Министерство образования и науки Российской Федерации

Саратовский социально-экономический институт (филиал)
РЭУ им. Г.В. Плеханова

Кафедра информационных систем в экономике

Распределенные системы

Учебное пособие

для студентов, обучающихся
по направлению подготовки
38.03.05 Бизнес-информатика

Саратов
2018

УДК 004.75
ББК 32.973.202
Р24

Рекомендует к печати редакционно-издательский совет
ССЭИ РЭУ им. Г.В. Плеханова 30.05.2016 г.

Авторы-составители:
А.В. Демина, О.Н. Алексенцева

Рецензенты:
кандидат социологических наук, доцент *Е.Г. Пчелинцева*,
кандидат технических наук, доцент *И.П. Волошин*

Р24 **Распределенные системы :** учебное пособие для студентов, обучающихся по направлению подготовки 38.03.05 Бизнес-информатика / [авт.-сост. А.В. Демина, О.Н. Алексенцева]. – Саратов : Саратовский социально-экономический институт (филиал) РЭУ им. Г.В. Плеханова, 2018. – 108 с.

В пособии рассматриваются основные понятия распределенных систем, их место в современной экономике, основные составляющие распределенных баз данных и методы их организации. Уделено внимание проектированию, ведению базы данных и создания запросов на языке структурированных запросов SQL.

Учебное пособие предназначено для студентов, обучающихся по направлению подготовки 38.03.05 Бизнес-информатика. Данное издание может быть полезно для магистрантов и специалистов.

УДК 004.75
ББК 32.973.202

Оглавление

Введение

Распределенные базы данных широко используются в различных информационных системах. За последние годы в *ddb*-технологии не было каких-либо значительных изменений, исключение составили технологии тиражирования данных. В области информатики наблюдается стабильность и каких-либо глобальных изменений не предвидится. Наиболее интересным направлением (включая *ddb*) является архитектура, проектирование и реализация распределенных информационных систем. Актуальными темами в направлении *ddb*-технологий являются системы с трехзвенной архитектурой, продукты класса *middleware* и объектно-ориентированные средства разработки распределенных приложений в стандарте *CORBA*. В ближайшие 3–5 лет *ddb*-технологии будут активно применяться и доминировать в отечественной информатике. *Ddb*-технологии должны стать технологической базой реальных интеграционных проектов. Изменения коснутся и архитектуры корпоративных информационных систем. На организации инфраструктуры корпораций большое влияние оказывают развитие интернета, технологии *Java*. Гибкость, открытость, простота развития и расширения гипертекстовой организации данных в отличие от жестких структур реляционных баз данных, плохо приспособленных для расширения, определяют использование *html* в качестве одного из основных средств разработки информационного пространства фирмы. Гипертекстовый подход позволяет интегрировать уже существующие информационные массивы, хранящиеся в базах данных, без особых проблем. Прообразом будущей корпоративной информационной системы является *Intranet*.

Глава 1

Введение в распределенные системы

Понятие распределенной системы

Сейчас используются различные понятия и определения распределенной системы (РС). Обобщая, их можно свести к следующим определениям:

1) распределенной системой называется набор независимых компьютеров, представляющийся их пользователям единой системой;

2) распределенной информационной системой (РИС) называется совокупность взаимодействующих друг с другом программных компонент. Каждая из них может рассматриваться как программный модуль (приложение), исполняемый в рамках отдельного *процесса*.

Пользователи и приложения единообразно работают в распределенных системах независимо от того, когда и где происходит это взаимодействие.

При такой работе информационные системы должны иметь возможность скрывать от пользователей различия между компьютерами и способы связи между ними. Еще одной важной характеристикой распределенных систем является способность обеспечения единообразной работы пользователей и приложений в распределенных системах.

Распределенные системы должны сравнительно легко поддаваться расширению и масштабированию. Выход из строя одной из частей распределенной системы не должен приводить к сбою всей распределенной системы, и пользователей не должны об этом уведомлять.

Для поддержания представления разных компьютеров и сетей в виде единой системы организация распределенной системы нередко включает в себя дополнительный уровень программного обеспечения, находящийся между прикладным уровнем и операци-

онной системой. Данная распределенная система называется системой *промежуточного уровня (middleware).*

Основными задачами распределенной обработки данных является облегчение доступа к удаленным ресурсам и контроль совместного использования компьютеров, файлов, данных в БД, веб-страницы и сети.

Для распределенной обработки данных распределенная система должна удовлетворять следующим требованиям: прозрачности, открытости, гибкости и масштабируемости, т. е. иметь возможность для расширения.

В распределенных системах процессы и ресурсы физически распределены по разным компьютерам, но пользователи этого не видят. Прозрачными называются распределенные системы, которые представляются пользователям и приложениям в виде единой системы.

Требование прозрачности применимо к разным сторонам распределенных систем и включает в себя прозрачность: доступа, местоположения, переноса, смены местоположения, репликации, параллельного доступа, отказов.

Прозрачность доступа заключается в скрытии разницы в способах представления и передачи данных компьютерами и операционной системой.

Прозрачность местоположения заключается в скрытии реального физического размещения ресурса. Важную роль в реализации этого отводится *именованию ресурса*.

Прозрачность переноса заключается в скрытии факта физического перемещения ресурса. При этом на доступ не влияет смена его местоположения.

Отличие прозрачности смены местоположения от *прозрачности переноса* выражается в том, что изменение местоположения ресурса может произойти при его использовании, например использование мобильными пользователями.

Прозрачность репликации (дублирования) заключается в скрытии факта наличия нескольких копий ресурса.

Прозрачность параллельного доступа заключается в скрытии от пользователя факта совместного использования ресурса. При этом, например, в механизме блокировок в БД, обеспечивается целостность и непротиворечивость ресурса.

Прозрачность отказов заключается в обеспечении нормальной работы при наличии отказов или скрытии факта отказа без уве-

домления об этом пользователя. Например, при перегрузке веб-сервера браузер выжидает необходимое время, а затем сообщает о недоступности страницы.

Не все эти требования могут быть полностью реализованы в распределенных системах, т. к. обеспечение прозрачности влияет на производительность.

Открытая распределенная система предлагает службы, вызов которых требует использование стандартных синтаксиса и семантики. Например, соответствие в сетях формата сообщений формату протоколов.

Интерфейсы в распределенных системах определяют службы и компоненты, которые, как правило, описываются языком определения интерфейсов (*Interface Definition Language, IDL*). Интерфейс описывается с помощью синтаксиса служб – имен доступных функций, типов параметров, возвращаемых значений и др. Семантику служб описать сложнее, т. к. эти описания задаются средствами естественного языка.

Открытые распределенные системы должны обладать свойствами интероперабельности, т. е. способности к взаимодействию, возможности переноса из одной системы в другую без изменения интерфейса. Система должна обладать гибкостью и иметь возможность легкого конфигурирования. Таким образом, открытые распределенные системы расширяются аппаратно и программно.

При построении открытых распределенных систем главным фактором является организация системы в виде наборов небольших, легко заменяемых и адаптируемых компонентов. Это позволяет определять интерфейсы как верхнего уровня, с которыми работают пользователи и приложения, так и интерфейсы между компонентами системы.

Масштабируемость (возможность расширения) системы может измеряться по трем различным показателям: *размер*, определяющий легкость подключения к системе новых ресурсов и пользователей; *площадь системы* определяет ресурсы и пользователей, которые могут быть разнесены в пространстве; *управление* системой должно быть простым при работе в независимых организациях.

Масштабируемость распределенных систем влияет на производительность и может ее снижать.

Практическая реализация масштабируемости, как правило, рассматривается наряду с такими требованиями, как производительность и безопасность.

Примерами ограничения масштабируемости могут являться: один сервер на нескольких пользователей; телефонный справочник, доступный онлайн; организация маршрутизации данных, передаваемых по сети. Большие объемы данных передаются по множеству каналов, при этом используются децентрализованные алгоритмы.

К основным технологиям масштабирования относятся:

1. Скрытие времени ожидания связи, которое применяется при географическом масштабировании. Основной идеей является попытка избежать ожидания ответа на запрос от удаленного сервера, что означает использование приложения-клиента с асинхронной связью. Приложение-клиент при получении ответа прерывает свою работу и вызывает обработчик для завершения обработки отправленного запроса с сервера. В системах пакетной обработки данных и параллельных приложениях используется *асинхронная* связь, в которой во время ожидания выполнения задачи сервера и завершения с ним связи предполагается выполнение других независимых задач.

В большинстве задач использование асинхронной связи с приложениями не всегда эффективно. Примерами могут служить задачи интерактивной обработки данных с использованием форм. При этом для сокращения объема передаваемых и обрабатываемых данных, которые обычно выполняются на стороне сервера, их перемещают на сторону клиента.

Чаще всего это касается проверки данных в формах. Более эффективно перемещать проверку данных всех полей формы на сторону клиента.

2. При *распределении* передаваемые данные разбиваются на более мелкие пакеты с последующим их распределением в системе. Например, в системе доменных имен интернета (*DNS*) пространство *DNS* организуется иерархически в виде дерева доменов (*domains*), которые разбиваются на зоны (по странам и областям деятельности). Отдельным сервером обрабатываются имена каждой зоны.

Служба доменных имен распределяется по нескольким серверам, это позволяет разгрузить сервер и исключить обработку всех запросов одним сервером.

Другим примером может служить веб-страница. Каждая веб-страница имеет уникальный *URL*-адрес. Физически среда веб разнесена по многим серверам. *URL*-адрес определяет имя сервера,

на котором хранится конкретный документ, что позволяет увеличивать количество документов не снижая производительности.

3. Репликация (дублирование) применяется для повышения доступности ресурсов и помогает уравнивать нагрузку компонентов, что улучшает производительность, помогая сократить время ожидания отклика данных.

Особой формой репликации является кеширование. При кешировании как и при репликации создаются копии ресурса, как правило в непосредственной близости от клиента. Кеширование предпринимается на стороне клиента, а репликация – на стороне сервера.

При репликации и кешировании создаются копии ресурса, ресурсы модифицируются и эти копии становятся различными. Изменения должны немедленно распространяться во все копии – в этом состоит основная проблема строгой непротиворечивости ресурса. При одновременном изменении двух или более копий порядок внесения изменений во все копии должен быть строго соблюден. Реализовать на практике это довольно сложно, а иногда – невозможно. Репликация может включать и отдельные не масштабируемые решения.

Концепции аппаратных решений

Существует несколько вариантов соединения и организации взаимного обмена процессоров в единую распределенную систему.

Распределенные системы можно разделить на *две* группы: мультипроцессорные, мультикомпьютерные. Системы, в которых компьютеры используют память совместно, называются *мультипроцессорными.* Если в распределенной системе каждый компьютер работает со своей памятью, то такие системы называются *мультикомпьютерными*. Мультипроцессорные системы имеют единое адресное пространство, которое используется всеми процессорами. В мультикомпьютерных системах каждая машина использует свою память, например – обычная сеть компьютеров. Топология распределенных сетей может быть различной: компьютеры могут соединяться по шинной, кольцевой или коммутируемой топологии.

Мультикомпьютерные системы подразделяются на гомогенные и гетерогенные. В гомогенных системах используется одна компьютерная сеть, построенная на единой топологии с использованием однотипных процессоров. Такие системы могут быть как параллельные, так и мультипроцессорные. Примером являются кластеры рабочих станций.

Гетерогенные системы могут содержать независимые компьютеры, соединенные разными сетями. Например, система может состоять из нескольких локальных сетей с коммутируемой магистралью *FDDI* или *ATM*.

Концепции программных решений

Наибольшее влияние на аппаратную часть распределенных систем оказывают программные решения. В первую очередь они влияют на удобство работы пользователя в распределенных системах. Распределенные системы помогают пользователям совместно использовать общие ресурсы, такие как: память, данные, процессоры, периферийное оборудование и сеть. Распределенные системы выполняют функции *менеджеров ресурсов*, т. е. их работа аналогична работе операционной системы. С другой стороны, распределенные системы скрывают от пользователя гетерогенность и сложность аппаратуры, предоставляя удобный интерфейс работы. Функции распределенной системы можно сравнить с функциями операционной системы.

Операционные системы в распределенных системах делятся на две категории: *сильно связанные* и *слабо связанные*. Сильно связанными операционными системами называются *распределенные* операционные системы, которые используются для управления мультипроцессорными, мультикомпьютерными и гомогенными системами. Их основная их цель – скрывать подробности управления аппаратным обеспечением от пользователя.

Слабо связанными операционными системами называются сетевые операционные системы, которые используются для управления гетерогенными мультикомпьютерными комплексами. Наряду с традиционными функциями управления ресурсами они обеспечивают доступ удаленных клиентов к локальному программному, аппаратному обеспечению и данным.

При создании распределенной системы недостаточно служб сетевой операционной системы. К ним необходимо добавить дополнительные элементы для организации прозрачной структуры системы. Данные элементы образуют *промежуточный уровень (middleware)* системы. Таким образом, программные средства играют основную роль в построении распределенной системы промежуточного уровня, между распределенными приложениями и операционными системами. Общая структура распределенной системы с промежуточным уровнем показана на рис. 1.

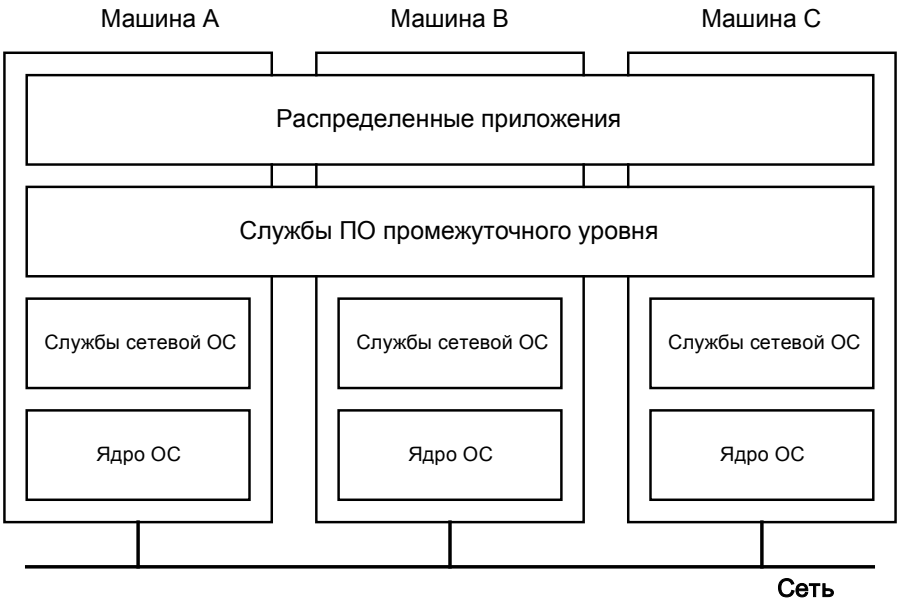


Рис. 1. Общая структура распределенной системы с промежуточным уровнем

Модели промежуточного уровня

При разработке распределенной системы используются две модели: *распределенная файловая система*, когда все объекты представляются в виде файлов, и система, основанная на *удаленных вызовах процедур.* Примером файловой системы является файловая система *Unix.* Модель *удаленных вызовов процедур* основана на скрытии сетевого обмена, при этом процессы могут вызывать процедуры, находящиеся на удаленной машине. При вызове процедуры и ее выполнении параметры передаются на удаленную машину, далее управление передается компьютеру-клиенту. Для пользователя это выглядит как обычный вызов процедуры.

Современные модели распределенных систем основаны на взаимодействии *распределенных объектов*. Примерами являются: *DCOM* и *CORBA*. В распределенных объектах каждый объект реализует свой интерфейс, который скрывает все внутренние детали выполнения функций системы от пользователя. Интерфейс основан на методах, реализуемых системой. Все, что видит процесс – это интерфейсы.

В системе *WWW* используется модель *распределенных документов*. В данной модели информация организуется в виде документов, размещенных, как правило, на удаленном сервере или на распределенных компьютерах-клиентах, при этом пользователь не видит, где размещен документ. Веб-страницы могут содержать гиперссылки на другие документы и страницы, видео, графику, звуковые файлы. При обращении к документу из адресной строки браузера формируется запрос серверу, на сервере выполняется код

генерации страницы, веб-страница формируется «налету» и в окне браузера мы видим виртуальную страницу.

Для низкоуровневой пересылки сообщений по сети используются сервисы промежуточного уровня, которые предназначены для поддержки прозрачности доступа путем предоставления высокоуровневых средств связи. Интерфейс транспортного уровня (*IP*) заменяется средствами прозрачного доступа к распределенным БД, файловым системам и веб-документам.

Общей службой для всех систем промежуточного уровня является *именование (naming).* Любой документ идентифицируется с помощью *URL*-адреса, содержащего имя хостинга, на котором находится документ с данным *URL*.

Служба *обеспечения сохранности данных* реализуется механизмом *распределенных транзакций*.

Служба *обеспечения защиты программ и данных* наряду с требованием масштабируемости является одной из наиболее трудно реализуемых в распределенной системе.

Модель «клиент – сервер»

В данной базовой модели распределенной системы все процессы обработки данных делятся на две взаимно перекрывающиеся группы. Серверами называются любые процессы, реализующие, например, обработку удаленной файловой системы или распределенной базы данных. Сервер предоставляет свои ресурсы и хранимые данные удаленным рабочим станциям и одновременно может использовать их ресурсы и данные. *Клиентами* называются процессы, посылающие запрос серверу на обработку данных с последующим ожиданием ответа от сервера. Клиент получает услуги сервера.

В рамках модели «клиент – сервер» взаимодействие может быть *синхронным* и *асинхронным.* При синхронном взаимодействии клиент ожидает завершения обработки своего запроса сервером, а при *асинхронном* взаимодействии клиент посылает серверу запрос и, не дожидаясь ответа сервера, продолжает свою работу. Модель «клиент – сервер» является основой описания различных взаимодействий удаленных и распределенных систем (рис. 2).

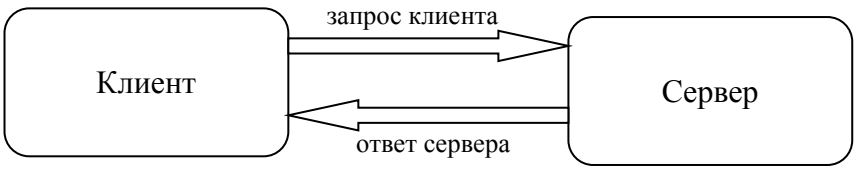


Рис. 2. Модель взаимодействия «клиент – сервер»

Между клиентом и сервером может не быть четкого разделения функций системы. Например, сервер распределенной базы данных, отвечающий за хранение таблиц этой базы данных, может быть одновременно клиентом и передавать запросы на файловые серверы.

Рассмотрим логические уровни взаимодействия пользователя с базой данных (рис. 3).

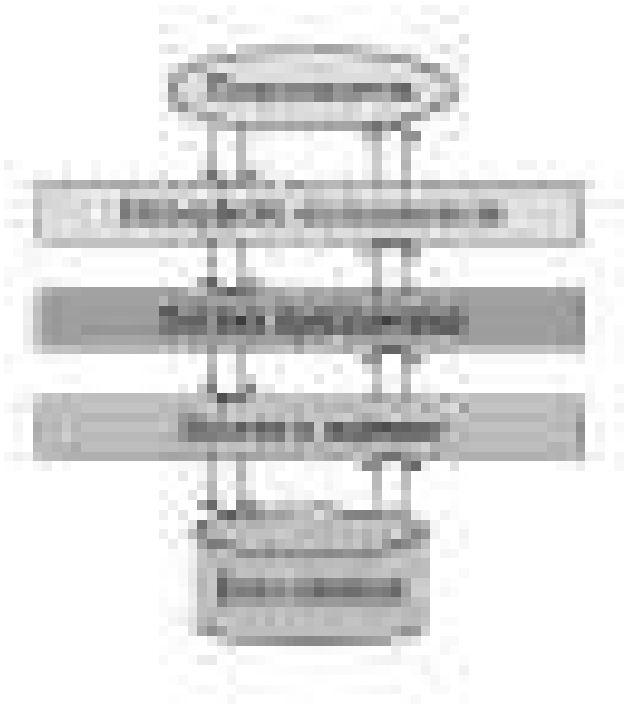


Рис. 3. Логические уровни приложения

На уровне клиента реализуется интерфейс. На уровне функциональности реализуется бизнес-логика приложения. На уровне данных содержатся программы, их обрабатывающие. Этот уровень отличается требованием целостности данных. Когда приложение не работает, данные сохраняются в файле или базе данных для последующего их использования. Данный уровень, как правило, реализуется на сервере.

Кроме сохранности на уровне данных обеспечивается поддержка целостности данных для различных приложений. Поддержание целостности данных означает, что на этом же уровне хранятся схемы базы данных, триггеры базы данных и хранимые процедуры.

Как правило, уровень данных реализуется в виде реляционной базы данных, что позволяет данным не зависеть от приложений. Реляционные базы данных в модели «клиент – сервер» позволяют отделить уровень обработки от уровня данных, при этом данные и их обработка рассматриваются независимо друг от друга.

Типы архитектур «клиент – сервер»

При разделении системы на три логических уровня возникает проблема физического распределения приложений по компьютерам-клиентам в модели «клиент – сервер». Самая простой организацией является использование компьютера-клиента и сервера.

1. На рабочих станциях (клиентах) устанавливаются программы-клиенты, которые реализуют интерфейс.

2. На серверах реализуются все остальные уровни обработки и уровни данных.

В такой организации системы существует проблема, состоящая в том, что все операции происходят на сервере, при этом клиент используется как простой терминал. В этом случае система не будет являться распределенной.

На практике пользователям системы требуется доступ к одним и тем же данным. Разделить функции системы можно между несколькими компьютерами, а логические уровни приложений между одной серверной частью приложения. Серверная часть приложения отвечает за доступ к данным и может находиться на нескольких компьютерах с клиентскими частями, реализующими интерфейс пользователя. Такое разделение будет наиболее простым. Таким образом, логика приложения может быть отнесена как к серверу, так и клиентам либо разделена между ними.

Программные решения могут реализовываться в виде минимальных функций интерфейса пользователя на клиенте (тонкий клиент) или передавать клиенту всю работу с пользовательским интерфейсом (толстый клиент). В этих случаях графический интерфейс с помощью протокола приложения отделяется от приложений и связывается с остальной частью приложения, находящейся на сервере. Внешний интерфейс позволяет реализовать функцию предоставления интерфейса приложения.

Архитектура, построенная по описанному принципу, называется клиент-серверной или двухзвенной (рис. 4). Аналогичные системы зачастую не относят к классу распределенных, но формально они считаются простейшими представителями *распределенных систем*.

Следующим видом архитектуры «клиент – сервер» является *трехзвенная архитектура*. В данной системе интерфейс пользователя, логика приложений и доступ к данным вынесены в самостоятельные подсистемы, работающие на независимых компьютерах (рис. 5).

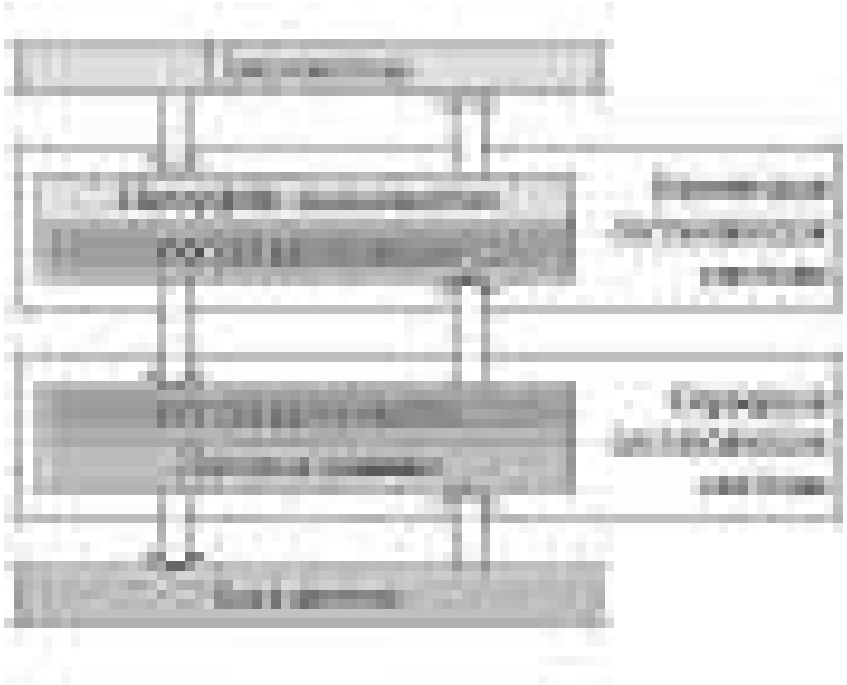


Рис. 4. Двухзвенная архитектура

При такой организации архитектуры программы уровня обработки хранятся на отдельном сервере (*сервере приложений*) или могут частично находиться на машинах клиентов и серверов. Типичным примером является обработка транзакций, при которой каждый отдельный процесс координирует все транзакции.

В трехзвенной архитектуре запрос пользователя обрабатывается последовательно клиентской логикой приложений, сервером логики приложения и сервером баз данных. Как правило, распределенная система – это система с более сложной архитектурой, чем трехзвенная.

