство Bookmark (необходимо всегда помнить, что при обращении к динамическому набору данные берутся именно из текущей записи). Для перемещения по динамическому набору имеется ряд методов: MoveFirst, MoseLast, MoveNext, MovePrevions, Move[n]. Например:

```
Dim rst As Recordset
    Set rst = Me.RecordsetClone
    rst.MoveNext
    Me.Bookmark = rst.Bookmark
```

Для определения начала и конца набора можно использовать свойства BOF (before of file – начало файла) и EOF (end of file – конец файла). Если в наборе нет записей, то BOF и EOF равны True. Если в наборе есть записи, то при открытии курсор обычно устанавливается на первой записи и BOF и EOF = False. Если курсор находится перед первой записью, то BOF = True, и если курсор находится после последней записи, то EOF = True.

Число записей в динамическим наборе можно получить с помощью свойства RecordCount. Это свойство возвращает реальное число записей только для динамического набора, основанного на таблице, т.е. имеющего тип dbOpenTable. После открытия динамических наборов любых других типов число записей в нем будет неизвестно до тех пор, пока курсор не будет перемещен на последнюю запись. Для непосредственной проверки на наличие в наборе записей лучше проверить свойство EOF. Если EOF будет равно True, то RecordCount также будет равен 0.

```
Public Function RecCount() As Long
Dim rstCount As Recordset
Dim dbs As Database
Set dbs = CurrentDB()
Set rstCont = dbs.OpenRecordset("Заказы")
If rstCount.Eof Then
    RecCount = 0
Else
    rstCount.MoveLast
    RecCount = rstCount.RecordCount
End If
rstCount.Close
Set dbs = Nothing
End Function
```

Поиск определенной записи. Для наборов, основанных на таблице, можно использовать метод Seek, который ищет запись, основываясь на сравнении данных некоторого индекса с заданными значениями (самый быстрый метод). Для примера запишем программу, которая будет проверять дублирование значений ключевого поля таблицы *Клиенты* перед сохранением записи. Для проверки результатов поиска в объекте Recordset имеется свойство NoMatch. Если NoMatch равно False, то запись с заданными условиями поиска отсутствует в наборе.

```
Dim rst As Recordset
Set rst = CurrentDb.OpenRecordset("Клиенты", dbOpenTable)
rst.Index = "PrimaryKey"
rst.Seek "=", Me![КодКлиента]
If Not rst.NoMatch Then
    MsgBox  "Клиент с таким табельным номером уже есть в БД"
    DoCmd.GoToControl "КодКлиента"
End If
rst.Close
```

Метод Seek всегда начинает поиск с начала набора, поэтому поиск по одному и тому же условию будет всегда приводить к одной и той же записи. Для поиска по динамическим наборам остальных типов можно применять также методы FindFirst, FindLast, FindNext и FindPrevions. Например, подсчитаем количество заказов определенного клиента:

```
intCount = 0
rstOrders.FindFirst "КодКлиента=" & Me![КодКлиента]
Do While rstOrders.NoMatch
    rstOrders.FindNext "КодКлиента=" & Me![КодКлиента]
    intCount = intCount + 1
Loop
```

Для поиска записей можно также применять сортировку и фильтрацию. Для сортировки записей лучше всего открыть новый динамический набор, основанный на запросе SQL с оператором ORDER BY. Для фильтрации можно задать свойство Filter объекта Recordset и затем обновить набор с помощью метода OpenRecordset. Ниже приведен текст программы, осуществляющей фильтрацию произвольного набора:

```
Function FilterField(rstTemp As Recordset, strField As String,
strFilter As String) As Recordset
    rstTemp.Filter = strField & " = '" & strFilter & "'"
    Set FilterField = rstTemp.OpenRecordset
End Function
```

Обновление динамических наборов. Обновление данных возможно только при работе с динамическими наборами типов dbOpenTable и dbOpenDynaset. Для редактирования некоторой записи необходимо вначале установить курсор на ней, перевести динамический набор в режим редактирования (метод Edit), а затем сохранить изменения с помощью метода Update.

```
rst.Edit
rst![Товар] = "Pentium 100"
rst![Цена] = 100
rst.Update
```

