Глава 3
Базы данных

§ 12
Информационные системы

Основные понятия

Первоначально вычислительные машины разрабатывались для выполнения расчётных задач — решения систем уравнений, определения траекторий полёта снарядов и спутников и т. п. Однако сейчас значительную часть времени компьютеры заняты обработкой нечисловых данных — текстов, изображений, звука и видео. В нашу жизнь вошли информационные системы, с помощью которых мы узнаём прогноз погоды и расписание поездов, определяем маршруты путешествий, заказываем билеты на самолёты, бронируем номера в гостиницах и т. п.

Информационная система (ИС) в широком смысле — это аппаратные и программные средства, предназначенные для того, чтобы своевременно обеспечить пользователей нужной информацией.

У информационной системы две основные задачи — она должна обеспечивать:

- хранение данных;
- доступ к данным, т. е. возможность искать и изменять данные.

Массивы данных, с которыми работают информационные системы, обычно имеют большой объём (нередко несколько гигабайтов и даже терабайтов) и размещаются во внешней памяти компьютера. Данные хранятся в таком виде, чтобы их было легко искать и изменять. Такие наборы данных называются базами данных.

База данных (БД) — это специальным образом организованная совокупность[1] данных о некоторой предметной области, хранящаяся во внешней памяти компьютера.

Данные сами по себе бесполезны, если мы не умеем с ними работать. Поэтому необходимо специальное программное обеспечение, которое позволяет искать и изменять данные.

Система управления базой данных (СУБД) — это программные средства, которые позволяют выполнять все необходимые операции с базой данных.

Хотя термины «база данных» и СУБД обозначают различные понятия, они неразрывно связаны: свойства базы данных определяются СУБД, которая ею управляет, и наоборот. Комплекс «БД + СУБД» называется *системой базы данных* (англ. *database system*) или *информационной системой* в узком смысле.

Вы знаете, что данные в компьютерном формате — это двоичные коды, которые могут обозначать всё, что угодно. Поэтому СУБД должна «знать» формат файлов (что где записано). В первых информационных системах каждая база данных имела свой собственный формат, который придумывал её автор. Это очень неудобно, потому что для каждой ИС нужно разрабатывать специальную программу для работы с базой данных. Более того, при любых изменениях в БД (например, при увеличении размера данных) надо переделывать все работающие с ней программы и переводить старые данные в новый формат.

Поэтому постепенно перешли к использованию специальных программ (они и получили название СУБД), которые занимались *только* хранением и обработкой данных, независимо от их содержания, и могли применяться в самых разных задачах. При этом вместе с основными данными нужно хранить описание их структуры — так называемые *метаданные*, «сведения о данных», которые использует СУБД для обращения к данным.

[1] Термин «совокупность» обозначает множество элементов, обладающих общими свойствами.

СУБД решают все задачи, связанные с управлением данными, в том числе:

- поиск данных;
- редактирование данных;
- выполнение несложных расчётов;
- обеспечение *целостности* (корректности, непротиворечивости) данных;
- восстановление данных после сбоев.

Как правило, пользователь работает с СУБД не напрямую, а через прикладную программу, в которой предусмотрен удобный ввод данных и оформление результатов (рис. 3.1).

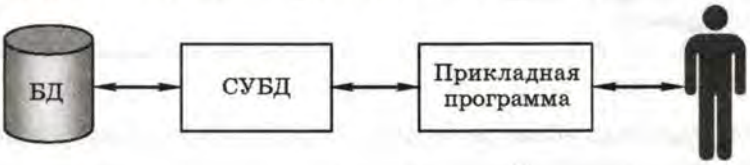


Рис. 3.1

Иногда функции СУБД и прикладной программы объединяются в одной программе (например, в OpenOffice.org Base или Microsoft Access).

Классификация

В простейшем случае база данных и СУБД находятся на одном компьютере. Такая информационная система называется **локальной**, с ней работает один пользователь. В современных браузерах (Google Chrome, Safari, Mozilla Firefox) есть встроенные средства, позволяющие создавать и использовать локальные ИС.

Преимущество локальных ИС — *автономность*, т. е. независимость от работы локальных и глобальных сетей. Их недостатки проявляются тогда, когда с БД должны работать несколько пользователей:

- базу данных нужно обновлять на каждом компьютере;
- невозможно «стыковать» изменения, вносимые пользователями.

В локальных ИС разработчики иногда применяют свои собственные форматы хранения данных, однако в этом случае *теряется переносимость* — возможность использования базы данных в других информационных системах.

Как правило, в современных информационных системах используют **удалённые базы данных**, расположенные на серверах (специально выделенных компьютерах) локальной или глобальной сети. В этом случае несколько пользователей могут одновременно работать с базой и вносить в неё изменения.

СУБД, работающие с удалёнными базами данных, можно разделить на два типа по способу работы с файлами:

- файл-серверные СУБД;
- клиент-серверные СУБД.

Файл-серверные СУБД (например, Microsoft Access) работают на компьютерах пользователей (они называются рабочими станциями) (рис. 3.2). Это значит, что сервер только хранит файлы, но не участвует в обработке данных. Когда пользователь вносит изменения в базу, СУБД с его рабочей станции блокирует файлы на сервере, чтобы их не могли изменить другие пользователи.

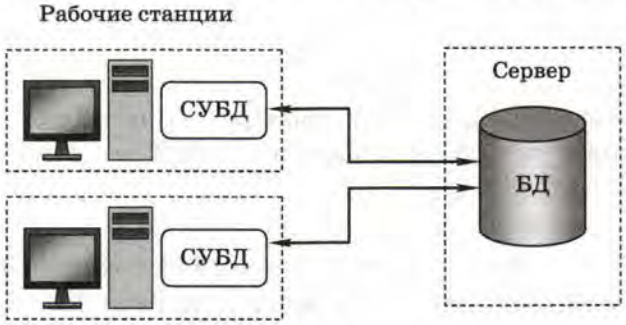


Рис. 3.2

При большом количестве пользователей проявляются недостатки файл-серверных ИС:

- обработку данных выполняют СУБД на рабочих станциях, поэтому компьютеры пользователей должны быть достаточно мощными;
- при поиске данных вся БД копируется по сети на компьютер пользователя, это создает значительную лишнюю нагрузку на сеть;
- слабая защита от неправомерного доступа к данным (защита устанавливается на рабочих станциях, а не в едином центре);
- ненадёжность при большом количестве пользователей, особенно если они вносят изменения в базу данных.

Чтобы избавиться от этих недостатков, нужно переместить СУБД на сервер.

Клиент-серверная СУБД (рис. 3.3) расположена на том же компьютере, где находится база данных. Она полностью берёт на себя всю работу с данными, т. е. читать и изменять данные в базе можно только с помощью этой СУБД.

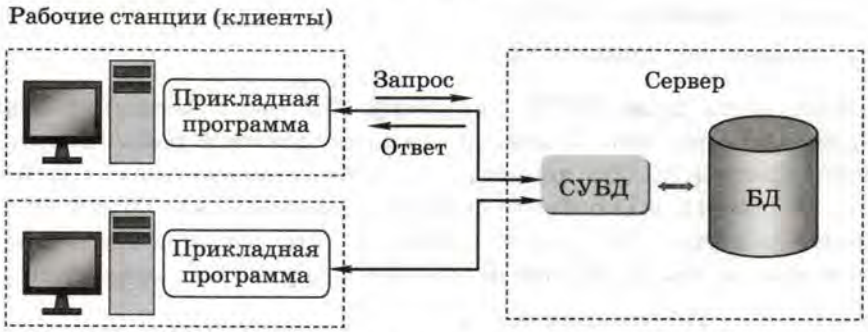


Рис. 3.3

На компьютере пользователя работает прикладная программа-клиент, которая по сети обращается к СУБД для выполнения операций с данными.

Задачи клиента:

- отправить серверу команду (запрос) на специальном языке;
- получив ответ сервера, вывести результат на экран пользователя или на печать.

В современных клиент-серверных СУБД для управления данными чаще всего используют **язык SQL** (англ. *Structured Query Language* — язык структурных запросов). Он содержит все команды, необходимые работы с данными, включая получение нужной информации, создание и изменение базы данных.

Задачи сервера:

- ожидать запросы клиентов по сети;
- при поступлении запроса поставить его в очередь на выполнение;
- выполнить запрос;
- передать результаты клиенту.

Отметим, что серверная и клиентская части могут быть установлены на одном компьютере.

Самые известные среди коммерческих клиент-серверных СУБД — *Microsoft SQL Server* и *Oracle*. Существуют бесплатные клиент-серверные СУБД: *Firebird* (www.firebirdsql.org), *PostgreSQL* (www.postgresql.org), *MySQL* (www.mysql.com). Свободная СУБД MySQL часто используется для управления базами данных на небольших веб-сайтах.

Достоинства клиент-серверных СУБД:

- основная обработка данных выполняется на сервере, поэтому рабочие станции могут быть маломощными;
- проще модернизация (достаточно увеличить мощность сервера);
- надёжная защита данных (устанавливается на сервере);
- снижается нагрузка на сеть, так как передаются только нужные данные (запросы и результаты выполнения запросов);
- надёжная работа при большом количестве пользователей (запросы ставятся в очередь).

Их *недостатки* — повышенные требования к мощности сервера и высокая стоимость коммерческих СУБД (Microsoft SQL Server и Oracle).

Многие современные информационные системы (например, поисковые системы в Интернете) работают с огромными объёмами данных, которые невозможно разместить на одном компьютере. Поэтому появились **распределённые базы данных**, расположенные на множестве компьютеров, и соответствующие СУБД для управления ими. Пользователь работает с распределённой базой данных точно так же, как и с обычной (нераспределённой). Главная проблема в этой области — обеспечение целостности и непротиворечивости данных, особенно при разрыве связи между компьютерами.

Транзакции

Сейчас в базах данных нередко хранится очень важная информация, прежде всего финансовая. Поэтому необходимо обеспечить максимально возможную защиту данных от различных сбоев (например, при отключении питания).

Предположим, что в банке нужно перевести 100 000 рублей со счёта 12345 на счёт 54321. Эта банковская операция включает несколько действий с базой данных:

1) прочитать сумму на счету 12345;
2) уменьшить её на 100 000 рублей и записать результат обратно;
3) прочитать сумму на счету 54321;
4) увеличить её на 100 000 рублей и записать результат обратно.

Представьте себе, что случится, если после выполнения первых двух действий произойдет сбой питания и действия 3 и 4 не будут выполнены. Ответ ясен — первый клиент просто потеряет 100 000 рублей со своего счёта, т. е. данные станут некорректными.

Чтобы этого не произошло, либо все шаги операции перевода денег должны быть выполнены, либо ни один шаг не должен быть выполнен. Кроме того, между отдельными шагами операции никакие другие действия не должны выполняться. Такие сложные многошаговые операции называются транзакциями (от англ. *transaction* — сделка).

Транзакция — это группа операций, которая представляет собой одно законченное действие. Транзакция должна быть выполнена целиком или не выполнена вообще.

Как же обеспечить выполнение транзакций? Для этого часто используется *журналирование* по тому же принципу, что и в файловых системах. Перед внесением изменений в базу данных СУБД создаёт копии всех данных, которые будут изменяться, и записывает в специальный файл (журнал) все операции, которые нужно выполнить. Затем эти операции выполняются фактически и, если всё завершено успешно, запись удаляется из журнала. Если произошёл сбой, в журнале будет найдена информация о тех операциях, которые уже были завершены, и база данных восстанавливается в исходное состояние (транзакция не выполнена).

Некоторые СУБД, например Firebird, не используют журнал, а при внесении изменений создают в базе данных новые записи, которые отмечаются как «рабочие» только тогда, когда транзакция завершена.

Вопросы и задания

1. Что такое информационная система? Из каких компонентов она состоит?
2. Что такое база данных? Какими свойствами она должна обладать?

3. Является ли базой данных бумажная картотека в библиотеке? Ответ обоснуйте.
4. Какие функции выполняет СУБД?
5. Почему произошёл переход от множества разнообразных форматов хранения данных к использованию универсальных СУБД? Приведите примеры и обоснуйте.
6. Что такое метаданные?
7. Объясните схему работы пользователя с базой данных.
8. Назовите достоинства и недостатки локальных ИС.
9. Назовите достоинства и недостатки файл-серверных СУБД.
10. В каких ситуациях вы могли бы рекомендовать использование файл-серверных СУБД? Ответ обоснуйте. Подготовьте сообщение.
11. Назовите достоинства и недостатки клиент-серверных СУБД.
12. Как разделяются функции между клиентской и серверной программами?
13. Что такое SQL?
14. Что такое распределённые базы данных?
15. Что такое транзакция?
16. Как обеспечивается защита данных в случае сбоев при использовании механизма транзакций?

Подготовьте сообщение

а) «Информационные системы вокруг нас»

б) «Технология клиент — сервер»

в) «Бесплатные СУБД»

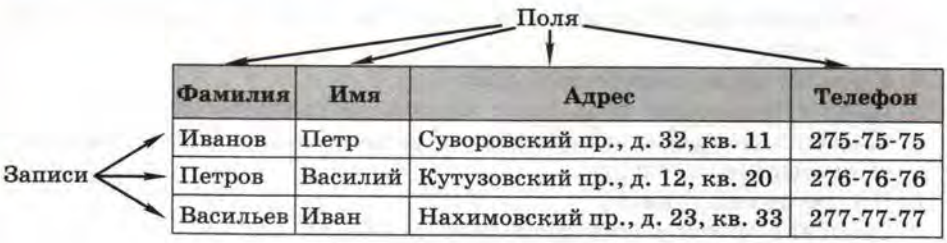
г) «Коммерческие и бесплатные СУБД: плюсы и минусы»

§ 13 Таблицы

Основные понятия

Данные, хранящиеся в современных базах данных, чаще всего удобно представлять в виде таблиц[1]. Например, так называемый «список контактов» (сведения о друзьях и знакомых) может выглядеть, как показано на рис. 3.4.

[1] В середине XX века широко применялись иерархические и сетевые базы данных, но сейчас они редко встречаются на практике. Самый известный современный пример иерархической базы данных — реестр в операционной системе Windows, где хранятся настройки самой системы и программ.



Фамилия	Имя	Адрес	Телефон
Иванов	Петр	Суворовский пр., д. 32, кв. 11	275-75-75
Петров	Василий	Кутузовский пр., д. 12, кв. 20	276-76-76
Васильев	Иван	Нахимовский пр., д. 23, кв. 33	277-77-77

Рис. 3.4

Столбцы таблицы называются **полями**, а строки — **записями**. Таблица на рис. 3.4 относится к типу «объект — свойства», т. е. запись — это описание некоторого объекта (в данном случае — человека), а поля содержат свойства этого объекта. В этой таблице четыре поля: *Фамилия*, *Имя*, *Адрес* и *Телефон* и три записи.

Количество и состав полей определяет разработчик базы данных, пользователи могут только изменять записи (добавлять, удалять, редактировать).

Любое поле должно иметь уникальное (неповторяющееся) имя. Например, нельзя назвать два поля *Фамилия*, но можно одно назвать *Фамилия*, а второе — *Девичья Фамилия*.