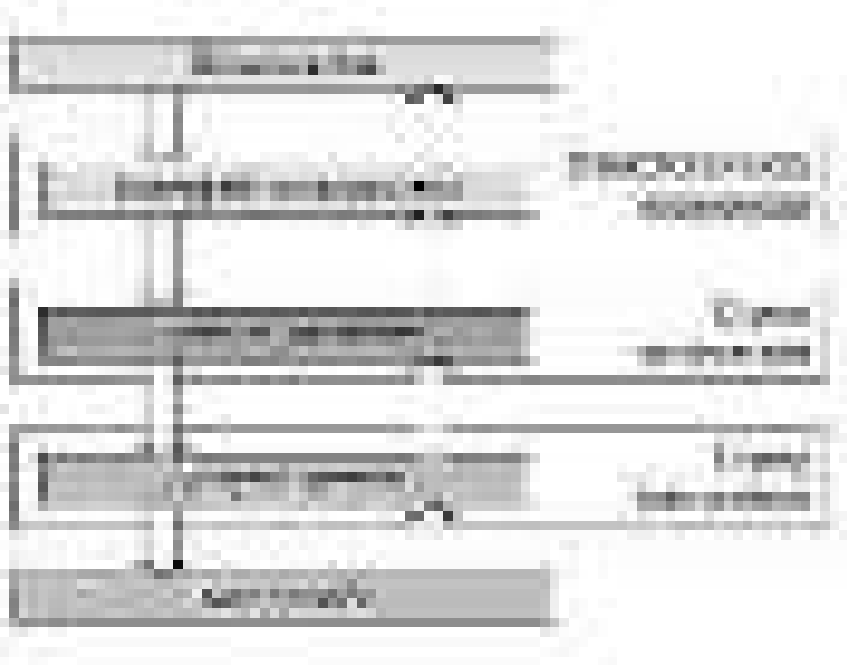


Рис. 5. Трехзвенная архитектура

Наряду с двухзвенной и трехзвенной архитектурами «клиент – сервер» используются многозвенные архитектуры, в которых усиливается разделение приложений на уровни интерфейса пользователя, компонентов обработки и данных. Разные уровни системы взаимодействуют между собой в соответствии с логической структурой организации приложений. В большинстве бизнес-приложений распределенная обработка данных схожа с организацией многозвенной архитектуры приложений клиент-сервер. Данный тип организации клиент-серверной архитектуры называется *вертикальным* распределением. Его особенностью является размещение логически разных приложений на разных машинах-клиентах.

Приложения для автоматизации деятельности предприятий реализуются в виде распределенных систем, при этом логика приложений распределяется между несколькими элементами системы и может выполняться на отдельном компьютере.

В случае, когда запросы пользователя не выполняются последовательно от интерфейса пользователя до единственного сервера баз данных, многозвенная архитектура разрастается в ширину. Такое распределение на клиенты и серверы называют горизонтальным распределением. Клиент или сервер могут содержать физически разделенные части логического модуля и независимо работать с каждой частью (рис. 6).

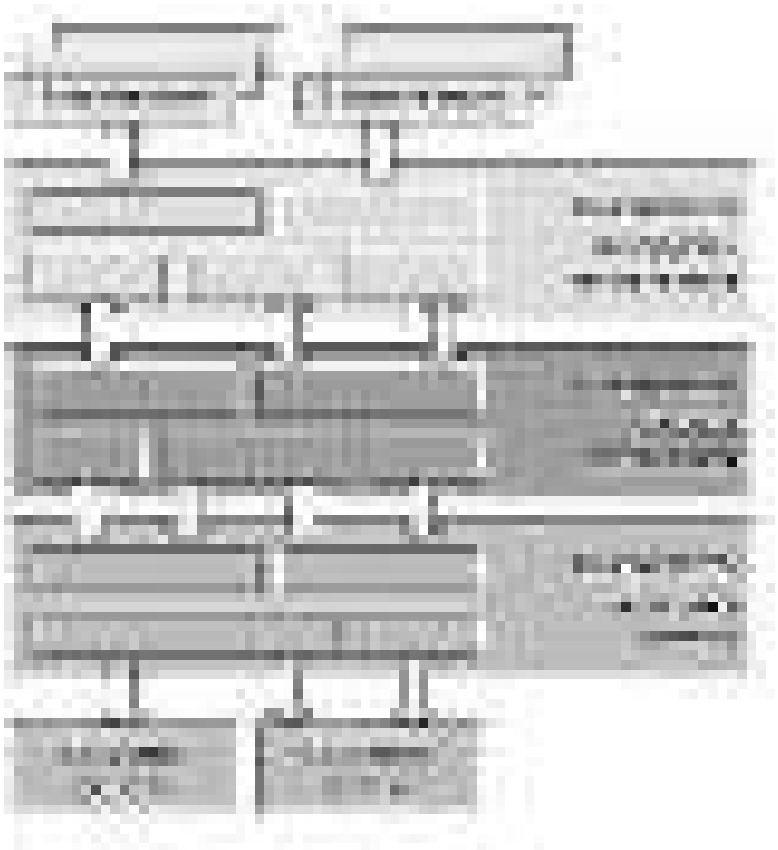


Рис. 6. Структура распределенной системы

Например, веб-сервер может быть реплицирован в локальной сети на несколько машин, тогда при изменении одной веб-страницы изменения будут рассылаться по всем серверам. Передача приходящего запроса серверу осуществляется по кольцу – от сервера к серверу, что позволяет выравнивать нагрузку серверов популярных веб-сайтов.

Клиенты распределяются аналогично. Простые приложения могут работать на распределенных клиентах без серверов (одноранговое распределение). Например, при работе двух и более пользователей, оба могут работать в одном и том же приложении одновременно.

Таким образом, распределенная система представляет собой объединенные в единую систему автономные компьютеры, работающие совместно. К их преимуществам относятся:

1) упрощение интеграции различных приложений в единую систему;

2) масштабируемость.

Данные преимущества приводят к усложнению программных модулей, снижению производительности и возможным проблемам с безопасностью.

Выделяют различные типы распределенных систем. Для управления аппаратными средствами взаимосвязанных корпоративных систем используются распределенные операционные системы. К ним относятся мультипроцессорные и гомогенные мультикомпьютерные системы. Данные распределенные системы воспринимаются в виде единой системы, хотя не состоят из автономных компьютеров. Сетевые операционные системы объединяют удаленные компьютеры, работающие под управлением своих операционных систем, при этом пользователи получают доступ к локальным службам каждой из рабочих станций.

Кроме сетевой операционной системы современные распределенные системы поверх сетевой операционной системы содержат промежуточный слой программного обеспечения, скрывающий гетерогенность и распределение рабочих станций. Модели распределенных систем организованы на удаленном вызове процедур и на распределенных объектах, файлах и документах.

Любая распределенная система основывается на модели внутренней организации. К наиболее распространенной модели относится модель «клиент – сервер», которая тесно связана с традиционным программированием. Данная модель состоит из отдельных

модулей, каждый их которых реализуется отдельной процедурой. Модули подразделяются три уровня: на уровень пользовательского интерфейса, уровень обработки и уровень данных. Как правило, сервер отвечает за уровень данных, клиент отвечает за уровень пользовательского интерфейса. Уровень обработки реализуется на клиенте, или на сервере, или может быть разделен между ними.

При построении крупных систем используется не только вертикальная, но и горизонтальная организация приложений «клиент – сервер». При горизонтальной организации клиенты и серверы физически распределяются и реплицируются на несколько компьютеров. Ярким примером применения горизонтального распределения является веб.

Контрольные вопросы

1. Дайте определение распределенной системе.
2. Перечислите основные задачи распределенных систем.
3. Назовите основные задачи распределенной обработки данных.
4. Какие существуют концепции аппаратных решений?
5. Перечислите концепции программных решений.
6. Что представляет собой модель «клиент – сервер»?

Глава 2

Модели взаимодействия компонентов РС

Понятие промежуточной среды

В распределенной системе один компьютер является сервером, все остальные машины являются удаленными рабочими станциями. Основой сетевого взаимодействия распределенных систем является модель взаимодействия открытых систем *OSI/ISO*. Данная модель разделяет процесс взаимодействия клиента и сервера на семь уровней: физический, канальный, сетевой, транспортный, сеансовый, прикладной, представительский.

Алгоритм взаимодействий в открытых системах описывается стандартными протоколами. Основным стеком протоколов является протокол *TCP/IP*. Протоколом транспортного уровня является протокол *TCP*, а протоколом сетевого уровня является протокол *IP*. Операционная система является связующим звеном с протоколом транспортного уровня и предоставляет интерфейс для верхних уровней основанный на сокетах. Сокеты обеспечивают элементарные операции низкого уровня для непосредственного обмена потоком бит между двумя процессами. В стеке протоколов *TCP/IP* сеансового или стандартного представительского уровня нет. К ним относят иногда защищенные протоколы *SSL/TLS*.

Протоколы можно разделить на 2 основных типа: с установлением соединения и без установления соединения. При установлении соединения для передачи данных отправитель и получатель устанавливают соединение, а после завершения передачи разрывают его. Примером соединения двух абонентов является телефон. Без установки соединения отправитель сразу посылает сообщение адресату.

Протокол *TCP/IP* на основе сокетов является стандартным, межплатформенным, но низкоуровневым сервисом для обмена данными между компонентами. Функции сеансового и представительского уровня берет на себя промежуточная среда, которая называется промежуточным программным обеспечением (рис. 7).

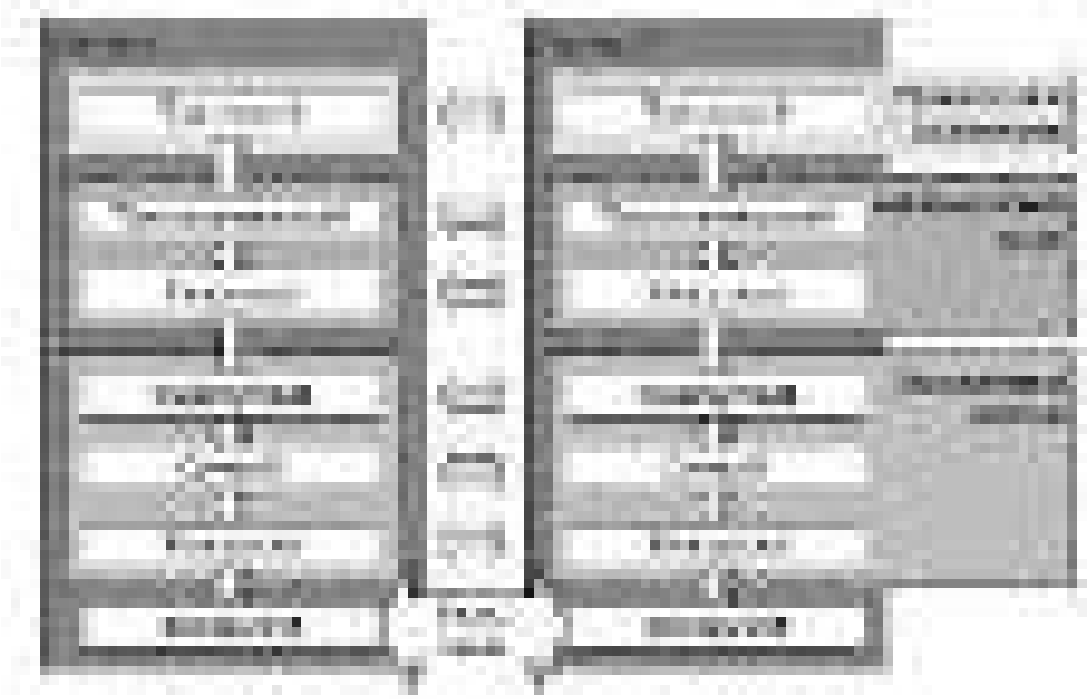


Рис. 7. Модель взаимодействия вычислительных систем

Промежуточная среда помогает разработчикам создавать открытые, масштабируемые и устойчивые распределенные системы. При этом промежуточная среда должна обеспечивать службы для взаимодействия компонент распределенной системы. К этим службам относятся:

– служба обеспечения единого и независимого от операционной системы механизма использования одними программными компонентами служб других компонент;

– служба обеспечения безопасности распределенной системы, в которую входят: аутентификация и авторизация всех пользователей, защита информации от искажения при передаче между компонентами и чтения посторонними лицами;

– служба обеспечения целостности данных, включающая управление транзакциями, распределенными между удаленными компонентами системами;

– служба распределения нагрузки на серверы с программными модулями;

– служба обнаружения удаленных модулей.

В разрезе одной распределенной системы могут использоваться несколько типов промежуточных сред. При проектировании распределенной системы каждая ее компонента предоставляет свои сервисы средствами единственной промежуточной среды и использует службы других компонент посредством этой же или другой промежуточной среды – эти среды могут быть различными.

При выделении промежуточного уровня изменяется базовая модель *OSI*. Промежуточный уровень включает в себя сеансовый уровень и уровень представления и содержит протоколы независящие от приложений.

Взаимодействие в модели *OSI* подразделяется на 7 уровней. На каждом уровне работает протокол, отвечающий за один аспект взаимодействия и предоставляющий интерфейс для работы с вышестоящим уровнем. Набор операций на каждом уровне определяет этот интерфейс. На прикладном уровне осуществляется передача сообщений, затем сообщение передается на следующий уровень и так далее до физического. На каждом уровне происходит инкапсуляция сообщения в пакет следующего уровня, при этом в заголовок добавляется служебная информация или метка завершения для разделения блоков данных на физическом уровне. На физическом уровне происходит реальная передача данных битовым потоком. На компьютере-приемнике выполняется обратное преобразование.

Стеком протоколов называется набор протоколов в модели *OSI*.

Существует различие между эталонной моделью *OSI* и реальными протоколами. Базовые протоколы модели *OSI* используются достаточно редко в отличие от интернет-протоколов. Модель *OSI* удобна для рассмотрения правил взаимодействия систем на уровнях и изучения реальных протоколов.

Реализация основных функций компьютерной сети осуществляется на физическом, канальном и сетевом уровнях.

За передачу сигналов отвечает ***физический уровень***. Протоколы данного уровня обеспечивают стандартизацию механических, электрических и сигнальных интерфейсов. На физическом уровне существует много стандартов для различных носителей и линий связи. Примерами являются: *Ethernet, RS-232, FDDI, Fast Ethernet*.

Основной задачей ***канального уровня*** является поиск и исправление ошибок с физического уровня. Канальный уровень осуществляет упаковку битового потока в дейтаграммы (кадры) и следит за их правильной передачей и приемом. В дейтаграмму для маркировки в начало и конец помещается специальная битовая маска и вычисляется контрольная сумма. Для восстановления правильной последовательности кадров при получении они нумеруются.

Основной задачей ***сетевого уровня*** является выбор оптимального пути (маршрутизация). Глобальные сети содержат множество машин, каждая из них соединена линиями связи с другими машинами. Отправленное сообщение проходит множество сетей, содержащих

маршрутизаторы, которые направляют сообщение по выбранному пути. На сегодняшний день распространенным сетевым протоколом является протокол соединения – *IP*-протокол (*Internet Protocol*).

Передаваемые сообщения на сетевом уровне называются пакетом. IP-протокол является протоколом, не требующим соединения.

Протоколом с соединением является виртуальный канал *ATM* на базе сетей *ATM*. Виртуальный канал в *ATM* представляет собой непрямое соединение, которое устанавливается от источника к приемнику и может проходить через несколько промежуточных *ATM*-коммутаторов. Набор виртуальных каналов образует виртуальный путь аналогично определению маршрута между двумя узлами.

На ***транспортном уровне*** работает протокол *TCP*, он отвечает за передачу и получение пакетов без потерь данных и в правильном порядке. Транспортный протокол *TCP* называется протоколом управления передачей данных (*Transmission Control Protocol*) и относится к базовому стеку сетевых протоколов, так как в нем реализованы все службы необходимые для построения сетевых приложений.

Стек протоколов *TCP/IP* является стандартом сетевых взаимодействий. Стеки протоколов интернета содержат в себе транспортный протокол *UDP* (*Universal Datagram Protocol*, не требующий соединения, он является универсальным протоколом датаграмм и является модификацией протокола IP.

Протоколы верхнего уровня образуют модель *OSI* и располагаются поверх протоколов транспортного уровня. К протоколам верхнего уровня относятся сеансовый, представительский и прикладной уровни.

Сеансовый уровень предоставляет средства синхронизации обмена и является расширением транспортного уровня, но в стек интернет-протоколов не входит.

Уровень представления. Отвечает за правильную интерпретацию сообщений (семантику). Это упрощает взаимодействие между машинами с разным внутренним представлением данных.

В модели *OSI* на прикладном уровне собраны все приложения и протоколы, передачи файлов, эмуляции терминала и др.

Модели взаимодействия компонентов распределенной системы

Обеспечение обмена данными между компонентами распределенной системы осуществляется сервисом промежуточной среды.

В настоящий момент существуют две *концепции организации программных компонент* распределенной среды:

1) обмен сообщениями между компонентами;

2) вызов процедур или методов объекта удаленной компоненты по аналогии с локальным вызовом процедуры.

Любое взаимодействие между удаленными компонентами распределенной системы основано на протоколе *TCP/IP*. Низкоуровневый обмен сообщениями на основе сетевых протоколов является первичным. С точки зрения промежуточной среды данный сервис не определяет формат передаваемого сообщения. Прикладные протоколы обмена данными более высокого уровня строятся на базе протоколов *TCP* или *HTTP*. На их базе строятся прикладные протоколы обмена сообщений для реализации более сложного обмена сообщениями или удаленного вызова процедур.

Удаленный вызов является моделью, происходящей от языков программирования высокого уровня, а не от реализации интерфейса транспортного уровня сетевых протоколов. Поэтому протоколы удаленного вызова должны обязательно базироваться на какой-либо системе передачи сообщений, включая как непосредственное использование сокетов *TCP/IP*, так и основанные на нем другие промежуточные среды для обмена сообщениями. Реализация высокоуровневых служб обмена сообщениями, в свою очередь, может использовать удаленный вызов процедур, основанный на более низкоуровневой передаче сообщений, использующей, например, непосредственно сетевые сокеты. Таким образом, одна промежуточная среда может использовать для своего функционирования сервисы другой промежуточной среды аналогично тому, как один протокол транспортного или сетевого уровня может работать поверх другого протокола.

Обмен сообщениями

Явный *обмен сообщениями* между процессами является основой многих РС.

Существует два метода передачи сообщений от одной удаленной системы к другой – *непосредственный обмен сообщениями* и *использование очередей сообщений*. В первом случае передача происходит напрямую, и она возможна только в том случае, если принимающая сторона готова принять сообщение в этот же момент времени.

Во втором случае используется посредник – *менеджер очередей сообщений*. Компонент посылает сообщение в одну из очередей ме-

неджера, после чего он может продолжить свою работу. В дальнейшем получающая сторона извлечет сообщение из очереди менеджера и приступит к его обработке.

Простейшей реализацией непосредственного обмена сообщениями является использование *транспортного уровня сети* через интерфейс сокетов, минуя какое-либо промежуточное программное обеспечение. Однако такой способ взаимодействия обычно не применяется в системах автоматизации предприятия, поскольку в этом случае реализация всех функций промежуточной среды ложится на *разработчиков приложения.* При таком подходе сложно получить расширяемую и надежную распределенную систему, поэтому для разработки прикладных распределенных систем обычно используются *системы очередей сообщений.*

Существует несколько разработок в области промежуточного программного обеспечения, реализующих высокоуровневые сервисы для обмена сообщениями между программными компонентами. К ним относятся, в частности, *Microsoft Message Queuing, IBM MQSeries и Sun Java System Message Queue.* Такие системы дают возможность приложениям использовать следующие базовые примитивы по использованию очередей:

– добавить сообщение в очередь;

– взять первое сообщение из очереди, процесс блокируется до появления в очереди хотя бы одного сообщения;

– проверить очередь на наличие сообщений;

- установить обработчик, вызываемый при появлении сообщений в очереди.

Менеджер очереди сообщений в таких системах может находиться на компьютере, отличном от компьютеров с участвующими в обмене компонентами. В этом случае сообщение первоначально помещается в исходящую очередь на компьютере с посылающей сообщения компонентой, а затем пересылается менеджеру требуемой.

Для создания крупных систем обмена сообщениями может использоваться маршрутизация сообщений, при которой сообщения не передаются напрямую менеджеру, поддерживающему очередь, а проходят через ряд промежуточных менеджеров очередей сообщений (рис. 8).

Использование очередей сообщений ориентировано на *асинхронный* обмен данными.

Основные достоинства таких систем:

– время функционирования сервера может быть не связано со временем работы клиентов;

– независимость промежуточной среды от средства разработки компонент и используемого языка программирования;

– считывать и обрабатывать заявки из очереди могут несколько независимых компонент, что дает возможность достаточно просто создавать устойчивые и масштабируемые системы.

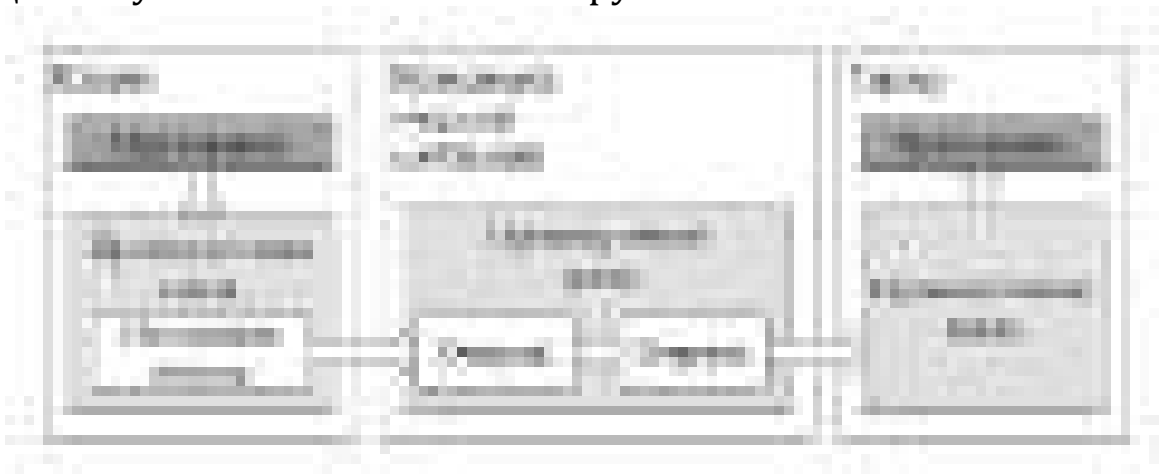


Рис. 8. Системы очередей сообщений

Недостатки систем очередей сообщений являются продолжением их достоинств:

– необходимость явного использования очередей распределенным приложением;

– сложность реализации синхронного обмена;

– определенные накладные расходы на использование менеджеров очередей;

– сложность получение ответа: передача ответа может потребовать отдельной очереди на каждый компонент, посылающий заявки.

Удаленный вызов процедур

Идея удаленного вызова процедур (*remote procedure call, RPC*) появилась в середине 1980-х гг. и заключалась в том, что при помощи промежуточного программного обеспечения функцию на удаленном компьютере можно вызывать, как и функцию на локальном компьютере. Чтобы удаленный вызов происходил прозрачно с точки зрения вызывающего приложения, промежуточная среда должна предоставить процедуру-заглушку (*stub*), которая будет вызываться клиентским приложением. После вызова процедуры-заглушки промежуточная среда преобразует переданные ей аргументы в вид, пригодный для передачи по транспортному протоколу, и передает их на удаленный компьютер с вызываемой функцией. На удаленном компьютере параметры извлекаются промежуточной средой из сообщения транспортного уровня и передаются вызываемой функции

(рис. 9). Аналогичным образом на клиентскую машину передается результат выполнения вызванной функции.

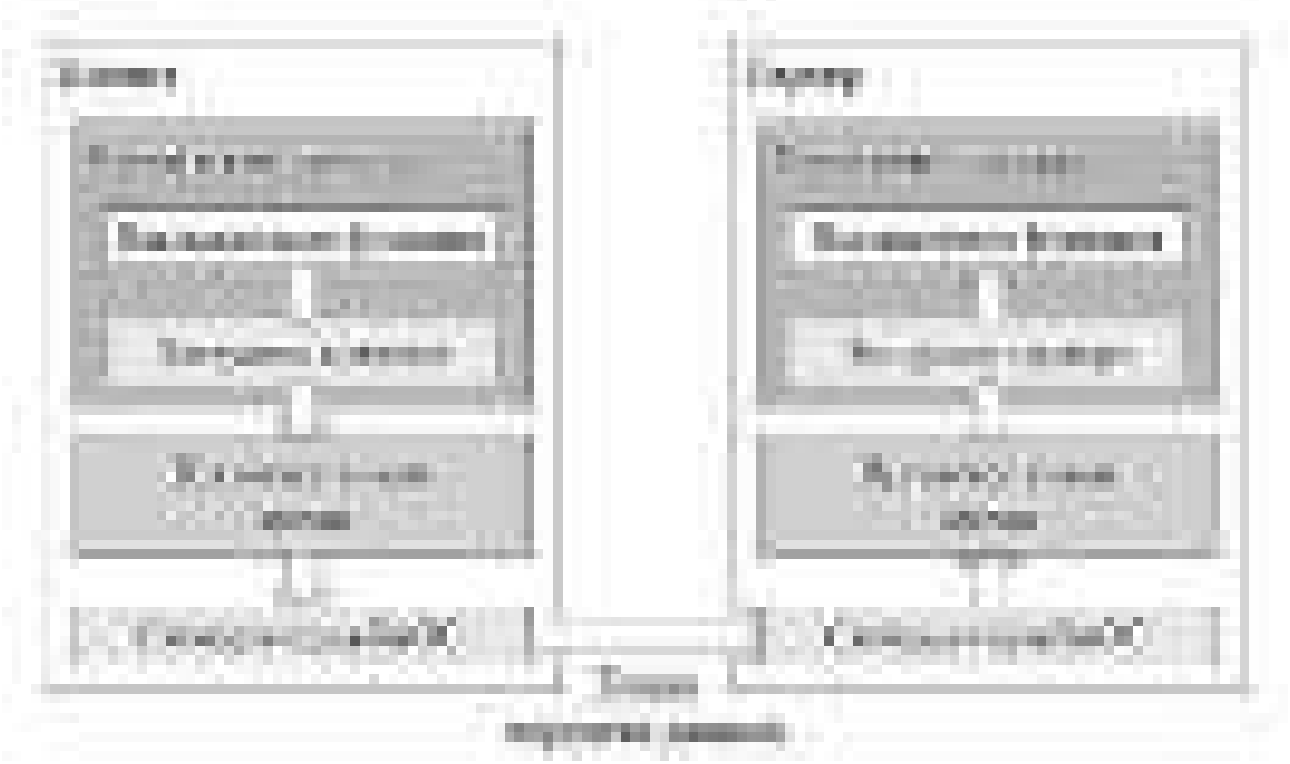


Рис. 9. Удаленный вызов процедур

Например, необходимо из процесса, запущенного на машине А вызвать процедуру на машине В. Информация может быть передана через *параметры* и возвращена в виде *результата*. При этом программист не должен замечать, откуда вызывается процедура (*прозрачность доступа*).

Процедура – это фрагмент программы, оформленный стандартным образом и доступный для использования другими программами с помощью стандартных операций вызова процедур.

