МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Институт прикладной математики и компьютерных наук
Кафедра вычислительной механики и математики

Утверждено на заседании кафедры
«Вычислительная механика и математика»
« 14 » января 2021 г., протокол № 5
с учетом изменений и дополнений,
утвержденных на заседании кафедры
«Вычислительная механика и математика»
«17» июня 2021г., протокол №10,
вступающих в силу с 1 сентября 2021 года

Заведующий кафедрой

_____В.В. Глаголев

СБОРНИК МЕТОДИЧЕСКИХ УКАЗАНИЙ
К ПРАКТИЧЕСКИМ РАБОТАМ
учебной дисциплины (модуля)

"Информационные технологии в социально-гуманитарной сфере -2"

основной профессиональной образовательной программы высшего образования – программы бакалавриата

по направлению подготовки
42.03.02 – Журналистика

с направленностью (профилем)
Региональные периодические издания и мультимедийная журналистика

Форма обучения: очная

Идентификационный номер образовательной программы: 420302-01-21

Тула 2021

Разработчик методических указаний:

Зотова С.В, ст. преподаватель каф ВММ 

(ФИО, должность, ученая степень, ученое звание)

Часть 1. Базы данных.

Содержание

Практическая работа № 1

Создание таблиц в базе данных Base Open Office.

Установление связей между таблицами

1.1. Цель работы

Приобретение навыков по созданию таблиц в СУБД

Base Open Office

и установление между ними связей различных типов

1.2. Теоретические положения

СУБД – система управления базами данных – комплекс средств для хранения, сортировки и фильтрации информации.

В Open Office базы данных имеют название Open Office.org.Base.

Для создания файла Open Office.org.Base необходимо из меню *Пуск* выбрать команду *Программы*, затем по флажкам перейти к Open Office.org.Base (рис. 1).

Далее в появившемся окне необходимо выбрать *Создать новую базу данных* и нажать кнопку *Готово*. После этого пользователю будет предложено ввести имя файла в соответствующую строку для имени. После ввода имени файла необходимонажать кнопку *Сохранить* (рис. 2).

Рисунок 2. Сохранение файла Open Office.org.Base

В случае, если файл был создан ранее и его требуется открыть, необходимо выбрать *Открыть существующий файл*, после чего указать имя файла. После этого откроется основное окно для проектирования базы данных. Для создания таблиц в открывшемся окне необходимо выбрать объект *Таблицы*(рис. 3).

Существуют 2 способа создания таблиц: режим *дизайна* и режим *мастера*. Режим *дизайна* предполагает самостоятельное создание таблиц пользователем. Режим *мастера* предполагает выбор пользователем для создания таблиц уже имеющихся шаблонов.

В поле *Задачи* необходимо выбрать действие: *Создать таблицу в режиме дизайна* или *Использовать мастер для создания таблиц* (рис. 4).

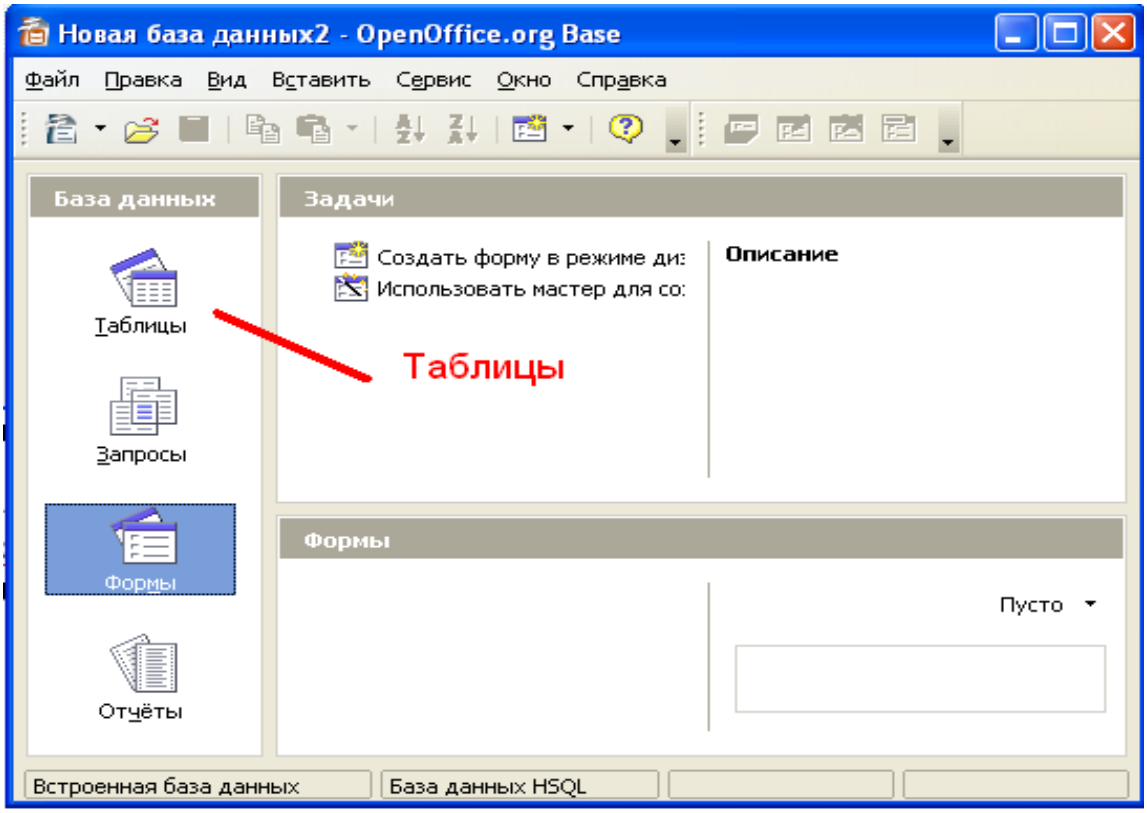


Рисунок 3. Окно проектирования базы данныхOpen Office.org.Base

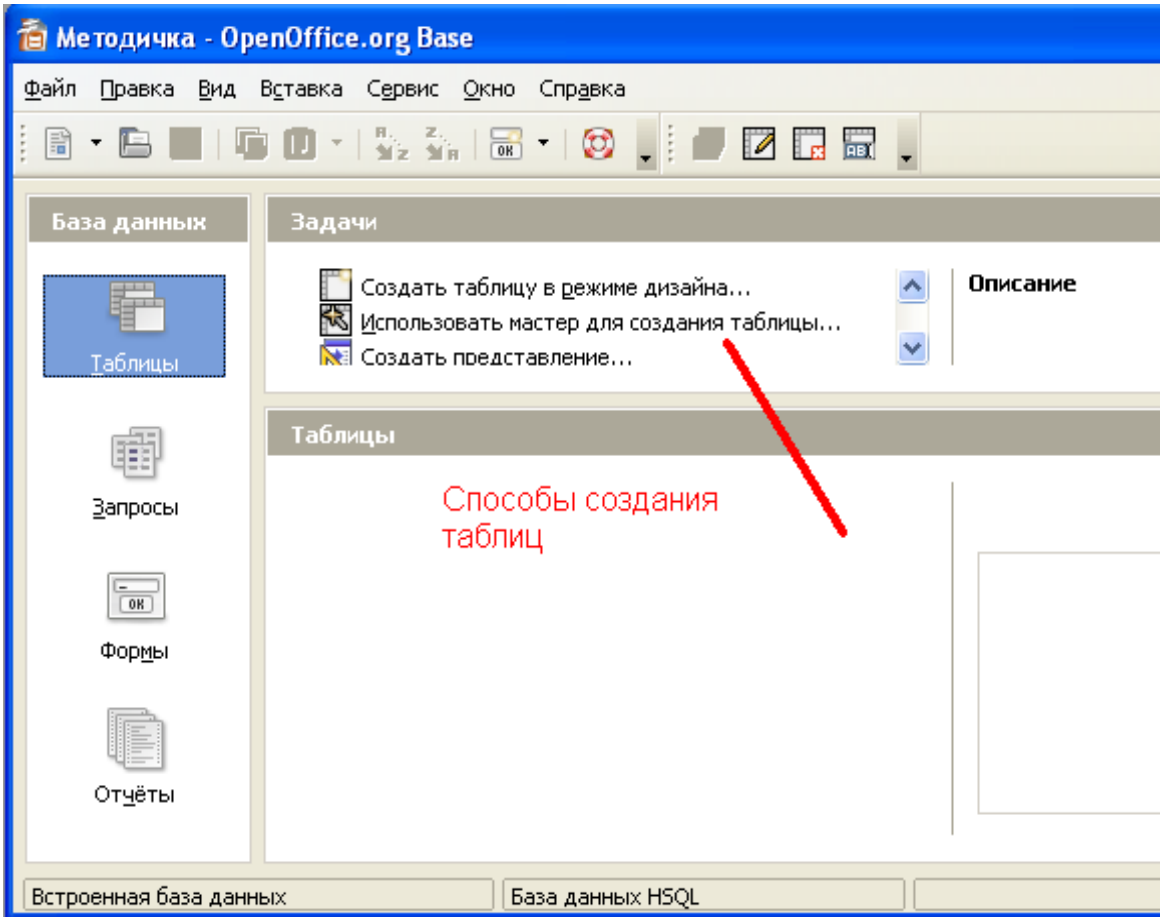


Рисунок 4. Способы создания таблиц

Основной принцип создания баз данных заключается в не повторении данных, т. е. разбиении совокупности данных по типам несомой ими информации. Например, чтобы не делать одной большой таблицы, содержащей данные о покупателе, его заказе, дате покупки, заказанных им товарах и прочее, необходимо разбить эти данные на несколько таблиц, одна из которых, к примеру, будет называться Заказчик и содержать исключительно его данные (ФИО, адрес, телефон, ИНН и т. д.), другая Заказы, содержащая исключительно данные о заказах (ИНН заказчика, номер заказа, дата и т. д.).

Такой подход облегчает поиск данных в базе и экономит место на электронных носителях.

Столбцы таблиц называются *полями*.

Одно (или несколько) поле (полей) таблицы должно быть уникальным, т. е. данные в этом поле не могут повторяться. Уникальное поле (поля) назначаются *ключевыми*. Примерами ключевых (уникальных) полей *числового типа* могут быть *№ паспорта, № страхового полиса*, т. к. не существует двух людей с одинаковыми номерами паспортов и страховых полисов. Уникальным может быть и порядковый номер какого-либо объекта. Также можно назначать ключевыми и поля текстового типа.

Создание таблиц в режиме мастера

После выбора действия *Создать таблицу с помощью мастера* откроется окно *Мастер таблиц* (рис. 5), где необходимо указать категорию создаваемой таблицы *Деловой* или *Персональный.*

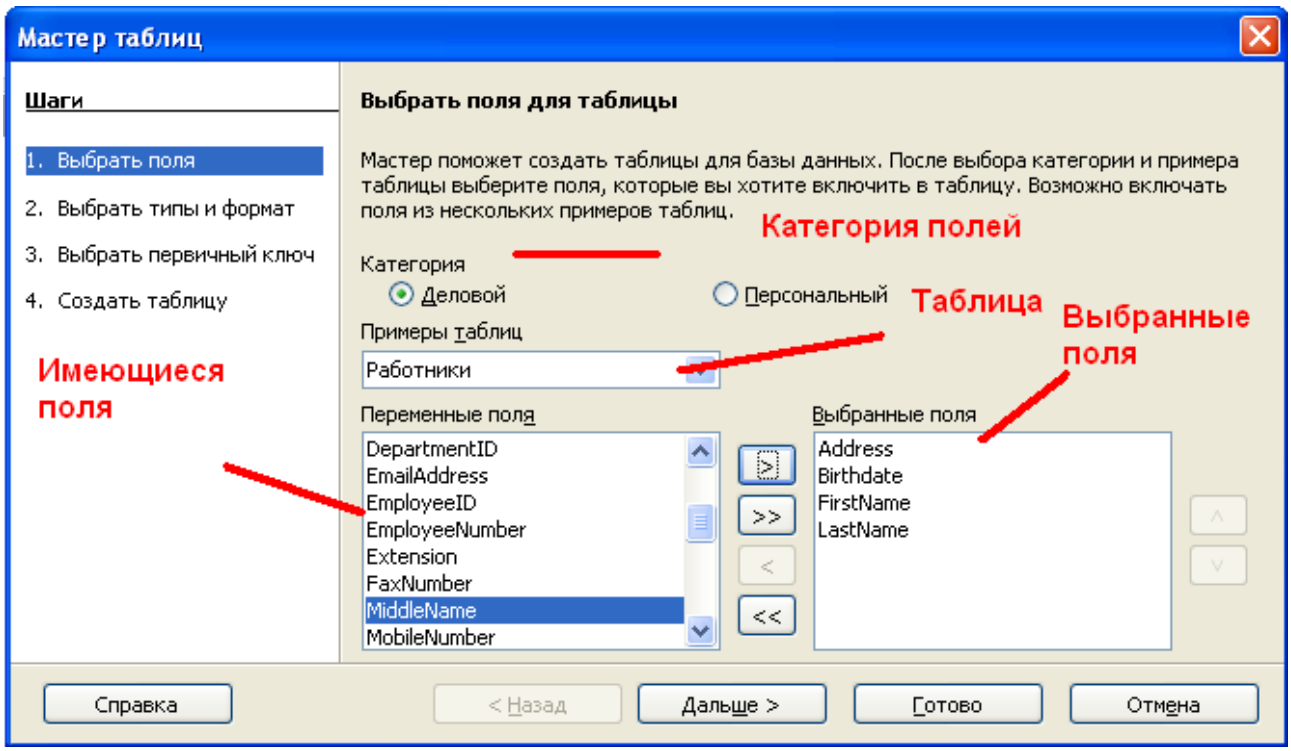


Рисунок 5. Создание таблиц в режиме мастера

Далее из примеров таблиц необходимо выбрать вид создаваемой таблицы, а затем выбрать ее поля. Для выбора полей необходимо в окне *Переменные поля* выделить мышкой нужное поле, а затем нажать >. При этом выделенное поле переместится в окно *Выбранные поля*. Если нажать >>, то в окно *Выбранные поля* переместятся все поля из окна *Переменные поля* (рис. 5).

На рисунке 5 показан пример выбора категории таблицы *Деловой*, вида таблицы *Работники*, полей *Адрес*, *Дата рождения*, *Имя, Фамилия*[1].

После этого необходимо нажать кнопку *Далее*. Появится окно (рис. 6), в котором пользователь должен определить шаги: выбрать поля (как правило, поля в этот момент уже выбраны на предыдущем этапе), выбрать типы и формат, выбрать первичный ключ, создать таблицу (таблица тоже уже создана).

Чтобы выбрать типы и формат в окне *Выбранные поля* необходимо отметить нужное поле, а затем указать его тип, обязательно оно или нет и его размерность (длину символов).

В этом же окне возможна русификация англоязычных полей. Для этого необходимо в графе с указанием имени поля указать имя поля на русском языке (рис. 6).

1 На рисунке поля англоязычные.

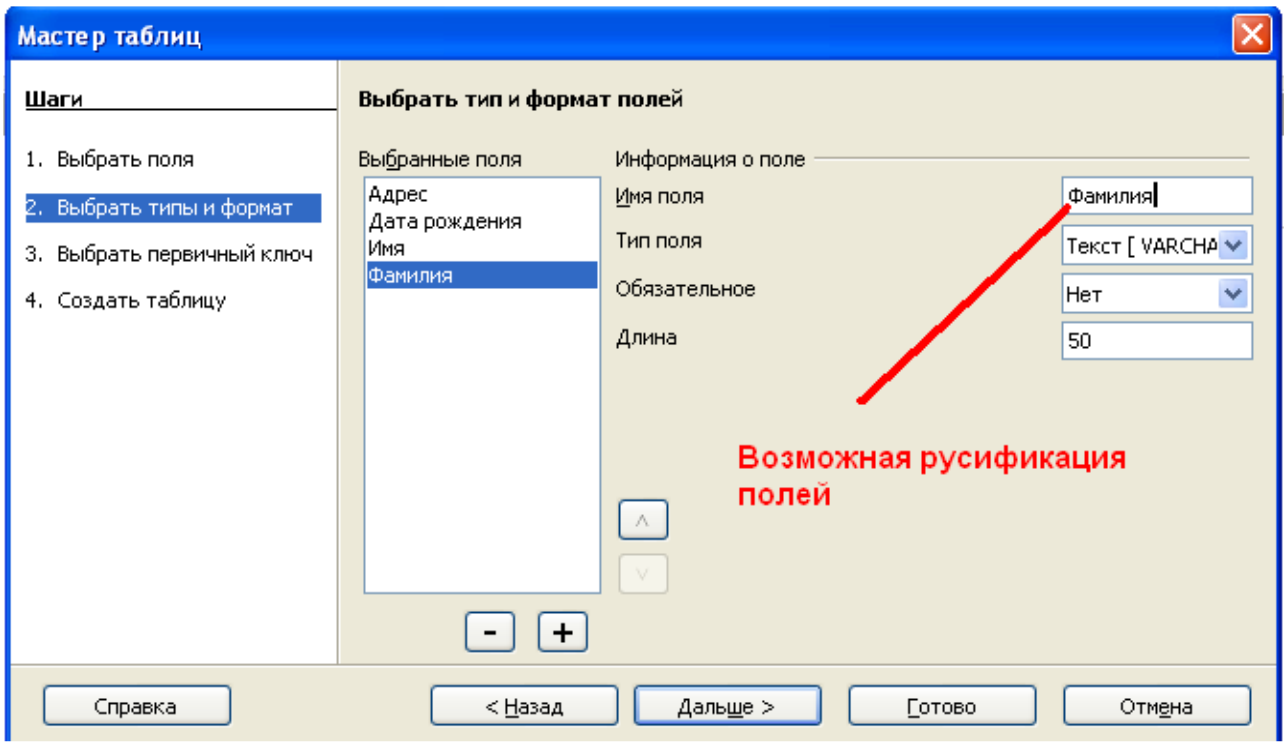


Рисунок 6. Создание таблиц в режиме мастера.

После этого необходимо снова нажать кнопку *Далее*. Появится окно (рис. 7), в котором также указываются ключевые поля таблицы. Чтобы указать ключевое поле, необходимо отметить мышью графу *Использовать существующее поле как первичный ключ* и в появившемся списке выбрать поле, которое будет являться ключевым. Если необходимо сделать ключевыми несколько полей, следует отметить мышью графу *Определить первичный ключ как комбинацию нескольких полей* и выбрать те поля, которые будут являться ключевыми.

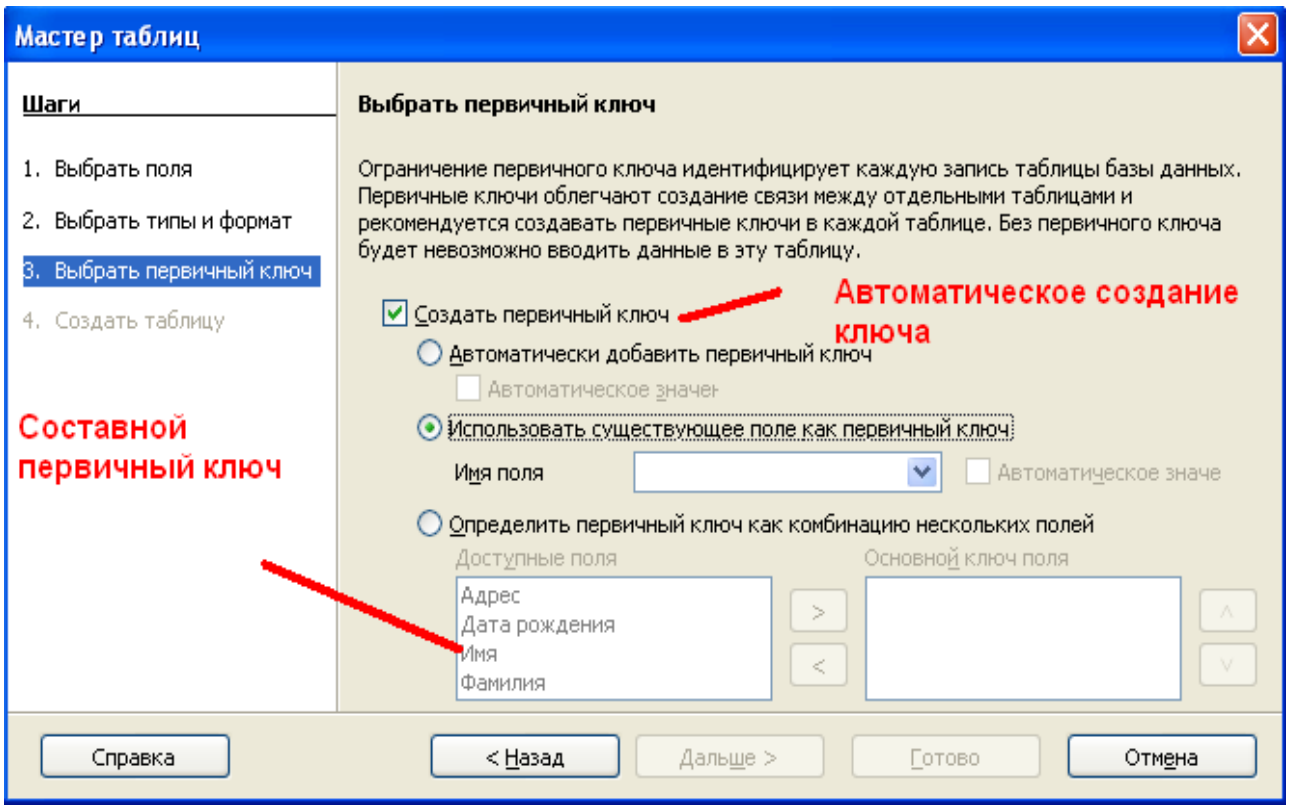


Рисунок 7. Выбор ключа в режиме мастера

После нажатия кнопки Далее пользователю будет предложено ввести имя таблицы. Для этого в специально отведенном для введения имени таблицы поле необходимо ввести ее название. По умолчанию таблица будет названа тем шаблонным именем, которое пользователь выбрал при создании таблицы.

В этом же окне будут предложены дальнейшие действия, которые можно произвести с созданной таблицей: *Немедленно вставить данные, Модифицировать дизайн таблицы, Создать форму на основе данной таблицы*. Для выбора одного из действий его необходимо отметить точкой.

Созданную в любом режиме таблицу можно редактировать. Для этого необходимо из контекстного меню, вызванного относительно имени таблицы, выбрать команду *Правка*. Таблица откроется в режиме дизайна.

Создание таблиц в режиме дизайна

После выбора действия *Создать таблицу в режиме дизайна* откроется окно, представляющее собой сложную таблицу, разделенную на несколько секторов: *имя поля, тип поля, описание, свойства поля* (рис. 8).

В секторе *имя поля* пользователь указывает название полей (столбцов) проектируемой таблицы. В секторе *тип поля* пользователь указывает тип вводимых в соответствующее поле данных. Существует несколько типов полей, наиболее часто используемыми из которых являются: *текст, число, дата, время, логическое.* В секторе *описание* пользователь указывает особенности вводимой в поля информации. В секторе свойства поля пользователь может указать обязательно ли поле или нет, длину вводимых в поля данных, значение поля, которое будет использовать база по умолчанию пользователя и пример формата (рис. 8).

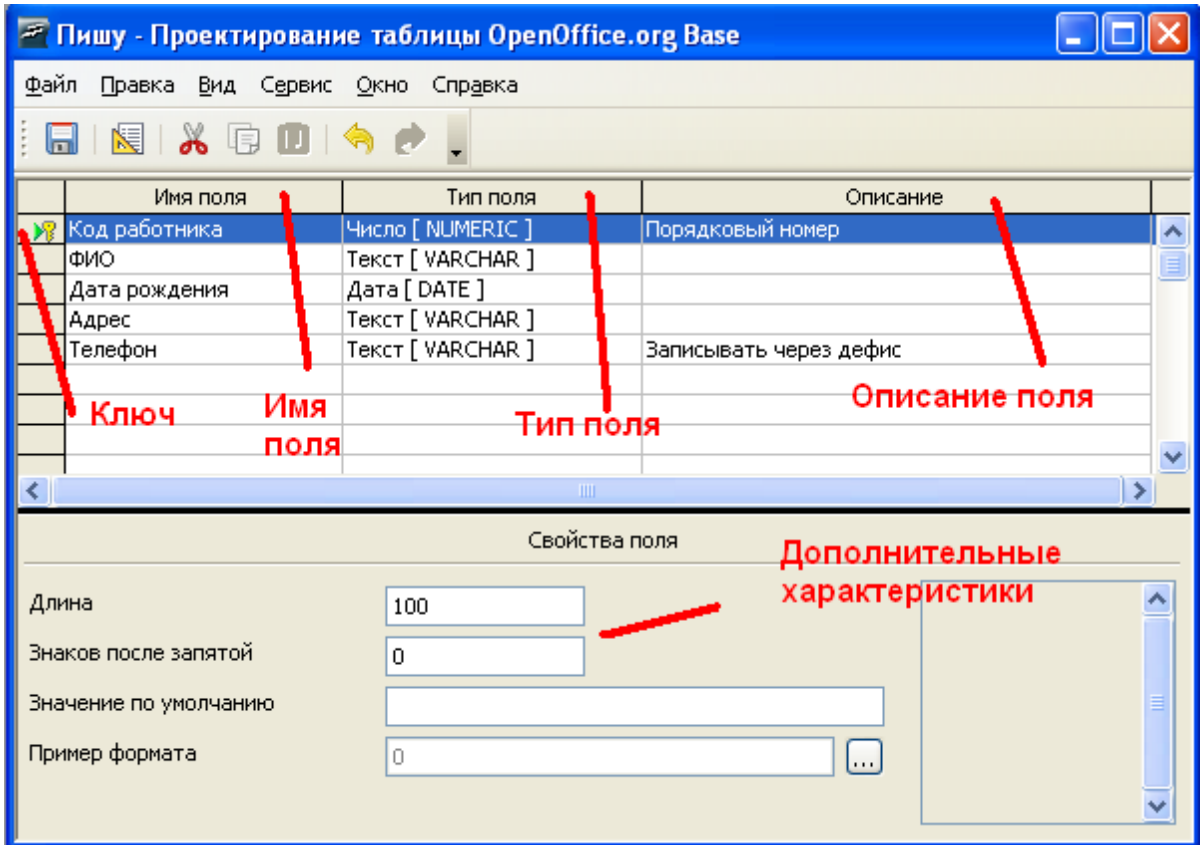


Рисунок 8. Проектирование таблиц в режиме дизайна

Как уже отмечалось ранее, важной особенностью проектирования баз данных является обозначение ключевого поля (ключевых полей). Ключевым полем является поле, значение которого в таблице не может повторяться.

Например, в приведенном примере (рис. 8) ключевым является поле *Код работника*. Значение этого поля повторяться. не может. В данном примере делать ключевым поле *ФИО* работника не целесообразно, т. к. в одной организации вполне могут находится работники с одинаковыми фамилией, именем и отчеством. Для других примеров можно делать ключевыми такие поля, *как № паспорта, ИНН, № страхового полиса, №*

пенсионного удостоверения и прочее, т. к. эти номера являются уникальными для каждого отдельно взятого физического лица.

Иногда целесообразно делать составной ключ, т. е. делать ключевым сразу несколько полей таблицы. Например, можно сделать ключом и *поле № паспорта* и *поле ИНН*.

Для того, что бы в режиме дизайна указать, какое поле является ключевым, необходимо справа относительно ключевого поля вызвать контекстное меню и из появившейся таблицы выбрать *Первичный ключ.*

Для того, чтобы снять первичный ключ необходимо проделать те же действия относительно поля, с которого необходимо снять ключ. Для того, чтобы сделать составной ключ необходимо, удерживая клавишу <Ctrl>, выделить те поля, которые будут являться ключевыми. При этом поля должны располагаться последовательно.

После того, как проектирование таблицы в режиме *дизайна* закончено, необходимо сохранить данные. Для этого необходимо нажать кнопку Сохранить на панели инструментов, после чего в появившемся окне указать имя созданной таблицы.

Например, показанная на рисунке 9 таблица будет называться *Работники*.

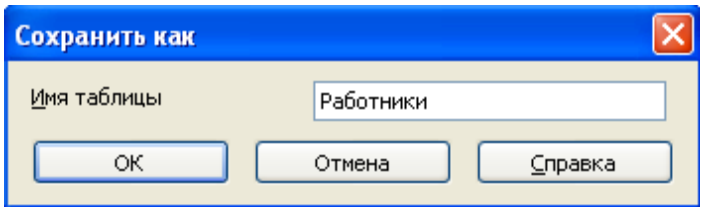


Рисунок 9. Ввод названия таблицы

Далее таблицу можно закрывать. Теперь в основном окне проектирования баз данных в разделе *Таблицы* можно увидеть созданную таблицу (рис. 10).

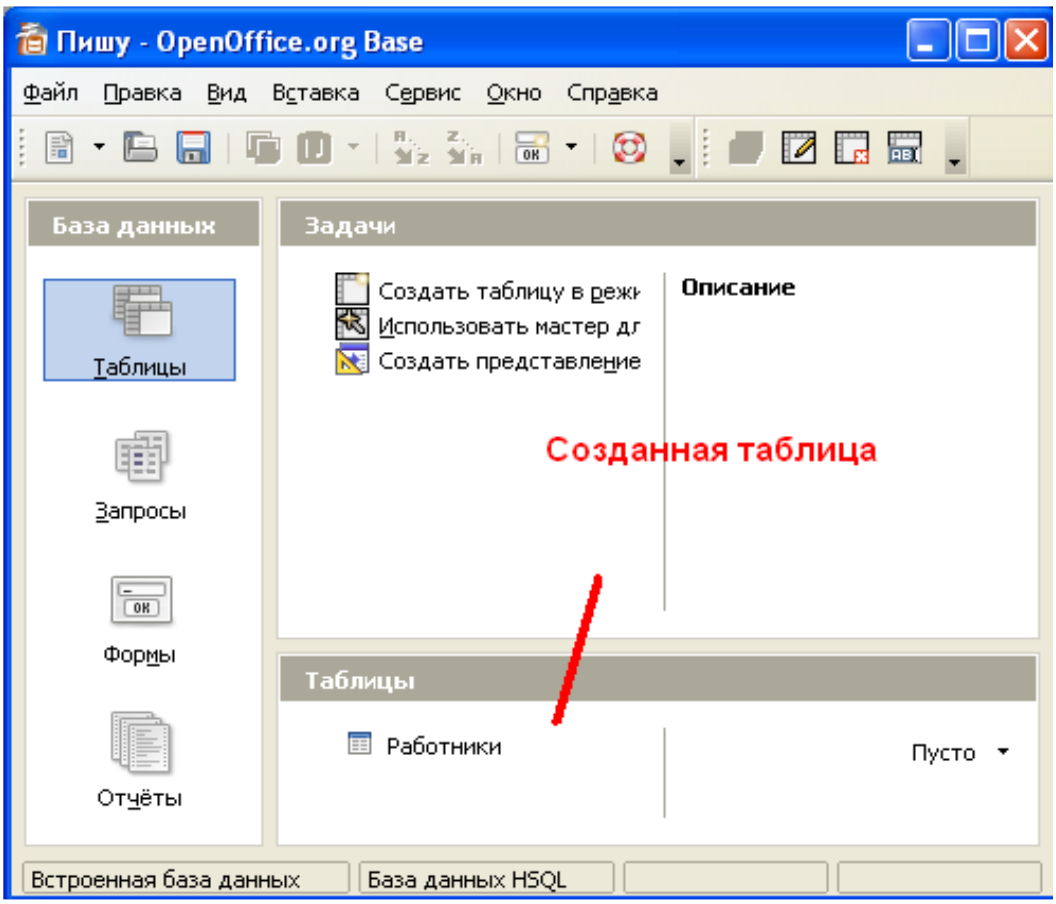


Рисунок 10. Созданная таблица

Установление связей между таблицами

Самым важным этапом для работы базы данных является создание связей между таблицами. Связи необходимы для того, что бы разрозненные таблицы работали как единое целое, т. е. как одна большая таблица.

Существуют следующие типы связей:

- Один к одному;
- Один ко многим;
- Много к одному;
- Много ко многим.

Связь один к одному устанавливается между ключевым и ключевым полем.

Связь один ко многим устанавливается между ключевым и не ключевым полем.

Связь много к одному устанавливается между не ключевым и ключевым полем.

Связь много о многим фактически представляет собой две связи с отношением один-ко-многим через третью таблицу, ключ которой состоит, по крайней мере, из двух полей.

Ключ, состоящий из одного поля называется *простым*, из двух полей и более - *составным*.

Связи устанавливаются между полями одинаковых типов (число-число, дата-дата, текст-текст и т .д.).

Прежде чем установить связь, составитель базы данных должен продумать ее логически. Например, на одной кафедре Вуза может быть только один заведующий кафедрой. Следовательно, связь между таблицами *Кафедра* и *Заведующий* должна быть *один к одному*. Или один заказчик может делать много заказов. Следовательно, связь между таблицами *Заказчик* и *Заказ* должна быть типа *один ко многим*. Или несколько Вузов имеют несколько кафедр и наоборот несколько различных кафедр могут располагаться в нескольких различных вузах. В последнем случае связь между таблицами *Вузы* и *Кафедры* должна быть *много ко многим*.

Для установления связей в базе данных *Base Open Office* необходимо в главном меню *Сервис* выбрать команду *Связи*. После этого откроется окно (рис. 11), в котором будет предложено добавить ранее созданные таблицы. Для добавления необходимых таблиц их необходимо выделить и нажать кнопку *Добавить*. После добавления таблиц окно *Добавить таблицы* следует закрыть.

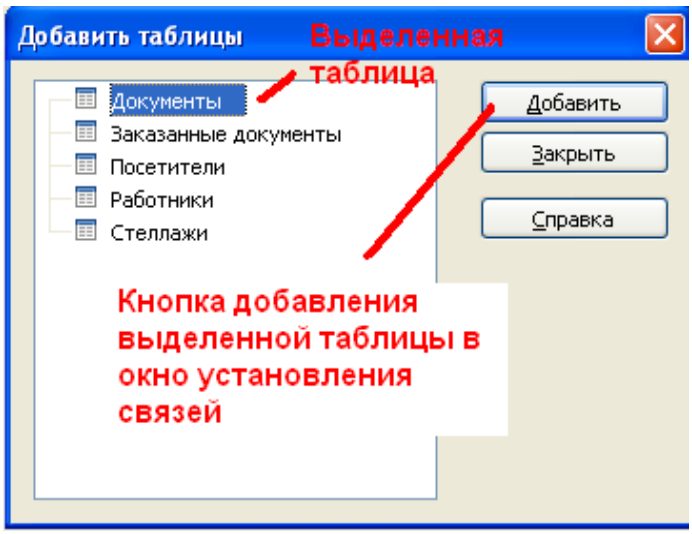
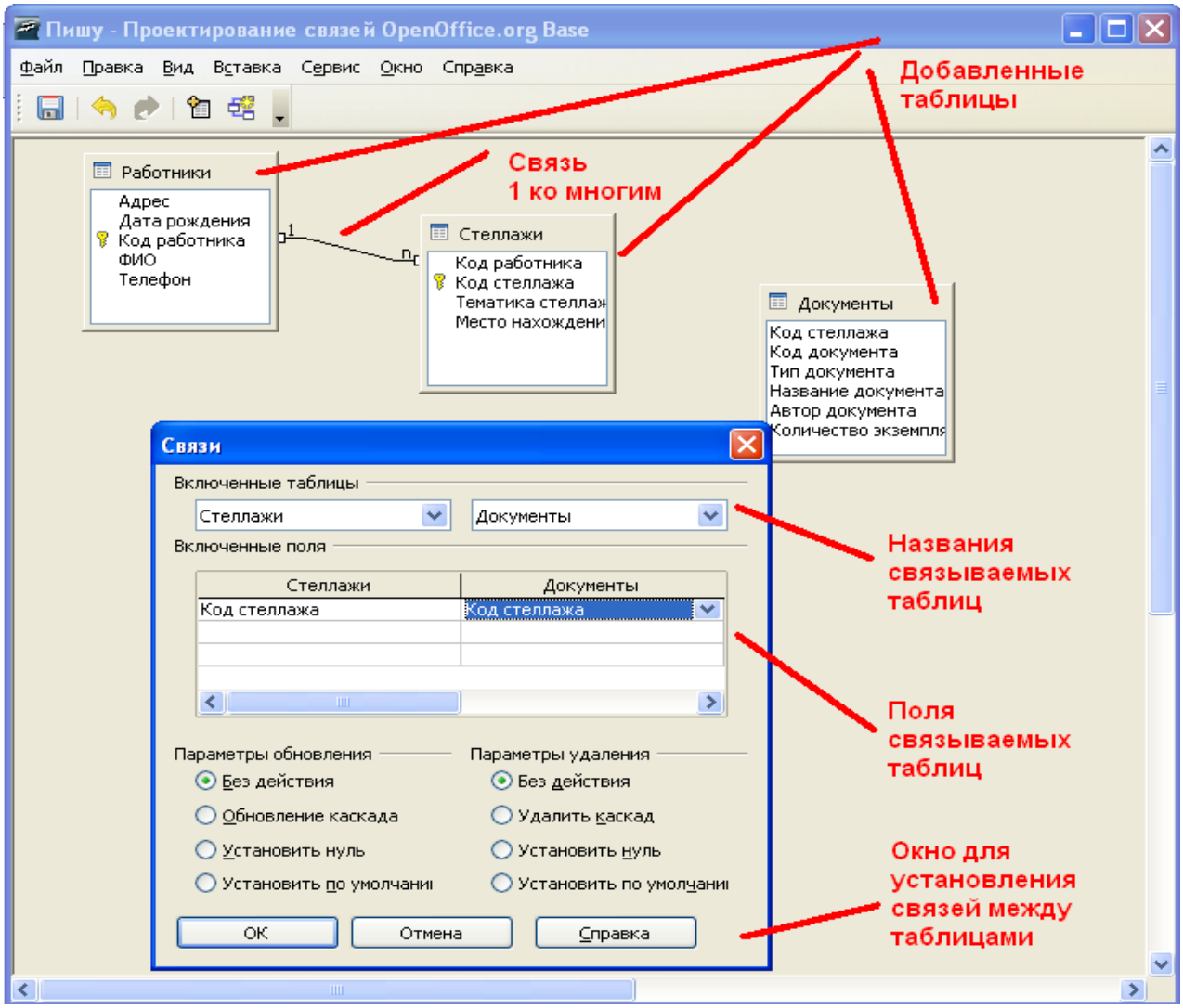


Рисунок 11. Добавление таблиц в окно создания связей

Теперь между добавленными таблицами необходимо установить связи. Для этого на панели инструментов в открытом окне необходимо нажать кнопку *Создать связь* или, удерживая нажатой левую кнопку мыши, перетащить связываемое поле одной таблицы на связываемое поле другой таблицы. После этого откроется окно, где будет предложено указать связываемые поля таблиц (рис. 12). Все существующие в таблицах поля указываются полем со списком. После этого, как связываемые поля



будут выбраны, необходимо нажать кнопку Ок.