Каждое поле имеет свой **тип**. Как правило, СУБД поддерживают следующие типы данных:

- целые числа;
- вещественные числа;
- денежные суммы;
- логические значения (битовые поля);
- текстовые данные;
- время, дата;
- произвольные двоичные данные, например закодированный звук, видео и т. д.

Некоторые поля могут быть обязательными для заполнения. Если обязательное поле не заполнено, СУБД не внесёт изменения в базу данных и выдаст сообщение об ошибке.

Ключ

При чтении и изменении данных в таблице очень важно убедиться, что мы обращаемся именно к нужной записи. Это означает, что для надёжной работы каждая запись должна содержать какое-то уникальное значение, отличающее её от всех остальных.

Ключ — это поле или комбинация полей, однозначно определяющая запись.

Это значит, что ключ обладает свойством *уникальности*: в таблице не может быть двух записей, у которых одинаковое значение ключа. Например, ключом может быть номер паспорта, номер мобильного телефона, регистрационный номер автомобиля, адрес электронной почты и т. п.

Иногда можно выделить в таблице несколько ключей, например номер внутреннего паспорта и номер заграничного паспорта. В этом случае один из них выбирается в качестве основного и называется **первичным ключом**.

Ключ, состоящий из одного поля, называется **простым**, а соответствующее поле таблицы — **ключевым полем**. Простой ключ часто называют *идентификатором* от слова «идентифицировать» — отличить один объект от другого.

Ключ, который состоит из нескольких полей, называется **составным**. Представим себе базу данных метеостанции, на которой через каждые 3 часа измеряются температура, влажность воздуха, скорость ветра и т. п. (рис. 3.5).

Дата	Время	Температура	Влажность	Скорость ветра
21.07.2012	12	25	75	4
21.07.2012	15	23	70	3
...	...	...	...	...

Рис. 3.5

Здесь ни одно поле не может быть ключом, потому что значения в каждом из них могут повторяться. Однако для каждой пары *Дата* + *Время* в таблице может быть только одна запись, поэтому комбинация полей *Дата* и *Время* — это ключ.

Второе свойство ключа — *несократимость*. Заметим, что в рассмотренном примере к паре *Дата* + *Время* можно добавить и другие поля таблицы, но такая группа полей уже не будет считаться ключом, потому что она сократима, т. е. из неё можно исключить все поля, кроме полей *Дата* и *Время*, сохранив свойство уникальности.

Теперь посмотрим на приведённую на рис. 3.4 таблицу «список контактов». Ни фамилия, ни имя не могут быть ключом, потому что есть много однофамильцев и людей с одинаковыми именами. Составить ключ из полей *Фамилия* и *Имя* тоже не получит-

ся (могут быть однофамильцы и тёзки одновременно). Адрес и домашний телефон также не могут быть ключом, потому что в одной квартире могут жить несколько человек, с которыми вы общаетесь. В принципе может получиться так, что ключом будет комбинация *всех* полей записи.

Работать с составными ключами при выполнении операций с базой данных на практике очень неудобно. В таких случаях часто добавляют в таблицу ещё одно поле — так называемый *суррогатный* (т. е. неестественный) ключ, например номер записи. Во многих СУБД есть возможность заполнять его автоматически при добавлении каждой новой записи (рис. 3.6). При этом пользователю не нужно задумываться об уникальности такого ключа.

Номер	Фамилия	Имя	Адрес	Телефон
1	Иванов	Петр	Суворовский пр., д. 32, кв. 11	275-75-75
2	Петров	Василий	Кутузовский пр., д. 12, кв. 20	476-76-76
3	Васильев	Иван	Нахимовский пр., д. 23, кв. 33	477-77-77

Рис. 3.6

Индексы

Представим себе, что в таблице с адресами и телефонами, о которой мы говорили выше, записаны данные большого количества людей (в реальных базах данных могут быть миллионы записей). Как найти всех Ивановых?

Учтём, что записи расположены в таблице в том порядке, в котором они вводились (возможно, без всякого порядка). Если эту задачу решает человек, он, скорее всего, будет просматривать поле *Фамилия* с первой до последней записи, отмечая всех с фамилией Иванов. Такой поиск называется **линейным** — он требует просмотра всех записей (вспомните линейный поиск элемента в массиве). Если в таблице 1024 записи, придётся сделать 1024 сравнения, в больших базах линейный поиск будет работать очень медленно.

Теперь представим себе, что записи **отсортированы** по фамилии **в алфавитном порядке**. Возьмём запись, стоящую в середине таблицы (если в таблице 1024 записи, возьмём 512-ю) и сравним фамилию с нужной. Допустим, в 512-й записи фамилия — Коню-

хов. Поскольку нам нужны Ивановы, все они заведомо находятся в верхней половине таблицы, поэтому всю вторую половину можно вообще не смотреть. Таким образом, мы в два раза уменьшили область поиска. Далее проверяем среднюю из оставшихся записей (256-ю) и повторяем процесс до тех пор, пока не найдём Иванова или в области поиска не останется больше записей. Как вы знаете, такой **поиск** называется **двоичным** (вспомните материал учебника 10 класса). Он позволяет искать очень быстро: если в базе N записей, то придется сделать всего около $\log_2 N$ сравнений. Например, для $N = 1024$ получаем 11 сравнений вместо 1024, т. е. мы смогли ускорить поиск почти в 100 раз! Для бóльших N ускорение будет еще более значительным.

Однако у двоичного поиска есть и недостатки:

- записи должны быть предварительно отсортированы по нужному полю;
- можно искать только по одному полю.

На практике искать нужно по нескольким полям в каждой таблице и нет возможности каждый раз сортировать записи (это очень долго для больших баз). Как же в такой ситуации обеспечить быстрый поиск? Как часто бывает в программировании, можно увеличить скорость работы алгоритма за счёт расхода дополнительной памяти.

Наверное, вы видели, что для ускорения поиска во многие книги включают *индекс* — список ключевых слов с указанием страниц, где они встречаются. В базах данных специально для поиска создаются дополнительные таблицы, которые также называются индексами.

Индекс — это вспомогательная таблица, которая служит для ускорения поиска в основной таблице.

Простейший индекс — это таблица, в которой хранится значение интересующего нас поля основной таблицы (например, *Фамилия*) и список номеров записей, где такое значение встречается. Записи в индексе упорядочены (отсортированы) по нужному полю, т. е. для приведённой на рис. 3.6 таблицы индекс по полю *Фамилия* выглядит так, как на рис. 3.7.

Фамилия	№
Васильев	3
Иванов	1
Петров	2

Рис. 3.7

Теперь, если нам нужны люди с фамилией *Иванов*, мы можем искать номера Ивановых в индексе, используя быстрый *двоичный* поиск (там фамилии стоят по алфавиту!). Затем, когда номера нужных записей определены, данные Ивановых можно взять из основной таблицы. Заметим, что искать что-то в основной таблице уже не нужно. Таким образом, можно использовать двоичный поиск по разным полям (даже по комбинациям нескольких полей). Однако нужно учитывать, что:

- для каждого типа поиска приходится строить новый индекс, он будет занимать дополнительное место во внешней памяти (например, на жёстком диске);
- при любом изменении таблицы (добавлении, изменении и удалении записей) необходимо перестраивать индексы так, чтобы сохранить требуемый порядок сортировки; к счастью, современные СУБД делают это автоматически без участия пользователя, но это может занимать достаточно много времени.

Для тренировки самостоятельно постройте вручную индексы по полям *Имя*, *Адрес* и *Телефон*.

Целостность базы данных

Целостность базы данных означает, что БД содержит полную и непротиворечивую информацию и удовлетворяет всем заданным ограничениям.

Прежде всего нужно обеспечить *физическую целостность БД*, т. е. защитить данные от разрушения в случае отказа оборудования (например, при отключении питания или выходе из строя жёстких дисков).

Очень важно, что все изменения данных выполняются с помощью *транзакций*, которые позволяют в случае сбоя «откатить назад» все начатые операции.

Периодически (например, раз в неделю) администраторы создают резервные копии всех данных на дисках и ведут журнал изменений. При сбое восстанавливается самая последняя сохранённая версия БД.

Также используют так называемые RAID-массивы жёстких дисков (англ. *Redundant Array of Independent Disks* — избыточный массив независимых дисков), где информация дублируется

и может быть автоматически восстановлена в случае выхода из строя одного или даже нескольких дисков.

Теперь представьте себе, что в базе данных отдела кадров по ошибке у работника указан 1698 год рождения, а в поле *Зарплата* введено отрицательное число. В этих случаях нарушается *логическая целостность*, т. е. непротиворечивость данных. Чтобы этого не произошло, вводят ограничения на допустимые значения полей (контроль данных):

- каждое поле имеет свой тип; например, СУБД не даст записать в поле целого типа произвольный текст — будет выведено сообщение об ошибке;
- некоторые поля (в первую очередь первичный ключ) объявляются обязательными для заполнения;
- для полей, значения которых не могут повторяться, строятся уникальные индексы (при повторении значения выдаётся сообщение об ошибке);
- вводятся условия, которые должны выполняться для значений отдельных полей: например, можно потребовать, чтобы значение поля, в котором хранится количество учеников в классе, было положительным;
- для сложных данных используются шаблоны ввода: например, для ввода семизначного номера телефона можно использовать шаблон ###-##-##, где # означает любую цифру;
- вводятся условия, которые должны выполняться для нескольких полей каждой записи: например, дата увольнения работника не может быть более ранней, чем дата приёма на работу.

Заметим, что целостность БД не гарантирует *достоверность* данных, а только означает, что выполнены все установленные ограничения на эти данные и таким образом исключены явные противоречия.

Вопросы и задания

1. Объясните значения слов «поле», «запись».
2. Зачем каждому полю присваивают свой тип?
3. Какие типы данных поддерживаются в современных СУБД?
4. Что такое ключ таблицы? Назовите и объясните два свойства ключа.
5. Чем отличаются простой и составной ключи?

6. Чем отличаются понятия «ключ» и «первичный ключ»?
7. Какие из следующих данных могут быть ключом, а какие не могут:
   а) фамилия;
   б) имя;
   в) номер читательского билета;
   г) адрес электронной почты;
   д) адрес веб-сайта;
   е) марка автомобиля?
8. Объясните, когда одни и те же данные в одной ситуации могут быть ключом, а в другой — нет (например, адрес электронной почты, марка стиральной машины и т. п.). Приведите примеры.
9. В каких случаях в качестве первичного ключа используют номер записи? Можно ли применять такой подход, если в таблице есть другое уникальное поле?
10. Какие методы поиска данных вы знаете?
11. Чем различаются линейный и двоичный поиск? Назовите их достоинства и недостатки.
12. Что такое индекс? Как он строится?
13. Можно ли для одной и той же таблицы построить несколько индексов?
14. Объясните принцип поиска с помощью индекса.
15. Что такое целостность базы данных? Какие виды целостности вы знаете?
16. Как обеспечивается физическая целостность данных?
17. Как обеспечивается логическая целостность данных?

Подготовьте сообщение

а) «Типы данных, хранящиеся в БД»
б) «Суррогатные ключи: за и против»
в) «Поиск с помощью индексов»
г) «Что такое транзакция?»
д) «Что такое RAID-массив?»

Задачи

1. В базе данных хранится 1 048 576 = 2^{20} записей. Оцените количество сравнений, которое придётся сделать при использовании линейного и двоичного поиска по одному из полей. Во сколько раз быстрее работает двоичный поиск?
2. В таблице три поля: *Дата*, *Номер заказа*, *Товар* и *Количество*. Что можно выбрать в качестве первичного ключа? Какие индексы можно построить?
3. В школьной базе данных хранятся сведения о выданных аттестатах. Таблица включает поля *Фамилия*, *Имя*, *Отчество*, *Дата рождения*, *Год выпуска*, *Номер паспорта*, *Номер аттестата*. Что можно выбрать в качестве первичного ключа этой таблицы?

4. Постройте индексы по полям *Дата*, *Товар* и *Количество* для следующей таблицы:

Номер	Дата	Заказ	Товар	Количество, т
1	12.09.10	12	Ананасы	12
2	12.09.10	13	Апельсины	12
3	13.09.10	14	Ананасы	15
4	13.09.10	15	Бананы	13
5	13.09.10	16	Апельсины	11

*5. Напишите программу, работающую с однотабличной базой данных. Содержание придумайте сами (например, видеотека, база данных зоопарка или что-либо другое). В программе должны быть предусмотрены:

- просмотр записей;
- добавление записей (в конец таблицы);
- удаление записей (по номеру);
- сортировка по одному полю.

Для хранения записей используйте текстовый файл.

§ 14 Многотабличные базы данных

Почему не собрать всё в одной таблице?

Мы только что рассмотрели простейшую структуру, в которой все данные сведены в одну таблицу и поэтому искать информацию достаточно просто. Однако у такой модели есть и *недостатки*:

- *дублирование данных*; например, в базе данных школьной библиотеки будет много раз храниться фамилия автора «Пушкин»;
- при изменении каких-то данных (например, адреса фирмы), возможно, придётся *менять несколько записей*;
- нет *защиты от ошибок ввода* (опечаток).

Однотабличная база данных — это аналог картотеки, в которой все карточки имеют одинаковую структуру. В то же время на практике в одной базе нужно хранить данные, относящиеся к объектам разных типов, которые связаны между собой. Поэтому

возникает вопрос: какую модель лучше использовать для описания и хранения этих данных?

Посмотрим, как можно организовать базу данных, в которой хранятся данные об альбомах музыкальных групп. Вот что получается, если свести все данные в одну таблицу (рис. 3.8).

Альбомы

Код альбома	Название	Группа	Год	Число композиций
1	Реки и мосты	Машина времени	1987	16
2	В круге света	Машина времени	1988	11
3	Группа крови	Кино	1988	11
4	Последний герой	Кино	1989	10

Рис. 3.8

В данном случае в качестве первичного ключа можно выбрать пару свойств *Название* + *Группа*, но работать с таким составным ключом неудобно, поэтому мы ввели суррогатный ключ — дополнительное поле *Код альбома* (целое число), оно обозначено знаком 🔑.

Сразу видим, что в этой таблице есть дублирование — название группы (символьная строка) повторяется для каждого альбома этой группы. Причина в том, что в данных таблицы на самом деле есть сведения не только об альбомах, но и о группах — объектах совершенно другого класса. Поэтому для хранения всей информации о группах нужно сделать отдельную таблицу (рис. 3.9).

Группы

Код группы	Название	Год создания
1	Машина времени	1969
2	Кино	1981

Рис. 3.9

Здесь первичным ключом может быть название группы (символьная строка), но для ускорения работы введён суррогатный ключ — *Код группы* (целое число). В таблицу можно добавить другие данные о группе, например фамилию лидера, город и т. п.

В таблице *Альбомы* теперь будут храниться не названия групп, а их коды (рис. 3.10).

Альбомы

Код альбома	Название	Код группы	Год	Число композиций
1	Реки и мосты	1	1987	16
2	В круге света	1	1988	11
3	Группа крови	2	1988	11
4	Последний герой	2	1989	10

Рис. 3.10

Таким образом, база данных состоит из двух таблиц и хранит сведения о двух классах объектов. Эти таблицы связаны, их связь можно показать на схеме (рис. 3.11).