Модель, с помощью которой реализуется такое взаимодействие, называется **Удаленный вызов процедур (*Remote Procedure Call, RPC*)**. Это один из ранних методов организации связи, лежащий в основе многих современных технологий распределенных систем.

В этой модели акцент делается на *сокрытии сетевого обмена* за счет того, что процессу разрешается вызывать процедуры, реализация которых находится на удаленной машине. При вызове процедуры параметры передаются на удаленную машину, где она выполняется, после чего управление передается в точку вызова процедуры. Внешне это выглядит как обычный вызов процедуры.

Таким образом, идея вызова удаленных процедур *RPC* состоит в расширении хорошо известного и понятного механизма передачи

управления и данных внутри программы, выполняющейся на одной машине, на передачу управления и данных через сеть. Средства удаленного вызова процедур предназначены для облегчения организации *распределенных вычислений.* Наибольшая эффективность использования *RPC* достигается в тех приложениях, в которых существует интерактивная связь между удаленными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются *RPC-ориентированными.*

Сама идея проста, но не проста в реализации, так как *процесс* и *процедура* находятся на разных машинах в разных *адресных пространствах.* Имеются проблемы и при передаче параметров.

Существует три возможных варианта удаленного вызова процедур.

1. *Синхронный вызов*: клиент ожидает завершения процедуры сервером и при необходимости получает от него результат выполнения удаленной функции.

2. *Однонаправленный асинхронный вызов*: клиент продолжает свое выполнение, получение ответа от сервера либо отсутствует, либо его реализация возложена на разработчика (например, через функцию клиента, удалено вызываемую сервером).

3. *Асинхронный вызов*: клиент продолжает свое выполнение, при завершении сервером выполнения процедуры он получает уведомление и результат ее выполнения, например через callback-функцию, вызываемую промежуточной средой при получении результата от сервера.

Вызов распределенных процедур

Реализация *удаленных вызовов существенно* сложнее реализации вызовов *локальных* процедур. Начнем с того, что поскольку вызывающая и вызываемая процедуры выполняются на *разных машинах,* то они имеют разные *адресные пространства*, и это создает проблемы при передаче параметров и результатов, особенно если машины разного типа. Так как *RPC* не может рассчитывать на разделяемую память, то это означает, что параметры *RPC* не должны содержать указателей на ячейки не стековой памяти и что значения параметров должны копироваться с одного компьютера на другой. Следующим отличием *RPC* от локального вызова является то, что он обязательно использует *нижележащую систему связи*, однако это не должно быть явно видно ни в определении процедур, ни в самих процедурах.

Удаленность вносит дополнительные проблемы. Выполнение вызывающей программы и вызываемой *локальной* процедуры в одной машине реализуется в рамках *единого процесса*. Но в реализации *RPC* участвуют как минимум *два процесса* – по одному в каждой машине.

В случае работы с удаленной процедурой основное отличие состоит в том, что вызов удаленной процедуры обслуживают два процесса: клиентский процесс и серверный процесс.

Процесс клиента отправляет серверу сообщение, в которое включены параметры вызываемой процедуры и ожидает ответного сообщения с результатами ее работы. При получении ответа результат считывается, и процесс продолжает работу.

Со стороны сервера процесс-обработчик вызовов находится в состоянии ожидания, и при поступлении сообщения считывает параметры процедуры, выполняет ее, отправляет ответ и становится в состояние ожидания следующего вызова (рис. 10).

Рис. 10. Сетевое взаимодействие через механизм RPC

Кроме того, существует ряд проблем, связанных с неоднородностью языков программирования и операционных сред: структуры данных и структуры вызова процедур, поддерживаемые в каком-либо одном языке программирования, не поддерживаются точно так же во всех других языках.

Между вызовами *локальных и удаленных процедур* есть еще несколько важных отличий.

1. Обработка ошибок. Клиент в любом случае должен получать уведомление об ошибках, возникающих при вызовах удаленных процедур на сервере или в сети.

2. Глобальные переменные. Поскольку сервер не имеет доступа к адресному пространству клиента, при вызовах удаленных процедур нельзя использовать скрытые параметры в виде глобальных переменных.

3. Производительность. Скорость выполнения удаленных процедур, как правило, на один или два порядка ниже скорости выполнения аналогичных локальных процедур.

4. Аутентификация. Поскольку вызовы удаленных процедур происходят по сети, необходимо использовать механизмы аутентификации клиента.

RPC-протокол не накладывает каких-либо требований на дополнительные связи между процессами и не требует *синхронности* выполняемых функций, т. е. вызовы могут быть асинхронными и взамонезависимыми, так что клиент во время ожидания ответа может выполнять другие процедуры. Сервер *RPC* может выделять для каждой функции отдельный процесс или виртуальную машину, поэтому может принимать следующие, не дожидаясь окончания работы предыдущих запросов.

Механизм работы RPC

Рассмотрим, как происходит *RPC*-вызов. Программист создает клиентскую и серверную часть приложения (в качестве серверной части может выступать системный сервис, входящий в поставку соответствующих операционных систем). Клиентская и серверная части запускаются на различных машинах, соединенных в сеть. Функциональный вызов перехватывает так называемая заглушка (*Stub*-процедура), которая перенаправляет локальный вызов в *RPC*-библиотеку времени-выполнения, в ней происходит преобразование данных под существующий транспортный протокол. Данные и служебная информация переносятся по сети, декодируются в *RPC*-библиотеке времени-выполнения, расположенной на сервере, и через заглушку попадают на серверную часть приложения. На сервере запускается соответствующий запросу программный код и результаты его работы по уже известной схеме передаются клиенту. *RPC* может работать и локально. При локальном вызове весь процесс передачи параметров происходит в одном адресном пространстве.

По своей функциональности *RPC* занимает промежуточное место между уровнем *приложения и транспортным уровнем*. В соответствии с моделью *OSI* этому положению соответствуют уровни *представления и сеансовый*. Таким образом, *RPC* теоретически независим от реализации сети, в частности, от сетевых протоколов транспортного уровня.

Обращение к удаленным объектам (RMI)

Технология RMI (Remote Method Invocation) – это развитие RPC (его объектная реализация).

При обращении к *RMI* клиент использует ссылку, содержащую сетевой адрес сервера и полный путь к объекту на сервере, включая локальный идентификатор объекта в адресном пространстве сервера. Также ссылка кодируется в стек протоколов, используемых для взаимодействия клиента и сервера.

RMI (*Remote Method Invocation*, т. е. вызов удаленного метода), которая интегрирована с *JDK 1.1*, является продуктом компании Jawasoft и реализует распределенную модель вычислений. *RMI* позволяет клиентским и серверным приложениям через сеть вызывать методы клиентов/серверов, выполняющихся в *Java Virtual Machine*. Хотя *RMI* считается легковесной и менее мощной, чем *CORBA* и *DCOM*, она обладает рядом уникальных свойств, таких как распределенное, автоматическое управление объектами и возможностью пересылать сами объекты от машины к машине.

Объект – это стандартно оформленный программный модуль, содержащий данные и операции над этими данными. Данные, содержащиеся в объекте, в программировании называются состояниями (свойствами), а операции над этими данными – *методы*. Доступ к методам можно получить через интерфейс объекта, предоставляемый системами программирования. Объект может реализовывать множество интерфейсов. Для описания интерфейса также может существовать несколько объектов.

Распределенный объект – это объект, *интерфейс* которого находится на другой машине. Характерная особенность распределенных объектов заключается в том, что их данные не распределяются. Они локализованы на одной машине. С других машин доступны только интерфейсы, реализованные в объекте.

При обращении клиента к распределенному объекту управление передаётся программе реализации *интерфейса* объекта аналогичной клиентской заглушки и называется посредником (*proxy*). Посредник

осуществляет транзит параметров от вызывающей программы к операционной системе клиента и обратно, как это делает клиентская заглушка. На стороне сервера транзит параметров от операционной системы сервера к методам объекта и обратно осуществляет аналог серверной заглушке, называемой программа-скелетон (каркас). Объекты распределенных систем существуют в формах, принятых в том или ином языке программирования (рис. 11).

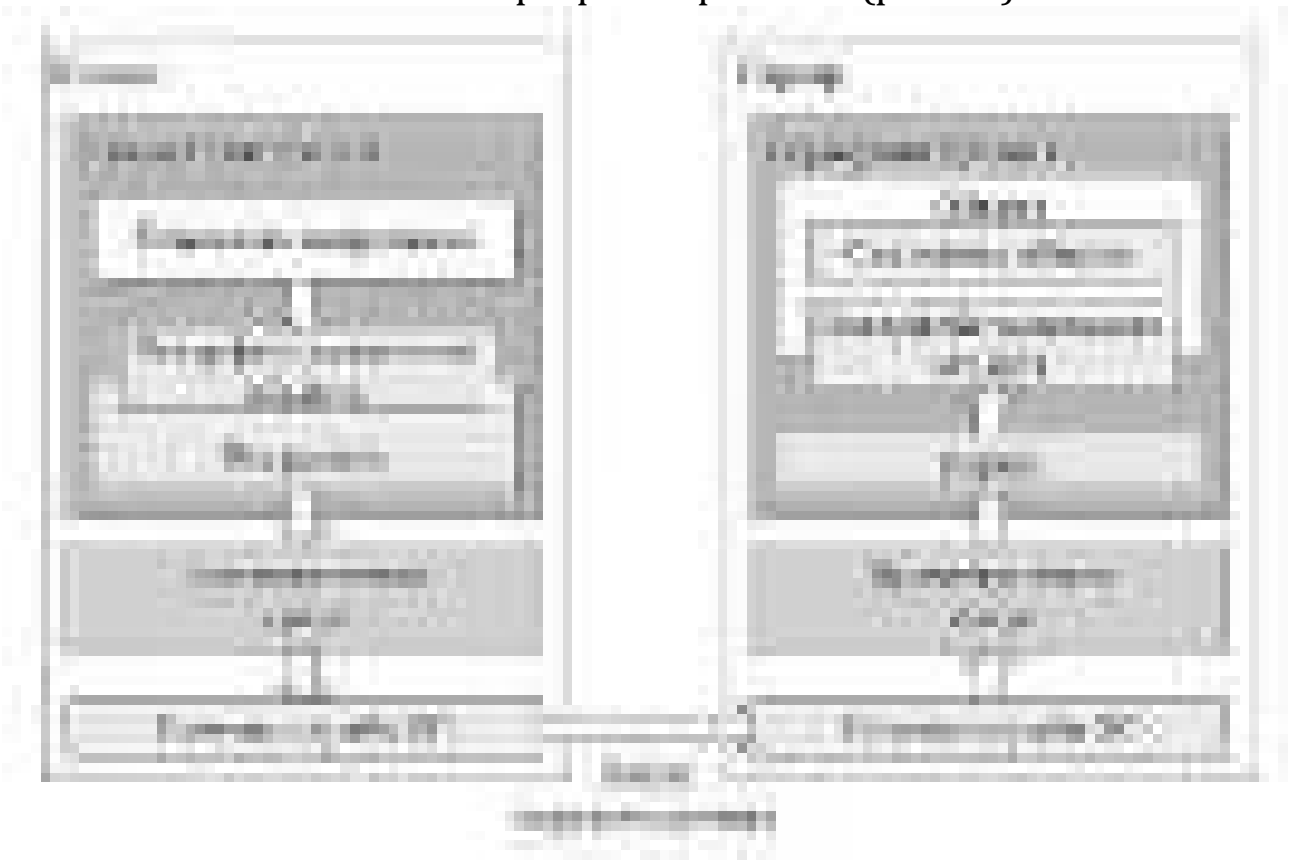


Рис. 11. Использование удаленных объектов

Для описания жизненного цикла в системах с удаленными объектами используются два дополнительных понятия:

– активация объекта: процесс перевода созданного объекта в состояние обслуживания удаленного вызова, то есть связывания его с каркасом и посредником;

– деактивация объекта: процесс перевода объекта в неиспользуемое состояние.

Выделяют три модели использования удаленных объектов:

– модель единственного вызова (*singlecall*);

– модель единственного экземпляра (*singleton*);

– модель активации объектов по запросу клиента (*client activation*).

Первых две модели иногда называют моделями серверной активации (*server activation*), хотя, строго говоря, активация всегда происходит на сервере после какого-либо запроса от клиента.

Связь на основе потоков данных

Связь на основе потоков данных применяется при передаче информации, не имеющей четких ограничений по объему и времени передачи.

Различают три режима передачи потоков данных.

1. *Асинхронный режим*. Временные ограничения на передачу потоков данных не накладываются.

2. *Синхронный режим*. Для каждого элемента потоков данных определяется максимально возможная задержка передачи.

3. *Изохронный режим*. Для каждого элемента данных определяется как максимально возможная, так и минимальная задержка передачи данных.

Потоки данных могут быть *дискретными* (потоки байт или потоки слов) и непрерывными (потоки бит). Также потоки данных могут быть простыми – содержащими только одну последовательность данных – и комплексными – содержащими несколько связанных потоков данных, называемых вложенными потоками данных.

Временные зависимости потоков данных выражаются в виде требований к качеству обслуживания, описывающих, что должна сделать распределенная система, для того чтобы гарантировать сохранение в потоке данных заданных временных соотношений. Для передачи потоков данных распределённая система должна захватить ресурсы, удовлетворяющие требованиям к качеству обслуживания.

Контрольные вопросы

1. Дайте определение промежуточной среды.
2. Охарактеризуйте модели взаимодействия компонентов.
3. Назовите особенности модели взаимодействия архитектуры «клиент – сервер».
4. Охарактеризуйте режимы передачи потоков данных.
5. Что такое технология обращения к удаленным объектам? Каковы ее особенности?

Глава 3

Современные технологии разработки распределенных систем

Технология веб-сервисов

Основы веб-сервисов

Веб-сервисы – это архитектура, обеспечивающая новый уровень распределенности доступа к данным. С помощью веб-сервисов разрабатывают и приобретают компоненты для их встраивания в ИС. Имеется возможность покупать время их работы, формировать программную среду, осуществляющую вызовы модулей из компонентов, которые могут принадлежать и поддерживаться различными независимыми провайдерами. Таким образом, функционал любой программы в сети может стать доступным через веб-сервис. Примером веб-сервиса является система *Passport* на *Hotmail*. Она позволяет аутентифицировать пользователей на собственном сайте.

Основой веб-сервисов являются технологии: *TCP/IP, HTML* и *XML*.

– *TCP/IP* – универсальный протокол передачи данных, понимаемый всеми сетевыми устройствами;

– *HTML* – универсальный язык гипертекстовой разметки для отображения информации на устройствах пользователей;

– *XML* (*Extensible Markup Language*) – универсальный язык, поддерживающий работу с различными типами данных.

Данные технологии являются универсальными и служат основой для понимания веб-сервисов. Интернет-технологии основаны на открытых, формально независимых от поставщиков технологиях, этим достигается их главное преимущество в концепции построения распределенных ИС. Использование технологий *TCP/IP, HTML, XML* позволяет применять их на любых операционных системах, серверах приложений и т. д. Таким образом, веб-сервисы позволяют интегрировать приложения различной природы строить распределенные ИС.

Веб-сервисы, которые организуют связь данных с программами, объектами, базами данных или деловыми операциями, называются *XML*-приложениями. Обмен *XML*-документами в виде сообщений осуществляется между веб-сервисом и программой. Формат передаваемых сообщений, интерфейс передачи сообщений, правила взаимосвязи содержания сообщения с сервисом приложения при его передаче туда и обратно, механизмы публикации и поиска интерфейсов определяют стандарты веб-сервисов.

Веб-сервисы могут использоваться во многих приложениях. Веб-сервисы могут запускаться с различных рабочих станций для обращения к интернет-приложениям. Например, к системе предварительных заказов или системе контроля выполнения заказов. Веб-сервисы активно используются для интеграции B2B приложений электронного бизнеса. Веб-сервисы также позволяют решать проблему интеграции приложений предприятия, осуществляя связь между несколькими приложениями как внутри предприятия, так и вне предприятия. Технологии веб-сервисов выполняют функции связующего звена между различными частями программного обеспечения.

Веб-сервисы обеспечивают стандартный способ взаимодействия с системами управления базами данных – *NET, J2EE, CORBA, ERP,* брокерами интеграции и др.

Работа веб-сервисов заключается в следующем: веб-сервис получает из сетевой среды стандартное *XML*-сообщение, преобразует *XML*-данные в понятный формат для конкретной прикладной программной системы, далее отправляет ответное сообщение. Базовое программное обеспечение может быть создано на любом языке программирования с использованием любой операционной системы и любого связующего программного обеспечения. Например, взаимодействие с веб-сервисами осуществляется посредством ввода данных в *HTML*-формы и отправки этих данных серверу путем добавления их в строку URL адреса:

http://www.google.com/search?q=Skate+boots&btnG=Google+Search

Этот пример иллюстрирует простоту веб-взаимодействия (например, поиска, покупки акций или запроса маршрута движения), где параметры и ключевые слова внедряются непосредственно в URL. В данном случае представлен простой запрос поиска *skate boots* (ботинки с коньками) в строке обращения к поисковой машине *Google*. Ключевое слово search (искать) представляет сервис, к которому будет осуществлено обращение, а параметр *Skate+boots*

является строкой поиска, которая была введена в *HTML*-форме на странице веб-сайта *Google*. Сервис поиска *Google* передаст этот запрос к различным поисковым машинам, которые вернут список *URL* для страниц, на которых имеется соответствие параметру поиска *Skate+boots*. Данный малоэффективный способ поиска в Сети полностью основан на установлении соответствия указанной текстовой строки и индексированных *HTML*-страниц.

XML – лучший способ отправки данных. *XML* предоставляет значительные преимущества при передаче данных через интернет. Отправка запроса в виде *XML*-документа имеет следующие преимущества: возможность определения типов данных и структур, большую гибкость и расширяемость. *XML* может представлять структурированные данные или данные определенного типа (например, допустимо указывать значение поля *size* (размер) как в виде строки цифр, так и в форме числа с плавающей точкой) и содержать больший объем информации, чем это допускает *URL*.

Благодаря тому, что *XML*-документы поддерживают разные типы данных, сложные структуры и объединение схем, современные технологии веб-сервисов обеспечивают значительное преимущество над существующими возможностями обращения к программным приложениям посредством *HTML* и *URL*.

Следующее поколение сети будет основано на программно-ориентированных взаимодействиях. Веб-сервисы предполагают использование созданных для взаимодействия людей глобальных сетей в совершенно иных целях.

Использование веб-сервисов очень выгодно с коммерческой точки зрения. За счет повсеместного распространения веб-сервисов интернет становится более эффективным, особенно при осуществлении коммерческих сделок. Сочетая прямой доступ к программным приложениям и коммерческим документам, веб-сервисы следующего поколения сети обеспечат полностью автоматическое взаимодействие, что позволит обращаться непосредственно к данным программ, игнорируя знакомые веб-страницы. Более того, основные компоненты веб-сервисов, скорее всего, будут предоставляться и публиковаться множеством различных компаний, специализирующихся на отдельных функциональных элементах (проверка полномочий, координация сделок, ведение счетов). Это обеспечит непосредственное взаимодействие «приложение – приложение» – принцип, лежащий в основе веб-сервисов и определяющий их суть и реализацию.

Технология веб-сервисов существует на очень высоком уровне абстракции, позволяя поддерживать множество одновременных определений, которые иногда бывают противоречивы. На простейшем уровне веб-сервисы могут восприниматься как интернет-ориентированные текстовые брокеры интеграции. Любые данные могут преобразовываться в *ASCII*-текст и обратно, и этот подход в течение долгого времени был общим знаменателем для систем графического вывода и систем управления базами данных. Ориентированные на использование текста системы также лежат в основе успешного развития интернета, на котором базируется дополнительная абстракция веб-сервисов. Любой компьютер или операционная система может поддерживать *HTML*, браузеры и веб-сервисы; и при получении по сети файлов им совершенно безразлично и даже неизвестно, с каким типом прикладной системы они взаимодействуют.

Преимущества и недостатки веб-сервисов

К преимуществам *веб-сервисов* можно отнести следующее:

- веб-сервисы позволяют компании интегрировать свои бизнес-процессы с бизнес-процессами бизнес-партнеров и клиентов при меньшей стоимости, чем с использованием других интеграционных технологий. Стоимость подобных решений на основе веб-сервисов доступна даже для *SMB* (*Small and Medium Business*), что откроет для таких компаний новые перспективы развития;
- поскольку веб-сервисы организуются в публичные реестры (*UDDI*-реестры, *ebXML*-реестры или иные), доступные заинтересованным лицам по всему миру, порог выхода компаний на новые рынки снижается, возможности же для наращивания клиентской базы, напротив, возрастают;
- веб-сервисы обеспечивают преемственность в отношении уже имеющихся в информационной системе компании, т. е. можно сказать, что веб-сервисы надстраиваются *над* существующими ИС, но не *вместо* них. Таким образом, обеспечивается сохранность уже сделанных инвестиций в *IT*-инфраструктуру и не идет увеличения требуемых, поскольку нет необходимости в радикальных изменениях;
- построение новых корпоративных решений с применением веб-сервисов реализуется быстрее и совокупно дешевле, поскольку основное внимание сосредотачивается на создании бизнес-логики решения, программирование самих веб-сервисов лишь по необходимости «обрамляет» этот процесс, не требуя больших трудозатрат

за счет эффективного применения повторно используемого кода и адаптированных средств разработки (*IDE* и *SDK*).

К недостаткам (минусам) веб-сервисов можно отнести:

- стандарты интеграции бизнес-процессов, вопросы управления транзакциями и выработка единых бизнес- и *IT*-политик, взаимодействующих посредством веб-сервисов компаний, находятся пока на стадии разработки корпорациями *IBM*, *XLANG* и *Microsoft*, и спецификации *WS-Coordination* и *WS-Transaction* – результат сотрудничества *IBM, Microsoft* и *BEA*. Очевидно, без их четкой формализации и опубликования построение ИС на основе веб-сервисов может идти лишь с переменным успехом;
- динамическое использование информации бизнес-реестров веб-сервисов, вызов веб-сервисов требует решения вопросов доверительности отношений между различными бизнес-реестрами. Кроме того, есть трудности в совместном использовании бизнес-реестров различных форматов (например, задача поиска определенного веб-сервиса в *UDDI*-реестре и *ebXML*-реестре требует различных подходов в силу различия *XML*-документов, описывающих один и тот же веб-сервис в каждом из этих реестров). Надо отметить, что есть попытки решить эту проблему созданием единого браузера реестров. Например, графическая утилита *Registry Browser* корпорации *Sun Microsystems*, реализующая набор интерфейсов *JAXR* (*Java API for XML Registries*);
- добавление к функциям сервера приложений функциональности провайдера веб-сервисов в силу новизны технологий может представлять определенную трудность;
- вопросы безопасности функционирования информационной системы на основе веб-сервисов пока не урегулированы до конца. Спецификация *WS-Security* – продукт деятельности корпораций *IBM* и *Microsoft* – в настоящее время достаточно молода, не «устоялась» и частично все еще дорабатывается. Однако в силу общности положений спецификации *WS-Security* уже готовится к выпуску следующий слой спецификаций, посвященных вопросам безопасности: *Web Services Policy Assertions, Web Services Policy Attachments, Web Services Policy Framework, Web Services Trust, Web Services Secure Conversation, Web Services Federation.*

Итак, преимущества представляют собой стратегические бизнес-преимущества компаний, а недостатки имеют технологический характер и обусловлены новизной технологий, решение этих проблем лишь вопрос времени.

Определение Веб-сервиса

Сервисом (service) называется ресурс, реализующий бизнес-функцию, обладающий следующими свойствами:

- является повторно используемым;
- определяется одним или несколькими явными технологически-независимыми интерфейсами;
- слабо связан с другими подобными ресурсами и может быть вызван посредством коммуникационных протоколов, обеспечивающих возможность взаимодействия ресурсов между собой.

Веб-сервисом называется программная система, идентифицируемая строкой *URL*, чьи интерфейсы и привязки определены и описаны посредством *XML*. Описание этой программной системы может быть найдено другими программными системами, которые могут взаимодействовать с ней согласно этому описанию посредством сообщений, основанных на *XML* и передаваемых с помощью интернет-протоколов.

Взаимодействие с Веб-сервисами

Веб-сервисы поддерживают несколько способов обмена сообщениями. Уровень абстракции, на котором оперируют веб-сервисы, подразумевает такие стили взаимодействия, как эмуляция удаленного вызова процедуры (*Remote Procedure Call, RPC*), асинхронный обмен сообщениями, однонаправленная передача сообщений, широковещание и публикация/подписка. Основные СУБД, такие как *Oracle, SQL Server* и *DB2,* поддерживают анализ *XML* и службы преобразования, обеспечивая непосредственное взаимодействие между веб-сервисами и СУБД. Производители связующего программного обеспечения обычно также предоставляют возможность привязки веб-сервисов к своим программным системам (серверам приложений и брокерам интеграции). Следовательно, для пользователя взаимодействие с веб-сервисами может проявляться в интерактивной или пакетной форме, поддерживающей синхронную и асинхронную модели связи; а также как пользовательский интерфейс, написанный с использованием *Java, VB (Visual Basic)*, офисных приложений, браузеров или «толстых» клиентов СУБД. Такое взаимодействие может привязываться к любому типу базовой (более низкого уровня) программной системы.

Веб-сервисы выполняют *RPC* и документно-ориентированное взаимодействия. Стандарты и технологии веб-сервисов обычно

подразумевают *два основных типа моделей* взаимодействия приложений:

– удаленный вызов процедуры (онлайновая);

– документно-ориентированный (пакетная).

RPC-ориентированные взаимодействия

RPC-ориентированные взаимодействия удобны для краткого обмена данными. В *RPC*-ориентированном взаимодействии запросы веб-сервисов приобретают форму вызова метода или процедуры с соответствующими входными или выходными параметрами. В отличие от документно-ориентированного взаимодействия, RPC-ориентированное взаимодействие производит отправку документа, специально отформатированного для передачи в отдельную логическую программу или базу данных.

