**Содержание**

[Тема 1. Информационные системы 4](#_Toc82244639)

[1.1. Основные понятия 4](#_Toc82244640)

[1.2. Факторы, влияющие на развитие корпоративных информационных систем 4](#_Toc82244641)

[1.2.1. Развитие методик управления предприятием 4](#_Toc82244642)

[1.2.2. Развитие общих возможностей и производительности компьютерных систем 5](#_Toc82244643)

[1.2.3. Развитие подходов к технической и программной реализации элементов информационной системы 5](#_Toc82244644)

[1.3. Основные составляющие корпоративных информационных систем 5](#_Toc82244645)

[1.4. Соотношение между составляющими информационной системы 6](#_Toc82244646)

[1.5. Классификация информационных систем 7](#_Toc82244647)

[1.5.1. Классификация по масштабу 7](#_Toc82244648)

[1.5.2. Классификация по сфере применения 8](#_Toc82244649)

[1.5.3. Классификация по способу организации 9](#_Toc82244650)

[1.6. Области применения и примеры реализации информационных систем 13](#_Toc82244651)

[1.6.1. Бухгалтерский учет 13](#_Toc82244652)

[1.6.2. Управление финансовыми потоками 14](#_Toc82244653)

[1.6.3. Управление складом, ассортиментом, закупками 14](#_Toc82244654)

[1.6.4. Управление производственным процессом 14](#_Toc82244655)

[1.6.5. Управление маркетингом 14](#_Toc82244656)

[1.6.6. Документооборот 14](#_Toc82244657)

[1.6.7. Оперативное управление предприятием 15](#_Toc82244658)

[1.6.8. Предоставление информации о фирме 15](#_Toc82244659)

[Тема 2. Системы документооборота. Lotus Notes 16](#_Toc82244660)

[2.1. Электронная почта 16](#_Toc82244661)

[2.2. Базы данных 16](#_Toc82244662)

[2.2.1. База данных Notes 17](#_Toc82244663)

[2.2.2. Формы и документы 17](#_Toc82244664)

[2.2.3. Представления и папки 17](#_Toc82244665)

[2.2.4. Коллективные и локальные базы данных 17](#_Toc82244666)

[Тема 3. Проектирование корпоративных информационных систем 19](#_Toc82244667)

[3.1. Общие сведения об управлении проектами 19](#_Toc82244668)

[3.1.1. Понятие проекта 19](#_Toc82244669)

[3.1.2. Классификация проектов 21](#_Toc82244670)

[3.1.3. Основные фазы проектирования информационной системы 21](#_Toc82244671)

[Тема 4. Типы баз данных 23](#_Toc82244672)

[Тема 5. Реляционная модель данных 23](#_Toc82244673)

[5.1. Учебная БД 25](#_Toc82244674)

[5.2. Создание таблиц 26](#_Toc82244675)

[5.3. Типы данных в БД 27](#_Toc82244676)

[Тема 6. Выборка данных из БД 27](#_Toc82244677)

[Тема 7. Выборка данных. Соединение 30](#_Toc82244678)

[Тема 8: Выборка. Подзапрос 32](#_Toc82244679)

[Тема 9. Обновление строк в таблице 35](#_Toc82244680)

[Тема 10. Удаление строк 36](#_Toc82244681)

[Тема 11. Вставка строк 36](#_Toc82244682)

[Тема 12. Управление транзакциями 37](#_Toc82244683)

[12.1. Три проблемы, связанные с параллельностью транзакций 38](#_Toc82244684)

[12.1.1. Проблема утраченного обновления 38](#_Toc82244685)

[12.1.2. Проблема зависимости от неподтвержденных обновлений 39](#_Toc82244686)

[12.1.3. Проблема противоречивости 39](#_Toc82244687)

[12.2. Решение проблем параллелизма 40](#_Toc82244688)

[12.2.1. Управление с помощью захватов (блокировок) 40](#_Toc82244689)

[12.2.2. Управление на основе многоверсионных объектов с метками времени 42](#_Toc82244690)

[Тема 13. Объектно-ориентированное программирование 44](#_Toc82244691)

[13.1. Основы языка Object Pascal 44](#_Toc82244692)

[13.1.1. Структура программы в Object Pascal 44](#_Toc82244693)

[13.1.2. Типы данных 46](#_Toc82244694)

[13.1.3. Простые и структурированные операторы языка 48](#_Toc82244695)

[13.1.4. Процедуры и функции 49](#_Toc82244696)

[13.1.5. Модули Object Pascal 50](#_Toc82244697)

[13.2. Объектно-ориентированное программирование 51](#_Toc82244698)

[13.2.1. Основные понятия 51](#_Toc82244699)

[13.2.2. Основные концепции 52](#_Toc82244700)

[13.2.3. Поля, свойства и методы 54](#_Toc82244701)

[Тема 14. Среда разработки Delphi 55](#_Toc82244702)

[14.1. Быстрая разработка приложений 55](#_Toc82244703)

[14.2. Главное окно Delphi 55](#_Toc82244704)

[14.2.1. Палитра компонентов 55](#_Toc82244705)

[14.2.2. Инспектор объектов 56](#_Toc82244706)

[14.3. Управление проектами 56](#_Toc82244707)

[14.4. Основные компоненты для построения простых приложений 56](#_Toc82244708)

[14.4.1. Формы 56](#_Toc82244709)

[14.4.2. Стандартные элементы интерфейса 57](#_Toc82244710)

[Тема 15. Доступ к БД в архитектуре фирмы Borland 58](#_Toc82244711)

[15.1. Доступ к данным с использованием BDE 58](#_Toc82244712)

[15.2. Компоненты доступа к данным 59](#_Toc82244713)

[Тема 16. Обработка исключительных ситуаций 64](#_Toc82244714)

# Тема 1. Информационные системы

## 1.1. Основные понятия

Под *информационной системой* обычно понимается прикладная программная подсистема, ориентированная на сбор, хранение, поиск и обработку информации.

Большинство информационных систем работает в режиме диалога с пользователем.

Типовые программные компоненты, входящие в состав информационной системы, включают:

* диалоговый ввод-вывод;
* логику диалога;
* прикладную логику обработки данных;
* логику управления данными;
* операции манипулирования файлами и (или) базами данных.

*Корпоративной информационной системой* (КИС) мы будем называть совокупность специализированного программного обеспечения и вычислительной аппаратной платформы, на которой установлено и настроено программное обеспечение.

## 1.2. Факторы, влияющие на развитие корпоративных инфор­ма­ционных систем

В последнее время все больше руководителей начинают отчетливо осознавать важность построения на предприятии корпоративной информационной системы как необходимого инструментария для успешного управления бизнесом в современных условиях.

Можно выделить три наиболее важных фактора, существенно влияющих на развитие корпоративных информационных систем:

* развитие методик управления предприятием;
* развитие общих возможностей и производительности компьютерных систем;
* развитие подходов к технической и программной реализации элементов инфор­мационной системы.

Рассмотрим эти факторы более подробно.

### 1.2.1. Развитие методик управления предприятием

Теория управления предприятием представляет собой довольно обширный пред­мет для изучения и совершенствования. Это обусловлено широким спектром посто­янных изменений ситуации на мировом рынке.

Все время растущий уровень конку­ренции вынуждает руководителей компаний искать новые методы сохранения своего присутствия на рынке и поддержания рентабельности своей деятельности.

Современная информационная система должна отвечать всем ново­введениям в теории и практике менеджмента.

Несомненно, это самый главный фак­тор, так как построение продвинутой в техни­ческом отношении системы, которая не отвечает требованиям по функциональности, не имеет смысла.

### 1.2.2. Развитие общих возможностей и производительности компью­терных систем

Прогресс в области наращивания мощности и производительности компьютерных систем, развитие сетевых технологий и систем передачи данных, широкие возмож­ности интеграции компьютерной техники с самым разнообразным оборудованием позволяют постоянно наращивать производительность информационных систем и их функциональность.

### 1.2.3. Развитие подходов к технической и программной реализации элементов инфор­мационной системы

Параллельно с развитием аппаратной части информационных систем на протяже­нии последних лет происходит постоянный поиск новых, более удобных и универ­сальных, методов программно-технологической реализации информационных си­стем.

Можно выделить три наиболее существенных новшества, оказавших колос­сальное влияние на развитие информационных систем в последние годы:

* *новый подход к программированию: с* начала 90-х годов объектно-ориентиро­ванное программирование фактически вытеснило модульное; до настоящего времени непрерывно совершенствуются методы построения объектных моде­лей. Благодаря внедрению объектно-ориентированных технологий программи­рования существенно сокращаются сроки разработки сложных информацион­ных систем, упрощаются их поддержка и развитие;
* благодаря *развитию сетевых технологий* локальные информационные систе­мы повсеместно вытесняются клиент-серверными и многоуровневыми реализациями;
* *развитие сети Интернет* принесло большие возможности работы с удаленны­ми подразделениями, открыло широкие перспективы электронной коммерции, обслуживания покупателей через Интернет и многое другое. Более того, опре­деленные преимущества дает использование Интернет - технологий в интрасе-тях предприятия (так называемые интранет - технологии).

## 1.3. Основные составляющие корпоративных информа­цион­ных систем

В составе корпоративных информационных систем можно выделить две относи­тельно независимых составляющих:

* *компьютерную инфраструктуру* организации, представляющую собой совокуп­ность сетевой, телекоммуникационной, программной, информационной и орга­низационной инфраструктур. Данная составляющая обычно называется *кор­поративной сетью.*
* *взаимосвязанные функциональные подсистемы,* обеспечивающие решение задач организации и достижение ее целей.

Первая составляющая отражает системно-техническую, структурную сторону лю­бой информационной системы.

По сути, это основа для интеграции функциональ­ных подсистем, полностью определяющая свойства информационной системы, определяющие ее успешную эксплуатацию.

Требования к компьютерной инфра­структуре едины и стандартизованы, а методы ее построения хорошо известны и многократно проверены на практике.

Вторая составляющая корпоративной информационной системы полностью относится к прикладной области и сильно зависит от специфики задач и целей предприятия.

Данная составляющая полностью базируется на компьютерной инфраструктуре пред­приятия и определяет прикладную функциональность информационной системы.

Требования к функциональным подсистемам сложны и зачастую противоречивы, так как выдвигаются специалистами из различных прикладных областей. Однако, в ко­нечном счете, именно эта составляющая более важна для функционирования органи­зации, так как для нее, собственно, и строится компьютерная инфраструктура.

## 1.4. Соотношение между составляющими информационной системы

Взаимосвязи между двумя указанными составляющими информационной систе­мы достаточно сложны. С одной стороны, эти две составляющие в определенном смысле независимы.

*Например, организация сети и протоколы, используемые для обмена данными между компьютерами, абсолютно не зависят от того, какие методы и программы планируется использовать на предприятии для организации бух­галтерского учета.*

С другой стороны, указанные составляющие в определенном смысле все же зави­сят друг от друга. Функциональные подсистемы в принципе не могут существо­вать без компьютерной инфраструктуры. В то же время компьютерная инфраструк­тура сама по себе достаточно ограничена, поскольку не обладает необходимой функциональностью.

*Например, невозможно эксплуатировать распределенную информаци­онную систему при отсутствии сетевой инфраструктуры. Хотя, имея развитую инфраструктуру, можно предоставить сотрудникам организации ряд полезных общесистемных служб (например, электронную почту и доступ в Интернет), упро­щающих работу и делающих ее более эффективной (в частности, за счет использо­вания более развитых средств связи).*

Таким образом, разработку информационной системы целесообразно начинать с построения компьютерной инфраструктуры (корпоративной сети) как наиболее важной составляющей, опирающейся на апробированные промышленные тех­нологии и гарантированно реализуемой в разумные сроки в силу высокой степе­ни определенности, как в постановке задачи, так и в предлагаемых решениях.

Корпоративная сеть создается на многие годы вперед, капитальные затраты на ее разработку и внедрение настолько велики, что практически исключают возмож­ность полной или частичной переделки существующей сети.

Функциональные подсистемы, в отличие от корпоративной сети, изменчивы по своей природе, так как в предметной области деятельности организации постоянно происходят более или менее существенные изменения.

Функциональность инфор­мационных систем сильно зависит от организационно-управленческой структуры организации, ее функциональности, распределения функций, принятых в органи­зации финансовых технологий и схем, существующей технологии документообо­рота и множества других факторов.

Разработку и внедрение функциональных подсистем можно выполнять постепен­но. Например, сначала на наиболее важных и ответственных участках выполнять разработки, обеспечивающие прикладную функциональность системы (внедрять системы финансового учета, управления кадрами и т.п.), а затем распространять прикладные программные системы и на другие, первоначально менее значимые области управления предприятием.

## 1.5. Классификация информационных систем

Информационные системы классифицируются по разным признакам. Рассмотрим наиболее часто используемые способы классификации.

### 1.5.1. Классификация по масштабу

По масштабу информационные системы подразделяются на следующие группы (рис. 1.1):

* одиночные;
* групповые;
* корпоративные.

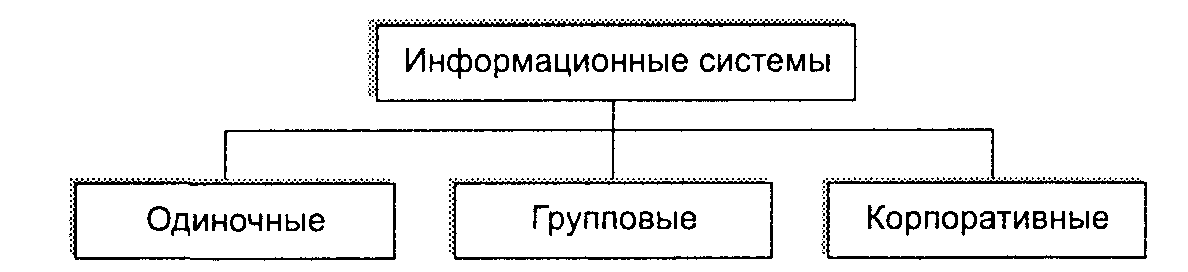


Рис. 1.1. Деление информационных систем по масштабу

**Одиночные информационные системы**

*Одиночные информационные системы* реализуются, как правило, на автономном персональном компьютере (сеть не используется).

Такая система может содержать несколько простых приложений, связанных общим информационным фондом, и рассчитана на работу одного пользователя или группы пользователей, разделяющих по времени одно рабочее место.

Подобные приложения создаются с помощью так называемых *настольных* или *локальных* систем управления базами данных (СУБД). Среди локальных СУБД наиболее известными являются Clarion, Clipper, FoxPro, Paradox, dBase и Microsoft Access.

**Групповые информационные системы**

*Групповые информационные системы* ориентированы на коллективное использова­ние информации членами рабочей группы и чаще всего строятся на базе локальной вычислительной сети.

При разработке таких приложений используются серверы баз данных (называемые также SQL-серверами) для рабочих групп.

Существует доволь­но большое количество различных SQL-серверов, как коммерческих, так и свобод­но распространяемых. Среди них наиболее известны такие серверы баз данных, как Oracle, DB2, Microsoft SQL Server, InterBase, Sybase, Informix.

**Корпоративные информационные системы**

*Корпоративные информационные системы* являются развитием систем для рабо­чих групп, они ориентированы на крупные компании и могут поддерживать тер­риториально разнесенные узлы или сети.

В основном они имеют иерархическую структуру из нескольких уровней. Для таких систем характерна архитектура кли­ент-сервер со специализацией серверов или же многоуровневая архитектура.

При разработке таких систем могут использоваться те же серверы баз данных, что и при разработке групповых информационных систем. Однако в крупных информа­ционных системах наибольшее распространение получили серверы Oracle, DB2 и Microsoft SQL Server.

Для групповых и корпоративных систем существенно повышаются требования к надежности функционирования и сохранности данных. Эти свойства обеспечива­ются поддержкой целостности данных, ссылок и транзакций в серверах баз данных.

### 1.5.2. Классификация по сфере применения

По сфере применения информационные системы обычно подразделяются на че­тыре группы (рис. 1.2):

* системы обработки транзакций;
* системы принятия решений;
* информационно-справочные системы;
* офисные информационные системы.