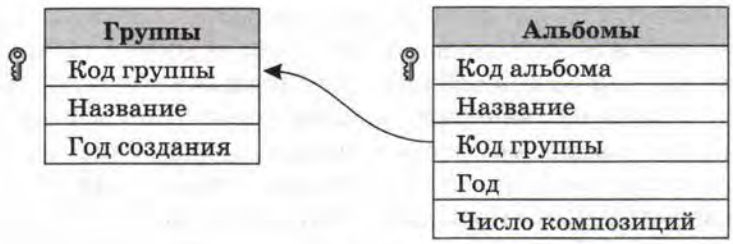


Рис. 3.11

На рисунке 3.11 перечислены поля каждой из таблиц и показана их связь: код группы в таблице *Альбомы* должен совпадать с кодом одной из групп в таблице *Группы*.

Внешний ключ (англ. *foreign key*) — это неключевое поле таблицы, связанное с первичным ключом другой таблицы.

Таким образом, *Код группы* — это внешний ключ в таблице *Альбомы*.

Итак, мы получили две связанные таблицы вместо одной. При этом:

- устранена избыточность (повторно хранятся только числовые коды);
- все изменения нужно выполнять только в одном месте;

- есть некоторая защита от ошибок при вводе данных — можно сделать так, что при заполнении таблицы *Альбомы* название группы будет выбираться из уже готового списка групп.

Однако кое-что *ухудшилось* из-за того, что данные разбросаны по разным таблицам:

- базами данных, в которых более 40–50 таблиц, сложно управлять из-за того, что разработчику трудно воспринимать информацию в таком «раздробленном» виде;
- при поиске приходится «собирать» нужные данные из нескольких таблиц.

Ссылочная целостность

Одна из важнейших задач СУБД — поддерживать *целостность базы данных*. В предыдущем параграфе мы уже говорили о том, как обеспечивается *физическая и логическая целостность*. В многотабличных базах данных необходимо обеспечить еще **ссылочную целостность**, т. е. правильность связей между таблицами.

Напомним, что если в таблице есть внешний ключ, он должен совпадать с одним из значений первичного ключа в другой таблице. Это значит, например, что в таблице *Альбомы* мы можем использовать только те коды групп, которые есть в таблице *Группы*. Если пользователь будет вводить несуществующие коды, СУБД должна выдать сообщение об ошибке и отказаться вносить изменения в базу данных.

При изменении кода группы (первичного ключа) нужно изменить соответствующие коды для всех альбомов этой группы.

Если пользователь удаляет какую-то группу из таблицы *Группы*, то для всех альбомов этой группы значение поля Код группы оказывается недействительным и целостность нарушается. В таких случаях возможно несколько вариантов действия СУБД:

- *запретить* удаление записи (из таблицы Группы) до тех пор, пока в базе есть связанные с ней подчинённые записи (в таблице *Альбомы*);
- выполнить *каскадное удаление*, т. е. вместе с удаляемой записью удалить также все связанные с ней подчинённые записи в других таблицах;
- *игнорировать* проблему, т. е. разрешить внести изменения.

Очевидно, что в последнем случае ссылочная целостность нарушается.

Типы связей

В базе данных, которую мы только что построили, использовались связь **«один ко многим»** (она обозначается также как «1:*N*» или «1–∞»). Это значит, что с одной записью в первой таблице могут быть связаны сколько угодно записей во второй таблице. Так в нашем случае каждый код группы может встречаться сколько угодно раз в таблице *Альбомы* (рис. 3.12).

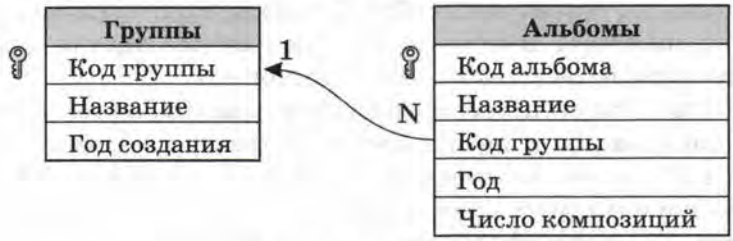


Рис. 3.12

При этом, как правило, на стороне «1» в связи участвует ключевое поле таблицы, а на стороне «N» — неключевое (для второй таблицы это внешний ключ).

В некоторых (достаточно редких) случаях используют связь **«один к одному»** (или «1:1»): каждой записи в первой таблице соответствует ровно одна запись в связанной таблице (рис. 3.13).

Сотрудники

Код	Фамилия	Имя	Отчество
1	Иванов	Петр	Сидорович
2	Петров	Сидор	Иванович
3	Сидоров	Иван	Петрович

Секретно

Код	Зарплата
1	20 000 р.
2	30 000 р.
3	40 000 р.

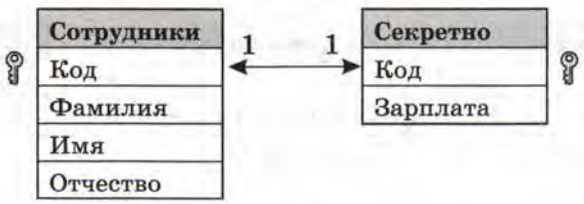


Рис. 3.13

В этом случае таблицы связаны через свои ключевые поля. Такую связь можно использовать для разделения большой таблицы на две части. Это важно, например, когда часть свойств объек-

та нужно сделать доступной всем, а другую часть — скрыть от некоторых пользователей.

Нередко при анализе исходных данных появляются связи типа «многие ко многим» (они обозначаются также «*M*:*N*» или «∞–∞»). Например, в школе каждый учитель может преподавать несколько разных предметов и каждый предмет обычно ведут несколько учителей. Поэтому связь между учителями и предметами — это связь «многие ко многим».

Как правило, современные СУБД явно не поддерживают связи «многие ко многим». Вместо этого используется третья дополнительная таблица и две связи «один ко многим».

В качестве примера рассмотрим базу данных кафе. Требуется хранить информацию о выполненных заказах, их составе и цене для того, чтобы строить итоговые отчёты автоматически. Все данные — это свойства двух классов объектов: заказов и блюд, входящих в меню. Поэтому будем использовать две таблицы (рис. 3.14).

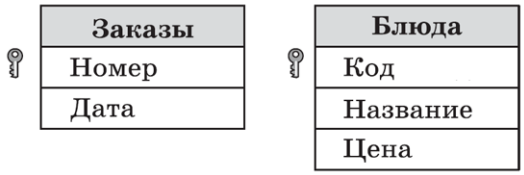


Рис. 3.14

Как они связаны? Очевидно, что каждое блюдо может быть включено во многие заказы. Вместе с тем каждый заказ может состоять из нескольких блюд. Таким образом, здесь существует связь «многие ко многим», которая в большинстве СУБД не поддерживается. Чтобы решить проблему, введём еще одну таблицу *Заказано*, которая связана с первыми двумя с помощью двух связей «1:*N*» (рис. 3.15).

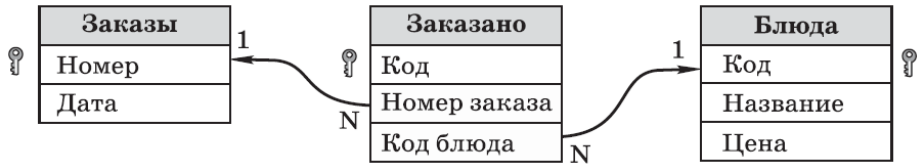


Рис. 3.15

Вот пример заполнения таблиц в такой базе данных (рис. 3.16).

Заказы

Номер	Дата
1	11.12.12
2	12.12.12

Заказано

Код	Номер заказа	Код блюда
1	1	1
2	1	3
3	1	4
4	2	1
5	2	2
6	2	2
7	2	5

Блюда

Код	Название	Цена
1	Борщ	80 руб.
2	Бифштекс	110 руб.
3	Гуляш	70 руб.
4	Чай	10 руб.
5	Кофе	50 руб.

Рис. 3.16

Как видно из этого примера, в состав заказа № 1 вошли борщ, гуляш и чай, а в заказ № 2 — борщ, два бифштекса и кофе. Обратите внимание, что в таблице *Заказано* было необходимо ввести дополнительное ключевое поле *Код* (суррогатный ключ), потому что в одном заказе может быть несколько одинаковых блюд.

Вопросы и задания

1. Почему собирать все данные в одной таблице во многих случаях невыгодно?
2. По какому принципу данные разбиваются на несколько таблиц?
3. Что такое внешний ключ таблицы?
4. Что такое ссылочная целостность БД? Как она обеспечивается?
5. Какие типы связей используются в многотабличных базах данных?
6. Когда применяется связь «1:1»? Какие поля при этом связываются?
7. Когда применяется связь «1:*N*»? Какие поля при этом связываются?
8. Таблицы *А* и *Б* связаны через свои ключевые поля. Что это за связь?
9. Связь между таблицами *А* и *Б* установлена через ключевое поле таблицы *Б* и неключевое поле таблицы *А*. Что это за связь?
10. Между таблицами *А* и *Б* по смыслу существует связь «многие ко многим». Как добиться этого в СУБД, которая такую связь не поддерживает?
11. Подумайте, какие изменения можно внести в базу данных, описанную в конце параграфа, если клиенты часто заказывают несколько одинаковых блюд.

Задачи

1. Фирма ведёт базу данных заказчиков, состоящую из двух связанных таблиц:

Заказчики

Код	Название	Код города
1	Иванов Т.М.	3
2	Пановко И.Т.	2
3	Черненко И.А.	3
4	Пановко А.И.	2
5	Иванова А.И.	1

Города

Код	Название
1	Москва
2	Санкт-Петербург
3	Пермь
4	Воронеж
5	Липецк

Сколько заказчиков располагается в Перми?

2. Для учёта отгруженных товаров к базе данных из предыдущего задания добавили ещё две таблицы:

Заказы

Накладная	Код заказчика	Артикул	Количество упаковок
1011	3	7576	10
1012	5	7576	20
1013	4	3889	25
1014	1	7825	30
1015	3	7576	10

Товары

Артикул	Название	Цена за упаковку
7576	Бумага	150 руб.
2325	Карандаши	200 руб.
3889	Фломастеры	350 руб.
2987	Дневники	400 руб.
7825	Пеналы	250 руб.

Определите:

а) Какие товары отправлены в каждый из городов?
б) Сколько бумаги отправлено в каждый из городов?
в) Какова общая стоимость товаров, отправленных в каждый из городов?

3. Во фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных фамилию и инициалы:

Персоны

Код	ФИО	Пол
71	Иванов Т.М.	М
85	Пановко И.Т.	М
13	Черненко И.А.	Ж
42	Пановко А.И.	Ж
23	Иванова А.И.	Ж
96	Пановко Н.Н.	Ж
82	Черненко А.Н.	М
95	Фукс Т.Н.	Ж
10	Фукс Н.А.	М
...	...	...

Дети

Код родителя	Код ребенка
23	71
13	23
85	23
82	13
95	13
85	42
82	10
95	10
...	...

а) бабушки А. И. Ивановой;
б) родного брата И. А. Черненко;
в) прадеда Т. М. Иванова;
г) внука И. Т. Пановко.

4. Во фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных фамилию и инициалы:

Персоны

Код	ФИО	Пол
86	Сизых И.Т.	М
83	Сизых А.И.	М
50	Малых А.Т.	Ж
79	Сидоров Т.М.	М
23	Сидоров А.Т.	М
13	Малых И.И.	Ж
98	Симоняк Т.Н.	Ж
11	Симоняк Н.И.	М
...	...	...

Дети

Код родителя	Код ребенка
98	83
86	13
79	50
86	83
13	50
79	23
13	23
98	13
86	11
...	...

а) племянника Н. И. Симоняка (*примечание*: племянник — сын сестры или брата);
б) всех родных братьев и сестёр Н. И. Симоняка;
в) бабушки А. Т. Малых;
г) дедушки А. Т. Сидорова.

5. Во фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных фамилию и инициалы:

Персоны

Код	ФИО	Пол
11	Косарева Л.П.	Ж
12	Левитин И.А.	М
24	Шумахер А.Ф.	Ж
45	Бланш А.А.	М
56	Васильева М.А.	Ж
83	Левитин Б.И.	М
94	Левитина В.И.	Ж
115	Кузнецов А.П.	М
140	Левитина Р.Б.	Ж
162	Левитин Л.Б.	М
171	Гайдарова З.Н.	Ж
186	Мурина С.А.	Ж
201	Кузнецов П.А.	М
...	...	...

Дети

Код родителя	Код ребенка
11	83
11	94
12	83
12	94
24	115
56	140
56	162
83	140
83	162
94	186
94	201
115	186
115	201
...	...

а) всех внуков и внучек И. А. Левитина;

б) родной сестры П. А. Кузнецова;

в) родного брата С. А. Муриной;

г) бабушки Р. Б. Левитиной.

6. Рыболов решил хранить сведения о своей добыче в базе данных. Он хочет сохранять следующую информацию о каждой рыбалке: дату, место, погоду, вес пойманной рыбы. Помогите рыболову грамотно построить многотабличную базу данных.
7. Рыболов из задачи 6 решил ещё запоминать, сколько каких рыб он поймал на каждой рыбалке. Как нужно изменить базу данных?
8. Строительной фирме нужно хранить в базе данных информацию о составе бригад рабочих (во главе с бригадиром) и о том, какая бригада какие заказы выполняла. Помогите грамотно построить многотабличную базу данных.
9. Альпинисты хотят сохранить в базе данных информацию о горных вершинах и о том, кто из них на какую вершину поднимался и в каком году. Помогите им грамотно построить многотабличную базу данных.
10. Вася получил задание составить базу данных, в которой хранится школьное расписание. Нужно учесть, что каждый предмет преподают несколько учителей, и каждый учитель может вести занятия по нескольким предметам. Помогите Васе грамотно построить многотабличную базу данных.

§ 15
Реляционная модель данных

Математическое описание базы данных

В середине XX века программное обеспечение для работы с базами данных было жёстко «привязано» к внутреннему представлению данных во внешней памяти компьютера, т. е. к структуре файлов с данными. Это означало, что при изменении формата файлов нужно было переделывать все работающие с ними программы. Поэтому возникли следующие задачи:

- разработать строгое математическое описание баз данных, независимое от способа хранения данных;
- разработать методы управления этими данными (поиска, изменения, добавления и т. п.), основанные на использовании математических операций.

Эти задачи удалось решить в 1970 г. англичанину Эдгару Кодду, который работал в фирме IBM. Он предложил новую модель данных, основанную на следующих идеях:

- все данные представляют собой свойства некоторых объектов;
- объекты делятся на классы (в теории баз данных они называются *сущностями*). Например, при описании данных о музыкальных группах можно использовать классы *Группа*, *Альбом*, *Песня* и т. п.;
- данные о некотором объекте — это набор свойств (*атрибутов*), в котором каждое свойство задаётся в виде пары «название — значение»; например, сведения о музыкальной группе «Кино» можно записать так:

(Название: «Кино», Лидер: «В. Цой», Год создания: 1981).