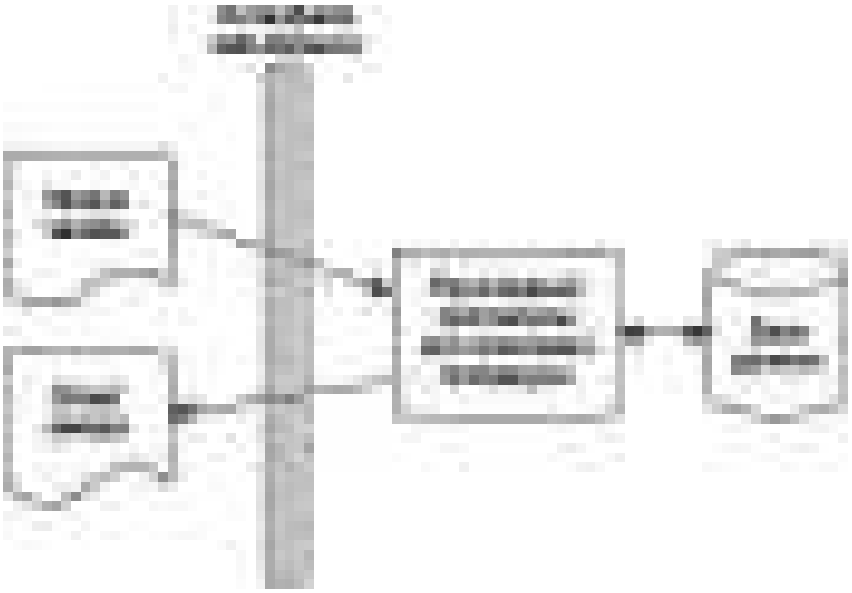


Рис. 12. Веб-сервисы поддерживают интерактивный *заказ в форме запроса/ответа*

Документно-ориентированные взаимодействия

Документно-ориентированные взаимодействия удобны для обмена большими объемами данных. При документно-ориентированном взаимодействии запросы веб-сервиса имеют форму завершенного *XML*-документа, предназначенного для обработки целиком.

Документно-ориентированные взаимодействия зачастую предполагают, что использующие веб-сервисы стороны заранее согласовали порядок оформления общих документов, таких как заказ на приобретение, счет за доставку или общий счет. Эти стороны обычно идентифицируются как «торговые партнеры» или «со-

трудничающие партнеры». Торговые партнеры также согласовывают общий поток выполнения процесса или модель взаимодействия при обмене документами, например, оговаривают необходимость подтверждения квитанции заказа на приобретение, передачу специальной информации о состоянии в ответ на запрос заказа или отправку сигнала оповещения по электронной почте после отгрузки заказа. В ходе реализации бизнес-процесса необходим обмен полными документами. Если до этого документ содержал общую, фрагментированную информацию, то теперь требуется согласованное заполнение специальных разделов, таких как цена покупки или обязательная дата доставки.

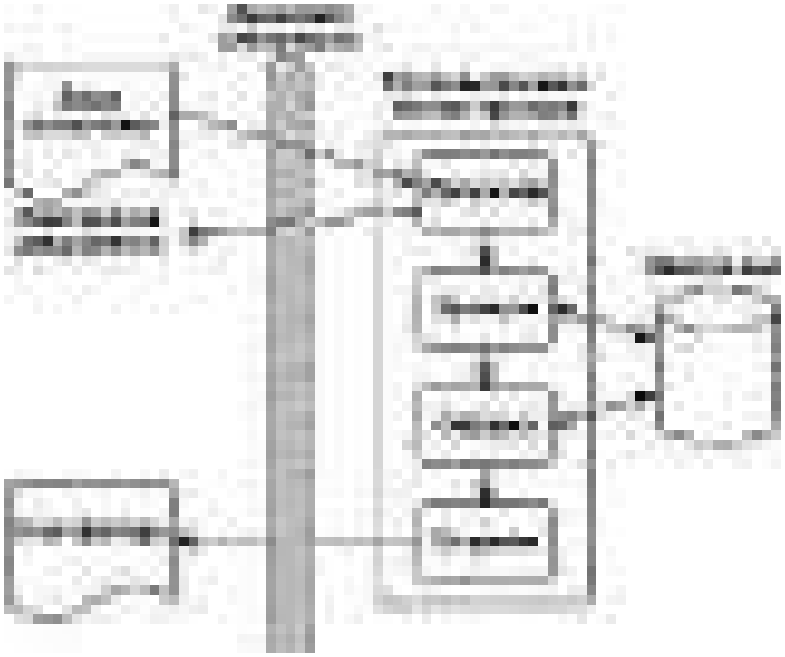


Рис. 13. Веб-сервис обрабатывает полный заказ на поставку

Технология Веб-сервисов

Порядок описания, поиска и взаимодействия веб-сервисов друг с другом определяют стандарты. Взаимодействующие через интернет программы должны уметь обнаруживать друг друга, находить информацию, позволяющую им осуществить связь, понимать, какая модель должна быть применена (простая, типа «запрос/ответ», или более сложная последовательность), и договариваться об использовании таких услуг, как защита информации, подтверждение передачи сообщений и составление сделок. Некоторые из этих сервисов реализуются существующими технологиями и предлагаемыми стандартами, а другие – нет. Использующее веб-сервисы сообщество стремится удовлетворить все эти требования, но это эволюционный процесс, как и сам интернет. С самого

начала инфраструктура и стандарты веб-сервисов подразумевали возможность расширения (также как до них *XML* и *HTML*), что позволяет использовать их сразу же после появления новых стандартов и технологий.

Веб-сервисы требуют использования нескольких смежных *XML*-технологий.

1. Язык *XML* – фундамент, на котором строятся веб-сервисы. Он предоставляет язык определения данных и порядок их обработки. XML представляет семейство связанных спецификаций, публикуемых и поддерживаемых интернет-консорциумом (*World Wide Web Consortium, W3C*) и другими организациями.

2. *SOAP* (*Simple Object Access Protocol*), разработанный консорциумом *W3C*, определяет формат запросов к веб-сервисам. Сообщения между веб-сервисом и его пользователем пакуются в так называемые *SOAP*-конверты (*SOAP envelopes*, иногда их ещё называют *XML*-конвертами). Само сообщение может содержать либо запрос на осуществление какого-либо действия, либо ответ – результат выполнения этого действия.

3. *WSDL* (*Web Services Description Language*) – технология, основанная на *XML*, определяющая интерфейсы веб-сервисов, типы данных и сообщений, а также модели взаимодействия и протоколы связывания. Перед развертыванием сервиса разработчик составляет его описание на языке *WSDL*, указывает адрес веб-сервиса, поддерживаемые протоколы, перечень допустимых операций, форматы запросов и ответов.

4. Технология *UDDI* (*Universal Description, Discovery and Integration*) – реестр веб-сервисов и механизм поиска. Он используется для хранения и упорядочения деловой информации, а также для нахождения указателей на интерфейсы Веб-сервисов.

Пример использования

Стандарты веб-сервисов обычно используются совместно и согласованно. После обнаружения *WSDL* в *UDDI* или другом месте генерируется *SOAP*-сообщение для отправки на удаленный сайт.

Как видно из рис. 14, при предоставлении документа по адресу веб-сервиса программа использует *XML*-схему определенного типа (такую как *WSDL*), позволяющую преобразовать данные из ее входного источника (в этом примере структурированный файл) и на основе того же *WSDL*-файла создать экземпляр *XML*-документа в формате, согласованном с целевым веб-сервисом. *WSDL*-файл ис-

пользуется для определения как входного, так и выходного преобразования данных.

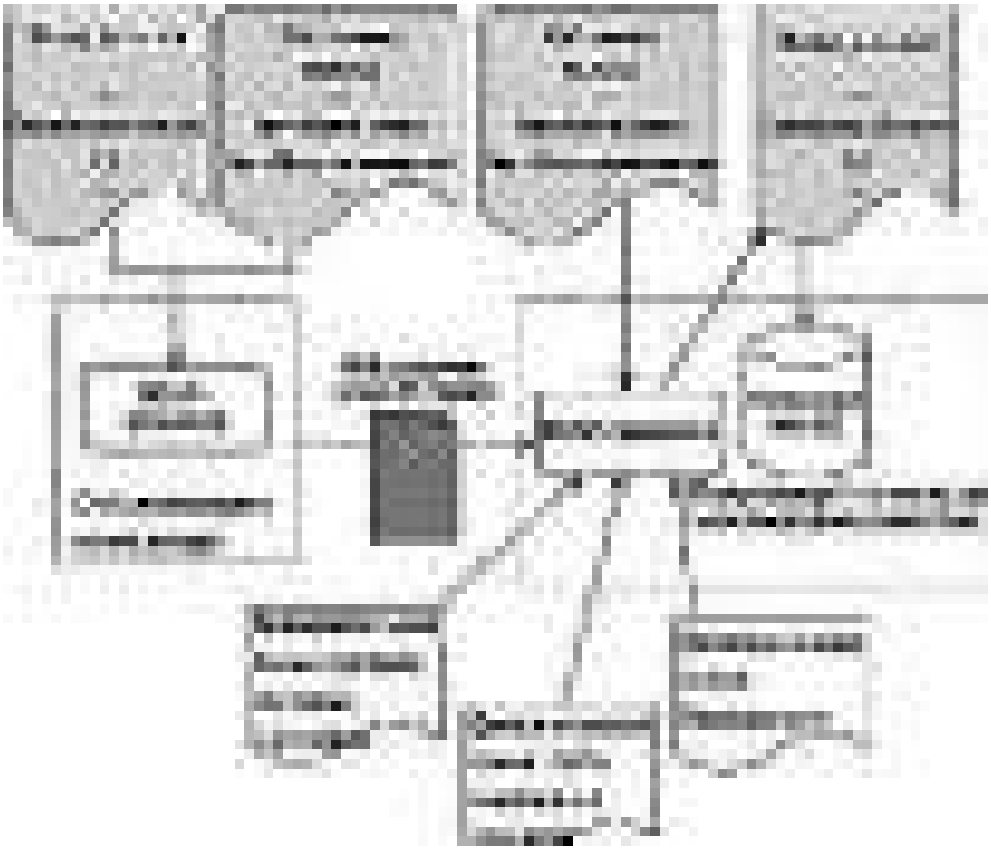


Рис. 14. Веб-сервисы используют XML-документы
и осуществляют преобразование данных

SOAP-процессор отправляющего компьютера преобразует данные из собственного формата в тип данных, предопределенный в соответствии с содержащейся в *WSDL*-файле *XML*-схемой на основе таблиц преобразования для текстов, значений с плавающей точкой и других данных. Таблицы преобразования «связывают» собственные типы данных с соответствующими конкретной *XML*-схеме. (Стандартное преобразование типов широко используется в *Java, Visual Basic, CORBA* и других известных системах. Многие средства *XML* позволяют настраивать специальные преобразования типов.) *SOAP*-процессор получающего компьютера выполняет обратное преобразование данных из типов *XML*-схемы в собственные типы данных.

Файлы описаний веб-сервисов обычно регистрируются с помощью *URL*. *URL*, повсеместно используемый в сети, указывает на IP-адрес, соответствующий веб-ресурсу. Схемы веб-сервисов являются одной из форм веб-ресурса, они содержатся в доступных через интернет файлах и к ним применим тот же механизм, что используется

при загрузке *HTML*-файлов. Главное отличие между загрузкой *HTML*-файла и обращением к ресурсу веб-сервиса заключается в том, что веб-сервис оперирует *XML*-документами, а не *HTML*-документами и опирается на соответствующие технологии, такие как использование схем, преобразование, проверка подлинности, что и обеспечивает поддержку удаленного соединения приложений. Но способ, согласно которому схемы веб-сервисов публикуются и загружаются, одинаков: *HTTP*-операция по указанному *URL*.

Для проверки достоверности сообщений веб-сервисы используют *XML*-схемы. После получения документа реализация веб-сервиса сначала должна проанализировать *XML*-сообщение и удостовериться в корректности данных, выполнить проверку качества услуг (*Quality-of-Service*), такую как проверку политики безопасности или соглашений торговых партнеров, а затем произвести последовательность связанных с данным документом коммерческих операций. Веб-сервис на вымышленном нами сайте *skateboots.com* размещен в папке *skateboots.com/order*, на которую и указывает *URL*.

Веб-сервис, доступный по данному интернет-адресу, идентифицируется с помощью публичного *WSDL*-файла, который может быть загружен на отправляющий компьютер и использоваться при генерации сообщения. Компания *Skateboots Company* также осуществляет отправку в общедоступный каталог *UDDI*-листинга, позволяющего клиентам находить компанию с помощью технологии *UDDI*. В общем случае любой, кто хочет взаимодействовать с веб-сервисом, размещающим или контролирующим по сети заказы для *Skateboots Company*, для генерации сообщения должен найти способ получения и использования *WSDL*-файла.

Размещенные по адресу skateboots.com программы представляют собой прослушивающий *HTTP*-процесс, связанный с соответствующими веб-сервисами для распознавания *XML*-сообщений, определенных в данном формате. Эти программы включают в себя *XML*-анализаторы и преобразователи. Кроме того, они осуществляют конвертацию данных *SOAP*-сообщения в форматы, необходимые для системы ввода заказов компании *Skateboots Company.*

Технологии веб-сервисов сформировались из основных структур. Этих технологий достаточно для построения, развертывания и публикации базовых веб-сервисов. Необходим лишь базовый протокол *SOAP*. С момента появления веб-сервисов к ним все время добавляются другие технологии. Хотя для деловой связи, а также для создания «моста» с несовместимыми технологиями вполне до-

статочно базовых принципов, данная форма веб-взаимодействия тем не менее была одобрена очень быстро.

Определение сервисно-ориентированной архитектуры

С функциональной точки зрения бизнес-приложение распадается на совокупность взаимодействующих между собой сервисов. Эту совокупность взаимодействующих сервисов можно отождествить с еще одним ключевым понятием – сервисно-ориентированной архитектурой:

Компонентная модель, состоящая из отдельных функциональных модулей приложений, называемых сервисами, имеющих определенные согласно некоторым общим правилам интерфейсы и механизм взаимодействия между собой, называется ***сервисно-ориентированной архитектурой (Service-Oriented Architecture, SOA)****.*

Переходя в мир абстракций, можно дать следующее, более сильное определение сервисно-ориентированной архитектуры:

Архитектура приложений, в рамках которой *все функции приложения* являются *независимыми сервисами* с ч*етко определенными интерфейсами*, которые можно вызывать в нужном порядке с целью формирования бизнес-процессов, называется **сервисно-ориентированной архитектурой**.

На сегодняшний день устоявшегося, принятого *IT*-сообществом, определения *SOA* нет. Здесь мы приводим определение, которое, на наш взгляд, наиболее полно и точно отражает современное состояние этой концепции).

Поясним второе определение:

- *«все функции приложения»* – как мы уже упомянули, любое приложение с функциональной точки зрения может быть представлено совокупностью функций; ресурс, реализующий функцию, есть сервис. Таким образом, приведенное определение требует для представления и реализации любого приложения в рамках *SOA* проведения его полной декомпозиции до уровня отдельных функций;
- *«являются независимыми сервисами»* – в понятие независимости сервиса вкладывается следующий смысл: сервисы функционируют независимо от других информационных систем, являются функционально самостоятельными объектами. Они представляют собой «черные ящики» для любых внешних приложений: внешние приложения не знают, как сервис формирует из входных данных выходные. Все, что им известно – необходимо подать на вход сервиса и что следует ожидать на его выходе;

- *«с четко определенными интерфейсами»* – функция (или функции), которую реализует данный сервис, должна быть однозначно описана согласно определенным, принятым для всех сервисов, правилам. Должен быть описан набор и типы входных данных, а также набор и типы выходных данных;
- *«с ... интерфейсами, которые можно вызывать»* – данное требование обусловлено необходимостью обеспечения взаимодействия между различными сервисами: для внешних по отношению к сервису информационных систем не должно иметь значения на каком языке программирования реализован сервис (веб-сервис), на какой программно-аппаратной платформе он функционирует, локально или удаленно он расположен. Внешняя информационная система должна иметь возможность взаимодействовать с сервисом (т. е. передать ему входные данные и получить выходные) вне зависимости от указанных его особенностей.

Требования к SOA

SOA, будучи практической концепцией, должна соответствовать определенным требованиям, предъявляемым к ней современным состоянием бизнес-отношений и информационных технологий, а также тенденциями их совместного развития:

1) обеспечивать преемственность инвестиций в *IT*, сохранение существующих информационных систем и их совместное эффективное использование для повышения *ROI* от *IT*-вложений;

2) обеспечивать реализацию различных типов интеграции:

a) *пользовательская интеграция (user integration)* – обеспечение взаимодействия информационной системы с конкретным пользователем;

b) *интеграция приложений (application connectivity)* – обеспечение взаимодействия приложений;

c) *интеграция процессов (process integration)* – интеграция бизнес-процессов;

d) *информационная интеграция (information integration)* – интеграция с целью обеспечения доступности информации и данных;

e) *интеграция новых приложений (build to integrate)* – интеграция новых приложений и сервисов в существующие информационные системы.

3) обеспечивать поэтапность внедрения вновь созданных и миграции существующих информационных систем;

4) иметь стандартизованную технологическую обеспеченность реализации и инструментарий разработки, предоставляющие наилучшие

возможности повторного использования приложений, внедрения новых и миграции существующих информационных систем;

5) позволять реализацию различных моделей построения информационных систем, в особенности таких, как портальные решения, *grid*-системы.

Сегодняшний уровень развития *SOA* позволяет утверждать, что все указанные требования в той или иной мере выполняются.

Необходимо отметить, что *SOA – не синоним веб-сервисов, а веб-сервисы – не единственный способ реализации SOA*. *SOA* не является технологией или набором технологий, это – концепция, абстрактное представление реализации информационных систем с помощью сервисов безотносительно конкретных технологий. Как нетрудно заметить, в *SOA* присутствуют элементы объектного подхода к построению информационных систем: декомпозиция (приложений на отдельные функции) и инкапсуляция. Однако подчеркнем, термин «объектно-ориентированный» по отношению к *SOA* не является корректным, т. к. в *SOA* отсутствуют все необходимые элементы объектно-ориентированной парадигмы. Более правильно называть *SOA* концепцией, использующей объектный подход.

Веб-сервисы, в свою очередь, являются лишь технологиями, с помощью которых можно эффективно реализовать сервисно-ориентированную архитектуру. Существуют и другие технологии, с применением которых можно реализовать *SOA*, например, упомянутая выше *CORBA.*

Преимущества SOA

Концепции *SOA* и Веб-сервисов не новы; на протяжении последних десяти лет были и другие технологии построения распределенных информационных систем, например, *CORBA* и *DCOM*. Однако, со временем решения, построенные на их основе, перестали удовлетворять основным требованиям, предъявляемым к бизнес-решениям: максимально низкая *TCO*, максимально высокий *ROI*, короткий цикл разработки и внедрения информационной системы, широкие возможности интеграции разнородных систем.

В отличие от известных моделей технологии веб-сервисов и *SOA*:

– являются *open-source*;

– основаны на общепринятых и общеупотребимых технологиях и стандартах: *XML, HTTP, TCP/IP*;

– позволяют достичь наилучших показателей *TCO* и *ROI*, нежели какие-либо иные существующие на сегодняшний день интеграционные технологии;

– находятся на уровне зрелости, необходимом и достаточном для успешного применения большим количеством *ISV* (англ. *Independent Software Provider*) по всему миру.

Стек технологий веб-сервисов

Итак, концепция сервисно-ориентированной архитектуры подразумевает реализацию бизнес-процессов предприятия в виде совокупности сервисов, взаимодействующих друг с другом либо с пользователями в определенной последовательности и в соответствии с определенными правилами. Отметим, что это весьма нетривиальная задача – определить эту «определенную последовательность» и «определенные правила» таким стандартным для разработки приложений образом, чтобы охватить весь спектр существующих сценариев взаимодействия бизнес-объектов, учитывая исторически сложившееся многообразие технической и технологической реализации этого взаимодействия и самих бизнес-объектов. Решить эту задачу в рамках какой-либо единой технологии пока не удалось. И хотя тенденция консолидации существующего множества технологий веб-сервисов в настоящее время налицо, для реализации сервисно-ориентированных архитектур с помощью веб-сервисов сейчас применяется совокупность технологий, образующих так называемый стек технологий веб-сервисов (см. рис. 15).

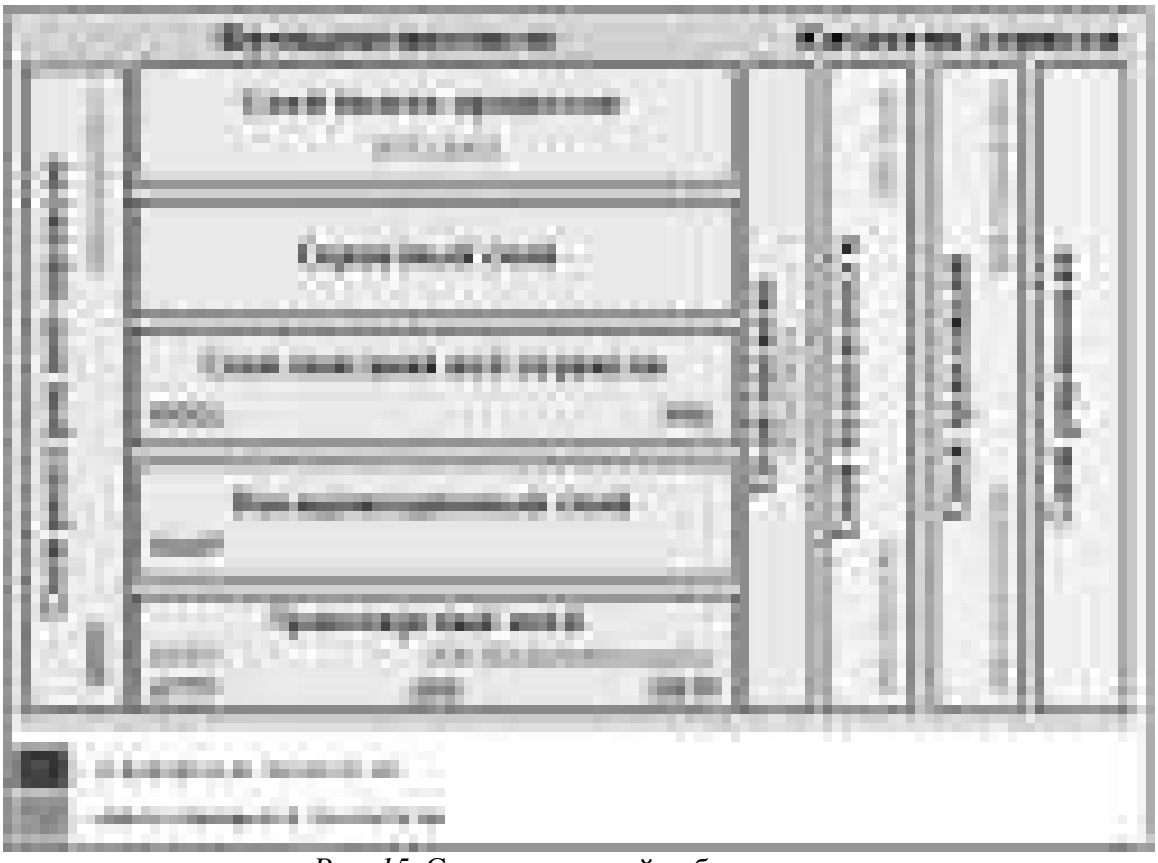


Рис. 15. Стек технологий веб-сервисов

Стек технологий веб-сервисов принципиально разбивается на следующие две составляющие:

- технологии, обеспечивающие функциональность веб-сервисов (*Functions*);
- технологии, обеспечивающие качество сервиса веб-сервисов (*Quality of service*).

Эти составляющие в свою очередь образуются несколькими слоями (*layers*):

– технологии, обеспечивающие функциональность веб-сервисов:

- транспортный слой (*transport layer*);
- коммуникационный слой (*service communication layer*);
- слой описаний сервисов (*service description layer*);
- сервисный слой (*service layer*);
- слой бизнес-процессов (*business process layer*);
- слой реестров сервисов (*service registry layer*).

– технологии, обеспечивающие качество сервиса веб-сервисов:

- слой политик (*policy layer*);
- слой безопасности (*security layer*);
- слой транзакций (*transaction layer*);
- слой управления (*management layer*).

В целях понимания назначения слоев дадим краткое описание каждого из них.