Рисунок 12. Установление связей между таблицами

После установления связей таблицы можно заполнить данными[2]. Следует отметить, что установленные связи накладывают на ввод данных соответствующие типам данных ограничения.

2 Также таблицы можно заполнять с помощью форм (Практическая работа № 2).

1.3. Задание на работу

Создание базы данных «Архив»

1. Создать таблицу *Работники*, имеющую пять полей:

- Код работника (число);

- ФИО работника (текст);

- Дата рождения (дата);

- Адрес проживания (текст);

- Домашний телефон (текст)[3].

Поле Код работника сделать ключевым.

2. Создать таблицу *Стеллажи*, имеющую четыре поля:

- Код работника (число);

- Код стеллажа (число);

- Тематика стеллажа (текст);

- Место нахождения стеллажа (текст).

Поле Код стеллажа сделать ключевым.

3. Создать таблицу *Документы*, имеющую шесть полей:

- Код стеллажа (число);

- Код документа (число);

- Название документа (текст);

- Автор документа (текст);

- Количество экземпляров (число);

- Стоимость одного часа пользования (число)[4].

Поле Код документа сделать ключевым.

4.Создать таблицу *Посетители*, имеющую четыре поля:

3 Если цифры номера телефона разделяются знаком «тире», то поле должно быть текстовом.
4 В этом поле следует указывать только цифру.

- Код посетителя (число);

- ФИО посетителя (текст);

- Место работы/учебы (текст);

- Должность (текст).

Поле Код посетителя сделать ключевым.

5. Создать таблицу *Заказанные документы*, имеющую три поля:

- Код посетителя (число);

- Код документа (число);

- Количество заказанных часов (число).

Поля Код посетителя и Код документа сделать ключевыми.

6. Установить следующие типы связей:

- между таблицами Работники и Стеллажи - один ко многим;

- между таблицами Стеллажи и Документы - один ко многим;

- между таблицами Посетитель и Заказ - один ко многим;

- между таблицами Документы и Заказ - один ко многим.

7. Произвольно заполнить таблицы (не менее 5 записей).

Принцип кодирования ключевых полей разрабатывается самостоятельно!

1.4. Требуемые результаты

В итоге должны получится типы связей (рис. 13), которые читаются следующим образом:

- каждый работник архива ответственен за несколько стеллажей;
- на каждом отдельно взятом стеллаже хранится несколько документов;
- каждый посетитель архива может заказать несколько документов;
- каждый документ может быть несколько раз заказан.

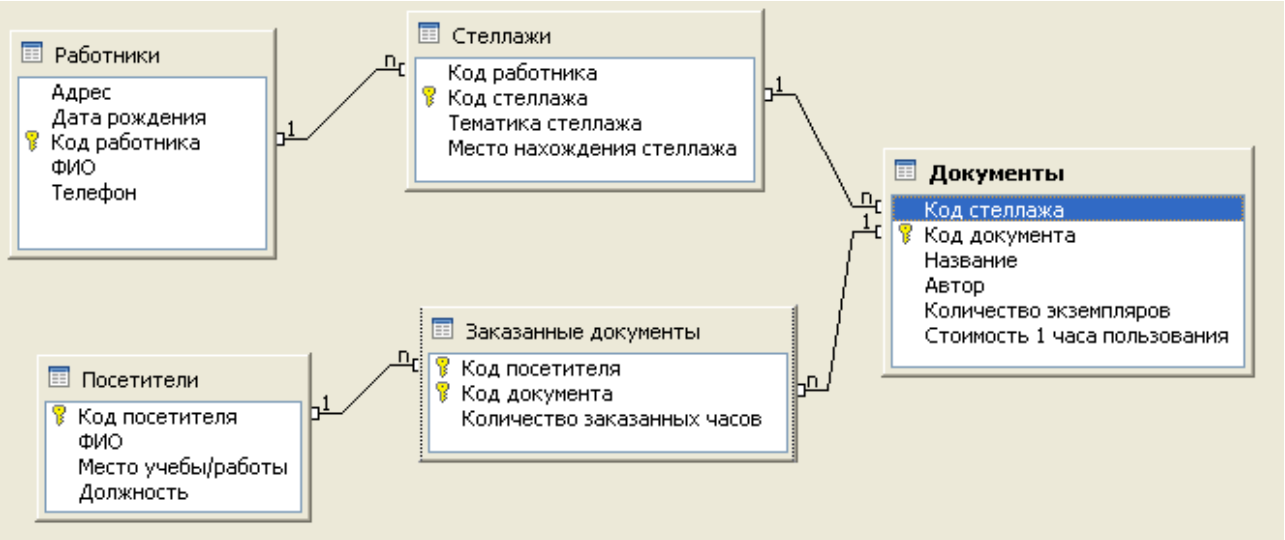


Рисунок 13. Связи базы данных *Архив*

1.5. Оформление отчета по работе

Отчет должен содержать:

1. Номер практической работы.

2. Название практической работы.

3. Цель практической работы.

4. Описание пунктов выполнения практической работы в соответствии с заданием на работу.

5. Вывод по работе.

1.6. Контрольные вопросы

1. Что такое СУБД?
2. Как создать файл Open Office.org.Base?
3. Какие существуют способы создания таблиц?
4. Чем отличаются способы создания таблиц?
5. Какие существуют типы связей?

6. Как создать связь между таблицами?
7. Как изменить связь между таблицами?
8. Что такое поле?
9. Что такое типы полей?
10. Какие типы полей существуют?
11. Какое поле называется ключевым?
12. Что такое запись?

Практическая работа № 2

Создание форм в базе данных Base Open Office.

2.1. Цель работы

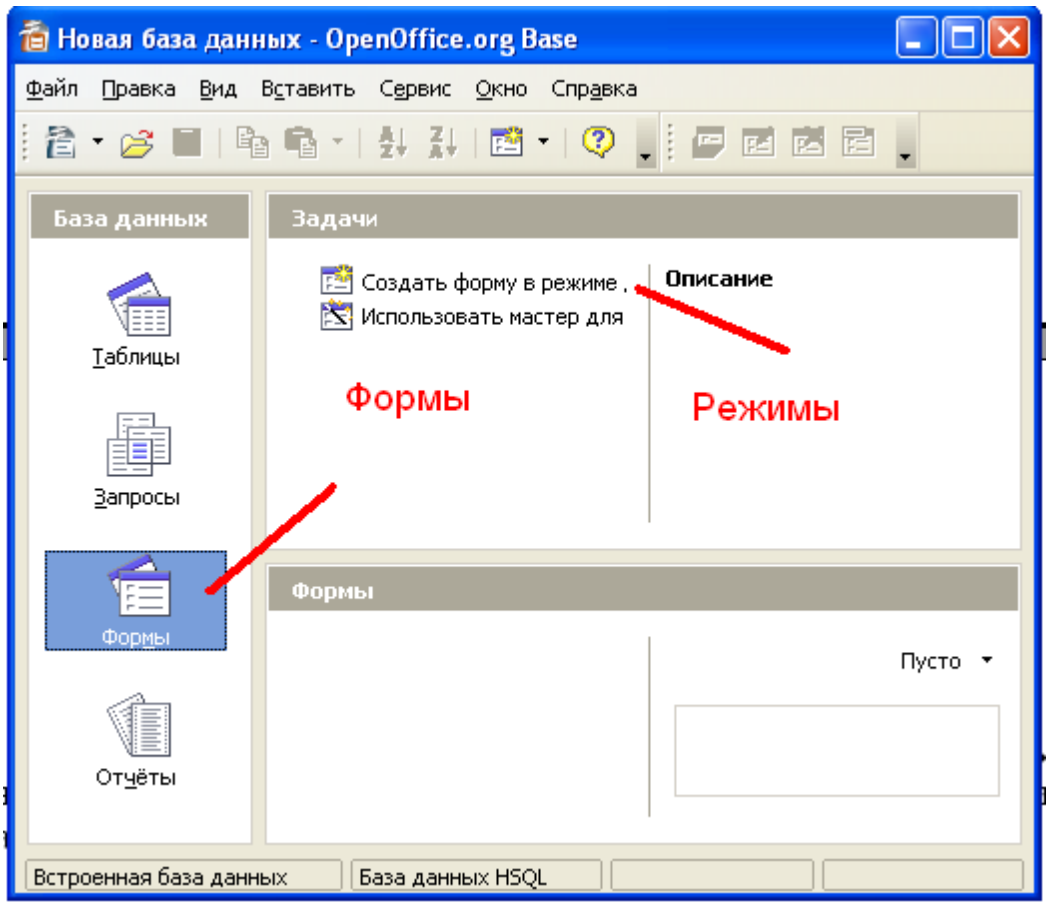
Приобретение навыков по созданию форм

в базе данных Base Open Office

2.2. Теоретические положения

Формы в СУБД необходимы для заполнения таблиц данными. Данные, внесенные в форму, автоматически отображаются в соответствующей таблице. Формы также необходимы для отображения данных базы в упорядоченном и привлекательном виде. Формы используют для редактирования, просмотра и печати данных.

Чтобы создать форму необходимо в окне проектирования базы



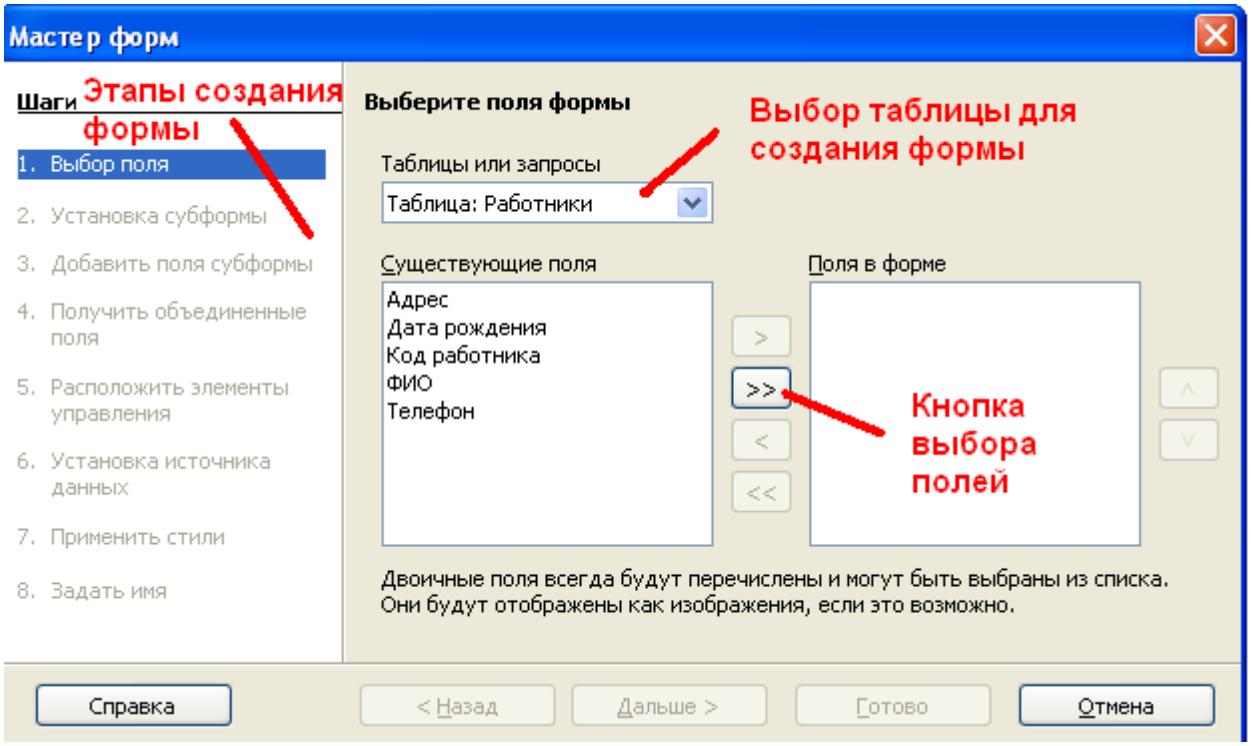
данных выбрать объект *Формы* (рис. 1).

Рисунок 1. Создание форм в Base Open Office

Для создания форм в Base Open Office, также как и для создания таблиц, можно использовать *режим мастера* и *режим дизайна.*

Создание форм в режиме мастера

После выбора для создания форм режима *мастера* откроется окно (рис.2), в котором необходимо указать таблицу для которой будет создана форма, а также поля таблиц, которые будут заполняться посредством создаваемой формы. Поля выбираются аналогично выбору полей при



создании таблиц в режиме *мастера*.

Рисунок 2. Создание форм в режиме *мастера*

После выбора полей необходимо нажать кнопку *Дальше.* Появляющиеся диалоговые окна позволяют:

- устанавливать субформу, т. е. создавать сложную форму, имеющую в своем составе основную и вложенную формы, соответствующие двум разным таблицам[5];
- получать объединенные поля (в случае установления субформы);
- располагать элементы управления формы, т. е. выбирать из предложенных шаблонов нужный вид формы;
- устанавливать источники данных, т.е. либо режима, при котором форма будет использоваться только для ввода новых данных, при этом ранее

5 Установление субформы приведено ниже.

вносимые данные отображаться не будут, либо режима отображения всех данных, при этом возможны варианты запрета изменения или удаления существующих данных, а также вариант запрета внесения новых данных (рис. 3);

- применять стили, такие как цвет и обрамление поля;
- присваивать имя созданной форме.

Все возможные этапы создания формы перечислены в диалоговых окнах создания форм, причем выделяется этап, соответствующий текущему моменту (рис. 2, 3).

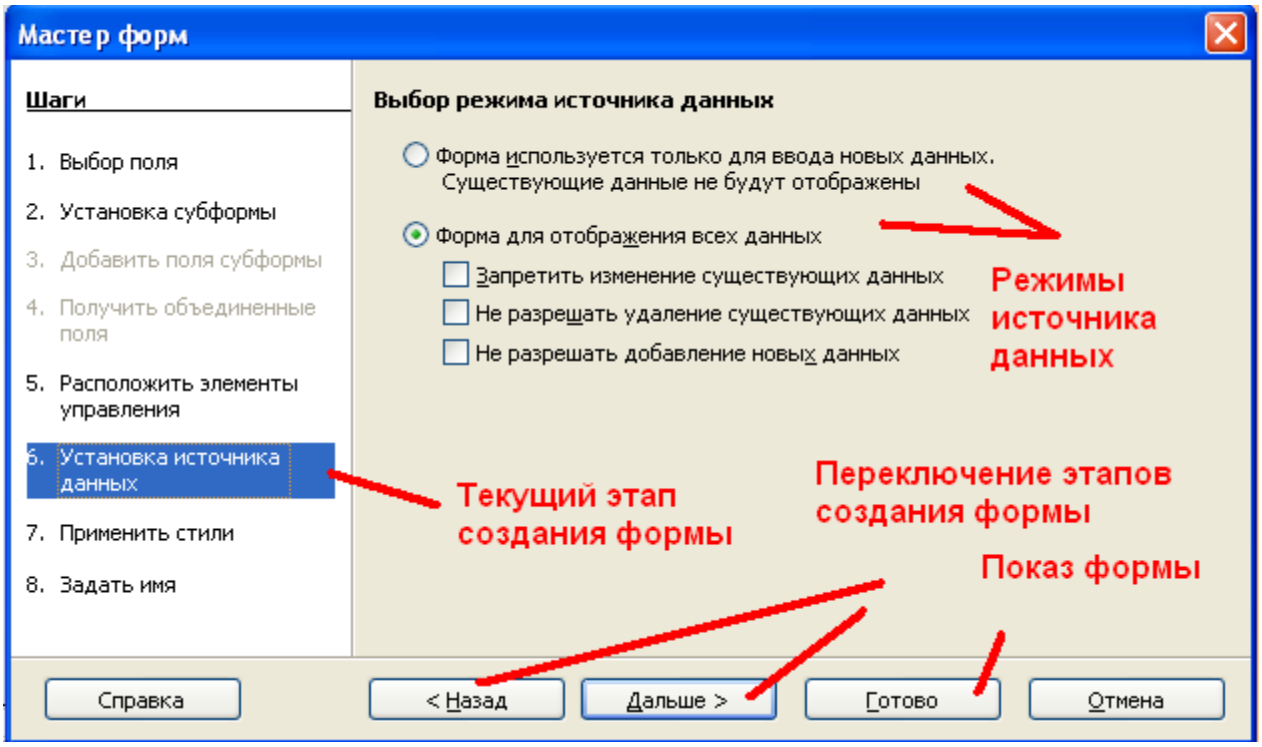


Рисунок 3. Этап создания формы *Установка источника данных*

После указания всех необходимых параметров необходимо нажать кнопку *Готово* (рис. 3). Кнопку *Готово* можно нажать и не проходя все этапы создания формы, т. е. сразу же после выбора таблицы и необходимых полей.

На рисунке 4 показана готовая форма для внесения данных в таблицу *Работники*.

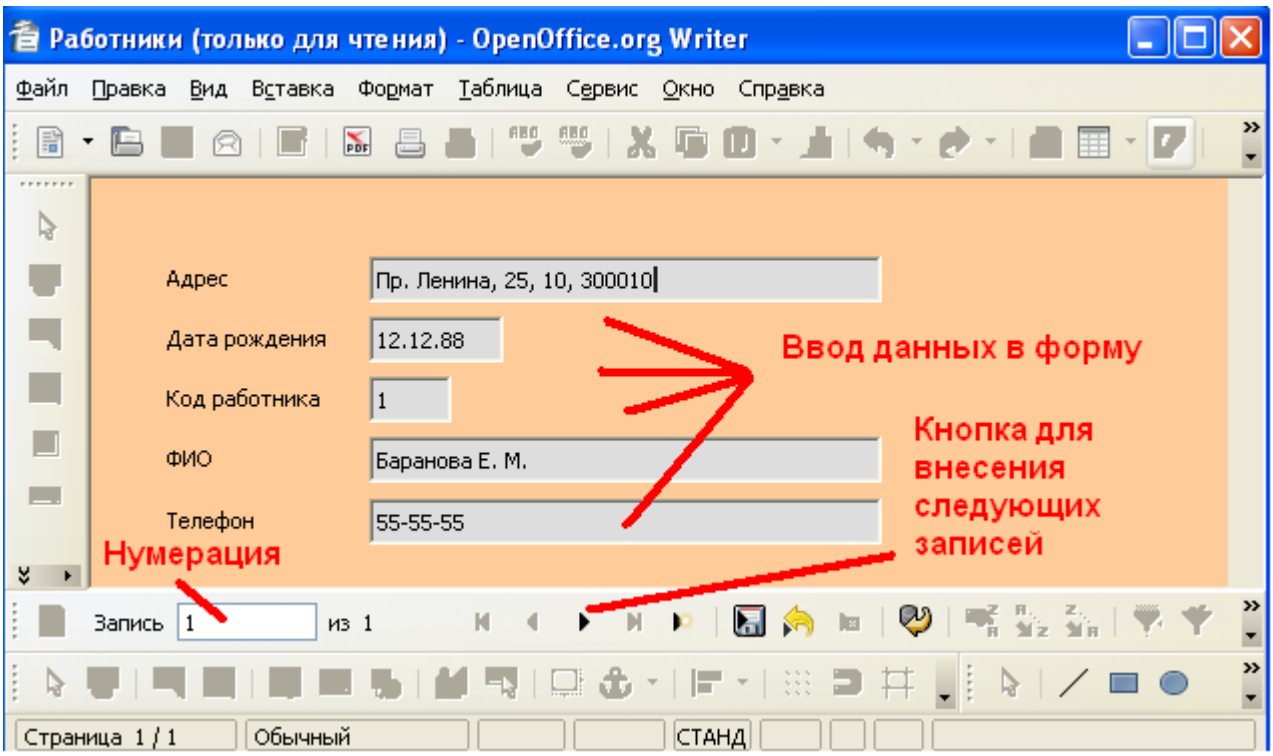


Рисунок 4. Готовая форма для таблицы *Работники*

После того как одна запись будет внесена, необходимо нажать на кнопку, указанную на рисунке 4. После нажатия этой кнопки поля для новых данных обновятся для внесения следующей записи. Количество записей, вносимых в форму, а соответственно и в таблицу, нумеруется (рис. 4).

Иногда необходимо создать сложную форму (субформу), имеющую в своем составе основную форму и подчиненную. Это необходимо для более оперативного ввода данных в таблицы. Для создания сложной формы необходимо в окне, показанном на рисунке 2, после выбора таблицы и соответствующих полей нажать кнопку *Дальше*. В появившемся окне (рис. 5) необходимо отметить флажком *Добавить субформу*, а затем нажать *Далее.* После этого появится окно, аналогичное представленному на рисунке 2, в котором необходимо указать таблицу и поля, которые будут соответствовать непосредственно вложенной (подчиненной) форме.

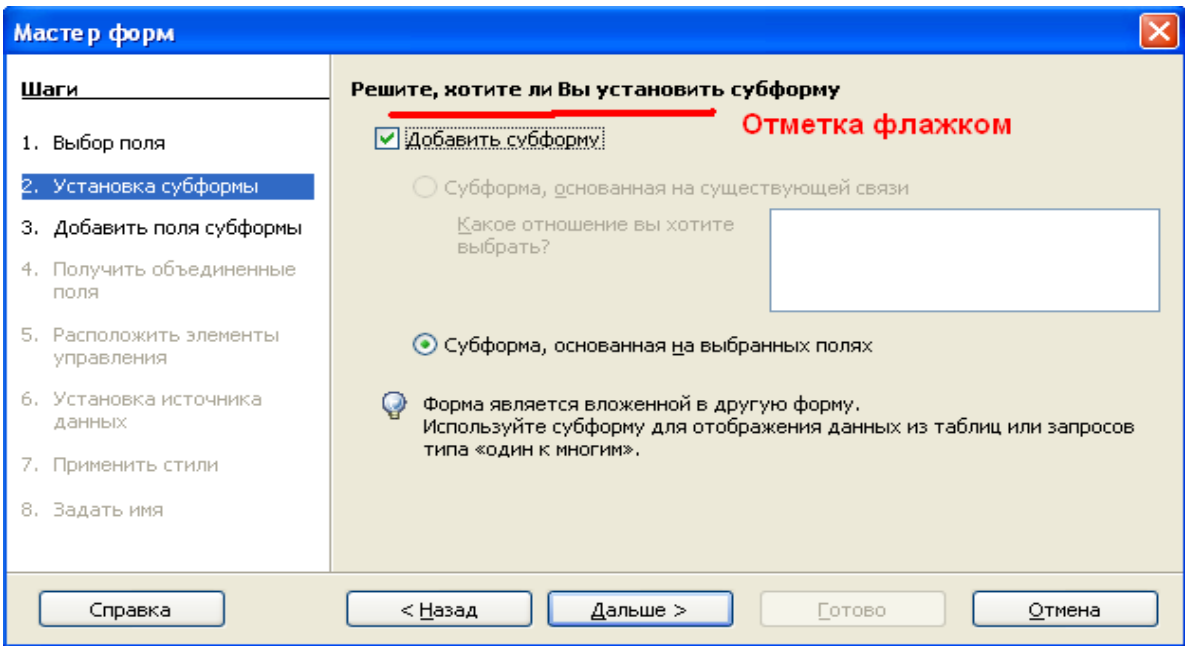


Рисунок 5. Установка субформы

После последующего нажатия кнопки *Далее* можно указать порядок связи полей форм (рис. 6).

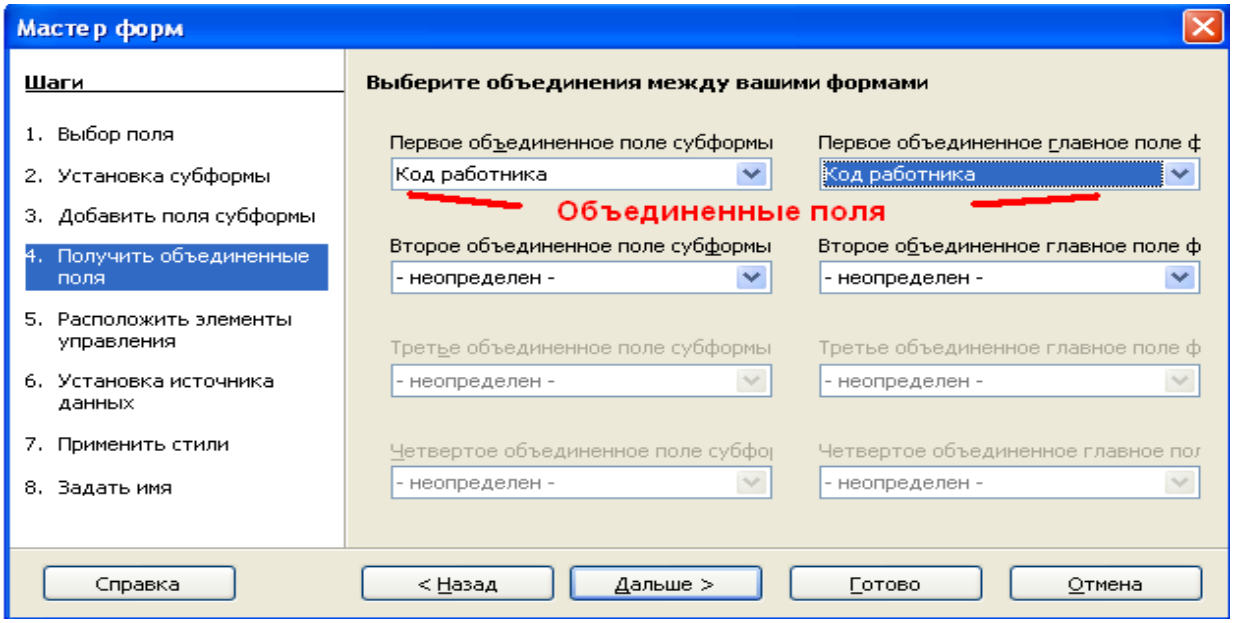


Рисунок 6. Объединение полей при установке субформы

Дальнейшие этапы создания субформы аналогичны этапам создания простой формы.

На рисунке 7 показана готовая субформа, имеющая основной форму *Работники* и вложенной форму *Стеллажи.*

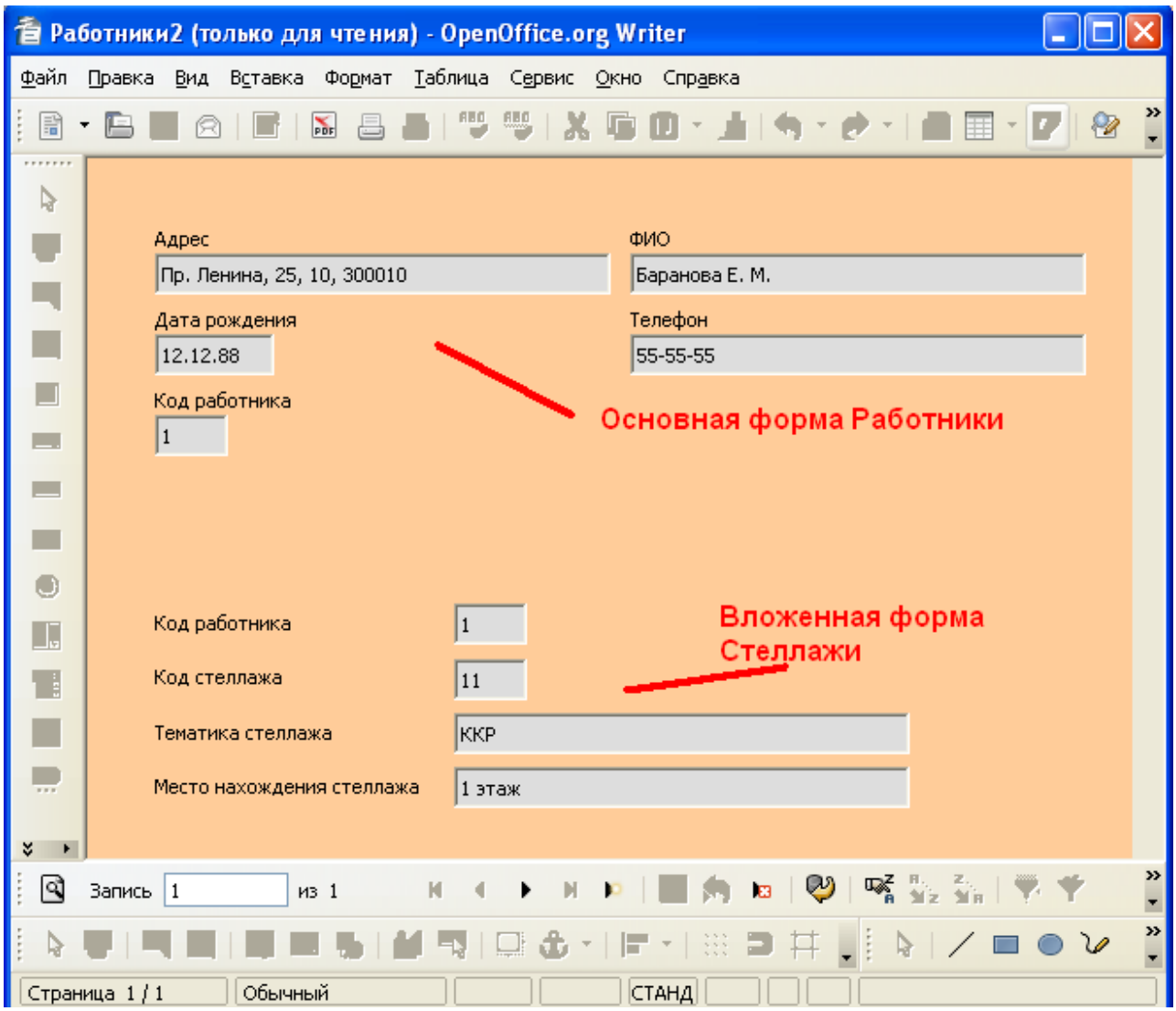
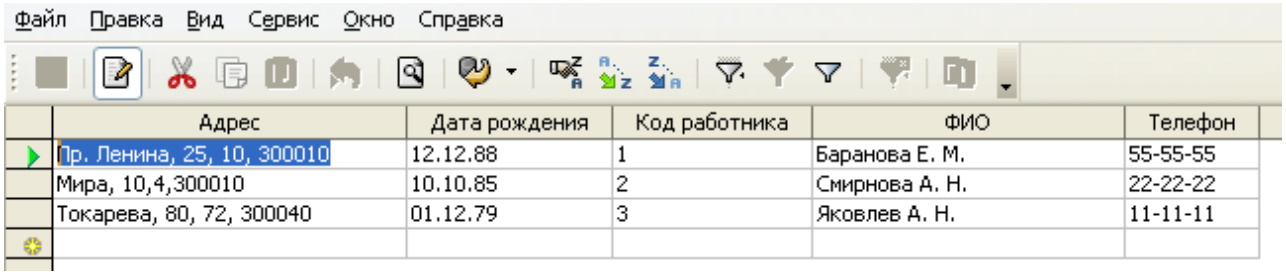


Рисунок 7. Готовая субформа

Как уже отмечалось выше, при внесении данных в формы заполняются соответствующие таблицы (рис. 7, 8)[6].



Адрес	Дата рождения	Код работника	ФИО	Телефон
Пр. Ленина, 25, 10, 300010	12.12.88	1	Баранова Е. М.	55-55-55
Мира, 10,4,300010	10.10.85	2	Смирнова А. Н.	22-22-22
Токарева, 80, 72, 300040	01.12.79	3	Яковлев А. Н.	11-11-11

Рисунок 7. Заполненная таблицы Работники

6 В случае указания при составлении формы соответствующих функций.

Файл Правка Вид Сервис Окно Справка

Код работника	Код стеллажа	Тематика стеллажа	Место нахождения стеллажа
1	11	ККР	1 этаж
1	12	Магистерские диссерта	1 этаж
2	21	ВКР	2 этаж
2	22	Дипломы	2 этаж
3	31	Кандидатские диссерта	1 этаж
3	32	Докторские диссертаци	1 этаж

Рисунок 8. Заполненная таблица Стеллажи

Также как и таблицы, любая форма может быть открыта в режиме дизайна с целью внесения в нее изменений.

Создание форм в режиме дизайна

После выбора для создания форм режима *дизайна* откроется окно (рис.9), на котором, пользуясь панелью инструментов *Элементы управления*, следует расположить элементы формы.

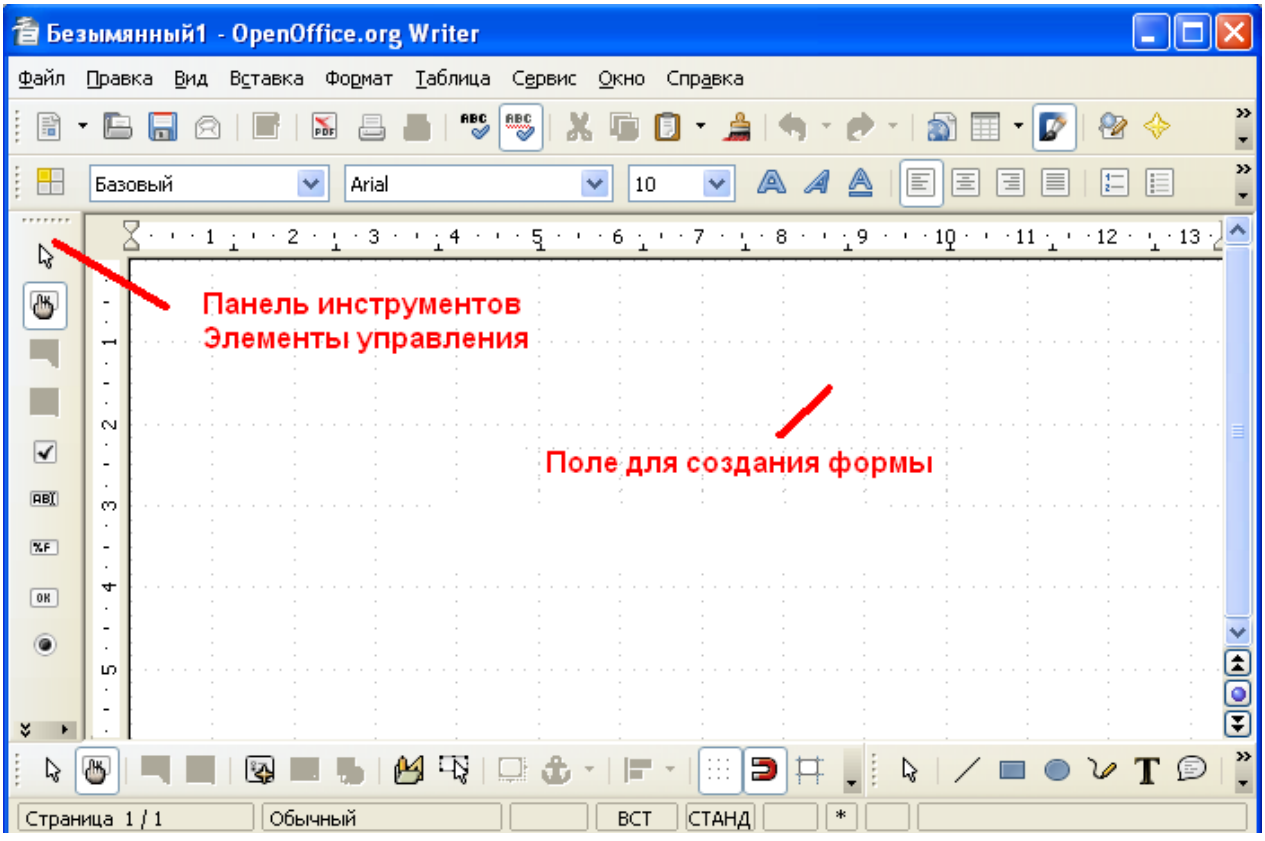


Рисунок 9. Окно для создания форм в режиме дизайна

Создание простой формы

В открывшемся окне следует произвольным образом расположить *элементы управления (элементы формы)*, например, поля (текстовое,

числовое, дата, время и т. д.) и указать названия этих полей. На рисунке 10 показан пример создания формы для таблицы *Работники.* Кнопка, позволяющая создавать текстовые поля также показана на рисунке 10. Свойства полей (элементов формы) можно указать путем выбора из контекстного меню команды *Элементы управления.* В открывшемся окне *Свойства* следует включить закладку *Общие.* Здесь указывается цвет фона, шрифт, обрамление поля (без обрамления или фрейма, плоское, трехмерное) и прочее (рис. 11).