Такой набор данных, описывающий свойства одного объекта, называется *кортежем*.

- Порядок перечисления свойств в кортеже не имеет значения.
- Все объекты одного класса имеют одинаковый набор свойств.

- Множество кортежей, описывающих объекты одного класса, называется *отношением* (англ. *relation*); например, отношение *Группы* можно записать в виде множества кортежей:

  (Название: «Машина времени», Лидер: «А. Макаревич», Год создания: 1969)
  (Название: «Кино», Лидер: «В. Цой», Год создания: 1981)
  (Название: «Аквариум», Лидер: «Б. Гребенщиков», Год создания: 1972)
  ...
- В отношении нет двух одинаковых кортежей.
- Порядок кортежей в отношении не определён.

«Кортеж» и «отношение» — это математические понятия, которые описывают связанные данные. Поэтому с ними можно работать, используя известные операции теории множеств и математической логики. Э. Кодд предложил набор операций с данными, представленными в виде отношений, который служит основой для работы большинства современных СУБД. Модель данных, введённая Коддом, получила название **реляционной модели данных** (от англ. *relation* — отношение).

Реляционные базы данных

Реляционная база данных — это база данных, которая основана на реляционной модели, т. е. представляет собой набор отношений.

Математическая теория Кодда никак не связана с тем, как именно хранятся данные. Однако несложно понять, что отношение удобно представлять в виде прямоугольной таблицы (рис. 3.17).

Группы

Название	Лидер	Год создания
Машина времени	А. Макаревич	1969
Кино	В. Цой	1981
Аквариум	Б. Гребенщиков	1972

Рис. 3.17

Эта таблица описывает отношение *Группы*. Здесь кортежи представлены в виде **записей** (строк), а атрибуты — это **поля** (столбцы) в таблице.

Идеи реляционной теории Кодда легко перевести на «табличный язык»:

- каждая таблица описывает один класс объектов;
- порядок расположения полей в таблице не имеет значения;
- все значения одного поля относятся к одному и тому же типу данных;
- в таблице нет двух одинаковых записей;
- порядок записей в таблице не определён.

Поэтому на практике часто используют ещё одно, более простое определение:

Реляционная база данных — это база данных, которую можно представить в виде набора таблиц.

Однако не любой набор таблиц можно считать реляционной базой данных. Как мы уже говорили, БД и СУБД неразрывно связаны, и для того чтобы отнести систему базы данных (БД + СУБД) к определённому типу, необходимо выяснить, какие методы управления данными используются в соответствующей СУБД.

Согласно реляционной теории, порядок перечисления свойств в кортеже (порядок столбцов в таблице) не определён, так же как и порядок кортежей в отношении (порядок строк в таблице). Поэтому методы работы с данными в реляционной БД не должны предполагать, что столбцы и строки таблиц расположены в каком-то порядке.

Для управления данными в большинстве современных информационных систем используется **язык SQL**, в который включены команды для:

- создания новых таблиц;
- добавления новых записей;
- изменения записей;
- удаления записей;
- выборки записей из одной или нескольких таблиц в соответствии с заданным условием и некоторые другие.

Команды языка SQL позволяют управлять данными, не «привязываясь» к формату их хранения, т. е. к порядку расположения столбцов и строк в таблицах. Для выполнения операций (выборки, вставки, удаления, изменения) используются только названия столбцов и таблиц. С помощью команд SQL можно выполнить все основные операции, введённые Коддом, поэтому **СУБД** (и соответствующие системы баз данных), которые используют язык SQL, традиционно называют **реляционными**[1].

Нормализация

Представим себе, что все данные о рейсах авиакомпании «ZX-Аэро» собраны в одной таблице:

Рейс	От	До	Самолёт	Дата
ZX 001	Москва	Берлин	Boeing 737	11.12.2012
ZX 002	Москва	Санкт-Петербург	Airbus A321	12.12. 2012
ZX 003	Санкт-Петербург	Берлин	Boeing 737	13.12. 2012

Мы сразу видим, что в таблице есть *избыточность* (дублирование): многие данные (названия городов и типов самолётов) хранятся несколько раз. При этом для хранения данных-дубликатов расходуется дополнительное место на диске, что может привести к его преждевременному заполнению и выходу из строя всей информационной системы.

Есть и другая проблема: при вводе всех данных вручную оператор может сделать опечатку и, например, вместо «Санкт-Петербург» ввести «СанктПетербург». В этом случае нарушается *целостность* базы данных, потому что в ней хранится название несуществующего города.

Чтобы избежать этих проблем, при проектировании базы данных обычно выполняют её *нормализацию*.

---

[1] В то же время язык SQL нередко критикуют за то, что он не полностью соответствует реляционной модели данных. Например, в SQL используется понятие «таблица» вместо «отношение»; порядок расположения столбцов таблицы задаётся при её создании; нет запрета на создание одинаковых строк.

Нормализация — это изменение структуры базы данных, которое устраняет избыточность и предотвращает возможные нарушения целостности.

В теории реляционных баз данных существует несколько уровней нормализации (так называемых нормальных форм). Мы покажем некоторые принципы нормализации на примерах.

1. **Любое поле должно быть неделимым.** Это значит, что таблицу вида, показанного на рис. 3.18, необходимо переделывать.

Сотрудник	Телефоны
Иванов Пётр Сидорович	123-45-67, (901) 111-22-33
Петров Сидор Иванович	345-67-89, (902) 222-33-44

Рис. 3.18

Здесь поле *Сотрудник* нужно разделить на три: *Фамилия, Имя* и *Отчество*. Поле *Телефоны* содержит два телефона: домашний и мобильный. Поэтому нужно изменить таблицу, по крайней мере, так (рис. 3.19).

Фамилия	Имя	Отчество	Телефон-Дом	Телефон-Моб
Иванов	Петр	Сидорович	123-45-67	(901) 111-22-33
Петров	Сидор	Иванович	345-67-89	(902) 222-33-44

Рис. 3.19

2. **Любое неключевое поле должно зависеть от ключа таблицы.** Например, в таблице на рис. 3.20 ключ — это регистрационный номер автомобиля.

Номер	Автомобиль	Владелец	Телефон
A123AA47	«Лада-Калина»	Иванов	155-77-23
T234TT78	«Ока»	Петров	277-34-67
B345BB98	«Мерседес»	Васильев	322-98-44
A365CC47	«Ауди»	Иванов	155-77-23

Рис. 3.20

Понятно, что телефон зависит не от регистрационного номера, а от владельца. Поэтому его нужно выносить в другую таблицу (рис. 3.21).

Автомобили

Номер	Автомобиль	Владелец
А123АА47	«Лада-Калина»	1
Т234ТТ78	«Ока»	2
В345ВВ98	«Мерседес»	3
А365СС47	«Ауди»	1

Владельцы

Код	Фамилия	Телефон
1	Иванов	155-77-23
2	Петров	277-34-67
3	Васильев	322-98-44

Рис. 3.21

Заметим, что здесь мы присвоили каждому владельцу уникальный числовой код и ввели в таблицу *Владельцы* суррогатный (искусственный) ключ.

3. **Не должно быть одинаковых по смыслу полей.** Например, фирма торгует бананами, апельсинами и яблоками и хранит данные о ежедневных продажах этих товаров (в килограммах) в такой таблице (рис. 3.22).

Дата	Бананы	Апельсины	Яблоки
21.05.2011	120	78	101
22.05.2011	153	99	65
23.05.2011	87	55	123

Рис. 3.22

Недостаток этой таблицы в том, что поля *Бананы*, *Апельсины* и *Яблоки* — одинаковые по смыслу, это товары. Если фирма начнёт работать с новым видом товара, придется добавлять новый столбец, т. е. менять структуру таблицы (это делает уже не пользователь, а администратор базы данных). Поэтому нужно выделить все товары в отдельную таблицу (рис. 3.23).

Продажи

Дата	Товар	Продано
21.05.2011	1	120
21.05.2011	2	78
21.05.2011	3	101
22.05.2011	1	153
...	...	...

Товары

Код	Название
1	Бананы
2	Апельсины
3	Яблоки

Рис. 3.23

С одной стороны, число строк в таблице увеличилось, но с другой — организация данных улучшилась. Теперь для добавле-

ния в базу нового товара достаточно добавить одну запись в таблицу *Товары*, структуру таблиц переделывать не нужно.

4. **Не нужно хранить данные, которые могут быть вычислены.** Предположим, что бухгалтер фирмы вводит в таблицу доходы и расходы фирмы за каждый месяц (в тысячах рублей), а также прибыль — разницу между доходом и расходом (рис. 3.24).

Дата	Доходы	Расходы	Прибыль
03.2011	155	128	27
02.2011	178	105	73
01.2011	194	159	35

Рис. 3.24

В самом деле, поле *Прибыль* нужно удалить из таблицы, поскольку это значение можно рассчитать как разность двух других. Во-первых, оно занимает лишнее место на диске, во-вторых, появляется возможность нарушения целостности — при ошибке ввода может оказаться, что прибыль не равна разности доходов и расходов, и данные станут противоречивыми. Позже вы узнаете, что все вычисления в базах данных можно делать с помощью *запросов*, которые СУБД выполняет по требованию пользователя, т. е. тогда, когда результаты этих вычислений действительно нужны.

Нужно понимать, что нормализация имеет свои недостатки. Например, выборка данных из нескольких таблиц может выполняться очень долго, и для ускорения работы иногда приходится применять денормализацию — специально вводить избыточность, нарушая требования нормализации. Например, в предельном случае можно все данные свести в одну таблицу. Однако при этом необходимо принимать меры для поддержки целостности базы данных.

Вопросы и задания

1. Опишите проблемы, возникавшие при работе с базами данных в середине XX века. Приведите примеры. Подготовьте сообщение.
2. Расскажите об основных идеях, на которых строится реляционная модель данных.
3. Объясните понятия «кортеж», «отношение».

4. Какие ограничения накладываются на операции с реляционной базой данных?
5. Как связана реляционная модель данных и табличное представление?
6. Какими свойствами должны обладать таблицы в реляционной базе данных?
7. Какие базы данных называются реляционными?
8. Что такое нормализация? Каковы её цели?
9. Как вы понимаете выражение «поле должно быть неделимым»? Приведите примеры.
10. Почему нужно стараться, чтобы структура базы данных (состав таблиц, количество и состав полей) не менялась во время её использования?
11. Почему не нужно хранить в БД данные, которые могут быть вычислены через другие данные?

Задачи

1. Выполните нормализацию однотабличной базы данных заповедника:

Год	Животные	Район	Количество
2009	Белки	Нижняя Балка	12
2009	Бурундуки	Верхняя Балка	5
2010	Еноты	Нижняя Балка	7
2010	Еноты	Овраг	3
2010	Белки	Верхняя Балка	10

2. Выполните нормализацию однотабличной базы данных по военным кораблям:

Год спуска на воду	Название	Проект	Экипаж
1980	Удалой	1155	220 чел.
1985	Адмирал Трибуц	1155	220 чел.
1987	Североморск	1155	220 чел.
1982	Москва	1164	510 чел.
1983	Варяг	1164	510 чел.

3. Выполните нормализацию однотабличной базы данных по автомобилям:

Год	Изготовитель	Город	Модель	Скорость	Цена
2007	ВАЗ	Тольятти	1119	165 км/ч	120000 руб.
1995	ВАЗ	Тольятти	11113	130 км/ч	50000 руб.
1992	КАМАЗ	Набережные Челны	5320	90 км/ч	200000 руб.
2006	КАМАЗ	Набережные Челны	55102	90 км/ч	450000 руб.
2007	БелАЗ	Жодино	75600	64 км/ч	1200000 руб.

4. Выполните нормализацию однотабличной базы данных по участникам музыкального конкурса:

Страна	Фамилия	Инструмент	Автор произведения	Место
Россия	Иванов	Фортепьяно	Рахманинов	1
Россия	Петров	Флейта	Лист	2
Германия	Шмидт	Скрипка	Моцарт	3
США	Смит	Скрипка	Рахманинов	4
США	Браун	Гобой	Моцарт	5

§ 16
Работа с таблицей

Далее для работы с базами данных мы будем использовать свободно распространяемую СУБД Base из пакета OpenOffice.org. Аналогичные приёмы работы применяются и в популярной коммерческой СУБД Microsoft Access (которая обладает ещё бóльшими возможностями), поэтому практические задания можно выполнять в любой из этих программ. В тексте мы будем обращать внимание на некоторые различия между ними.

В СУБД OpenOffice.org Base вся база данных представляет собой один файл с расширением odb. В нём находятся:

- **таблицы**, в которых хранятся данные;
- **формы** — диалоговые окна, с помощью которых пользователь вводит и изменяет данные;

- **запросы** — обращения к базе данных, в результате которых отбираются нужные данные или выполняются какие-то другие действия, например, изменение или удаление записей;
- **отчёты** — шаблоны документов, предназначенных для вывода данных на печать.

Просмотр таблицы

После открытия файла появляется основное окно базы данных (рис. 3.25).

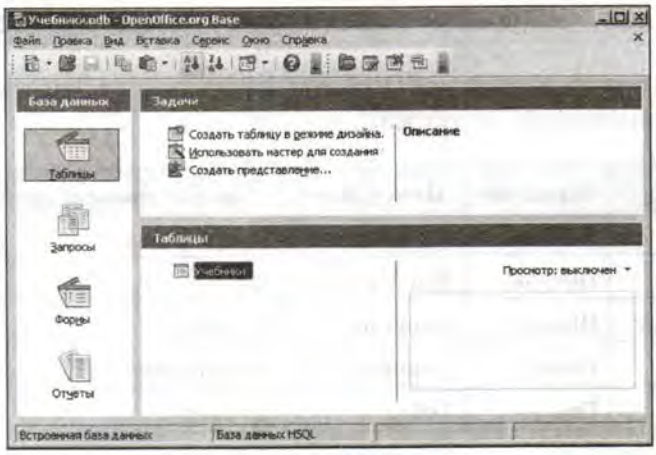


Рис. 3.25

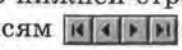
Сейчас в левой части выбрана вкладка *Таблицы*, а в правой части мы видим список всех таблиц (здесь одна таблица *Учебники*).

После двойного щелчка на таблице она открывается в отдельном окне в режиме редактирования данных (рис. 3.26).

Код	Авторы	Название
1	Чуракова Н.А.	Русский язык
2	Чуракова Н.А. и др.	Русский язык. Ч. 1, 2, 3.
3	Бетенькова Н.М., Горецкий В.Г., Фонин Д.С.	Азбука. Ч. 1, 2.
4	Соловейчик М.С., Кузьменко Н.С.	К тайнам нашего языка. Уче
5	Соловейчик М.С., Кузьменко Н.С.	К тайнам нашего языка. Ч.
6	Соловейчик М.С., Кузьменко Н.С.	К тайнам нашего языка. Ч.
7	Соловейчик М.С., Кузьменко Н.С.	К тайнам нашего языка. Ч.
8	Андрианова Т.М.	Букварь
9	Андрианова Т.М., Илюхина В.А.	Русский язык

Запись 6 из 237 (1)

Рис. 3.26

В левой части расположена область выделения, щелчок в ней выделает **текущую** (активную, рабочую) **запись**, которая обозначается треугольником. Номер текущей записи и общее число записей можно увидеть в нижней строке. Там же находятся кнопки для перехода по записям  и кнопка  для вставки новой записи (она всегда добавляется в конец таблицы).