Таблица 1

Описание слоев стека технологий

№ п/п	Наименование слоя	Назначение слоя	Технологии, реализующие слой
Функциональность (Functions)			
1	Транспортный слой (*Transport layer*)	Описывает средства обмена данными между веб-сервисами	Стандартные: *HTTP*, *JMS* (для *Java*-приложений), *SMTP* Нарождающиеся: *WS-ReliableMessaging, BEEP*
2	Коммуникационный слой (*Service communication layer*)	Описывает средства формализации механизмов использования транспортных протоколов веб-сервисами. Используя метафоры, можно отождествить транспортный протокол с дорогой между веб-сервисами, а механизмы его использования, определяемые коммуникационным слоем, с грузовыми машинами, перевозящими по ней от сервиса к сервису сообщения	Стандартные: *SOAP* Нарождающиеся: *REST*

№ п/п	Наименование слоя	Назначение слоя	Технологии, реализующие слой
3	Слой описаний сервисов (*Service description layer*)	Описывает средства формализации интерфейсов веб-сервисов с целью обеспечения их функционирования независимо от программно-аппаратной платформы реализации или языка программирования. Различают два вида описаний сервиса: - операционное (*operational*); - полное (*complete*)	Стандартные: X*ML, WSDL* Нарождающиеся: *ebXML*
4	Сервисный слой (*Service layer*)	Описывает программное обеспечение, вызываемое с помощью *WSDL*-описаний интерфейсов веб-сервисов. В частности, это сами вВеб-сервисы	
5	Слой бизнес-процессов (*Business process layer*)	Описывает возможности организации веб-сервисов для реализации бизнес-процессов и потоков работ. При этом определяются правила, задающие последовательность взаимодействия веб-сервисов с целью удовлетворения бизнес-требованиям	Стандартные: в настоящее время нет Нарождающиеся: *BPEL4WS*
6	Слой реестров сервисов (*Service registry layer*)	Описывает возможности организации веб-сервисов в иерархические библиотеки, позволяющие публикацию, поиск и вызов веб-сервисов по их *WSDL*-описаниям интерфейсов	Стандартные: *UDDI* Нарождающиеся: *WS-Inspection*
Качество сервиса (*Quality of service*)			
7	Слой политик (*Policy layer*)	Описывает правила и условия, согласно которым веб-сервисы могут быть использованы. Поскольку данные правила и условия относятся как к функциональному аспекту веб-сервисов, так и к аспекту обеспечения качества сервиса, данный слой является общим для обоих аспектов	Стандартные: в настоящее время нет Нарождающиеся: *WS-Policy, WS-PolicyAssertions* и *WS-PolicyAttachment*
8	Слой безопасности (*Security layer*)	Описывает возможности обеспечения безопасности веб-сервисов и безопасности их функционирования (авторизация, аутентификация и разделение доступа)	Стандартные: *WS-Security* Нарождающиеся: *WS-SecureConversation, WS-Federation, WS-Authorization, WS-Trust* и *WS-Privacy*
9	Слой транзакций (*Transaction layer*)	Описывает свойство транзакционности распределенных систем на основе веб-сервисов для обеспечения надежности их функционирования	Стандартные: в настоящее время нет Нарождающиеся: *WS-Transaction* и *WS-Coordination*
10	Слой управления (*Management layer*)	Описывает возможности управления веб-сервисами и характеристиками их функционирования	

Представленный выше стек технологий веб-сервисов вводит иерархию во множество технологий веб-сервисов в соответствии с их функциональным назначением. При этом в таблице указаны лишь наиболее широко применяемые и устоявшиеся технологии. Стандартными названы технологии, получившие официальный статус стандартов международных консорциумов по разработке *IT*-стандартов (*W3C*). В действительности, спектр технологий, описывающих те или иные аспекты использования веб-сервисов либо сервисно-ориентированных архитектур, крайне широк, в частности потому, что процесс разработки данных технологий является открытым – любая компания, некоммерческое объединение специалистов или даже один специалист может разработать и опубликовать спецификацию разработанной им технологии. В настоящее время это даже стало серьезной проблемой рынка веб-сервисов – количество спецификаций столь велико, что при фактической нерегламентированности глобального процесса их разработки, ввода в действие и использования, появляется опасность ввергнуть индустрию в хаос и технологическую разобщенность. А технологическая разобщенность – как раз то, от чего хотели уйти, разрабатывая веб-сервисы и закладывая в качестве их концептуальной и технологической основы открытые и наиболее широко применяемые *IT*-технологии.

Принципы взаимодействия Веб-сервисов в рамках сервисно-ориентированной архитектуры

Итак, технологический фундамент Веб-сервисов образуется следующими технологиями:

- *eXtensible Markup Language (XML);*
- *Simple Object Access Protocol (SOAP);*
- *Universal Description, Discovery and Integration (UDDI);*
- *Web Services Description Language (WSDL).*

Данные технологии обеспечивают реализацию базовых свойств веб-сервиса, описанных в его определении. Они же лежат в основе взаимодействия веб-сервисов в рамках сервисно-ориентированной архитектуры (см. рис. 16).

Различают следующие три основных архитектурных компонента сервисно-ориентированной архитектуры:

- *пользователь сервиса*: приложение, программный модуль либо сервис, осуществляющий поиск и вызов необходимого сервиса из реестра сервисов по описанию сервиса, а также использующий

сервис, предоставляемый провайдером сервиса в соответствии с интерфейсом сервиса;

- *провайдер сервиса*: приложение, программный модуль либо сервис, осуществляющий реализацию сервиса в виде веб-сервиса, прием и исполнение запросов пользователей сервиса, а также публикацию сервиса в реестре сервисов;
- *реестр сервисов*: библиотека сервисов, предоставляющая пользователям сервиса средства поиска и вызова необходимого сервиса и принимающая запросы провайдеров сервисов на публикацию сервисов.

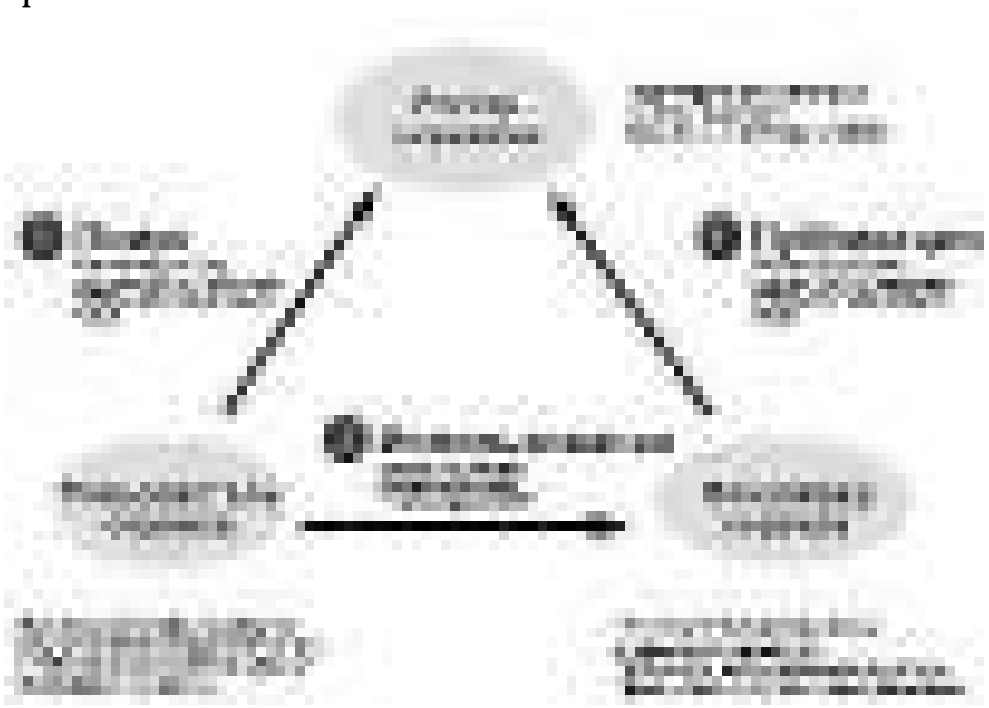


Рис. 16. Взаимодействие между компонентами сервисно-ориентированной архитектуры

Каждый компонент может играть либо лишь одну роль (быть, например, только пользователем сервиса) либо одновременно сразу несколько ролей (например, быть провайдером одних сервисов и пользователем других).

Заметим, что в данном описании компонентов сервисно-ориентированной архитектуры и взаимодействия между ними следует различать термины «сервис» и «веб-сервис». Под сервисом понимается бизнес-функция, под веб-сервисом – программная реализация бизнес-функции (сервиса).

В ходе взаимодействия друг с другом компоненты сервисно-ориентированной архитектуры выполняют следующие основные операции:

публикация: для того, чтобы сервис был доступным (вызываемым) пользователям сервиса, необходимо сделать его интерфейс известным им;

поиск: пользователь сервиса должен иметь возможность найти в реестре сервисов необходимый сервис, удовлетворяющий заданным критериям;

связывание и вызов: после получения описания сервиса, пользователь сервиса должен иметь возможность вызвать и использовать сервис в соответствии с описанием сервиса.

Рассматривая взаимодействие компонентов сервисно-ориентированной архитектуры необходимо отметить наличие (и различие) следующих двух артефактов:

описание сервиса: определяет формат запроса и отклика при взаимодействии пользователя сервиса и провайдера сервиса, а также требуемое качество сервиса;

сервис: собственно сервис, который может быть вызван и использован пользователем сервиса в соответствии с опубликованным интерфейсом сервиса.

С целью выработки и популяризации стандартов, описывающих взаимодействие веб-сервисов в рамках *COA*, создано международное объединение около 150 ведущих компаний, называющееся *WS-I* (*Web Services Interoperability Organization*). Важным результатом работы данного объединения стало создание и утверждение в 2003 г. так называемого *WS-I Basic Profile 1.0* – пакета базовых спецификаций веб-сервисов, взаимоувязанных с целью обеспечения широких возможностей взаимодействия веб-сервисов.

Для создания веб-сервисов могут использоваться такие инструменты, как *Open Net* (компании *Sun*), *.NET* (*Microsoft*) и *Web Services* (*IBM*). Существуют также инструменты с открытыми кодами (*open source frameworks*), например, проект *Mono Project* (*www.go-mono.com*), который предоставляет систему компиляции, выполнения кода и библиотек для работы одних и тех же веб-сервисов на всех платформах, включая *Unix*.

Контрольные вопросы

1. Перечислите основные технологии разработки распределенных систем.
2. Опишите преимущества и недостатки веб-сервисов.
3. Дайте определение сервисно-ориентированной архитектуры.
4. Какие технологии обеспечивают функциональность веб-сервисов?
5. Перечислите принципы взаимодействия веб-сервисов в рамках сервисно-ориентированной архитектуры.

Глава 4

Распределенные БД

Свойства РБД

Распределенные базы данных невозможно рассматривать вне контекста более общей и более значимой темы распределенных информационных систем.

Под распределенной (*distributed database – ddb*) обычно подразумевают базу данных, включающую фрагменты из нескольких баз данных, которые располагаются на различных узлах сети компьютеров, и, возможно управляются различными СУБД. Распределенная база данных выглядит с точки зрения пользователей и прикладных программ как обычная локальная база данных.

Лучшее определение распределенных баз данных (*ddb*) предложил Дэйт. Он установил 12 свойств или качеств идеальной *ddb*:

- локальная автономия (*local autonomy*);
- независимость узлов (*no reliance on central site*);
- непрерывное функционирование (*continuous operation*);
- прозрачность расположения (*location independence*);
- прозрачная фрагментация (*fragmentation independence*);
- прозрачное тиражирование (*replication independence*);
- обработка распределенных запросов (*distributed query processing*);
- обработка распределенных транзакций (*distributed transaction processing*);
- независимость от оборудования (*hardware independence*);
- независимость от операционных систем (*operationg system independence*);
- прозрачность сети (*network independence*);
- независимость от баз данных (*database independence*).

Локальная автономия. Это качество означает, что управление данными на каждом из узлов распределенной системы выполняется

локально. База данных, расположенная на одном из узлов, является неотъемлемым компонентом распределенной системы. Будучи фрагментом общего пространства данных, она, в то же время функционирует как полноценная локальная база данных; управление ею выполняется локально и независимо от других узлов системы.

Независимость от центрального узла. В идеальной системе все узлы равноправны и независимы, а расположенные на них базы являются равноправными поставщиками данных в общее пространство данных. База данных на каждом из узлов самодостаточна – она включает полный собственный словарь данных и полностью защищена от несанкционированного доступа.

Непрерывное функционирование. Это качество можно трактовать как возможность непрерывного доступа к данным (известное «24 часа в сутки, 7 дней в неделю») в рамках *ddb* вне зависимости от их расположения и вне зависимости от операций, выполняемых на локальных узлах. Это качество можно выразить лозунгом «данные доступны всегда, а операции над ними выполняются непрерывно».

Прозрачность расположения. Это свойство означает полную прозрачность расположения данных. Пользователь, обращающийся к ddb, ничего не должен знать о реальном, физическом размещении данных в узлах информационной системы. Все операции над данными выполняются без учета их местонахождения. Транспортировка запросов к базам данных осуществляется встроенными системными средствами.

Прозрачная фрагментация. Это свойство трактуется как возможность распределенного (то есть на различных узлах) размещения данных, логически представляющих собой единое целое. Существует фрагментация двух типов: горизонтальная и вертикальная. Первая означает хранение строк одной таблицы на различных узлах (фактически, хранение строк одной логической таблицы в нескольких идентичных физических таблицах на различных узлах). Вторая означает распределение столбцов логической таблицы по нескольким узлам.

Рассмотрим пример, иллюстрирующий оба типа фрагментации. Имеется таблица *employee* (*emp_id, emp_name, phone*), определенная в базе данных на узле в Фениксе. Имеется точно такая же таблица, определенная в базе данных на узле в Денвере. Обе таблицы хранят информацию о сотрудниках компании. Кроме того, в базе данных на узле в Далласе определена таблица *emp_salary* (*emp_id, salary*).

Тогда запрос получить «информацию о сотрудниках компании» может быть сформулирован так:

```
select * from employee@phoenix, employee@denver order by emp_id
```

В то же время запрос «получить информацию о заработной плате сотрудников компании» будет выглядеть следующим образом:

```
select employee.emp_id, emp_name, salary from employee@denver, employee@phoenix, emp_salary@dallas order by emp_id
```

Прозрачность тиражирования. Тиражирование данных – это асинхронный (в общем случае) процесс переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы. В данном контексте *прозрачность* тиражирования означает возможность переноса изменений между базами данных средствами, невидимыми пользователю распределенной системы. Данное свойство означает, что тиражирование возможно и достигается внутрисистемными средствами.

Обработка распределенных запросов. Это свойство *ddb* трактуется как возможность выполнения операций выборки над распределенной базой данных, сформулированных в рамках обычного запроса на языке *SQL*. То есть операцию выборки из *ddb* можно сформулировать с помощью тех же языковых средств, что и операцию над локальной базой данных. Например:

```
select customer.name, customer.address, order.number, order.date
from customer@london, order@paris where customer.cust_number =
order.cust_number
```

Обработка распределенных транзакций. Это качество *ddb* можно трактовать как возможность выполнения операций обновления распределенной базы данных (*insert, update, delete*), не разрушающее целостность и согласованность данных. Цель достигается применением двухфазного протокола фиксации транзакций (*two-phase commit protocol*), ставшего фактическим стандартом обработки распределенных транзакций. Его применение гарантирует согласованное изменение данных на нескольких узлах в рамках распределенной (глобальной) транзакции.

Независимость от оборудования. Это свойство означает, что в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей – от мэйнфреймов до ПК.

Независимость от операционных систем. Это качество вытекает из предыдущего и означает многообразие операционных систем, управляющих узлами распределенной системы.

Прозрачность сети. Доступ к любым базам данных может осуществляться по сети. Спектр поддерживаемых конкретной СУБД

сетевых протоколов не должен быть ограничением системы с распределенными базами данных. Данное качество формулируется максимально широко – в распределенной системе возможны любые сетевые протоколы.

Независимость от баз данных. Это качество означает, что в распределенной системе могут использоваться СУБД различных производителей, и возможны операции поиска и обновления в базах данных различных моделей и форматов.

Исходя из определения Дэйта, можно рассматривать ddb как слабосвязанную сетевую структуру, узлы которой представляют собой локальные базы данных. Локальные базы данных автономны, независимы и самоопределены; доступ к ним обеспечивается СУБД в общем случае от различных поставщиков. Связи между узлами – это потоки тиражируемых данных. Топология ddb варьируется в широком диапазоне – возможны варианты иерархии, структур типа «звезда» и т. д. В целом топология ddb определяется географией информационной системы и направленностью потоков тиражирования данных.

Механизм распределенных транзакций

Транзакция называется распределенной, если она охватывает операции нескольких взаимодействующих компонент распределенной системы. Каждая из этих компонент может работать с какими-либо СУБД или иными службами. Примером является использование очереди сообщений, либо работа с файлами. При откате транзакции все данные операции отменяются. Для этого необходимо чтобы промежуточная среда поддерживала управление транзакциями, распределенными между несколькими компонентами, при этом компоненты распределенной системы, которые не участвуют в транзакции, не должны работать с какими-либо ресурсами или службами.

Распределенные транзакции являются одним из важнейших элементов поддержания целостности данных в распределенной системе.

Целостность данных

В *ddb* поддержка целостности и согласованности данных ввиду свойств 1–2 представляет собой сложную проблему. Ее решение – синхронное и согласованное изменение данных в нескольких локальных базах данных, составляющих *ddb* – достигается применением протокола двухфазной фиксации транзакций. Если *ddb* одно-

родна – то есть на всех узлах данные хранятся в формате одной базы и на всех узлах функционирует одна и та же СУБД, то используется механизм двухфазной фиксации транзакций данной СУБД. В случае же неоднородности ddb для обеспечения согласованных изменений в нескольких базах данных используют менеджеры распределенных транзакций. Это возможно, если участники обработки распределенной транзакции – СУБД, функционирующие на узлах системы, поддерживают ха-интерфейс, определенный в спецификации *dtp* консорциума *x/open*. В настоящее время ха-интерфейс имеют *ca-openingres*, *informix, Microsoft SQL Server, Oracle, sybase*. Если в *ddb* предусмотрено тиражирование данных, то это сразу предъявляет дополнительные жесткие требования к дисциплине поддержки целостности данных на узлах, куда направлены потоки тиражируемых данных. Проблема в том, что изменения в данных инициируются как локально, на данном узле, так и извне, посредством тиражирования. Неизбежно возникают конфликты по изменениям, которые необходимо отслеживать и разрешать.

Прозрачность расположения

Это качество ddb в реальных продуктах должно поддерживаться соответствующими механизмами. Разработчики СУБД придерживаются различных подходов. Рассмотрим пример из *Oracle*. Допустим, что ddb включает локальную базу данных, которая размещена на узле в Лондоне. Создадим вначале ссылку (*database link*), связав ее с символическим именем (*london_unix*), транслируемым в *ip*-адрес узла в Лондоне.

```
create public database link london.com connect to london_unix using oracle_user_id;
```

Теперь мы можем явно обращаться к базе данных на этом узле, запрашивая, например, в операторе *select* таблицу, хранящуюся в этой базе:

```
select customer.cust_name, order.order_date from customer@london.com, order where customer.cust_number = order.cust_number;
```

Очевидно, однако, что мы написали запрос, зависящий от расположения базы данных, поскольку явно использовали в нем ссылку. Определим *customer* и *customer@london.com* как синонимы:

```
create synonym customer for customer@london.com
```

и в результате можем написать полностью независимый от расположения базы данных запрос:

```
select customer.cust_name, order.order_date from customer, order where customer.cust_number = order.cust_number
```

Задача решается с помощью оператора *sql create synonym,* который позволяет создавать новые имена для существующих таблиц. При этом оказывается возможным обращаться к другим базам данных и к другим компьютерам. Так, запись в СУБД *informix*:

```
create synonym customer for client@central:smith.customer
```

означает, что любое обращение к таблице customer в открытой базе данных будет автоматически переадресовано на компьютер *central* в базу данных *client* к таблице *customer*. Оказывается возможным переместить таблицу из одной базы данных в другую, оставив в первой базе ссылку на ее новое местонахождение, при этом все необходимые действия для доступа к содержимому таблицы будут сделаны автоматически.

С помощью *create synonym* можно определить, например, таблицу структуры *customer*, в которой хранятся строки с записями о клиентах компании, находящихся в Японии:

```
create synonym japan_customer for customer@hq.sales.asia.japan
```

Во многих СУБД задача управления именами объектов *ddb* решается путем использования глобального словаря данных, хранящего информацию о *ddb*: расположение данных, возможности других СУБД (если используются шлюзы), сведения о скорости передачи по сети с различной топологией и т. д.

Обработка распределенных запросов

Выше уже упоминалось это качество *ddb*. Обработка распределенных запросов (*distributed query – dq*) – задача более сложная, нежели обработка локальных, она требует интеллектуального решения с помощью особого компонента – оптимизатора *dq*. Обратимся к базе данных, распределенной по двум узлам сети. Таблица *detail* хранится на одном узле, таблица *supplier* – на другом. Размер первой таблицы – 10 000 строк, размер второй – 100 строк (множество деталей поставляется небольшим числом поставщиков). Допустим, что выполняется запрос:

```
select detail_name, supplier_name, supplier_address from detail, supplier
where detail.supplier_number = supplier.supplier_number.
```

Результирующая таблица представляет собой объединение таблиц *detail* и *supplier*, выполненное по столбцу *detail.supplier_number* (внешний ключ) и *supplier.supplier_number* (первичный ключ). Данный запрос – распределенный, т. к. затрагивает таблицы, принадлежащие различным локальным базам данных. Для его нормального выполнения необходимо иметь обе исходные таблицы на одном узле. Следовательно, одна из таблиц должна быть пере-

дана по сети. Очевидно, что это должна быть таблица меньшего размера, то есть таблица *supplier*. Следовательно, оптимизатор dq-запросов должен учитывать такие параметры, как размер таблиц, статистика распределения данных по узлам, объем данных, передаваемых между узлами, скорость коммуникационных линий, структуры хранения данных, соотношение производительности процессоров на разных узлах и т. д. От интеллекта оптимизатора dq впрямую зависит скорость выполнения распределенных запросов.

Межоперабельность

В контексте *ddb* межоперабельность означает две вещи. Во-первых, это качество, позволяющее обмениваться данными между базами данных различных поставщиков. Как, например, тиражировать данные из базы данных *informix* в *Oracle* и наоборот? Известно, что штатные средства тиражирования в составе данной конкретной СУБД позволяют переносить данные в однородную базу. Так, средствами *ca-ingres/replicator* можно тиражировать данные только из ingres в ingres. Как быть в неоднородной *ddb*? Ответом стало появление продуктов, выполняющих тиражирование между разнородными базами данных.

Во-вторых, это возможность некоторого унифицированного доступа к данным в *ddb* из приложения. Возможны как универсальные решения (стандарт *odbc*), так и специализированные подходы. Очевидный недостаток *odbc* – недоступность для приложения многих полезных механизмов каждой конкретной СУБД, поскольку они могут быть использованы в большинстве случаев только через расширения *sql* в диалекте языка данной СУБД, но в стандарте *odbc* эти расширения не поддерживаются.

Специальные подходы – это, например, использование шлюзов, позволяющее приложениям оперировать базами данных в «чужом» формате так, будто это собственные базы данных. Цель шлюза – организация доступа к унаследованным (*legacy*) базам данных, служит для решения задач согласования форматов баз данных при переходе к какой-либо одной СУБД. Так, если компания долгое время работала на СУБД *ims* и затем решила перейти на *Oracle*, то ей очевидно потребуется шлюз в *ims*. Следовательно, шлюзы можно рассматривать как средство, облегчающее миграцию, но не как универсальное средство межоперабельности в распределенной системе. Универсального решения задачи межоперабельности в этом контексте не существует – все определяется конкретной

ситуацией, историей информационной системы и массой других факторов; *ddb* конструирует архитектор, имеющий в своем арсенале отработанные интеграционные средства, которых на рынке сейчас очень много.

Технология тиражирования данных

Принципиальная характеристика тиражирования данных (*data replication – dr*) заключается в отказе от физического распределения данных. Суть *dr* состоит в том, что любая база данных (как для СУБД, так и для работающих с ней пользователей) всегда является локальной; данные размещаются локально на том узле сети, где они обрабатываются; все транзакции в системе завершаются локально.

Тиражирование данных – это асинхронный перенос изменений объектов исходной базы данных в базы, принадлежащие различным узлам распределенной системы. Функции *dr* выполняет, как правило, специальный модуль СУБД - сервер тиражирования данных, называемый репликатором (так устроены СУБД ca-openingres и *sybase*). В *informix-online dynamic server* репликатор встроен в сервер, в *Oracle 7* для использования dr необходимо приобрести дополнительно к *Oracle7 dbms* опцию *replication option.*

Специфика механизмов *dr* зависит от используемой СУБД. Простейший вариант *dr* – использование «моментальных снимков» (snapshot). Рассмотрим пример из oracle:

```
create snapshot unfilled_orders refrash complete start with to_date ('dd-mon-yy hh23:mi:55') next sysdate + 7 as select customer_name, customer_address, order_date from customer@paris, order@london where customer.cust_name = order.customer_number and order_complete_flag = "n".
```

«Моментальный снимок» в виде горизонтальной проекции объединения таблиц *customer* и *order* будет выполнен в 23:55 и будет повторяться каждые 7 дней. Каждый раз будут выбираться только завершенные заказы.

Реальные схемы тиражирования, разумеется, устроены более сложно. В качестве базиса для тиражирования выступает транзакция к базе данных. В то же время возможен перенос изменений группами транзакций периодически или в некоторый момент времени, что дает возможность исследовать состояние принимающей базы на определенный момент времени.

Детали тиражирования данных полностью скрыты от прикладной программы; ее функционирование никак не зависит от работы репликатора, который целиком находится в ведении администратора базы данных. Следовательно, для переноса программы в рас-

пределенную среду с тиражируемыми данными не требуется ее модификации.

Синхронное обновление *ddb* и *dr*-технология в определенном смысле антиподы. Краеугольный камень первой – синхронное завершение транзакций одновременно на нескольких узлах распределенной системы, то есть синхронная фиксация изменений в *ddb*. Ее «Ахиллесова пята» – жесткие требования к производительности и надежности каналов связи. Если база данных распределена по нескольким территориально удаленным узлам, объединенным медленными и ненадежными каналами связи, а число одновременно работающих пользователей составляет сотни и выше, то вероятность, что распределенная транзакция будет зафиксирована в обозримом временном интервале, становится чрезвычайно малой. В таких условиях (характерных, кстати, для большинства отечественных организаций) обработка распределенных данных практически невозможна.

Сильная сторона *dr*-технологии в том, что она не требует синхронной фиксации изменений. В действительности далеко не во всех задачах требуется обеспечение идентичности БД на различных узлах в любое время. Достаточно поддерживать тождественность данных лишь в определенные критические моменты времени. Можно накапливать изменения в данных в виде транзакций в одном узле и периодически копировать эти изменения на другие узлы.

Налицо преимущества *dr*-технологии. Во-первых, данные всегда расположены там, где они обрабатываются, следовательно, скорость доступа к ним существенно увеличивается. Во-вторых, передача только операций, изменяющих данные (а не всех операций доступа к удаленным данным), и к тому же в асинхронном режиме позволяет значительно уменьшить трафик. В-третьих, со стороны исходной базы для принимающих баз репликатор выступает как процесс, инициированный одним пользователем, в то время как в физически распределенной среде с каждым локальным сервером работают все пользователи распределенной системы, конкурирующие за ресурсы друг с другом. В-четвертых, никакой продолжительный сбой связи не в состоянии нарушить передачу изменений. Дело в том, что тиражирование предполагает буферизацию потока изменений (транзакций); после восстановления связи передача возобновляется с той транзакции, на которой тиражирование было прервано.