Метод Cancel отменяет внесенные изменения. Для удаления текущей записи существует метод Delete. Этот метод удаляет запись без выдачи каких-либо предупреждений и не меняет положения курсора. Для перехода на следующую запись необходимо вызвать метод MoveNext. Для добавления новой записи служит метод AddNew. После заполнения новой записи значениями ее необходимо сохранить с помощью метода Update, потому как если после добавления новой записи перейти к другой записи без сохранения или просто закрыть динамический набор, то добавляемая запись будет потеряна. Следующий пример изменяет значения полей *Цена* и *Количество* таблицы *Заказы*, отфильтрованной для определенного клиента.

```
Dim rst As Recordset
Dim dbs As Database
Set dbs = CurrentDb()
Set rst = dbs.OpenRecordset("Заказы", dbOpenDynaset)
rst.Filter = "[КодКлиента]=" & Me![Код клиента]
```

```
Set rst = rst.OpenRecordset
If Not rst.EOF Then
  With rst
    Do Until .EOF
      .Edit
      ![Цена] = rst![Цена] * 1.2
      ![Количество] = ![Количество] * 2
      .Update
      .MoveNext
    Loop
    .Close
  End With
End If
dbs.Close
```

Работа с полями. Получить доступ к значениям и свойствам полей некоторой таблицы можно, открыв соответствующий динамический набор. Коллекция полей Fields является коллекцией по умолчанию для объекта типа Recordset. Тогда получить доступ к некоторому полю можно по его имени или по индексу в коллекции, например: rst.Fields(Field1”), rst.Fields(4) или rst![Field1]. После получения ссылки на объект типа Field можно читать и изменять его свойства, например:

```
Dim fld As Field
Set fld = rst!Количество
fld.DefaultValue = 1
fld.ValidationRule = "BETWEEN 1 AND 1000"
fld.Value = 100
```

ЛИТЕРАТУРА

1. Скакун, В.В. Системы управления базами данных. Пособие для студентов факультета радиофизики и электроники / В.В. Скакун. – Минск: Издательский центр БГУ, 2008. 114 с.
2. Змитрович, А.И. Базы данных и знаний: учебное пособие / А.И. Змитрович, В.В. Апанасович, В.В. Скакун. – Минск: Изд. центр БГУ, 2007. – 364 с.
3. Хомоненко, А. Д. Базы данных. Учебное пособие / А. Д. Хомоненко СПб: Корона, 2002. – 672 с.
4. Грофф, Д. SQL. Полное руководство / Д. Грофф, П. Вайнберг, Э. Оппель. Вильямс, 2014. – 960 с.
5. Бондарь, А. Г. Microsoft SQL Server 2012 / Бондарь, А. Г. СПб.: БХВ-Петербург, 2013. – 608 с
6. Михеева, В. Microsoft Access 2003 / В. Михеева и И. Харитонова. – СПб: БХВ, 2004. – 1072 с.
7. Дейт, Дж. Введение в базы данных / Дж. Дейт; пер. с. англ. 8 издание. СПб: Питер, 2005. – 1328 с.

СОДЕРЖАНИЕ

Учебное издание

Скакун Виктор Васильевич
Эйсмонт Ирина Сергеевна

МОДЕЛИ ДАННЫХ И СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Методические указания к лабораторным работам

Для студентов специальностей 1-98 01 01 «Компьютерная безопасность», 1-31 03 07 «Прикладная информатика»

В авторской редакции

Ответственный за выпуск *И. С. Эйсмонт*

Подписано в печать 14.09.2020. Формат 60×84/16.
Бумага офсетная. Усл. печ. л. 5,81. Уч.-изд. л. 4,63.
Тираж 50 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/270 от 03.04.2014.
Пр. Независимости, 4, 220030, Минск.

Отпечатано с оригинал-макета заказчика
на копировально-множительной технике
факультета радиофизики и компьютерных технологий
Белорусского государственного университета.
Ул. Курчатова, 5, 220064, Минск.