Поиск и сортировка

При нажатии на кнопку (или на клавиши Ctrl+F) открывается окно, в котором можно задать образец и режимы поиска (искать вперёд, назад, в текущем поле или во всех полях и т. д.).

Сортировка — это расстановка записей в определенном порядке. Простейший способ сортировки — по текущему столбцу (в котором стоит курсор). Для этого служат кнопки  и  на главной панели инструментов[1], задающие соответственно алфавитный и обратный алфавитный порядок. Кнопка  позволяет применить несколько уровней сортировки, например книги одного автора отсортировать ещё и по году издания.

Важно понимать, что при сортировке *физическое расположение записей в базе данных* (*в файле на диске*) *не изменяется*, они переставляются в списке только при выводе на экран.

Фильтры

На уроках химии вы использовали фильтрацию, чтобы отделить осадок от жидкости. В базах данных фильтр служит для того, чтобы оставить нужные записи и временно скрыть ненужные.

Фильтр — это условие для отбора записей.

Самый простой вариант — это быстрый фильтр (или «фильтр по выделенному»). Он отбирает все записи, в которых значение текущего поля совпадает с активной ячейкой таблицы. Нужно сделать активной ячейку, содержащую требуемое значение, и щёлкнуть на кнопке  на панели инструментов.

Подчеркнём, *что при фильтрации записи, не удовлетворяющие заданному условию, не удаляются из таблицы, а временно*

[1] В русской версии Microsoft Access на аналогичных кнопках написаны русские буквы А и Я.

скрываются. Чтобы снова показать все записи, нужно отменить фильтр, нажав кнопку (**Отменить фильтр**). При включенном фильтре эта кнопка выделяется серым фоном (активна).

Сложный фильтр, устанавливающий ограничения на несколько полей, можно задать с помощью кнопки (**Фильтр по умолчанию**). В диалоговом окне задается сложное условие, в котором можно использовать логические операции «И» (AND) и «ИЛИ» (OR) (рис. 3.27).

Рис. 3.27

Кнопка удаляет установленный фильтр и отменяет сортировку. Помните, что сначала выполняется операция «И», а потом — операция «ИЛИ».

Фильтры служат для быстрого отбора записей по простым условиям. С каждой таблицей может храниться только один фильтр. Если нужно постоянно использовать несколько разных фильтров или более сложные условия отбора, применяют *запросы*, с которыми мы познакомимся далее.

Вопросы и задания

1. Какие объекты хранятся в файле базы данных OpenOffice.org Base?
2. Как вы думаете, какие достоинства и недостатки имеет идея хранения всех объектов БД в одном файле? Ответ обоснуйте.
3. Почему СУБД (в отличие от табличных процессоров) не разрешает вставлять новую запись в середину таблицы?
4. Что такое сортировка?
5. Изменяется ли при сортировке расположение записей в файле?
6. Что такое многоуровневая сортировка?
7. Что такое фильтр?
8. Какие варианты установки фильтров есть в СУБД OpenOffice.org Base?
9. Можно ли хранить в базе данных несколько разных фильтров для одной таблицы?

Задачи

1. Объясните, есть ли разница между следующими фильтрами?
   а) `Предмет='Математика' AND Класс=2 OR Год издания>2009`
   б) `Предмет='Математика' OR Класс=2 AND Год издания>2009`
   в) `Предмет='Математика' OR Год издания>2009 AND Класс=2`
   г) `Предмет='Математика' AND Год издания>2009 OR Класс=2`
   Есть ли среди этих фильтров два таких, которые отбирают одни и те же записи?

2. Дана таблица с результатами тестирования по различным предметам:

Фамилия	Пол	Математика	Русский язык	Химия	Информатика	Биология
Сомов	м	75	65	70	90	58
Кротов	м	83	75	59	87	60
Белочкина	ж	55	92	64	65	86
Окунев	м	75	68	72	70	56
Судакова	ж	68	70	56	58	60
Щукина	ж	76	58	78	80	85

   Сколько записей будет отобрано с помощью фильтра?
   а) `Пол = 'ж' AND Химия > Биология`
   б) `Пол = 'ж' OR Химия > Биология`
   в) `Пол = 'м' AND Математика > Информатика`
   г) `Пол = 'м' OR Математика > Информатика`
   д) `Пол = 'ж' AND Русский язык > 70 OR Информатика > 80`
   е) `Пол = 'ж' OR Русский язык > 70 AND Информатика > 80`
   ж) `Пол = 'м' AND Информатика > 80 OR Русский язык > 60`
   з) `Пол = 'м' OR Информатика > 80 AND Русский язык > 60`

3. Какой по счёту будет запись с фамилией Белочкина, если отсортировать таблицу из предыдущего задания по полю:
   а) *Фамилия* (по алфавиту);
   б) *Математика* (по убыванию);
   в) *Русский язык* (по убыванию);
   г) *Химия* (по возрастанию);
   д) *Информатика* (по возрастанию);
   е) *Биология* (по убыванию)?

§ 17
Создание однотабличной базы данных

Работать с готовой базой данных вы уже научились, теперь займёмся созданием новой базы «с нуля». Построим однотабличную базу данных *Футбол*, в которую запишем количество побед, ничьих и поражений нескольких футбольных команд за последний сезон, а также среднюю зарплату футболистов (рис. 3.28).

	Команда	Победы	Ничьи	Поражения	Зарплата
	Аметист	10	7	3	13 290 руб.
	Бирюза	5	8	7	12 500 руб.
	Восход	13	5	2	22 000 руб.
	Закат	7	8	5	18 780 руб.
	Коллектор	11	6	3	20 200 руб.
	Кубань	6	12	2	14 000 руб.
	Малахит	12	3	5	17 340 руб.
	Ротор	8	12	0	15 820 руб.
	Статор	9	10	1	19 300 руб.
	Финиш	12	0	8	12 950 руб.

Рис. 3.28

Запустим пакет OpenOffice.org и выберем создание новой базы данных. **Мастер** (программный модуль, который облегчает выполнение стандартных действий) предлагает на выбор три варианта:

- создать новую базу данных;
- открыть существующую базу;
- подключиться к существующей базе, например, к БД Microsoft Access, MySQL или к адресной книге почтовой программы.

Выберем первый пункт. После этого предлагается *зарегистрировать* базу данных — в этом случае вы сможете использовать данные из неё в других программах OpenOffice.org.

В новой базе ещё нет ни одной таблицы. Создать новую таблицу можно двумя способами, которые перечислены в области **Задачи** (рис. 3.29).

Рис. 3.29

Режим дизайна иначе называется **Конструктор**, это означает, что всю работу приходится выполнять вручную. Мастер позволяет быстро построить таблицу на основе одного из готовых шаблонов.

Выбираем режим дизайна. Появляется окно, где нужно в первом столбце ввести названия всех полей новой таблицы, а во втором — выбрать их тип из выпадающего списка (рис. 3.30). Здесь использованы три различных типа данных:

- **Текст [VARCHAR]** — текстовая строка;
- **Целое [INTEGER]** — целое число;
- **Десятичное [DECIMAL]** — десятичное число, которое хранится с заданным числом знаков в дробной части и применяется для работы с денежными суммами.

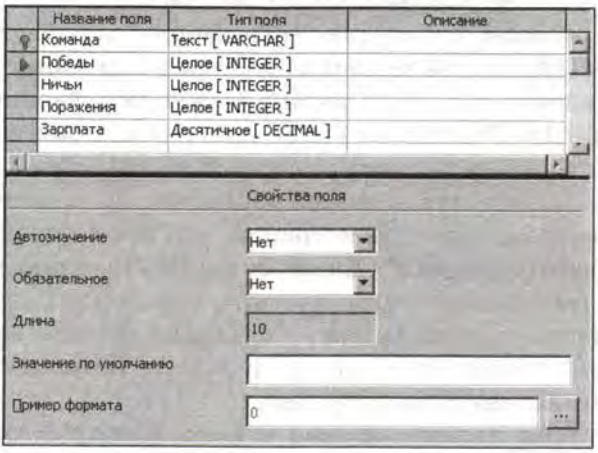


Рис. 3.30

Среди других форматов отметим следующие:

- **Вещественное [REAL]** — вещественное число (может иметь дробную часть);
- **Памятка [LONGVARCHAR]** — текст большого размера;
- **Картинка [LONGVARBINARY]** — двоичные данные большого размера;
- **Логическое [BOOLEAN]** — логическое значение («да»/«нет»);
- **Дата [DATE]**;
- **Время [TIME]**;
- **Дата/Время [TIMESTAMP]**.

Поле *Команда* — это первичный ключ. Чтобы сделать поле ключевым, нужно щёлкнуть правой кнопкой мыши в области выделения (слева от названия поля) и выбрать пункт **Первичный ключ** в контекстном меню. Если выделить несколько полей (щёлкая в области выделения при нажатой клавише Ctrl), можно таким же способом создать *составной ключ* таблицы, состоящий из всех выделенных полей.

В нижней части окна Конструктора настраиваются свойства выделенного поля, например:

- максимальный размер для текста;
- количество знаков в дробной части для десятичного числа;
- значение по умолчанию (которое автоматически вписывается в поле при создании новой записи).

Если в таблице в качестве первичного ключа используется поле-счётчик (его значение увеличивается на единицу при добавлении новой записи), для него нужно установить свойство **Автозначение** равным **Да**. Такое поле будет заполняться автоматически.

Можно потребовать, чтобы значение какого-то поля обязательно было задано для каждой добавляемой записи (вспомните обязательные поля при регистрации на веб-сайтах). Для этого нужно установить значение **Да** свойства **Обязательное**.

Формат вывода значения на экран (например, число знаков в дробной части, выравнивание, выделение отрицательных значений красным цветом и т. п.) задаётся с помощью кнопки  в нижней части окна Конструктора.

Для создания индекса нужно в окне Конструктора щёлкнуть на кнопке  на панели инструментов. Появится окно, в котором можно создавать, удалять и изменять индексы. Индекс по первичному ключу создается автоматически. Любой индекс можно сделать уникальным, в этом случае СУБД не допустит повторения значений индекса.

При закрытии окна Конструктора будет предложено сохранить таблицу и ввести ее название, затем новая таблица появляется в списке таблиц. Для работы с таблицами в окне базы данных служат кнопки верхней панели инструментов:

 — открыть таблицу в режиме редактирования данных;

 — перейти в режим дизайна (в окно Конструктора);

 — удалить таблицу;

 — переименовать таблицу.

Эти же операции можно выполнить с помощью контекстного меню, щёлкнув на имени таблицы правой кнопкой мыши. Для входа в Конструктор нужно выбрать команду **Изменить**.

Вопросы и задания

1. Что такое мастер?
2. Что значит зарегистрировать базу данных?
3. Какие способы создания таблиц вы знаете? Чем они различаются? Приведите примеры.
4. Зачем каждому полю таблицы присваивается некоторый тип данных?
5. Какие типы данных поддерживает OpenOffice.org Base?
6. В чем особенность значений типа **DECIMAL**? Зачем они используются?
7. Как сделать поле первичным ключом? Как изменить ключ таблицы?
8. Как изменить свойства поля?
9. Что обозначают свойства **Автозаполнение** и **Обязательное**?
10. Как изменить формат вывода значений поля?
11. Зачем может понадобиться создавать новый индекс? Как это сделать?
12. Какой индекс в таблице строится автоматически?
13. Как перейти в режим дизайна с помощью контекстного меню?

Задача

Создайте новую базу данных *Футбол* и сохраните её в своей папке. Выполните следующие задания:

а) постройте таблицу, которая рассмотрена в тексте параграфа;

б) определите правильный тип полей, сделайте поле *Команда* первичным ключом;

в) заполните таблицу данными (например, можно использовать данные рис. 3.28);

г) отсортируйте записи по убыванию количества побед;

д) примените фильтр, который отбирает только команды, имеющие более 10 побед и меньше 5 поражений.

§ 18
Запросы

Для пользователя любой информационной системы в первую очередь важно, чтобы он смог выбрать из базы данных ту информацию, которая ему в данный момент нужна. Для этого используются запросы.

Запрос — это обращение к СУБД для отбора записей или выполнения других операций с данными.

Как вы уже знаете, в большинстве современных СУБД для управления данными (т. е. для составления запросов) используется **язык SQL** (англ. *Structured Query Language* — язык структурных запросов). Изучение этого языка выходит за рамки школьного курса, поэтому мы будем использовать в основном визуальные средства составления запросов программы OpenOffice.org Base. В то же время вы сможете посмотреть, как выглядит составленный вами запрос на языке SQL.

Конструктор запросов

Продолжим работу с базой данных *Футбол*, которую мы недавно создали. Перейдём в окне базы данных на вкладку **Запросы** и выберем в области **Задачи** вариант **Создать запрос в режиме дизайна**. Появится окно Конструктора, программа предложит добавить в рабочую область таблицу, из которой будут выбираться данные.

Названия полей, которые нужно включить в запрос, можно перетаскивать мышью в пустые столбцы бланка, расположенного в нижней части окна (рис. 3.31).

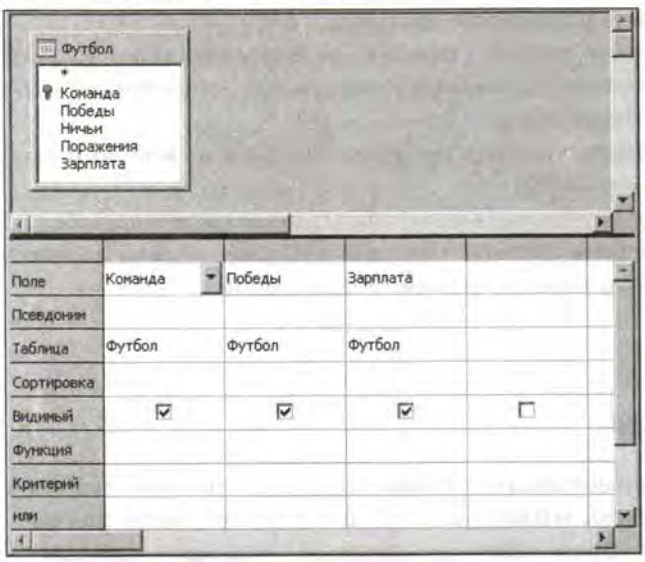


Рис. 3.31

В нашем случае в запрос добавлены поля *Команда*, *Победы* и *Зарплата*.

Чтобы увидеть результаты запроса, нужно нажать клавишу F5 или щёлкнуть на кнопке  на панели инструментов. Результат выполнения запроса — это таблица, которая не хранится в памяти, а строится в момент выполнения запроса. Однако она напрямую связана с данными: если вы измените что-то в результатах запроса, эти изменения будут внесены в базу данных. Такие изменения разрешены только тогда, когда в запросе есть первичный ключ таблицы.