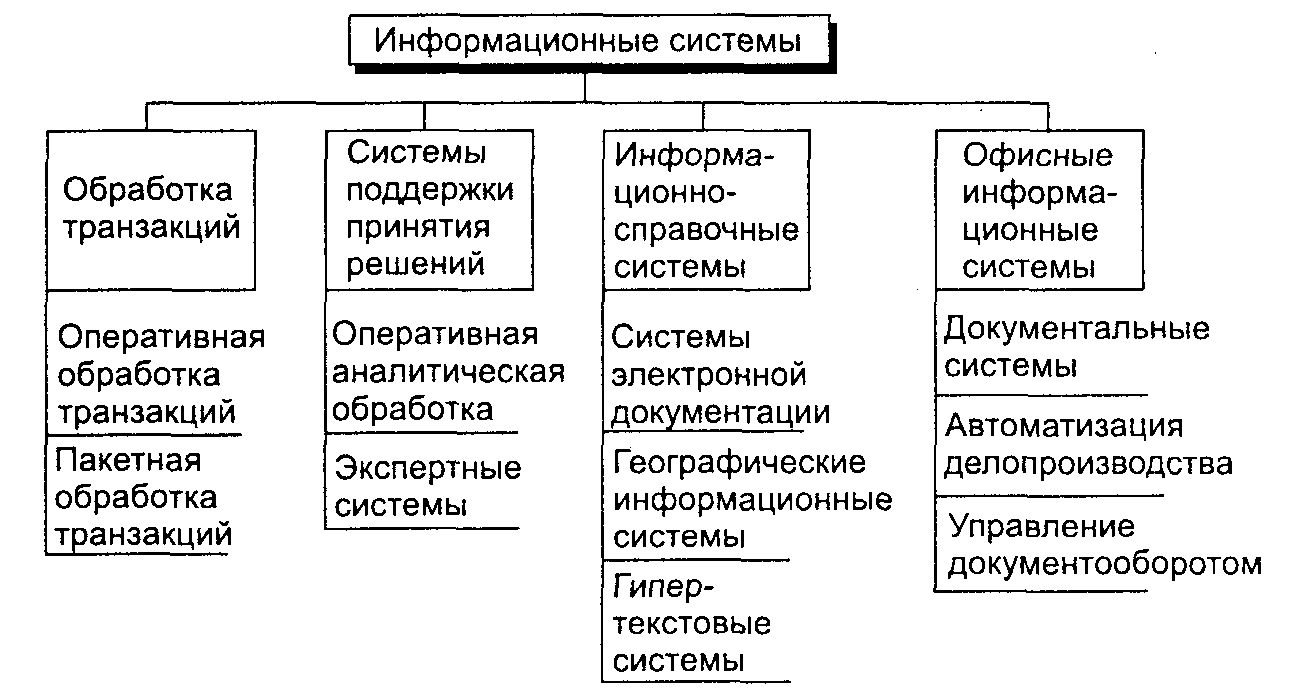


Рис. 1.2. Деление информационных систем по сфере применения

*Системы обработки транзакций,* в свою очередь, по оперативности обработки дан­ных, разделяются на пакетные информационные системы и оперативные инфор­мационные системы. В информационных системах организационного управления преобладает режим оперативной обработки транзакций - OLTP (OnLine Tran­saction Processing), для отражения актуального состояния предметной области и любой момент времени, а пакетная обработка занимает весьма ограниченную часть. Для систем OLTP характерен регулярный (возможно, интенсивный) поток доволь­но простых транзакций, играющих роль заказов, платежей, запросов и т.п. Важ­ными требованиями для них являются:

* высокая производительность обработки транзакций;
* гарантированная доставка информации при удаленном доступе к БД по теле­коммуникациям.

*Системы поддержки принятия решений -* DSS (Decision Support Systeq) - пред­ставляют собой другой тип информационных систем, в которых с помощью до­вольно сложных запросов производится отбор и анализ данных в различных раз­резах: временных, географических и по другим показателям.

Обширный класс *информационно-справочных систем* основам на гипертекстовых документах и мультимедиа. Наибольшее развитие такие информационные систе­мы получили в сети Интернет.

Класс *офисных информационных систем* нацелен на перевод бумажных докумен­тов в электронный вид, автоматизацию делопроизводства и управление докумен­тооборотом.

**ПРИМЕЧАНИЕ:** *Следует отметить, что приводимая классификация по сфере применения в доста­точной степени условна. Крупные информационные системы очень часто обладают признаками всех перечисленных выше классов. Кроме того, корпоративные инфор­мационные системы масштаба предприятия обычно состоят из ряда подсистем, от­носящихся к различным сферам применения.*

### 1.5.3. Классификация по способу организации

По способу организации групповые и корпоративные информационные системы подразделяются па следующие классы (рис. 1.3):

* системы на основе архитектуры файл-сервер;
* системы на основе архитектуры клиент-сервер;
* системы на основе многоуровневой архитектуры;
* системы на основе Интернет/ интранет - технологий.

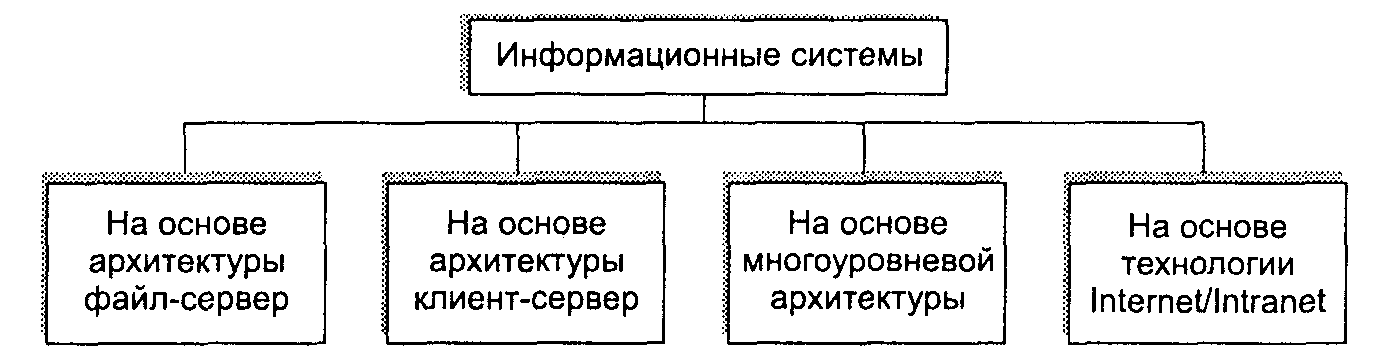


Рис. 1.3. Деление информационных систем по способу организации

В любой информационной системе можно выделить необходимые функциональ­ные компоненты (табл. 1.1), которые помогают понять ограничения различных архитектур информационных систем. Рассмотрим более подробно особенности вариантов построения информационных приложений.

**Таблица 1.1.**

Типовые функциональные компоненты информационной системы

|  |  |  |
| --- | --- | --- |
| **Обозна­чение** | **Наименование** | **Характеристика** |
| **PS** | Presentation Services  (средства представления) | Обеспечиваются устройствами, принимающими ввод от пользователя и отображающими то, что сообщает ему компонент логики представления PL, с использованием соответствующей прог­рам­мной поддержки |
| **PL** | Presentation Logic  (логика представления) | Управляет взаимодействием между пользо­ва­телем и ЭВМ. Обрабатывает действия поль­зователя при выборе команды в меню, нажатии кнопки или выборе элемента из списка |
| **BL** | Business or Application Logic  (прикладная логика) | Набор правил для принятия решений, вы­чис­лений и операций, которые должно выполнить приложение |
| **DL** | Data Logic  (логика управления данными) | Операции с базой данных (SQL-операторы), которые нужно выполнить для реализации прикладной логики управления данными |
| **DS** | Data Services  (операции с базой данных) | Действия СУБД, вызываемые для выполнения логики управления данными, такие как мани­пули­рование данными, определения данных, фиксация или откат транзакций и т.п. СУБД обычно компилирует SQL-предложения |
| **FS** | File Services  (файловые операции) | Дисковые операции чтения и записи данных для СУБД и других компонентов. Обычно являются функциями операционной системы (ОС) |

**Архитектура файл-сервер**

*Архитектура файл-сервер* не имеет сетевого разделения компонентов диалога **PS и PL** и использует компьютер для функций отображения, что облегчает построе­ние графического интерфейса.

Файл-сервер только извлекает данные из файлов, так что дополнительные пользователи и приложения добавляют лишь незначи­тельную нагрузку на центральный процессор. Каждый новый клиент добавляет вычислительную мощность к сети.

Объектами разработки в файл - серверном приложении являются компоненты приложения, определяющие логику диалога **PL,** а также логику обработки **BL** и управления данными **DL**. Разработанное приложение реализуется либо в виде законченного загрузочного модуля, либо в виде специального кода для интер­претации.

Однако такая архитектура имеет существенный недостаток: при выполнении не­которых запросов к базе данных клиенту могут передаваться большие объемы данных, загружая сеть и приводя к непредсказуемости времени реакции.

Значи­тельный сетевой трафик особенно сильно сказывается при организации удален­ного доступа к базам данных на файл-сервере через низкоскоростные каналы связи. Одним из вариантов устранения данного недостатка является удаленное управление файл - серверным приложением в сети. При этом в локальной сети размещается сервер приложений, совмещенный с телекоммуникационным сер­вером (обычно называемым сервером доступа), в среде которого выполняются обычные файл - серверные приложения. Особенность состоит в том, что диалого­вый ввод-вывод поступает от удаленных клиентов через телекоммуникации. Приложения не должны быть слишком сложными, иначе велика вероятность перегрузки сервера, или же нужна очень мощная платформа для сервера прило­жений.

**ПРИМЕЧАНИЕ:** *Одним из традиционных средств, на основе которых создаются файл-серверные си­стемы, являются локальные СУБД. Однако такие системы, как правило, не отвечают требованиям обеспечения целостности данных (в частности, они не поддерживают транзакции). Поэтому при их использовании задача обеспечения целостности дан­ных возлагается на программы клиентов, что приводит к усложнению клиентских при­ложений. Однако эти инструменты привлекают своей простотой, удобством исполь­зования и доступностью. Поэтому файл-серверные информационные системы до сих пор представляют интерес для малых рабочих групп и, более того, нередко использу­ются в качестве информационных систем масштаба предприятия.*

**ПРИМЕР:**

Центральное хранилище

###### БД

Данные о сче­тах всех видов (валютные, руб­левые)

###### Компьютер контролера

###### СУБД

Программа обработки рублевых вкладов

Запрос о счете вкладчика «Х»

Файл, содержащий данные о счетах всех вкладчиков

**Архитектура клиент-сервер**

*Архитектура клиент-сервер* предназначена для разрешения проблем файл-сервер­ных приложений путем разделения компонентов приложения и размещения их там, где они будут функционировать наиболее эффективно.

Особенностью архитектуры клиент-сервер является использование выделенных серверов баз данных, пони­мающих запросы на языке структурированных запросов SQL (Structured Query Language) и выполняющих поиск, сортировку и агрегирование информации.

Большинство конфигураций клиент-сервер использует двухуровневую модель, в ко­торой клиент обращается к услугам сервера. Предполагается, что диалоговые ком­поненты **PS** и **PL** размещаются на клиенте, что позволяет обеспечить графиче­ский интерфейс. Компоненты управления данными **DS** и **FS** размещаются на сервере, а диалог (PS, PL), логика **BL** и **DL** - на клиенте. Двухуровневое опреде­ление архитектуры клиент-сервер использует именно этот вариант: приложение работает у клиента, СУБД - на сервере (рис. 1.4.).

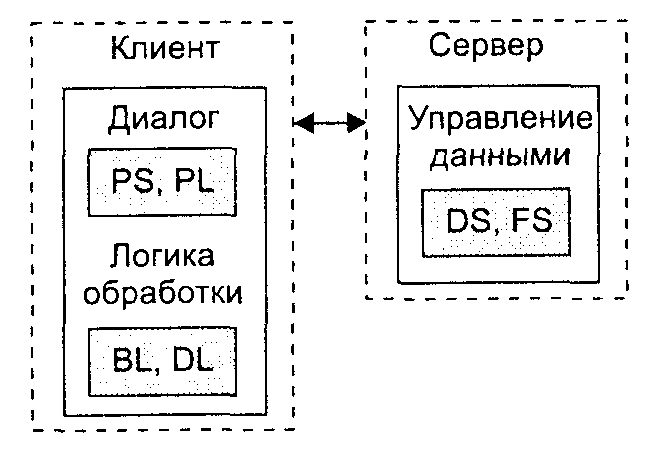


Рис. 1.4. Классический вариант клиент - серверной информационной системы

Поскольку эта схема предъявляет наименьшие требования к серверу, она обла­дает наилучшей масштабируемостью. Однако сложные приложения, вызывающие большое взаимодействие с БД, могут жестко загрузить как клиента, так и сеть. Результаты SQL-запроса должны вернуться клиенту для обработки, пото­му что там находится логика принятия решения. Такая схема приводит к допол­нительному усложнению администрирования приложений, разбросанных по раз­личным клиентским узлам.

Для сокращения нагрузки на сеть и упрощения администрирования приложений компонент **BL** можно разместить на сервере. При этом вся логика принятия реше­ний оформляется в виде *хранимых процедур* и выполняется на сервере БД.

*Хранимая процедура -* процедура с операторами SQL для доступа к БД, вызывае­мая по имени с передачей требуемых параметров и выполняемая на сервере БД. Хранимые процедуры могут компилироваться, что повышает скорость их выпол­нения и сокращает нагрузку на сервер.

Хранимые процедуры улучшают целостность приложений и БД, гарантируют акту­альность коллективно используемых операций и вычислений. Улучшается сопро­вождение таких процедур, а также безопасность (нет прямого доступа к данным).

**ПРИМЕЧАНИЕ:** *Следует помнить, что перегрузка хранимых процедур прикладной логикой может пе­регрузить сервер, что приведет к потере производительности. Эта проблема особен­но актуальна при разработке крупных информационных систем, в которых к серверу может одновременно обращаться большое количество клиентов. Поэтому в большин­стве случаев следует принимать компромиссные решения: часть логики приложения размещать на стороне сервера, часть - на стороне клиента. Такие клиент-серверные системы называются системами с разделенной логикой. Данная схема при удачном разделении логики позволяет получить более сбалансированную загрузку клиентов и сервера, но при этом затрудняется сопровождение приложений.*

Двухуровневые схемы архитектуры клиент-сервер могут привести к некоторым проблемам в сложных информационных приложениях с множеством пользовате­лей и запутанной логикой. Решением этих проблем может стать использование многоуровневой архитектуры.

**ПРИМЕР:**

Центральное хранилище

###### БД

Данные о сче­тах всех видов (валютные, руб­левые)

###### Компьютер контролера

###### СУБД

Программа обработки рублевых вкладов

Запрос о счете вкладчика «Х»

Информация о счете вкладчика «Х»

###### Клиент БД

**Многоуровневая архитектура**

Многоуровневая архитектура стала развитием архитектуры клиент-сервер и в своей классической форме состоит из трех уровней:

* нижний уровень представляет собой приложения клиентов, выделенные для выполнения функций и логики представлений **PS** и **PL** и имеющие программ­ный интерфейс для вызова приложения на среднем уровне;
* средний уровень представляет собой сервер приложений, на котором выполня­ется прикладная логика **BL** и с которого логика обработки данных **DL** вызыва­ет операции с базой данных **DS**;
* верхний уровень представляет собой удаленный специализированный сервер базы данных, выделенный для услуг обработки данных **DS** и файловых опера­ций **FS** (без риска использования хранимых процедур).

Подобную концепцию обработки данных пропагандируют, в частности, фирмы Oracle, Sun, Borland и др.

Трехуровневая архитектура позволяет еще больше сбалансировать нагрузку на разные узлы и сеть, а также способствует специализации инструментов для раз­работки приложений и устраняет недостатки двухуровневой модели клиент-сер­вер.

Централизация логики приложения упрощает администрирование и сопровожде­ние. Четко разделяются платформы и инструменты для реализации интерфейса и прикладной логики, что позволяет с наибольшей отдачей реализовывать их спе­циалистам узкого профиля. Наконец, изменения прикладной логики не затраги­вают интерфейса, и наоборот. Но поскольку границы между компонентами **PL, BL и DL** размыты, прикладная логика может появиться на всех трех уровнях. Сер­вер приложений с помощью монитора транзакций обеспечивает интерфейс с кли­ентами и другими серверами, может управлять транзакциями и гарантировать це­лостность распределенной базы данных. Средства удаленного вызова процедур наиболее соответствуют идее распределенных вычислений: они обеспечивают из любого узла сети вызов прикладной процедуры, расположенной на другом узле, передачу параметров, удаленную обработку и возврат результатов.

С ростом систем клиент-сервер необходимость трех уровней становится все более очевидной. Продукты для трехзвенной архитектуры, так называемые мониторы транзакций, являются относительно новыми. Эти инструменты в основном ори­ентированы на среду UNIX, однако прикладные серверы можно строить на базе Microsoft Windows NT с использованием вызова удаленных процедур для органи­зации связи клиентов с сервером приложений. На практике в локальной сети мо­гут использоваться смешанные архитектуры (двухуровневые и трехуровневые) с одним и тем же сервером базы данных. С учетом глобальных связей архитектура может иметь больше трех звеньев. В настоящее время появились новые инстру­ментальные средства для гибкой сегментации приложений клиент-сервер по раз­личным узлам сети.

Таким образом, многоуровневая архитектура распределенных приложений по­зволяет повысить эффективность работы корпоративной информационной си­стемы и оптимизировать распределение ее программно-аппаратных ресурсов. Но пока на российском рынке по-прежнему доминирует архитектура клиент-сервер.

**Интернет/ интранет - технологии**

В развитии технологии Интернет/интранет основной акцент пока что делается на разработке инструментальных программных средств. В то же время наблюдается отсутствие развитых средств разработки приложений, работающих с базами дан­ных. Компромиссным решением для создания удобных и простых в использова­нии и сопровождении информационных систем, эффективно работающих с база­ми данных, стало объединение Интернет/ интранет - технологии с многоуровневой архитектурой. При этом структура информационного приложения приобретает следующий вид: браузер - сервер приложений - сервер баз данных - сервер ди­намических страниц - web-сервер.

Благодаря интеграции Интернет/ интранет - технологии и архитектуры клиент-сер­вер процесс внедрения и сопровождения корпоративной информационной систе­мы существенно упрощается при сохранении достаточно высокой эффективности и простоты совместного использования информации.

## 1.6. Области применения и примеры реализации инфор­ма­цион­ных систем

В последние несколько лет компьютер стал неотъемлемой частью управленческой системы предприятий. Однако современный подход к управлению предполагает еще и вложение денег в информационные технологии. Причем чем крупнее пред­приятие, тем больше должны быть подобные вложения.