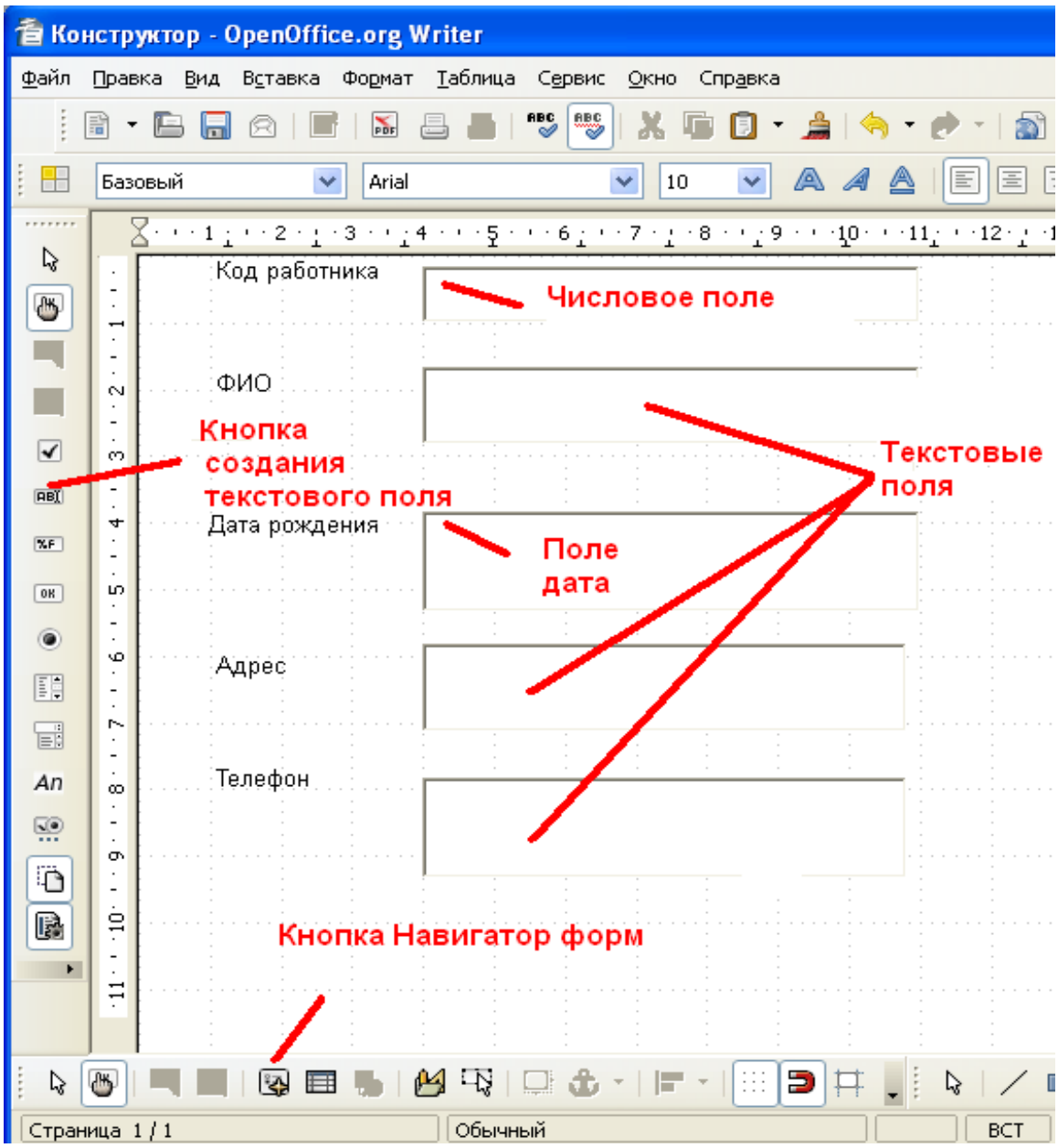


Рисунок 10. Создание форм в режиме дизайна

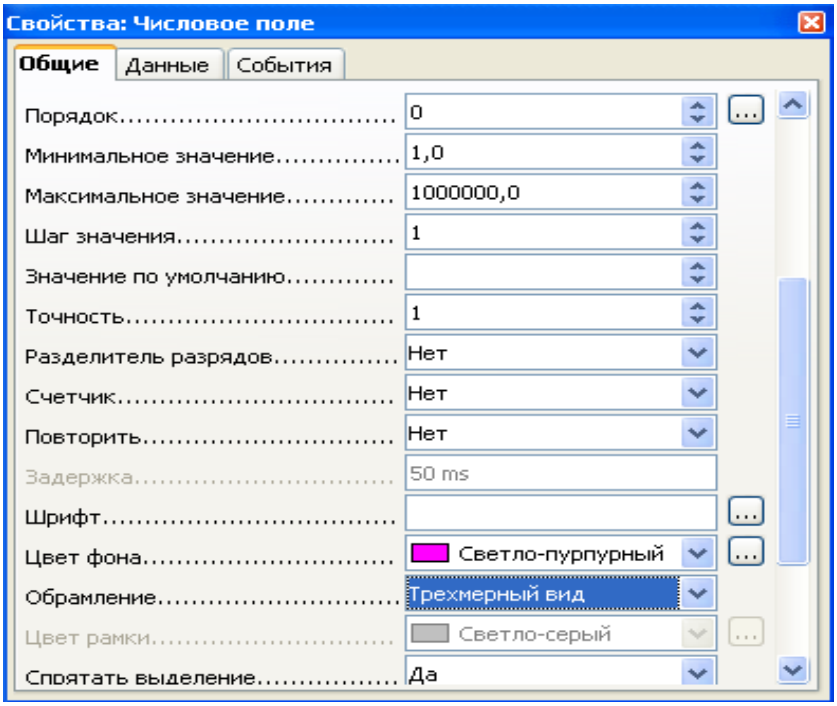


Рисунок 11. Окно Свойства. Закладка Общие

Для того, чтобы заменить текстовое поля на любое другое, необходимо поле, надлежащее замене, выделить, вызвать относительно него контекстное меню и выбрать команду *Заменить на* (рис. 12).

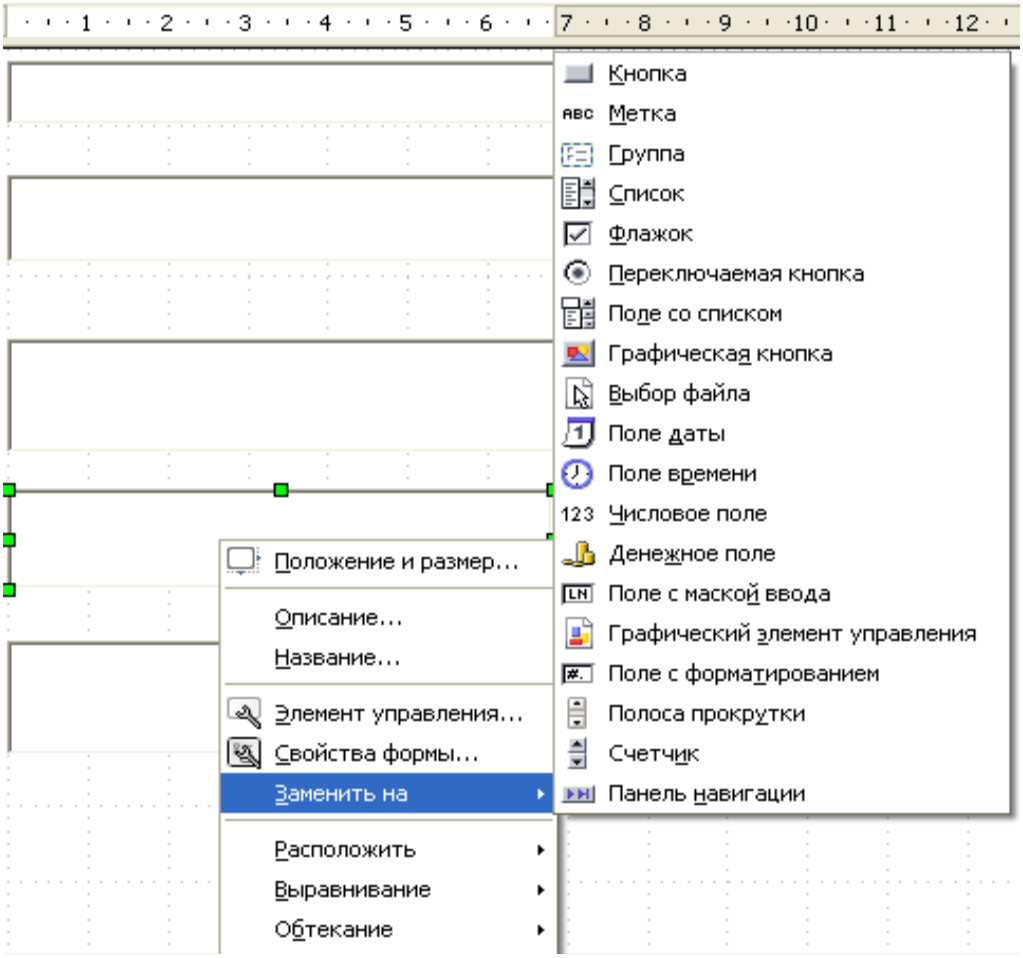


Рисунок 12. Замена поля

Далее следует установить связь между созданной формой и таблицей (запросом), данные которой (которого) будет пополнять или отображать создаваемая форма. Для этого на панели инструментов *Дизайн формы* следует нажать кнопку *Навигатор форм* (рис. 10). Открытое окно *Навигатор форм* показано на рисунке 13.

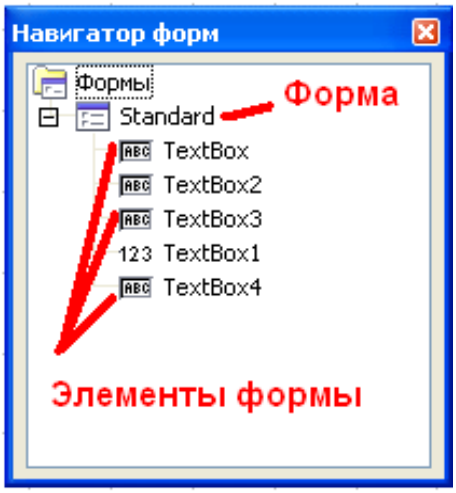
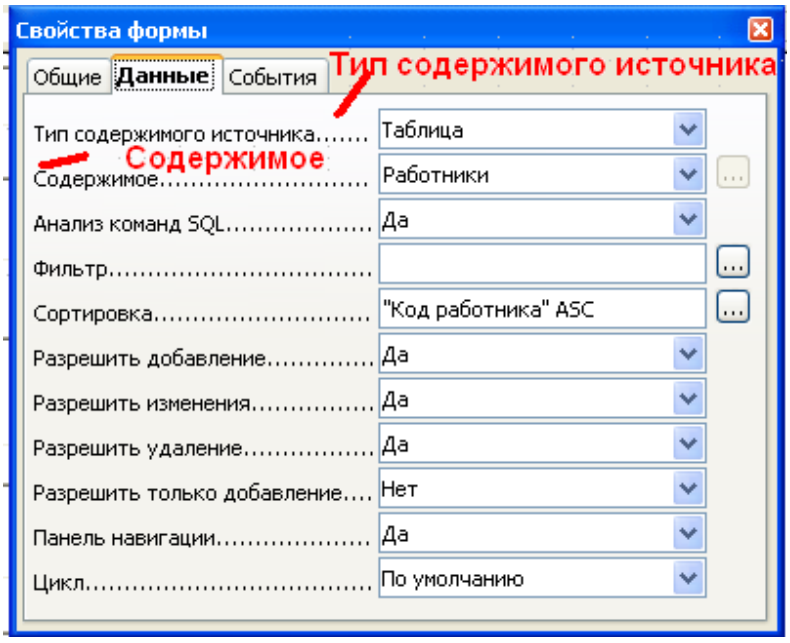


Рисунок 13. Навигатор форм

В окне *Навигатор форм* относительно самой формы (Standard) следует вызвать контекстное меню и выбрать команду *Свойства*. В появившемся окне *Свойства формы* следует выбрать закладку *Данные* и указать *Тип содержимого источника* (таблицу или запрос, данные которых будет отображать или пополнять форма) и *Содержимое* (имя таблицы или запроса). В данном случае *Тип содержимого источника* - таблица,

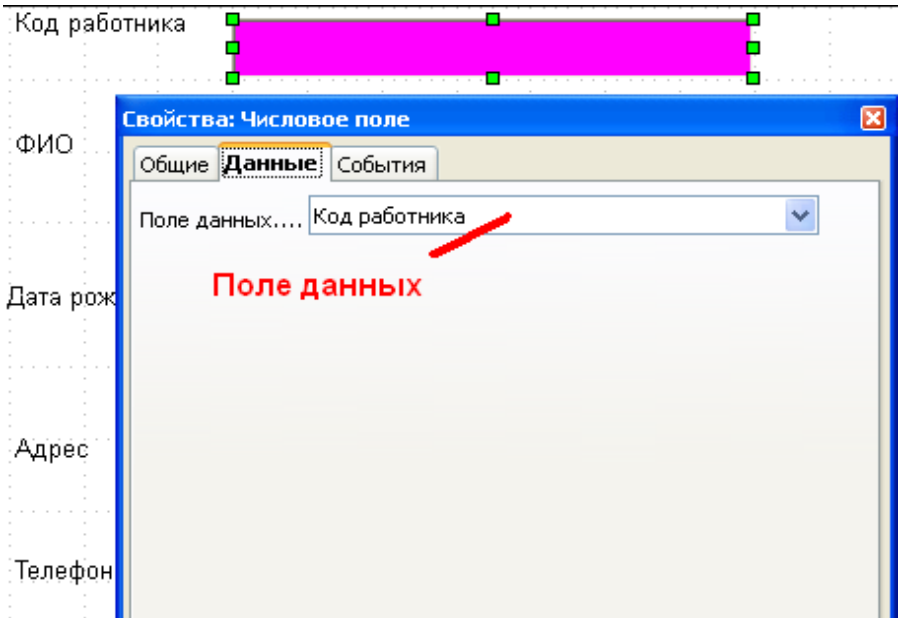


содержимое — Работники (рис. 14).

Рисунок 14. Окно Свойства формы. Закладка Данные

Теперь каждому элементу формы (в данным случаям полям) следует указать поле данных, т. е. то поле таблицы или запроса, которые должно

будет пополниться или отобразиться при запуске создаваемой формы. Для этого в окне *Свойства* (вызываемом через команду *Элементы управления* контекстного меню) следует выбрать закладку *Данные* и выбрать из списка



поле данных (рис. 15).

Рисунок 15. Окно Свойства. Закладка Данные[7]

Созданная таким образом форма также как и созданная в реждиме мастера форма позволяет в зависимости от указанных свойств, вводить или отображать данные таблиц (запросов).

Создание субформы

Для создания субформы в режиме дизайна следует использовать *Навигатор форм.*

Например, создадим субформу, состоящую из трех таблиц: *Работники, Стеллажи, Документы.*

Вначале создается простая форма *Работники.* Затем в окне *Навигатор форм* при помощи контекстного меню добавляется еще одна форма (рис. 16).

7 Окно *Свойства* — окно свойств элементов формы. Окно *Свойства формы* — окно свойств всей создаваемой формы. Окно *Свойства* можно открыть и в окне *Навигатор форм*.

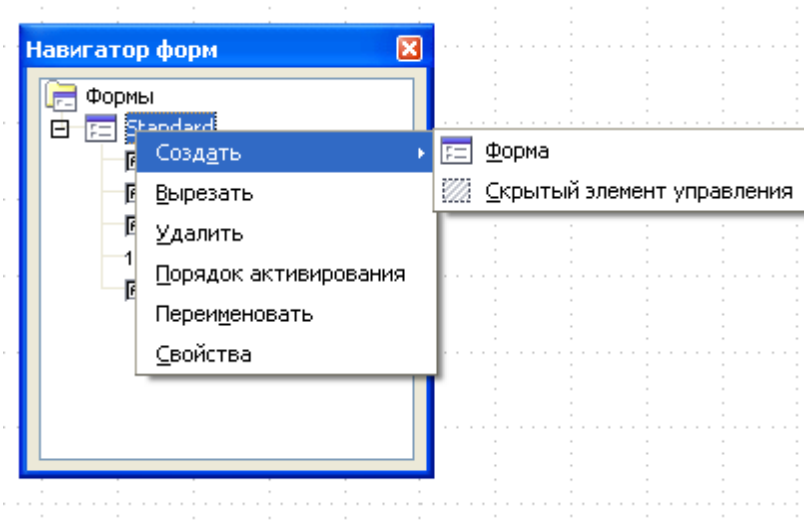
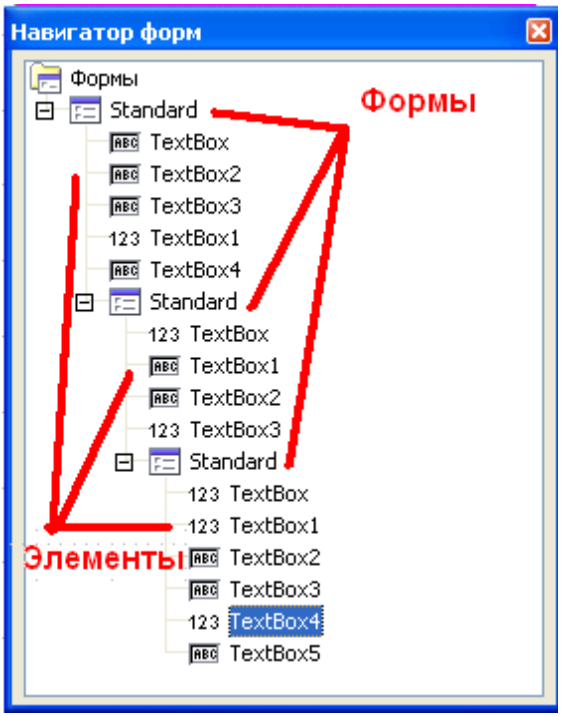


Рисунок 16. Добавление формы

После добавления формы следует обычным способом наносить элементы управления (поля) и их надписи. Все нанесенные элементы управления отображаются в окне *Навигатора форм* (рис. 17). Количество



добавления форм неограниченно.

Рисунок 17. Навигатор форм, содержащий три формы

Далее следует в окне *Свойства формы* (для каждой из трех форм) указать *Тип содержимого источника* и *Содержимое*. В данном случае для

всех трех форм *Тип содержимого источника* один и тот же — таблица, а *Содержимое* — соответственно *Работники, Стеллажи, Документы.*

Теперь для каждого элемента формы (поля) в окне *Свойства* с включенной закладкой *Данные* следует указать *Поле данных.*

Между тремя формами, входящими в состав субформы, должна быть организована связь, иначе записи, сделанные в указанных полях не будут друг другу соответствовать. Для создания связи между формами необходимо относительно подчиненной формы вызвать окно *Свойства форм* и включить закладку *Данные.* В появившемся окне следует найти запись *Связь с главным полем* и нажать на кнопку вызова окна связей, показанную на рисунке 18.

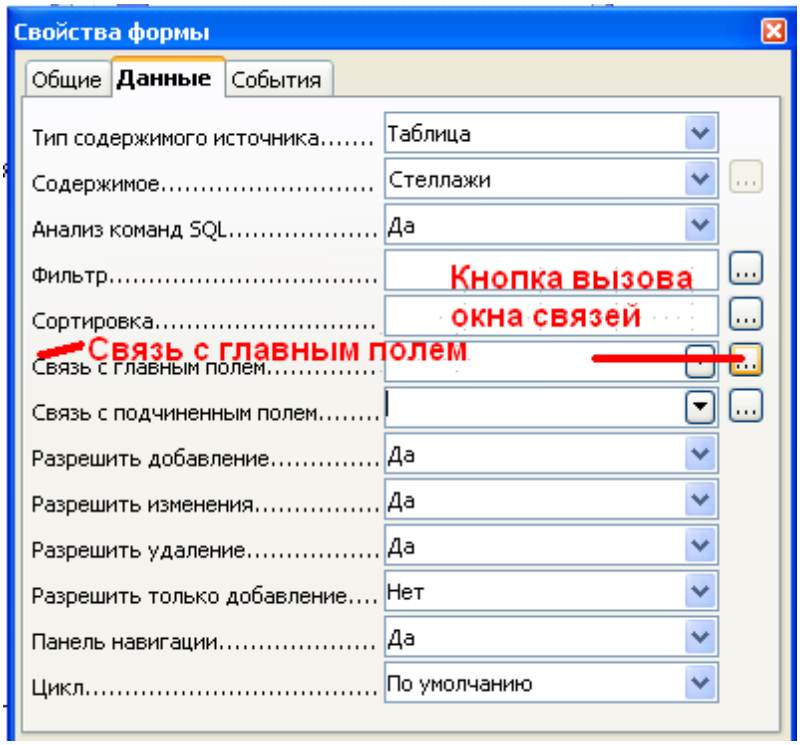


Рисунок 18. Установление связей между формами

После этого откроется окно, где по принципу установления связей между таблицами следует установить связь между формами (рис. 19).

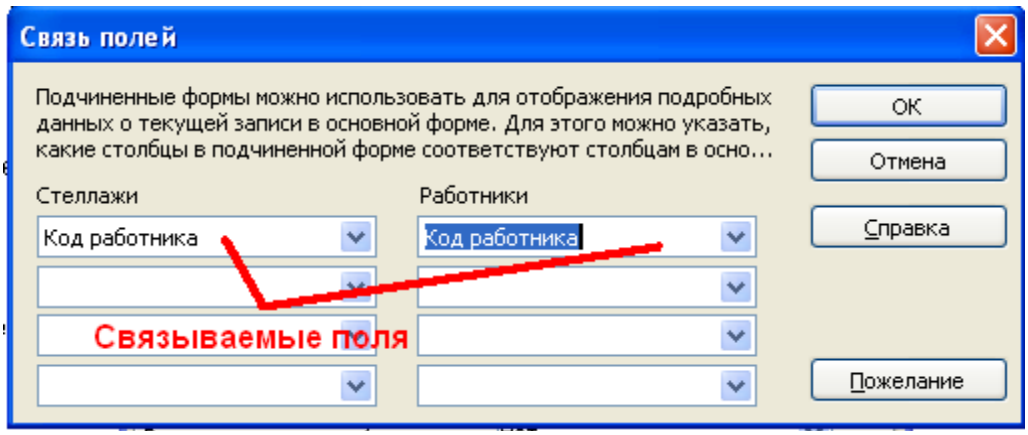


Рисунок 19. Связь полей форм.

После установления связей окно *Свойства форм* будут выглядеть следующим образом (рис. 20).

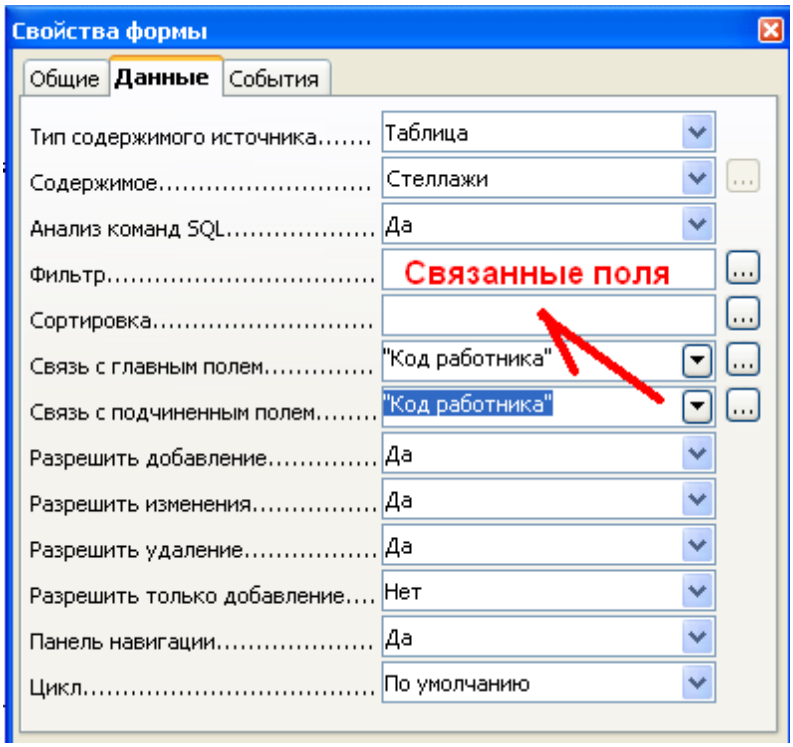


Рисунок 20. Окно *Свойства форм* после установления связей между формами

Такую же процедуру по установлению связей следует проделать и между остальными формами[8].

8 Тип связи между формами такой же, как и между соответствующими таблицами.

В создаваемую форму можно вставлять рисунки или картинки из галереи. Соответствующие кнопки представлены на рисунке 21.

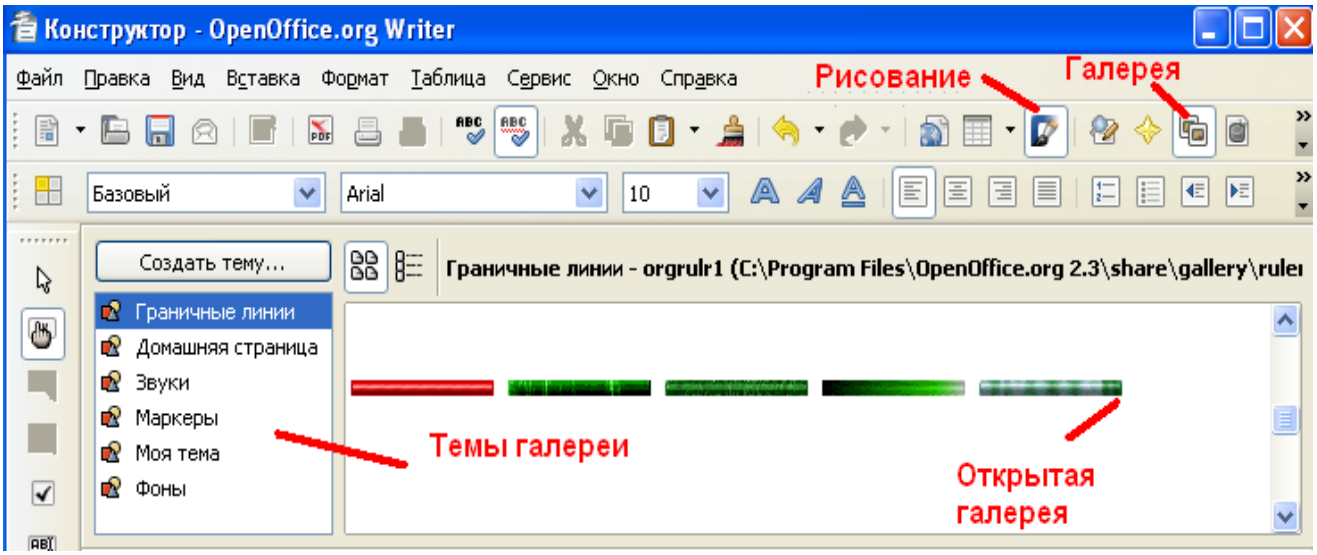


Рисунок 21. Оформление формы

2.3. Задание на работу

Создание базы данных «Архив»

- В *режиме мастера* создать произвольные формы для таблиц *Работники* и *Стеллажи*.
- В *режиме дизайна* создать произвольную форму для таблицы *Документы*.
- В *режиме мастера* создать субформу для таблиц *Посетители* и *Заказанные документы*. Основная форма — *Посетители,* вложенная — З*аказанные документы*.
- В *режиме дизайна* создать произвольную субформу, состоящую не менее чем из трех форм.
- Произвольно оформит форму рисунками и картинками.
- Заполнить таблицы произвольными данными (не менее 5 записей).

2.4. Требуемые результаты

В результате выполнения лабораторной работы должны получиться 5 формы: 3 простых и 2 сложных. Данные, вводимые в формы, должны отобразится в соответствующих таблицах.

На рисунке 21 показан пример субформы, состоящей из трех форм: *Работники, Стеллажи, Документы.*

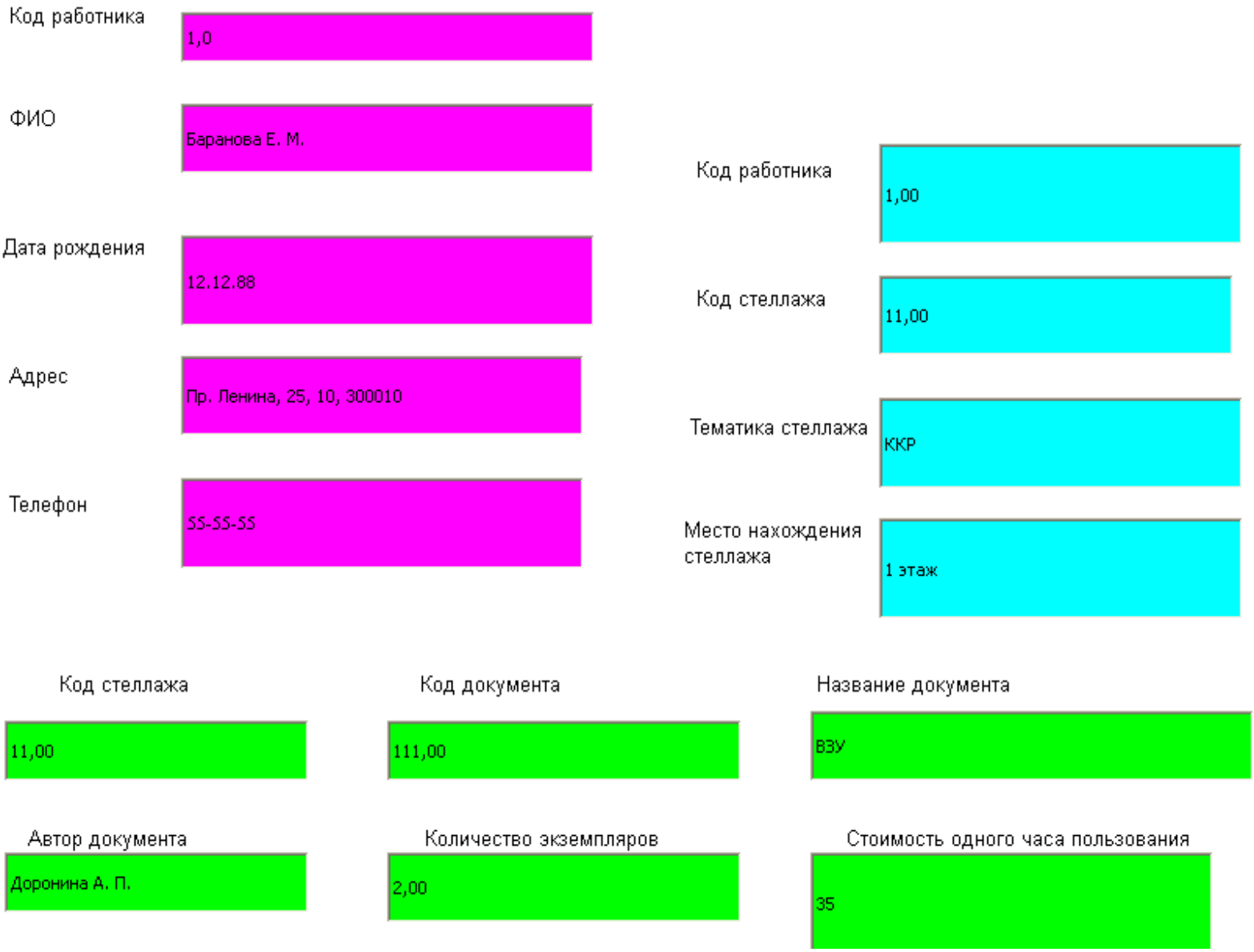


Рисунок 21. Субформа, состоящая из трех форм

2.5. Оформление отчета по работе

Отчет должен содержать:

1. Номер практической работы.

2. Название практической работы.

3. Цель практической работы.

4. Описание пунктов выполнения практической работы в соответствии с заданием на работу.

5. Вывод по работе.

2.6. Контрольные вопросы

13. Что такое формы относительно понятия СУБД?
14. Как создать форму в режиме мастера?
15. Как создать форму в режиме дизайна?
16. Что такое субформа?
17. Что такое основная форма?
18. Что такое вложенная форма?
19. Что такое установка источников данных?
20. Что значит расположить элементы управления?
21. Какие стили можно применить для создания форм?
22. Что такое *Навигатор форм*?
23. Чем окно *Свойства* отличается от окна *Свойства формы*?
24. Как установить связь между формами субформы?
25. Что такое тип содержимого источника? Содержимое? Поле данных?
26. Как установить цвет и обрамление поля при создании формы в режиме дизайна?
27. Каким образом можно оформить форму?

Практическая работа № 3

Создание запросов в базе данных Base Open Office.

3.1. Цель работы

Приобретение навыков по созданию запросов

в базе данных Base Open Office

Теоретические положения

Если в процессе работы с базой данных необходимо обратиться только к подмножеству хранимых в ней данных, рекомендуется составить *запрос*. Запрашиваемые данные должны отделяться от совокупности данных хорошо сформулированным условием, которое называется *условием фильтра*. *Запрос* - это новое представление отфильтрованных данных. Открыв запрос, можно увидеть текущие данные в виде определенной таблицы.

Для создания запросов также можно использовать *режим мастера* или *режим дизайна*.

Создание запросов в режиме мастера

Запрос на выборку. Запрос с условием. Итоговый запрос

Предположим, заведующей архива требуется список дней рождения работников. Данные о работниках хранятся в таблице *Работники*, однако, помимо требуемой заведующей информации, в этой таблицы хранятся и другие данные, являющиеся в данной ситуации лишними. Таким образом, требуется выбрать из таблицы *Работники* данные, необходимые в текущий момент времени. Такой запрос называется *запросом на выборку*.

Режим мастера для создания запросов аналогичен аналогичен этому же режиму при создании форм.

После выбора *режима мастера* откроется подобное при создании форм окно (рис. 1), в котором необходимо указать таблицу, на данных которой будет сформирован запрос, а также поля, непосредственно составляющие суть запроса.

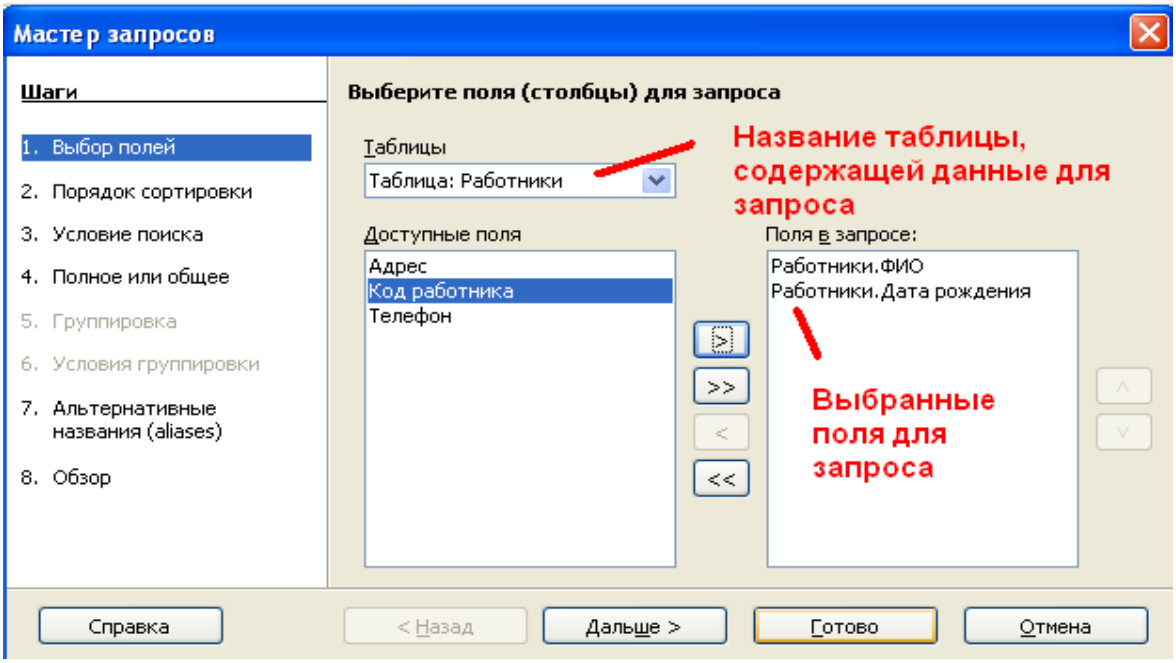


Рисунок 1. Формирование запроса в *режиме мастера*

После выбора таблицы и соответствующих полей необходимо нажать кнопку *Дальше.* Появится окно, в котором можно осуществить сортировку полей. Например, поле *Дата рождения* целесообразно отсортировать по возрастанию (с начала по конец года). Текстовое поле *ФИО* подвергать сортировке не следует (рис. 2)[9].

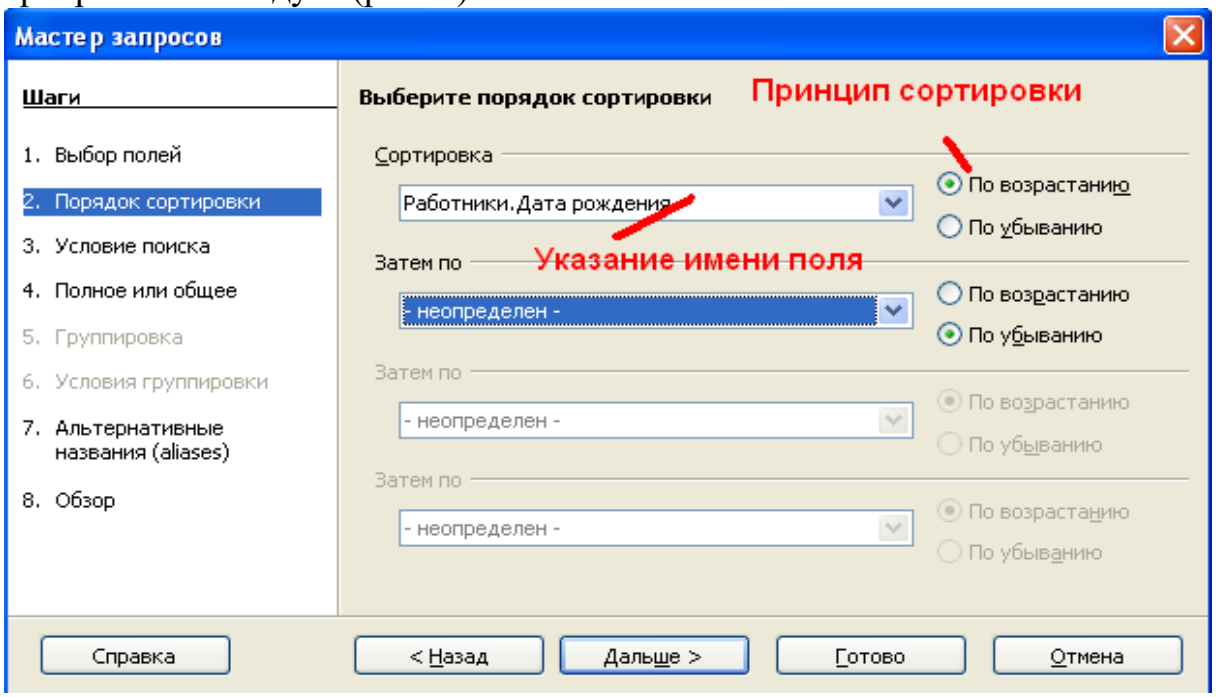


Рисунок 2. Сортировка полей запроса

После следующего нажатия кнопки *Дальше* появится окно, в котором при желании можно указать *условие поиска (отбора информации)*.