Запрос, который мы построили, программа переводит на язык SQL и отправляет СУБД для выполнения. Чтобы увидеть SQL-запрос, нужно щёлкнуть на кнопке  (повторный щелчок возвращает обратно в режим Конструктора). В данном случае мы увидим такой запрос:

```
SELECT "Команда", "Победы", "Зарплата" FROM "Футбол"
```

В английском языке слово *select* означает «выбрать», а *from* — «из». Таким образом, здесь выбираются три поля из таблицы *Футбол*. В режиме SQL запрос можно редактировать вручную, если вы знаете язык SQL.

Перейдём обратно в Конструктор и сделаем так, чтобы список команд был отсортирован по убыванию числа побед. Для этого в столбце *Победы* нужно установить параметр **Сортировка** равным **по убыванию** (выбрать из выпадающего списка). Теперь можно заново выполнить запрос, нажав F5, и посмотреть на результат.

Если закрыть окно Конструктора, программа предложит сохранить запрос и ввести его имя. После этого запрос появится в списке запросов в основном окне базы данных. Двойной щелчок на имени запроса открывает окно с результатами. Нажав правую кнопку мыши на заголовке столбца таблицы в этом окне, можно настроить формат столбца (тип данных, выравнивание).

Чтобы перейти в Конструктор, нужно выделить название запроса и щёлкнуть на кнопке  на панели инструментов или выбрать пункт **Изменить** из контекстного меню.

Критерии отбора

Теперь отберём только те команды, которые одержали более 10 побед. Для этого в столбце *Победы* зададим критерий отбора *>10* (строка **Критерий**) и проверим запрос.

Добавим второе условие отбора в той же строке **Критерий**: для поля *Зарплата* установим критерий *>15000*. Теперь СУБД отберёт только те записи, для которых одновременно выполняются оба критерия, т. е. условия в одной строке объединяются с помощью логической операции «И» (AND).

Перейдём в режим SQL, найдём в условии слово **AND** и заменим его на **OR** («ИЛИ»). Вернувшись в Конструктор, обнаружим, что условие *>15 000* «переехало» на одну строчку ниже (у этой строчки заголовок **или**). Снова выполним запрос и убедимся, что теперь отбираются команды, для которых выполняется хотя бы одно из двух условий. Таким образом, условия, записанные в одной строке, объединяются с помощью операции «И», а условия в разных строках — с помощью операции «ИЛИ».

Для текстовых данных можно указывать не только точное значение, но и шаблон (вспомните, как строятся маски имён файлов). Например, если в критерий отбора для поля *Команда* ввести

```
LIKE 'К*'
```

то будут отобраны только те команды, название которых начинается с буквы «К». Здесь слово `LIKE` (из языка SQL) обозначает «такой, как...», «похожий на...», а звёздочка — любое количество любых символов. Кроме звёздочки можно использовать знак вопроса, обозначающий один любой символ.

Обратите внимание, что в бланке запроса есть строка **Видимый**, где в каждом столбце расположен флажок (выключатель). Он отключается тогда, когда для данного поля нужно задать условие отбора или сортировку, а выводить его на экран не нужно.

Запросы с параметрами

Создадим запрос, отбирающий команды, в которых зарплата больше заданной, скажем, больше 15 000 рублей. Для этого в критерий отбора для поля *Зарплата* нужно добавить условие *>15 000*.

Если мы захотим изменить это значение, нужно будет изменить запрос. Чаще всего пользователь не имеет права изменять запросы (это делает только администратор базы данных), поэтому возникает проблема: как дать пользователю возможность менять значение в запросе, не меняя сам запрос? Чтобы решить эту задачу, применяют **запросы с параметрами**.

Параметры — это данные, которые пользователь вводит при выполнении запроса.

В конструкторе запроса параметр задаётся с помощью двоеточия, за которым следует имя запроса, например:

```
>=:Минимальная_зарплата.
```

Для соединения двух слов используется знак подчёркивания, потому что в OpenOffice.org Base имя параметра не может включать пробелы[1].

Когда выполняется запрос, на экране появляется окно, в котором пользователь должен ввести значения всех параметров (рис. 3.32).

Рис. 3.32

Вычисляемые поля

В теоретической части этой темы мы говорили о том, что в базе данных не нужно хранить значения, которые можно вычислить по другим известным данным. Например, найдём количество очков, которые набрала каждая команда в чемпионате, учитывая, что за победу начисляется 3 очка, а за ничью — одно очко.

В бланке запроса перетащим поле *Зарплата* за его заголовок вправо, освободив 3 столбца. В первые два пустых столбца добавим поля *Ничьи* и *Поражения*, а в третьем вместо имени поля введем нужную нам формулу

```
Ничьи+3*Победы
```

В строке **Псевдоним** можно ввести осмысленный заголовок этого столбца — *Очки*, который и будет появляться в таблице с результатами запроса.

С новым столбцом **(вычисляемым полем)** можно делать всё, что и с обычными столбцами, соответствующими реальным полям

[1] В Microsoft Access этого ограничения нет, там имя параметра заключается в квадратные скобки.

таблицы, — сортировать, устанавливать условия отбора[1]. Например, можно выбрать сортировку по убыванию, чтобы в начале таблицы оказались команды с самым высоким результатом.

Другие типы запросов

Мы рассмотрели только один вид запросов — запросы на выборку данных. Однако язык SQL поддерживает и другие команды, с помощью которых можно полностью управлять базой данных: создавать и удалять таблицы; добавлять, изменять и удалять записи и т. д.

Чтобы использовать эти возможности, в OpenOffice.org Base нужно выбрать пункт **Сервис – SQL** в меню. Кроме того, можно также составить запрос на выборку нужных записей в *Конструкторе*, затем перейти в режим просмотра SQL-запроса (кнопка), вручную изменить запрос нужным образом (например, из запроса на выборку сделать запрос на удаление) и щёлкнуть на кнопке **(Выполнить команду SQL)**.

Вопросы и задания

1. Что такое запрос? Зачем используются запросы?
2. На каком языке составляются запросы к СУБД?
3. Можно ли строить и изменять запрос, не используя Конструктор?
4. Объясните, зачем нужны строки **Псевдоним**, **Сортировка** и **Критерий** в бланке запроса.
5. Как вы думаете, ограничивает ли структура бланка (одна строка **Критерий** и несколько строк **или**) возможность создания сложных запросов? Ответ обоснуйте (вспомните материал по математической логике).
6. Как увидеть результаты выполнения запроса?
7. Как изменить существующий запрос?
8. Как вы думаете, каков будет результат выполнения следующего SQL-запроса?

```
SELECT "Команда" FROM "Футбол" WHERE "Победы" > 10
```

Проверьте ваш ответ с помощью программы OpenOffice.org Base.

9. Зачем нужны запросы с параметрами?
10. Что такое вычисляемое поле? Как его построить в Конструкторе?
11. Какие операции кроме выборки данных можно выполнить с помощью SQL-запросов?

[1] В OpenOffice.org Base числа в условиях отбора для вычисляемого поля нужно ставить в апострофы: *>'10'*.

Задачи

1. Откройте базу данных *Футбол*, созданную ранее. Постройте следующие запросы и настройте формат вывода данных:
   а) запрос с именем *Запрос85*, который отбирает всю информацию о командах, имеющих более 8 побед и меньше 5 поражений; команды должны быть расставлены по убыванию числа побед;
   б) запрос с именем *ЗапросЗарплата*, который отбирает команды, где зарплата игроков не меньше суммы, введённой пользователем; команды должны быть расставлены по убыванию зарплаты;
   в) запрос с именем *ЗапросОчки*, в котором для каждой команды вычисляется количество набранных очков (за победу — 3 очка, за ничью — 1 очко); команды должны быть расставлены по убыванию количества набранных очков.

   Посмотрите, как эти запросы записываются на языке SQL.
2. Проверьте, какие данные отбирает такой SQL-запрос:
   ```
   SELECT * FROM "Футбол"
   ```
   Посмотрите, как он выглядит в бланке Конструктора. Сделайте выводы.
3. Постройте запрос, отбирающий данные о всех командах, в названии которых есть буква «а» (возможно, в середине). Посмотрите, как он запишется на языке SQL.
4. Постройте запрос, отбирающий данные о всех командах, в названии которых третья буква — «а». Посмотрите, как он запишется на языке SQL.

§ 19
Формы

Пользователи информационных систем для работы с базой данных используют прикладные программы, которые для выполнения любых операций с данными обращаются к СУБД. Некоторые программы, в том числе OpenOffice.org Base и Microsoft Access, могут выполнять функции как СУБД, так и прикладной программы. В них можно построить интерфейс для пользователя, предоставив ему возможность ввода, изменения и удаления записей с помощью диалоговых окон, которые называются **формами**.

Форма создаётся на основе таблицы или запроса. Как и все объекты базы данных, форму можно строить вручную (в **режиме дизайна**) или с помощью **мастера**. На этот раз мы будем использовать второй метод. Перейдите на вкладку **Формы** и выберите (в области **Задачи**) вариант **Использовать мастер для создание формы**.

В первом окне мастера нужно выбрать источник данных — таблицу или запрос (рис. 3.33).

Рис. 3.33

Кнопки со стрелками позволяют добавить поля на форму или отменить выбор. Кнопка с одной стрелкой перемещает только выделенные поля, а кнопка со сдвоенной стрелкой — сразу все поля. Для перехода к следующему шагу мастера нужно нажимать кнопку **Дальше**.

Второй этап (**Установка субформы**, подчинённой формы) мы пропустим, просто нажав на кнопку **Дальше**. Затем нужно выбрать расположение полей с данными, определить стиль оформления и задать имя формы. После нажатия на кнопку **Готово** на последнем шаге мы увидим форму, которую построил мастер (рис. 3.34).

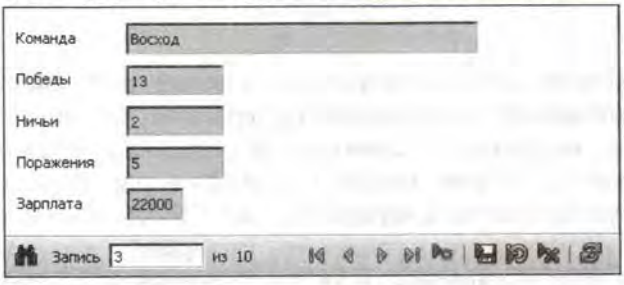


Рис. 3.34

В нижней части окна видны кнопки для поиска, перехода между записями, сохранения и отмены сделанных изменений, удаления записи и др.

Так же как таблицы и запросы, формы можно редактировать в Конструкторе. Для перехода в Конструктор нужно выделить название формы в окне базы данных и щёлкнуть на кнопке на панели инструментов или выбрать пункт **Изменить** из контекстного меню.

На нижней панели инструментов в окне Конструктора расположены кнопки для настройки формы и её элементов. Кнопка позволяет переключаться в режим просмотра данных и обратно. С помощью кнопки можно изменить общие **свойства формы**: название, источник данных, разрешения на добавление, изменение и удаление записей.

Для того чтобы изменить фон формы, нужно щёлкнуть правой кнопкой мыши на свободном месте на форме, выбрать пункт **Страница** из контекстного меню и перейти на вкладку **Фон**. На ней можно выбрать цвет фона или фоновый рисунок.

Чтобы изменить **свойства отдельного элемента**, нужно сначала этот элемент выделить. Вы можете заметить, что щелчок мышью выделяет сразу два элемента: поле и связанную с ним надпись. Чтобы работать с ними по отдельности, нужно при выделении удерживать нажатой клавишу Ctrl. Элементы можно перетаскивать мышью, изменять их размеры за маркеры на рамке.

Для настройки свойств выделенного элемента нужно щёлкнуть на кнопке на нижней панели инструментов или выбрать пункт **Элемент управления** из контекстного меню. На вкладке **Общие** задаётся название элемента, шрифт, цвет фона, размеры, координаты, формат данных, выравнивание и т. п. Вкладка **Данные** определяет источник данных для элемента (название поля).

Кнопка открывает окно **Навигатора форм**. В нём показаны названия всех элементов, щелчком мышью можно выбрать любой из них. С помощью кнопки на форму добавляется новое поле.

Панель **Элементы управления** позволяет добавить на форму флажки (для логических полей), радиокнопки (выбор одного из вариантов), списки, метки и другие элементы (рис. 3.35).

Рис. 3.35

Растровый рисунок можно добавить на форму с помощью команды меню **Вставка – Изображение**. Панель инструментов **Рисование** предназначена для работы с векторной графикой (рис. 3.36).

Рис. 3.36

Таким образом, форма позволяет не только получить доступ к данным, но и оформить их в виде, удобном для пользователя.

Вопросы и задания

1. Что такое форма? Зачем используются формы?
2. Что служит источником данных для формы?
3. Как изменить фон формы?
4. Как изменить источник данных формы?
5. Как выделить поле данных, не выделяя связанную с ним метку?
6. Зачем нужен Навигатор форм?
7. Как можно добавить на форму растровые и векторные рисунки? Попробуйте построить форму, в которой используется графика.

Задача

Постройте форму для просмотра и ввода данных в таблицу *Футбол*. Оформите её так, чтобы работа с данными была как можно проще и понятнее.

§ 20 Отчёты

Отчёт — это документ, предназначенный для вывода данных на печать.

Подготовка отчётов — это задача прикладной программы, а не СУБД. Однако OpenOffice.org Base и Microsoft Access совмещают обе функции, поэтому в них можно оформлять и печатать отчеты. На рисунке 3.37 показан отчёт, построенный на основе запроса к базе данных *Футбол*.

Турнирная таблица

Автор: Василий
Дата: 12.10.11

Команда	Победы	Ничьи	Поражения	Очки	Зарплата
Восход	13	5	2	44	22 000 руб.
Малахит	12	3	5	39	17 340 руб.
Коллектор	11	6	3	39	20 200 руб.
Статор	9	10	1	37	19 300 руб.
Аметист	10	7	3	37	13 290 руб.
Финиш	12	0	8	36	12 950 руб.
Ротор	8	12	0	36	15 820 руб.
Кубань	6	12	2	30	14 000 руб.
Закат	7	8	5	29	18 780 руб.
Бирюза	5	8	7	23	12 500 руб.

Страница 1/1

Рис. 3.37

Для создания отчёта перейдём в окне базы данных на вкладку **Отчеты** и используем мастер (ссылка для запуска мастера находится в области **Задачи**).

Первый шаг напоминает работу мастера создания форм — нужно выбрать источник данных (таблицу или запрос) и определить нужные поля. На следующих шагах задаются метки для обозначения полей (по умолчанию имена полей), группировка (мы пока её не будем использовать), сортировка, стиль оформления. На последнем шаге нужно определить имя отчёта.

Новый отчёт появляется в окне базы данных на вкладке **Отчеты**. Если дважды щёлкнуть на его названии, отчет открывается на экране в готовом виде, его сразу же можно распечатать.