Но dr-технология данных не лишена недостатков. Например, невозможно полностью исключить конфликты между двумя версия-

ми одной и той же записи. Он может возникнуть, когда вследствие асинхронности два пользователя на разных узлах исправят одну и ту же запись в тот момент, пока изменения в данных из первой базы данных еще не были перенесены во вторую. При проектировании распределенной среды с использованием *dr*-технологии необходимо предусмотреть конфликтные ситуации и запрограммировать репликатор на какой-либо вариант их разрешения. В этом смысле применение *dr*-технологии – наиболее сильная угроза целостности *ddb*. Таким образом, *dr*-технологию нужно применять крайне осторожно, только для решения задач с жестко ограниченными условиями и по тщательно продуманной схеме, включающей осмысленный алгоритм разрешения конфликтов.

Архитектура «клиент – сервер»

Распределенные системы – это системы «клиент – сервер». На сегодняшний день мы можем говорить, что существует по меньшей мере три модели «клиент – сервер»:

- модель доступа к удаленным данным (*rda*-модель);
- модель сервера базы данных (*dbs*-модель);
- модель сервера приложений (*as*-модель).

Первые две являются двухзвенными и не могут рассматриваться в качестве базовой модели распределенной системы (ниже будет показано, почему). Трехзвенная модель хороша тем, что в ней интерфейс с пользователем полностью независим от компонента обработки данных. Трехзвенной ее можно считать постольку, поскольку явно выделены компонент интерфейса с пользователем и компонент управления данными (и базами данных в том числе), а между ними расположено программное обеспечение промежуточного слоя (*middleware*), выполняющее функции управления транзакциями и коммуникациями, транспортировки запросов, управления именами и множество других. *Middleware* – главный компонент распределенных систем и, в частности, *ddb*-систем. Возможная ошибка, которую мы совершаем на нынешнем этапе, – полное игнорирование *middleware* и использование двухзвенных моделей «клиент – сервер» для реализации распределенных систем.

Существует фундаментальное различие между технологией «*sql*-клиент – *sql*-сервер» и технологией продуктов класса *middleware* (например, *менеджера распределенных транзакций tuxedo system*). В первом случае клиент явным образом запрашивает данные, зная структуру базы данных (имеет место так называемый

data shipping, то есть «поставка данных» клиенту). Клиент передает СУБД *sql*-запрос, в ответ получает данные. Имеет место жесткая связь типа «точка – точка», для реализации которой все СУБД используют закрытый *sql*-канал (например, *Oracle sql*net*). Он строится двумя процессами: *sql/net* на компьютере-клиенте и *sql/net* на компьютере-сервере и порождается по инициативе клиента оператором connect. Канал закрыт в том смысле, что невозможно, например, написать программу, которая будет шифровать *sql*-запросы по специальному алгоритму.

В случае трехзвенной схемы клиент явно запрашивает один из сервисов (предоставляемых прикладным компонентом), передавая ему некоторое сообщение (например) и получает ответ также в виде сообщения. Клиент направляет запрос в информационную шину (которую строит *tuxedo system*), ничего не зная о месте расположения сервиса. Имеет место так называемый function shipping (то есть «поставка функций» клиенту). Важно, что для клиента база данных (в том числе и *ddb*) закрыта слоем сервисов. Он ничего не знает о ее существовании, т. к. все операции над базой данных выполняются внутри сервисов.

Сравним два подхода. В первом случае мы имеем жесткую схему связи «точка – точка» с передачей открытых sql-запросов и данных, исключающую возможность модификации и работающую только в синхронном режиме «запрос – ответ». Во втором случае определен гибкий механизм передачи сообщений между клиентами и серверами, позволяющий организовывать взаимодействие между ними многочисленными способами.

Таким образом, речь идет о двух принципиально разных подходах к построению информационных систем «клиент – сервер». Первый из них устарел и явно уходит в прошлое. Дело в том, что *SQL* (ставший фактическим стандартом общения с реляционными СУБД) был задуман и реализован как декларативный язык запросов, но отнюдь не как средство взаимодействия «клиент – сервер». Только потом он был «притянут за уши» разработчиками СУБД в качестве такого средства. На волне успеха реляционных СУБД в последние годы появилось множество систем быстрой разработки приложений для реляционных баз данных (*visualbasic, powerbuilder, sql windows, jam* и т. д.). Все они опирались на принцип генерации, кода приложения на основе связывания элементов интерфейса с пользователем (форм, меню и т. д.) с таблицами баз данных. И если для быстрого создания несложных приложений

с небольшим числом пользователей этот метод подходит как нельзя лучше, то для создания корпоративных распределенных информационных систем он абсолютно непригоден.

Для этих задач необходимо применение существенно более гибких систем класса *middleware* (*tuxedo system, teknekron*), которые и составляют предмет нашей профессиональной деятельности и базовый инструментарий при реализации больших проектов.

Контрольные вопросы

1. Дайте определение распределенной базы данных.
2. Перечислите 12 свойств идеальной распределенной базы данных.
3. В чем суть технологии тиражирования данных?
4. Опишите три модели «клиент – сервер».
5. Приведите примеры различия между технологией «sql-клиент – sql-сервер» и технологией продуктов класса *middleware*.

Глава 5

Язык структурированных запросов SQL

Приложение-клиент при работе по технологии «клиент – сервер» формирует запрос к серверу, на котором расположена БД, на языке *SQL*, являющемся стандартом для реляционных БД. Удаленный сервер принимает запрос и переадресует его *SQL*-серверу БД. *SQL*-сервер интерпретирует запрос, выполняет его в базе данных, формирует результаты выполнения запроса и выдает его приложению-клиенту.

SQL (*Structured Query Language*) переводится как *структурированный язык запросов.* Он предназначен для работы с реляционными БД. *SQL* сам определяет, какую наиболее эффективную последовательность операций следует использовать для их получения – эти детали не требуется указывать в запросе к базе данных при использовании *SQL*.

По технологии «клиент – сервер» запросы пользовательских персональных компьютеров (клиентов) обрабатываются на специальных серверах баз данных (серверах), а на пользовательские компьютеры возвращаются лишь результаты обработки запроса. При этом, естественно, нужен единый язык общения с сервером, и в качестве такого языка используется *SQL*. Поэтому все современные версии реляционных СУБД (*DB2, Oracle, Ingres, Informix, Sybase*) используют технологию «клиент – сервер» и язык *SQL*.

В настоящее время существует две разновидности языка *SQL*: интерактивный и программный (встроенный).

Интерактивный *SQL* используется для функционирования непосредственно в базе данных. В этом случае введенная пользователем команда выполняется немедленно.

Программный (встроенный) *SQL* предназначен для встраивания запросов в прикладную программу. Существует несколько видов программного *SQL*: статический, динамический и *API*-интерфейсы.

Статический *SQL* – разновидность программного *SQL*, предназначенная для встраивания *SQL*-операторов в текст программы на языке программирования высокого уровня.

Динамический *SQL* – разновидность программного *SQL*, предназначенная для встраивания *SQL*-операторов в текст программы на языке программирования высокого уровня, допускающая дина-

мическое формирование и выполнение запросов во время работы программы.

API-интерфейсы – разновидность программного SQL, основанная на использовании библиотек функций, разработанных для обеспечения связи прикладной программы с СУБД посредством выполнения SQL-запросов.

Существует несколько основных типов операторов SQL.

1. DDL (Data Definition Language) – язык определения данных. Он дает возможность создавать различные объекты БД и переопределять их структуру, например, создавать или удалять таблицы. Примеры операторов:

CREATE
ALTER
DROP
RENAME.

2. DML (Data Manipulation Language) – язык манипуляции данными. Он дает возможность манипулировать данными внутри объектов реляционной БД. Примеры операторов:

INSERT
UPDATE
DELETE.

3. DQL (Data Query Language) – язык запросов к данным. Он используется для построения запросов к реляционным БД. Примеры операторов:

SELECT.

4. DCL (Data Control Language) – язык управления данными. Он позволяет осуществлять контроль над возможностью доступа пользователей к данным внутри БД, а также для назначений пользователям привилегий доступа. Примеры операторов:

ALTER PASSWORD
GRANT REVOKE.

5. DAL (Data Administration Language) – язык администрирования данных. Он дает возможность выполнять аудит и анализ операций внутри БД. Примеры операторов:

START AUDIT
STOP AUDIT.

6. Команды управления транзакциями. Существуют три команды, которые используются для управления транзакциями:

COMMIT – для сохранения изменений;
ROLLBACK – для отмены изменений;
SAVEPOINT – для установки особых точек возврата.

Несмотря на то, что стандарт языка SQL определяется *ANSI* (*американским национальным институтом стандартов*) и *ISO* (*международной организацией по стандартизации*), разработчики коммерческих СУБД без уведомления расширяют SQL ANSI, интегрируют дополнительные возможности в этот язык. Поэтому в настоящее время существует большое количество диалектов SQL.

Типы данных

Понятие «тип данных» в БД полностью адекватно понятию «тип данных» в языках программирования.

Тип данных определяет:

- каков размер области памяти, занимаемой объектом;
- как интерпретируется эта память;
- какие действия можно выполнять с этим объектом.

В SQL используются следующие основные типы данных, форматы которых могут несколько различаться для разных СУБД:

а) *символьные типы данных* содержат буквы, цифры и специальные символы:

- **CHAR** или **CHAR(n)** – символьные строки фиксированной длины. Длина строки определяется параметром **n**. **CHAR** без параметра соответствует **CHAR(1)**. Для хранения таких данных всегда отводится **n** байт вне зависимости от реальной длины строки;
- **VARCHAR(n)** – символьная строка переменной длины. Для хранения данных этого типа отводится число байт, соответствующее реальной длине строки;

б) *целые типы данных* поддерживают только целые числа (дробные части и десятичные точки не допускаются). Над этими типами разрешается выполнять арифметические операции и применять к ним агрегирующие функции (определение максимального, минимального, среднего и суммарного значений столбца реляционной таблицы):

- **INTEGER** или **INT**– целое, для хранения которого отводится, как правило, 4 байта. *(Замечание: число байт, отводимое для хранения того или иного числового типа данных зависит от используемой СУБД и аппаратной платформы.)* Интервал значений от – 2 147 483 647 до + 2 147 483 648;
- **SMALLINT** – короткое целое (2 байта), интервал значений от – 32 767 до +32 768;

в) *вещественные типы данных* описывают числа с дробной частью:

- **FLOAT** и **SMALLFLOAT** – числа с плавающей точкой (для хранения отводится обычно 8 и 4 байта соответственно);

- **DECIMAL(p)** – тип данных, аналогичный **FLOAT**, с числом значащих цифр **p**;
- **DECIMAL(p,n)** – аналогично предыдущему, **p** – общее количество десятичных цифр, **n** – количество цифр после десятичной запятой;

г) *денежные типы данных* описывают, естественно, денежные величины. Если ваша система такого типа данных не поддерживает, то используйте DECIMAL(p,n):

- **MONEY(p,n)** – все аналогично типу **DECIMAL(p,n)**. Вводится только потому, что некоторые СУБД предусматривают для него специальные методы форматирования;

д) дата и время – используются для хранения даты, времени и их комбинаций. Большинство СУБД умеет определять интервал между двумя датами, а также уменьшать или увеличивать дату на определенное количество времени:

- **DATE** – тип данных для хранения даты;
- **TIME** – тип данных для хранения времени;
- **DATETIME** – тип данных для хранения моментов времени (год + месяц + день + часы + минуты + секунды + доли секунд);

е) двоичные типы данных – позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т. д.); Определения этих типов наиболее сильно различаются от системы к системе, часто используются ключевые слова:

- **BINARY;**
- **BYTE;**
- **BLOB.**

Знакомство с языком SQL в MS Access

Рассмотрим создание ИС «Мое имущество». Задача состоит в анализе данных, описанных в предметной области и создании БД в среде *MS Access* в рамках реляционной модели.

Предметная область: необходимо разработать информационную систему «Мое имущество» содержащую сведения о мебели, посуде и др. имущества; их ценах, датах покупки, месте покупки, моделях, страховках и расстановках в комнатах (см. рис. 17).

Логическое проектирование БД

Основная цель проектирования – определение схемы БД:

1) выбор таблиц, которые будут входить в БД;
2) выбор столбцов, принадлежащих каждой таблице;
3) выбор взаимосвязей между таблицами и столбцами.

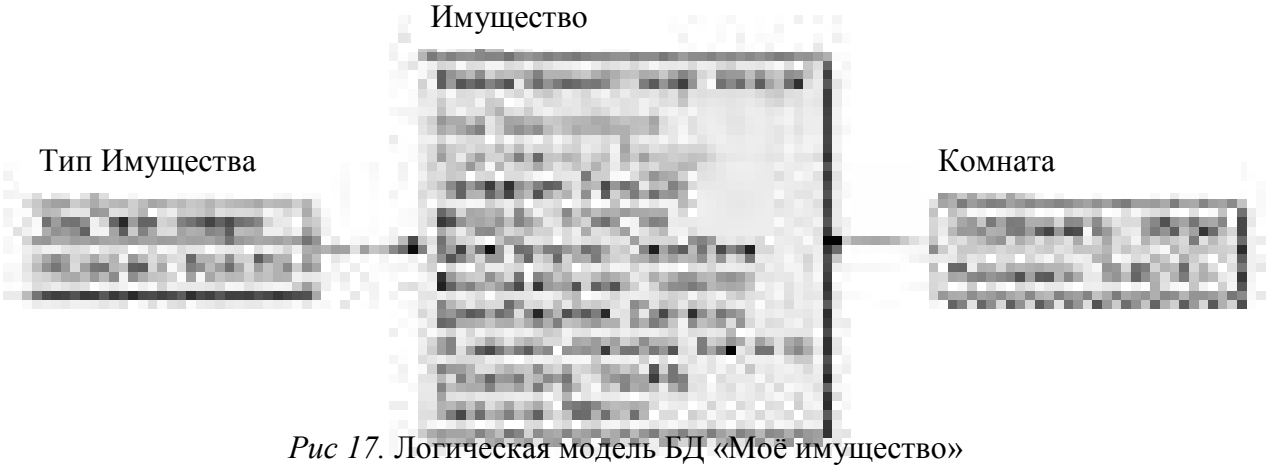
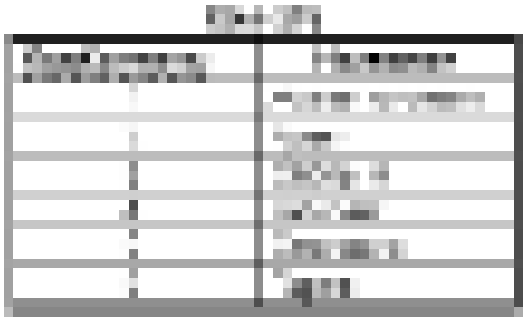


Рис 17. Логическая модель БД «Моё имущество»

Физическое проектирование БД

Запустите СУБД Access. Создайте файл БД с именем «Мое имущество». Создайте структуру таблицы «Комната», заполните и сохраните ее.

Имя поля	Тип данных	Размер поля
КодКомнаты	Числовой	Целое, ключевое поле
Название	Текстовый	15



Создайте структуру таблицы «ТипИмущества», заполните и сохраните ее:

Имя поля	Тип данных	Размер поля
КодТипа	Числовой	Целое, ключевое поле
Название	Текстовый	15

Создайте структуру таблицы «Имущество», заполните и сохраните ее:

Имя поля	Тип данных	Размер поля
ИнвентарныйНомер	Integer (ключевое поле)	
КодТипа	Integer	
КодКомнаты	Integer	
Название	Text	25
Модель	Text	25
ДатаПокупки	Data/Time	
МестоПокупки	Text	20
ЦенаПокупки	Currency	
ОценкаСтоимости	Currency	
Страховка	Yes/No	
Заметки	Memo	

В столбец «КодТипа» введите коды из таблицы «ТипИмущества».
В столбец «КодКомнаты» введите коды из таблицы «Комната».

Создание схемы данных

Выберите «таблицы базы данных»: выберите «Сервис | Схема данных». Добавьте в схему таблицы «Имущество», «Комната», «ТипИмущества». Установите связь между полями «КодТипа» в таблице «ТипИмущества» и «КодТипа» в таблице «Имущество», выберите «Обеспечение целостности данных», выберите «Каскадное обновление связанных полей», выберите «Каскадное удаление связанных записей», нажмите «Создать». Установите связь между полями «КодКомнаты» в таблице «Комната» и «КодКомнаты» в таблице «Имущество», выберите «Обеспечение целостности данных», выберите «Каскадное обновление связанных полей», выберите «Каскадное удаление связанных записей», нажмите «Создать». Сохраните схему данных.

Создание запросов с помощью QBE

Запросы – средства для работы с информацией, хранящейся в таблицах. С их помощью можно сортировать, фильтровать, отбирать данные из различных таблиц, производить вычисления, группировать и модифицировать данные в таблицах. Запросы пишутся на языке *SQL*, позволяющем строить предложения, приближенные к естественному английскому языку и определяющие структуру запроса. Другой метод построения запроса – *QBE* (построение по образцу). Он позволяет с помощью графических средств строить и наглядно видеть структуру запроса.

Выборка названия и цены

Создайте запрос в режиме конструктора на основе таблицы «Имущество», отображающий «Название» и «Цену имущества». Выполните запрос: выберите «Запрос | Запуск», на экране должен быть результат запроса – таблица из двух столбцов «Название» и «Цена», и сохраните его с названием «Проектирование двух столбцов».

Просмотрите код запроса на *SQL*: выберите «Вид | Режим *SQL*».

Пример. Выборка названия и цены из таблицы «Имущество»

```
SELECT Имущество.Название, Имущество.Цена
FROM Имущество
```

Пример. Проектирование вычисляемого столбца

Создайте новый запрос на основе запроса «Проектирование двух столбцов» в режиме конструктора.

Определите вычисляемый столбец: выберите новый столбец в бланке *QBE*, в списке «Поле» введите выражение «Цена» в долларах: «[Цена]/28», выберите «Вывод на экран».

Выполните запрос и сохраните его под именем «Проектирование вычисляемого столбца». Результат запроса – виртуальная таблица из трех столбцов.

Просмотрите код запроса в режиме *SQL*.

Пример: Выборка двух столбцов таблицы «Имущество» и вычисляемый столбец «Цена в долларах»

```
SELECT Имущество.Название, Имущество.Цена, [Цена]/28 AS [Цена в долларах]
FROM Имущество
```

Выборка строк с ценой больше 3000 руб.

Создайте новый запрос в режиме конструктора на основе старого: выберите «Выборка двух столбцов», в поле «Условие отбора столбца» «Цена» введите «>3 000».

Выполните запрос. Результат запроса – виртуальная таблица, строки которой содержат значения в столбце «Цена» больше 3 000 руб.

Просмотрите код запроса в режиме *SQL*.

Пример: Выборка строк таблицы «Имущество» с ценой больше 3000 руб.

```
SELECT Имущество.Название, Имущество.Цена
FROM Имущество
WHERE ((Имущество.ЦенаПокупки>3000))
```

Итоговый запрос, вычисляющий сумму стоимости вещей по каждой комнате

Создайте новый запрос на основе таблиц «Имущество» и «Комната».

Определите столбцы запроса: из таблицы «Комната» – поле «Название», из таблицы «Имущество» – «ЦенаПокупки».

В режиме конструктора выведите строку групповых операций: выберите «Вид», выберите «Групповые операции».

Введите функцию: в столбце «Цена» в строке «Групповые операции» выберите «Sum».

Выполните запрос и сохраните его. Результат – виртуальная таблица с суммами стоимости вещей по каждой комнате.

Просмотрите код запроса в режиме *SQL*.

Пример: Итоговый запрос с суммой вещей по каждой комнате

```
SELECT Комната.Название, Sum(Имущество.Цена) AS Sum_Цена
FROM Комната INNER JOIN Имущество ON Комната.КодКомнаты = Имущество.КодКомнаты
GROUP BY Комната.Название
```

Технология работы с языком SQL в MS Access

Для имитации работы в среде клиент/сервер можно использовать настольную СУБД *MS Access*. В качестве примеров, приведем последовательность действий пользователя в *MS Access* трех версий.

Microsoft Access 2007, 2010

Запустите СУБД *MS Access*. Выполните команду «Пуск» – «Программы» – «MS Access». Запустится данная программа. Далее выполните команду «Системное меню» – «Создать».

Откроется окно задания названия новой базы данных. В поле «Имя файла» наберите «Фамилия» и нажмите «Создать».

Откроется окно новой базы данных. Перейдите на панель «Создание». Для создания запроса (все команды *SQL* в данном случае будут запросами) выполните следующие действия:

1) нажмите кнопку «Конструктор запросов»;

2) в окне конструктора закройте диалоговое окно «Добавление таблицы» без добавления таблиц в окно конструктора;

3) на панели нажмите кнопку «Режим *SQL*»;

4) откроется окно ввода команды *SQL*. Введите инструкции *SQL*. Например,

```
SELECT ИМЯКЛИЕНТА, ГОРОД
FROM CUSTOMER
```

5) при закрытии этого окна с введенной инструкцией *SQL* появится вопрос: «Введите имя запроса». Наберите имя, например, «Запрос1» и нажмите «ОК»;

6) для выполнения созданного запроса необходимо щелкнуть на нем на вкладке «Запросы окна базы данных *MS Access*», либо в режиме *SQL* нажать кнопку «Выполнить».

Конструкции языка SQL

Создание таблиц

Таблицы создаются командой «CREATE TABLE». Эта команда создает пустую таблицу. Команда «CREATE TABLE» определяет имя

таблицы и саму таблицу в виде описания набора имен столбцов, указанных в определенном порядке. Она также определяет типы данных и размеры столбцов. Каждая таблица должна иметь, по крайней мере, один столбец.

Синтаксис команды CREATE TABLE:

CREATE TABLE <table–name>
(<column name> <data type>[(<size>)],
<column name> <data type>[(<size>)], ...);

Значение аргумента *size* зависит от типа данных. Если вы его не указываете, СУБД будет устанавливать значение автоматически.

Пример:

В дальнейшем будем рассматривать базу данных, состоящую из трех таблиц, приведенных ниже.

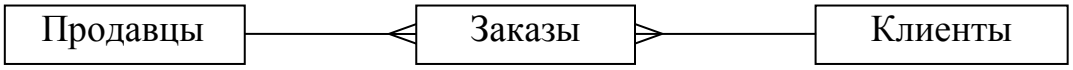


Таблица «Продавцы»:

КодПродавца	ИмяПродавца	Город	Комиссионные
11	Браун	Лондон	0.12
12	Герман	Прага	0.13
14	Смит	Лондон	0.11
17	Алекс	Барселона	0.15
13	Джон	Нью Йорк	0.10

«КодПродавца» – уникальный номер, назначенный каждому продавцу;

«Город» – расположение продавца (город);

«Комиссионные» – комиссионные продавцов в десятичном виде.

Таблица «Клиенты»:

КодКлиента	ИмяКлиента	ГородКлиента	Рейтинг	КодПродавца
21	Хофман	Берлин	100	11
22	Джованни	Венеция	200	13
23	Луи	Гринобль	200	12
24	Даниил	Н.Новгород	300	12
26	Бил	Нью Йорк	100	11
28	Игорь	Гринобль	300	17
27	Янни	Венеция	100	14

«КодКлиента» – уникальный номер, назначенный каждому клиенту;

«ИмяКлиента» – имя клиента;

«ГородКлиента» – расположение клиента (город);

«Рейтинг» – код указывающего уровень предпочтения данного клиента перед другими (рейтинг);

«КодПродавца» – номер продавца назначенного этому клиенту.

Таблица «Заказы»:

КодПриобретения	Суммапокупки	ДатаПокупки	КодКлиента	КодПродавца
1	18.69	2008.10.03	28	17
2	767.19	2008.10.03	21	11
3	1900.10	2008.10.03	27	14
4	5160.45	2008.10.03	23	12
5	1098.16	2008.10.03	28	17
6	1713.23	2008.10.04	22	13
7	100.0	2008.10.05	24	12
8	200.0	2008.10.06	26	11

«КодПриобретения» – уникальный номер, данный каждому приобретению;

«СуммаПокупки» – значение суммы покупки;

«ДатаПокупки» – дата покупки;

«КодКлиента» – номер клиента делающего приобретение;

«КодПродавца» – номер продавца, продающего приобретение.

Для примера рассмотрим, как создать таблицу продавцов:

```
CREATE TABLE Продавцы
(КодПродавца      integer,
ИмяПродавца       char (10),
Город             char (10),
Комиссионные      number)
```

Удаление таблиц – DROP TABLE

Удалить можно только пустую таблицу. Заполненная таблица с находящимися в ней строками не может быть удалена, т. е. таблица перед удалением должна быть очищена. Команда на удаление таблицы имеет следующий вид:

```
DROP TABLE < table name >
Например: DROP TABLE Продавцы
```

Изменение таблицы, после того как она была создана

Для изменения таблиц используется команда «ALTER TABLE». Она может добавлять столбцы к таблице, удалять столбцы, изменять их размеры, а также добавлять или удалять ограничения. Эта команда – не часть стандарта *ANSI*, поэтому в разных системах она имеет разные возможности.

Типичный синтаксис чтобы добавить столбец к таблице:

ALTER TABLE <table name> ADD <column name>
<data type> <size>;

Например:

«ALTER TABLE Продавцы ADD Новый столбец CHAR(7)» – добавляет в таблицу «Продавцы» новый столбец символьного типа длиной 7 байт.

Введение ограничений

Когда вы создаете или изменяете таблицу, вы можете устанавливать ограничения на значения, вводимые в поля. В этом случае *SQL* будет отклонять любые данные, которые нарушают заданные ограничения. Ограничение может устанавливаться как отдельно для каждого конкретного столбца, так и на всю таблицу в целом. Для определения ограничения на столбец Вы вставляете специальное выражение в конец имени столбца после типа данных и перед запятой. Далее показан синтаксис для команды «CREATE TABLE» с учетом ограничений:

CREATE TABLE < table name >
(<column name> <data type> <column constraint>,
<column name> <data type> <column constraint> ...;

С помощью ограничений («CONSTRAINT») можно устанавливать индексы (структуры БД, представляющие собой указатель на конкретную запись в таблице) для полей таблицы, первичный и внешний ключи, определять отношение и целостность данных.

Чтобы предохранить поле таблицы от ввода в него *пустых* (*null*) значений, нужно при создании таблицы указать для соответствующего столбца ключевое выражение «NOT NULL».