Например, та же самая заведующая архивом просит не полный список дат рождения своих работников, а список работников, рожденных до какой-

9 Текстовые поля сортируются от А до Я или наоборот.

то определенной даты. В этом случае в появившемся окне необходимо указать условие отбора, показанное на рисунке 3.

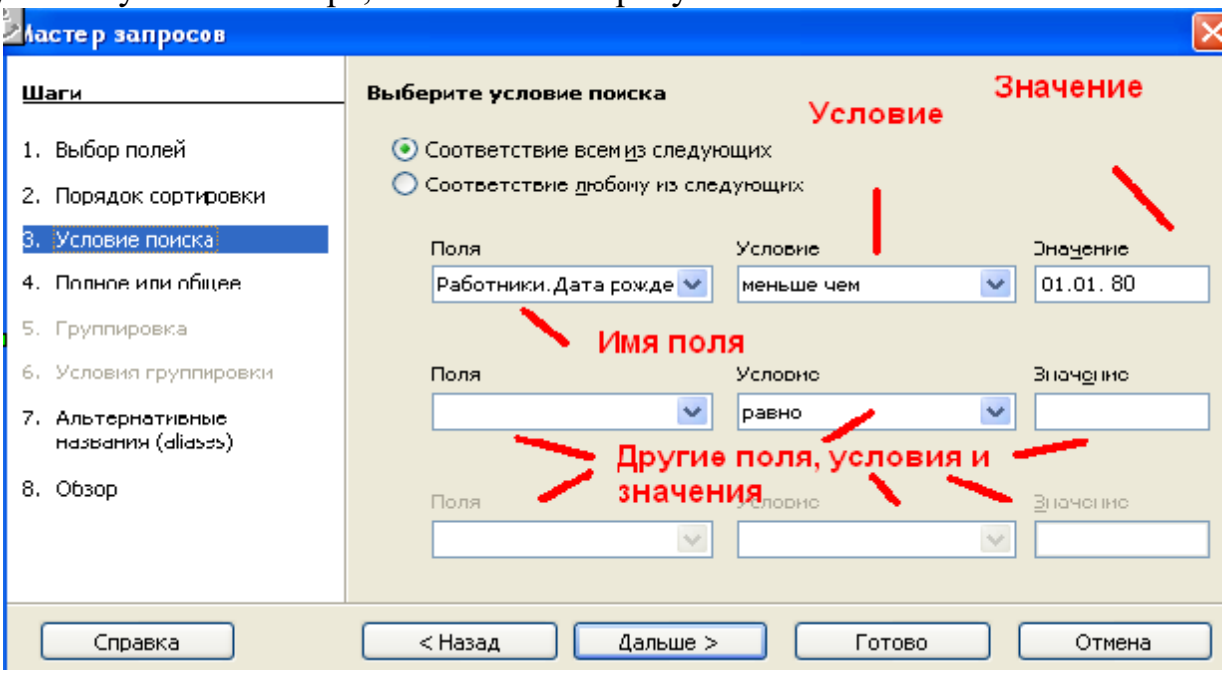


Рисунок 3. Формирование условия поиска (отбора информации)

После нажатия кнопки *Дальше* появится окно, в котором можно указать нужен ли пользователю *Детальный запрос* или требуется более общий *Итоговый запрос*. Итоговый запрос показывает только данные агрегатных функций (сумму, среднее, минимум, максимум). При формировании итогового запроса необходимо указать агрегатную функцию и поле, к которой эту функцию следует применить. Поля итогового запроса, к которым будут применены агрегатный функции, можно добавлять или удалять посредством кнопок, показанных на рисунке 4.

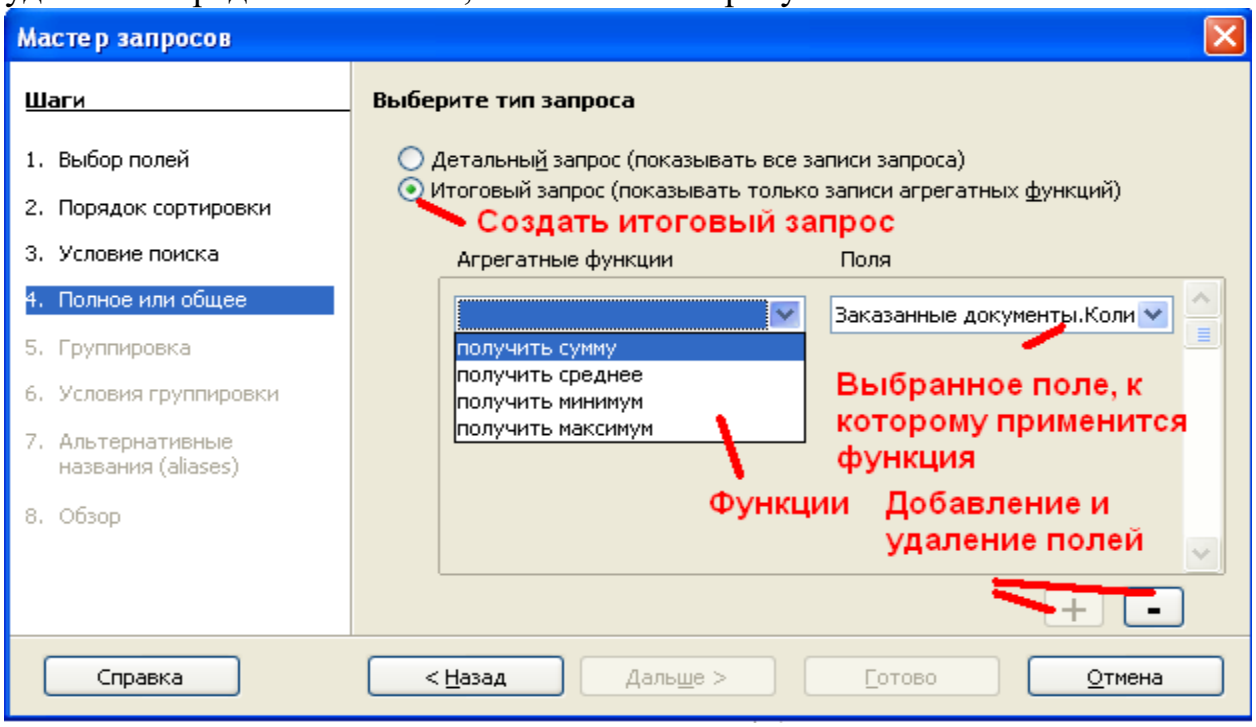


Рисунок 4. Формирование итогового запроса

При формировании итогового запроса поля, к которым не применяются агрегатные функции, показаны не будут.

Следующее окно, открывающееся после нажатия кнопки *Дальше*, позволяет присвоить полям, формирующим запрос, какое-либо альтернативное (любое другое) название (рис. 5). Это необходимо для быстрого реагирования на несомую запросом информацию.

В следующем окне, открываемом посредством нажатия кнопки *Дальше*, необходимо провести проверку (обзор) данных создаваемого запроса. В случае необходимости внесения изменений можно воспользоваться кнопкой *Назад*. В случае правильности исходных данных запроса следует нажать кнопку *Готово* (рис. 7).

Открытый *запрос на выборку* показан на рисунке 8.

Открытый *запрос на выборку с условием отбора* представлен на рисунке 9.

Открытый итоговый запрос показан на рисунке 10.

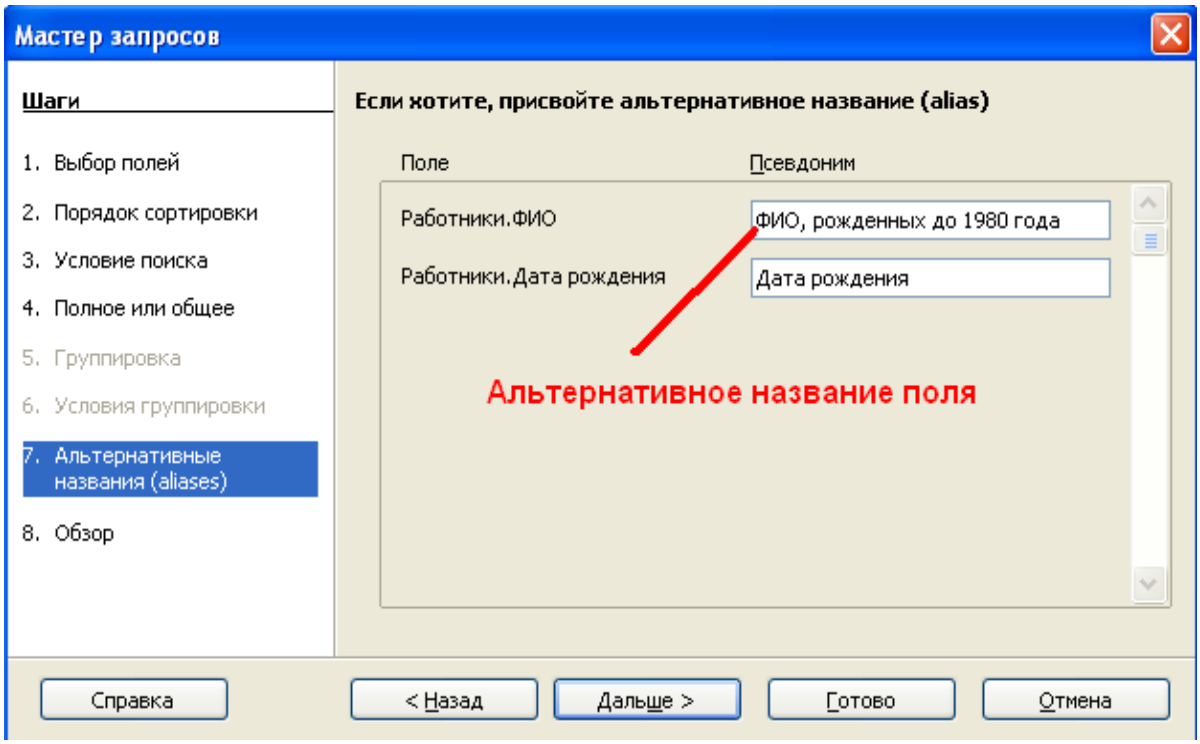


Рисунок 6. Присваивание полям альтернативных имен

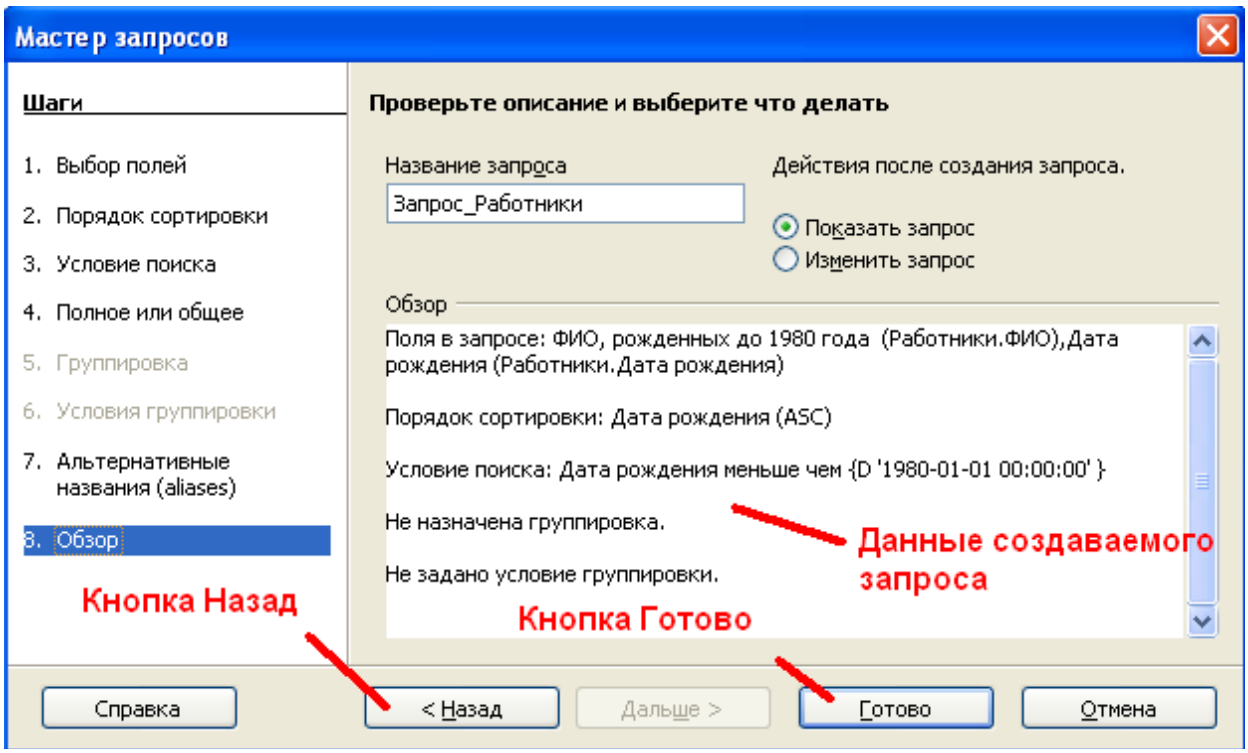


Рисунок 7. Обзор данных создаваемого запроса

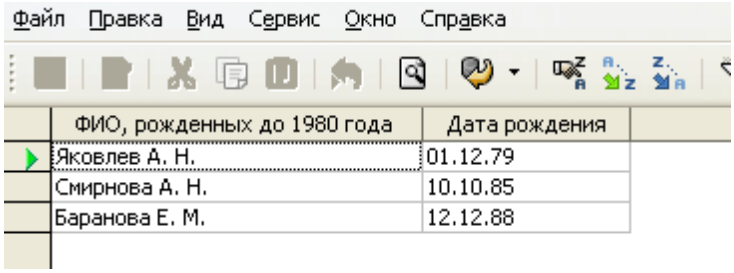


Рисунок 8. Открытый запрос на выборку

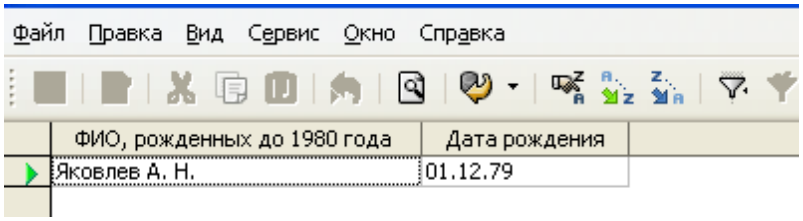


Рисунок 9. Открытый запрос на выборку с условием отбора

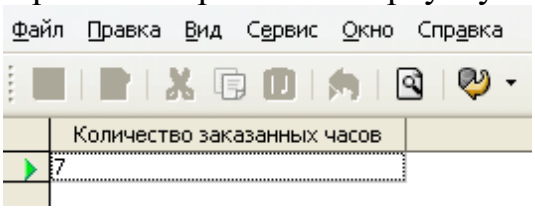


Рисунок 10. Открытый итоговый запрос

Создание запросов в режиме дизайна
Запрос на выборку. Запрос с условием. Итоговый запрос.

Предположим, необходимо создать запрос в *режиме дизайна*, также как в предыдущем примере (с использованием *режима мастера*), отфильтровывающий информацию по датам дней рождения работников.

После выбора *режима дизайна* откроется окно, где необходимо будет добавить таблицу, на основании данных которой требуется сформировать запрос (рис. 11).

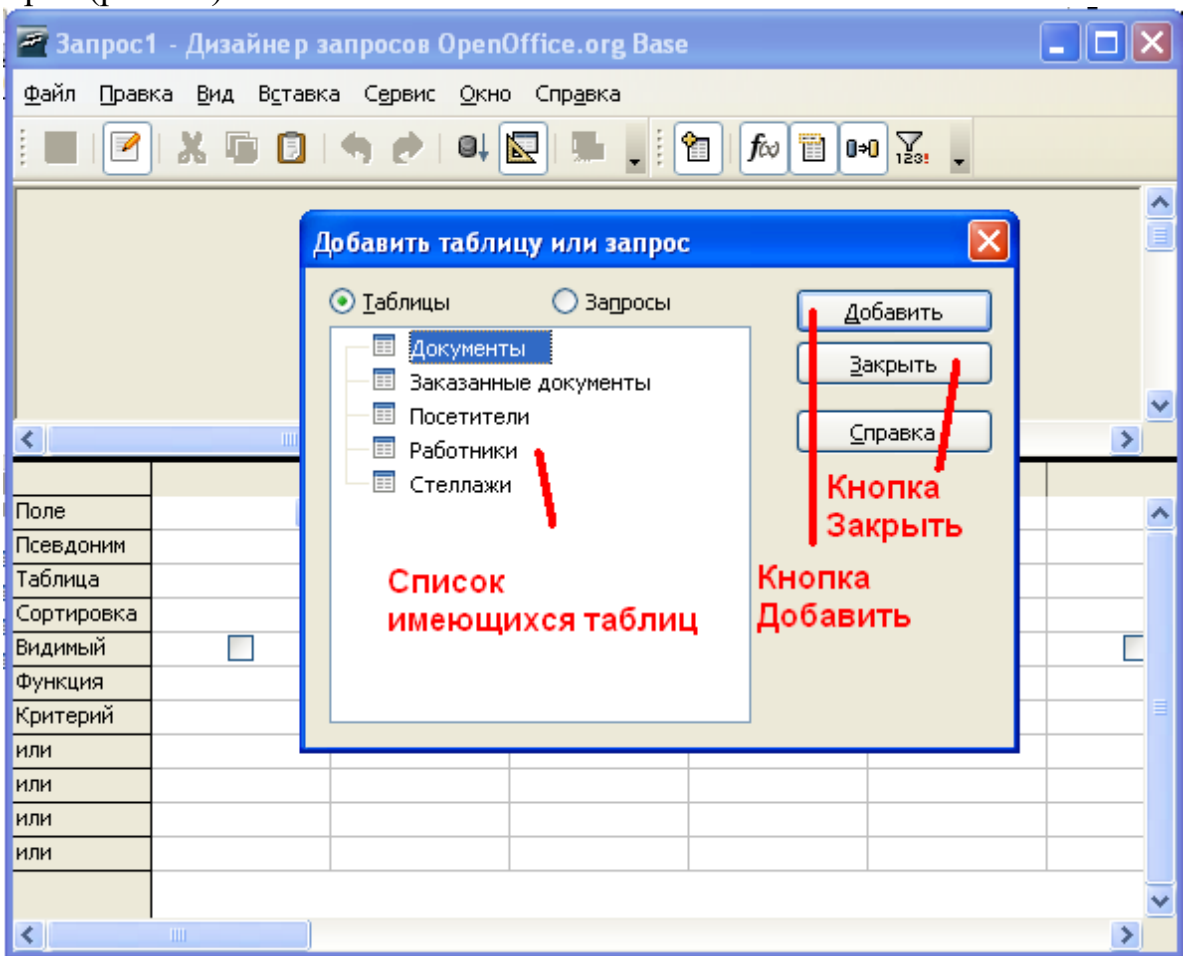


Рисунок 11. Добавление таблицы при создание запроса на выборку в режиме дизайна

После добавления таблицы окно *Добавить таблицу или запрос* необходимо закрыть.

Далее необходимо указать участвующие в формировании запроса поля. Для этого в нижней части окна *Дизайнер запросов* в графе *Поле* следует выбрать из предложенного списка необходимые поля. Список соответствует имеющимся в добавленной таблице полям. В графе *Сортировка* следует указать принцип сортировки (по возрастанию или по убыванию).Здесь же в графе *Псевдоним* можно указывать альтернативное имя поля, указывать видимость или невидимость выбранного поля, функции, критерий отбора и дополнительные критерии фильтрации (рис. 12).

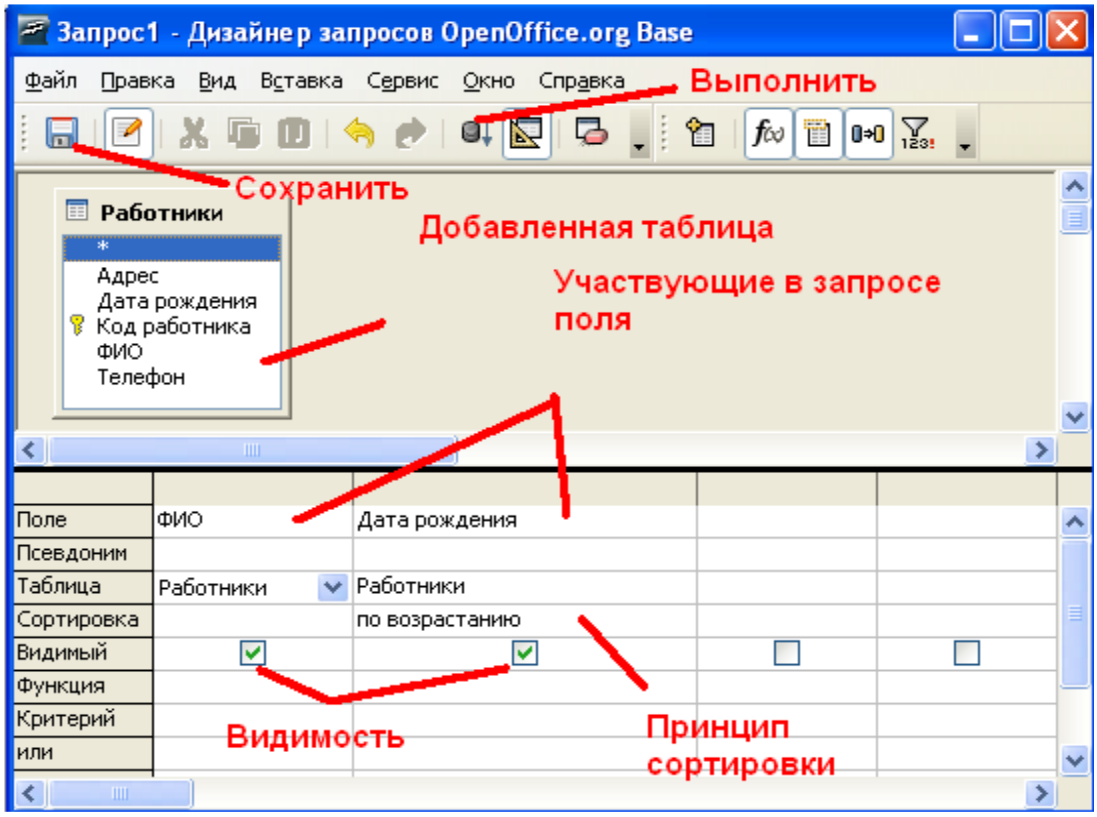


Рисунок 12. Создание запроса на выборку в режиме дизайна

После этого запрос следует сохранить и закрыть или выполнить (рис. 12).

Чтобы указать условие поиска (отбора информации) необходимо в графе *Критерий* указать условие фильтра (рис. 13).

К основному критерию могут добавляться дополнительные условия отбора информации, которые при выполнении запроса будут соединяться логическим неразделительным ИЛИ. В конструкторе запросов это дополнительные графы *или* (рис. 14).

Чтобы в режиме дизайна создать *итоговый запрос* необходимо в графе Функции указать агрегатную функцию. Например, пользователя интересует информация о сумме заказанных часов пользования документами архива. Соответствующий запрос представлен на рисунке 15.

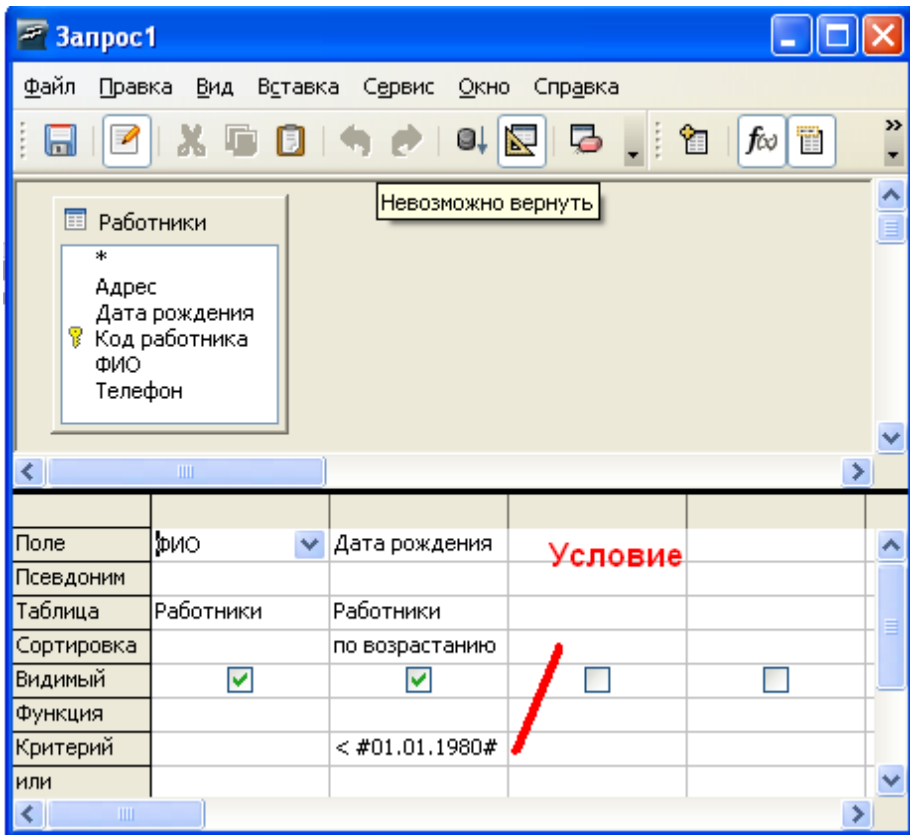


Рисунок 13. Создание запроса на выборку с условием в режиме дизайна

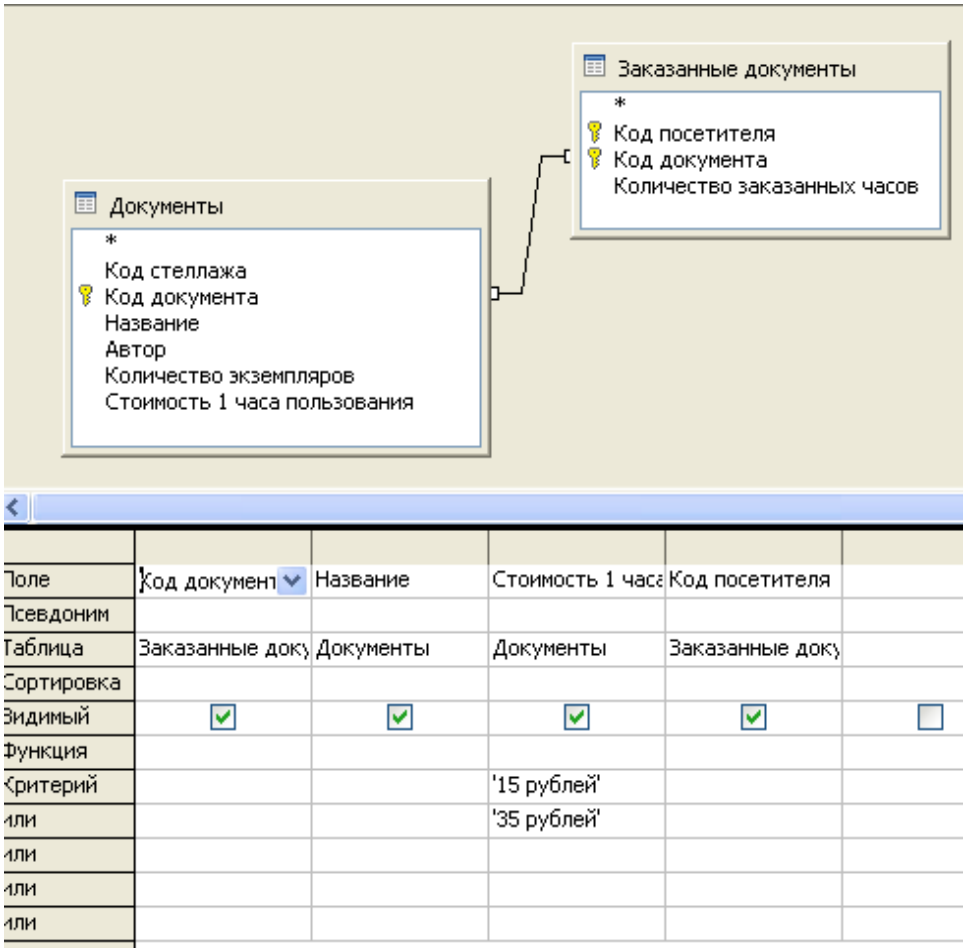


Рисунок 14. Запрос с дополнительными критериями отбора

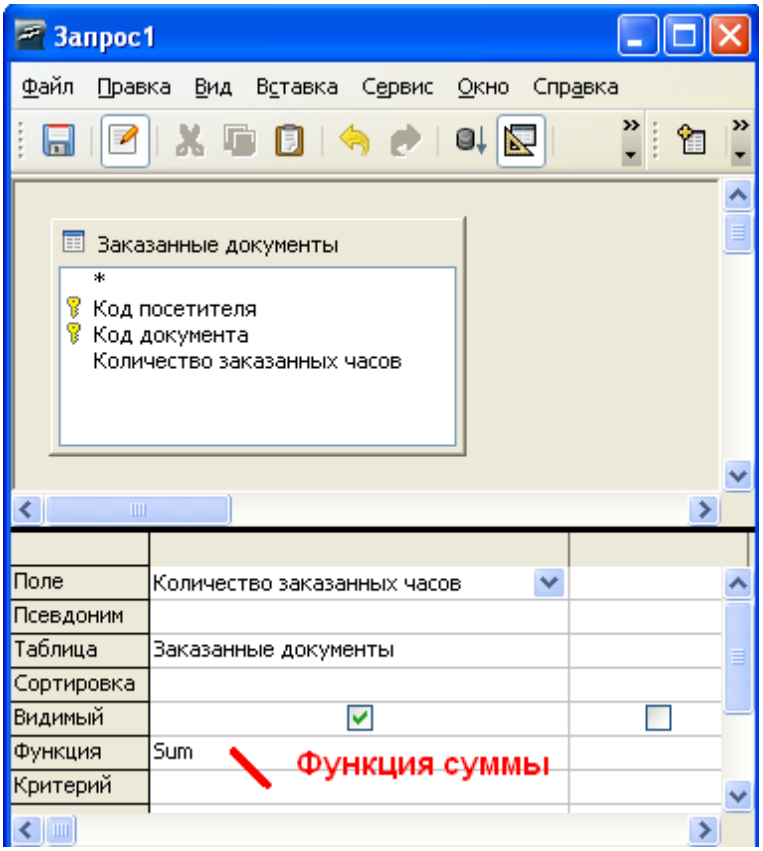


Рисунок 15. Создание итогового запроса в режиме дизайна

Открытые запросы аналогичны представленным на рисунках 8-10.

Перекрестный запрос. Запрос с параметром. Запрос с вычислением

Очень часто требуется отфильтровать информацию базы данных, хранящуюся в разных таблицах, таблицах или запросах, а также разных запросах. Запрос, поля которого взяты из разных источников хранения данных, называется перекрестным.

Перекрестный запрос создается путем добавления нескольких таблиц или запросов. Несвязанные источники данных (таблицы или запросы) необходимо связать по принципу создания связей между таблицами.

Например, требуется отфильтровать информацию о том, какими работниками архива обслужены посетители, причем эта информация требуется исключительно по тематике стеллажа ВКР. Соответствующий запрос, создаваемый в режиме дизайна, представлен на рисунке 16.На рисунке 17 представлен открытый перекрестный запрос[10].

Иногда требуется составить универсальный *запрос с переменным параметром*. Чтобы создать запрос с переменными параметрами, необходимо в графе *Критерий* использовать знак равенства и двоеточие, т. е. ввести записать =:x. При выполнении запроса программа открывает

10 Перекрестный запрос с использованием полей связанных таблиц и запросов в методических указаниях не представлен в силу специфичности предметной области базы данных «Архив».

диалоговое окно для ввода выражения, которому будет присвоена переменная x[11].

На рисунке 18 показан запрос с параметром в режиме дизайна. На рисунке 18 представлен запуск запроса с параметром.

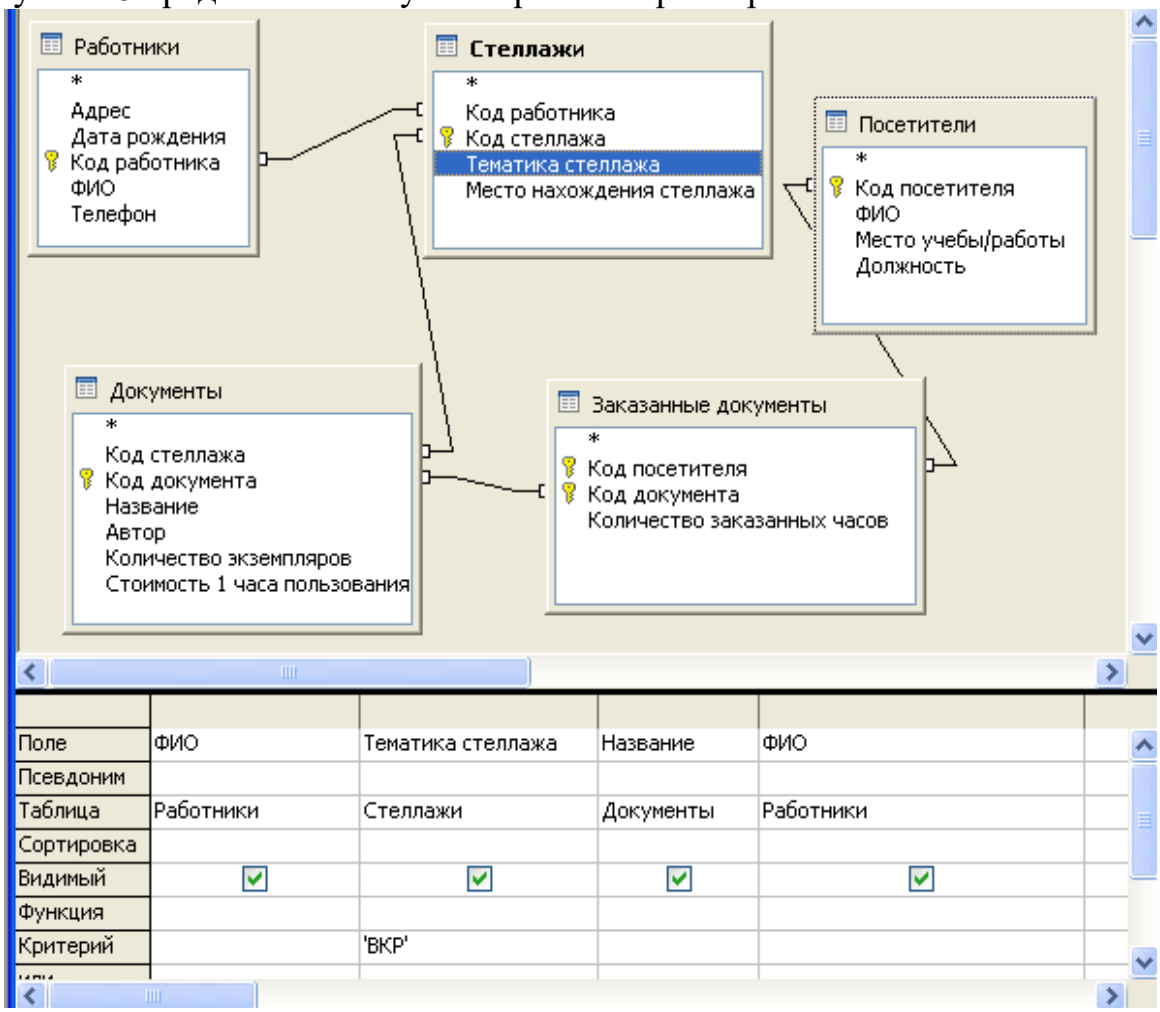


Рисунок 16. Перекрестный запрос с условием отбора

	ФИО	Тематика стеллажа	Название	ФИО1
▶	Смирнова А. Н.	ВКР	ЭВМ	Иванов А. Д.
	Смирнова А. Н.	ВКР	Экспертные системы и общество	Ковалева П. Л.

Рисунок 17. Открытый перекрестный запрос с условием отбора

11 На месте переменной x может быть использована любая другая переменная.