Благодаря стремительному развитию информационных технологий наблюдается расширение области их применения. Если раньше чуть ли не единственной облас­тью, в которой применялись информационные системы, была автоматизация бух­галтерского учета, то сейчас наблюдается внедрение информационных техноло­гий во множество других областей.

Эффективное использование корпоративных информационных систем позволяет делать более точные прогнозы и избегать воз­можных ошибок в управлении.

Из любых данных и отчетов о работе предприятия можно извлечь массу полезных сведений. И информационные системы как раз и позволяют извлекать максимум пользы из всей имеющейся в компании информации.

Именно этим фактом и объясняются жизнеспособность и бурное развитие инфор­мационных технологий - современный бизнес крайне чувствителен к ошибкам в управлении, и для принятия грамотного управленческого решения в условиях не­определенности и риска необходимо постоянно держать под контролем различ­ные аспекты финансово-хозяйственной деятельности предприятия (независимо от профиля его деятельности).

Поэтому можно вполне обоснованно утверждать, что в жесткой конкурентной борь­бе большие шансы на победу имеет предприятие, использующее в управлении со­временные информационные технологии.

Рассмотрим наиболее важные задачи, решаемые с помощью специальных про­граммных средств.

### 1.6.1. Бухгалтерский учет

Это классическая область применения информационных технологий и наиболее ча­сто реализуемая на сегодняшний день задача. Такое положение вполне объяснимо. Во-первых, ошибка бухгалтера может стоить очень дорого, поэтому очевидна выго­да использования возможностей автоматизации бухгалтерии. Во-вторых, задача бух­галтерского учета довольно легко формализуется, так что разработка систем авто­матизации бухгалтерского учета не представляет технически сложной проблемы.

**ПРИМЕЧАНИЕ:** *Тем не менее, разработка систем автоматизации бухгалтерского учета является весь­ма трудоемкой. Это связано с тем, что к системам бухгалтерского учета предъявля­ются повышенные требования в отношении надежности и максимальной простоты и удобства в эксплуатации.*

### 1.6.2. Управление финансовыми потоками

Внедрение информационных технологий в управление финансовыми потоками так­же обусловлено критичностью этой области управления предприятия к ошибкам. Неправильно построив систему расчетов с поставщиками и потребителями, можно спровоцировать кризис наличности даже при налаженной сети закупки, сбыта и хоро­шем маркетинге. И наоборот, точно просчитанные и жестко контролируемые условия финансовых расчетов могут существенно увеличить оборотные средства фирмы.

### 1.6.3. Управление складом, ассортиментом, закупками

Далее, можно автоматизировать процесс анализа движения товара, тем самым, от­следив и зафиксировав те двадцать процентов ассортимента, которые приносят восемьдесят процентов прибыли. Это же позволит ответить на главный вопрос - как получать максимальную прибыль при постоянной нехватке средств?

«Заморозить» оборотные средства в чрезмерном складском запасе - самый про­стой способ сделать любое предприятие, производственное или торговое, потен­циальным инвалидом. Можно просмотреть перспективный товар, вовремя не вло­жив в него деньги.

### 1.6.4. Управление производственным процессом

Управление производственным процессом представляет собой очень трудоемкую задачу. Основными механизмами здесь являются планирование и оптимальное управление производственным процессом.

Автоматизированное решение подобной задачи дает возможность грамотно пла­нировать, учитывать затраты, проводить техническую подготовку производства, оперативно управлять процессом выпуска продукции в соответствии с производ­ственной программой и технологией.

Очевидно, что чем крупнее производство, тем большее число бизнес - процессов участвует в создании прибыли, а значит, использование информационных систем жизненно необходимо.

### 1.6.5. Управление маркетингом

Управление маркетингом подразумевает сбор и анализ данных о фирмах-конкурен­тах, их продукции и ценовой политике, а также моделирование параметров внешне­го окружения для определения оптимального уровня цен, прогнозирования прибы­ли и планирования рекламных кампаний. Решение большинства этих задач могут быть формализованы и представлены в виде информационной системы, позволяю­щей существенно повысить эффективность управления маркетингом.

### 1.6.6. Документооборот

Документооборот является очень важным процессом деятельности любого пред­приятия. Хорошо отлаженная система учетного документооборота отражает ре­ально происходящую на предприятии текущую производственную деятельность и дает управленцам возможность воздействовать на нее. Поэтому автоматизация документооборота позволяет повысить эффективность управления.

### 1.6.7. Оперативное управление предприятием

Информационная система, решающая задачи оперативного управления предпри­ятием, строится на основе базы данных, в которой фиксируется вся возможная информация о предприятии. Такая информационная система является инструмен­том для управления бизнесом и обычно называется корпоративной информаци­онной системой.

Информационная система оперативного управления включает в себя массу программ­ных решений автоматизации бизнес - процессов, имеющих место на конкретном пред­приятии. Одно из наиболее важных требований, предъявляемых к таким информаци­онным системам, - гибкость, способность к адаптации и дальнейшему развитию.

### 1.6.8. Предоставление информации о фирме

Активное развитие сети Интернет привело к необходимости создания корпора­тивных серверов для предоставления различного рода информации о предприя­тии. Практически каждое уважающее себя предприятие сейчас имеет свой web-сервер. Web-сервер предприятия решает ряд задач, из которых можно выделить две основные:

* создание имиджа предприятия;
* максимальная разгрузка справочной службы компании путем предоставления потенциальным и уже существующим абонентам возможности получения не­обходимой информации о фирме, предлагаемых товарах, услугах и ценах.

Кроме того, использование web-технологий открывает широкие перспективы для электронной коммерции и обслуживания покупателей через Интернет.

# Тема 2. Системы документооборота. Lotus Notes

Lotus Notes является наиболее успешным программным продуктом корпорации Lotus Development и относится к категории продуктов *группового программного обеспечения*.

Notes представляет собой мощный программный продукт с широким набором средств. Большинство этих средств помогает вводить и использовать информацию совместно с другими пользователями.

Notes реализует один из способов совместного использования информации, предлагая вам своего рода информационный архив, который вы можете использовать совместно с другими людьми. Этот архив можно интерпретировать как базу данных или как электронную картотеку, в которой вы можете хранить и организовывать определенным образом почти любой вид информации.

Notes предоставляет в ваше распоряжение *шесть* групп, в которые вы можете организовывать базы данных. Каждая из этих групп называется *страницей рабочего пространства*. Эти страницы используются для того, чтобы упорядочить по категориям базы данных.

Все вместе эти шесть страниц составляют ваше *рабочее пространство* в Notes.

## 2.1. Электронная почта

В большинстве офисов электронная почта почти полностью заменила бумажную переписку для внутриофисной связи и даже связи между различными компаниями.

Одним из наиболее важных компонентов Notes является организация почтовой службы.

Ваш компьютер и компьютеры ваших коллег соединяются с сервером Notes при помощи сети. Сервер содержит вашу область памяти и ваш почтовый ящик

## 2.2. Базы данных

Базы данных предоставляют вам возможность хранить информацию в общедоступном месте.

Lotus Notes – это совокупность технологий, предназначенных для эксплуатации и разработки систем коллективного пользования.

Можно выбелить пять основных взаимосвязанных компонент Lotus Notes:

1. Документоориентированные базы данных;

2. Система реплицирования баз данных;

3. Система защиты информации;

4. Электронная почта;

5. Интегрированная среда разработки баз данных.

***Документоориентированные базы данных***

База данных содержит совокупность документов. Документ представляет собой основной модуль информации в Lotus Notes и фактически является электронным эквивалентом документа, отпечатанного на бумаге.

Большинство документов Notes содержит текст, графику, рисунки.

***Система реплицирования баз данных***

Notes обладает архитектурой клиент / сервер. Информация оптимальным образом распределена между центральным сервером Notes локальными компьютерами пользователей. Notes предоставляет доступ к информации вне зависимости от местонахождения пользователей. Вы можете находится дома или в командировке, работать с копией базы данных и периодически связываться с сервером Notes для синхронизации своей копии базы данных с центральной базой на сервере. Для этого используется реплицирование – мощный и гибкий механизм обмена информацией между центральной базой на сервере и копией базы пользователя.

***Интегрированная среда разработки баз данных***

Интегрированная среда разработки баз данных позволяет разработчикам создавать необходимые им базы данных.

### 2.2.1. База данных Notes

Самой важной компонентой Lotus Notes являются базы данных. База данных Notes храниться в файле с расширением \*.NSF и содержит следующие элементы:

* документы с данными, которые надо хранить в базе данных;
* формы – бланки, которые используются при создании и отображении этих документов;
* списки документов, хранящихся в базе данных;
* информация о правах доступа пользователей к базе данных;
* специальные программы, называемые агентами, предназначенные для выполнения определенных действий в базе данных.

### 2.2.2. Формы и документы

Основная единица хранения базы данных Notes – документ, которые является эквивалентом бумажного документа.

Разработчик базы данных заранее определяет, какой тип документов, должен храниться в базе данных, и проектирует одну или несколько форм для создания этих документов.

Форма – это некоторый бланк, который должен заполнить пользователь чтобы поместить документ в базу данных. Каждая база данных имеет свой набор форм, подготовленных разработчиком базы.

Например, база данных "Секретарь" может иметь формы:

* "Корреспонденция";
* "Звонок";
* "Абонент".

Формы содержат некоторые области, помеченные уголками. Эти области формы называются полями. В поля вводятся данные, которые должны храниться в базе.

Также в форме присутствует некоторая текстовая информация, которую в режиме заполнения формы изменить нельзя (надписи в форме).

### 2.2.3. Представления и папки

Другими важными элементами базы данных Notes являются представления и папки.

*Представления –* это списки документов, находящихся в базе данных, с краткой информацией о содержимом этих документов.

Каждая база данных должная иметь по крайней мере одно представление.

Представление может содержать как список всех документов базы данных, так и список части документов, в зависимости от того, как спроектировано представление.

*Папки* также являются списками документов, но для них не устанавливается никакого критерия отбора документов. В начальном состоянии папка может быть пуста, но в любое время вы можете добавить в нее требуемый документ или переместить его из одной папки в другую.

Один и тот же документ может храниться сразу в нескольких папках.

### 2.2.4. Коллективные и локальные базы данных

Система Notes построена в архитектуре клиент / сервер.

*Сервером Notes* называют компьютер в локальной сети, на котором установлено соответствующее программное обеспечение, организующее совместную работу и обслуживание рабочих станций Notes (клиентов), присоединенных к этой сети. Это программное обеспечение также называют *сервером Notes.* Сервер обычно выполняет задачу хранения общих баз данных, их защиту, управление доступом к базам и обеспечение работы почты Notes.

*Рабочая станция* – компьютер, подсоединенный к сети, в которой находится сервер Notes и на котором установлено соответствующее программное обеспечение для рабочих станций Notes, называемое *клиентом Notes.*

Файл с базой данных может находиться либо на сервере Notes, либо на рабочей станции. Базы данных, расположенные на жестком диске рабочей станции, называются *локальными.*

Базы данных, расположенные на сервере Notes, называются базами *коллективного пользования* или *общими*, с ними одновременно могут работать все пользователи, подсоединенные к сети.

Для пользователей, работающих на рабочей станции, в том числе и на удаленной (дома), Notes позволяет создать реплику базы данных, расположенной на сервере.

# Тема 3. Проектирование корпоративных информацион­ных систем

Разработка корпоративной информационной системы, как правило, выполняется для вполне *определенного предприятия*.

Особенности предметной деятельности предприятия, безусловно, будут оказывать влияние на структуру информацион­ной системы. Но в то же время структуры разных предприятий *в целом похожи между собой*. Каждая организация, независимо от рода ее деятельности, состоит из ряда подразделений, непосредственно осуществляющих тот или иной вид дея­тельности компании. И эта ситуация справедлива практически для всех организа­ций, каким бы видом деятельности они ни занимались.

Таким образом, любую организацию можно рассматривать как совокупность взаи­модействующих элементов (подразделений), каждый из которых может иметь свою, достаточно сложную, структуру. Взаимосвязи между подразделениями тоже до­статочно сложны. В общем случае можно выделить три вида связей между подраз­делениями предприятия:

* ***функциональные связи*** - каждое подразделение выполняет определенные виды работ в рамках единого бизнес процесса;
* ***информационные связи*** - подразделения обмениваются информацией (докумен­тами, факсами, письменными и устными распоряжениями и т.п.);
* ***внешние связи*** - некоторые подразделения взаимодействуют с внешними си­стемами, причем их взаимодействие также может быть как информационным, так и функциональным.

Общность структуры разных предприятий позволяет сформулировать некоторые единые принципы построения корпоративных информационных систем.

В общем случае процесс разработки информационной системы может быть рас­смотрен с *двух точек зрения*:

* ***по содержанию действий разработчиков*** (групп разработчиков). В данном слу­чае рассматривается статический аспект процесса разработки, описываемый в терминах основных потоков работ: исполнители, действия, последователь­ность действий и т. п.;
* ***по времени, или по стадиям жизненного цикла разрабатываемой системы***. В дан­ном случае рассматривается динамическая организация процесса разработки, описываемая в терминах циклов, стадий, итераций и этапов.

## 3.1. Общие сведения об управлении проектами

Информационная система предприятия разрабатывается как некоторый проект. Многие особенности управления проектами и фазы разработки проекта (фазы жизненного цикла) являются общими, не зависящими не только от предметной области, но и от характера проекта (неважно, инженерный это проект или эконо­мический). Поэтому имеет смысл вначале рассмотреть ряд общих вопросов управ­ления проектами.

### 3.1.1. Понятие проекта

Проект - это ограниченное по времени целенаправленное изменение отдельной системы с изначально четко определенными целями, достижение которых опреде­ляет завершение проекта, а также с установленными требованиями к срокам, ре­зультатам, риску, рамкам расходования средств и ресурсов и к организационной структуре.

Можно выделить следующие основные отличительные признаки проекта как объекта управления:

* изменчивость - целенаправленный перевод системы из существующего в не­которое желаемое состояние, описываемое в терминах целей проекта;
* ограниченность конечной цели;
* ограниченность продолжительности;
* ограниченность бюджета;
* ограниченность требуемых ресурсов;
* новизна для предприятия, для которого реализуется проект;
* комплексность - наличие большого числа факторов, прямо или косвенно вли­яющих на прогресс и результаты проекта;
* правовое и организационное обеспечение - создание специфической органи­зационной структуры на время реализации проекта.

Рассматривая планирование проектов и управление ими, необходимо четко осоз­навать, что речь идет об управлении неким динамическим объектом. Поэтому система управления проектом должна быть достаточно *гибкой*, чтобы допускать воз­можность модификации без глобальных изменений в рабочей программе.

В системном плане проект может быть представлен «черным ящиком», входом которого являются технические требования и условия финансирования, а итогом работы - достижение требуемого результата (рис. 3.1). Выполнение работ обеспе­чивается наличием необходимых ресурсов:

* материалов;
* оборудования;
* человеческих ресурсов.

Эффективность работ достигается за счет управления процессом реализации про­екта, которое обеспечивает распределение ресурсов, координацию выполняемой последовательности работ и компенсацию внутренних и внешних возмущающих воздействий.

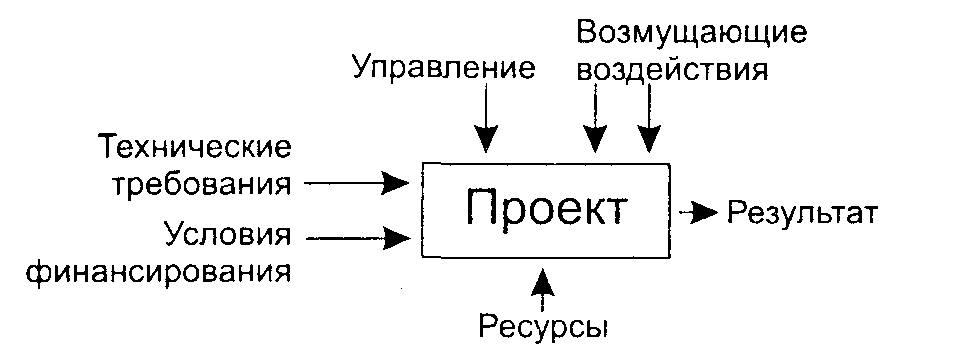


Рис. 3.1. Представление проекта в виде "черного ящика"

С точки зрения теории систем управления проект как объект управления должен быть *наблюдаемым* и *управляемым*, то есть выделяются некоторые характеристи­ки, по которым можно постоянно контролировать ход выполнения проекта (свой­ство наблюдаемости).

Кроме того, необходимы механизмы своевременного воз­действия на ход реализации проекта (свойство управляемости).

Свойство управляемости особенно актуально в условиях неопределенности и измен­чивости предметной области, которые нередко сопутствуют проектам по разработке информационных систем.

Для обоснования целесообразности и осуществимости проекта, анализа хода его реализации, а также для заключительной оценки степени достижения поставлен­ных целей проекта и сравнения фактических результатов с запланированными существует ряд характеристик проекта.

К важнейшим из них относятся технико-экономические показатели:

* объем работ;
* сроки выполнения:
* себестоимость;
* экономическая эффективность, обеспечиваемая реализацией проекта;
* социальная и общественная значимость проекта.