Например, очевидно, что первичные ключи никогда не должны быть *пустыми*, поэтому наш пример по созданию таблицы Продавцы может быть модифицирован следующим образом:

CREATE TABLE Продавцы
(КодПродавца integer NOT NULL,
ИмяПродавца char (10),
Город char (10),
Комиссионные number)

Достаточно часто требуется быть уверенным, что все значения, введенные в столбец, отличаются друг от друга. Если вы помещаете ограничение столбца *UNIQUE* в поле при создании таблицы, база данных отклонит любую попытку ввода в это поле для одной из строк, значения, которое уже представлено в другой строке. Например:

```
CREATE TABLE Продавцы
(КодПродавца integer UNIQUE,
ИмяПродавца char (10),
Город char (10),
Комиссионные float)
```

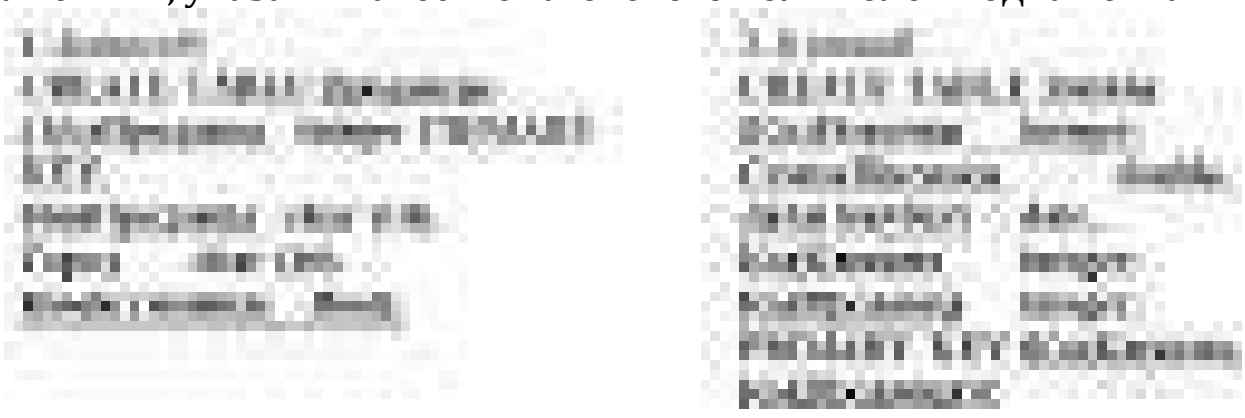
Создание первичных ключей

Первичный ключ создается с помощью выражения *PRIMARY KEY*. Таблица может содержать только один первичный ключ. Существует два способа создания первичного ключа:

1) в виде ограничения на столбец (для простого первичного ключа);

2) в виде ограничения на таблицу (для простого или составного первичного ключа).

Выполните запросы и откройте созданные таблицы в режиме конструктора. Убедитесь, что поле «КодПродавца» стало ключевым, а поля «КодКлиента», «КодПродавца» являются составным ключом. Аналогично измените запрос на создание таблицы «Клиенты», указав в качестве ключевого поля поле «КодКлиента».



Создание внешних ключей (создание схемы данных)

Внешний ключ создается с помощью выражения *FOREIGN KEY*. Назначение *FOREIGN KEY* – это ограничение допустимых значений поля множеством значений первичного ключа, ссылка на который указывается при описании данного ограничения. Для внешнего ключа также существует два способа его назначения.

1–й способ – в виде ограничения на столбец:

```
CREATE TABLE Заказы
(КодПокупки        integer,
СуммаПокупки       double,
ДатаПокупки        date,
КодКлиента         integer REFERENCES Клиенты (КодКлиента),
КодПродавца        integer REFERENCES Продавцы (КодПродавца))
```

2–й способ – в виде ограничения на таблицу:

```
CREATE TABLE Заказы
(КодПокупки integer,
СуммаПокупки       double,
ДатаПокупки        date,
КодКлиента         integer,
КодПродавца        integer,
FOREIGN KEY (КодКлиента) REFERENCES Клиенты (КодКлиента),
FOREIGN KEY (КодПродавца) REFERENCES Продавцы
(КодПродавца))
```

Два примера, приведенные выше, определяют поля «КодКлиента» и «КодПродавца» как внешние ключи таблицы «Заказы», ссылающиеся на первичные ключи «КодКлиента» и «КодПродавца» таблиц «Клиенты» и «Продавцы» соответственно.

Если в родительской таблице у первичного ключа уже указано ограничение *PRIMARY KEY*, то при задании ограничения *FOREIGN KEY*, накладываемого на таблицу или на столбцы, можно в скобках не указывать список столбцов первичного ключа.

Выполните запрос, предварительно удалив ранее связанные таблицы. Откройте схему данных и убедитесь, что связи *1 : M* созданы.

Ввод значений в таблицы

Все строки в *SQL* вводятся с использованием команды модификации *INSERT*. Предложение *INSERT* имеет один из следующих форматов:

```
INSERT INTO <table name | view name>
[(column [,column] ...)]
VALUES ( <value> [,<value>] ... );
```

или

```
INSERT INTO <table name | view name>
[(column [,column] ...)]
подзапрос
```

Так, например, чтобы ввести строку в таблицу «Продавцов», вы можете использовать следующее условие:

```
INSERT INTO Продавцы
VALUES (11, 'Браун', 'Лондон', 0.12)
```

Вы можете также указывать столбцы, куда хотите вставить значение имени. Это позволяет вставлять имена в любом порядке. Например:

```
INSERT INTO Продавцы (ИмяПродавца, Комиссионные, КодПродавца, Город)
VALUES ('Браун',0.12, 11, 'Лондон')
```

Вы можете также использовать команду *INSERT* чтобы получать или выбирать значения из одной таблицы и помещать их в другую, чтобы использовать их вместе с запросом. Чтобы сделать это, вы просто заменяете предложение *VALUES* (из предыдущего примера) на соответствующий запрос:

```
INSERT INTO Лондон
SELECT * FROM Продавцы
WHERE Город = 'Лондон'
```

С помощью новых запросов заполните таблицы «Продавцы», «Клиенты», «Заказы». Выполните запросы и просмотрите полученные таблицы в режиме таблицы.

Удаление строк

Вы можете удалять строки из таблицы командой модификации – *DELETE*. Она может удалять только введённые строки, а не индивидуальные значения полей. Предложение *DELETE* имеет следующий формат:

```
DELETE FROM <table name | view name>
[WHERE search–condition]
```

Например, чтобы удалить все содержание таблицы «Продавцы», вы можете ввести следующее условие:

```
DELETE FROM Продавцы
```

Чтобы определить какие строки будут удалены, вы должны использовать условие. Например, чтобы удалить продавца «Джон» из таблицы, вы можете ввести:

```
DELETE FROM Продавцы
WHERE КодПродавца = 13
```

Удалите из таблицы «Клиенты» клиента Билли. Выполните запрос и посмотрите на результат, открыв таблицу «Клиенты» в режиме таблицы.

Изменение значения поля

Это выполняется командой «UPDATE». Эта команда содержит предложение *UPDATE*, в которой указано имя используемой таблицы и предложение *SET* которое указывает на изменение, которое нужно сделать для определенного столбца. Предложение *UPDATE* имеет два формата. Первый из них:

```
UPDATE <table name | view name>
SET column = expression [, column = expression] ...
[WHERE search–condition]
```

где expression – это столбец | выражение | константа | переменная.

Второй вариант:

```
UPDATE <table name>
SET column = expression, ...
[ FROM table–list ]
[ WHERE search–condition ]
```

Например, чтобы изменить рейтинг всех клиентов на 200, вы можете ввести:

```
UPDATE Клиенты
SET Рейтинг = 200
```

Для модифицирования определенных строк надо использовать условие, как и в *DELETE*. Вот как например можно выполнить изменение, одинаковое для всех клиентов продавца Браун (имеющего «КодПродавца»=11):

```
UPDATE Клиенты
SET Рейтинг = 200
WHERE КодПродавца = 11
```

Предложение *SET* может назначать любое число столбцов, отделяемых запятыми. Все указанные назначения могут быть сделаны для любой табличной строки, но только для одной в каждый момент времени. Например:

```
UPDATE Продавцы
SET ИмяПродавца = 'Гибсон', Город = 'Бостон', Комиссионные = 0.10
WHERE КодПродавца = 14
```

Вы можете использовать арифметические выражения в предложении *SET* команды «UPDATE». Например:

```
UPDATE Продавцы
SET Комиссионные = Комиссионные * 2
```

Извлечение информации из таблицы (простейшие запросы)

Для извлечения информации из таблиц применяется команда «SELECT». В самой простой форме, в команде указываются столбцы, которые вы бы хотели видеть, и имя таблицы из которой они будут

извлекаться. Например, вы могли бы вывести таблицу «Продавцов» без столбца «КодПродавца», введя следующую команду:

```
SELECT ИмяПродавца, Город, Комиссионные
FROM Продавцы
```

Чтобы вывести все столбцы таблицы «Продавцы»:

```
SELECT *
FROM Продавцы
```

Если вы не хотите чтобы какие-либо значения дублировались в списке вывода, надо воспользоваться аргументом *DISTINCT* (отличие), который обеспечивает вас способом устранять двойные значения из вашего предложения *SELECT*. *DISTINCT* может указываться только один раз в данном предложении *SELECT*. Пример:

```
SELECT DISTINCT КодПродавца
FROM Заказы
```

SQL дает возможность вам устанавливать критерии, определяющие выбор строк для вывода. Предложение *WHERE* команды *SELECT* позволяет устанавливать условия, значения которых могут быть верными или неверными для любой строки таблицы. Команда извлекает только те строки из таблицы, для которых такое утверждение верно. Например, вы хотите видеть имена и комиссионные всех продавцов в Лондоне. Вы можете ввести такую команду:

```
SELECT ИмяПродавца, Комиссионные
FROM Продавцы
WHERE Город = ‘Лондон’
```

Работа с несколькими таблицами
Объединение таблиц

Объединение является одним из видов операций в реляционных базах данных. Оно определяет связи между несколькими таблицами и выводит информацию из них в терминах этих связей. В виду того, что в разных таблицах могут быть столбцы с одинаковыми именами, для идентификации нужного столбца используется префикс имени таблицы.

При объединении таблицы, представленные списком в предложении *FROM* запроса, отделяются запятыми. Условие запроса может ссылаться на любой столбец любой связанной таблицы и, следовательно, может использоваться для связи между ними. Обычно сравниваются значения в столбцах различных таблиц, чтобы определить, удовлетворяет ли *WHERE* установленному условию. Самый

простой способ объединения – это декартово произведение, его можно выполнить следующим образом:

```
SELECT Клиенты.*, Продавцы.*
FROM Продавцы, Клиенты
```

Но в данном случае получится бесполезная таблица с множеством ненужной информации. Если из декартова произведения убрать ненужные строки и столбцы, то можно получить нужные данные. Это реализуется с помощью *WHERE* фразы.

Например, если вы хотите увидеть все комбинации продавцов и клиентов для данного города, то вы должны ввести следующее:

```
SELECT Клиенты.ИмяКлиента, Продавцы.ИмяПродавца,
Продавцы.Город
FROM Продавцы, Клиенты
WHERE Продавцы.Город = Клиенты.Город
```

В объединении *SQL* исследует каждую комбинацию строк двух или более возможных таблиц и проверяет эти комбинации по их условиям. Чаще всего используется объединение через целостность.

Пример: показ имен всех клиентов соответствующих продавцам, которые их обслуживают

```
SELECT Клиенты.ИмяКлиента, Продавцы.ИмяПродавца
FROM Клиенты, Продавцы
WHERE Продавцы.КодПродавца = Клиенты.КодПродавца
```

Объединения, которые используют условия, основанные на равенствах, называются *объединениями по равенству*, это наиболее общий вид объединения, но имеются и другие.

Объединение таблицы с собой

Для объединения таблицы с собой вы можете сделать каждую строку таблицы одновременно и комбинацией ее с собой и комбинацией с каждой другой строкой таблицы. Затем вы оцениваете каждую комбинацию в терминах условия. Это объединение – такое же, как и любое другое объединение между двумя таблицами, за исключением того, что в данном случае обе таблицы идентичны.

Когда вы объединяете таблицу с собой, все повторяемые имена столбца, заполняются префиксами имени таблицы. Чтобы ссылаться к этим столбцам внутри запроса, вы должны иметь два различных имени для этой таблицы. Вы можете сделать это с помощью определения временных имен, называемых ***псевдонимами.*** Они определяются через пробел после имени таблицы в предложении *FROM* запроса.

Пример: нахождение всех пар клиентов, имеющих один и тот же рейтинг.

```
SELECT a.ИмяКлиента, b.ИмяКлиента, a.Рейтинг
FROM Клиенты a, Клиенты b
WHERE a.Рейтинг = b.Рейтинг
```

В данном случае *SQL* ведет себя так, как если бы он соединял две таблицы, называемые *a* и *b*. Псевдоним существует только на время выполнения команды. В вышеприведенном примере есть избыточные строки, два значения для каждой комбинации. Значение *A* в псевдониме сначала выбирается в комбинации со значением *B* во втором псевдониме, а затем значение *A* во втором псевдониме выбирается в комбинации со значением *B* в первом псевдониме, при этом каждая строка сравнивается сама с собой. Простой способ избежать этого заключается в том, чтобы налагать ограничения на два значения так, чтобы один мог быть меньше другого или предшествовал ему в алфавитном порядке. В этом случае те же самые значения в обратном порядке выбраны снова не будут.

Пример:

```
SELECT a.ИмяКлиента, b.ИмяКлиента, a.Рейтинг
FROM Клиенты a, Клиенты b
WHERE a.Рейтинг = b.Рейтинг
AND a.ИмяКлиента < b.ИмяКлиента
```

В данном примере первая комбинация выводится, если удовлетворяет второму условию, но та же комбинация в обратном порядке уже не будет ему удовлетворять, и наоборот. Вы можете использовать псевдонимы и тогда, когда хотите создать альтернативные имена для таблиц в команде. Вы можете использовать любое число псевдонимов для одной таблицы в запросе. Вам не всегда обязательно использовать каждый псевдоним или таблицу, которые упомянуты в предложении *FROM* запроса или в предложении *SELECT*. Вы можете также создать объединение, которое включает и различные таблицы и псевдонимы одиночной таблицы.

Простые вложенные подзапросы

С помощью *SQL* вы можете вкладывать запросы внутрь друг друга. Обычно внутренний запрос генерирует значение, которое проверяется во внешнем запросе, определяющем верно оно или нет.

Пример: предположим, что мы знаем имя продавца (Смит), но не знаем значение его поля «КодПродавца» и хотим извлечь все заказы из таблицы «Заказов». Сделать это можно следующим образом:

```
SELECT *
FROM Заказы
WHERE КодПродавца =
( SELECT КодПродавца FROM Продавцы
WHERE ИмяПродавца = 'Смит' )
```

Сначала выполняется внутренний запрос, а затем его результаты используются для формирования внешнего запроса («КодПродавца» сравнивается с результатом подзапроса). В данном случае подзапрос должен выбрать один и только один столбец, а тип данных этого столбца должен совпадать с типом того значения, с которым он будет сравниваться в предикате. Вы можете в некоторых случаях использовать *DISTINCT*, чтобы подзапрос генерировал одиночное значение.

Пример: предположим, что мы хотим найти все заказы для тех продавцов, которые обслуживают Хофман («КодКлиента»=21).

```
SELECT * FROM Заказы
WHERE КодПродавца =
( SELECT DISTINCT КодПродавца
FROM Заказы
WHERE КодКлиента = 21 )
```

В данном случае подзапрос выведет только одно значение 11, но в принципе может быть много подобных значений, и тогда *DISTINCT* выберет только одно.

Один тип функций, который автоматически может производить одиночное значение для любого числа строк, – агрегатная функция, ее и можно использовать в подзапросе.

Для того чтобы увидеть все заказы, имеющие сумму приобретений выше средней на 4-е октября, нужно создать два запроса:

1. Запрос «СРЕДНЯЯ СУММА ПОКУПКИ», вычисляющий среднюю сумму покупок:

```
SELECT Avg(Заказы.СуммаПокупки) AS СуммаПокупки
FROM Заказы
```

2. На основе предыдущего запроса создать запрос, сравнивающий суммы покупок на 4 октября, со средней суммой всех приобретений.

```
SELECT Заказы.ДатаПокупки, Заказы.СуммаПокупки
FROM Заказы, [Средняя сумма покупки]
WHERE (((Заказы.ДатаПокупки)=’10.04.2008’) AND
((Заказы.СуммаПокупки)>[Средняя сумма покупки]![СуммаПокупки]))
```

Вы можете использовать подзапросы, которые производят любое число строк, если вы используете специальный оператор *IN*. Когда вы используете *IN* с подзапросом, *SQL* просто формирует набор значений для *IN* из вывода подзапроса.

Пример: показать все заказы для продавцов, находящихся в Лондоне.

```
SELECT * FROM Заказы
WHERE КодПродавца IN
(SELECT КодПродавца
FROM Продавцы
WHERE Город = 'Лондон')
```

Действия основных команд языка *SQL*

Команда *SQL*	Выполняемое действие
CREATE TABLE	Создание таблицы
DROP TABLE	Удаление таблицы
ALTER TABLE	Изменение таблицы
NOT NULL	Ограничение на использование пустых значений
UNIQUE	Ограничение на использование одинаковых значений
PRIMARY KEY	Создание первичного ключа
FOREIGN KEY	Создание внешнего ключа
INSERT	Добавление строк в таблицу
DELETE	Удаление строк из таблицы
UPDATE	Изменение значений в таблице
SELECT	Извлечение информации из таблиц

Контрольные вопросы

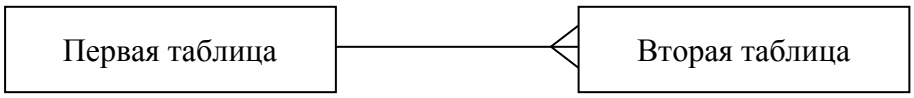
1. Перечислите основные типы операторов *SQL*.
2. Напишите пример *SQL* запроса на создание таблицы.
3. Напишите пример *SQL* запроса на выборку строк/столбцов таблицы.
4. Напишите пример *SQL* запроса на объединение таблицы.
5. Напишите пример *SQL* запроса на удаление таблицы.
6. Напишите пример SQL запроса на создание первичных ключей в таблице.

Задания к расчетной работе

Общее задание

1. Напишите запросы на создание таблиц, приведенных в задании, учитывая что:

а) первая таблица связана со второй связью «один ко многим»;



б) значения первого поля в каждой таблице должно быть уникальными и не содержать значений *NULL*;

в) первые поля в таблицах являются первичными ключами;

г) последнее поле во второй таблице является внешним ключом.

2. Напишите запросы на добавление приведенных в задании данных в созданные таблицы.

Варианты заданий

Вариант 1. Строительная организация состоит из нескольких подразделений. В базе данных должны содержаться сведения о:

а) подразделениях строительной организации (подразделение представляется номером подразделения, названием, специализацией);

б) сотрудниках (данными о служащем являются его табельный номер, ФИО, год рождения, должность, подразделение в котором он работает).

Подразделение

№_подразделения	Название	Количество_ПК	Специализация
15	Плановый отдел	12	Составление планов работ
21	Сметно-договорной отдел	15	Выполнение и проверка сметных расчетов
23	Цех железобетонных изделий	1	Изготовление ЖБ изделий
48	СМУ–1	2	Производство СМР
52	СМУ–2	3	Производство СМР

Сотрудник

Таб_№	ФИО	Год_рождения	Должность	Подразделение
5383	Сидоров Иван Михайлович	1958	Экономист	15
6852	Иванов Петр Сергеевич	1960	Начальник цеха	23
6578	Морозова Анастасия Андреевна	1975	Экономист	15
4852	Бирюков Леонид Ильич	1980	Начальник отдела	15
6548	Волков Дмитрий Александрович	1955	Прораб	48
3216	Зайцев Кирилл Викторович	1965	Мастер	48
6536	Касатонова Юлия Олеговна	1983	Сметчик	21

1. Напишите запрос, который увеличивает «Количество_ПК» во всех подразделениях на 5 шт.

2. Напишите запрос, переводящий сотрудников «СМУ–1» в «СМУ–2».

3. Напишите запрос, который выводит «№_Подразделения», «Название» и «Специализацию» из таблицы «Подразделение».

4. Напишите запрос, который вывел бы список всех сотрудников «Планового отдела».

5. Напишите запрос, который вывел бы таблицу «Сотрудник» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Сотрудник» список подразделений, в которых работают сотрудники. Подразделения не должны повторяться.

7. Напишите запрос, считающий средний возраст сотрудников.

8. Напишите запрос на создание списка, состоящего из ФИО сотрудника и названия его подразделения для всех подразделений, в которых количество компьютеров меньше 10.

9. Напишите запрос на удаление всех сотрудников, работающих в подразделении № 23.

Вариант 2. Строительная организация ведет работы на нескольких объектах. В базе данных должны содержаться сведения о:

а) заказчиках (данными о заказчике являются номер заказчика, его наименование, адрес, количество сотрудников);

б) объектах (данными об объекте являются его номер, наименование, сметная стоимость работ, планируемая дата окончания работ, заказчик).

Заказчик

№_заказчика	Наименование	Адрес	Количество_сотрудников
15	ОАО «Парус»	ул. Тимирязева, 30	30
21	ЗАО «Берег»	пр. Ленина, 45	52
23	ООО «Корвет»	пр. Гагарина, 28	108
48	ОАО «Консул»	пер. Союзный, 4	24
52	ГОУ ВПО ННГАСУ	ул. Ильинская, 65	1850

Объект

№_объекта	Наименование	Сметная_стоимость	Дата_окончания	Заказчик
265	Поликлиника	150	01.04.2010	21
546	Школа	82	01.05.2009	21
845	Жилой дом по ул. Тимирязева	136	01.08.2011	52
828	Котельная № 1	25	01.06.2009	15
145	Жилой дом по ул. Гоголя	140	01.03.2009	52
548	Жилой дом по пр. Гагарина	250	01.11.2012	23
753	Котельная № 1	23	01.07.2010	48

1. Напишите запрос, который сокращает «Количество_сотрудников» у всех заказчиков на 5.
2. Напишите запрос, передающий объекты от заказчика ОАО «Консул» к ООО «Корвет».
3. Напишите запрос, который выводит «Наименование», «Адрес» и «Количество_сотрудников» из таблицы «Заказчик».
4. Напишите запрос, который вывел бы список всех объектов заказчика ЗАО «Берег».
5. Напишите запрос, который вывел бы таблицу «Объект» со столбцами в обратном порядке.
6. Напишите запрос, извлекающий из таблицы «Объект» список заказчиков этих объектов. Заказчики не должны повторяться.
7. Напишите запрос, выводящий наименование и сметную стоимость самого дорогого объекта.

8. Напишите запрос на создание списка, состоящего из «Наименования объекта» и «Наименования его заказчика» для всех заказчиков, у которых работает более 100 человек.

9. Напишите запрос на удаление всех объектов заказчика № 21.

Вариант 3. У строительной организации несколько складов. В базе данных должны содержаться сведения о:

а) складах (данными о складе являются его номер, адрес, вид хранящихся строительных материалов, расстояние до областного центра);

б) строительных материалах (данными о материале являются его номер, наименование, единица измерения, остаток, склад).

Склад

№_склада	Адрес	Вид_материалов	Расстояние
1	д. Крутово	сыпучие	5
2	пос. Веканово	отделочные	10
3	пос. Заскочиха	отделочные	15
4	д. Орлово	отделочные	8
5	д. Комарово	кирпич	12

Стройматериал

№_материала	Наименование	Ед_изм	Остаток	Склад
5466	Цемент	кг	680	1
7898	Краска	кг	350	4
1232	Шпатлевка	кг	260	2
4565	Кирпич глиняный	м3	68	5
7535	Песок	т	250	1
1595	Известь	т	9	3
8542	Кирпич силикатный	м3	120	5

1. Напишите запрос, который уменьшает остаток всех строительных материалов на 10 %.

2. Напишите запрос, переводящий строительные материалы, находящиеся на складе в пос. Веканово, на склад в пос. Заскочиха.

3. Напишите запрос, который выводит «Адрес», «Вид_материалов» и «Расстояние» из таблицы «Склад».

4. Напишите запрос, который вывел бы список всех стройматериалов, находящихся на складе в д. Комарово.

5. Напишите запрос, который вывел бы таблицу «Стройматериал» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Стройматериал» список складов, где хранятся эти стройматериалы. Склады не должны повторяться.

7. Напишите запрос, выводящий наименование и номер склада стройматериала с самым большим остатком.

8. Напишите запрос на создание списка, состоящего из «Наименования стройматериала» и «Адреса склада», где он хранится для всех складов, расположенных на расстоянии не менее 12 км от областного центра.

9. Напишите запрос на удаление всех стройматериалов, хранящихся на складе № 1.

Вариант 4. В управлении механизации несколько типов машин (бульдозеры, автокраны и т. д.). В базе данных должны содержаться сведения о:

а) типах машин (данными о типе являются его номер, название, дальность перегона (км), назначение);

б) машинах (данными о машине являются инвентарный номер, название, местонахождение базы, количество, тип).

Тип_машины

№_Типа	Название	Дальность_перегона	Вид_работ
1	Землеройные	30	Земляные работы
2	Грузовые	1000	Перевозка грузов
3	Бетоносмесительные	200	Перевозка растворов
4	Специализированные	100	Строительные работы
5	Сваебойные	30	Забивка свай

Машина

№_Машины	Название	База	Количество	Тип
234	Бульдозер	Москва	2	1
345	Трактор	Калуга	3	4
654	Грейдер	Обнинск	1	1
642	Автобетоносмеситель	Новгород	3	3
854	Грузовик	Москва	4	2
321	Тягач	Новгород	1	2
643	Одноковшовый погрузчик	Калуга	2	1

1. Напишите запрос, который увеличивает «Дальность_перегона» у всех типов машин на 10 км.

2. Напишите запрос, меняющий тип специализированных машин на грузовые.

3. Напишите запрос, который выводит «Название», «Дальность_перегона» и «Вид_работ» для всех типов машин.

4. Напишите запрос, который вывел бы список всех машин, имеющих тип «Землеройные».

5. Напишите запрос, который вывел бы таблицу «Машина» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Машина» список типов этих машин. Типы не должны повторяться.

7. Напишите запрос, выводящий название машины и базу, имеющую самое большое количество.

8. Напишите запрос на создание списка, состоящего из «Названия машины» и «Названия ее типа» для всех типов, дальность перегона которых не менее 200 км.