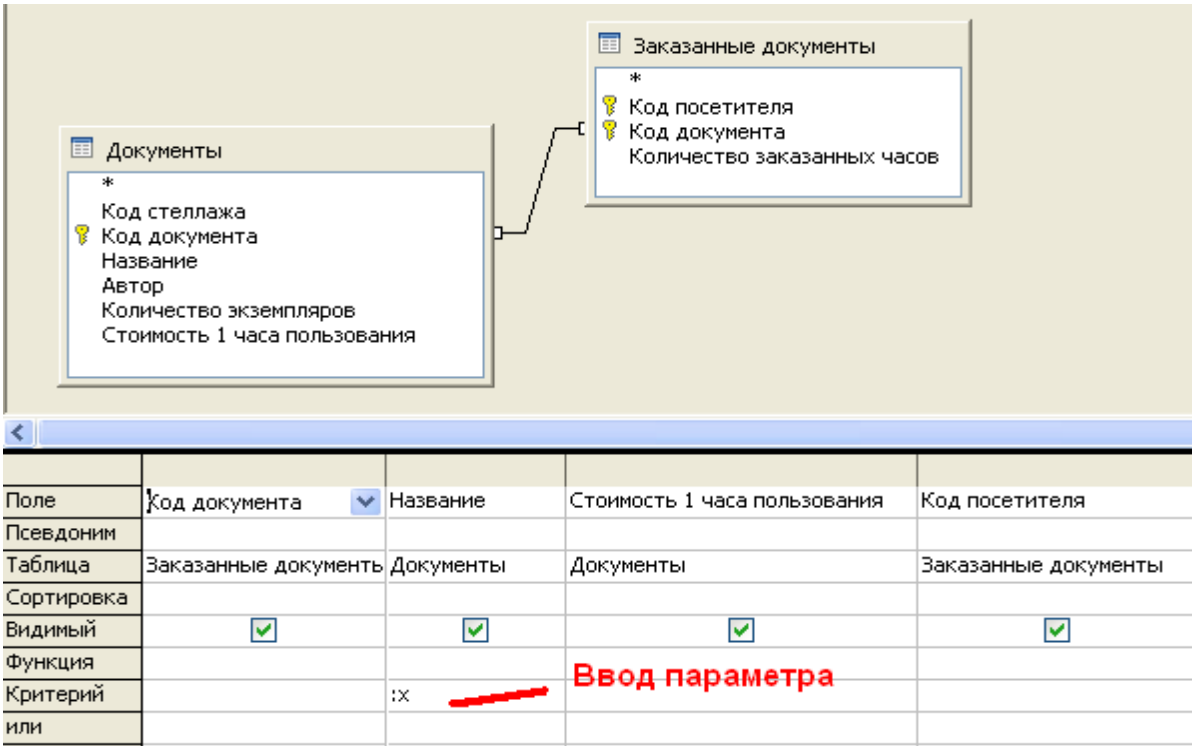


Рисунок 18. Запрос с параметром в режиме дизайна

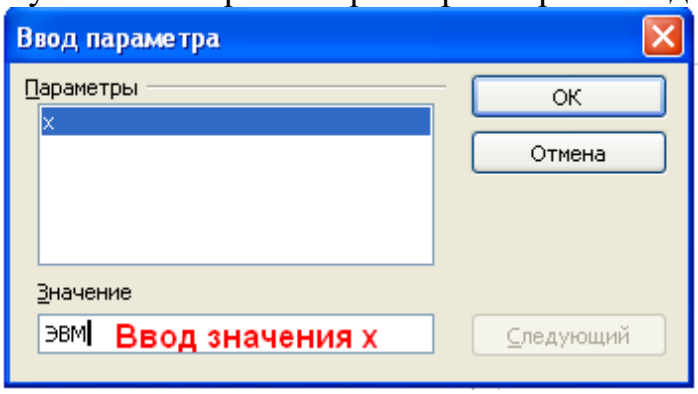


Рисунок 19. Запуск запроса с параметром

Представленный на рисунке 19 запрос будет выдавать только те записи, в которых название документа соответствует записи «ЭВМ».

Открытый запрос с параметром, соответствующим записи ЭВМ, представлен на рисунке 20.

Код документа	Название	Стоимость 1 часа пользования	Код посетителя
212	ЭВМ	35 рублей	100000

Рисунок 20. Открытый запрос с параметром

Запрос с вычислением предполагает ввод какого-либо выражения, например, складывающего или перемножающего некоторые поля из таблиц или других запросов.

Предположим, необходимо получить запрос, показывающий какая сумма денег выручена от пользования заказанными документами. Запрос «Сумма денег» должен иметь следующие поля: *Код документа, Стоимость 1 часа пользования, Количество заказанных часов, Сумма*. Информация

должна быть отсортирована по убыванию. Соответствующий запрос, сформированный с использованием *режима дизайна*, показан на рисунке 20.

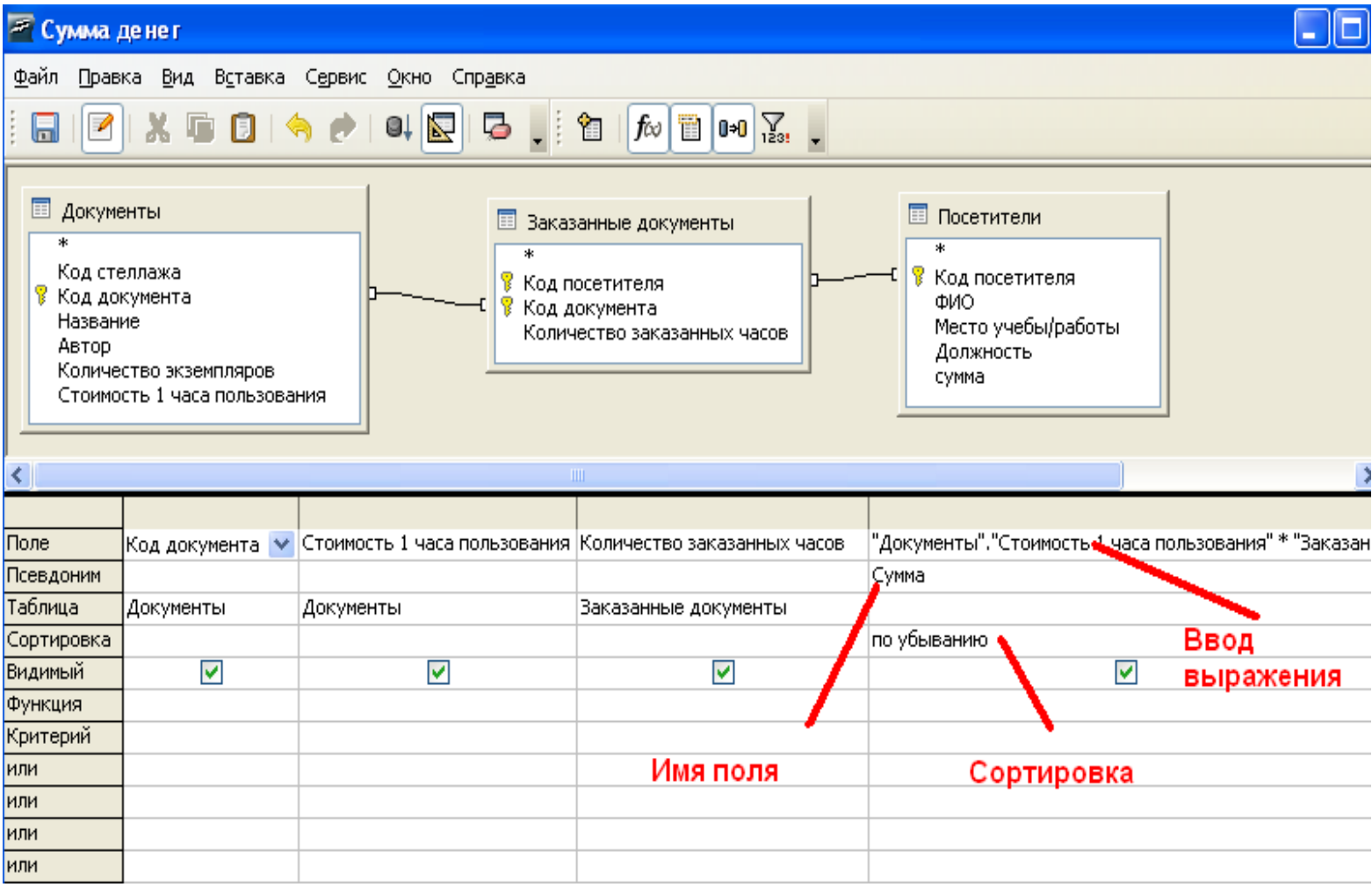


Рисунок 21. Запрос с расчетом

На рисунке 22 показано вводимое в запрос «Сумма денег» выражение.

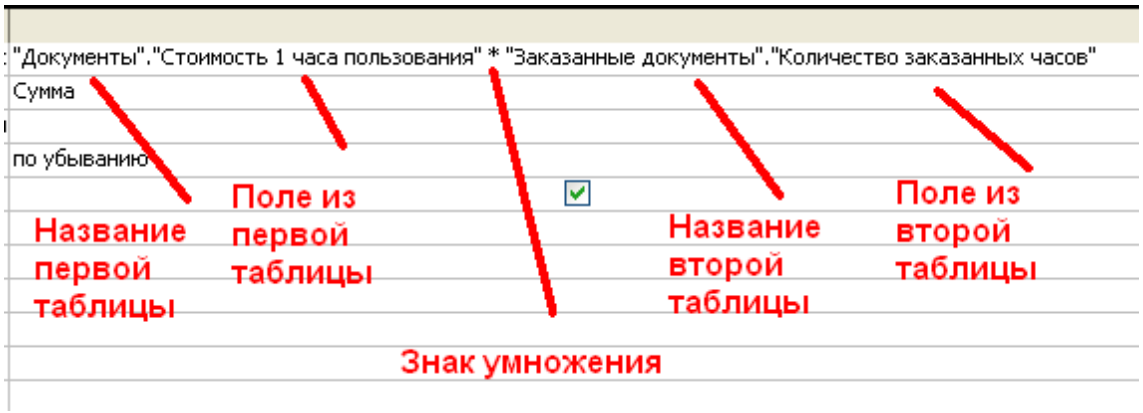


Рисунок 22. Ввод выражения в запрос

Выполненный запрос с расчетом «Сумма денег» представлен на рисунке 23.

	Код документа	Стоимость 1 часа пользования	Количество заказанных часов	Сумма
▶	111	35	2	70
	211	30	2	60
	311	45	1	45
	112	25	1	25
	212	15	1	15

Подсчитанная сумма денег, отсортированная по убыванию

Рисунок 23. Выполненный запрос «Сумма денег»

3.3. Задание на работу

Создание базы данных «Архив»

1. В *режиме мастера* создать запрос на выборку, содержащий информацию о номерах телефонов работников архива. Работников архива расположить в алфавитном порядке.
2. В *режиме мастера* создать произвольный запрос на выборку с произвольным критерием отбора.
3. В *режиме мастера* создать произвольный итоговый запрос.
4. В *режиме дизайна* создать произвольный перекрестный запрос.
5. В *режиме дизайна* создать произвольный запрос с параметром.
6. В *режиме дизайна* создать произвольный запрос с условием по дате.
7. В *режиме дизайна* создать запрос с вычислением (вводом выражения).

3.4. Требуемые результаты

В результате выполнения лабораторной работы должны получиться 7 запросов. Каждому запросу должно быть присвоено уникальное имя, отражающее его суть.

3.5. Оформление отчета по работе

Отчет должен содержать:

1. Номер практической работы.

2. Название практической работы.

3. Цель практической работы.

4. Описание пунктов выполнения практической работы в соответствии с заданием на работу.

5. Вывод по работе.

3.6. Контрольные вопросы

1. Какие функции выполняют запросы?
2. Как создать запрос в режиме мастера?
3. Какие запросы можно реализовать в режиме мастера?
4. Как построить запрос в режиме дизайна?
5. Какие запросы можно реализовать в режиме дизайна?
6. Что такое запрос на выборку?
7. Что такое критерий поиска (отбора)?
8. Что такое итоговый запрос?
9. Что такое запрос с параметром?
10. Что такое перекрестный запрос?
11. Как создать запрос с вычислением (вводом выражения)?

Практическая работа № 4

Создание отчетов в базе данных Base Open Office.

4.1. Цель работы

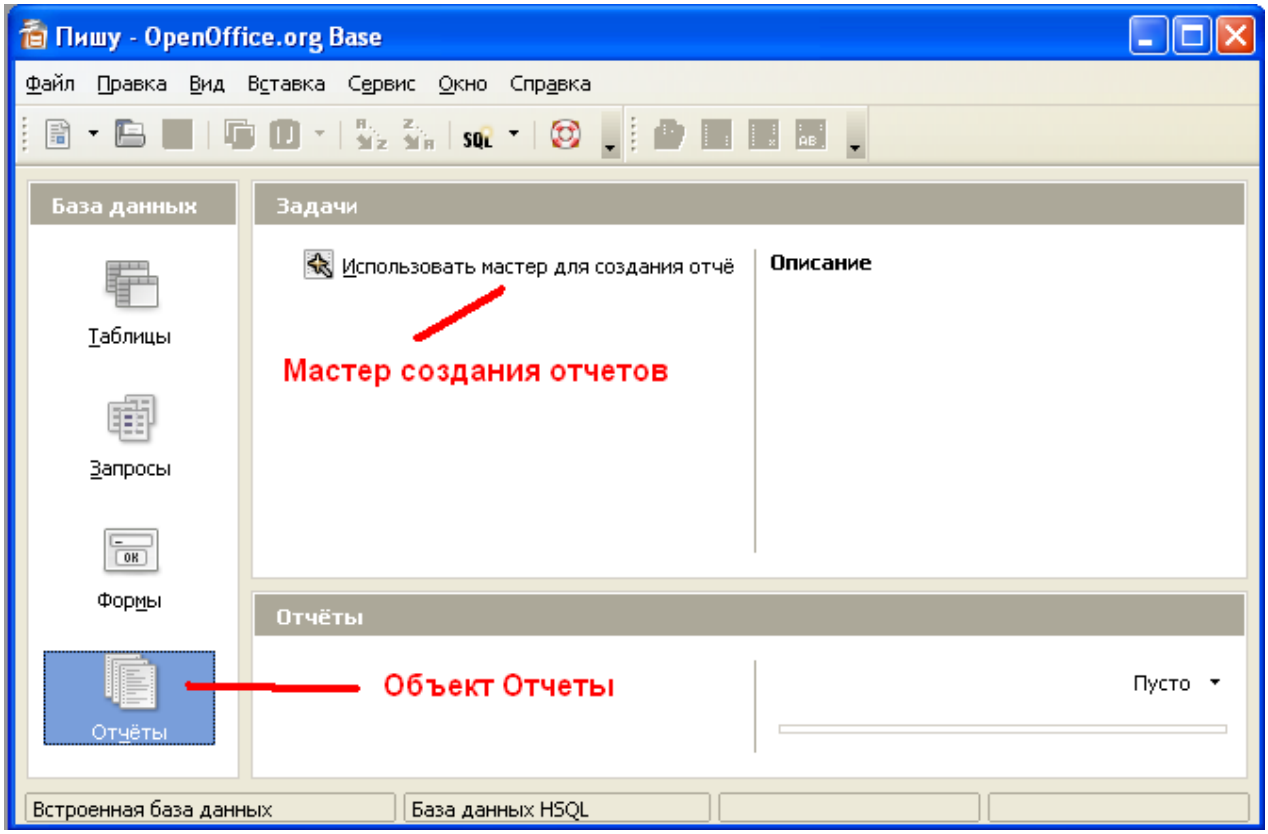
Приобретение навыков по созданию отчетов

в базе данных Base Open Office

Теоретические положения

Отчет - это текстовый документ, отображающий текущие данные в удобном для печати виде.

Чтобы создать отчет необходимо в окне проектирования базы данных выбрать объект *Отчеты* (рис. 1). Как видно из рисунка 1 отчеты



в Base Open Office можно создавать только в *режиме мастера*.

Рисунок 1. Создание отчетов в Base Open Office

После выбора для создания отчетов режима *мастера* откроется макет отчета в виде листа бумаги, на котором указаны название отчета, автор, дата, а также номера входящих в отчет страниц (рис. 2).

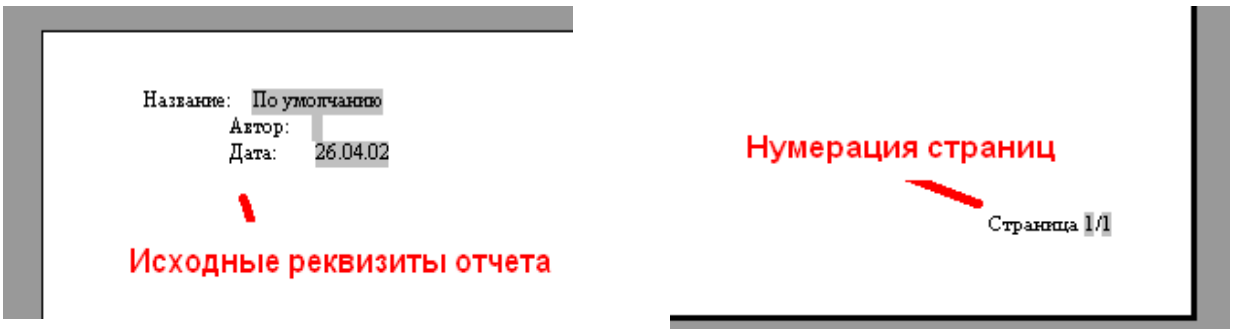


Рисунок 2. Фрагменты макета отчета.

Правый верхний и левый нижний углы

Поверх макета отчета откроется окно (рис. 3), в котором необходимо указать название таблиц или запросов[12], на данных которых будет создан отчет. Предположим, пользователю необходим отчет о датах дней рождения сотрудников архива (рис. 3).

Поля выбираются аналогично выбору полей при создании таблиц, форм и запросов в режиме *мастера*.

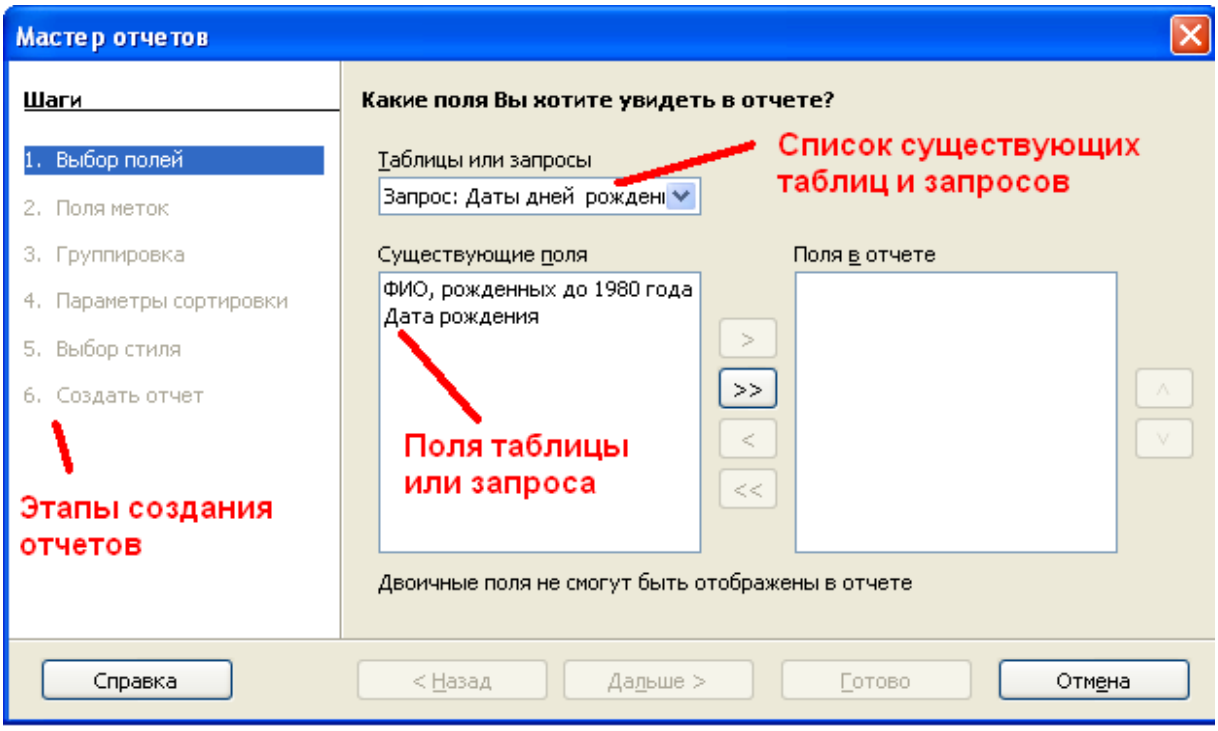


Рисунок 3. Создание отчетов в режиме мастера

После нажатия кнопки *Дальше* откроется окно, в котором по желанию можно указать выбранным полям метки, т. е. указать то имя поля, которое будет непосредственно прописано в отчете (рис. 4). Например, для поля *Дата рождения* введем метку *Дата рождения (не позднее 1980 года)*.

12 Отчеты, как правило, создаются на основании запросов. Однако возможны отчеты, созданные на основании таблиц.

После нажатия кнопки *Дальше* появится окно, в котором можно произвести группировку полей отчета. Для этого необходимо выделить поле и нажать кнопку, указанную на рисунке 5. Выбранное поле отобразится в части окна *Группировка и* отобразится на макете отчета (рис. 6).

На рисунке 6 вместо фиктивной надписи *x* будет расположена соответствующая запись поля, о чем в окне, представленном на рисунке 5, существует предупреждение.

Группировать можно произвольное количество полей отчета.

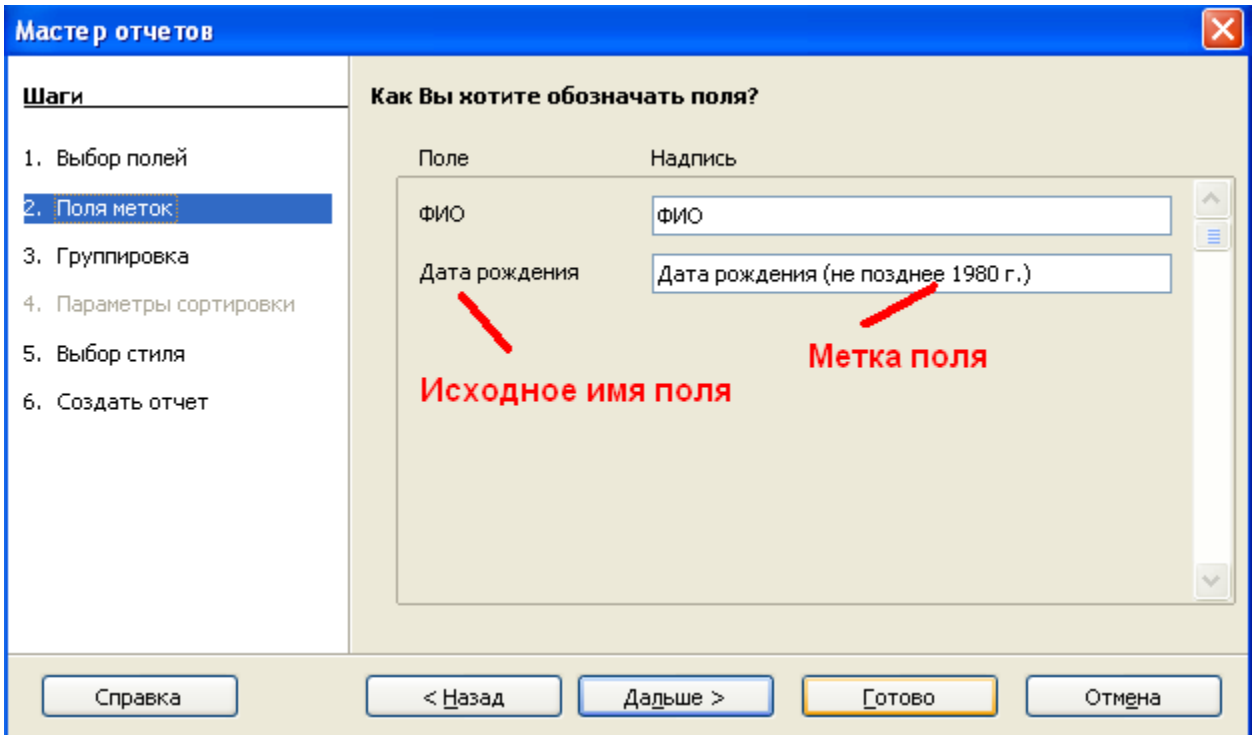


Рисунок 4. Этап указания меток полей при создании отчетов

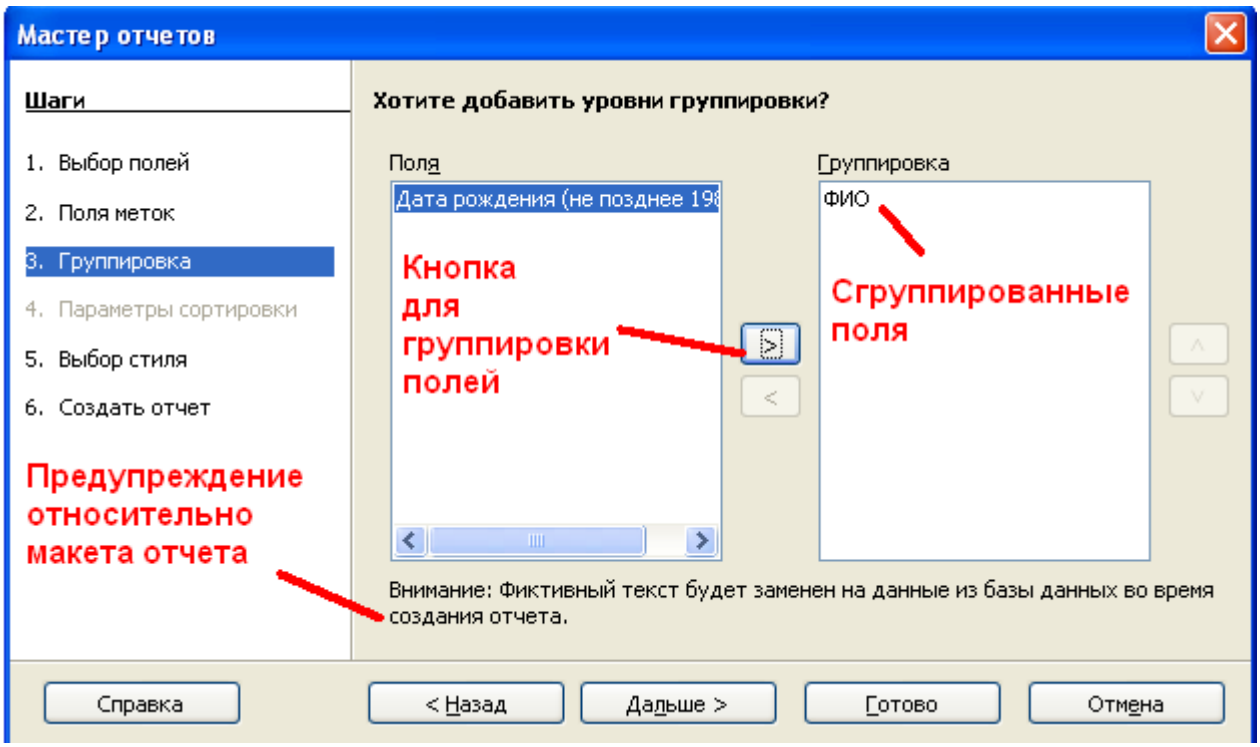


Рисунок 5. Группировка полей при создании отчетов

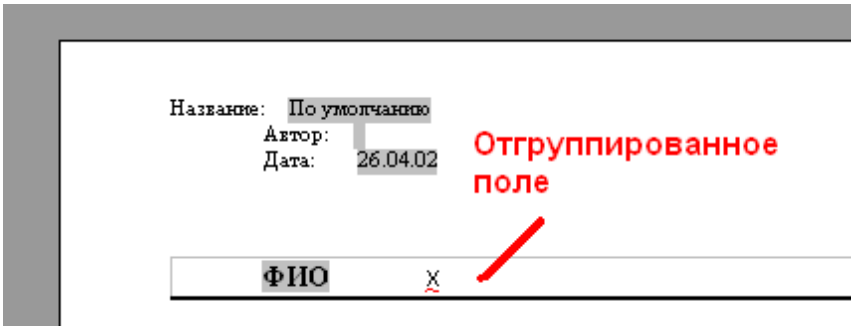


Рисунок 6. Фрагмент макета отчета после группировки полей

В окне *Параметры сортировки* пользователь указывает подлежащие упорядочиванию поля и условия сортировки таким же образом, как при создании запросов[13].

После нажатия кнопки *Дальше* появится окно, в котором можно по желанию пользователя установить внешний вид отчета, а именно

- разметку данных (расположение полей и записей отчета в соответствии с предложенными шаблонами);
- разметку верхнего и нижнего колонтитула (разметка шаблонного рисунка, отражающего содержательность отчета);
- ориентацию страницы (книжная или альбомная).

13 В данном отчете сортировка невозможна, т. к. в базе данных имеется всего один человек с датой рождения до 1980 года. Поэтому этап Параметры сортировки недоступен.

Изменения при выборе внешнего вида отчета отражаются на его макете.

На рисунке 7 показано окно *Выбор стиля отчета,* в котором указаны следующие элементы внешнего вида отчета:

- разметка данных - *Структурированный с отступом;*
- разметку верхнего и нижнего колонтитула - *Обычный;*
- ориентация страницы - *Книжная.*

На рисунке 8 показан соответствующий макет отчета.

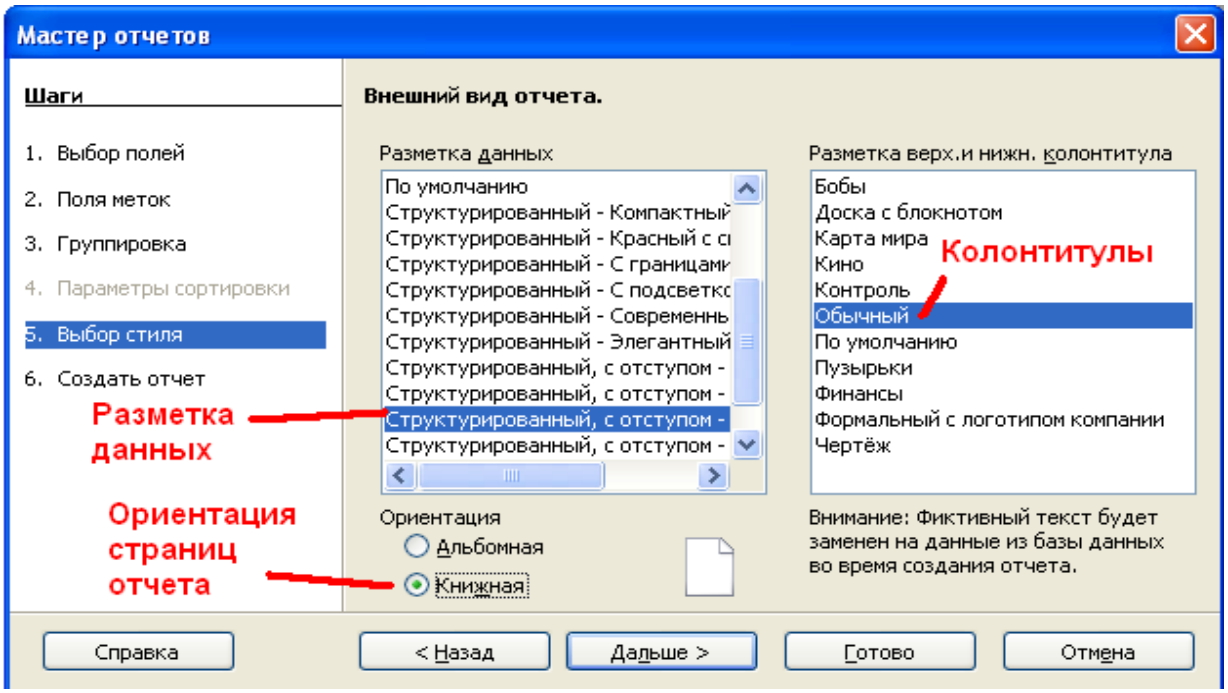


Рисунок 7. Выбор стиля отчета

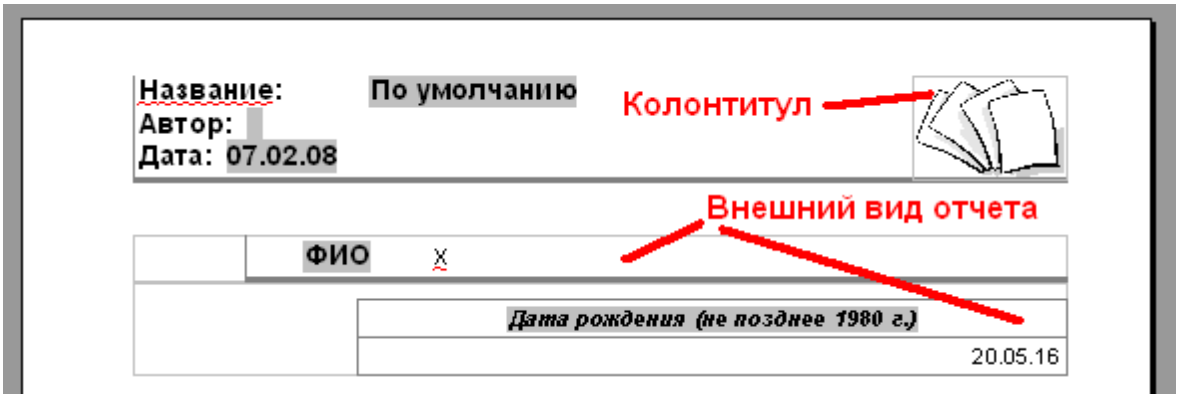


Рисунок 8. Фрагмент макета отчета

В следующем окне, открывающемся после нажатия кнопки *Дальше*, предлагается указать

28. заголовок отчета;
29. тип отчета (статический или динамический);
30. дальнейшие действия после завершения создания отчета (модифицировать шаблон отчета или непосредственно создать отчет). Выбор этих действий

доступен в случае указания динамического типа отчета.

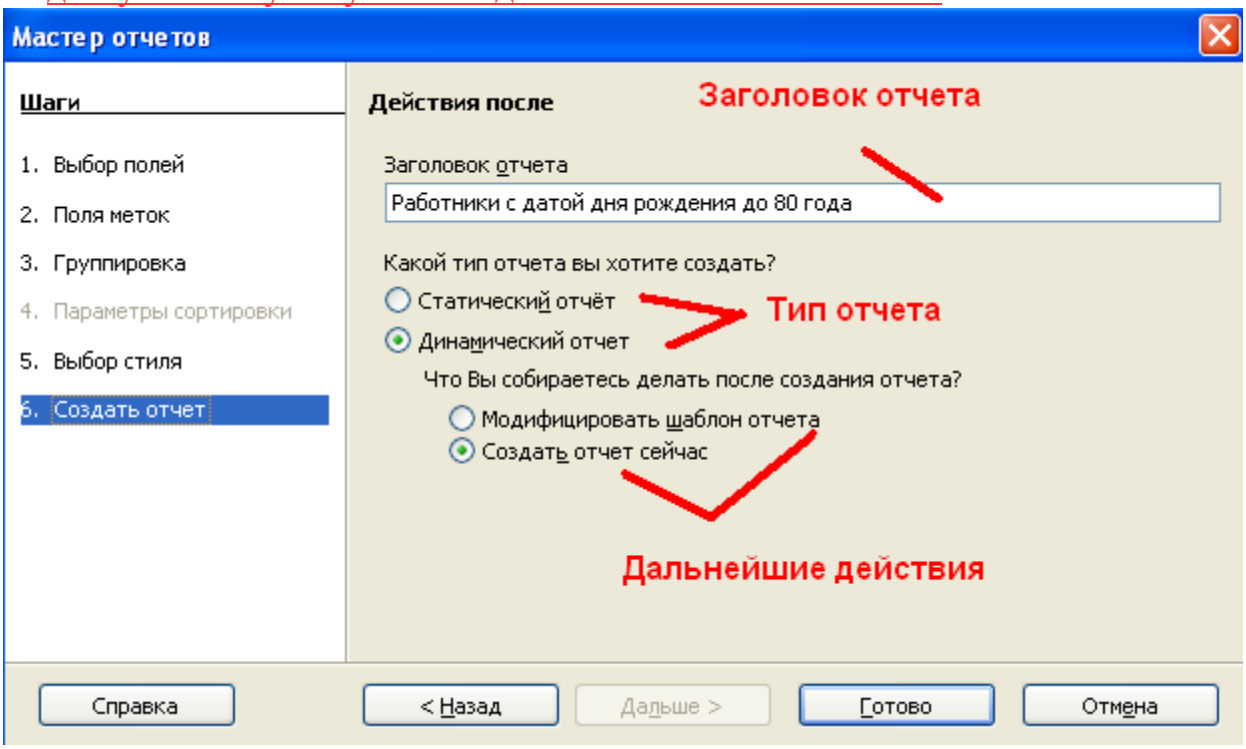


Рисунок 9. Заключительный этап создания отчета

Для создания отчета необходимо нажать кнопку *Готово*. На рисунке 10 показан открытый отчет.

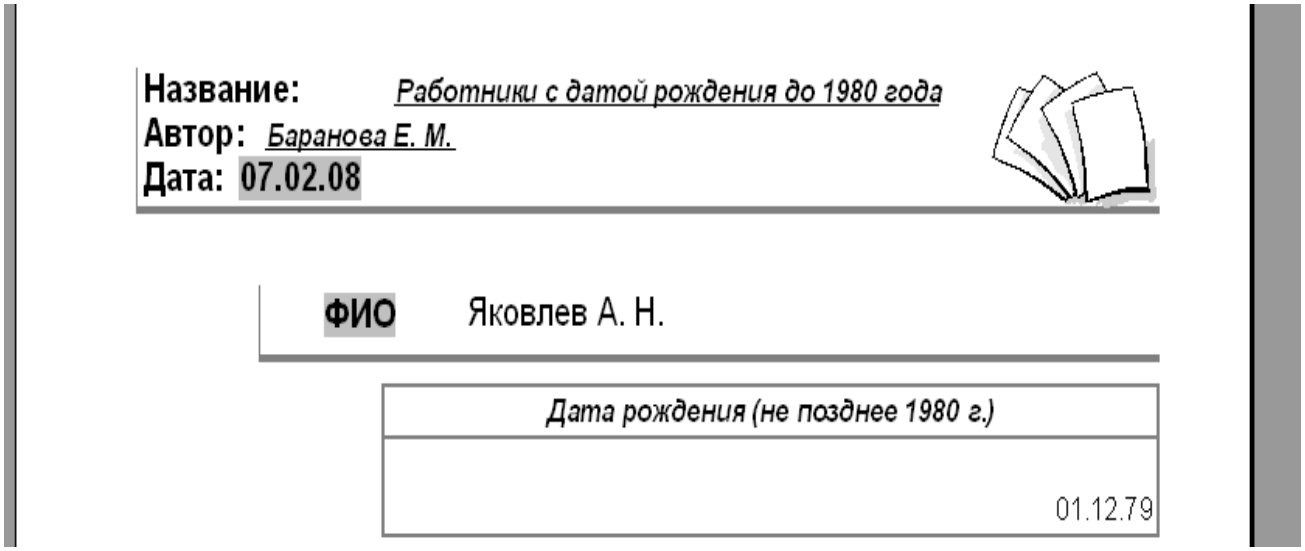


Рисунок 10. Фрагмент открытого отчета

Готовый отчет можно модифицировать. Для этого в окне создания базы данных относительно нужного отчета следует выбрать команду *Правка* и в открывшемся окне с помощью мыши ввести требуемые изменения, например, переместить границы полей, указать ФИО автора отчета.