### 3.1.2. Классификация проектов

Проекты могут сильно отличаться по сфере приложения, составу, предметной об­ласти, масштабам, длительности, составу участников, степени сложности, значи­мости результатов и т.п. Проекты могут быть классифицированы по самым раз­личным признакам. Отметим основные из них.

**Класс проекта** определяется но составу и структуре проекта. Обычно различают:

* ***монопроект*** (отдельный проект, который может быть любого типа, вида и мас­штаба);
* ***мультипроект*** (комплексный проект, состоящий из ряда монопроектов и тре­бующий применения многопроектного управления).

**Тип проекта** определяется по основным сферам деятельности, в которых осуще­ствляется проект. Можно выделить пять основных типов проекта:

* технический;
* организационный;
* экономический;
* социальный;
* смешанный.

ПРИМЕЧАНИЕ: Разработка информационных систем относится, скорее всего, к техническим проек­там, которые имеют следующие особенности:

* *главная цель проекта четко определена*, но отдельные цели должны уточняться по мере достижения частных результатов;
* *срок завершения и продолжительность проекта определены заранее*, жела­тель­но их точное соблюдение, однако они также могут корректироваться в зави­си­мос­ти от полученных промежуточных результатов и общего прогресса проекта.

**Масштаб проекта** определяется по размерам бюджета и количеству участников:

* мелкие проекты;
* малые проекты;
* средние проекты;
* крупные проекты.

Можно также рассматривать масштабы проектов в более конкретной форме - отраслевые, корпоративные, ведомственные проекты, проекты одного предпри­ятия.

### 3.1.3. Основные фазы проектирования информационной системы

Каждый проект, независимо от сложности и объема работ, необходимых для его выполнения, проходит в своем развитии определенные состояния: от состояния, когда "проекта еще нет", до состояния, когда "проекта уже нет". Совокупность ступеней развития от возникновения идеи до полного завершения проекта приня­то разделять на **фазы** (стадии, этапы).

В определении количества фаз и их содержания имеются некоторые отличия, по­скольку эти характеристики во многом зависят от условий осуществления конк­ретного проекта и опыта основных участников. Тем не менее, логика и основное содержание процесса разработки информационной системы почти во всех случа­ях являются общими. Можно выделить следующие фазы развития информационной системы:

* формирование концепции;
* разработка технического задания;
* проектирование;
* изготовление;
* ввод системы в эксплуатацию.

Рассмотрим каждую из них более подробно.

***Концептуальная фаза***

Главным содержанием работ на этой фазе является определение проекта, разра­ботка его концепции, включающая:

* формирование идеи, постановку целей;
* формирование ключевой команды проекта;
* изучение мотивации и требовании заказчика и других участников:
* сбор исходных данных и анализ существующего состояния;
* определение основных требований и ограничений, требуемых материальных, финансовых и трудовых ресурсов;
* сравнительную оценку альтернатив;
* представление предложений, их экспертизу и утверждение.

***Разработка технического предложения***

Главным содержанием этой фазы является разработка технического предложения и переговоры с заказчиком о заключении контракта. Общее содержание работ этой фазы:

* разработка основного содержания проекта, базовой структуры проекта;
* разработка и утверждение технического задания;
* планирование, декомпозиция базовой структурной модели проекта:
* составление сметы и бюджета проекта, определение потребности в ресурсах;
* разработка календарных планов и укрупненных графиков работ;
* подписание контракта с заказчиком;
* ввод в действие средств коммуникации участников проекта и контроля за хо­дом работ.

***Проектирование***

На этой фазе определяются подсистемы, их взаимосвязи, выбираются наиболее эффективные способы выполнения проекта и использования ресурсов. Характер­ные работы этой фазы:

* выполнение базовых проектных работ;
* разработка частных технических задании;
* выполнение концептуального проектирования;
* составление технических спецификаций и инструкций;
* представление проектной разработки, экспертиза и утверждение,

***Разработка***

На этой фазе производятся координация и оперативный контроль работ по проек­ту, осуществляется изготовление подсистем, их объединение и тестирование. Ос­новное содержание:

* выполнение работ по разработке программного обеспечения:
* выполнение подготовки к внедрению системы;
* контроль и регулирование основных показателей проекта.

***Ввод системы в эксплуатацию***

На этой фазе проводятся испытания, опытная эксплуатация системы в реальных условиях, ведутся переговоры о результатах выполнения проекта и о возможных новых контрактах. Основные виды работ:

* комплексные испытания;
* подготовка кадров для эксплуатации создаваемой системы;
* подготовка рабочей документации, сдача системы заказчику и ввод ее в экс­плуатацию;
* сопровождение, поддержка, сервисное обслуживание;
* оценка результатов проекта и подготовка итоговых документов;
* разрешение конфликтных ситуации и закрытие работ по проекту;
* накопление опытных данных для последующих проектов, анализ опыта, состо­яния, определение направлений развития.

*Начальные фазы проекта имеют решающее влияние на достигаемый результат, так как в них принимаются основные решения, определяющие качество информацион­ной системы. При этом обычно 30 % вклада в конечный результат проекта вносят фазы концепции и предложения, 20 % - фаза проектирования, 20 % - фаза изготовления, 30 % - фаза сдачи объекта и завершения проекта.*

Кроме того, на обнаружение ошибок, допущенных на стадии системного проекти­рования, расходуется примерно и два раза больше времени, чем на последующих фазах, а их исправление обходится в пять раз дороже. Поэтому на начальных ста­диях проекта разработку следует выполнять особенно тщательно. Наиболее часто на начальных фазах допускаются следующие ошибки:

* ошибки в определении интересов заказчика;
* концентрация на маловажных, сторонних интересах;
* неправильная интерпретация исходной постановки задачи;
* неправильное или недостаточное понимание деталей;
* неполнота функциональных спецификаций (системных требований);
* ошибки в определении требуемых ресурсов и сроков;
* редкая проверка на согласованность этапов и отсутствие контроля со стороны заказчика (нет привлечения заказчика).

# Тема 4. Типы баз данных

Ядром любой базы данных является модель данных. Модель данных представляет собой множество структур данных, ограничений целостности и операций манипулирования данными. С помощью модели данных могут быть представлены объекты предметной области и взаимосвязи между ними.

**Модель данных** – совокупность структур данных и операций их обработки.

К числу классических относятся следующие модели данных:

1) Иерархическая модель данных.

Появление иерархической модели связано с тем, что в реальном мире очень многие связи соответствуют иерархии, когда один объект выступает как родительский, а с ним может быть связано множество подчиненных объектов.

2) Сетевая модель данных.

3) Реляционная модель данных.

# Тема 5. Реляционная модель данных

Реляционная модель ориентирована на организацию данных в виде двумерных таблиц.

Каждая реляционная таблица обладает следующими свойствами:

* каждый элемент таблицы – один элемент данных;
* все столбцы в таблице однородные, т.е. все элементы в столбце имеют одинаковый тип (числовой, символьный и т.д.) и длину;
* каждый столбец имеет уникальное имя;
* одинаковые строки в таблице отсутствуют;
* порядок следования строк и столбцов может быть произвольным.

Реляционная база данных – это совокупность таблиц, содержащих информацию, которая должна храниться в базе данных.

Набор средств для управления подобным хранилищем называется *реляционной системой управления базами данных.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ФИО | Муж | Жена | Сын | Дочь | Внук | Внучка |
| Иванов | NIL | Иванова | NIL | NIL | NIL | NIL |
| Сидоров | NIL | Сидорова | NIL | Ирина | NIL | NIL |
| Петрова | Петров | NIL | NIL | NIL | Сергей | NIL |
| Кирсанов | NIL | Кирсанов | Михаил | NIL | NIL | Юлия |
| Петухов | NIL | Петухова | NIL | NIL | NIL | NIL |

Рис. 5.1. Реляционная база данных, состоящая из 1-ой таблицы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ФИО | Муж | Жена |  | ФИО | Сын | Дочь |
| Иванов | NIL | Иванова |  | Сидоров |  | Ирина |
| Сидоров | NIL | Сидорова |  | Кирсанов | Михаил |  |
| Петрова | Петров | NIL |  |  |  |  |
| Кирсанов | NIL | Кирсанов |  | ФИО | Внук | Внучка |
| Петухов | NIL | Петухова |  | Петрова | Сергей |  |
|  |  |  |  | Кирсанов |  | Юлия |

Рис. 5.2. Реляционная база данных, состоящая из 3-х таблиц

Реляционная БД - это БД в которой:

* данные воспринимаются пользователем как таблицы и только как таблицы;
* операторы, с помощью которых пользователь манипулирует данными, служат для генерации новых таблиц на основе старых.

Например, имеется БД из одной таблицы, в которой 3 столбца и 4 строки:

Таблица "Винный погреб"

|  |  |  |
| --- | --- | --- |
| **Сорт** | **Год** | **Количество\_бутылок** |
| Каберне | 80 | 12 |
| Рислинг | 89 | 14 |
| Изабелла | 80 | 28 |
| Мадера | 77 | 1 |

С помощью специальных операторов выделим из этой таблицы:

а) подмножество строк

SELECT Сорт, Год, Количество\_бутылок FROM Винный\_погреб WHERE Год = 80;

В результате получаем:

|  |  |  |
| --- | --- | --- |
| **Сорт** | **Год** | **Количество\_бутылок** |
| Каберне | 80 | 12 |
| Изабелла | 80 | 28 |

б) подмножество столбцов

SELECT Сорт, Количество\_бутылок FROM Винный\_погреб

В результате получаем:

|  |  |
| --- | --- |
| **Сорт** | **Количество\_бутылок** |
| Каберне | 12 |
| Рислинг | 14 |
| Изабелла | 28 |
| Мадера | 1 |

Оба результата являются таблицами, а операторы SELECT - это операторы языка запросов к БД, называемого SQL (Structured Query Language – Язык структурированных запросов).

## 5.1. Учебная БД

Учебная БД состоит из 3 таблиц.

1) Таблица поставщиков товаров "S".

|  |  |  |  |
| --- | --- | --- | --- |
| nomer | familia | kapital | gorod |
| S1 | Смит | 20 | Лондон |
| S2 | Джонс | 10 | Париж |
| S3 | Блейк | 30 | Париж |
| S4 | Кларк | 20 | Лондон |
| S5 | Адамс | 30 | Саров |

2) Таблица деталей, которые они поставляют "P".

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| nomer | nazvan | zvet | ves | gorod |
| P1 | Гайка | Красный | 12 | Лондон |
| P2 | Болт | Зеленый | 17 | Париж |
| P3 | Винт | Голубой | 17 | Рим |
| P4 | Винт | Красный | 14 | Лондон |
| P5 | Кулачок | Голубой | 12 | Париж |
| P6 | Блюм | Красный | 19 | Лондон |

3) Таблица поставок (какой поставщик, какие детали поставляет) "SP".

|  |  |  |
| --- | --- | --- |
| nomer\_s | nomer\_p | kol |
| S1 | P1 | 300 |
| S1 | P2 | 200 |
| S1 | P3 | 400 |
| S1 | P4 | 200 |
| S1 | P5 | 100 |
| S1 | P6 | 100 |
| S2 | P1 | 300 |
| S2 | P2 | 400 |
| S3 | P2 | 200 |
| S4 | P2 | 200 |
| S4 | P4 | 300 |
| S4 | P5 | 400 |

S - поставщики. Каждый поставщик имеет уникальный номер, фамилию (не обязательно уникальную), капитал и местонахождение.

P - детали (их виды). Каждый вид детали имеет уникальный номер, название, цвет, вес и город, где хранится этот вид детали. Каждый вид имеет только один вид и вес и хранится на складе только одного города (упрощение).

SP - поставки деталей. Связывает между собой две предыдущие таблицы.

Например, первая строка связывает поставщика S1 (Смит) с деталью P1 (гайка), т.е. означает поставку трехсот деталей вида P1 поставщиком S1.

Пусть для любого поставщика имеется только одна поставка детали некоторого вида, т.е. пара (nomer\_s, nomer\_p) является уникальной.

На этом примере отметим несколько важных для РБД обстоятельств:

1) Значения данных во всех клетках таблицы являются одиночными, а не множествами, поэтому в таблице SP имеем:

S2 P1

S2 P2

а не так:

S2 {P1, P2}

2) В каждом столбце хранятся данные только одного типа.

3) БД представляется только в виде явных значений данных. Не существует никаких явных связей и указателей, соединяющих одну таблицу с другой. Связь представляется не с помощью указателя, а значениями клеток в таблицах.

4) Строки в таблицах не требуют никакого упорядочивания.

5) Редко возникают ситуации, когда упорядочение столбцов является существенным. Порядок столбцов в РБД, как правило, не имеет значения.

## 5.2. Создание таблиц

Структура БД определяется оператором CREATE.

CREATE TABLE S (

nomer CHAR(5),

familia CHAR(20),

kapital SMALLINT,

gorod CHAR(15) );

CREATE TABLE P (

nomer CHAR(6),

nazvanie CHAR(20),

zvet CHAR(7),

ves SMALLINT,

gorod CHAR(15) );

CREATE TABLE SP (

nomer\_s CHAR(5),

nomer\_p CHAR(6),

kol INTEGER );

Это операторы создания таблиц языка SQL. Они создают имена таблиц, столбцов и типы данных для этих столбцов.

Оператор CREATE, как и все операторы SQL, выполняемый. Как только СУБД его исполнит, появится таблица, которая будет пустой, т.е. будет содержать лишь заголовки столбцов. Для наполнения таблиц есть другие операторы.

## 5.3. Типы данных в БД

1. Символьный - CHAR(<длина>). Длина <= 32767.

2. Целочисленный со знаком

а) INTEGER <= | 2 147 483 647 |

б) SMALLINT <= | 32767 |

3. Действительное число - FLOAT

Содержит 7 значащих десятичных цифр (доверительные).

1.175494351E-38 - самое маленькое

3.402823446E+38 - самое большое

4. Дата – DATE. 1 янв 100 ... 29 фев 32768

# Тема 6. Выборка данных из БД

Пример по учебной БД: *Выбрать номера и капиталы всех поставщиков из Парижа.*

Выборка данных делается с помощью оператора SELECT.

SELECT nomer, kapital FROM S WHERE gorod='Париж'

Результат:

|  |  |
| --- | --- |
| nomer | kapital |
| S2 | 10 |
| S3 | 30 |

Общий, но не полный вид оператора SELECT.

SELECT [DISTINCT] <список элементов> FROM <список таблиц>

[WHERE <условия>]

[GROUP BY <список столбцов>]

[ORDER BY <список сортировки>]

**Серия примеров.**

1. Простая выборка. Выбрать номера всех поставляемых деталей.

SELECT nomer\_p FROM SP

Результат:

|  |
| --- |
| nomer\_p |
| P1 |
| P2 |
| P3 |
| P4 |
| P5 |
| P6 |
| P1 |
| P2 |
| P2 |
| P2 |
| P4 |
| P5 |

2. Выборка с исключением дубликатов.

SELECT DISTINCT nomer\_p FROM SP.

Результат:

|  |
| --- |
| nomer\_p |
| P1 |
| P2 |
| P3 |
| P4 |
| P5 |
| P6 |

3. Выборка вычисляемых значений. Выбрать номера и капиталы всех поставщиков в рублях, если в таблице S капиталы заданы в $.

SELECT nomer, kapital\*30 FROM S

Результат:

|  |  |
| --- | --- |
| nomer | COLUMN2 |
| S1 | 600 |
| S2 | 300 |
| S3 | 900 |
| S4 | 600 |
| S5 | 900 |

Части SELECT и WHERE могут содержать арифметические выражения и константы.

SELECT nomer, 'Капитал в рублях =', kapital\*30 FROM S

Результат:

|  |  |  |
| --- | --- | --- |
| nomer | COLUMN2 | COLUMN3 |
| S1 | Капитал в рублях = | 600 |
| S2 | Капитал в рублях = | 300 |
| S3 | Капитал в рублях = | 900 |
| S4 | Капитал в рублях = | 600 |
| S5 | Капитал в рублях = | 900 |

4. Выборка всех столбцов. Выбрать все сведения обо всех поставщиках.

SELECT \* FROM S

Результат: копия таблицы S.

Этот оператор эквивалентен по своему действию оператору:

SELECT nomer, familia, kapital, gorod FROM S.

Звездочка ("\*") обозначает список всех столбцов в таблице(ах), указанных в части FROM в том порядке, в котором эти столбцы заданы в операторе CREATE TABLE. Звездочка может дополняться именем таблицы.

SELECT S.\* FROM S

5. Ограниченная выборка. Выбрать номера поставщиков из Парижа, капитал которых больше 20.

SELECT nomer FROM S WHERE (kapital>20) AND (gorod='Париж')

Результат:

|  |
| --- |
| nomer |
| S3 |

Условие части WHERE может включать в себя

* операции сравнения: =, <>, >, >=, <=, <;
* двоичной алгебры: AND, OR, NOT;
* операции принадлежности: IN, NOT IN;
* операции похожести: LIKE;
* скобки: ( ).

6. Выборка с упорядочением. Выбрать номера и капиталы поставщиков из Парижа в порядке убывания их состояния (размера капитала).

SELECT nomer, kapital FROM S WHERE gorod='Париж' ORDER BY kapital DESC

ASC - упорядочение по возрастанию значений (по умолчанию).

DESC - упорядочение по убыванию значений.

Каждый столбец, упомянутый в части ORDER BY должен присутствовать в результирующей таблице, поэтому, например, такой оператор недопустим:

SELECT nomer FROM S ORDER BY gorod;

Правильно будет:

SELECT nomer, gorod FROM S ORDER BY gorod;

7. Выборка с применением принадлежности (IN). Выбрать детали, вес которых равен: 12, 16, 17.