9. Напишите запрос на удаление всех машин с типом 1.

Вариант 5. В строительной организации несколько бригад. В базе данных должны содержаться сведения о:

а) бригадах (данными о бригаде являются код бригады, фамилия бригадира, число работников, вид выполняемых работ);

б) работниках (данными о работнике являются его табельный номер, ФИО, год рождения, разряд, код бригады).

Бригада

Код_бригады	Бригадир	Число_работников	Вид_работ
1	Сидоров	10	Малярные
2	Петров	9	Штукатурные
3	Иванов	5	Сантехнические
4	Кузнецов	8	Каменные
5	Смирнов	10	Электротехнические

Работник

Таб_№	ФИО	Год_рождения	Разряд	Бригада
3258	Морозова Анастасия Андреевна	1960	3	1
4342	Павлинов Алексей Владимирович	1975	4	2
5428	Пигалов Максим Александрович	1985	3	1
5321	Рязанцев Максим Вячеславович	1961	5	5
4418	Смирнов Сергей Александрович	1963	5	3
3021	Чистова Ксения Марковна	1950	4	5
5152	Чуважова Анастасия Сергеевна	1981	3	2

1. Напишите запрос, который увеличивает «Число_работников» во всех бригадах на 2 человека.
2. Напишите запрос, переводящий работников из бригады № 2 в бригаду № 1.
3. Напишите запрос, который выводит «Бригадир», «Число_работников», «Вид_работ» из таблицы «Бригада».
4. Напишите запрос, который вывел бы список всех работников бригады, выполняющей каменные работы.
5. Напишите запрос, который вывел бы таблицу «Работник» со столбцами в обратном порядке.
6. Напишите запрос, извлекающий из таблицы «Работник» список кодов бригад. Коды не должны повторяться.
7. Напишите запрос, считающий средний возраст работников.
8. Напишите запрос на создание списка, состоящего из «ФИО_работника» и «Вида_работ» бригады для всех бригад, в которых число работников больше 8.
9. Напишите запрос на удаление всех работников бригады № 1.

Вариант 6. При изготовлении строительных изделий требуется несколько видов ресурсов (цемент, гравий, металлопрокат и т. п.). В базе данных должны содержаться сведения о:

а) ресурсах (данными о ресурсе являются код ресурса, наименование, единица измерения, необходимый запас);

б) потребностях ресурсов при изготовлении изделий (данными о потребности являются код, наименование, дата поставки необходимого ресурса, код данного ресурса).

Ресурс

Код	Наименование	Ед_изм	Запас
2346	Цемент	т	120
2364	Гравий	т	245
5678	Песок	т	330
6753	Металлический профиль	м	560
3783	Арматура	т	300

Потребность

Код	Изделие	Дата	Расход	Ресурс
9087	Лестничный марш	02.10.2009	1,5	2346
7453	Лестничный марш	03.10.2009	2	5678
8754	Плита перекрытия	15.10.2009	1,9	2346
6543	Стеновая панель	01.11.2009	1,8	5678
8765	Металлическая ферма	10.11.2009	9	6753
7645	Стеновая панель	01.11.2009	0,8	2346
9876	Плита перекрытия	10.10.2009	2,4	3783

1. Напишите запрос, который сокращает «Запас» всех ресурсов на 5 %.
2. Напишите запрос, меняющий дату поставки ресурса в таблице «Потребность» на 05.11.2009 для тех записей, где используется «Песок».
3. Напишите запрос, который выводит «Наименование», «Ед_изм» и «Запас» из таблицы «Ресурс».
4. Напишите запрос, который вывел бы список всех записи из таблицы «Потребность», где используется цемент.
5. Напишите запрос, который вывел бы таблицу «Потребность» со столбцами в обратном порядке.
6. Напишите запрос, извлекающий из таблицы «Потребность» список ресурсов. Ресурсы не должны повторяться.
7. Напишите запрос, выводящий наименование изделия и дату поставки соответствующего ресурса, расход которого максимален.
8. Напишите запрос на создание списка, состоящего из Наименования изделия и Наименования соответствующего ресурса для ресурсов, которые измеряются в тоннах.
9. Напишите запрос на удаление записей из таблицы «Потребность», где используется ресурс № 2346.

Вариант 7. На заводе железобетонных конструкций существует несколько технологических линий изготовления ж/б конструкций и изделий. В базе данных должны содержаться сведения о:

а) технологических линиях (данными о линии являются номер, название, число рабочих мест, дата очередного профилактического обслуживания);

б) железобетонных изделиях (данными об изделии являются код, наименование, цена изделия (тыс. руб.), план выпуска в смену (шт.), технологическая линия).

Линия

№_линии	Название	Число_мест	Дата
1	Плиты	4	01.03.2010
2	Стеновые панели	5	15.03.2010
3	Лестничные марши	3	01.12.2009
4	Фермы	7	01.11.2009
5	Спец. изделия	5	30.11.2009

Изделие

Код	Наименование	Цена	План	Линия
9087	Плита перекрытия 6 м	80	20	1
7453	Плита перекрытия 12 м	120	25	1
8754	Ж/б ферма 12 м	240	10	4
6543	Стеновая панель	75	25	2
8765	Лестничный марш	60	18	3
7645	Ж/б ферма 6 м	150	13	4
9876	Фундаментный блок	50	30	5

1. Напишите запрос, который сокращает «Число_мест» на всех технологических линиях на 1.
2. Напишите запрос, увеличивающий стоимость изделий, которые изготавливаются на технологической линии «Плиты» на 1 тыс. руб.
3. Напишите запрос, который выводит «Название», «Число_мест» и «Дата» из таблицы «Линия».
4. Напишите запрос, который вывел бы список всех изделий, изготавливаемых на технологической линии «Фермы».

5. Напишите запрос, который вывел бы таблицу «Изделие» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Изделие» список номеров технологических линий. Номера не должны повторяться.

7. Напишите запрос, выводящий наименование и цену самого дорогого изделия.

8. Напишите запрос на создание списка, состоящего из «Наименования изделия» и «Названия технологической линии» для всех линий, число рабочих мест которых не менее 4.

9. Напишите запрос на удаление всех изделий, изготавливаемых на технологической линии № 1.

Вариант 8. Процесс возведения здания можно расчленить на множество бригадных процессов и определить набор ресурсов, используемых в каждом процессе. База данных должна содержать сведения о:

а) бригадных процессах (данными о процессе являются код, наименование, трудоемкость (человеко-час), дата начала работ);

б) выполнение работ (данными о выполнении работ являются его код, наименование бригады и количество человек в ней, минимальный разряд членов бригады, код выполняемого процесса).

Процесс

Код	Наименование	Трудоемкость	Дата
1	Укладка плит перекрытия	16	01.03.2009
2	Возведение стен	18	05.03.2009
3	Установка лестничных маршей	3	05.03.2009
4	Установка оконных и дверных блоков	12	01.04.2009
5	Прокладка инженерных систем	25	15.04.2009

Выполнение

Код	Бригада	Количество	Разряд	Процесс
234	Монтажники	5	3	1
345	Крановщик	1	5	1
654	Сантехники	2	3	5
642	Отделочники	4	4	4
854	Монтажники	4	3	2
321	Крановщик	1	5	2
643	Монтажники	3	4	3

1. Напишите запрос, который сокращает «Трудоемкость» всех процессов на 1 человеко-час.

2. Напишите запрос, повышающий минимальный разряд членов бригады, участвующей в процессе «Возведение стен».

3. Напишите запрос, который выводит «Наименование», «Трудоемкость» и «Дата» из таблицы «Процесс».

4. Напишите запрос, который вывел бы все записи из таблицы «Выполнение», которые связаны с процессом « Укладка плит перекрытия».

5. Напишите запрос, который вывел бы таблицу «Выполнение» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Выполнение» список кодов процессов. Процессы не должны повторяться.

7. Напишите запрос, выводящий название бригады и минимальный разряд ее рабочих для бригады с самым большим количеством человек.

8. Напишите запрос на создание списка, состоящего из «Наименования бригады» и «Наименования процесса» для всех процессов, трудоемкость которых не менее 18 человеко-часов.

9. Напишите запрос на удаление всех записей в таблице «Выполнение», связанных с процессом № 1.

Вариант 9. Строительное подразделение ведет работу на нескольких объектах. В базе данных должны содержаться сведения:

а) об объектах (данными об объекте являются его номер, наименование, сметная стоимость работ (млн руб.), процент выполнения работ);

б) о поставках ресурсов (данными о поставке ресурсов являются код поставки, наименование ресурса, единица измерения, количество, объект).

Объект

№_объекта	Наименование	Стоимость	Выполнение
1	Поликлиника	150	90
2	Школа	82	80
3	Жилой дом по ул. Тимирязева	136	50
4	Котельная № 1	25	20
5	Жилой дом по ул. Гоголя	140	30

Поставка_ресурсов

Код	Ресурс	Ед_изм	Количество	Объект
3258	Цемент	кг	680	1
4342	Краска	кг	350	4
5428	Шпатлевка	кг	260	2
5321	Кирпич глиняный	м3	68	5
4418	Песок	т	250	1
3021	Известь	т	9	3
5152	Кирпич силикатный	м3	120	5

1. Напишите запрос, который увеличивает выполнение по всем объектам на 1 %.

2. Напишите запрос, переводящий ресурсы, предназначенные для объекта «Жилой дом по ул. Тимирязева» на объект «Школа».

3. Напишите запрос, который выводит «Наименование», «Стоимость», «Выполнение» из таблицы «Объект».

4. Напишите запрос, который вывел бы список всех поставок ресурсов для объекта «Поликлиника».

5. Напишите запрос, который вывел бы таблицу «Поставка_ресурсов» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Поставка_ресурсов» список объектов. Объекты не должны повторяться.

7. Напишите запрос, выводящий наименование объекта и процент выполнения, для объекта, имеющего самую высокую сметную стоимость.

8. Напишите запрос на создание списка, состоящего из «Наименования ресурса» и «Наименования объекта», где он используется для всех объектов, выполнение по которым не более 60 %.

9. Напишите запрос на удаление всех записей из таблицы «Поставка_ресурсов», предназначенных для объекта № 1.

Вариант 10. Завод-изготовитель поставляет нескольким получателям строительные изделия и конструкции. В базе данных должны содержаться сведения о:

а) получателях (данными о получателе являются его код, наименование, адрес, удаленность от завода);

б) поставках (данными о поставке являются ее шифр, наименование изделия, единица измерения, цена ед. измерения, получатель).

Получатель

Код	Наименование	Адрес	Удаленность
5241	ООО «Гранит»	ул. Ильинская, 30	102
3820	ЗАО «Протект»	ул. Должанская, 1	50
2450	ЧП Кулик	ул. Архангельская, 28	92
3054	ОАО «Маяк»	пр. Ленина, 49	72
1568	АО «Строй-НН»	пр. Гагарина, 37	28

Поставка

Шифр	Изделие	Ед_изм	Цена	Получатель
1238	Кирпич	шт.	8	2450
1237	Плитка	$м^2$	420	1568
1247	ГВЛ	шт.	400	3820
7421	Ламинат	$м^2$	999	3054
1241	Стеклопакет	шт.	14999	2450
5421	Гвозди	кг	70	2450
3248	Шифер	шт.	230	3820

1. Напишите запрос, который увеличивает «Цену» всех поставок на 10 руб.
2. Напишите запрос, передающий поставки от «ЗАО "Протект"» в «ООО "Гранит"».
3. Напишите запрос, который выводит «Наименование», «Удаленность» и «Адрес» из таблицы «Получатель».
4. Напишите запрос, который вывел бы всю информацию о поставках «ЧП Кулик».
5. Напишите запрос, который вывел бы таблицу «Поставка» со столбцами в обратном порядке.
6. Напишите запрос, извлекающий из таблицы «Поставка» список получателей. Получатели не должны повторяться.
7. Напишите запрос, считающий среднюю цену поставок.
8. Напишите запрос на создание списка, состоящего из «Изделие», и «Наименование его получателя» для всех получателей, которые расположены далее 70 км от завода.
9. Напишите запрос на удаление всех поставок получателя с кодом 2450.

Вариант 11. Строительная организация получает строительные изделия и материалы от нескольких поставщиков. В базе данных должны содержаться сведения о:

а) поставщиках (данными о поставщике являются его номер, индекс, наименование, адрес);

б) получаемых изделиях (данными об изделии являются его шифр, наименование, единица измерения, количество, поставщик).

Поставщик

№_поставщика	Наименование	Адрес	Количество_ сотрудников
5241	ООО «Гранит»	ул. Ильинская, 30	30
3820	ЗАО «Протект»	ул. Должанская, 1	52
2450	ЧП Кулик	ул. Архангельская, 28	108
3054	ОАО «Маяк»	пр. Ленина, 49	24
1568	АО «Строй-НН»	пр. Гагарина, 37	185

Изделие

Шифр	Наименование	Ед_изм	Количество	Поставщик
1238	Шлакоблок	шт.	50	1568
1237	Цемент	т	18	3820
1247	Стеновая панель	шт.	50	3820
7421	Труба	м	320	3054
1241	Дверной блок	шт.	260	2450
5421	Плита перекрытия	шт.	82	3820
3248	Оконный блок	шт.	240	2450

1. Напишите запрос, который сокращает «Количество_сотрудников» у всех поставщиков на 2.

2. Напишите запрос, увеличивающий количество изделий поставщика ЧП Кулик на 5 шт.

3. Напишите запрос, который выводит «Наименование», «Адрес» и «Количество_сотрудников» из таблицы «Поставщик».

4. Напишите запрос, который вывел бы список всех изделий, поставляемых ЗАО «Протект».

5. Напишите запрос, который вывел бы таблицу «Изделие» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Изделие» список номеров поставщиков. Номера не должны повторяться.

7. Напишите запрос, выводящий наименование и адрес поставщика, где работает минимальное количество сотрудников.

8. Напишите запрос на создание списка, состоящего из «Наименования изделия» и «Поставщик» для всех поставщиков, количество сотрудников которых не менее 100.

9. Напишите запрос на удаление всех изделий, поставляемых поставщиком № 2450.

Вариант 12. Строительные изделия и конструкции поставляются с нескольких заводов-изготовителей. В базе данных должны содержаться сведения:

а) об изделиях (данными об изделии являются его код, наименование, единица измерения, цена за единицу измерения (руб.));

б) поставках (данными о поставке являются ее шифр, наименование получателя, дата поставки, количество изделий в единице измерения, код изделия).

Изделие

Код	Наименование	Ед_изм	Цена
1247	Кирпич	1000 шт.	30000
7421	Плитка	$м^2$	420
1241	ГВЛ	шт.	400
5421	Ламинат	$м^2$	1000
3248	Стеклопакет	шт.	14000

Поставка

Шифр	Получатель	Дата	Количество	Изделие
5241	ЗАО «Берег»	01.12.2008	20	1247
3820	ЗАО «Протект»	01.12.2008	550	1241
2450	ЧП Кулик	03.12.2008	50	7421
3054	ОАО «Маяк»	03.12.2008	300	1241
1568	АО «Строй-НН»	04.12.2008	60	7421
1248	ОАО «Парус»	04.12.2008	70	5421
7452	ЗАО «Берег»	05.12.2008	230	3248

1. Напишите запрос, который увеличивает «Цену» всех изделий на 10 руб.

2. Напишите запрос, меняющий кирпич на плитку в соответствующих поставках.

3. Напишите запрос, который выводит «Наименование», «Ед_изм» и «Цену» из таблицы «Изделие».

4. Напишите запрос, который вывел бы всю информацию о поставках плитки.

5. Напишите запрос, который вывел бы таблицу «Поставка» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Поставка» список кодов изделий. Коды не должны повторяться.

7. Напишите запрос, считающий среднюю цену изделий.

8. Напишите запрос на создание списка, состоящего из названия получателя и наименования изделия для всех изделий, цена которых не более 500 руб.

9. Напишите запрос на удаление всех поставок изделия с кодом 7421.

Вариант 13. В строительном вузе преподаватели проводят занятия. В базе данных должны содержаться сведения о:

а) преподавателях (данными о преподавателе являются табельный номер, ФИО, должность, оклад);

б) занятиях (данными о занятии являются номер занятия, название дисциплины, почасовая ставка оплаты (руб.), день недели, преподаватель).

Преподаватель

Таб_№	ФИО	Должность	Оклад
5241	Сидоров В.В.	профессор	20000
3820	Петров В.П.	доцент	12000
2450	Лисин А.Н.	доцент	12000
3054	Киров Д.О.	ст. преподаватель	10000
1568	Королева О.М.	ассистент	8000

Занятие

№_занятия	Дисциплина	Ставка	День_недели	Преподаватель
1238	Теоретическая механика	115	понедельник	2450
1237	Сопротивление материалов	135	понедельник	5241
1247	Математика	115	среда	3820
7421	Делопроизводство	75	среда	1568
1241	Теоретическая механика	115	вторник	2450
5421	Строительные материалы	95	вторник	3054
3248	Сопротивление материалов	135	среда	5241

1. Напишите запрос, который увеличивает почасовую ставку оплаты на 5 руб. для всех занятий.

2. Напишите запрос, передающий занятия от преподавателя В.П. Петрова В.В. Сидорову.

3. Напишите запрос, который выводит «ФИО», «Должность» и «Оклад» из таблицы «Преподаватель».

4. Напишите запрос, который вывел бы информацию обо всех занятиях В.В. Сидорова.

5. Напишите запрос, который вывел бы таблицу «Занятие» со столбцами в обратном порядке.

6. Напишите запрос, извлекающий из таблицы «Занятие» список номеров преподавателей. Номера не должны повторяться.

7. Напишите запрос, считающий средний оклад преподавателей.

8. Напишите запрос на создание списка, состоящего из «Дисциплина» и «День недели», когда проводится занятие для всех преподавателей, оклад которых не менее 12 000 руб.

9. Напишите запрос на удаление всех занятий преподавателя с табельным номером 2450.

Вариант 14. В строительной компании ведется учет рабочего времени. Необходимо обеспечить начисление заработной платы. В базе данных должны содержать сведения о:

а) рабочих (данными о служащем являются табельный номер, ФИО, должность, тариф (руб./час);

б) карточки учета рабочего времени (они содержат номер карточки, дату, день недели, количество отработанных часов, номер рабочего).

Рабочий

Таб_№	ФИО	Должность	Тариф
3258	Морозова Анастасия Андреевна	штукатур	100
4342	Павлинов Алексей Владимирович	монтажник	120
5428	Пигалов Максим Александрович	монтажник	120
5321	Рязанцев Максим Вячеславович	крановщик	250
4418	Смирнов Сергей Александрович	каменщик	150

Карточка

№_карточки	Дата	День_недели	Часы	Рабочий
1238	01.12.2008	понедельник	8	3258
1237	01.12.2008	понедельник	8	4342
1247	03.12.2008	среда	8	5321
7421	03.12.2008	среда	4	4342
1241	04.12.2008	четверг	4	5321
5421	04.12.2008	четверг	8	4342
3248	05.12.2008	пятница	7	4448

1. Напишите запрос, который увеличивает тариф на 10 руб./час для всех рабочих.
2. Напишите запрос, меняющий записи в таблице «Карточка», где указан в качестве исполнителя рабочий Павлинов Алексей Владимирович, на рабочего Пигалова Максима Александровича.
3. Напишите запрос, который выводит «ФИО», «Должность» и «Тариф» из таблицы «Рабочий».
4. Напишите запрос, который вывел бы все записи из таблицы «Карточка», связанные с Рязанцевым Максимом Вячеславовичем.
5. Напишите запрос, который вывел бы таблицу «Карточка» со столбцами в обратном порядке.
6. Напишите запрос, извлекающий из таблицы «Карточка» список табельных номеров рабочих. Номера не должны повторяться.
7. Напишите запрос, считающий средний тариф рабочих.
8. Напишите запрос на создание списка, состоящего из «Дата» и «День недели», когда работали рабочие с окладом не более 150 руб.
9. Напишите запрос на удаление всех записей из таблицы «Карточка», связанных с рабочим, имеющим табельный номер 3258.

Вариант 15. Поставщик продает товары различных производителей. Необходимо обеспечить работу системы обработки заказов. В базе данных должны содержаться сведения о:

а) товарах (данными о товаре являются код, наименование, единица измерения, цена единицы);

б) заказах (данными о заказе являются код, дата заказа, стоимость заказа, вид оплаты, код товара,).

Товар

Код	Наименование	Ед_изм	Цена
3258	Плита перекрытия	шт.	60000
4342	Стеновая панель	шт.	60000
5428	Лестничный марш	шт.	50000
5321	Ферма металлическая	шт.	350000
4418	Оконный блок	шт.	9000

Заказ

Код	Дата	Количество	Вид_оплаты	Товар
5466	01.12.2008	50	нал.	3258
7898	01.12.2008	18	безнал.	5321
1232	03.12.2008	50	безнал.	3258
4565	03.12.2008	320	нал.	4418
7535	04.12.2008	260	безнал.	4418
1595	04.12.2008	82	нал.	5428
8542	05.12.2008	240	безнал.	4342

1. Напишите запрос, который увеличивает цену всех товаров на 1 000 руб.
2. Напишите запрос, увеличивающий количество лестничных маршей во всех заказах, где они встречаются, на 5 шт.
3. Напишите запрос, который выводит «Наименование», «Ед_изм» и «Цену» из таблицы «Товар».
4. Напишите запрос, который вывел бы список всех заказов «Оконных блоков».
5. Напишите запрос, который вывел бы таблицу «Заказ» со столбцами в обратном порядке.
6. Напишите запрос, извлекающий из таблицы «Заказ» список кодов товаров. Коды не должны повторяться.
7. Напишите запрос, выводящий дату и количество заказа для самого дешевого товара.
8. Напишите запрос на создание списка, состоящего из «Дата заказа» и «Наименование товара» для всех товаров не дороже 60 000 руб.
9. Напишите запрос на удаление всех заказов товаров с кодом 3258.

Заключение

Постоянное увеличение производительности автоматизированных рабочих мест обработки информации и рост возможностей современных программно-аппаратных средств информационного обмена позволяют по-новому взглянуть на проблему реализации современных информационных систем. Реальным становится совместное использование данных, одновременно хранящихся на нескольких ЭВМ. Указанная возможность обеспечивается ростом популярности интернета и технологии *WWW*, к которым в последнее время проявляется повышенный интерес со стороны разработчиков информационных систем. Данные тенденции заключаются в использовании интернета в качестве универсального транспорта при создании распределенных систем удаленного мониторинга, контроля и управления объектами различного назначения.

Распределенные системы используются во всех секторах экономики, поэтому в последнее время много внимания уделяется технологиям разработки распределенных приложений, охватывающих несколько независимых компьютеров. В течение последних десяти лет было создано большое число технологий и стандартов, использование которых должно было помочь разработчикам в создании распределенных приложений масштаба предприятия. Рассмотренные в пособии основные понятия распределенных систем, их место в современной экономике, основные составляющие распределенных баз данных и методы их организации позволят студентам приобрести опыт в проектировании и ведении баз данных распределенных систем. Представлены основные подходы к распределенной обработке информации в вычислительных сетях и организации распределенных приложений. Проведен обзор основных подходов к организации распределенных вычислительных систем: методы удаленных вызовов процедур, многослойные клиент-серверные системы, многоагентные системы, технологии одноранговых вычислений. Рассмотрены сервис-ориентированный подход к построению распределенных вычислительных систем, технологии создания распределенных систем *RMI, CORBA и DCOM*. Проведен анализ особенности объединения информационных систем с помощью технологий *CORBA* и *WWW*. Приводятся составляющие

моделей взаимодействия клиентов и серверов в рассматриваемых технологиях. Даются сравнительные характеристики систем с точки зрения функциональной полноты и возможности работы на различных программно-аппаратных платформах. Приводятся выводы относительно влияния систем создания распределенных объектов на развитие интернета.

Технологии *Java* и *CORBA* прекрасно дополняют друг друга в качестве универсального, мощного средства для решения проблемы объединения систем, основанных на технологии *WWW* с подобными распределенными информационными системами. Технологии *Java* и *CORBA* лучше всего подходят для интегрирования систем, основанных на технологии *WWW*, с крупными информационными системами. Использование этих технологий уже сейчас позволяет поднять технологию *WWW* на новый уровень – уровень распределенных систем.

Список рекомендуемой литературы

1. Басс Л., Клементс П., Кацман Р. Архитектура программного обеспечения на практике. 2-е изд. Л. : Питер, 2010.

2. Горев А., Ахаян Р., Макашарипов С. Эффективная работа с СУБД. СПб. : Питер, 2014.

3. Диго С.М. Базы данных: проектирование и использование : учебник. М. : Финансы и статистика, 2014.

4. Дунаев В.В. Базы данных. Язык SQL для студента. СПб. : БХВ–Петербург, 2013.

5. Когаловский М.Р. Перспективные технологии информационных систем. М. : ДМК Пресс : Компания АйТи, 2010.

6. Корнелюк В.К., Веккер З.Е., Зиновьев Н.Б. ACCESS 97. М. : СОЛОН, 2011.

7. Таненбаум Э.М. Распределенные системы. Принципы и парадигмы. СПб. : ПИТЕР, 2010.

8. Фуфаев Э.В, Фуфаев Д.Э. Разработка и эксплуатация удаленных баз данных : учебник для студ. сред. проф. образования. М. : Академия, 2012.

9. Харитонова И., Рудикова Л. Microsoft Office Access 2007. СПб. : БХВ–Петербург, 2012.

10. Цимбал А.А., Аншина М. Технологии создания распределенных систем. СПб. : Питер, 2012.

11. Чернышова Г.Ю., Пчелинцева Е.Г. Информационные технологии : учеб. пособие. Саратов, 2010.

Учебное издание

Распределенные системы

Учебное пособие

Авторы-составители:
Демина Анна Васильевна
Алексенцева Ольга Николаевна

Редактор Ю.В. Семенова
Компьютерная верстка – А.А. Угланов

Подписано в печать 26.04.2018 г. Формат 60 × 84 $^{1}/_{16}$.
Уч.-изд. л. 4,9. Усл. печ. л. 6,3.
Тираж 50 экз. Заказ 63.

Саратовский социально-экономический институт (филиал)
РЭУ им. Г.В. Плеханова.
410003, г. Саратов, ул. Радищева, 89.