С использование тех же кнопок на панели инструментов, что и при создании форм, отчет можно оформлять картинками и рисунками.

4.3. Задание на работу

Создание базы данных «Архив»

1. Создать произвольный отчет по одной из составляющей базу данных таблице.
2. Создать произвольные отчеты по ранее созданным запросам.
3. Произвольным образом оформить отчеты.

4.4. Требуемые результаты

В результате выполнения лабораторной работы должны получиться 8 отчетов (один по таблице, семь по запросам). Каждому отчету должно быть присвоено уникальное имя, отражающее его суть и указан автор.

4.5. Оформление отчета по работе

Отчет должен содержать:

1. Номер практической работы.

2. Название практической работы.

3. Цель практической работы.

4. Описание пунктов выполнения практической работы в соответствии с заданием на работу.

5. Вывод по работе.

4.6. Контрольные вопросы

8. Какие функции выполняют отчеты?
9. Какие существуют режимы создания запросов?
10. Какие существуют этапы создания отчетов?
11. Что такое метки?
12. Что такое макет отчета?
13. Что такое группировка?
14. Каким образом модифицируется готовый отчет?
15. Что такое колонтитул?
16. Что такое разметка данных?
17. Что такое статистический и динамический отчеты?

Часть 2. WEB – моделирование.

Тема 1. Язык HTML. Создание Web – страниц

Основой *World Wide Web* является язык гипертекстовой разметки *HTML (Hyper Text Markup Language)*. *HTML* – набор соглашений для разметки документов, которые определяют внешний вид документов на экране компьютера при доступе к ним с использованием программы браузера.

HTML – документ (или *Web* – страница) – это обычный текстовый файл с расширением *htm* или *html*, составленный на языке *HTML* и содержащий информацию, которая предназначена для публикации в *WWW*. Для работы подойдет практически любой текстовый редактор, например, в операционной системе *Windows* в числе стандартных есть два подходящих текстовых редактора – это *Блокнот* и *WordPad*.

Существуют редакторы, специально разработанные для создания *Web* – страниц, например *Microsoft FrontPage* или *Adobe Dreamweaver* (ранее известный как *Macromedia Dreamweaver*). Такие редакторы позволяют создавать *Web* – страницы с помощью визуальных инструментов, а затем уже в режим отображения кода и вносить изменения. Но если создавать с их помощью обычные *Web* – страницы сможет и новичок, то вносить изменения в режиме работы с кодом сумеет только человек, хотя бы немного знакомый с языком *HTML*. По этой причине для пользователей, еще только начинающих разбираться с *HTML*, следует использовать данные редакторы лишь в режиме работы с кодом, чтобы полностью контролировать ход создания *Web* – страницы (синтаксис языков *HTML* и *JavaScript* подсвечивается).

В настоящее время *JavaScript* используется в основном для создания встраиваемых в *Web* – страницы сценариев, позволяющих полностью управлять как самими *Web* – страницами, так и браузерами, в которых эти страницы открыты. Таким образом, можно сказать, что язык используется в основном для создания интерактивных *Web* – страниц и *Web* – приложений.

При создании *HTML – документ* он разбивается на элементы: заголовки, абзацы, рисунки, таблицы, списки и т.д. Для каждого элемента задается команда языка *HTML*, называемая *тегом*.

Тег – это фрагмент кода, который описывает определенный элемент документа HTML и заключается в угловые скобки < >.

Теги имеют *атрибуты* – это компоненты тега, содержащие указание о том, как браузер должен воспринять и обработать тег. Атрибут записывается после имени тега перед закрывающейся скобкой > и состоит из пары «имя атрибута =”значение”».

Комментарии располагаются в символах <!-…->.

Таблица 1.1

Основные теги HTML-документа. Теги форматирования шрифта и абзаца

Назначение	**Вид тегов**	**Примечание**
Общая структура документа HTML		
Тип документа	<HTML></HTML>	Начало и конец документа
Имя документа	<HEAD></HEAD>	Не отображается браузером
Заголовок	<TITLE></TITLE>	Содержимое строки заголовка окна браузера
Тело документа	<BODY></BODY>	Содержимое WEB-страницы
Структура содержания документа		
Внутренние	<H№> текст </H№>	Где № – номер уровня

заголовки различного уровня		заголовка (от 1 до 6). Например, <H1>…</H1> - заголовок 1-го уровня.
Заголовок с выравниванием	<H№ ALIGN=”LEFT| CENTER| RIGHT”> текст </H№>	LEFT - по левому краю, CENTER - по центру, RIGHT - по правому краю.
Форматирование абзацев		
Создание абзаца (параграфа)	<P> текст </P>	Абзацы отделяются двойным межстрочным интервалом
Перевод строки внутри абзаца	 	Одиночный тег
Выравнивание абзаца	<P ALIGN=”LEFT”>текст </P> <P ALIGN= ”CENTER”>текст </P> <P ALIGN=”RIGHT”> текст</P> <P ALIGN=”JUSTIFY”> текст </P>	LEFT - по левому краю CENTER - по центру RIGHT - по правому краю JUSTIFY – по ширине
Разделительная горизонтальная линия между абзацами	<HR SIZE=«?»>	Одиночный тег. «?» - толщина линии в пикселях. Толщину линии можно не указывать.
Форматирование шрифта		
Жирный	<B> текст </B>	<B>**Жирный**</B>
Курсив	<I> текст </I>	<I> *Курсив* </I>
Подчеркнутый	<U> текст </U>	<U> Подчеркнутый </U>
Перечеркнутый	<S> текст </S>	<S> ~~Перечеркнутый~~</S>
Увеличенный размер	<BIG> текст </BIG >	
Уменьшенный размер	<SMALL> текст </SMALL>	
Верхний индекс	^{текст}	^{Верхний индекс}
Нижний индекс	_{текст}	_{Нижний индекс}
Размер шрифта	<FONT SIZE=«?»> текст </FONT>	?- значения от 1 до 7 или относительное изменение

		(например, +2)
Базовый размер шрифта	<BASEFONT SIZE=«?»>	Одиночный тег ? – размер от 1 до 7; по умолчанию равен 3 и задается для всего документа в целом
Гарнитура шрифта	<FONT FASE=«название1, название2, …»> текст </FONT>	Текст оформляется первым, установленным на компьютере шрифтом из списка названий
Цвет шрифта	<FONT COLOR=”$$$$$$”> текст </FONT>	Цвет задается либо ключевым словом, либо шестнадцатеричным кодом с символом # RED –красный, #FF0000 – шестнадцатеричный код – красного цвета
Создание списков		
Нумерованный	<OL>элементы списка</OL>	<OL> <LI>Элемент списка 1 </LI> <LI>Элемент списка 2</LI> <LI>Элемент списка 3</LI> </OL>
Маркированный	<UL> элементы списка </UL>	
Элемент списка	<LI> элементы списка </LI>	

Таблица 1.2

Таблица основных цветов

Цвет	**Color’s name**	**Шестнадцатеричный код цвета**		
		Red	**Green**	**Blue**
Черный	Black	00	00	00
Темно-синий	Navy	00	00	80
Голубой	Blue	00	00	FF
Зеленый	Green	00	80	00
Темно-зеленый	Teal	00	80	80
Салатный	Lime	00	FF	00

Цвет	Color's name	Шестнадцатеричный код цвета		
		Red	Green	Blue
Бледно-голубой	Aqua	00	FF	FF
Вишневый	maroon	80	00	00
Фиолетовый	Purple	80	00	80
Оливковый	Olive	80	80	00
Серый	Gray	80	80	80
Светло-серый	Silver	C0	C0	C0
Красный	Red	FF	00	00
Лиловый	Fuchsia	FF	00	FF
Желтый	Yellow	FF	FF	00
Белый	White	FF	FF	FF

Задание на выполнение:

Пример1

1. Создать файл с гипертекстовым документом:

- Запустить редактор Блокнот, ввести текст:

Приветствую Вас на моей первой web-страничке!

- Сохранить файл в созданной папке. При сохранении, в окне диалога **Сохранить как…** в строке **Тип файла:** выбрать вариант **Все файлы (*.*)** , а в строке **Имя файла** задать имя с расширением .**htm**, например **1_name.htm** (где **name** – ваше имя)

- Закрыть документ, найти его пиктограмму в окне **Мой компьютер** или в окне программы **Проводник.**

- Открыть файл. Проанализировать, *с помощью какого приложения отображается файл* и как выглядит введенная фраза.

2. Ввести теги, определяющие структуру html-документа:

- С помощью контекстного меню открыть файл с помощью

редактора Блокнот. Ввести приведенные ниже теги, в разделе заголовка документа (между тегами <**TITLE**> </**TITLE**>) указать свою фамилию.

<HTML>

<HEAD> <TITLE> Фамилия </TITLE>

</HEAD>

<BODY>

Приветствую Вас на моей первой web-страничке!

</BODY>

</HTML>

- **Сохранить** документ под тем же именем, обновить его отображение в браузере (выполнить **Вид/Обновить** или нажать кнопку **Обновить** на панели инструментов). Проанализировать произошедшие изменения в отображении документа.

3. Отредактировать документ:

- Вызвать меню браузера **Вид/Просмотр HTML-кода** и добавить после текста **«Приветствую Вас на моей первой web-страничке!»** текст подписи:

Студент группы NNN Фамилия Имя

Сохранить документ (но не закрывать) и обновить его просмотр в браузере.

- Используя одиночный тег **
, отредактировать документ так, чтобы подпись начиналась с новой строки, а **Фамилия Имя – в следующей строке. Просмотреть в браузере новый вариант.

Внимание! ***После каждого изменения документ нужно сохранять, а просмотр в браузере начинать с обновления загрузки документа с помощью кнопки «Обновить» на панели инструментов.***

4. Оформить фрагменты текста с помощью стилей **Заголовков**:

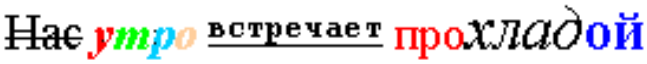
- Первую строку документа оформить стилем **Заголовок 1-го уровня** с помощью парного тега **<H1> …</H1>**. Вторую строку оформить как **Заголовок 6-го уровня**, а третью как **Заголовок 4-го уровня**.
- Просмотреть документ в браузере, изменяя настройку отображения шрифтов (меню **Вид / Размер шрифта / Самый крупный, Средний, Мелкий** и **Самый мелкий**).
- Поменять стиль оформления первой строки на **Заголовок 2 уровня**, второй строки – на **Заголовок 5 уровня**, последней строки - на **Заголовок 3-го уровня**.

5. Выполнить форматирование шрифта:

- После строки **Фамилия Имя** добавить еще одну строку текста

Нас утро встречает прохладой

- Оформить фразу по приведенному ниже образцу.



В слове УТРО все буквы должны иметь **разные цвета**. В слове ПРОХЛАДОЙ оформить буквы ПРО – **красным** цветом, ОЙ – **синим**.

- Оформить строку с подписью (**Студент группы NNN Фамилия Имя**) **курсивом**, размер шрифта задать относительным изменением. Использовать теги **<FONT SIZE=«+2»>** и **<I>**
- Просмотреть полученный документ в браузере.

6. Выполнить форматирование абзацев:

- Создать новый документ **2_name.htm**, сохранить его в той же рабочей папке.
- Ввести текст (использовать копирование текста из документа **1_name.htm**):

<HTML>

<HEAD> <TITLE> Фамилия </TITLE>

</HEAD>

<BODY>

**Приветствую Вас на моей второй web-страничке!
 Монолог Гамлета**

</BODY>

</HTML>

- Выровнять текст **по центру**.
- Ввести текст:

Быть иль не быть – вот в чем вопрос. Что благороднее: сносить удары неистовой судьбы – иль против моря невзгод вооружиться, в бой вступить. И все покончить разом…

- Оформить выравнивание абзаца **по ширине**.
- Ограничить абзац горизонтальными разделительными линиями сверху и снизу, используя тег <HR>.
- Скопировать монолог и разбить его на абзацы. Выровнять **по центру**.

Быть иль не быть – вот в чем вопрос.
Что благороднее: сносить удары
Неистовой судьбы – иль против моря
Невзгод вооружиться, в бой вступить
И все покончить разом…

- Сохранить документ.
- Просмотреть документ в окне браузера, изменяя размер окна.

7. Выполнить оформление списков:

- Создать новый документ **3_name.htm**, сохранить его в той же рабочей папке жесткого диска.
- Ввести текст:

<HTML>

<HEAD> <TITLE> Фамилия </TITLE>

</HEAD>

<BODY>

Приветствую Вас на моей третьей web-страничке!

</BODY>

</HTML>

- Дополнить текст документа (между тегами <BODY>…</BODY>) следующим текстом:

Я знаю как оформлять:
Шрифты,
Заголовки,
Абзацы

- Оформить три последние строки как **список нумерованный**. Для этого использовать следующую конструкцию тегов:

<OL>

<LI> Шрифты, </LI>

<LI> Заголовки, </LI>

<LI> Абзацы </LI>

</OL>

- Поменять оформление списка на **список маркированный**. Использовать теги <UL>, </UL>

- Создать «смешанный» список:

Я знаю как оформлять:

1. **Шрифты**
   - **Размер**
   - **Цвет**
   - **Гарнитуру**
   - **Индексы**
2. **Заголовки**
   - **От 1-го до 6-го уровня**
3. **Абзацы**
   - **Выравнивание**
   - **Разрыв строк внутри абзаца**
   - **С использованием переформатирования**.

Пример 2

```
<html>
  <head>
    <title> привет </title>
  </head>
  <body bgcolor = “red”>
     <h1> Заголовок первого уровня </h1>
     <h2> Заголовок второго уровня </h2>
     <p> под строкой черта<hr>
     <p> строка с красной стоки </p>
    Еще одна  с новой строки <br>
    <FONT face=”Georgia” color=”green” size=”7”>Странный шрифт
      зеленого цвета</FONT><br>
   <B>Полужирный текст</B>
   <I>Курсивный текст</I>
   <U>Подчеркнутый текст</U>
   <UL> <LH>Маркированный список</LH>
     <LI> первый элемент
     <LI> второй элемент
   </UL>
   <OL> Нумерованный список
      <LI> первый элемент
      <LI> второй элемент
   </OL>
  </body>
   </html>
```

Задание для самостоятельного выполнения

На основании изложенного материала создать *Web* – страницу вида:

Параметры для характеристики файла (фон желтый)

1. Полное имя файла (голубой);

2. Объём файла в байтах (красный);

3. Дата создания файла (зеленый);

4. Время создания файла (оранжевый);

5. Специальные атрибуты файла (серый):

- R (*Read only*) – только для чтения,
- H (*Hidden*) – скрытый файл,
- S (*System*) – системный файл,
- A (*Archive*) – архивированный файл

Тема 2. Работа с таблицами и рисунками на языке HTML

2.1. Вставка в HTML-документ рисунков. Создание закладок и гиперссылок

Таблица 2.1

Основные теги вставки рисунков, закладок и гиперссылок

Вставка изображений		
Вставка графического файла	<IMG SRC=”файл” WIDTH=”ширина” HEIGHT=”высота” >	*Пример*: <IMG SRC=”grafica.gif” WIDTH=”550” HEIGHT=”240” ALIGN=” right” ALT=”Графический файл”>
Выравнивание картинки относительно текста	<IMG SRC=”файл” ALIGN=”left|right|top|bottom”>	
Вывод текста всплывающей подсказки при наведении курсора мыши на рисунок	<IMG SRC=”файл” ALT=”текст”>	
Вставка ссылок		
Ссылки на другую страницу	<A HREF=” страница”> текст </A>	<A HREF=”str2.htm”> Ссылка1</A>
Ссылка на закладку в другом документе	<A HREF=” страница # имя закладки”> текст</A>	<A HREF=” index.htm #met1”> На главную страницу </A>”
Ссылка на закладку в том же документе	<A HREF=”#имя закладки ”> текст </A>	<A HREF=” #metka1”> Ссылка2</A>
Определить закладку	<A NAME=”имя закладки”>текст</A>	<A NAME=”metka1”></A>
Цвет фона, текста и ссылок		
Фоновая картинка	<BODY BACKGROUND=”файл рисунка”>	<BODY BACKGROUND
Цвет фона	<BODY BGCOLOR=”#$$$$$$”>	

Цвет текста	<BODY TEXT=”#$$$$$$”>	=”grafica.gif” TEXT=”black” (черный) LINK=”#FF0000” (красный) VLINK=”#FFFF00” (желтый) ALINK=”#FFFFFF” (белый) </BODY>
Цвет ссылки	<BODY LINK=”#$$$$$$”>	
Цвет пройденной ссылки	<BODY VLINK=”#$$$$$$”>	
Цвет активной ссылки	<BODY ALINK=”#$$$$$$”>	

Каждая таблица начинается открывающимся тегом <TABLE> и заканчивается тегом </TABLE>.

4. TR – элемент создания строки,
5. TD – элемент, определяющий содержимое ячейки данных,
6. TH – элемент, определяющий ячейку заголовка.

Атрибуты тега <TABLE>:

- **width** – задает ширину таблицы. Например тег <TABLE width =40%> задает таблицу с длиной всех строк, равной 40% от ширины окна.
- **align** – выравнивание таблицы в документе, имеет значения:left,right,center.
- **border** – задает вывод рамки таблицы. Если его значение не определенно как в Примере, то рамка таблицы будет иметь толщину 1 пиксель.

Атрибуты тегов <TR> и<TD>:

1. **width и height –** устанавливает размеры ячеек строки: ширину и высоту.
2. **align** – выравнивание содержимого в ячейках, имеет значения:left,right,center и justify (выравнивание по левому и правому краю).

3. **valign** – определяет выравнивание содержимого по вертикали, имеет значения: top (выравнивание по верхнему краю ячейки),bootom(по нижнему краю ячеек),middle(центрирование по вертикали)

Цвет в таблице:

- **bgcolor-** определяет цвет фона в таблице. В зависимости от того в какой из тегов (TABLE, TR, TD, TH) этот атрибут вводится будет задаваться цвет всей таблицы, строки, ячейки или заголовка.

Объединение ячеек таблицы:

- **rownspan –** объединяет ячейки смежных строк. Значение атрибута задает количество объединяемых ячеек. Например < TD rownspan=2 > устанавливает объединение двух смежных ячеек.
- **colspan –** объединяет ячейки смежных столбцов. Например < TD colspan=3 > формирует одну ячейку данных из трех ячеек смежных столбцов.

Пример

```
<html>
  <head>
    <title>Таблица</title>
  </head>
 <body>
    <H1>Список фамилий</H1><hr>
   <TABLE border=1>
      <TR><TH>Фамилия</TH><TH>Имя</TH></TR>
      <TR><TD>Петров</TD><TD>Иванов</TD></TR>
      <TR><TD>Сидоров</TD><TD>Петя</TD></TR>
   </TABLE>
 </body>
</html>
```

Фамилия	**Имя**
Петров	Иван
Сидоров	Петя

Задание для самостоятельного выполнения

1. Создать *Web* – страницу вида:

Квартальный отчет

ФИО	**Оплата**	**Дата**	**Подпись**
Андреев А.А.	120	12.1.06	
Воронов О.Л.	120	12.1.06	
Петров В.В.	240	12.2.06	
Сидоров Ф.Ф.	60	12.2.06	
Темофеев П.Д.	100	13.2.06	
Федоров Ф.Ф.	60	12.3.06	
Итог			**700**
Отчет по месяцам		Январь	**240**
		Февраль	**400**
		Март	**60**
Итог			**700**
Гл. бухгалтер Морозова Т.М.			

Тема 3. Основы языка JavaScript.

JavaScript - это язык программирования, используемый в составе страниц HTML для увеличения функциональности и возможностей взаимодействия с пользователями. Он был разработан фирмой Netscape в сотруднечестве с Sun Microsystems на базе языка Sun's Java. С помощью JavaScript на Web-странице можно сделать то, что невозможно сделать стандартными тегами HTML. Скрипты выполняются в результате наступления каких-либо событий, инициированных действиями пользователя. Создание Web-документов, включающих программы на JavaScript, требует наличее текстового редактора и подходящего браузера. Некоторые просмоторщики включают в себе встроенные редакторы, поэтому необходимость во внешнем редакторе отпадает. Несмотря на отсутствие прямой связи с языком Java[14], JavaScript может обращаться к внешним свойствам и методам Java-апплетов, встроенных в страницу HTML. Разница сводится к тому, что апплеты существуют вне браузера, в то время как программы JavaScript могут работать только внутри браузера.

JavaScript-сценарии интегрируются в HTML-документы с использованием пары дескрипторов **<script>** и **</script>.** HTML-документ может содержать множество подобных пар, причем каждая из них зачастую заключает в себе более одного набора операторов JavaScript. Для **<script>** обязательно требуется присутствие **</script>.**

Атрибуты дескриптора <**script**>.

- **Defer** - Это атрибут логического типа, который используется для уведомления браузера о том, генерирует ли JavaScript-сценарий какое-либо содержимое.

[14] **JavaScript и Java** - это два разных языка программирования. Java - это объектно-ориентированный язык программирования и запускается при помощи компилятора и вспомогательных файлов. Разрабатываемые с помощью Java программы могут работать как законченные приложения либо как встроенные в Web-страницу апплеты. И хотя они встроены в страницу HTML, они храняться на клиенской машине как отдельные файлы. Напротив, JavaScript, размещаются внутри HTML страницы и не могут существовать, как отдельные программы и функционируют, будучи запущенными в браузерах типа Netscape Navigator или Internet Explorer

- **language** - Атрибут для определения языка и версии, используемых внутри дескрипторов.
- **src** - Этот атрибут определяет URL внешнего исходного JavaScript-файла.
- **type** - Атрибут, пришедший на замену language; он сообщает браузеру, какой язык используется внутри дескрипторов.

Для того чтобы уведомить JavaScript о том, что конкретный идентификатор должен использоваться в качестве переменной, в первую очередь, его потребуется объявить. Для объявления переменных в JavaScript используется ключевое слово var, за которым следует новое имя переменной. Имя будет резервироваться как переменная, которую можно использовать в качестве области памяти для хранения любых данных. Следующий пример демонстрирует возможность одновременного объявления нескольких переменных (между отдельными именами должны ставятся запятые).

```
var internetAddress;
var n;
var i, j, k;
var isMouseOverLink, helloMessage;
```

После объявления переменной можно присваивать начальное значение. Подобное действие носит название *инициализации* и выполняется при помощи операцией присваивания =.

Переменную можно инициализировать непосредственно при ее объявлении либо позже, в любом месте сценария. Установка начального значения переменной во время объявления упрощает запоминание, какой тип значений первоначально планировалось хранить в переменной. Уточним предыдущий пример, добавив необходимые инициализации:

```
var internetAddress = "name@company.com";
```

```
var n = 0.00 ;
var i = 0, j = 0, k;
var isMouseOverLink = false;
var helloMessage = "Hello, thank you for coming!";
k = 0;
```

Несложно заметить, что все переменные, кроме k, были проинициализированы во время объявления, а k инициализируется позже. JavaScript читает строки кода сверху вниз, выполняя директивы одну за другой. До инициализации переменная остается *неопределенной (undefined)*, поэтому прочитать ее значение нельзя. Попытка чтения значения переменной перед ее инициализацией приведет к ошибке во время выполнения приложения.

3.1. Типы данных

При сохранении порции данных (более известных под именем *значения),* JavaScript автоматически относит ее к одной из пяти категорий:

Таблица 3.1

Тип данных	Значение
number	-19, 3.14159
boolean	true, false
string	"Элементарно, дорогой Ватсон", " "
function	unescape, write
object	window, document, null

Переменные типа number хранят действительные или целые числа, boolean — значения true или false, а string содержат строковые литералы, включая пустую строку. В табл. 1 пустая строка представляется с помощью двух двойных кавычек.

Функциональный тип (function) либо определяется пользователем, либо относится к встроенным функциям. Например, функция unescape является встроенной в JavaScript. Они также относятся к типу данных function. Функции служат одной цели — разделению задач, предназначенных для выполнения в одной программе. Можно считать функцию уведомлением JavaScript о необходимости выполнения конкретного списка связанных команд и выдачи сигнала, как только это будет сделано. При вызове функции можно передать значения, называемые *аргументами (arguments).* Аргументы можно использовать в качестве *переменных (variables)* в пределах одного блока операторов (локальные переменные). Как только данные присвоены переменной, становится возможной обработка данных либо использование их в вычислениях.

Локальная переменная может меняться только в пределах функции, в которой она объявлена, в то время как глобальная переменная может изменяться в любом месте документа.

Ниже приводится синтаксис объявления функции в JavaScript:

function *имяфункции* ([аргумент1] [... , аргументN]) {

[операторы]}

Теоретически функцию можно объявлять в любом месте в пределах раздела **<script>.** Существует единственное ограничение — нельзя объявлять функцию в пределах другой функции или управляющей структуры. Однако имейте в виду, одни блоки HTML-документа загружаются раньше других,

равно как и любые сценарии, встроенные в эти HTML-блоки. В этой связи рекомендуется объявлять функции HTML-документа в разделе **<head>.** Объявление всех функций в этом разделе гарантирует, что функции будут доступны, если какому-то сценарию потребуется вызвать их немедленно. Для вызова функции напишите ее имя вместе с набором круглых скобок. В эти круглые скобки можно заключать любые аргументы, которые будут передаваться в функцию. Функции могут возвращать значения. Функции, возвращающие значения, можно использовать в выражениях.

Функции, принадлежащие объектам, в JavaScript называются *методами (methods)*. Главная программа размещается внутри раздела **<body>** и обрамляется парой дескрипторов **<script>.**

Базовые клиентские объекты JavaScript, такие как window или document, имеют тип данных object. Переменные типа object, или просто *объекты,* могут хранить объекты. Переменная со значением null относится к типу object. Инициализация переменной значением null позволяет избежать ошибок, если нет уверенности в будущем ее использовании. Как правило, в других языках программирования требуется определять тип данных, представляемых новой переменной. На протяжении всей программы тип любого значения, присваиваемого переменной, должен быть определенным и неизменным. Более того, если переменной присваиваются значения различных типов, это приводит к ошибке. Для JavaScript, слабо типизированного языка, подобное не имеет места. Не потребуется определять типы данных. Кроме того, одной и той же переменной можно присваивать значения разных типов. Переменная JavaScript способна в любой момент принять новое значение. Рассмотрим примеры допустимого использования переменных JavaScript:

```
var carLength ;
```

```
carLength = 4 + 5 ;
document.writeln(carLength);
carLength = "9 футов";
document.writeln(carLength);
```

После объявления переменной carLength ей присваивается значение 4+5. В JavaScript число 9 сохраняется с типом number. Присваивая carLength значение другого типа "9 футов", в этой переменной сохраняется строковый литерал (string). Операция подобного рода уменьшает количество действий, которые могли бы потребоваться в других языках программирования для уведомления о факте смены типов значений.

3.2. Преобразование типов

JavaScript обеспечивает два встроенных метода для преобразования строк в числа: parseint() и parseFloat(). Обе функции в качестве параметров принимают строки и пытаются преобразовывать строковые данные в числовые значения. parseint() преобразует строку в целочисленное значение, а parseFloat() — в значение с плавающей точкой. Например, приведенный ниже код возвращает значение 123:

```
var myString = new String ("123.88888");
document.write(parseint(myString));
```

Следующий код возвращает значение 1234.0012121:

```
var myString = new String("1234.0012121");
document.write(parseFloat(myString));
```

Обе программы начинают преобразование с левой стороны строки и продолжают его до тех пор, пока не встретится нечто, не являющееся цифрой (0—9), десятичной точкой (.) либо знаком плюса или минуса (+/-).

После столкновения с нецифровым символом оставшаяся часть строки игнорируется. Приведенный ниже код вернет значение 1234.01:

```
var myString = new String ("1234.01 — это общая сумма") ;
document.write(parseFloat(myString));
```

С другой стороны, следующий код возвращает значение *NaN* (*Not a Number* — не число), поскольку знак доллара не является цифрой:

```
var myString = new String ("$1234.01 — это общая сумма");
document.write(parseFloat(myString));
```

При работе с текстовыми объектами без функций преобразования не обойтись. Поскольку свойство value объекта Text возвращает строку, эти данные придется преобразовывать каждый раз, когда дело доходит до необходимости трактовки текста, вводимого пользователем, как цифрового значения.

Можно также преобразовывать численные значения в строки. Однако упомянутое действие можно было выполнить, основываясь на том, как JavaScript выполняет операцию сложения (+). Если во время сложения элементов выражения JavaScript встречала строку, с этого момента все выражение расценивалось как строка. Например, 35+100 возвращает цифровое значение 135. С другой стороны, 35+"100" в результате дает строковое значение "35100".

Обратите внимание, что JavaScript выполняет операции сложения слева направо, поэтому перед преобразованием выражения в строку можно сложить, скажем, два числа. Например, 10 + 20 + "40" вернет строковое значение "3040". А вот "40" + 10 + 20 выдает в качестве результата строковое значение "401020" из-за того, что левее всех числовых значений находится строковый литерал.

Многие языки программирования требуют обязательного преобразования числовых значений перед их использованием в качестве строковых. Все что требуется предпринять — это либо поместить числовое значение в строковое выражение, либо добавить к нему пустую строку. Например, выражение 1300+ "" возвращает строковое значение "1300".

3.3. Операции

3.3.1. Комбинации операции присваивания и арифметических операций

Комбинации операции присваивания и арифметических операций:

x += y сокращенная запись для x = x + y

x - = y сокращенная запись для x = x - y

x *= y сокращенная запись для x = x * y

x /= y сокращенная запись для x = x / y

x %= y сокращенная запись для x = x % y

Операция взятия по модулю обозначается знаком процента (%). Найти модуль двух операндов означает найти остаток после деления первого операнда на второй. В примере x = 10 % 3 переменная x получит значение 1, поскольку результатом деления 10 на 3 будет 3 с остатком 1. При помощи операции взятия по модулю можно легко определить, что одно число

является множителем другого: при этом модуль этих двух чисел будет равен 0.

Для записи **инкремента** применяется ++, а для **декремента** - - :

++i то же самое, что и i = i + 1

- - i то же самое, что и i = i – 1

Положение оператора инкремента важно при присваивании:

y = ++i значение i увеличивается на 1 и присваивается переменной y ;

y = i++ значение i присваивается переменной y, а потом увеличивается на 1.

3.3.2. Операции сравнения

= = Операция равенства. Возвращает **true,** если операнды равны между собой.

!= Операция неравенства. Возвращает **true,** если операнды не равны между собой.

> Операция "больше". Возвращает **true,** если значение левого операнда больше значения правого операнда.

>= Операция "больше либо равно". Возвращает **true,** если значение левого операнда больше либо равно значения правого операнда.

< Операция "меньше". Возвращает **true,** если значение левого операнда меньше значения правого операнда.

<= Операция "меньше либо равно". Возвращает **true**, если значение левого операнда меньше либо равно значения правого операнда.

3.3.3.Логические операции

&& Логическая операция И (конъюнкция). Возвращает **true,** если оба выражения имеют значения **true.** В противном случае возвращается **false.** Рассмотрим примеры:

(1 > 0) && (2 > 1) возвращает **true.**

(1 > 0) && (2 < 1) возвращает **false.**

|| Логическая операция ИЛИ (дизъюнкция). Возвращает **true,** если хотя бы одно из значений левое или правое равно **true.** Если ни одно из них не равно **true,** возвращается **false.**

Рассмотрим примеры:

(1 > 0) || (2 < 1) возвращает **true.**

(1 < 0) || (2 < 1) возвращает **false.**

! Логическая операция НЕ (отрицание) - унарная операция, которая возвращает противоположное значение булева выражения. Если выражение равно **true,** возвращается **false,** а если **false,** возвращается **true.** Эта операция не изменяет значения выражения, поскольку работает подобно арифметической операции отрицания. Рассмотрим примеры:

! (1 > 0) возвращает **false.**

! (1 < 0) возвращает **true.**

3.3.4. Условные операции

В JavaScript есть одна тернарная операция ? и :, которая позволяет присвоить значение переменной в зависимости от выполнения условия. Рассмотрим следующий пример:

```
var sing = (a>=0) ? “Положительное” : “Отрицательное”;
```

В зависимости от результата выражения до вопросительного знака переменная принимает одно из значений: если истинно, то вычисляется

выражение до двоеточия, если ложно, то выражение, стоящее после двоеточия. В данном случае, если переменная a больше или равна 0, то переменная sing принимает значение “Положительное”, иначе переменная sing принимает значение “Отрицательное”.

3.3.5. Операция typeof

Операция **typeof** возвращает тип данных, хранящихся в операнде в текущий момент времени. Это оказывается особенно полезным при выяснении, была ли определена переменная. Рассмотрим следующие примеры:

typeof unescape возвращает строку "**function**",

typeof undefinedVariable возвращает строку "**undefined**",

typeof 33 возвращает строку "**number**",

typeof "A String" возвращает строку "**string**",

typeof true возвращает строку "**boolean**",

typeof null возвращает строку "**object**".

Тема 4. Пользовательские объекты

JavaScript не относится к объектно-ориентированным языкам программирования, но JavaScript позволяет использовать уже существующие программные объекты, встроенные в обозреватель.

Пользовательские объекты JavaScript тесно связаны с массивами. Массивы — средство структурирования данных в контейнере. Какими бы мощными не были бы массивы, они не в состоянии удовлетворить все запросы разработчиков. Несмотря на то, что массивы хранят данные, они не могут хранить поведение. *Объект* – это сложный тип данных, включающий в себя множество переменных – *свойств*, набор функций для манипуляции этими переменными – *методов*, взаимосвязь между действиями пользователя над объектом и внешней функции обработки события – *событий*.

Для создания объекта JavaScript необходим *конструктор (constructor).* Конструктор — это специальная функция JavaScript, которая определяет вид объекта и его поведение. Сам по себе конструктор не создает объекты. Вместо этого он обеспечивает шаблон вида реализованного объекта. (Создание экземпляра объекта называется *реализацией (instantiating)* объекта.) Базовая

структура реализованного объекта выглядит так:

```
function object (parameter1 , parameters2, …) {
this.property1 = parameter 1
this.property2 = parameter 2
this.property3 = parameter 3
this.property4 = parameter4
this.method1 = function1
this.method2 = function2
```

}

Для задействования объекта, объявленного в конструкторе необходимо создать его экземпляр. Для этих целей используется оператор new, синтаксис которого выглядит так:

```
экземплярОбъекта = new объектный
Тип (параметр1, параметр2, параметр3, ...)
```

Доступ к свойствам и методам объекта в JavaScript осуществляется в соответствие с точечной нотацией. Эта запись, показанная в следующем синтаксическом примере, обеспечивает иерархический способ доступа к свойствам и выполнениям методов.

```
имяОбъекта. имяСвойства
имяОбъекта. имяМетода (аргументы)
```

К текущему объекту обращаются через специальную переменную this. В объявлении типа объекта переменная this ссылается на сам объект. Этот объект (текущий объект) является объектом, для которого объявляется метод.

Пример

Создать пользовательский экземпляр объекта (книги):

```
<html>
  <head>
   <title>Пользовательский объект книги </title>
   <script  type ="text/javascript">
```

```
        function book(title, author, ISBN, subject, rating) {
          this.title = title;
          this.author = author;
          this.ISBN = ISBN;
          this.subject = subject;
          this.rating = rating;
          this.show = show;
        }
        function show() {
          objWindow = window.open ("", "", "width=600 ,height=300" ) ;
          objWindow.document.write( "<html><body>" ) ;
          objWindow.document.write("<h1>Экземпляр объекта</h1>") ;
          objWindow.document.write("<p>") ;
          objWindow.document.write("Название книги: " + this.title + "<br>") ;
          objWindow.document.write("Автор: " + this.author + "<br>");
          objWindow.document.write("Код книги: " + this.ISBN + "<br>");
          objWindow.document.write("Сюжет: " + this.subject + "<br>");
          objWindow.document.write("Рейтинг:"  + this.rating + "<br>")
          objWindow.document.write("</body ></html>") ;
          objWindow.document.close() ;
        }
      </script>
    </head>
  <body>
   <script  type="text/javascript">
      dbBook = new book("Дэвид Флэнаган",  "Java. Справочник", "1-57521-118-1",
"Справочник", 5);
      dbBook.show();
   </script>
  </body>
```

```
</html>
```

Помимо точечной нотации, существуют и другие способы обращения к свойствам объекта. Следующий пример показывает способ доступа к объекту как к массиву:

```
имяОбъекта [ "имяСвойства" ]
```

А вот доступ через индексацию:

```
имяОбъекта [целочисленныйИндекс]
```

Последняя методика возвращает атрибут с номером целочисленный индекс.

Задание для самостоятельного выполнения

Создать экземпляр книги, создать объект «Студент» и сделать экземпляр со своими данными.