SELECT \* FROM P WHERE ves IN (12, 16, 17);

Результат:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| nomer | nazvan | zvet | ves | gorod |
| P1 | Гайка | Красный | 12 | Лондон |
| P2 | Болт | Зеленый | 17 | Париж |
| P3 | Винт | Голубой | 17 | Рим |
| P5 | Кулачок | Голубой | 12 | Париж |

8. Выборка с применением похожести (LIKE). Выбрать все детали, название которых начинается с буквы 'в'.

SELECT \* FROM P WHERE nazvan LIKE 'в%';

Результат:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| nomer | nazvan | zvet | ves | gorod |
| P3 | Винт | Голубой | 17 | Рим |
| P4 | Винт | Красный | 14 | Лондон |

Общий вид оператора похожести: <имя столбца> LIKE <строковая константа>

<имя столбца> типа CHAR.

<строковая константа> - это образец, в котором:

* "\_" - любой одиночный символ,
* "%" - любое количество любых символов, в том числе нулевое, остальные символы означают сами себя.

# Тема 7. Выборка данных. Соединение

Операция, в которой выборка данных проводится более чем из одной таблицы называется "соединение". Результирующая таблица также называется соединением.

**Простое эквисоединение**

Пример:

Выбрать все данные о поставщиках и деталях, размещенных в одном и том же городе, то есть соразмещенных.

SELECT S.\*, P.\* FROM S, P WHERE S.gorod=P.gorod;

Очевидно, что нужные данные получаются из двух таблиц S и P - они указаны в части FROM, а условие соединения записывается в части WHERE.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Поля таблицы "S" | | | | Поля таблицы "P" | | | | |
| nomer | familia | kapital | gorod | nomer | nazvan | zvet | ves | gorod |
| S1 | Смит | 20 | Лондон | P1 | Гайка | Красный | 12 | Лондон |
| S1 | Смит | 20 | Лондон | P4 | Винт | Красный | 14 | Лондон |
| S1 | Смит | 20 | Лондон | P6 | Блюм | Красный | 19 | Лондон |
| S2 | Джонс | 10 | Париж | P2 | Болт | Зеленый | 17 | Париж |
| S2 | Джонс | 10 | Париж | P5 | Кулачок | Голубой | 12 | Париж |
| S3 | Блейк | 30 | Париж | P2 | Болт | Зеленый | 17 | Париж |
| S3 | Блейк | 30 | Париж | P5 | Кулачок | Голубой | 12 | Париж |
| S4 | Кларк | 20 | Лондон | P1 | Гайка | Красный | 12 | Лондон |
| S4 | Кларк | 20 | Лондон | P4 | Винт | Красный | 14 | Лондон |
| S4 | Кларк | 20 | Лондон | P6 | Блюм | Красный | 19 | Лондон |

Поставщик из Сарова не попал в результирующую таблицу, так как в этом городе не хранятся никакие детали. Аналогично, результат не содержит детали P3, так как ее город - Рим.

Некоторые замечания

* Оба столбца в условии соединения должны быть либо числовыми, либо строковыми.
* Совсем не обязательно, чтобы столбцы в условии соединения назывались одинаково, хотя это бывает часто.
* В случае операции равенства ("=") в условии соединения, оно называется эквисоединением; в условии соединения могут быть и другие операции.
* Часть WHERE может включать и другие дополнительные условия, помимо условия соединения.
* Можно предусмотреть выборку не всех, а некоторых столбцов; кроме того, первоначальный оператор можно еще более упростить:

SELECT \* FROM S, P WHERE S.gorod=P.gorod;

* По определению, эквисоединение должно порождать два полностью одинаковых столбца; если исключить один из этих столбцов, то результат будет называться естественным соединением.
* Можно образовать соединение любого числа таблиц.

**Соединение по условию "больше, чем"**

Пример:

Выбрать все данные о поставщиках и деталях таких, что город поставщика следует за городом, где хранится деталь в алфавитном порядке.

SELECT S.\*, P.\* FROM S, P WHERE S.gorod > P.gorod;

**Соединение с дополнительным условием**

Пример:

Выбрать все комбинации номеров поставщиков, номеров деталей и состояний такие, что поставщик и деталь соразмещены. Опустить поставщиков с капиталом 20.

SELECT S.nomer, P.nomer, S.kapital FROM S, P

WHERE (S.gorod = P.gorod) AND (kapital <> 20);

S.gorod = P.gorod - условие соединения

kapital <> 20 - дополнительное условие

Результат:

|  |  |  |
| --- | --- | --- |
| nomer | nomer | kapital |
| S2 | P2 | 10 |
| S2 | P5 | 10 |
| S3 | P2 | 30 |
| S3 | P5 | 30 |

**Соединение таблицы с ней самой**

Пример:

Выбрать все пары номеров поставщиков такие, что оба поставщика находятся в одном и том же городе.

SELECT PERV.nomer, VTOR.nomer FROM S PERV, S VTOR

WHERE PERV.gorod = VTOR.gorod;

Таблица S дважды упоминается в части FROM, чтобы различать два ее упоминания вводятся произвольные псевдонимы.

Результат:

|  |  |
| --- | --- |
| nomer | nomer |
| S1 | S1 |
| S1 | S4 |
| S2 | S2 |
| S2 | S3 |
| S3 | S3 |
| S3 | S2 |
| S4 | S4 |
| S4 | S1 |
| S5 | S5 |

Можно привести этот результат в порядок, если расширить часть WHERE.

SELECT PERV.nomer, VTOR.nomer

FROM S PERV, S VTOR

WHERE (PERV.gorod = VTOR.gorod) AND (PERV.nomer < VTOR.nomer);

Последнее условие ("<") даст двоякий эффект:

а) оно исключает пары номеров вида (x,x);

б) гарантирует, что пары вида (x,y) и (y,x) не будут появляться одновременно.

Результат:

|  |  |
| --- | --- |
| nomer | nomer |
| S1 | S4 |
| S2 | S3 |

# Тема 8: Выборка. Подзапрос

Подзапрос - это SELECT, который вложен во внешний оператор SELECT.

Обычно подзапросы требуются для выборки значений, когда во внешнем запросе применяется операция принадлежности (IN).

**Простой подзапрос**

Пример:

Выбрать фамилии поставщиков, которые поставляют деталь P2.

SELECT familia FROM S

WHERE nomer IN (SELECT nomer\_s FROM SP WHERE nomer\_p = 'P2')

Результат:

|  |
| --- |
| familia |
| Смит |
| Джонс |
| Блейк |
| Кларк |

При обработке полного запроса сначала обрабатывается подзапрос. Этот подзапрос возвращает множество номеров поставщиков поставляющих деталь P2, а именно множество: ('S1', 'S2', 'S3', 'S4').

Поэтому первоначальный запрос эквивалентен следующему простому запросу:

SELECT familia FROM S WHERE nomer\_s IN ('S1', 'S2', 'S3', 'S4');

Задачу из примера можно решить, применяя соединение.

SELECT familia FROM S, SP WHERE (S.nomer = SP.nomer\_s) AND (nomer\_p = 'P2');

**Подзапрос с несколькими уровнями вложенности**

Пример:

Выбрать фамилии поставщиков, которые поставляют, по крайней мере, одну красную деталь.

SELECT familia FROM S WHERE nomer\_s IN

(SELECT nomer\_s FROM SP WHERE nomer\_p IN

(SELECT nomer FROM P WHERE zvet = 'красный'))

Результат:

|  |
| --- |
| familia |
| Смит |
| Джонс |
| Кларк |

Если использовать соединение трех таблиц, то результат следующий:

Результат:

|  |
| --- |
| familia |
| Смит |
| Джонс |
| Кларк |
| Смит |
| Смит |

**Подзапрос с операцией отличной от IN**

Пример:

Выбрать номера поставщиков, находящихся в том же городе, что и поставщик S1.

SELECT nomer FROM S WHERE gorod =

(SELECT gorod FROM S WHERE nomer = 'S1');

Результат:

|  |
| --- |
| nomer |
| S1 |
| S4 |

Если заранее известно, что подзапрос возвратит в точности 1 значение, то вместо IN можно употребить операцию сравнения (=, <>, >, ...). Однако, если подзапрос вернет более одного значения, то возникнет ошибка.

**Стандартные функции**

На вопросы типа "сколько строк в таблице?" отвечают стандартные функции языка SQL (агрегирующие функции):

COUNT - количество строк

SUM - сумма значений в столбце

AVG - среднее арифметическое по значениям в столбце

MAX - наибольшее значение в столбце

MIN - наименьшее значение в столбце

SUM и AVG применяются только к столбцам числового типа.

После имени функции, в скобках, нужно записывать имя столбца, а перед ним может присутствовать слово DISTINCT. Оно указывает, что дубликаты следует исключить перед тем как будет применяться функция.

Специальный вид функции COUNT(\*) предназначен для подсчета всех строк в таблице, без исключения дубликатов.

**Функция в части SELECT**

Пример:

Подсчитать общее количество поставщиков.

SELECT COUNT(\*) FROM S

Результат:

|  |
| --- |
| COUNT |
| 5 |

**Функция в части SELECT с исключением дубликатов**

Пример:

Подсчитать общее количество поставщиков, поставляющих детали.

SELECT COUNT(DISTINCT nomer\_s) FROM SP;

Результат:

|  |
| --- |
| COUNT |
| 4 |

Функция в части SELECT с условием в части WHERE

Пример:

Подсчитать общее количество поставляемых деталей P2 (сколько деталей по всем поставкам?).

SELECT SUM (kol) FROM SP WHERE nomer\_p = 'P2';

Результат:

|  |
| --- |
| SUM |
| 1000 |

**Группирование записей**

Пример:

Требуется подсчитать общее количество поставляемых деталей по каждому виду деталей. То есть, по каждому виду выдать номер детали и суммарный объем поставок.

SELECT nomer\_p, SUM(kol) FROM SP GROUP BY nomer\_p;

Результат:

|  |  |
| --- | --- |
| nomer\_p | SUM |
| P1 | 600 |
| P2 | 1000 |
| P3 | 400 |
| P4 | 500 |
| P5 | 500 |
| P6 | 100 |

Как исполняется такой оператор?

1) Часть GROUP BY перекомпоновывает таблицу, указанную в части FROM в группы, таким образом, чтобы в каждой группе все строки имели одно и то же значение в столбце, указанном в GROUP BY.

2) К каждой группе перекомпонованной таблицы (а не к каждой исходной строке) применяется часть SELECT.

Каждое выражение в части SELECT должно принимать единственное значение для группы. То есть, оно может быть либо самим столбцом в части GROUP BY, либо арифметическим выражением, содержащим такой столбец, либо константой, либо функцией (например, SUM), которая сводит разные значения к одному.

# Тема 9. Обновление строк в таблице

Оператор обновления (модификации) строк в одной таблице имеет вид:

UPDATE <имя таблицы>

SET <имя столбца> = <выражение>{, <имя столбца> = <выражение>}

[ WHERE <условие> ];

Все строки в таблице, которые удовлетворяют <условию>, изменяются в соответствии с присваиваниями в части SET.

Примеры:

**Обновление единственной строки**

Изменить цвет детали P2 на желтый и увеличить ее вес на 5.

UPDATE P SET zvet='желтый', ves=ves+5 WHERE nomer='P2';

**Обновление нескольких строк**

UPDATE S SET kapital=kapital\*2 WHERE gorod='Лондон'

**Обновление нескольких таблиц**

Изменить номер поставщика с S2 на S9.

UPDATE S SET nomer='S9' WHERE nomer='S2';

UPDATE SP SET nomer='S9' WHERE nomer='S2';

В одном операторе невозможно обновить две таблицы!

После первого оператора БД стала противоречивой. Теперь в ней есть поставки у которой нет поставщика. В этом противоречивом состоянии БД будет находиться до завершения второго оператора. Изменения порядка операторов принципиально ничего не меняет. Поэтому, с точки зрения целостности БД важно обеспечить завершение обоих операторов, а не одного из них (см. дальше).

# Тема 10. Удаление строк

DELETE

FROM <имя таблицы>

[ WHERE <условие> ];

Этот оператор удаляет все строки из таблицы, которые удовлетворяют <условию>.

# Тема 11. Вставка строк

Две формы оператора INSERT

а)

INSERT

INTO <имя таблицы> [ ( <имя столбца>{, <имя столбца>} ) ]

VALUES ( <константа>{, <константа>} );

б)

INSERT

INTO <имя таблицы> [ ( <имя столбца>{, <имя столбца>} ) ]

<подзапрос>;

В форме (а) в таблицу вставляется строка из констант, перечисленных в части VALUES. Причем, i-ая константа соответствует i-му столбцу в части INTO.

В форме (б) сначала вычисляется <подзапрос> (результат которого - таблица) и результат подзапроса вставляется в таблицу <имя таблицы>.

Пример:

Добавить в таблицу P деталь P7: город Москва, вес 2, название и цвет пока не известны.

INSERT INTO P (nomer, gorod, ves) VALUES ('P7', 'Москва', 2);

В таблице P появляется новая строка. Порядок столбцов в части INTO не обязательно должен совпадать с порядком столбцов при создании таблицы.

**Вставка единственной строки с опущенными именами столбцов**

Добавить в таблицу P деталь P8: название "звездочка", цвет розовый, вес 14, город Берлин.

INSERT INTO P VALUES ('P8', 'звездочка', 'розовый', 14, 'Берлин');

Если имена столбцов не заданы, то предполагается перечисление всех столбцов таблицы в порядке их перечисления в операторе CREATE.

**Вставка множества строк**

Для каждой поставляемой детали получить ее номер и объем ее поставки, сохранить результат в отдельной таблице.

CREATE TABLE VREMEN

(nomer CHAR(6),

sum\_postavki INTEGER)

INSERT INTO VREMEN SELECT nomer\_p, SUM(kol) FROM SPGROUP BY nomer\_p

# Тема 12. Управление транзакциями

Вернемся к примеру в котором заменяется номер поставщика S2 на S9. Обобщим этот пример. Допустим, есть некий язык программирования, внутри которого можно выполнять операторы SQL. Напишем процедуру, которая будет менять номер поставщика с SX на SY.

Trans: proc(SX, SY);

ON ERROR: begin ROLLBACK; RETURN end;

EXEC UPDATE S SET nomer=SY WHERE nomer=SX;

EXEC UPDATE SP SET nomer\_s=SY WHERE nomer\_s=SX;

EXEC COMMIT;

end;

Эта процедура воспринимается ее пользователем как неделимая (атомарная операция). На самом деле эта процедура состоит из 2-х операторов SQL. И между этими операторами нарушается согласованность БД (есть поставки у которых нет поставщика). Чтобы сохранить согласованность БД нужно уметь выполнять последовательность операторов SQL как неделимую операцию.

*Транзакция* - это последовательность операторов SQL, которая либо выполняется целиком, либо не выполняется совсем.

Предполагается при этом, что транзакция переведет некоторое согласованное состояние БД в другое согласованное состояние, но не гарантирует сохранение согласованности во все промежуточные моменты времени.

* Последовательность может состоять из одного оператора SQL.
* Транзакцию создает программист.

СУБД гарантирует, что если транзакция проводила некоторые изменения в БД (UPDATE, INSERT, DELETE) и затем, по какой-либо причине, произошла ошибка до нормального завершения транзакции, то эти изменения будут отменены.

Таким образом, транзакция либо полностью исполняется, либо полностью отменяется, как будто бы она не исполнялась вообще.

Вопрос: Как СУБД узнает когда начинается и когда заканчивается транзакция?

Транзакция заканчивается, чаще всего, как и в этом примере, с помощью операторов COMMIT (принять, подтвердить) и ROLLBACK (отменить, откатиться).

Оператор COMMIT сообщает СУБД об успешном завершении транзакции, о том, что БД вновь должна находиться в согласованном состоянии и все изменения в БД следует сделать постоянными.

Напротив, оператор ROLLBACK сообщает СУБД о неудачном завершении транзакции, о том, что БД находится, возможно, в противоречивом состоянии и что все сделанные внутри транзакции изменения (UPDATE, INSERT, DELETE) следует отменить.

Даже если во время исполнения транзакции сломается компьютер, то тот оператор ROLLBACK, который должен был выполняться по ошибке, будет все равно выполнен при рестарте СУБД.

Как отменяются изменения?

С помощью журнала, в котором записываются все операции изменения БД.

В частности, значения таблиц до и после изменения.

Транзакция начинается:

* первым оператором, после соединения с БД;
* первым оператором, после конца предыдущей транзакции.

Транзакция заканчивается:

* оператором COMMIT;
* оператором ROLLBACK;
* отсоединением от БД, причем предполагается ROLLBACK.

## 12.1. Три проблемы, связанные с параллельностью тран­зак­ций

Всякая СУБД, которая позволяет любому количеству пользователей одновременно пользоваться данными в одной и той же БД, должна иметь механизм, обеспечивающий независимость пользователей друг от друга, по крайней мере, кажущуюся.

Такой механизм нужен чтобы избежать *трех случаев ошибочного исполнения транзакции*, когда транзакция правильная сама по себе, может приводить к ошибочному результату из-за вмешательства со стороны некоторой другой транзакции, тоже правильной.

Таким образом, будем обсуждать чередование операторов SQL, по крайней мере, в двух параллельных и правильных транзакциях, которое может привести к ошибочному в целом результату.