Для того чтобы изменить структуру и оформление отчёта, нужно выделить его название в окне базы данных и щёлкнуть на кнопке  на панели инструментов или выбрать пункт **Изменить** из контекстного меню. Отчет редактируется в текстовом процессоре, вы можете задать название отчёта и имя автора, а также изменить расположение и оформление элементов, например выравнивание, шрифт и т. п.

Вопросы и задания

1. Что такое отчёт?
2. Откуда берутся данные, которые выводятся в отчете?
3. Как изменить расположение и оформление элементов отчета?

Задача

Постройте отчёт вывода на печать данных из запроса *ЗапросОчки*. Оформите его примерно так, как показано на рис. 3.37.

§ 21
Работа с многотабличной базой данных

Таблицы и связи между ними

Вспомним про базу данных кафе, о которой мы говорили в конце § 14 (рис. 3.38).

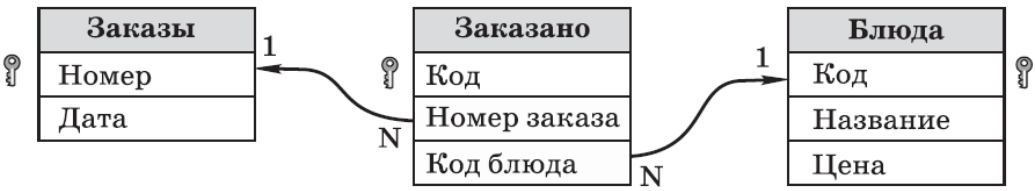


Рис. 3.38

Построим в новой базе данных (назовём её *Кафе*) все необходимые таблицы (пока без связей). Поскольку в этой базе несколько таблиц, далее мы будем использовать общепринятые обозначения типа *Блюда*.[*Название*] — это означает «поле *Название* таблицы *Блюда*».

Не забудьте, что связи устанавливаются только между однотипными полями, т. е. поля *Заказано*.[*Номер заказа*] и *Заказано*.[*Код блюда*] должны быть целого типа (**INTEGER**), чтобы их можно было связать соответственно с номером заказа и кодом блюда. Для поля *Блюда*.[*Цена*] выберите десятичный тип (**DECIMAL**) и денежный формат для вывода на экран.

Чтобы установить связи между таблицами, выберем пункт меню[1] **Сервис – Связи**. С помощью специального окна добавим в рабочую область (она пока пустая) все три таблицы (рис. 3.39).

Рис. 3.39

Теперь можно «схватить» мышью название какого-то поля и перетащить его на название поля другой таблицы, с которым нужно установить связь. С помощью этого метода установим все связи, показанные на схеме в начале параграфа. После этого окно можно закрыть, сохранив изменения.

Чтобы изменить или удалить связи, снова зайдите в меню **Сервис – Связи**. Для удаления связи её нужно выделить щелчком мышью и нажать клавишу Delete.

Теперь остаётся заполнить таблицы (можно придумать свои данные или взять их из § 14).

Запрос данных из нескольких таблиц

Мы построим два запроса к базе данных *Кафе* — простой и итоговый. Начнём с простого запроса, в котором будет собрана информация по всем сделанным заказам (рис. 3.40).

Заказ	Дата	Блюдо	Цена
1	11.12.12	борщ	80 руб.
1	11.12.12	гуляш	70 руб.
1	11.12.12	чай	10 руб.
2	12.12.12	борщ	80 руб.
2	12.12.12	бифштекс	110 руб.
2	12.12.12	бифштекс	110 руб.
2	12.12.12	кофе	50 руб.

Рис. 3.40

[1] В программе Microsoft Access нужно щёлкнуть на кнопке **Схема данных** на вкладке **Работа с базами данных**.

Посмотрим, откуда нужно взять данные. Номер заказа (в столбце *Заказ*) и дата хранятся в таблице *Заказы*, а название блюда и цена — в таблице *Блюда*. Перейдём к созданию запроса в режиме дизайна и добавим в рабочую область две названные таблицы (рис. 3.41).

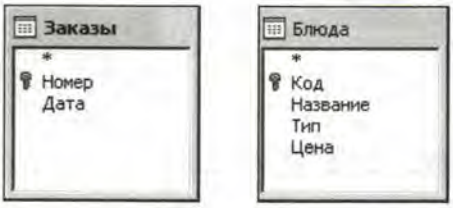


Рис. 3.41

Теперь добавим в столбцы бланка все нужные поля и выполним запрос. Результаты окажутся неожиданными — запрос выдаёт слишком много записей, больше, чем есть на самом деле. Дело в том, что таблицы в запросе оказались несвязанными: мы не добавили таблицу *Заказано*. Из неё не берутся данные, но она служит для связи таблиц в единую систему.

Чтобы исправить ситуацию, выберем пункт меню **Вставка – Добавить таблицу или запрос** и добавим таблицу *Заказано*. После этого запрос отберёт данные правильно. Остаётся только дать столбцам запроса понятные названия, для этого используется строка **Псевдоним** в бланке запроса.

Теперь окно конструктора запросов можно закрыть, при сохранении дайте запросу название *ЗапросЗаказы*. Обратите внимание, что нельзя называть запрос именем, которое совпадает с именем таблицы (например, *Заказы*) или другого запроса.

Двойной щелчок на имени запроса запускает его на выполнение. Щёлкнув правой кнопкой мыши на заголовке столбца, можно изменить его оформление. Установите таким образом денежный формат для поля *Цена* и выравнивание вправо для всех числовых данных.

Итоговый запрос

Теперь построим запрос, который считает общую стоимость каждого заказа (рис. 3.42).

	Заказ	Дата	К оплате
▶	1	11.12.12	160 руб.
	2	12.12.12	350 руб.

Рис. 3.42

Как видно из рис. 3.42, нам нужны номер и дата заказа (из таблицы *Заказы*) и цены блюд (из таблицы *Блюда*), которые нужно как-то сложить. Посмотрев на схему соединения таблиц, легко понять, что для правильного объединения данных нужно добавить в запрос таблицу *Заказано*, хотя ни одно её поле в результате запроса не выводится.

Создадим новый запрос в режиме дизайна и добавим в рабочую область таблицы *Заказы*, *Заказано* и *Блюда*. Перетащим в бланк запроса нужные поля и введём правильные заголовки столбцов (как на образце) в строке *Псевдоним*. Если выполнить этот запрос, мы увидим цены отдельных блюд, а не общую стоимость.

Чтобы подсчитать сумму цен по каждому заказу, в поле *Цена* найдём строку **Функция**[1] и там выберем вариант **Сумма**. Кроме того, надо указать, какие группы записей объединять при суммировании. Для этого в той же строке **Функция** в полях запроса *Номер* и *Дата* выберем вариант **Group** (Группировка). Это значит, что сумма считается для каждой уникальной комбинации «номер запроса – дата». Поскольку у нас номер заказа уникальный, фактически будет рассчитана сумма каждого заказа. Выполним запрос и проверим, что он работает правильно. Теперь можно закрыть окно Конструктора, сохранив запрос под именем *ЗапросКОплате*. Запустим запрос двойным щелчком и настроим формат вывода данных.

Кроме функции **Сумма** в итоговых запросах можно использовать и другие функции, например количество, среднее значение, минимум, максимум.

Формы

Построим форму, показанную на рис. 3.43, в которой для каждого номера заказа выдаётся его дата, состав заказанных блюд с ценами и общая сумма.

Форма создаётся на основе таблицы или запроса. В данном случае мы хотим получить информацию о заказе, поэтому основной источник данных — это таблица — *Заказы*.

Информацию о названиях блюд и ценах можно было бы взять из таблицы *Блюда*,

Рис. 3.43

[1] В Microsoft Access эта строка называется **Групповая операция**. Чтобы она появилась в бланке запроса, нужно щёлкнуть по кнопке Итоги на вкладке Конструктор.

но она напрямую не связана с таблицей *Заказы*, поэтому так сделать не получится. Однако у нас есть запрос *ЗапросЗаказы*, где эти данные объединены, поэтому состав заказа и цены на отдельные блюда мы возьмём из этого запроса.

Как получить общую сумму? Вспомним про *ЗапросКОплате*, где сумма для каждого заказа уже найдена. Таким образом, форма объединяет информацию из таблицы *Заказы* и двух запросов. Связь между ними устанавливается по номеру заказа — это поле есть везде.

Как мы уже упоминали, основной источник данных — это таблица *Заказы*. Щёлкнем на её имени правой кнопкой мыши и выберем пункт **Мастер форм** из контекстного меню. Включим в форму все поля.

На втором шаге мастер предлагает добавить **субформу** (подчинённую форму) — дополнительный источник данных, связанный с главной формой. Отметим флажок **Добавить субформу** (рис. 3.44).

Рис. 3.44

Мастер определил, что для таблицы *Заказы* подчинённой является таблица *Заказано* (они связаны отношением 1:*N*) и предлагает выбрать эту таблицу. Однако нам нужно использовать запрос, поэтому отметим второй вариант: **Субформа, основанная на выбранных полях**. Далее выберем поля *Заказ*, *Блюда* и *Цена* из запроса (рис. 3.45).

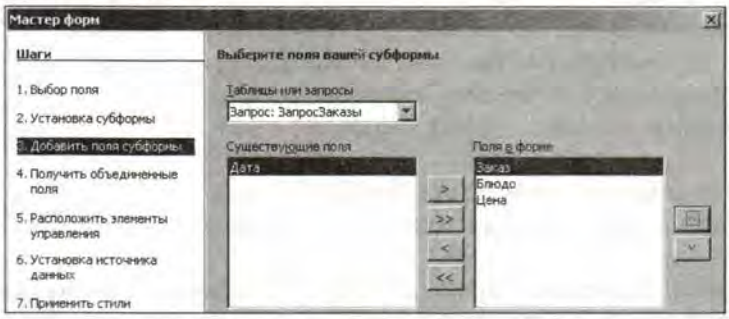


Рис. 3.45

Несмотря на то что поле *Заказ* не будет выводиться на экран, его нужно включить в субформу, потому что через это поле на следующем шаге работы мастера устанавливается её связь с главной таблицей (рис. 3.46).

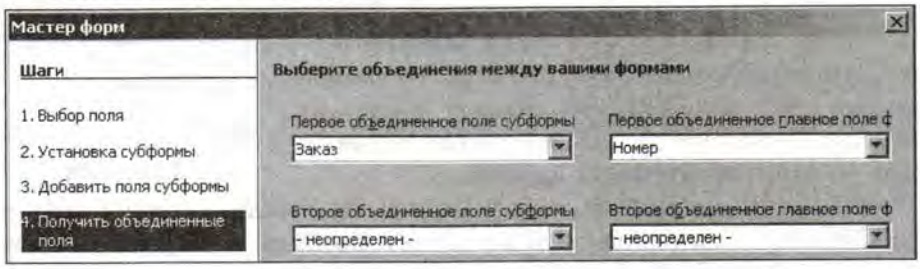


Рис. 3.46

Затем выберем расположение элементов главной и подчинённой форм (рис. 3.47).

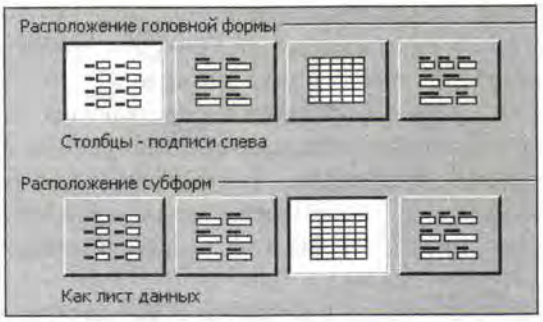


Рис. 3.47

После завершения работы мастера нужно отрегулировать размеры полей и настроить формат вывода. Например, для правильного отображения цены нужно войти в конструктор форм, щёлкнуть правой кнопкой мыши на столбце *Цена* и выбрать пункт **Заменить на – Поле валюты**. Количество десятичных знаков можно изменить с помощью свойства **Точность** в окне свойств столбца (пункт **Столбец** в контекстном меню). Столбец *Заказ* в подчинённой форме нужно просто удалить (пункт **Удалить столбец** в контекстном меню).

Осталось вывести общую стоимость заказа. Напомним, что её нужно взять из другого источника данных — запроса *ЗапросКОплате*. Это значит, что необходимо добавить к форме ещё одну

субформу. Мастер тут не поможет, придётся выполнить эту операцию вручную.

Откроем форму в **режиме дизайна (Конструктора)** и запустим **Навигатор форм**, щёлкнув на кнопке на нижней панели инструментов (рис. 3.48). Здесь **MainForm** — это главная форма, а **SubForm** — подчинённая (её содержимое можно раскрыть, щёлкнув на знаке «+»). Остальные ветви дерева — это метки и поля данных.

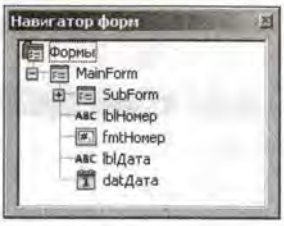


Рис. 3.48

Щёлкнем правой кнопкой мыши на имени главной формы и выберем в контекстном меню пункт **Создать – Форма**. В Навигаторе форм появится новая форма с именем *Форма*, но на экране ничего не меняется, ведь мы пока не добавили в эту форму ни одного элемента.

Выделите название новой формы и создайте метку (надпись) с текстом «К оплате». Для этого надо щёлкнуть на кнопке на панели **Элементы управления** (обычно она расположена слева) и выделить область по размеру метки (под таблицей). Свойства метки настраиваются с помощью кнопки на нижней панели инструментов или пункта **Элемент управления** из контекстного меню. Обратите внимание, что в окне Навигатора форм эта метка принадлежит новой субформе. Теперь аналогично добавьте на форму поле валюты, в котором будет выводиться сумма оплаты для выбранного заказа. Кнопка для вставки такого поля находится на панели дополнительных элементов управления:

Чтобы вывести эту панель на экран, щёлкните по кнопке . Перед вставкой нового поля в окне *Навигатора форм* нужно выделить вторую субформу, которой это поле должно принадлежать.

Осталось связать новую субформу с источником данных. Для этого выделим субформу в окне Навигатора форм и выберем **Свойства** в контекстном меню. На вкладке **Данные** установим тип содержимого — запрос, в следующей строчке выберем *ЗапросКОплате* (рис. 3.49).

Рис. 3.49

Теперь в окне свойств добавленного поля валюты на вкладке **Данные** надо выбрать поле данных *КОплате*. После этого форма полностью работоспособна.

Отчёты

Используя подстроенный ранее запрос, составим подробный отчёт по всем заказам (рис. 3.50). Обратите внимание, что заказы сгруппированы по датам, кроме того, данные о блюдах, относящихся к одному заказу, тоже расположены вместе. Это **отчёт с группировкой**, причем здесь использованы два уровня группировки — сначала по дате, а потом — по номеру заказа.


Заказы
Исполнитель: Василий Петров
Дата: 15.12.12
Дата 11.12.12
Заказ 1
Блюдо | Цена
гупяш | 40 руб.
борщ | 30 руб.
рассольник | 20 руб.
Дата 12.12.12
Заказ 2
Блюдо | Цена
компот | 10 руб.