Тема 5. Иерархия объектов JavaScript

Согласно правилам языка JavaScript все элементы на веб-странице выстраиваются в иерархическую структуру:

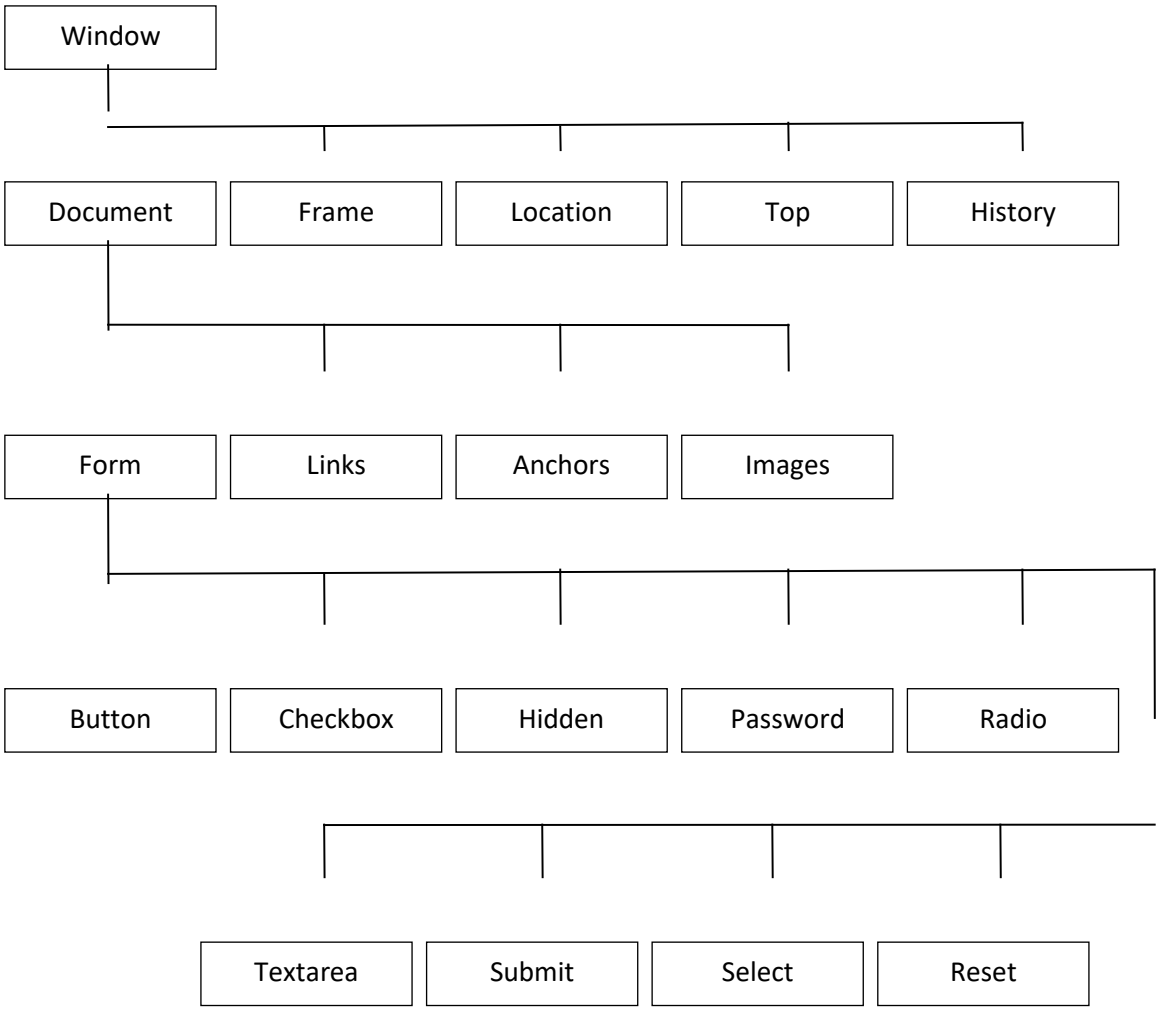


Рис.1 Иерархическая структура.

Обращаясь к объекту, указывайте путь в иерархии, например, устанавливается цвет фона документа в рамке с именем main:

```
Window. main.document.bgColor = ‘red’
```

5.1. Объект Window

Объект **Window** не имеет эквивалентного дескриптора HTML, тем не менее, он создается при открытии нового окна браузера. Следующий пример показывает, как работать с объектом **Window** в коде JavaScript. Предположим, что требуется добавить текст в строку состояния окна:

```
window.status = 'Добро пожаловать на мою домашнюю страницу!';
```

Подобно другим объектам, **Window** имеет несколько различных свойств и методов. Поскольку **Window** — это объект самого верхнего уровня, некоторые из объектов могут вызываться или упоминаться без **window,** перед ними. Пример — метод **alert().**

Таблица 5.1

Некоторые наиболее часто используемые методы и свойства объекта Window

Метод	atob()	Декодирование строки, закодированной с помощью кода base 64
	alert()	Выводит диалоговое окно предупреждения, отображающее переданную в метод текстовую строку.
	back()	Загружает предыдущую страницу вместо экземпляра **window.**
	blur()	Убирает фокус из окна.
	clearTimeout()	Сброс таймера
	close()	Закрывает экземпляр окна.
	confirm()	Отображает диалоговое окно подтверждения с

		кнопками Ok и Отмена.
	find()	Отображает диалоговое окно поиска, в которое можно вводить текст для поиска на текущей странице.
	focus()	Устанавливает фокус на указанное окно.
	forwardf)	Загружает следующую страницу вместо экземпляра **window.**
	home()	Загружает определенную пользователем страницу в экземпляр **window.**
	moveBy()	Перемещает окно в соответствие с заданным смещением.
	moveTo()	Перемещает окно в заданную позицию.
	open()	Открывает новый экземпляр окна.
	print()	Вызывает диалоговое окно печати, позволяющее вывести на печать текущее окно.
	prompt()	Отображает диалоговое окно приглашения на ввод команды.
	scroll()	Выполняет прокрутку документа в окне к указанной позиции.
	setTimeout()	Вызывает функцию или выполняет выражение по прошествии определенного числа миллисекунд.
Свойства	closed	Определяет, был ли экземпляр window закрыт.
	document	Ссылается на всю информацию относительно документа, находящемся в данном окне.
	defaultStatus	Определяет заданное по умолчанию сообщение в строке состояния окна.
	frames	Ссылается на всю информацию относительно фреймов данного окна.

	history	Ссылается на URL-адреса, которые посетил данный пользователь.
	length	Определяет количество фреймов в текущем окне.
	name	Содержит имя окна.
	parent	Ссылается на родительское окно, которое отображает текущий фрейм.
	self	Ссылается на текущее окно.
	status	Ссылается на сообщение, отображаемое в строке состояния окна.
	top	Ссылается на родительское окно, отображающее текущий фрейм.
	window	Ссылается на текущее окно.

Пример 1

Работа с окном предупреждения:

```
<html>
  <head>
   <title>Работа с предупреждением</title>
    <script>
      alert("Это предупреждение");
    </script>
  </head>
  <body>
  </body>
</html>
```

Пример 2

Работы с окном запроса:

```
<html>
  <head>
   <title>Работа с запросом</title>
  </head>
  <body>
    <script>
     var name = prompt("Напишите свое имя", "Введите имя")
     document.write("Привет, "+name+ ", спасибо, что зашли.")
    </script>
  </body>
</html>
```

5.2. Объект Document

Несмотря на то, что объект **Window** является объектом верхнего уровня в иерархии, **Document,** возможно, один из наиболее важных.

Таблица 5.2

Некоторые наиболее часто используемые методы и свойства объекта Document

Методы	close()	Закрывает поток вывода документа.
	getSelection()	Возвращает текст, выделенный пользователем на странице.
	open()	Открывает поток вывода документа.
	write()	Добавляет текст к документу.
	Writeln()	Добавляет к документу текст и символ новой строки.

Свойства	alinkColor	Цвет активной ссылки.
	bgColor	Цвет фона документа.
	fgColor	Цвет текста в документе.
	formName	Определяет экземпляр **Form,** к которому обращаются с использованием атрибута **name** в дескрипторе **<form>.**
	height	Определяет высоту документа в точках.
	images	Массив объектов **Image.**
	lastModified	Дата последней модификации документа.
	layers	Массив объектов **Layer.**
	linkColor	Цвет ссылок.
	referrer	URL документа, с которым был связан текущий документ.
	title	Заголовок документа.
	URL	URL текущего документа.
	vlinkColor	Цвет посещенных ссылок.
	width	Определяет ширину документа в точках.

Чаще всего объект Document используется для генерации HTML-страниц при помощи JavaScript. Для упомянутых целей можно применять методы document.write() или document.writeln(). Например, приведенный ниже код отображает HTML-текст, переданный в качестве параметра:

```
<html>
 <body>
 <script type="text/javascript">
 document.write ( "<H1 >Текст , созданный JavaScript</H1 >" ) ;
 </script>
 </body>
```

</html>

5.3. Объект Link

HTML-ссылки — основные элементы любого документа сети, позволяющие переходить на другие Web-страницы с помощью простого щелчка. Расположение документа несущественно; он может находиться на том же самом сервере либо за тысячи километров от него. Все, что требуется обеспечить — это допустимость данного URL. JavaScript-эквивалент гипертекстовой ссылки — объект **Link,** который определяется в HTML-синтаксисе так:

<a href=”адрес файла или URL ”

[name="Имя объекта"]

[target=" Имя окна"]

[onclick="Имя метода "]

[onmouseover= " Имя метода "] >

linkText

Пример

Вызов по ссылке файла 2.html (файл находится в том же каталоге иначе нужно указывать полный адрес):

```
<html>
<head><title> пример использования ссылки </title></head>
<body>
Текст до ссылки.
<a href="2.html">
 Ссылка
```

```
</a>
</body>
</html>
```

Задание для самостоятельного выполнения

Создать страницу с заголовками, ссылками, заданными цветами фона документа, ссылок и текста, а так же с диалоговыми окнами либо alert(), либо prompt().

Тема 6. Форма. Часть 1

Объект **Form** — это контейнер, содержащий введенные пользователем данные. Он может содержать множество объектов, включая объекты Text, Button, Radio и Select.

В HTML объект Form определяется так:

```
<FORM
[NAME="Имя формы" ]
[ACTION="URL сервера (адрес)" ]
[ENCTYPE="Тип кодировки " ]
[METHOD=GET | POST]
[TARGET=" Имя окна" ]
[ onSubmit= "Имя метода" ] >
</FORM>
```

Многие из свойств объекта **Form** работают с дополнительной информацией, которую форма отправляет на сервер. Можно отправлять форму, используя один из двух процессов. Во-первых, можно вызвать метод **submit()** объекта **Form,** а во-вторых — нажать на кнопку "Submit", которая автоматически отправит соответствующую ей форму.

- **action** (то же, что и параметр **ACTION=**). Свойство **action** определяет URL сервера, куда отправляется форма.

Form1.action=http://www.acadians.com/js/surv.cgi

- **method** (то же, что и параметр **METHOD=).** Свойство **method** определяет, каким образом форма отправляется серверу. Наиболее часто применяется значение GET, однако, можно использовать также и **POST.** Этот параметр основан на процессе со стороны сервера, поэтому при

проектировании HTML- форм необходимо учитывать требования программы сервера.

- **target** (то же, что и параметр **TARGET=).** Свойство **target** определяет целевое окно, которому сервер должен послать ответную информацию. Если свойство **target** не определено, сервер отображает результаты в окне, отправившем форму.

Form действует как контейнерный объект для всех других объектов формы. Объект **Form** имеет свойство **elements,** которое применяется для ссылки на элемент формы или перебора всех элементов в форме для специфических задач. Порядок массива полностью основан на порядке следования элементов HTML-формы в исходном файле. Первый перечисленный элемент — **elements[0],** второй — **elements[1]** и т.д. Можно ссылаться на каждый элемент **Form** с помощью имени либо его индекса в массиве элементов. Например, если объект **Text** с именем **LastName** является первым элементом в форме, на него можно ссылаться с помощью следующего кода:

```
custLastName = Forml.elements [0].value
```

Кроме того, допускается также и такой код:

```
custLastName = Forml.LastName.value
```

Свойство length определяет количество элементов в форме:

```
document.forml.elements.length
```

6.1. Поля ввода данных

Для большинства задач объект **Text** — это элемент, который наиболее часто используется для хранения данных, вводимых пользователем. Объект **Text** используется для захвата однострочной произвольной информации. Имеет следующий синтаксис:

<INPUT

TYPE="text"

[NAME="Имя объекта "]

[VALUE="Значение в поле ввода "]

[SIZE= размер]

[MAXLENGTH= размер]

[onBlur="Имя метода "]

[onChange=" Имя метода "]

[onFocus=" Имя метода "]

[onSelect=" Имя метода "] >

Например, чтобы определить объект Text для последнего имени, используют следующее:

<INPUT TYPE="text" NAME=LastName SIZE=20 MAXLENGTH=25>

6.2. Многострочные поля ввода данных

Можно использовать объект Textarea для ввода данных произвольной формы, занимающих несколько строк. Объект Textarea определяется с помощью стандартного синтаксиса HTML:

<TEXTAREA

NAME=" Имя объекта"

ROWS="число строк "

COLS="число столбцов "

[WRAP="off|virtual|physical"]

[onBlur ="Имя метода"]

[onChange="Имя метода"]

[onFocus="Имя метода"]

[onSelect="Имя метода"] >

Текст на мониторе

</TEXTAREA>

Например, при использовании Textarea для передачи по сети комментариев объект определяется следующим образом:

<textarea name=" Comments" rows=12 cols=78>

</textarea>

Выполняется установка опции переноса текста с помощью параметра WRAP= HTML-дескриптора <TEXTAREA>.

31. off. По умолчанию в объекте Textarea текст не переносится.

12. virtual. Если параметр WRAP= установлен в virtual, строки переносятся на экране на границе области объекта Textarea, однако новая строка определяется только через нажатие клавиши "ENTER".

13. physical. Если параметр WRAP= установлен в physical, строки переносятся на экране, а при отправке данных на сервер в конце каждой экранной строки устанавливается символ возврата каретки.

Методы:

onBlur Убирает фокус из текстовой области.

onFocus Устанавливает фокус на текстовую область.

onSelect Выделяет текст в текстовой области..

onChange Вызывает обработчик для указанного события.

6.3. Использование кнопок на странице

В HTML существует три типа кнопок, которые можно использовать в формах: Button, Submit и Reset. Как видите, два из них являются специализированными формами универсального объекта Button. С помощью обычного HTML-синтаксиса кнопка определяется так:

```
< INPUT
TYPE = "button | submit | reset"
[NAME = "Имя кнопки"]
[VALUE = "Текст на кнопке"]
[OnClik = "Метод кнопки"] >
```

Эти три типа кнопок предназначены для различных целей:

• Кнопка Submit отправляет форму, в которой она содержится, на сервер, основываясь на параметрах формы.

• Кнопка Reset сбрасывает значения полей в текущей форме, восстанавливая ранее установленные значения по умолчанию.

• Объект Button — это универсальный объект без предварительно определенного встроенного поведения. При работе с этим объектом необходимо добавлять к кнопке обработчик событий onClick.

6.4.Строка ввода пароля

Объект **Password** преследует одну цель: захватывать значения пароля, введенного пользователем. Объект **Password** аналогичен объекту **Text,** но отображает любой символ, напечатанный пользователем в поле, в виде

звездочки (*). Этот объект определяется в синтаксисе HTML следующим образом:

```
<INPUT
TYPE="password"
[NAME="Имя объекта"]
[VALUE="defaultPassword"]
[SIZE= целое число]>
```

Ниже приведен пример:

```
<INPUT TYPE="password" NAME="passwordField" SIZE=15>
```

Задание для самостоятельного выполнения

Создать форму с полями ввода и кнопками, подсчитать количество элементов на форме.

Тема 7. Условные операторы

Оператор if применяется следующим образом:

```
if (условие) {
 [ операторы]
}
```

В качестве *условия* может указываться любое логическое выражение. Если результат *условия* равен **true,** выполняются *операторы,* посему работа программы продолжается. Если *условие* возвращает **false,** *операторы* игнорируются и работа продолжается. Обычно набор операторов заключается в фигурные скобки. Это придает сценарию логический вид и оказывается особенно полезным в случае вложенных операторов if (т.е. при использовании одного оператора if внутри другого).

Иногда одной конструкции **if** оказывается недостаточно. Зачастую требуется также зарезервировать набор операторов, которые будут выполняться в случае, когда условное выражение возвращает **false.** Это можно сделать, добавив еще один блок непосредственно после блока if:

```
if (условие) {
 операторы
} else {
 операторы
}
```

Кроме того, имеется возможность скомбинировать часть else с другим оператором if. Использование такого метода позволяет оценить несколько разных возможных сценариев перед выполнением нужной операции. Изящество данного метода состоит в том, что в конце можно определить тот же сегментом **else.** Формат упомянутого типа оператора выглядит так:

```
if (условие) (
 операторы
} else if (условие) {
 операторы
}else {
 операторы
}
```

Пример

Нахождение максимального числа:

```
<html>
  <head>
  <title> нахождение максимального числа </title>
  <script language="JavaScript">
    function max(obj){
        var a =1*obj.a1.value;
        var b =1*obj.b1.value;
        var c =1*obj.c1.value;
        var max=a;
        if(b>max) max=b;
        if(c>max) max=c;
        obj.max1.value=max;}
```

```
</script>
</head>
<body>
    <form name="form1">
   Число1: <input type="text"  name="a1" size="20">
   <p>Число2: <input type="text"  name="b1" size="20"></p>
   <p>Число3: <input type="text"  name="c1" size="20"></p>
   <p>максимальное число <input type="text"  name="max1" size="20"></p>
   <p><input type="button" value="ok" name="B1" onClick="max(form1)">
   <input type="reset" value="Cancel" name="B2"><br>
</form>
 </body>
</html>
```

При объявлении переменных применено действие умножения на 1 (var a =1*obj.a1.value;), что обеспечивает преобразование пользовательских строковых переменных a1, b1, c1 к численным переменным a,b,c.

Задание для самостоятельного выполнения

1. Измените пример так, чтобы находилось минимальное число.

2. Создать страницу, содержащую диалоговой панелью с выбором о закрытии окна (метод confirm()). По нажатию кнопки Ok окно должно закрываться, а по нажатию кнопки Отмена выдавать сообщение о продолжении программы.

Тема 8. Циклы

Оператор for используется для организации цикла в сценарии. Рассмотрим его синтаксис:

```
for ([ выражение_инициализации] ;
[выражение_условия]; [выражение_цикла]) {
операторы
}
```

Три выражения, заключенные в квадратные скобки, не являются обязательными, тем не менее, если даже убрать одно из них, точка с запятой все же потребуется. Именно точка с запятой обеспечивает должное место для каждого выражения. Выражение инициализации обычно применяется в целях инициализации переменной и даже объявления, что конкретная переменная является счетчиком цикла. Выражение условия должно иметь значение **true** перед каждым выполнением операторов, заключенных в фигурные скобки. Наконец, выражение цикла, как правило, инкрементирует или декрементирует значение переменной, которая используется в качестве счетчика цикла.

Пример 1

Нахождение чисел в интервале, делящихся без остатка на 7:

```
<html>
  <head>
    <title>Числа делящиеся на 7</title>
    <script language="JavaScript">
```

```
        function res(obj)    {
              var m = 1*obj.m1.value;
               var n = 1*obj.n1.value;
              var res = " ";
              for (var i = m; i <=n; i++)  {
                  if (i % 7 == 0) {
                      res = res +" " + i;
                  }
              }
              if (res == " ") {
                res = "таких чисел нет";
              }
               obj.res1.value = res;
        }
    </script>
  </head>
  <body>
        <form name="form1">
         Введите начало интервала: <input type="text" name="m1" size="20"><br>
         Введите конец интервала: <input type="text" name="n1" size="20"><br>
         <p>Числа делящиеся на 7: <input type="text" name="res1" size="30"></p>
          <p><input onclick="res(form1)" type="button" name="B1" value="ok"> </p>
         <input type="reset" name="B2" value="Cancel"><br>
        </form>
    </body>
</html>
```

Подобно операторам **if,** циклы **for** также могут быть вложенными.

Пример 2

Создание матрицы 10x10:

```
<html>
  <head>
   <title>Вложенные циклы </title>
  </head>
  <body>
   <script>
     document.write("Все координаты x,y между (0,0) и (9,9) :<br>") ;
     for (var x = 0; x < 10; ++x) {
         for (var y = 0; y < 10; ++y) {
            document.write("(" +x+ "," +y+ "),");
         }
         document.write('<br>');
     }
     document.write("<br>После завершения цикла x равно : " +x) ;
     document. write ("<br>После завершения цикла y равно : " +y) ;
   </script>
  </body>
</html>
```

Оператор **while** действует подобно циклу **for,** однако не включает в себя функции инициализации и приращения переменных во время их объявления. Переменные необходимо объявлять заранее, а инкремент или декремент их определять в рамках блока операторов. Рассмотрим синтаксис **while:**

```
while (выражение_условия) {
операторы
}
```

Пример 3

Суммирование целых чисел от 0 до 10:

```
<html>
  <head>
   <title>JavaScript Unleashed</title>
  </head>
  <body>
    <script type="text/javascript">
       var i = 1;
       var result = 0;
       var status = true;
       document.write("0");
       while(status){
          document.write(" + " + i) ;
          if(i == 10) {status = false;}
          result +=i++;
        }
        document.writeln(" = " + result);
    </script>
  </body>
</html>
```

Конструкция **do..while** работает подобно оператору **while** за исключением того, что условное выражение не проверяется вплоть до завершения первой итерации. Этот способ гарантирует, что набор

операторов, находящихся в пределах фигурных скобок, будет выполнен, по крайней мере один раз.

Пример 4

Использование операторов **while** и **do..while**:

```
<html>
 <head>
 <title>Пример while</title>
  <script>
   count = 1;
   while (count < 5) {
      document.write ("Счетчик" + count + "<BR>");
      count ++;
   }
   document.write ("<BR><BR>");
   count = 0;
   while (count < 5) {
      count ++;
      document.write ("Счетчик" + count + "<BR>");
   }
   document.write ("<BR><BR>");
   count = 0;
   do {
      document.write ("Счетчик" + count + "<BR>");
      count ++;
   }while (count <= 5)
  </script>
 </head>
```

```
  <body>
  </body>
</html>
```

Операторы **break и continue** используются для прерывания любого цикла, **break** полностью завершает цикл, а **continue** останавливает текущую итерацию цикла, при этом программа переходит на начало следующей итерации.

Оператор **switch** позволяет определять все возможные значения данной переменной для исполнения всех необходимых задач. Если текущее значение не равно одному из определенных заранее, программа переходит к разделу **default.** Синтаксис оператора:

```
switch (выражение) {
 case N1 : операторы; break;
 case N2 : операторы; break;
 ......
 default: операторы;
}
```

Пример 5

Определение дня недели по введенному числу:

```
<html>
  <head>
    <title> Календарь </title>
    Пример определения дня недели по введеному числу
```

```
  <script language="JavaScript">
     function numday(obj){
       var n=obj.n1.value;
       var s;
       switch (n) {
         case "1" : s="понедельник"; break;
         case "2" : s="вторник"; break;
         case "3" : s="среда"; break;
         case "4" : s="четверг"; break;
         case "5" : s="пятница"; break;
         case "6" : s="суббота"; break;
         case "7" : s="воскресение"; break;
         default: s="ошибка";
       }
       obj.s1.value=s;
     }
   </script>
 </head>
 <body>
    <form name="form1">
     Число: <input type="text" name="n1" size="20">
     <p>название дня недели: <input type="text"  name="s1" size="20"></p>
     <p><input type="button" value="определить"  onClick="numday(form1)"></p>
     <p><input type="reset" value="Отмена"></p>
    </form>
 </body>
</html>
```

Задание для самостоятельного выполнения

1.Написать программу вычисления факториала с использованием цикла **for**.

2. Написать программу вычисления факториала с использованием циклов **while** и **do..while**.

3. Создать калькулятор, т.е. два поля ввода чисел, поле выбора знака (+, -, *, /), вывод результата, кнопки обновления данных и калькуляции.

Тема 9. Массивы

Таблица 9.1

Методы и свойства объекта Array.

Тип	*Элемент*	*Описание*
Метод	concat()	Соединяет элементы существующего массива.
	join()	Соединяет все элементы массива в одну строку.
	pop()	Удаляет последний элемент массива.
	push()	Добавляет элементы в конец массива.
	reverse()	Изменяет порядок следования элементов в массиве.
	shift()	Удаляет элементы в начале массива.
	slice()	Возвращает часть массива.
	sort()	Сортирует элементы в массиве.
	splice()	Вставляет и удаляет элементы из массива.
	toSource()	Преобразует элементы в строку с квадратными скобками.
	toString()	Преобразует элементы массива в строку.
	unshift()	Добавляет элементы в начало массива.
	valueOf()	Возвращает массив элементов, отделенных запятыми.
Свойство	index	Возвращает индекс совпадения в строке для массива, созданного в соответствие с регулярным выражением.
	input	Для массива, созданного в соответствие с регулярным выражением свойство возвращает исходную строку.
	length	Количество элементов в массиве.

	prototype	Позволяет добавлять свойства к экземплярам объекта **Array.**

Экземпляр объекта Array создается при помощи оператора new и операторов, заполняющих массив элементами данных. Например, код должен выглядеть следующим образом:

```
var coffee = new Array ();
coffee[0] ="Ethiopian Sidamo";
coffee[l] ="Kenyan";
coffee[2] ="Cafe Verona";
coffee[3] ="Sumatra";
coffee[4] ="Costa Rica";
coffee [5] ="Columbian";
coffee[6] ="Bristan";
```

Альтернативой определению объекта **Array** может быть указание элементов данных как параметров запроса **new.** Следующая строка является функциональным эквивалентом предыдущего примера.

```
var coffee = new Array("Ethiopian Sidamo","Kenyan", "Cafe Verona", "Sumatra", "Costa Rica", "Columbian", "Bristan");
```

Обратите внимание, что размер массива явно не задается; это не характерно для многих языков программирования. В JavaScript определение размера массива не обязательно. Именно это свойство позволяет расширить массив, добавляя новые элементы данных. С другой стороны, если необходимо, размер массива можно задать явно в new:

```
var coffee = new Array (7) ;
```

Изменение размеров массива также можно осуществить путем определения элемента данных в позиции n. Если n превышает существующее количество элементов в массиве, размер массива увеличивается до n+1. Посмотрите на следующий пример:

```
var javaDrinks = new Array();
javaDrinks[0] = "Regular coffee";
javaDrinks[1] = "Decaf coffee";
javaDrinks[2] = "Cafe Mocha";
j avaDrinks [ 3 ] = "Cafe au Lait";
javaDrinks[199] = "Cafe Latte" ;
```

Размер массива **javaDrinks** будет составлять 200, даже притом, что определено только 5 элементов данных. При доступе к неопределенным элементам возвращается значение **null.**

Выяснить размер массива поможет свойство **length** объекта Array. Свойство **length** объекта **Array** предназначено только для чтения. Отсюда вывод — изменить размеры массива за счет присваивания нового значения свойству **length** нельзя.

Пример 1

Таблица, заполненная с помощью массива:

```
<html>
  <head>
  <title>Таблица</title>
  </head>
```

```
  <body>
      <Table name='tb' id='tb' border=1></Table>>
      <script>
          arData = new Array(3);
          arData[0] =  new Array("1","2","3","4","5");
          arData[1] =  new Array("a","b","c","d","f");
          arData[2] =  new Array("а","б","в","г","д");
          for (i=0; i < arData.length; i++) {
               tb.insertRow(i);
               for (j=0; j < arData [i].length; j++) {
                    tb.rows[i].insertCell(j);
                    tb.rows[i].cells[j].width = 50;
                    tb.rows[i].cells[j].align ='center';
                    tb.rows[i].cells[j].innerText = arData [i] [j];
               }
          }
       </script>
  </body>
</html>
```

Пример 2

Применение методов и свойств массива:

```
<html>
<head>
<title>Массив</title>
<script>
 function find_month() {
 month = prompt ("Введите месяц: ");
```

```
if (month) {
var i = NaN;
list = months.join ("$");
position = list.indexOf(month);
if (position > -1) {
var index = 0;
if (position) {
sub_list = list.substring (0, position);
sub_array = sub_list.split ("$");
index = sub_array.length - 1;
}
alert("Месяц "+ months[index] +" в массиве находится под индексом " + index);
}
else
alert("Месяц "+ month +" в массиве отсутствует " )
}
}
</script>
</head>
<body>
<script>
months = new Array ("январь", "февраль", "март", "апрель", "май", "июнь", "июль", "август", "сентябрь", "октябрь", "ноябрь");
document.write("Исходный массив: ", months.toString(), "<br><br>")
months[11] = "декабрь";
document.write("Новый массив: ", months.join("; "), "<br><br>");
spring_months = months.slice(2,5);
autumn_months = months.slice(8,11);
offseason = spring_months.concat(autumn_months);
```

```
                document.write("весенние месяцы: ", spring_months.join("; "), "<br><br>");
                document.write("осенние месяцы: ", autumn_months.join("; "), "<br><br>");
                document.write("весенние и осенние месяцы: ", offseason.join("; "),
"<br><br>");
                months.reverse();
                document.write("Инвертированный список: ", months.join("; "), "<br><br>");
                months.sort();
                document.write("месяцы в алфавитном порядке: ", months.join("; "),
"<br><br>");
        </script>
        <form>
        <input type ='button' value ='Найти месяц' onClick='find_month()'>
        </form>
       </body>
       </html>
```

Задание для самостоятельного выполнения

1.Создать массив чисел, вывести его на экран, найти максимальное число из массива.

2. Выбрать зимние и летние месяцы и объединить их, удалить по элементу массива с начала и в конце, добавить несколько других элементов с обоих концов, вывести количество элементов нового массива.

Тема 10. Объект Date

С помощью объекта **Date** в JavaScript можно работать со значениями даты и времени. Но перед тем как приступить к работе с этим объектом, необходимо разобраться в трех важных вещах:

• В соответствие с соглашением UNIX, датой появления JavaScript считается 1 января 1970 г. Соответственно, с датами, предшествующими указанной, работать нельзя.

• При создании объекта **Date** базой, на которой основывается отсчет времени внутри объекта, является клиентская машина. Таким образом, на ней желательно наличие работоспособных и точных часов. Учитывайте это при написании JavaScript-сценариев, чувствительных ко времени.

• JavaScript отслеживает значения даты и времени в форме миллисекунд с момента даты основания (1/1/1970).

Таблица 10.1

Методы и свойства объекта Date

Метод	getDate()	Возвращает день месяца (от 1 до 31).
	getDay()	Возвращает день недели (от 0 до 6).
	getFullYea()	Возвращает год в четырех символах по местному времени.
	getHours()	Возвращает час суток (от 0 до 23).
	getMilliseconds()	Возвращает миллисекунды.
	getMinutes()	Возвращает минуты в пределах часа (от 0 до 59).
	getMonth()	Возвращает месяцы года (от 0 до 11).

	getSeconds()	Возвращает секунды (от 0 до 59).
	getTime()	Возвращает количество миллисекунд, начиная с 1/1/1970 00:00:00.
	getTimeZoneOffset()	Возвращает смещение часового пояса в минутах по отношению к GMT/UTC.
	getUTCDate()	Возвращает день месяца.
	getUTCDay()	Возвращает день недели, преобразованный к универсальному времени.
	getUTCFullYear()	Возвращает четырехзначное представление года, преобразованное к универсальному времени.
	getUTCHours()	Возвращает часы, преобразованные к универсальному времени.
	getUTCMilliseconds()	Возвращает миллисекунды, преобразованные к универсальному времени.
	getUTCMinutes()	Возвращает минуты, преобразованные к универсальному времени.
	getUTCMonth()	Возвращает месяц, преобразованный к универсальному времени.
	getUTCSeconds()	Возвращает секунды, преобразованные к универсальному времени.
	getYear()	Возвращает номер года, начиная с 1900 г.
	parse()	Преобразует строковые данные в миллисекунды.
	setDate()	Устанавливает день месяца (от 1 до 31).
	setFuNYear()	Устанавливает год как четырехзначное число.

	setHours()	Устанавливает часы дня (от 0 до 23).
	setMilliseconds()	Устанавливает миллисекунды.
	setMinutes()	Устанавливает минуты в часе (от 0 до 59).
	setMonth()	Устанавливает месяц в году (от 0 до 11).
	setSeconds()	Устанавливает секунды в минуте (от 0 до 59).
	setTime()	Устанавливает секунды в минуте (от 0 до 59).
	setUTCdate()	Устанавливает количество миллисекунд, начиная с 1/1/1970 00:00:00.
	setUTCFullYear()	Устанавливает день месяца в соответствии с универсальным временем.
	setYear()	Устанавливает количество лет, начиная с 1900 г.
	toGMTString()	Преобразует дату в строку в соответствии с мировым форматом.
	toLocalString()	Возвращает строку даты в формате локальной системы.
	toSource()	Возвращает источник-объект Date.
	toString()	Возвращает дату и время как строку в соответствии с местным временем.
	ToUTCString()	Возвращает данные и время как строку в соответствии с мировым временем |UTC).
	UTC()	Преобразует значения с разделителями-запятыми в миллисекунды даты по UTC.
	valueOf()	Возвращает эквивалент объекта Date в миллисекундах.
Свойство	prototype	Свойство, позволяющее добавлять методы и свойства к объекту Date.

Методы getUTC… , setUTC… возвращают или устанавливают метод относительно универсального времени.

Таблица 10.2

Параметры, необходимые для создания экземпляра объекта Date.

Параметр	Описание	Пример
Отсутствие параметров	Создает объект с текущими датой и временем.	var today = new Date()
«месяц дд, гггг чч:мм: С»	Создает объект с указанными датой (дд — день, гггг — год) и временем (чч — часы, мм — минуты, ее — секунды). При этом все пропущенные значения считаются нулевыми.	var someDate = new Date (“September 27, 2000”)
гг, мм, дд	Создает объект с указанными датой из набора целочисленных значений (гг — год, мм — месяц, дд — день)	var someDate = newDate(00, 1, 0)
гг, мм, дд, чч, мм, сс	Создает объект с указанными датой и временем из набора целочисленных значений(гг — год, мм — месяц, дд — день, чч — часы, мм — минуты, се — секунды). При этом все пропущенные значения считаются нулевыми.	var someDate = new Date(00, 7, 24, 6, 29, 50)

После создания объекта можно использовать все его методы для получения или установки значения даты. Например, для возврата текущей даты следует записать:

```
var today = new Date () ;
result = today.getDate();
```

Для изменения месяца, определенного в объекте appt типа **Date** потребуется записать:

```
var appt = new Date (2000,10,20) ;
result = appt.setMonth (7) ;
```

Пример

Выведение на страницу даты и времени:

```
<html>
 <head>
  <title>Untitled Document</title>
  <script language=”JavaScript”>
   nmonths=new Array (“январь”, “февраль”, “март”, “апрель”, “май”, “июнь”, “июль”, “август”, “сентябрь”,  «октябрь», «ноябрь», «декабрь»);
   ndays=new Array («воскресение», «понедельник», «вторник», «среда», «четверг», «пятница», «суббота»);
   function DateTime(){
      var now=new Date()
      var minute=now.getMinutes()
      var second=now.getSeconds()
      var hour=now.getHours()
      var min=((minute<10)?”:0”:”:”)+minute
      var sec=((second<10)?”:0”:”:”)+second
      var str=” “
      year=now.getYear()+””
      month=now.getMonth()
      imonth=nmonths[month]
      day=now.getDay()
      iday=ndays[day]
      str+=now.getDate()+” “+imonth+” “+year+” года<br>”
```

```
        str+=" "+hour+min+sec+"<br>"
        str+="сегодня – "+iday
        document.write(str)
    }
  </script>
 </head>
 <body>
  <center>
  <script language="JavaScript">
    DateTime()
  </script>
 </body>
</html>
```

Задание для самостоятельного выполнения

Вывести на страницу дату и время, преобразованные к универсальным; поменять дату и время на произвольные (заданные преподавателем).

Тема 11. Форма. Часть 2

11.1 Переключатель

Объект **Radio** используется для предоставления пользователям возможностей выбора одиночной опции из группы параметров. Если выбирается одна опция в пределах набора, другие в тот же момент выбраны быть не могут. Щелчок на переключателе снимает выделение с любого другого выбранного ранее переключателя.

```
<INPUT
TYPE=”radio”
[NAME=”Имя группы” ]
[VALUE=” value”]
[CHECKED]
[onClick=”Имя Метода”]>
[Текст на мониторе]
```

Способ группировки элементов основан на параметре NAME= переключателя. Каждый элемент в объекте **Radio** должен использовать в этом параметре одно и то же значение.

Пример 1

Вычисление площади выбранной правильной фигуры:

```
<html>
  <head>
  <title>Переключатель</title>
```

```
<script language="JavaScript">
  function test(k){
      var a=1*form1.a1.value;
      if (a!="")
        {res=k*a*a;}
      else alert("Введите число");
      form1.res1.value=res;
  }
</script>
</head>
<body>
<form name="form1">
  Введите число
<input type="text" name="a1" size=10><hr>
Выберете фигуру <hr>
<input type="radio" name="k" value=1 onClick="test(1)"> квадрат<br>
<input type="radio" name="k" value=3.14 onClick="test(3.14)"> круг<br>
<input type="radio" name="k" value=0.433 onClick="test(0.433)">равносторонний треугольник<br>
Площадь:  <input type="text" name="res1" size=10>
<input type="reset" value="Отмена">
</form>
</body>
</html>
```

Одной из наиболее распространенных задач, которые должны решаться при использовании объекта **Radio,** является проблема получения значения выбранного в данный момент переключателя. Для этого потребуется определить, какой из переключателей выбран, и затем вернуть его значение. Вместо написания оригинального кода каждый раз, проще воспользоваться универсальной функцией, называемой **getRadioValue()**, которая может

возвращать значение объекта **Radio,** передаваемого в метод в качестве параметра.

Пример 2

Определение позиции выбранного переключателя:

```
<html>
  <head>
    <script language=”JavaScript” >
      function getRadioValue(radioObject) {
        var value = null
        for (var i=0; i<radioObject.length;  i++) {
          if (radioObject[i].checked) {
            value = radioObject[i].value
          break }
        }
        return value
      }
    </script>
  </head>
  <body>
    <form name=”form1”>
      <p><input type=radio name=”group” value=”Алиса”>Алиса</p>
      <p><input type=radio name=”group” value=”ДДТ”>ДДТ</p>
      <p><input type=radio name=”group” value=”Чайф”>Чайф</p>
      <input type=button value=”Показать выбранную группу”  onClick =
“alert(getRadioValue(this.form.group))”>
    </form>
  </body>
```

</html>

Задание для самостоятельного выполнения

Создать калькулятор с выбором действия через переключатель.