### 12.1.1. Проблема утраченного обновления

-------------------------+-----------------------+-----------------

Время TA ! Значения ! TB

-------------------------+-----------------------+-----------------

t0 start ! !

t1 Сч1=SELECT R ! Сч1(TA)=50, R=50 !

t2 ! ! start

t3 ! Сч1(TB)=50, R=50 ! Сч1=SELECT R

t4 UPDATE R=Сч1+15 ! R=65 !

t5 COMMIT ! !

t6 ! R=40 ! UPDATE R=Сч1-10

t7 ! ! COMMIT

-------------------------+-----------------------+-----------------

TA выбирает строку R в t1, TB - в t3. TA обновляет R в t4, исходя из значений "увиденных" в t1, а TB обновляет эту же строку в t6, исходя из значений "увиденных" в t3. Очевидно, что в итоге в R остается лишь значение записанное TB, которая перекрывает значение записанное TA не глядя на него.

### 12.1.2. Проблема зависимости от неподтвержденных обновлений

Возникает в том случае, если одной транзакции позволяется выбрать данные из строки, которая уже была обновлена другой транзакцией, но не успела подтвердить обновление.

------------------------+-----------------------+-----------------

Время TA ! Значения ! TB

------------------------+-----------------------+-----------------

t0 ! R=50 ! start

t1 ! R=20 ! UPDATE R=20

t2 start ! !

t3 Сч1=SELECT R ! Сч1(TA)=20 !

t4 ! R=50, Сч1(TA)=20 ! ROLLBACK

------------------------+------------------------+-----------------

TA "видит" неподтвержденное изменение, которое позже отменяется TB.

Следовательно, TA работает при ошибочном предположении, что R имеет значение 20, а на самом деле оно 50.

### 12.1.3. Проблема противоречивости

-------------------------+-----------------------+-----------------

Время TA ! Значения ! TB

-------------------------+-----------------------+-----------------

t0 start ! R1=40, R2=50, R3=30 !

t1 Сч1=SELECT R1 ! Сч1(TA)=40,Сум(TA)=40 !

t2 Сч2=SELECT R2 ! Сч2(TA)=50,Сум(TA)=90 !

t3 ! ! start

t4 ! Сч3(TB)=30 ! Сч3=SELECT R3

t5 ! R3=20 ! UPDATE R3=Сч3-10

t6 ! Сч1(TB)=40 ! Сч1=SELECT R1

t7 ! R1=50 ! UPDATE R1=Сч1+10

t8 ! ! COMMIT

t9 Сч3=SELECT R3 ! Сч3(TA)=20,Сум(TA)=110!

------------------------+------------------------+-----------------

Сум(TA) должна = 120

Здесь TA суммирует остатки на счетах, а TB переносит сумму 10 со счета 3 на счет 1. В результате параллельного выполнения TA получает неверную сумму.

Различия между этой и предыдущей проблемой в том, что здесь TA не зависит от неподтвержденных изменений, поскольку TB сделало все изменения постоянными еще до того, как TA "увидела" счет 3.

Все три случая ошибочного выполнения транзакций возникают только тогда, когда хотя бы одна транзакция изменяет данные в БД. Конфликтов не бывает, когда все транзакции только выбирают (читают) данные.

## 12.2. Решение проблем параллелизма

Существует два основных подхода к управлению параллельными транзакциями:

- основанный на захвате объекта БД (блокировке);

- основанный на метках времени (многоверсионных объектах).

### 12.2.1. Управление с помощью захватов (блокировок)

Главная идея захвата проста: если для транзакции нужно, чтобы некоторый объект БД (обычно строка таблицы) не изменялся до ее завершения, то она устанавливает захват этого объекта. Захват заключается в том, что объект изолируется от других транзакций.

Различают два вида режимов захвата:

1) совместный режим (режим С) и

2) монопольный режим (режим М).

Ради упрощения будем обсуждать захваты только одной строки в таблице.

**Алгоритм захватов**

1. Если TA устанавливает М-захват строки R, то запрос из TB на любого типа захват строки R приведет к тому, что TB перейдет в состояние блокировки.

TB будет находиться в этом состоянии, пока TA не снимет захват.

2. Если TA устанавливает С-захват на строку R, то:

а) запрос из TB на М-захват строки R заставит TB перейти в состояние блокировки и TB будет заблокирована, пока TA не снимет свой захват.

б) запрос из TB на С-захват строки R будет удовлетворен, то есть TB также будет удерживать С-захват строки R.

Обсуждение п.2:

- в отсутствии захватов строки R будет удовлетворен запрос на захват любого вида этой строки;

- во время М-захвата строки R в запросе любого вида на захват R будет отказано;

- во время С-захвата строки R будут удовлетворяться только С-запросы.

3. Запросы транзакций на захват строки всегда являются неявными. Когда транзакция исполняет оператор SELECT она автоматически устанавливает С-захват. Когда транзакция изменяет, добавляет, удаляет строку, она автоматически устанавливает М-захват. Если же транзакция получила строку R в С-захват, а потом стала изменять строку R, то транзакция автоматически повышает вид захвата с С на М.

**Решение трех проблем с помощью механизма блокировок в СУБД SQL Anywhere**

1\*) Решение первой проблемы (утраченное обновление)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | ТА | Значения | ТВ | Комментарий |
| t1 | start |  |  |  |
| t2 | Сч1=Select R | Сч1 (ТА)=50 |  | С-запрос на R от ТА и захват |
| t3 |  |  | start |  |
| t4 |  | Сч1 (ТВ)=50 | Сч1=Select R | С-запрос на R от ТВ и захват |
| t5 | UPDATE R=Сч1+15 |  |  | М-запрос на R от ТА и отказ |
| t6 | ждать |  |  | ТА ждет завершения ТВ |
| t7 |  |  | UPDATE R=Сч1+15 | М-запрос на R от ТВ и отказ |
| t8 |  |  | ждать | ТВ ждет завершения ТА |

**Захваты и тупики**

С помощью захватов решается проблема утраченного обновления, но появляется *проблема тупика* - взаимной блокировки нескольких транзакций, при которой каждая из транзакций ждет, пока другая снимет захват.

Если возникает тупиковая ситуация, то ее обнаруживает и разрушает СУБД.

Чтобы ликвидировать тупик, СУБД выбирает из вовлеченных в тупик транзакций одну - жертву - и принудительно выполняет ее откат.

2\*) Решение проблемы зависимости от неподтвержденных обновлений с помощью механизма блокировки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | ТА | Значения | ТВ | Комментарий |
| t0 |  | R=50 | start |  |
| t1 |  | R=20 | UPDATE R=20 | М-запрос на R от ТВ и захват |
| t2 | start |  |  |  |
| t3 | Сч1=SELECT R |  |  | С-запрос на R от ТА и отказ |
| t4 | ждать |  |  | ТА ждет завершения ТВ (блокир) |
| t5 | ждать | R=50 | ROLLBACK | ТВ заканчивается откатом, ТА разблокируется |
| t6 | Сч1=SELECT R | Сч1 (ТА)=50, R=50 |  | С-запрос на R от ТА и захват |

3\*) Решение проблемы противоречивости с помощью механизма блокировки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | ТА | Значения | ТВ | Комментарий |
| t0 | start | R1=40, R2=50, R3=30 |  |  |
| t1 | Сч1=SELECT R1 | Сч1 (ТА)=40,  Сум (ТА)=40 |  | С-запрос на R1 от TA и захват |
| t2 | Сч2=SELECT R2 | Сч2 (ТА)=50,  Сум (ТА)=90 |  | С-запрос на R2 от TA и захват |
| t3 |  |  | start |  |
| t4 |  | Сч3 (ТВ)=30 | Сч3=SELECT R3 | С-запрос на R3 от TВ и захват |
| t5 |  | R3=20 | UPDATE R3=Сч3-10 | М-запрос на R3 от TB и захват |
| t6 |  | Сч1 (ТВ)=40 | Сч1=SELECT R1 | С-запрос на R1 от ТВ и захват |
| t7 |  |  | UPDATE R1=Сч1+10 | М-запрос на R1 от ТВ и отказ, т.к. есть С-запрос на R1 от ТА |
| t8 |  |  | ждать | ТВ ждет завершения ТА |
| t9 | Сч3=SELECT R3 |  | ждать | С-запрос на R3 от ТА и отказ, т.к. есть М-запрос на R3 от ТВ |
| t10 | ждать |  | ждать | ТА ждет завершения ТВ |
|  |  |  |  | Тупик, который приведет к принудительному откату одной из транзакций |

### 12.2.2. Управление на основе многоверсионных объектов с метками времени

Главная идея этого подхода не в том, чтобы изолировать объект в БД от другой транзакции, а в том, чтобы создать отдельную копию объекта для каждой параллельной транзакции. Тогда транзакции от своего начала и до конца будут работать с собственными копиями объекта. По завершении транзакции все сделанные ей изменения могут быть перенесены из ее копии объекта в БД.

Некоторые обстоятельства:

1) Очевидно, что беспрепятственно изменить свою копию сможет лишь одна из параллельных транзакций, которая сделает это первой (TA). СУБД не позволит другой транзакции (TB) изменить тот же объект и, либо аварийно завершит TB, либо переведет TB в ожидание до конца TA, что зависит от заданных для TB условий старта.

Если TB будет ждать завершения TA, то возможны два исхода:

а) TA подтверждает свои изменения (COMMIT), тогда TB аварийно завершается;

б) TA отменяет свои изменения (ROLLBACK), тогда TB переносит свои изменения в БД и продолжает работать.

2) Каждая параллельная транзакция может беспрепятственно читать и всегда видеть в точности то состояние БД, которое было когда транзакция началась, плюс ее собственные изменения, но не может видеть изменения в БД, сделанные другими параллельными транзакциями.

**Решение трех проблем с помощью многоверсионных объектов в СУБД Interbase**

Объектом для транзакции является строка.

Interbase создает новую версию строки всякий раз, когда какая-либо транзакция изменяет или удаляет эту строку. Interbase помечает эту версию строки временем появления и именем транзакции, которая воздействовала на строку.

По запросу любой транзакции на чтение этой строки, Interbase предоставляет ей самую свежую версию строки, но уже подтвержденную до старта этой транзакции.

1\*\*) Решение первой проблемы (утраченное обновление)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | ТА | Значения | ТВ | Комментарий |
| t0 |  | R (t0, TX)=50 |  | Последняя подтвержденная версия R на t0 |
| t1 | start |  |  |  |
| t2 | Сч1=SELECT R | Сч1 (ТА)=50 |  | ТА получает подтвержденную версию от времени t0<t1 |
| t3 |  |  | start |  |
| t4 |  | Сч1 (ТВ)=50 | Сч1=SELECT R | ТВ получает подтвержденную версию от времени t0<t3 |
| t5 | UPDATE R=Сч1+15 | R (t5, ТА) = 65 ? |  | Не известно, будет ли подтверждено изменение |
| t6 |  | Сч1 (ТВ)=50 | Сч1=SELECT R | Версия от t5 для TB не доступна, так как t5>t3 |
| t7 |  |  | UPDATE R=Сч1-10 | TB блокируется, т.к. есть версия R от t5 и t5>t3 |
| t8 |  |  | ждать | ТВ ждет завершения ТА |
| t9 | COMMIT | R (t9,TA)=65 | конфликт | Подтвержденная версия от t9 для TB не доступна, т.к. t9>t3 |
| t10 |  |  | ROLLBACK | TB должна отменить свои изменения |
| t11 |  |  | start (ТВ') |  |
| t12 |  | Сч1 (ТВ')=65 | Сч1=SELECT R | t12>t9 |
| t13 |  | R (t13, TB')=55 ? | UPDATE R=Сч1-10 |  |
| t14 |  | R (t14, TB')=55 | COMMIT |  |

TA беспрепятственно изменило R, поскольку сделало это первой. TB не удалось изменить R, так как она не "видела" самой последней подтвержденной версии.

(1\*\*) решает проблему (1) и, в отличие от (1\*), исключает тупиковую ситуацию.

2\*\*) Решение проблемы зависимости от неподтвержденных обновлений с помощью многоверсионных объектов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | ТА | Значения | ТВ | Комментарий |
| t0 |  | R=50 | start | Последняя подтвержденная версия R на t0 |
| t1 |  | R(t1,TB)=20 ? | UPDATE R=20 | Не известно, будет ли подтвер­ждено изменение |
| t2 | start |  |  |  |
| t3 | Сч1=SELECT R | R(t3,TA)=50, Сч1(ТА)=50 |  | Последняя подтвержденная версия R на t3 |
| t4 |  | R=50, R(t4,TA)=50, Сч1(ТА)=50 | ROLLBACK | ТВ отменяет свои изменения и заканчивается, т.е. изменения ТВ в БД не переносятся |
| t5 |  |  |  | ТА может спокойно продол­ать работу |

3\*\*) Решение проблемы противоречивости с помощью многоверсионных объектов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | ТА | Значения | ТВ | Комментарий |
| t0 | start | R1=40, R2=50, R3=30 |  | Последние подтвержденные версии R1, R2, R3 |
| t1 | Сч1=SELECT R1 | R1(t1,TA)=40, Сч1(ТА)=40, Сум(ТА)=40 |  |  |
| t2 | Сч2=SELECT R2 | R1(t2,TA)=50, Сч2(ТА)=50, Сум(ТА)=90 |  |  |
| t3 |  | R1(TB)=40, R2(TB)=50,  R3(TB)=30 | start |  |
| t4 |  | Сч3(ТВ)=30 | Сч3=SELECT R3 |  |
| t5 |  | R3(TB)=20 ? | UPDATE R3=Сч3-10 | Не известно, будет ли подтверждено изменение |
| t6 |  | Сч1(ТВ)=40 | Сч1=SELECT R1 |  |
| t7 |  | R1(TB)=50 ? | UPDATE R1=Сч1+10 | Не известно, будет ли подтверждено изменение |
| t8 |  | R1(TB)=50, R3(TB)=20  R1=50, R2=50, R3=20 | COMMIT | Конфликт: подтвержденные версии R1 и R3 для ТА не доступны, т.к. t8>t0 |
| t9 | ROLLBACK |  |  | ТА должна сделать откат |
| t10 | start (TA') | R1(TA')=50, R2(TA')=50,  R3(TA')=20 |  | Последние подтвержденные версии R1, R2, R3 |
| t11 | Сч1=SELECT R1 | Сч1(ТА')=50, Сум(ТА')=50 |  |  |
| t12 | Сч2=SELECT R2 | Сч2(ТА')=50, Сум(ТА')=100 |  | 50+50=100 |
| t13 | Сч3=SELECT R3 | Сч3(ТА')=20, Сум(ТА')=120 |  | 100+20=120  Получили правильный резуль­тат для ТА: 120 |

# Тема 13. Объектно-ориентированное программирование

Существует две модели построения программ:

* процессно-ориентированная;
* объектно-ориентированная.

*Процессно-ориентированная*. В данной модели программа представляется как ряд последовательно выполняемых операций (процедур). Языки программирования, в которых реализован процессно-ориентированный подход к построению программ, называются процедурными.

*Объектно-ориентированная*. При использовании данной модели программа рассматривается как совокупность объектов – отдельных фрагментов кода, обеспечивающих выполнение определенных действий и объединяющих данные и методы управления ими.

## 13.1. Основы языка Object Pascal

### 13.1.1. Структура программы в Object Pascal

Программа, написанная на языке Object Pascal, состоит из ряда разделов. Начало каждого раздела указывается с помощью специальных зарезервированных слов. В общем виде программа Object Pascal имеет следующий вид:

|  |  |
| --- | --- |
| Program Name\_of\_program  Uses  Unit1, Unit2, Unit3  Label  Label1, Label2  Type  id\_type1 = type\_def1;  id\_type2 = type\_def2;  Var  id\_var1: type\_def1;  id\_var2, id\_var3: id\_type2;  Const  id\_const1 = value1;  id\_const2 = value2;  id\_const3 = expression1;  Procedure proc1;  текст процедуры  Function Func1: type\_def1;  текст функции  begin  текст программы  end. | Заголовок программы  Раздел объявления используемых модулей  Раздел объявления используемых меток  Раздел описания типов, определяемых пользователем  Раздел объявления переменных  Раздел описания констант  Раздел объявления процедур и функций, используемых в программе |

**Заголовок программы.** В заголовке после служебного слова *Program* указывается имя программы.

**Раздел объявления модулей.** Начало данного раздела указывается с помощью директивы *Uses*.

Разделов объявления меток, типов, констант и переменных может быть несколько, и они могут следовать в любом порядке.

**Раздел объявления меток.** Начало раздела указывается с помощью директивы *Label*.

**Раздел описания типов.** В Object Pascal существует довольно большое количество стандартных типов. Однако при разработке программ, особенно при использовании объектно-ориентированного программирования, программисту необходима возможность создавать свои типы данных. Для описания пользовательских типов используется раздел объявления типов, начало которого определяет директивы *Type*. Самым простым вариантом объявления собственного типа является просто объявление типа, аналогично уже существующему, например:

Type

id\_type1 = integer;

**Раздел переменных.** Начало раздела объявляется с помощью служебного слова *Var*. В данном разделе должны быть описаны все переменные, которые будут использоваться в программе.

Компилятор Object Pascal не допускает использование переменных, не объявленных в разделе Var. Объявление переменной, не используемой в программе, не приводит к ошибке компиляции, однако компилятор будет выдавать предупреждение о том, что переменная объявлена, но никогда не используется.