Рис. 3.50

Перейдём на вкладку **Запросы**, выделим запрос с именем *ЗапросЗаказы* и выберем пункт **Мастер отчетов** из контекстного меню. Добавим в отчёт все поля запроса, а на третьем шаге установим два уровня группировки — сначала по полю *Дата*, затем — по полю *Заказ* (рис. 3.51).

Рис. 3.51

Остальные шаги выполняются так же, как и для простых отчётов, с которыми мы работали раньше.

К сожалению, в настоящей версии OpenOffice.org Base можно строить только самые простые отчёты. Например, невозможно подсчитать общую стоимость каждого заказа или включить в отчёт данные из двух источников (таблиц или запросов), как это мы делали для форм. Нужно отметить, что Microsoft Access обладает значительно бóльшими возможностями для создания профессиональных отчётов.

Вопросы и задания

1. Как установить связи между таблицами?
2. Как вы думаете, по каким признакам программа при установке связи автоматически определяет ее тип (1:1, 1:*N*, *N*:1)?
3. Как строится запрос с выбором данных из нескольких таблиц?

4. Почему иногда в запрос приходится добавлять таблицы, данные из которых не появляются в результатах запроса?
5. Что такое итоговый запрос? Зачем он нужен?
6. Что такое группировка?
7. Что означает группировка по нескольким полям?
8. Объясните, что даёт группировка при построении отчёта.
9. Какие функции можно использовать в итоговых запросах?
10. Как построить форму с подчинённой формой (с субформой)?
11. Как добавить подчинённую форму с помощью Навигатора форм?
12. Как связать новую субформу с главной формой?

Подготовьте сообщение

а) «Работа с базами данных в браузере Firefox»

б) «Работа с базами данных в браузере Chrome»

Задача

Постройте базу данных, запросы, форму и отчёт так, как показано в тексте параграфа.

§ 22
Нереляционные базы данных

Проблемы реляционных БД

Несмотря на то что реляционные (табличные) БД хорошо себя зарекомендовали и составляют подавляющее большинство реально используемых баз данных, они не универсальны и в некоторых задачах их применение приводит к серьёзным проблемам.

Во-первых, для того чтобы построить надёжно работающую реляционную БД, необходимо представить исходные данные как набор взаимосвязанных таблиц. Это требует серьёзных усилий, кроме того, данные становятся менее понятными для человека, потому что он мыслит не таблицами, а объектами, которые имеют определённый набор свойств.

Во-вторых, данные, связанные с одним объектом, разбросаны по разным таблицам, поэтому при поиске их приходится «вытаскивать» с помощью сложных запросов, которые выполняются сравнительно медленно при больших объёмах данных.

В-третьих, во всех реляционных базах данных структура данных чётко определена, причем она задаётся разработчиком при создании базы, и изменить её весьма непросто. Теперь представим себе, что нам нужно хранить документы, которые могут иметь разное количество свойств с разными названиями и типами данных, причём эти свойства заранее не определены и могут меняться со временем. В этом случае использовать реляционные БД, по крайней мере, очень сложно, поскольку они для этой цели не предназначены.

В-четвёртых, объемы данных, которые нужно обрабатывать, всё время возрастают, сейчас базы данных поисковых систем могут достигать нескольких *петабайтов*[1]. Поэтому один компьютер с этим справиться не в силах (система может получать сотни тысяч запросов в секунду). Возникает задача распределить нагрузку на большое количество (иногда десятки тысяч) серверов, связанных между собой через Интернет. Если при этом применять реляционную базу данных, то для выполнения даже простых запросов нужно обращаться ко многим серверам, это недопустимо замедляет поиск. Подобная проблема возникает при «облачных» вычислениях, при которых данные пользователя хранятся на серверах в Интернете. Таким образом, реляционные БД хороши, когда вся база находится на одном компьютере, но они плохо *масштабируются*. Это значит, что при увеличении объема данных и количества запросов не удаётся увеличивать мощность системы, просто добавляя новые сервера. Причина в том, что реляционная модель плохо подходит для распределённых информационных систем.

Базы данных «ключ — значение»

В последние годы появились базы данных нового типа, получившие название «ключ — значение» (англ. *«key-value» database*), которые хорошо показали себя в распределённых системах с большой нагрузкой, в том числе в поисковых системах Интернета.

Базу данных «ключ — значение» можно представить себе как огромную таблицу, в каждой ячейке которой могут храниться произвольные данные («значения»), их структура никак не ограничена. Каждому значению присваивается некоторый код («ключ»), по которому его можно найти. Все данные, относящиеся к конкретном объекту, хранятся в одном месте, поэтому при запросе не нуж-

[1] 1 Пбайт = 2^{10} Тбайт = 2^{50} байт.

Например, *Google* обрабатывает более 20 Пб данных в день, *Facebook* хранит 1,5 Пб фотографий, *Twitter* генерирует 2 Пб данных в год.

но обращаться к разным таблицам, а достаточно просто найти значение по ключу.

СУБД поддерживает только добавление записи, поиск значения по ключу, а также изменение и удаление найденной таким образом записи. Никакие связи между значениями в явном виде не поддерживаются. Хотя объект может содержать ссылки на другие объекты (их ключи), СУБД не проверяет их правильность. Обеспечение надёжности и целостности данных возлагается не на СУБД, а на прикладную программу, которая работает с базой данных.

Ключи — это **хэш-коды** хранящихся данных (значений). Ключи объединяются в группы так, что все данные, связанные с ключами одной группы, хранятся на одном сервере. Таким образом, по ключу можно сразу определить нужный сервер и напрямую получить от него данные. За счет этого обеспечивается масштабируемость — если один сервер не справляется с нагрузкой, нужно добавить ещё один и разделить данную группу ключей на две части.

Многие базы типа «ключ — значение» хранят **документы** — объекты, которые имеют произвольный набор полей-свойств, например:

```
{
  ключ: 123123978623476239476923 7
  автор: "А.С. Пушкин"
  название: "Евгений Онегин"
}
```

Важно, что другие документы могут иметь совершенно другой набор полей. Такие базы данных называют **документоориентированными**.

Итак, базы данных «ключ — значение» обладают **достоинствами**, которые принципиально важны в некоторых задачах:

- *масштабируемость* — возможность наращивания мощности распределенной системы простым добавлением новых серверов;
- *простота* представления данных, близость к человеческому восприятию.

В то же время у них есть и **недостатки**:

- СУБД *не поддерживают связи* между данными, не обеспечивает целостность данных;

- нет стандарта на язык описания и управления данными (для реляционных БД таким стандартом стал язык SQL);
- основной вид запросов — поиск значения по ключу, поэтому *очень сложно*, например, *выполнить сортировку данных*.

В первую очередь, базы данных «ключ — значение» используются при «облачных» вычислениях: в поисковой системе Google (система хранения данных *BigTable*), интернет-магазине *Amazon* (www.amazon.com, база данных *SimpleDB*), социальной сети *Facebook* (www.facebook.com, СУБД *Cassandra*), сервисе микроблогов Twitter (twitter.com, СУБД *Cassandra*).

Кроме того, есть бесплатные СУБД этого класса, например *MongoDB* (www.mongodb.org) или *CouchDB* (couchdb.apache.org).

Вопросы и задания

1. Назовите недостатки реляционных БД. В каких задачах они проявляются?
2. Что такое распределённые базы данных?
3. Что такое масштабируемость? Почему реляционные БД плохо масштабируются?
4. Как вы понимаете выражение «ключ — значение»?
5. Как обеспечиваются связи между объектами в таких БД? Как обеспечивается целостность данных?
6. Вспомните, что такое хэширование и хэш-коды. Как можно выполнить масштабирование на основе хэш-кодов?
7. Что такое документоориентированные базы данных?
8. Назовите достоинства и недостатки баз данных типа «ключ — значение». В каких задачах их имеет смысл использовать?
9. Что вы думаете о дальнейшей судьбе реляционных БД в связи с появлением новых принципов хранения данных? Проведите исследование.

Подготовьте сообщение

«Нереляционные базы данных: за и против»

Задача

*Попробуйте поработать с какой-нибудь документоориентированной СУБД.

§ 23
Экспертные системы

Эксперт — это человек, который обладает глубокими теоретическими *знаниями* и практическим *опытом работы* в некоторой области. Например, врач-эксперт хорошо ставит диагноз и лечит потому, что имеет медицинское образование и большой опыт лечения пациентов. Он не только знает факты, но и понимает их взаимосвязь, может объяснить причины явлений, сделать прогноз, найти решение в конфликтной ситуации.

Эксперт может ответить на вопросы, на которые нельзя найти ответы в поисковых системах Интернета. Более того, эксперт часто может предложить решение **плохо поставленных задач**, например при нехватке, неточности или противоречивости исходных данных. Для этого он использует свой опыт и интуицию, которые сложно представить в виде формального алгоритма. В то же время *нельзя гарантировать* правильность решения, принятого в таких сложных случаях, хотя иногда эксперт может примерно оценить *вероятность* своей версии. Например, можно получить ответ: «С вероятностью 80% вы поступите в вуз».

Экспертная система — это компьютерная программа, задача которой — заменить человека-эксперта при выработке рекомендаций для принятия решений в сложной ситуации.

Разработка экспертных систем — это одно из направлений развития **искусственного интеллекта**, потому что такие программы пытаются моделировать мышление человека. Результат работы экспертной системы — это не числа, а конкретный совет (рекомендация) в словесной форме.

Экспертные системы применяются в медицине, электронике, геологии, для решения военных и управленческих задач. Первая экспертная система Dendral была создана в Стэнфордском университете в конце 1960-х гг. для определения строения органических молекул по их химическим формулам и свойствам.

В составе экспертной системы выделяют три основные части:

- *базу знаний*;
- *блок получения решения* («решатель»);
- *интерфейс с пользователем*.

База знаний отличается от базы данных тем, что в ней хранятся не только факты, но и правила, по которым из фактов делаются выводы.

Факты — это утверждения, которые считаются истинными, например:

- у окуня есть жабры;
- Иван — отец Марьи;
- Волга впадает в Каспийское море.

Правила обычно формулируются в виде «если..., то», например:

- если x — животное и x дышит жабрами, то x — рыба;
- если x — отец y и y — отец z, то x — дед z;
- если x состоит из атомов углерода и обладает высокой твёрдостью, то x — алмаз.

Таким образом, в отличие от данных, знания представляют собой общие связи предметов, понятий и явления; часто их формулировка содержит переменные[1].

В разработке экспертной системы участвуют эксперты и специально обученный специалист — **инженер по знаниям**, задача которого — представить знания экспертов в такой форме, чтобы они могли быть записаны в базу знаний. Важно, чтобы эту базу знаний можно было постепенно пополнять — добавлять новые правила вывода независимо от предыдущих.

Второй блок экспертной системы — «решатель» — это программа, которая моделирует рассуждения эксперта, используя предоставленные ей исходные данные и информацию из базы знаний. В результате работы она не только выдаёт заключение, но и может подробно показать, как оно было получено (какие правила были использованы). Решатель — это универсальная программа, которая может работать с любой базой знаний понятного ей формата.

Как правило, пользователь работает с экспертной системой в режиме диалога, отвечая на её вопросы. Рассмотрим простейшую экспертную систему для определения класса животных. Предположим, что в базу знаний внесены следующие правила:

- если у животного есть перья, то это птица;
- если животное дышит жабрами, то это рыба;

[1] Вспомните, что в логике утверждение с переменными называется *предикатом*.

- если животное кормит детёнышей молоком, то это млекопитающее;
- если животное — млекопитающее и ест мясо, то это хищник.

Диалог пользователя с экспертной системой может быть, например, таким (ответы пользователя выделены курсивом):

– Это животное кормит детей молоком?
– *Нет.*
– Это животное имеет перья?
– *Нет.*
– Это животное дышит жабрами?
– *Да.*
– Это рыба.

Для того чтобы определить последовательность вопросов, эксперт и инженер по знаниям строят дерево решений, например такое (рис. 3.52).

Рис. 3.52

Конечно, эта экспертная система неполна и в некоторых случаях класс животного определить с её помощью не получится (см. знаки вопроса на схеме).

Простую экспертную систему можно написать на любом языке программирования, в котором есть операторы ввода, вывода и ветвления. Однако для профессиональных разработок в этой области чаще всего применяют специальные языки, например **язык логического программирования Пролог**. Программа на Прологе — это набор фактов и правил вывода. Алгоритм решения задачи писать не нужно, решающая система сама находит ответы на вопросы, заданные в определённой форме.

Итак, экспертная система обладает следующими свойствами:

- применяется в определённой достаточно узкой области;
- использует базу знаний, которая может расширяться;
- может применяться при неточных и противоречивых данных;
- выдаёт ответ в виде рекомендации (совета по принятию решения);
- может показать, как получено решение (какие правила применялись).

В последние годы прогресс в области экспертных систем не очень заметен, что связано с их серьёзными недостатками:

- опыт и интуицию экспертов очень сложно формализовать, свести к чётким правилам;
- отладка и проверка экспертных систем очень сложна, таким образом, трудно гарантировать правильность выводов (это особенно важно в военной области);
- экспертные системы не способны самообучаться, для их поддержки необходима постоянная работа инженера по знаниям.

Вопросы и задания

1. Что такое экспертная система? Из каких элементов она состоит?
2. Чем отличается база знаний от базы данных?
3. Перечислите особенности экспертных систем. В каких областях они применяются?
4. Что входит в задачи инженера по знаниям?
5. Почему развитие экспертных систем в последние годы идёт не очень активно?

Подготовьте сообщение

а) «Что такое база знаний?»

б) «Что делает инженер по знаниям?»

в) «Применение экспертных систем»

г) «Язык программирования Пролог»

Задачи

1. Придумайте какую-нибудь задачу, в которой нужно принять решение. Постройте для неё базу знаний в форме «если..., то...» и дерево решений. Это может быть, например, прогноз погоды, отдых на выходных, поездка за рубеж и т. п.

*2. Постройте для вашей задачи экспертную систему на каком-нибудь языке программирования.

Практические работы

Работа № 13 «Работа с готовой таблицей»
Работа № 14 «Создание однотабличной базы данных»
Работа № 15 «Создание запросов»
Работа № 16 «Создание формы»
Работа № 17 «Оформление отчёта»
Работа № 18 «Язык SQL»
Работа № 19 «Построение таблиц в реляционной БД»
Работа № 20 «Создание формы с подчинённой»
Работа № 21 «Создание запроса к реляционной БД»
Работа № 22 «Создание отчёта с группировкой»
Работа № 23 «Нереляционные БД»
Работа № 24 «Простая экспертная система»

ЭОР к главе 3 на сайте ФЦИОР (http://fcior.edu.ru)

- Проектирование объектов данных
- Ввод данных в БД
- Проектирование экранных форм
- Запросы на выборку данных
- Проектирование отчётов

Самое важное в главе 3

- Информационная система (ИС) в широком смысле — это аппаратные и программные средства, предназначенные для того, чтобы своевременно обеспечить пользователей нужной информацией.
- База данных (БД) — это специальным образом организованная совокупность данных о некоторой предметной области, хранящаяся во внешней памяти компьютера.
- Система управления базой данных (СУБД) — это программные средства, которые позволяют выполнять все необходимые операции с базой данных.