11.2. Флажки

Объект **Checkbox** — это тот же объект **Form,** но предназначенный, прежде всего для обозначения логических данных **(true** или **false). Он** действует как переключатель, который может быть включен или выключен пользователем либо JavaScript-кодом. Для определения **Checkbox** используют следующий HTML-синтаксис:

```
<INPUT
TYPE=»checkbox»
[NAME=” Имя объекта” ]
[VALUE=” value”]
[CHECKED]
[onClick=»Имя метода «] >
[Текст на мониторе]
```

Следующее соглашение для пользовательского интерфейса связано с тем, что **Checkbox** не должен вызывать выполнение какого-либо действия (как это делают объекты **Button).** В результате его обработчик событий **onClick,** вероятно, будет применяться не столь широко. Исключением из этого правила является изменение состояния других объектов формы.

Возможно, наиболее важным свойством объекта **Checkbox** является свойство **checked.** Это свойство проверяется для определения, выставил ли пользователь флажок. Для проверки объекта **Checkbox** использовать свойство **value** нельзя.

Пример

Вывод выбранных флажков:

```
<html>
 <head>
 <title>Флажки</title>
 </head>
 <body>
    <script language=”JavaScript”>
        function chek()  {
            var a=form1.C1.checked;
    var b=form1.C2.checked;
    var c=form1.C3.checked;
    var d=form1.C4.checked;
            var res=””;
     if(a == true)
              { res = res+” Выбран флажок 1; “;}
            if(b == true)
              { res = res+” Выбран флажок 2; “;}
    if(c == true)
              { res = res+” Выбран флажок 3; “;}
    if(d == true)
              { res = res+”Выбран флажок 4. “;}
            form1.res1.value=res;
```

```
    }
  </script>
  <form name=”form1”>
      <p><input type=”checkbox” name=”C1” value=”Флажок1”>Флажок1</p>
        <p><input type=”checkbox” name=”C2” value=”Флажок2”>Флажок2</p>
        <p><input type=”checkbox” name=”C3” value=”Флажок3”>Флажок3</p>
        <p><input type=”checkbox” name=”C4” value=”Флажок4”>Флажок4</p>
        <input type=”button” value=”Выполнить” onClick=”chek()”>
        <input type=”text” name=”res1” size=”80”>
        <input type=”reset” value=”Отмена”>
  </form>
 </body>
</html>
```

Задание для самостоятельного выполнения

Создать группу переключателей по цветам и группу флажков для фруктов, определить, по нажатию кнопки, какой цвет и какие фрукты выбраны.

11.3. Списки

Объект **Select** — один из наиболее полезных и гибких объектов **Form.** Базовый синтаксис HTML для объекта **Select** выглядит так:

```
<SELECT
[NAME=»Имя объекта «]
[SIZE=»размер списка « ]
```

[MULTIPLE]

[onBlur=»Имя метода «]

[onChange=» Имя метода «]

[onFocus=» Имя метода»]>

<OPTION VALUE=”optionValue”

[SELECTED] >Текст на мониторе </OPTION>

[<OPTION VALUE=”optionValue”> Текст на мониторе</OPTION>]

</SELECT>

Благодаря своей гибкости, объект **Select** может принимать три различных формы: список выбора, прокручиваемый список и прокручиваемый список с мульти выбором.

- Список выбора — это выпадающий список опций, в котором пользователь может выбрать только одну опцию. Список выбора обычно отображает одно значение на экране, но при нажатии на стрелку список расширяется и отображает все элементы выбора. В отличие от комбинированных списков Windows, вводить значение в список выбора не допускается; можно только выбирать элемент из существующего С ссива элементов. Ключ к определению списка выбора заключается в присвоении параметру SIZE значения 1 (либо вообще никакого значения). Это даст гарантию того, что список будет отображать только одну строку.
- Второй формой объекта Select является прокручиваемый список, во многих средах называемый *окном списка.* Он отображает указанное число элементов одновременно в формате списка. При этом пользователь может прокручивать список элементов вверх или вниз, чтобы просматривать и те элементы, которые не уместились в окне. Для создания прокручиваемого списка единственное изменение в HTML-определении <select> необходимо проделать в параметре SIZE=. Если его значение больше 1, объект Select преобразуется в прокручиваемый список.

- Последняя форма, в которой может быть представлен объект Select, — это прокручиваемый список с мульти выбором. Он выглядит так же, как обычный прокручиваемый список, но ведет себя по-другому. Из этой формы объекта Select можно выбирать один и более элементов. Решение задачи выбора множества элементов зависит от среды. В большинстве случаев можно либо перемешать мышь с нажатой кнопкой поверх требуемых элементов, либо удерживать в нажатом состоянии клавишу Shift или Ctrl во время щелчков на элементах списка. Для определения прокручиваемого списка с мульти выбором необходимо в определение объекта Select добавить параметр MULTIPLE.

В списке выбора или прокручиваемом списке можно определять значение выбранной опции с помощью комбинации свойств **options и selectedlndex** объекта **Select.** Например, если требуется определить песню, которая была выбрана в примере объекта **Songs,** используют следующий код:

```
favorite = document.form1.songs.options
[document.forml.songs.selectedIndex].value
```

Свойство **options** — это массив, содержащий каждую опцию, определенную внутри объекта **Select.** Используя это свойство, можно обращаться к свойствам каждой опции. С помощью свойства **Selectedlndex** возвращается индекс выбранной опции. Когда **options и Selectedlndex** используются в комбинации, возвращается значение выбранной в данный момент опции. С учетом требований точечной нотации JavaScript, выбранное в данный момент значение может включать длинные строки текста программы. Этого можно избежать с помощью универсального метода **getSelectValue().**

```
<html>
 <head>
```

```
  <script language = “JavaScript”>
    function getSelectValue(selectObject) {
      return selectObject.options[selectObject.selectedIndex].value
    }
  </script>
 </head>
 <body>
  <form name=”form1”>
    Песня: <select NAME=”songs” SIZE=1>
    <option VALUE=”Ветер”>Ветер</option>
    <option VALUE=”Метель августа”>Метель августа</option>
    <option VALUE=»Последняя осень»>Последняя осень</option>
    <option VALUE=»Ночь Людмила»>Ночь Людмила</option>
    <option VALUE=»Майский дождь»>Майский дождь</option>
    <option VALUE=»Это все»>Это все</option>
    </SELECT><p>
    <input type=button value=”Выбор песни” onClick = “alert(getSelectValue(this.form.songs))”>
  </form>
 </body>
</html>
```

При использовании списка с мульти выбором необходимо применять свойство **selected** массива опций для определения состояния каждой опции в списке.

Пример 1

Выпадающий список газет:

```
<html>
<head>
```

```
<title>Список газет</title>
 <script>
 links = new Array();
        links[0] =”http://www.gaseta.ru/”;
        links[1] =”http://www.utro.ru/”;
        links[2] =”http://www.kp.ru/”;
        links[3] =”http://www.eg.ru/”;
        links[4] =”http://ng.apress.ru/”;
        links[5] =”http://www.hockeynews-online.com/”;
 function relocate(){
         choice = document.form1.linklist.selectedIndex;
         window.location = links[choice];
 }
 </script>
</head>
<body>
 <form name = “form1”>
 <select name = “linklist”>
 <option value=0>Газета.Ru</option>
 <option value=1>Утро.Ru</option>
 <option value=2>Комсомольская правда</option>
 <option value=3>Экспресс газета</option>
 <option value=4>Наша газета</option>
 <option value=5>Футбол и хоккей</option>
 </select>
 <input type = “button” value = “Перейти по ссылке” onClick = “relocate()”>
 </form>
</body>
</html>
```

Пример 2

Список авиарейсов с мульти выбором:

```
<html>
<head>
<title>Рейсы</title>
 <script>
 function show_list(){
        document.form1.selection.value = “”;
        for (i=0; i<document.form1.flights.length; i++){
        if (document.form1.flights[i].selected)
                document.form1.selection.value =document.form1.selection.value +
(document.form1.flights[i].text + “. “+
 document.form1.flights[i].value + “\n”);
        }
       }
 </script>
</head>
<body>
 <H2>Авиарейсы</H2>
 <form name = “form1”>
 <P><Select name = “flights” multiple>
 <Option value = “Рейс SA0316, 8:05”>Москва->Киев</Option>
        <Option value = “Рейс SA2505, 8:37”>Москва->Ереван</Option>
        <Option value = «Рейс SA0180, 8:55»>Москва->Санкт-Петербург</Option>
        <Option value = «Рейс SA5513, 9:12»>Москва->Владивосток</Option>
        <Option value = “Рейс SA3346, 9:33”>Москва->Красноярск</Option>
        <Option value = “Рейс SA0410, 10:00”>Москва->Киев</Option>
 </Select>
```

```
<Input type = “button” value = “Показать выбранные” onClick = show_list()> </P><P>
<textarea name = “selection” id = “selection” rows = 5 cols = 50>
</textarea>
</form>
</body>
</html>
```

Задание для самостоятельного выполнения

Создать список любых предметов и преобразовать выпадающий список в список с прокруткой, в список с мульти выбором; обеспечить показ выбранного элемента.

Тема 12. Работа с графикой

Всего для визуальных эффектов накатов используется четыре типа событий. Обработчики событий активизируется при следующих действиях:

- *onMouseDown* – щелчок кнопки мыши на элементе;
- *onMouseUp* – отпускание кнопки после щелчка на элементе;
- *onMouseOver* – наведение указателя мыши на элемент;
- *onMouseOut* – перемещение указателя мыши с элемента.

Графические изображения вставляются в *Web* – страницы с помощью одинарного тега <*IMG*>.

```
<img
 [name=»Имя рисунка»]
src=» Адрес файла изображения»
[height=»Ширина «]
[width=»Высота «]
[alt=»Альтернативный текст «]
[border=»Толщина границы «]
[align=»left»|»right» | «top»|»absmiddle»|»absbottom»|»texttop»|»middle»|»baseline»|
«bottom»]
[hspace=»Расстояние до текста страницы по горизонтали «]
[wspace=» Расстояние до текста страницы по вертикали «]>
```

Обязательным является только атрибут src. Рисунки должны располагаться либо в том же каталоге, что и файл *Web* – страницы, либо в

атрибуте src должен быть указан полный адрес рисунка. Высота и ширина рисунка указываются в пикселях. Атрибут alt выводит строку текста на месте графического изображения при ускоренном скачивании страницы. Атрибут align позволяет управлять относительным местоположением изображения и «обтекающего его текста».

Например, для вывода изображения dot.gif в *HTML* – коде необходимо воспользоваться следующим синтаксисом:

```
<img src=”dot.gif” height=”200” width=”200”>
```

В графическом редакторе создайте рисунок ris.gif .

Пример

Визуальное удаление изображения:

```
<html>
  <head>
   <title> Пример визуального удаления изображения </title>
   <script language=”JavaScript”>
        function picture(){
             var w =document.ris.width;
             if (w>150) {document.ris.width=w-10;
                  document.ris.src=”ris.gif”;
                  setTimeout(“picture()”,500);}}
   </script>
 </head>
 <body>
    <img src=”ris1.gif” name=ris onMouseOver=”picture()”>
 </body>
```

```
</html>
```

Задание для самостоятельного выполнения

Создать страницу с примером визуального приближения изображения.

Тема 13. Объект Frame

Фрейм (frame) — это подокно полного окна браузера; его спецификации определяются разработчиком. Фреймы позволяют одновременно в одном и том же окне отображать сразу несколько документов.

Объект **Frame,** по сути, аналогичен объекту **Window,** и с ним работают подобными же методами. В случае одиночного окна объект **Window** представляет собой объект самого высокого уровня. В случае фреймов окно самого высокого уровня считается родительским, а его дочерние окна считаются объектами **Frame.**

Фреймы создаются с помощью дескрипторов <FRAMESET> и определяются дескрипторами <FRAME>.

```
<html>
<head>
<title>creating frames</title>
</head>
<frameset cols=”60%,*,5%”>
<frame src=”docl.html” name=”framel”>
<frame src=”doc2.html” name=”frame2”>
<frame src=”doc3.html” name=”frame3”>
</frameset>
</html>
```

За счет добавления нескольких строк кода можно легко создать окно с тремя фреймами:

```
<html>
<нead>
<title>window frames: window one</title>
```

```
</head>
<frameset rows=”45%,45%” cols=”60%,60%” onload=”alert(‘windows have frames’)”>
<frame src=”docl.html” name=”framel”>
<frame src=”doc2.html” name=”frame2”>
<frame src=”doc3.html” name=”frame3”>
<frame src=”doc4.html” name=”frame4”>
</frameset>
</html>
```

Дескриптор **<FRAMESET>** имеет атрибут для описания способа разделения окна на фреймы. Этим атрибутом может быть **COLS** или **ROWS** (для столбцов или строк), но не оба сразу. Разработчик определяет количество строк или столбцов фреймов в окне и размеры каждого фрейма. Для каждой строки или столбца может определяться, либо абсолютный размер в пикселях, либо относительный размер в процентах от окна, либо размер вычисляется как остаток от окна после установки других фреймов. Следующая строка создает два фрейма в виде столбцов, один размером 60% от окна, а другой — в оставшейся части окна (указан с помощью «*», в данном случае этот размер составляет 40%):

```
<frameset cols=»60%,*»>
```

Следующая строка кода создает снова два фрейма, но без использования знака процента. Абсолютная ширина первого фрейма — 60 пикселов:

```
<frameset cols=»60,*»>
```

Как и прежде, второй фрейм создается в оставшейся части окна.

Атрибуты дескриптора **<FRAME>** определяют документ, который будет загружаться в фрейм, имя фрейма, границы фрейма, полосу прокрутки и опции изменения размеров. Все следующие атрибуты дескриптора **<FRAME>** — необязательны:

• Атрибут SRC — URL (относительный или абсолютный) документа, который будет загружаться во фрейм. Документ может быть получен с того же сервера в виде фреймового файла или с другого сервера. Если атрибут SRC не используется, фрейм будет пустым. Это пустое пространство можно заполнить чем угодно, особенно, если для записи содержимого в фрейм воспользоваться JavaScript.

• Атрибут NAME, если определен, обеспечивает средства для ссылки на фрейм из другого фрейма, а также из JavaScript. JavaScript может также ссылаться на фрейм через массив **frames,** который рассматривается далее в главе. Значение атрибута **NAME** должно начинаться с алфавитно-цифрового символа.

• Границы фрейма устанавливаются с помощью двух атрибутов, **MARGINWIDTH и MARGINHEIGHT:**

• Атрибут **MARGINWIDTH** управляет боковыми границами фрейма. Этот атрибут может принимать значение в пикселях, начиная с единицы. Максимальное значение ограничено только размером фрейма. (Невозможно установить границы таким образом, чтобы не осталось места для отображения документа.)

• Атрибут **MARGINHEIGHT** аналогичен **MARGINWIDTH** за исключением того, что он управляет верхней и нижней границами.

• Если не указаны значения **MARGINWIDTH и MARGINHEIGHT,** браузер выбирает для них значения по умолчанию.

• Атрибут **SCROLLING** определяет полосу прокрутки для фрейма. Значениями для **SCROLLING** могут быть **YES, NO** и **AUTO.** YES означает, что полоса прокрутки всегда присутствует в фрейме. NO означает, что полоса прокрутки не отображается. AUTO отображает полосу прокрутки

только, Сли документ по размерам превышает фрейм. Значение по умолчанию для **SCROLLING — AUTO.**

• **NORESIZE** запрещает пользователю изменять размеры фрейма. Для этого атрибута значение не указывается. По умолчанию, если этот атрибут не определен, весь фрейм получит изменяемые размеры. Изменять размеры фреймов, которые имеют атрибут **NORESIZE,** нельзя.

Дескрипторы **<FRAMESET> и <FRAME>** обычно размещают в теле документа. Поскольку дескрипторы **<BODY>** необязательны, дескрипторы **<FRAMESET>** размещают после раздела **<HEAD>** документа, а дескрипторы **<BODY>** опускают, как показано ниже:

```
<html>
<head>
<title>frame demo</title>
</head>
<frameset cols=”60%,*”>
<frame src=”docl.html” name=”framel”>
<frame src=”doc2.html” name=”frame2”>
</frameset>
</html>
```

Многие авторы устанавливают также дескрипторы **<NOFRAMES>** для отображения сообщений в браузерах, не использующих фреймы. Содержимое может включать в себя как HTML-дескрипторы, так и текст. Поддерживающий фреймы браузер игнорирует информацию между дескрипторами **<NOFRAMES>** и **</NOFRAMES>,** тогда как браузер, не поддерживающий фреймы, игнорирует **<FRAMESET>, <FRAME> и <NOFRAMES>,** но отображает информацию, находящуюся между дескрипторами **<NOFRAMES>** и **</NOFRAMES>**. Дескрипторы <NOFRAMES> размещают внутри дескрипторов **<FRAMESET>**:

```
<title>frame demo</title>
<frameset cols=”60%,*”>
<frame src=”docl.html”
<frame src=” doc2.html”
<noframes>
код и текстовое содержимое для отображения в браузерах, не поддерживающих фреймы
</noframes>
</frameset>
```

Можно создавать вложенные фреймы, в которых фрейм будет разделен еще на несколько других фреймов.

```
<frameset cols=”30%,*”>
 <frame src=”docl.html” name=”frame1”>
 <frameset rows=”*,20%”>
 <frame src=”doc2.html” name=”frame2”>
 <frame src=”doc3.html” name=”frame3”>
 </frameset>
</frameset>
```

JavaScript может ссылаться и на свойства других документов, находящихся в фреймах, а также использует определенные особенности ссылок на эти свойства. На верхнее окно и его фреймы необходимо ссылаться с учетом их иерархических отношений. В ссылке на окно могут быть указаны свойства **Window,** такие как **top, parent, window и self.** Свойство **top** ссылается на главное окно (окно браузера), а **parent** — на мультифреймовое окно, в котором содержится текущий фрейм. Для фреймов, содержащихся в верхнем окне, **parent** и **top** представляют собой одно и то же. Свойства **self** и **window** ссылаются на текущий фрейм или окно.

Существенная выгода, которую дают фреймы, заключается в возможности модификации документа во фрейме, не затрагивая при этом другой фрейм. Возможно, самой простой методикой обновления следует

считать использование ссылок для изменения текущего фрейма (загружая новый документ). Для того чтобы направить новый документ в другой фрейм, используют дескриптор якоря, имеющий атрибут target, для которого можно определить значение имени фрейма или относительное имя. Относительные имена, применяющиеся для целевых ссылок, всегда начинаются с символа «_» и всегда записываются в нижнем регистре. Эти имена — _top, _parent и _self — соответствуют свойствам окна top, parent и self. Однако не существует имени _window. Следующий код представляет фреймы, в которых ссылка в одном фрейме должна обновить другой фрейм:

```
<frameset cols=”10%,*”>
<frame src=”docl.html” name=”framel”>
<frameset cols=”50%,*”>
<frame src=”frame2.html” name=”frame2”>
<frame src=”doc3.html” name=”frame3”>
</frameset>
```

Файл frame2.html содержит дополнительную информацию для установки фреймов в окне:

```
<frameset rows=”*,20%”>
<frame src=”doc4.html” name=”frame4”>
<frame src=”doc5.html” name=” frames “>
</frameset>
```

Для загрузки нового документа в frame2 с помощью ссылки в doc4.html (в frame4) эту ссылку записывают следующим образом:

```
<a href=”new.html” target=”_parent”>
```

Для загрузки документа в верхнее окно и очистки всех фреймов используют такую строку:

```
<a href=”new.html” target=”_top”>
```

Поскольку ссылки в JavaScript являются объектами, a href и target — свойствами ссылок, можно создать динамическую ссылку, присваивая другие значения href и target:

```
<a href=”#” target= “_top” onclick=’this.href=”new.html”; this.target=”_self”;’>
```

Для поддерживающих JavaScript браузеров документ new.html загрузится в текущий фрейм; если же JavaScript не поддерживается, то текущий документ загружается в верхнее окно.

Другой способ обновления фрейма, который не обязательно требует применения ссылок, связан с использованием свойства **href** объекта **Location.** Это свойство можно также применять вместе с обработчиком событий кнопки формы или с функцией.

Пример

Страница с фреймами:

Файл multiframe.html

```
<html>
<head>
<title>одновременное изменение нескольких файлов</title>
</head>
  <frameset cols=”150,*”>
      <frame name=”l” src=”link.htm”>
      <frameset rows=”80,*”>
                        <frame name=”t” src=”title1.htm”>
                       <frame name=”b” src=”body1.htm”>
      </frameset>
  </frameset>
<noframes>
```

```
</noframes>
</html>
```

Файл link.htm

```
<html>
<head>
<title>Ссылки</title>
  <script>
    function LinkFrames(title, body){
                parent.frames[1].location.href=title;
                parent.frames[2].location.href=body;
                return false;
    }
  </script>
</head>
<body>
   <h2>Ссылки</h2><br>
        <a href=”body1.htm” target=”b” onClick=”return LinkFrames(‘title1.htm’, ‘body1.htm’);”>
        Ссылка 1</a><br>
        <a href=”body2.htm” target=”b” onClick=”return LinkFrames(‘title2.htm’, ‘body2.htm’);”>
        Ссылка 2</a><br>
</body>
</html>
```

Файл title1.htm

```
<html>
 <head>
```

```
    <title>Заголовок 1</title>
  </head>
  <body>
     <h2> Заголовок 1 </h2>
  </body>
</html>
```

Файл title2.htm

```
<html>
  <head>
    <title>Заголовок 2</title>
  </head>
  <body>
     <h2> Заголовок 2 </h2>
  </body>
</html>
```

Файл body1.htm

```
<html>
  <head>
    <title>Содержимое 1</title>
  </head>
  <body>
     Первая страница
  </body>
</html>
```

Файл body2.htm

```
<html>
  <head>
    <title>Содержимое 2</title>
  </head>
  <body>
     Вторая страница
  </body>
</html>
```

Задание для самостоятельного выполнения

Создать страницу с четырьмя изменяющимися фреймами (в них надписи, рисунки, ссылки).

Тема 14. Визуальные эффекты. Бегущие строки

Эффект бегущей строки относится к числу наиболее распространенных в среде разработчиков Web-сайтов. Для создания бегущей строки в JavaScript необходимо:

• Создать HTML-форму с внедренным объектом **Text.**

• Записать функцию, прокручивающую текст внутри объекта **Text.**

• Ассоциировать функцию с обработчиком событий окна **onLoad** для запуска процесса.

Для начала создайте объект **Form** и внутри него объект **Text.** При этом рекомендуется соответствующим образом назвать эти объекты, поскольку они создаются исключительно для реализации бегущей строки и ни для чего более. Например, форму стоит назвать «marqueeForm», а объект Text — «marqueeText» или как-то по-другому, однако максимально близко по смыслу. После этого необходимо ввести сообщение как атрибут value дескриптора <input>. Ниже приводится код <form>:

```
<CENTER>
<form name=”marqueeForm”>
<input name=”marqueeText” size=”50” value=”THIS JUST IN…JavaScript is selected as official scripting language of the 2000 Summer Olympics”>
</form>
</center>
```

После определения контейнера для бегущей строки можно приступить к основной части. Для начала определяются две глобальные константы, используемые в бегущей строке: скорость прокручивания (в миллисекундах) и количество знаков, прокручиваемых за определенный временной промежуток. Затем потребуется создать объект **String** с именем marqueeMessage:

```
var SCROLL_RATE = 100;
var SCROLL_CHARS = 1;
var marqueeMessage = new String();
```

Следующий шаг — создание функции JSMarquee(), придающей обыкновенному объекту Text вид бегущей строки. Поскольку бегущая строка должна постоянно обновляться, не удастся обойтись без рекурсии JSMarquee() за счет вызова setTimeout() в начале функции. Глобальная переменная SCROLL_RATE используется как параметр метода, определяющий частоту вызова функции. Основная задача функции JSMarquee() заключается в получении значения объекта Text и присвоению его строке marqueeMessage. Затем можно смоделировать перенос слов, отделяя порцию данных сообщения спереди и переписывая эту порцию в конец строки. Для этого используется метод substring() объекта String:

```
function JSMarquee(){
setTimeout (' JSMarquee () ' , SCROLL_RATE) ;
marqueeMessage =
document.marqueeForm.marqueeText.value;
document.marqueeForm.marqueeText.value =
marqueeMessage. substring ( SCROLL_CHARS ) +
marqueeMessage.substring( 0, SCROLL_CHARS ) ;}
```

Пример 1

Бегущая строка:

```
<html>
 <head>
 <title>JavaScript Unleashed</title>
 <script type=”text/javascript”>
   var SCROLL_RATE = 100;
```

```
   var SCROLL_CHARS = 1;
   var MESSAGE = “THIS JUST IN . . . JavaScript is selected as “ + “official scripting language of the 2000 Summer Olympics . . . “;
   var marqueeMessage = new String();
   function JSMarquee() {
      setTimeout (‘JSMarquee()’, SCROLL_RATE);
      marqueeMessage = document.marqueeForm.marqueeText.value;
      document.marqueeForm.marqueeText.value=marqueeMessage.substring(SCROLL_CHARS) + marqueeMessage.substring(0, SCROLL_CHARS);
   }
 </script>
 </head>
 <body onload=”JSMarquee()”>
  <center>
   <form name=”marqueeForm”>
    <input name=”marqueeText” size=”50”>
    <script type=”text/javascript”>
      document.marqueeForm.marqueeText.value=MESSAGE;
    </script>
   </form>
  </center>
   </body>
</html>
```

Для создания бегущей строки можно использовать специальный дескриптор **<MARQUEE>**. Атрибуты этого дескриптора контролируют параметры перемещения текста:

- **Behavior** – способ выполнения бегущей строки:
  - *scroll* - текст исчезает за краем страницы;
  - *slide* – после выполнения заданного числа циклов текст остается у левого или правого поля страницы;

➢ *alternate* – направление текста в строке меняется на противоположное после выполнения каждого цикла, по завершении текст остается у края страницы.

- **BGColor** – цвет фона бегущей строки.
- **Direction** (*left, right*) – направление перемещения текста.
- **Height** – высота бегущей строки.
- **Hspace** – отступ в пикселях текста бегущей строки от левого и правого полей страницы.
- **Loop** – число показов текста в строке:

1. ? – целочисленное значение;
2. *Infinite* – бесконечное повторение эффекта по циклу.

- **Scrollamount** – смещение текста в пикселях за один шаг.
- **Scroldelay** – временный промежуток между смещениями в миллисекундах, по умолчанию 60мс.чтобы ускорить перемещение текста, переустановите минимальное значение смещения текста в атрибуте Truespeed.
- **Vspace –** отступ в пикселях по вертикали от текста до рамки бегущей строки.
- **Truespeed –** минимальное значение смещения текста, по умолчанию 60мс.
- **Width** – ширина бегущей строки в пикселях.

Пример 2

Управление свойствами бегущей строки с помощью кнопок формы:

```
<html>
<head>
<title>Бегущая строка</title>
<script>
  function set_marquee(text, color) {
    marquee.innerText = text;
```

```
    marquee.bgColor = color;
  }
</script>
</head>
<body>
  <MARQUEE name=”marquee” id=”marquee”></MARQUEE>
  <form>
    <p>
        <input type=”button” value=”Красный” onClick=”set_marquee(‘Красная строка’,’red’);”>
        <input type=”button” value=”Желтый” onClick=”set_marquee(‘Желтая строка’,’yellow’);”>
        <input type=”button” value=”Зеленый” onClick=”set_marquee(‘Зеленая строка’,’green’);”>
        </p>
  </form>
</body>
</html>
```

Задание для самостоятельного выполнения

Создать на странице бегущую строку любым из представленных способов и либо через атрибуты тега, либо через свойства объекта поменять цвет фона, направление перемещения текста, ускорить или замедлить текст, изменить другие физические параметры.

Тема 15. Слои

До недавнего времени одним из фундаментальных ограничений Web-страниц была невозможность расположения текста или изображений точно в указанном месте на странице. Кроме того, не существовало способов наложения HTML-элементов друг на друга. В настоящий моментдля создания слоев используется элемент **<Div>**, являющийся блочным элементом разметки. Для управления слоями необходимо использовать таблицы стилей (CSS).

Свойства CSS перечислены в приведенном ниже списке:

• **position –** используется для сообщения браузеру, как и где размещать элементы, к которым применяется это свойство. Существует четыре различных значения этого свойства:

absolute Это значение сообщает браузеру, что будут определяться абсолютные координаты элемента. При использовании данного значения применяются также свойства top и left. Если они не определены, браузер полагает, что координаты x и y (верхний левый угол окна браузера) равны 0.

Fixed Это значение аналогично absolute, за исключением того, что не перемещается при прокрутке окна.

relative Это значение позволяет сдвигать заданный элемент относительно предыдущего элемента.

static Это значение по умолчанию, которое используется в визуализации HTML в современных версиях браузеров.

• **left, right, top и bottom –** эти свойства, имеющие числовые значения для координат, нельзя использовать все сразу. Например, элемент не может установить свойство **right**, если уже установлено свойство **left.** То же самое можно сказать и относительно свойств **top** и **bottom**. Если попытаться

сделать это, большинство браузеров по умолчанию перейдут к значениям свойства, заданным последним в определении используемых стилей.

• **height и width –** позволяют определять высоту и ширину слоев. Эти два свойства принимают числовые значения для определения того, насколько высоким и широким должен быть слой.

Пример1

Применение вышеописанных свойств слоя:

```
<html>
 <head>
  <title>Слои</title>
 </head>
 <body>
      <div   id=”first” style=”position:absolute; top:0px; left:0px; height:20px;  width:40px;
        background-color:green;”>
        1 слой
      </div>
      <div   id=”second” style=”position:absolute; top:0px;   right:0px; height:10px;
        width:30px; background-color:red;”>
        2 слой
      </div>
      <div   id=”third” style=”position:absolute; bottom:0px;   right:0px; height:100px;
       width:50px; background-color:blue;”>
        3 слой
      </div>
      <div   id=”fourth” style=”position: absolute; bottom: 0px;   left: 0px; height: 100px;
       width: 200px; background-color: yellow;”>
        4 слой
      </div>
```

```
<div    id=”fifth” style=”position: absolute; top: 50%;   left: 50%;
  background-color:violet;”>
  5 слой
</div>
</body>
</html>
```

• **z-index** – содержит числовое значение, которое определяет расположение слоя относительно окна браузера, которое по умолчанию является нулевым (0). Установка для **z-index** значения 1 означает, что слой размещается поверх окна браузера. Это порядок, в котором слои появляются на странице. При этом если имеются накладывающиеся друг на друга слои, и у одного из них **z-index** равен 1, то он налагается на другой. Данное свойство чаще всего используется для изменения порядка, а не для определения стандартного (по умолчанию) порядка при создании Web-страниц.

Пример 2

Управление порядком слоев:

```
<html>
 <head>
 <title>Управление порядком</title>
 <script>
 function moveTxt(){
        if(ani1.style.zIndex == 0)
        {
              ani1.style.zIndex =1;
 ani2.style.zIndex =0;

        }
 else {ani1.style.zIndex =0;
```

```
ani2.style.zIndex =1;
}
}
</script>
</head>
<body>
<div id=”ani1” style=”position: absolute; left: 50; top: 50; height: 100;width: 250; background-color: red; z-Index: 0”>
Слой 1</div>
<div id=”ani2” style=”position: absolute; left: 80; top: 80; height: 100;width: 250; background-color: green; z-Index: 1”>
Слой 2</div>
<div id=”control” style=”position: absolute; left: 80; top: 250”>
<input value=”Поменять слои местами” type=”button” onClick=”moveTxt()”>
</div>
</body>
</html>
```

• **visibility -** это свойство отвечает за сокрытие и отображение слоев.

Таблица 15.1

collapse	Это значение аналогично всегда hidden, кроме случаев использования в таблицах.
hidden	Это значение скрывает элемент.
inherit	Это значение по умолчанию, определенное в браузере; принимает то же самое значение, что и его родительский элемент.
visible	Это значение делает элемент видимым.

Пример 3

Бегущие строки, созданные с помощью слоев:

```
<html>
<head>
<title>Бегущая строка</title>
 <script>
 function moveTxt(){
        if(ani1.style.pixelLeft<800)
                {
                ani1.style.pixelLeft+=3;
                        setTimeout("moveTxt()", 50);
                }
        }
 </script>
</head>
<body onLoad="moveTxt()" onUnload="alert('Вы нас уже покидаете?')">
 <div id="ani1" style="position: absolute; left: 10; top: 20">
        Бегущая строка</div>
</body>
</html>
```

Задание для самостоятельного выполнения

Создать страницу с примером управления видимостью слоев.

1 Основная литература

1. Макарова, Н. В. Информатика : учебник для вузов / Н. В. Макарова, В. Б. Волков .— М. [и др.] : Питер, 2011 .— 574 с. : ил .— (Учебник для вузов) (Стандарт третьего поколения) .— Библиогр. в конце гл. — ISBN 978-5-496-00001-7 (в пер.)

2. Елович, И. В. Информатика : учебник для вузов / И. В. Елович, И. В. Кулибаба ; под ред. Г. Г. Раннева .— Москва : Академия, 2011 .— 395 с. : ил. — (Высшее профессиональное образование: Информатика) (Бакалавриат) .— ISBN 978-5-7695-7975-2

3. Острейковский, В. А. Информатика : учебник для вузов / В. А. Острейковский .— 5-е изд., стер. — М. : Высш. Шк., 2009 .— 512 с. : ил .— ISBN 978-5-06-006134-5

4. Степанов, А.Н. Информатика : учеб. пособие для вузов / А.Н.Степанов .— 5-е изд. — М.[и др.] : Питер, 2007 .— 765с. : ил. — (Учебник для вузов).— Библиогр. в конце кн. — ISBN 978-5-469-01348-8 /в пер./ : 250.52. (АУЛ-1,21)

5. Цветкова А.В. Информатика и информационные технологии [электронный ресурс]: учебное пособие / А. В.Цветкова.— Саратов: Научная книга, 2012.— 190 с.— Режим доступа: http://www.iprbookshop.ru/6276. — Режим доступа : ЭБС «IPRbooks», по паролю

2 Дополнительная литература

1. Воройский, Ф.С. Информатика. Новый систематизированный толковый словарь-справочник. Введение в современные информационные и телекоммуникационные технологии в терминах и фактах [электронный ресурс] /Ф.С. Воройский..— М.: Физмат-лит, 2011.— 760 с.— Режим доступа: http://www.iprbookshop.ru/12990.— ЭБС «IPRbooks», по паролю

2. Губарев, В. В. Информатика. Прошлое, настоящее, будущее [электронный ресурс]: учебное пособие для вузов / В. В.Губарев.— М.: Техносфера, 2011.— 432 с.— (Мир программирования). – ISBN 978-5-94836-288-5. – Режим доступа: http://www.iprbookshop.ru/13281.— ЭБС «IPRbooks», по паролю

3. Информатика [электронный ресурс]: учебное пособие/ С.В. Тимченко [и др.]; ТУСУР.— Томск: Эль Контент, 2011.— 160 с.—ISBN 978-5-4332-0009-8. – Режим доступа: http://www.iprbookshop.ru/13935.— ЭБС «IPRbooks», по паролю

4. Патрикова, Е. Н. Компьютерная визуализация выпускных квалификационных работ [электронный ресурс]: учебное пособие по специальности 170400.65 «Стрелково-пушечное, артиллерийское и ракетное оружие» /Е. Н. Патрикова; ТулГУ. – Тула : Изд-во ТулГУ, 2012.- 98 с. : ил. – ISBN: 978-50-7679-0 .- Режим доступа: https://tsutula.bibliotech.ru/Reader/Book/ 2013122613254884169400005123. – Электронный читальный зал «Библиотех», по паролю

5. Патрикова, Е. Н. Визуализация материалов курсовых и дипломных работ в приложении Microsoft Powerpoint [электронный ресурс]: учебное пособие по направлению 030500.62 и специальности 030501.65 Юриспруденция /Е. Н. Патрикова; Междунар. Юрид. Ин-т Тул. Филиал. – Электрон. Текстовые данные.—

Москва, 2011.- 65 с. : ил. – Режим доступа: https://tsutula.bibliotech.ru/Reader/Book/2013040914312936592600008149.- Электронный читальный зал «Библиотех», по паролю

3 Периодические издания

1. Информационные технологии : теоретический и прикладной научно-технический журнал .— 2013- .— М. : Новые технологии, 2013 - .— ISSN 1684-6400.
2. Информационные технологии и вычислительные системы : [журнал] / учредитель РАН, Ин-т системного анализа.—М., 2013-. Основан в 1995 г. – Выходит ежеквартально. – ISSN 2071-8632
3. Открытые системы. СУБД [электронный ресурс] : [журнал].- М.:Открытые системы, 2013- . – ISSN 1028-7493. – Режим доступа : http://elibrary.ru/projects/subscription/ rus_titles_ open.asp .- eLibrary.ru, со всех компьютеров библиотеки ТулГУ, по паролю
4. Прикладная информатика [электронный ресурс] : научно-практический журнал .— М. : Маркет ДС, 2013 - .— Выходит 6 раз в год .— ISSN 1993-8314.- Режим доступа : http://elibrary.ru/projects/subscription/rus_titles_open.asp.-eLibrary.ru, со всех компьютеров библиотеки ТулГУ, по паролю

4. Интернет-ресурсы

1. *Электронный читальный зал "БИБЛИОТЕХ"* : учебники авторов ТулГУ по всем дисциплинам.- Режим доступа: https://tsutula.bibliotech.ru/, по паролю.- Загл. С экрана
2. ЭБС *IPRBooks* универсальная базовая коллекция изданий.-Режим доступа: *http://www.iprbookshop.ru/*, по паролю.- .- Загл. с экрана
3. Научная Электронная Библиотека *eLibrary* – библиотека электронной периодики, режим доступа: *http://elibrary.ru/* , по паролю.- Загл. с экрана.
4. НЭБ КиберЛенинка научная электронная библиотека открытого доступа, режим доступа *http://cyberleninka.ru/* ,свободный.- Загл. с экрана.
5. Единое окно доступа к образовательным ресурсам: портал [Электронный ресурс]. - Режим доступа : *http: //window.edu.ru.* ,свободный - Загл. с экрана.