При объявлении переменной указывается идентификатор и через двоеточие – тип переменной, например:

var

id\_var1: integer;

Идентификаторы переменных одного типа можно перечислять через запятую:

var

id\_var1, id\_var2, id\_var3: integer;

Для нестандартных типов имя типа должно быть описано в разделе Type, находящемся выше раздела Var, в котором оно используется.

**Раздел констант.** Данный раздел содержит объявления констант и начинается с директивы Const. Константа фактически является переменной, значение которой устанавливается не в процессе выполнения программы, а на этапе компиляции. Значение константы не может изменяться программно. В объявлении константы можно использовать не только конкретные значения, но и выражения. Тип константы определяется присваиваемым ей значением или типом результата, получаемого при вычислении выражения.

### 13.1.2. Типы данных

**Простые типы данных**

В Object Pascal к простым типам данных относятся:

* порядковые;
* вещественные;
* тип дата/время.

Порядковые типы. Каждый из порядковых типов имеет конечное число значений. Следовательно, их можно упорядочить и с каждым из них сопоставить порядковый номер значения.

В Object Pascal к порядковым типам относят:

* целые;
* логические;
* символьные;
* перечисляемые;
* диапазонные.

*Целые типы.*

|  |  |  |
| --- | --- | --- |
| **Название** | **Диапазон значений** | **Размер, байт** |
| Byte | 0 … 255 | 1 |
| ShortInt | -128 … +127 | 1 |
| SmallInt | -32768 … +32767 | 2 |
| Word | 0 … 65535 | 2 |
| Integer | -2147483648 … +2147483647 | 4 |
| LongWord | 0 … 4294967295 | 4 |

*Логические типы.*

|  |  |
| --- | --- |
| **Название** | **Размер, байт** |
| Boolean | 1 |

Логические переменные могут принимать одно из двух значений: True или False.

*Символьные типы.*

|  |  |
| --- | --- |
| **Название** | **Размер, байт** |
| Char | 1 |

*Перечисляемые типы.*

Перечисляемый тип – это тип, определенный путем явного перечисления всех возможных его значений.

В программе перечисляемые типы описываются после ключевого слова Type и находятся в списке внутри круглых скобок:

type

test = (Level1, Level2, Level3);

season = (winter, spring, summer, autumn);

drink = (tea, coffee, cocoa, water)

Первое значение в списке имеет порядковый номер 0, второе 1 и т.д. Максимальный порядковый номер – 65536.

*Тип – диапазон.*

Тип – диапазон тоже является порядковым типом. Тип – диапазон – это подмножество значений любого порядкового типа, определяемое минимальным и максимальным значением. В подмножество входят все значения, находящиеся в этих границах.

type

number = 1..20

letter = 'a'..'z'

var

number1: 500..700

Минимальное значение не должно быть больше максимального. Тип – диапазон поддерживает все операции базового типа.

Вещественные типы. В переменных вещественных типов содержатся числа, состоящие из целой и дробной частей.

|  |  |  |
| --- | --- | --- |
| **Название** | **Диапазон значений** | **Размер, байт** |
| Real | 2.9Е-39 – 1.7Е38 | 6 |
| Single | 1.5Е-45 – 3.4Е38 | 4 |
| Double | 5.0Е-324 – 1.7Е308 | 8 |

Стандартные математические функции Object Pascal.

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Описание** | **Пример** | **Результат** |
| Trunc (x)  Int (x) | Возвращает целое значение аргумента | Trunc (700.40)  Int (700.40) | 700  700 |
| Frac (x) | Выделяет дробную часть числа | Frac (56.45) | 0.45 |
| Pi | Задает число Пи |  |  |
| Random | Генерирует случайным образом число от 0 до 1 |  |  |
| Sqr (x) | Возводит аргумент в квадрат | Sqr (2) | 4 |
| Sqrt (x) | Извлекает из аргумента квад­ратный корень | Sqrt (9) | 3 |

Тип дата/время. Для одновременного хранения даты и времени определен тип дата/время. Этот тип определяется стандартным идентификатором TDateTime.

Функции для работы с датой и временем

|  |  |
| --- | --- |
| **Название** | **Описание** |
| Date: TDateTime | Возвращает текущую дату |
| DateToStr (D: TDateTime) | Преобразует дату в строку символов |
| DateTimeToStr (D: TDateTime) |  |
| DecodeDate () | Выделяет в дате и возвращает год, месяц и день |
| DecodeTime () | Выделяет во времени и возвращает часы, минуты, секунды и милисекунды |
| Now: (D: TDateTime) | Возвращает текущую дату и время |
| Time: (D: TDateTime) | Определяет текущее время |
| TimeToStr (T: TDateTime) | Преобразует время в строку |

procedure TForm1.Button1Click(Sender: TObject);

var

NowDate: TDateTime;

Year, Month, Day, Hour, Min, Sec, MSec: Word;

begin

NowDate:=Now;

DecodeDate(NowDate, Year, Month, Day);

DecodeDate(NowDate, Hour, Min, Sec, MSec);

end;

**Структурированные типы данных**

Структурированные типы данных определяют наборы однотипных или разнотипных элементов.

В Object Pascal определены следующие структурные типы:

* строки;
* массивы;
* множества;
* записи;
* файлы;
* классы.

Строковые типы.

* ShortString (длина до 256 символов);
* String.

Массивы.

* статистические;

var

A: array[1..10] of integer //одномерный массив

B: array[0..9,0..4] of double //многомерный массив

* динамические.

var

A: array of char;

### 13.1.3. Простые и структурированные операторы языка

Оператор в программе – это единое неделимое предложение, выполняющее какое-либо алгоритмическое действие.

Все операторы Object Pascal можно разделить на простые и структурированные.

*Простыми* считаются операторы, которые не содержат в себе других операторов.

К ним относятся:

* оператор присваивания;
* вызов процедуры;
* пустой оператор.

*Структурированными* являются операторы, которые состоят из других операторов.

К ним относятся:

* составной оператор;
* условный оператор **if**;
* условный оператор **case**;
* оператор цикла **repeat**;
* оператор цикла **while**;
* оператор цикла **for**;
* оператор над записями **with**.

Составной оператор – это последовательность операторов, заключенный в зарезер­ви­рованные слова **begin** и **end**.

Условный оператор **if** предназначен для выбора к выполнению одного из двух действий, в зависимости от некоторого условия.

**if** условие **then** действие1 **else** действие2

Условный оператор **case** позволяет осуществлять выбор из любого количества вариантов.

**Case** ключ выбора **of**

константа выбора : оператор;

константа выбора : оператор;

**else**

операторы

**end**;

Оператор цикла **repeat** организует выполнение цикла, состоящего из любого числа операторов, с неизвестным заранее числом повторений.

**Repeat**

тело цикла

**until**

условие;

Оператор цикла **while**. Цикл прекращает работу, если условие ложно.

**While** условие **do** оператор;

Оператор цикла **for** организует выполнение оператора заранее известное количество раз.

**for** параметр цикла:=начальное значение **to** конечное значение **do**

оператор;

### 13.1.4. Процедуры и функции

Процедуры и функции представляют собой блоки программного кода, имеющие точно такую же структуру, как и программа (не могут содержать раздел uses).

**Процедуры**

Пример программной реализации процедуры:

procedure proc\_id (<список параметров>);

const //Раздел описания локальных констант

const1 = value1;

type //Раздел описания локальных типов

type\_id1 = type\_def1;

var

var\_id1: type\_id1;

var\_id2, var\_id3 : type\_def2;

begin

… //Текст процедуры

end;

Свойства процедуры:

* количество передаваемых параметров не ограничено;
* внутри процедуры формальные параметры представляют собой обычные переменные или константы;
* вызов процедуры:

…

proc\_id(A,B)

…

* параметры, указываемые при вызове процедуры, называются фактическими. Они представляют собой переменные или константы, описанные в программе.

**Функции**

Функции отличаются от процедур тем, что их идентификатор возвращает некоторое значение.

function MyFunc(A: integer) : single

begin

…

end

### 13.1.5. Модули Object Pascal

При разработке программ в среде Delphi широко используются *модули.* Они позволяют объединить логически связанные типы данных, переменные, процедуры и функции в один программный блок.

Все идентификаторы, описанные в модуле, могут быть использованы в других программных блоках. Для использования идентификаторов, описанных в модуле в программе, достаточно объявить имя модуля в разделе uses.

Структура модуля Object Pascal имеет вид:

unit name\_of\_unit; //Заголовок модуля

interface //Блок интерфейса

uses

unit1, unit2;

const

const1 = value1;

const2 = expression1;

type

type1 = type\_def1;

var

var\_id1 : type\_def1;

var\_id2 : type\_def2;

procedure proc\_id1;

function func\_id1 : type\_def2;

implementation //Блок реализации

uses

unit3, unit4;

const

const3 = value2;

type

type2 = type\_def3;

var

var\_id3, var\_id4 : type\_def4;

procedure proc\_1;

begin

…

end;

function func1 : type\_def5;

begin

…

end;

procedure proc\_id1;

begin

…

end;

function func\_id1 : type\_def2;

begin

…

end;

end.

*Заголовок модуля* состоит из ключевого слова unit и имени модуля, которое обязательно должно совпадать с именем файла. Заголовок является обязательным.

*Блок интерфейса* содержит описание констант, типов, переменных, процедур и функций, которые будут доступны в других программах и модулях. В блоке интерфейса описываются только заголовки процедур и функций. Их текст приводится в разделе реализации.

*Блок реализации* может содержать объявление констант, типов, переменных, процедур и функций. Все эти объявления доступны только в данном модуле. Также в разделе размещается реализация всех процедур и функций, заголовки которых объявлены в блоке интерфейса.

## 13.2. Объектно-ориентированное программирование

В основе объектно-ориентированного программирования лежит идея объединения в одной структуре данных и действий, которые выполняются над этими данными.

В объектно-ориентированном программировании базовыми единицами программ и данных являются *классы*.

### 13.2.1. Основные понятия

**Классы**

Класс – это структура данных, которая может содержать в своем составе переменные, функции и процедуры.

Переменные в зависимости от назначения, называются *полями*, или *свойствами*. Процедуры и функции, входящие в состав класса, называются *методами*.

В Object Pascal определен структурный тип **class**.

Тип *class* обязательно должен быть описан как пользовательский тип в разделе *type*, например:

type

TMyClass = Class

field1: type\_definition1;

field2: type\_definition2;

procedure method1;

function method2 : type\_definition3;

end;

Затем в разделе *var* может быть объявлена переменная объектного типа:

var

Object1 : TMyClass;

При объявлении класса вначале описываются поля, а затем методы. Поля класса являются переменными, входящими в состав его структуры. Они предназначены для использования внутри класса.

В описании объектного типа присутствуют только заголовки методов. Сами методы описываются в разделе реализации того модуля, в котором объявляется новый объектный тип.

**Объекты**

*Объектом* или экземпляром класса называется переменная объектного типа.

Чтобы объект мог обмениваться данными с другими объектами, используются свойства. Свойства объекта определяют его состояние. Изменение состояния объекта производится только через вызов методов этого объекта.

### 13.2.2. Основные концепции

Объектно-ориентированное программирование базируется на трех основных понятиях: инкапсуляции, наследовании и полиморфизме.

**Инкапсуляция** – комбинирование данных с процедурами и функциями, которые манипулируют этими данными.

Например, окружность описывается координатами центра и радиусом (данные). Кроме того, над окружностью можно проделывать различные действия (методы):

* вычислять ее длину и площадь ограниченного ею круга;
* проверять находится ли некоторая точка внутри данной окружности и т.п.

Класс, описывающий объект "окружность", может выглядеть следующим образом:

type

TCircle = class

x,y : double;

r : double;

function area : double; // Вычисление площади окружности

function circumference : double; // Вычисление длины окружности

function inside (x,y : double) : Boolean; //Проверка нахождения точки внутри окр-ти

end;

Для работы с классом необходимо создать его экземпляр, то есть описать в разделе *var* переменную данного объектного типа:

var

Circle : TCircle;

Доступ к полям класса осуществляется следующим образом:

|  |  |
| --- | --- |
| …  Circle.x:=5;  Circle.y:=20;  Circle.r:=10;  … | …  with Circle do begin  x:=5;  y:=20;  r:=10;  end;  … |

Аналогичным образом производится и вызов методов. Например, чтобы рассчитать площадь окружности, используется следующая строка:

|  |  |
| --- | --- |
| A:=Circle.area | Методу area не нужно передавать никаких данных. Подразумевается, что метод применяется к экземпляру класса, внутри которого он определен. Таким образом, для расчета площади метод area использует данные, содержащиеся в поле r данного экземпляра класса. |

Инкапсуляция позволяет обеспечить защиту данных от внешнего вмешательства или неправильного использования.

**Наследование** – это возможность использования уже определенных классов для построения иерархии классов, производных от них.

Новый класс может быть определен на основе уже имеющегося класса. При этом новый класс наследует как данные старого класса, так и методы их обработки.

Например, на основе класса, описывающего объект "окружность", можно создать класс, описывающий объект "кольцо". Причем часть свойств и методов у этих объектов будут общими: координаты центра, радиус внешней окружности, метод расчета длины внешней окружности. Поэтому при объявлении класса "кольцо" не нужно заново описывать эти свойства и методы:

TRing = class (TCircle)

r2 : double;

function area : double; // Вычисление площади кольца

function circumference2 : double; // Вычисление длины внутренней окружности

function inside (x,y : double) : Boolean; // Проверка нахождения точки внутри кольца

end;

В объявлении класса TRing указываются функции расчета площади кольца и определения попадания некоторой точки с заданными координатами внутрь кольца. Хотя имена этих методов совпадают с именами соответствующих методов для класса TCircle, их реализация должна быть иной, так как они применяются к разным геометрическим фигурам.

Если имена методов, объявляемых в дочернем классе, совпадают с именами полей или методов, то говорят, что они *перекрываются*.

В Object Pascal при объявлении объектного типа имя наследуемого класса указывается в круглых скобках после слова class. По умолчанию считается, что класс, определяемый пользователем, является наследником от класса TObject.

Например, объявления, представленные ниже идентичны.

TMyClass = class

TMyClass = class (TObject)

**Полиморфизм** - это свойство родственных классов решать схожие по смыслу проблемы разными способами. В рамках Object Pascal поведенческие свойства класса определяются набором входящих в него методов.

Изменяя алгоритм того или иного метода в потомках класса, программист может придавать этим потомкам отсутствующие у родителя специфические свойства.

Для изменения метода необходимо перекрыть его в потомке, то есть объявить в потомке одноименный метод и реализовать в нем нужные действия.

В результате в объекте – родителе и объекте – потомке будут действовать два одноименных метода, имеющие разную алгоритмическую основу и, следовательно, придающие объектам разные свойства. Это и называется полиморфизмом объектов.

### 13.2.3. Поля, свойства и методы

Класс является сложной структурой данных, объединяющей переменные, функции и процедуры в одном типе данных.

Переменные, входящие в состав класса, называются *полями*. Процедуры и функции класса обычно называют *методами*. Свойства класса представляют собой поля, обращение к которым производится через специальные методы.

**Поля**

Поля класса представляют собой переменные, объявленные внутри класса. Объявление полей класса должно предшествовать объявлению методов и свойств.

Например, класс, содержащий одно поле и один метод, будет описываться следующим образом:

TSampleClass = class (TObject) //Объявление нового класса

FSample : integer; //Поле класса

procedure SampleMethod; //Метод класса

end;

**Свойства**

Прямое обращение к полям, определяющим состояние объекта, противоречит принципам объектно-ориентированного программирования. Поэтому для обмена данными с другими объектами используются свойства, обращение к которым может выполняться не напрямую, а только через соответствующие методы.

В этом и заключается отличие свойств от полей, к которым можно обращаться непосредственно.

Для объявления свойств используется служебное слово *property*.

Так как свойство может обмениваться данными только через соответствующие методы, то при объявлении свойства обычно используют три элемента: свойство и два метода, обеспечивающие обращение к нему (чтение и запись).

TSampleClass = class (TObject)

FSample : integer;

procedure SetProp : TPropType; //Метод записи

function GetProp (NewValue : TPropType); //Метод чтения

property SampleProp : TPropType read GetProp write SetProp; //Объявление свойства

end;

При обращении к свойству класса нет необходимости в явном виде вызывать методы, обеспечивающие чтение значения свойства или изменение его значения.

…

SampleObject.SampleProp:= NewValue;

…

Value:= SampleObject.SampleProp;

…

Синтаксически обращение к свойству может выглядеть точно так же, как и обращение к полю.

**Методы**

Методы предназначены для манипулирования данными, входящими в состав класса. Фактически методы представляют собой обычные процедуры и функции, которым разрешен доступ ко всем полям класса.

Методы объявляются в описании класса после объявления полей.

# Тема 14. Среда разработки Delphi

## 14.1. Быстрая разработка приложений

*Быстрая разработка приложений* (RAD – Rapid Application Development) основывается на визуализации процесса создания программного кода.

Средства быстрой разработки приложений основываются на компонентной архитектуре. При этом *компоненты* являются объектами, объединяющими данные и методы, а также свойства.

Компоненты Delphi поддерживают PME-модель (Property, Method, Events – свойства, методы, события), позволяющую изменять поведение компонентов без необходимости создания новых классов.

Процедура разработки интерфейса средствами RAD сводится к набору последовательных операций, включающих:

* размещение компонентов интерфейса в нужном месте;
* задание моментов времени их появления на экране;
* настройку связанных с ними атрибутов и событий.

Система визуального программирования Delphi позволяет в полной мере реализовать современные концепции программирования, включая:

* объектно-ориентированный подход;
* визуальные средства быстрой разработки приложений, основанные на компонентной архитектуре;
* возможность работы с базами данных.

## 14.2. Главное окно Delphi

Интегрированная среда разработки Delphi включает в себя четыре основные части:

* главное окно;
* инспектор объектов;
* редактор форм;
* редактор кода.

Главное окно состоит из трех разделов:

* главное меню;
* панель инструментов;
* палитра компонентов.

### 14.2.1. Палитра компонентов

Палитра компонентов используется для отображения компонентов, содержащихся в библиотеке компонентов Delphi.

В соответствии с выполняемыми ими функциями все расположенные в палитре компоненты разделены на группы, каждая из которых размещается на отдельной странице палитры.

Стандартная конфигурация палитры компонентов Delphi 3 содержит 13 страниц, каждая из которых предоставляет разнообразные компоненты и элементы управления:

|  |  |
| --- | --- |
| Standard | стандартные элементы управления оконного интерфейса Windows; |
| Additional | специализированные элементы управления интерфейса Windows; |
| Win32 | элементы интерфейса, содержащиеся системных библиотеках Windows; |
| Data Access | компоненты, обеспечивающие доступ к информации, хранящейся в базах данных, и использующие процессор баз данных BDE; |
| Data Controls | компоненты для отображения и редактирования информации, хранящейся в базах данных. |

### 14.2.2. Инспектор объектов

Инспектор объектов является одним из важнейших инструментов разработки приложения и используется для настройки опубликованных свойств компонента.

Окно инспектора объектов содержит выпадающий список и две вкладки:

* Properties;
* Events.

## 14.3. Управление проектами

В среде Delphi работа ведется проектами – наборами файлов, из которых состоит приложение. Ими могут быть:

* файлы с исходным текстом модулей (расширение .PAS);
* формы, являющиеся графическим представлением приложения (расширение .DFM);
* файлы проектов (расширение .DPR).

Каждому файлу формы обязательно соответствует файл с исходным текстом модуля, но файл с исходным текстом модуля не обязательно должен иметь соответствующую ему форму.

В начале работы среда разработки фактически предоставляет в распоряжение готовую программу, состоящую из одного окна с заголовком Form1.

## 14.4. Основные компоненты для построения простых прило­жений

Условно все компоненты Delphi можно разделить на две группы:

* визуальные (компоненты интерфейса);
* невизуальные (системные компоненты).

*Визуальные компоненты* видны как во время разработки, так и во время выполнения программы. Визуальные компоненты используются для создания интерфейса пользователя.

*Невизуальные компоненты* видны только во время разработки. Они предназначены для разработки логической структуры приложения.

Все компоненты Delphi являются потомками класса TComponent и имеют большое количество общих свойств и событий.

### 14.4.1. Формы

Любое приложение, разработанное в среде Delphi, должно содержать, по крайней мере, одну форму.

Форма представляет собой окно приложения на этапе разработки и обеспечивает создание интерфейса пользователя, являясь контейнером для размещения элементов интерфейса.

Различаются два типа форм – модальные и немодальные.

*Модальные формы* не позволяют передавать фокус ввода в другие окна приложения до тех пор, пока модальное окно не закрыто. Пример модального окна – окно диалога.

*Немодальные формы* могут передавать управление другим окнам приложения, оставаясь открытыми.

**Класс TForm**

Все свойства класса TForm можно разделить на две группы – опубликованные свойства, т.е. те свойства, которые отображаются в окне инспектора объектов во время разработки приложения, и свойства, которые можно изменять только в процессе выполнения программы.

*Основные опубликованные свойства класса TForm*

|  |  |  |
| --- | --- | --- |
| **Свойство** | **Тип** | **Описание** |
| Caption | TCaption | Заголовок окна |
| Color | TColor | Цвет фона окна |

Кроме свойств класс TForm включает ряд методов, которые могут быть полезны при разработке приложения.

*Основные методы класса TForm*

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| procedure Close | Вызывает метод CloseQuery и, если он возвращает true, закрывает форму |
| function CloseQuery : Boolean | Используется для определения, может ли форма быть закрыта |
| procedure Hide | Скрывает форму, не уничтожая ее |
| procedure Show | Отображает форму в немодальном режиме |
| function ShowModal : integer | Отображает форму в модальном режиме |

В классе TForm определен ряд методов-обработчиков событий, которые позволяют задавать реакцию экземпляра класса TForm на определенные действия.

|  |  |
| --- | --- |
| **Метод-обработчик** | **Описание** |
| OnActivate | Вызывается при передаче форме фокуса ввода |
| OnClick | Вызывается при одиночном щелчке на форме |
| OnClose | Вызывается при закрытии формы |
| OnShow | Вызывается при отображении формы |

### 14.4.2. Стандартные элементы интерфейса

В библиотеке Delphi содержится ряд компонентов, которые предназначены для создания стандартных элементов интерфейса приложений Windows. Все эти компоненты доступны в палитре компонентов и могут размещаться на формах.

*Основные классы визуальных компонентов*

TForm - форма

TMainMenu - главное меню

TMenuItem - элемент меню

TPopupMenu - выпадающее меню

TLabel - текст

TEdit - ввод и редактирование строки

TMemo - ввод и редактирование текста

TButton - кнопка

TCheckBox - переключатель

TRadioButton - радио-кнопка

TListBox - список (выбор из списка)

TComboBox - комбинированный ввод (ввод строки или выбор из списка)

TScrollBar - элемент для "прокручивания"

TGroupBox - рамка для группирования элементов

TRadioGroup - рамка для группирования радио-кнопок

TPanel - панель

TImage - растровое изображение

*Основные свойства некоторых визуальных компонентов*

Name - имя компонента

Caption - текст в заголовке или в элементе

Color - цвет

Enabled - доступность/ недоступность компонента

Font - шрифт

Hint - текст всплывающей подсказки

Visible - видимость/ невидимость компонента

WordWrap - переносить ли слова

Width - длина по горизонтали

Height - высота

# Тема 15. Доступ к БД в архитектуре фирмы Borland

Delphi обладает мощными средствами для разработки приложений, управляющих базами данных на основе собственного *процессора баз данных* BDE.

BDE взаимодействует с базами данных через драйверы. Для большинства наиболее распространенных баз данных разработаны стандартные драйверы.

Кроме того, обеспечивается возможность взаимодействия с базой данных через драйвер ODBC. Доступ к SQL-серверам обеспечивает система драйверов SQL Links, позволяющая разрабатывать приложения для серверов Oracle, Informix, Sybase, DB2 и InterBase.

## 15.1. Доступ к данным с использованием BDE

Компьютер - клиент

SQL Explorer

Прикладные программы

Настройка

Borland Delphi Engine (BDE)

Локальные БД

BDE Administrator

Драйвер фирмы Microsoft

Драйверы:

ODBC

Драйверы:

SQL Links

Компьютер - сервер

SQL-сервер:

InterBase

SQL-сервер:

DB2

SQL-сервер:

Sybase

SQL-сервер:

Informix

SQL-сервер:

Oracle

Рис. 15.1. Схема доступа к БД

BDE разрешает реляционный (на языке SQL) и навигационный (понятия текущей, следующей и предыдущей строки и т.п.) способы доступа к БД, доступ к локальным и серверным БД.

Для каждой БД строится свой драйвер, который учитывает особенности СУБД и транслирует вызовы функций общего вида в вызовы частного вида.

Каждой БД в BDE обычно назначают условное имя (alias), через которое к ней идет обращение. Перенаправив условное имя на другую БД, можно легко переключить прикладную программу на другие данные и даже на данные другого формата. Назначение условных имен производится программами SQL Explorer или BDE Administrator.

## 15.2. Компоненты доступа к данным

Любое приложение, работающее с базами данных, должно обеспечивать ряд типовых функциональных возможностей, включающих:

* подключение к базе данных;
* считывание информации из таблиц этой базы данных;
* редактирование данных и навигация по набору данных.

Обращение к базам данных производится с помощью специальных компонентов, использующих функции BDE:

TTable – обеспечивает доступ к таблицам локальных баз данных и управление ими;

TQuery – использует для доступа к базе данных SQL-запросы, поэтому позволяет работать как с локальными, так и с удаленными базами данных.

Любые прикладная программа на Delphi, желающая использовать БД, должна иметь в своем составе, по меньшей мере, три слоя компонентов.

Табличная "сетка"

TDBGrid

Навигация

TDBNavigator

Визуальные компо­ненты (пользова­тель­ский интер­фейс)

TDataSource

TDataSource

Промежуточный слой

TTable

TQuery

Доступ к БД

BDE

BDE

…

Рис. 15.2. Схема слоев компонентов в прикладной программе

Простая форма предназначена для просмотра и обновления единственной таблицы.

Для такой формы нужны следующие компоненты:

TQuery (или TTable) - 1 шт.

TDataSource - 1 шт.

TDBGrid - 1 шт.

TDBNavigator - 1 шт.

*Основные свойства этих компонент*

|  |  |  |
| --- | --- | --- |
| **Свойство** | **Значение** | **Комментарий** |
| **TQuery** | | |
| RequestLive | True | Можно обновлять таблицу в сетке и передавать обновле­ния в БД |
| DatabaseName | Student | Условное имя БД (alias) |
| Name | Q1 | Имя этого компонента |
| SQL | Select \* from P | SQL - оператор |
| **TDataSource** | | |
| DataSet | Q1 | Имя компонента TQuery, связанного с БД |
| Name | DS1 | Имя этого компонента |
| **TDBGrid** | | |
| DataSource | DS1 | Имя компонента TDataSource |
| **TDBNavigator** | | |
| DataSource | DS1 | Имя компонента TDataSource |
| VisibleButtons | nbFirst, … | Список разрешенных кнопок для перемещения по таблице |

Компоненты верхних слоев ссылаются на имена компонентов в нижних слоях.

Компонент TTable предназначен для доступа к единственной таблице файл-серверной СУБД, а TQuery - к любому количеству таблиц в клиент-серверной СУБД.

Обычно TTable используется в навигационном режиме для обработки единичных строк в таблице, а TQuery - для групповых операций с помощью SQL.

Оба компонента могут открыть набор данных методом Open или двоичным свойством Active и закрыть методом Close или тем же свойством Active.

Для просмотра и изменения НД обычно используются компоненты: TDBNavigator, TDBGrid и др., которые автоматически вызывают нужные методы компонентов TTable и TQuery.

Примеры программного доступа к НД:

Пример 1: Изменение в БД

Q1 : TQuery;

...

Q1.Open; // Открыть НД - выполнить оператор SQL SELECT

...

Q1.Edit; // Перевести в режим редактирования

Q1.FieldByName('gorod').AsString := 'Москва';

// Столбец 'Gorod' интерпретируется как строка

// Клетке присваивается значение 'Москва'

Q1.Post; // Запомнить изменения в БД

...

Q1.Close; // Закрыть НД

Существуют другие типы-свойства:

AsBoolean : Boolean;

AsDataTime : TDataTime;

AsFloat : double;

AsInteger : integer;

Пример 2: Изменение БД с подтверждением

if Q1.Modified then // НД был изменен?

if MessageDlg('Записать?', mtConfirmation[mbYes,mbNo],0) = mrYes then

Q1.Post; // Запомнить изменения в БД

Свойство Modified принимает значение true, если хотя бы одна клетка НД была изменена пользователем.

Пример 3: Подсчет суммы значений в столбце

var

sum : integer;

begin

sum := 0;

Q1.Open;

Q1.First; // Переход к 1-ой строке

while not Q1.EOF do

begin

sum := sum + Q1.FieldByName('ves').AsInteger;

Q1.Next;

end;

Q1.Close;

...

end;

Некоторые особенности класса TQuery

1) Свойство SQL может содержать только один оператор SQL, хотя в нем могут упоминаться несколько таблиц.

2) Операторы SELECT из этого свойства выполняются методом Open, а операторы INSERT, DELETE и UPDATE методом ExecSQL.

Например:

а)

with Query1 do

begin

Close; { закрыть НД }

SQL.Clear; { удалить предыдущий оператор SQL }

SQL.Add('SELECT \* FROM P'); { задать новый оператор SQL }

Open; { будет выполнять этот оператор SQL }

end;

б)

with Query1 do

begin

Close; { закрыть НД }

SQL.Clear; { удалить предыдущий оператор SQL }

SQL.Add('DELETE FROM S'); { задать новый оператор SQL }

ExecSQL; { выполняет другой оператор SQL, помимо SELECT }

end;

3) Свойство RequestLive задает желание изменять данные в БД путем навигации и правки НД, т.е. правки результата оператора SELECT.

Однако истина (true) в свойстве RequestLive не гарантирует, что правка будет разрешена. Для разрешения требуется, кроме того:

а) обращаться только к одной таблице;

б) не применять часть ORDER BY;

в) не применять функции вида SUM, AVG и пр.;

г) не применять вычисляемые столбцы.

4) При любых значениях свойств RequestLive и CanModify изменять БД можно операторами UPDATE, INSERT, DELETE и методом ExecSQL.

5) Изменяющийся оператор SQL можно построить двумя способами:

а)

var

x : string[10];

...

with Query1 do

begin

Close; { закрыть НД }

SQL.Clear; { удалить предыдущий оператор SQL }

SQL.Add('SELECT \* FROM P WHERE ves='+x); { задать новый оператор SQL }

Open; { будет выполнять этот оператор SQL }

end;

x - строковая переменная, содержащая заданный пользователем вес детали.

б) Пусть свойство SQL содержит оператор с двумя параметрами:

SELECT \* FROM P WHERE gorod=:Gor AND ves=:Massa

тогда с помощью метода ParamByName можно дать параметру некоторое значение.

Пример:

var

Town, x : string[30];

...

with Query1 do

begin

Close; { закрыть НД }

SQL.Clear;

SQL.Add('SELECT \* FROM P WHERE gorod=:Gor AND ves=:Massa');

ParamByName('Gor').Value := Town;

ParamByName('Massa').Value := x;

Open; { будет выполнять оператор SQL }

end;

6) Пусть свойство SQL содержит оператор, который возвращает одиночный результат. Например: SELECT SUM(kapital) FROM S;

Этот результат удобно извлекать с помощью метода FieldByName.

Пример:

var

Y : string[30];

...

with Query1 do

begin

Close;

Open;

Y := FieldByName('SUM(kapital)').Value; { или .AsString }

Label1.Caption := Y;

end;

Компоненты для односвязной формы

Такая форма содержит одну связь и две таблицы: главную и подчиненную.

Таблицы связаны между собой через столбцы, имеющие одинаковые значения в строках. Связь между таблицами возможна только по тем столбцам, которые имеют индексы.

Для такой формы нужны следующие компоненты:

TTable - для каждой таблицы по 1 шт.

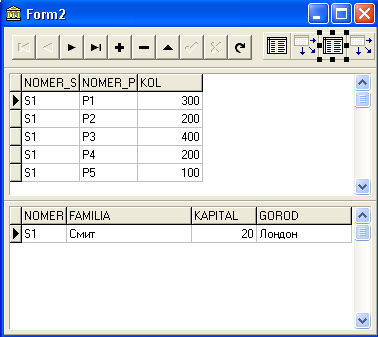
TDataSource - для каждой таблицы по 1 шт.

TDBGrid - для каждой таблицы по 1 шт.

TDBNavigator - 1 шт. (общая)

*Основные свойства этих компонент*

|  |  |  |
| --- | --- | --- |
| **Свойство** | **Значение** | **Комментарий** |
| **T1:TTable (главная таблица)** | | |
| DatabaseName | Student | Условное имя БД (alias) |
| Name | T1 | Имя этого компонента |
| TableName | SP | Имя таблицы (поставок) |
| **DS1:TDataSource (главная таблица)** | | |
| DataSet | T1 | Имя компонента TTable, связанного с БД |
| Name | DS1 | Имя этого компонента |
| **Grid1:TDBGrid (главная таблица)** | | |
| DataSource | DS1 | Имя компонента TDataSource |
| **T2:TTable** | | |
| DatabaseName | Student | Условное имя БД (alias) |
| Name | T2 | Имя этого компонента |
| TableName | S | Имя таблицы (поставщиков) |
| MasterSource | DS1 | Имя компонента TDataSource, связанного с глав­ной таблицей (источник данных главной таблицы) |
| IndexFieldNames | Nomer | Имя столбца индекса в подчиненной таблице |
| MasterFields | Nomer\_s | Имя связующего столбца из главной таблицы |
| **DS2:TDataSource (подчиненная таблица)** | | |
| DataSet | T2 | Имя компонента TTable, связанного с БД |
| Name | DS2 | Имя этого компонента |
| **Grid2:TDBGrid (подчиненная таблица)** | | |
| DataSource | DS2 | Имя компонента TDataSource |
| **TDBNavigator** | | |
| DataSource | DS1 | Имя главного компонента TDataSource |



# Тема 16. Обработка исключительных ситуаций

Исключительная ситуация – это событие, прерывающее нормальное выполнение программы. Иначе говоря, исключительная ситуация является ошибкой, возникающей во время выполнения программы.

В Object Pascal определены две конструкции для работы с исключительными ситуациями.

**Блок try … except**

Блок try … except используется для реакции на конкретную исключительную ситуацию.

try

statement1;

statemant2;

…

except

on Exception1 do statemant3;

on Exception2 do statemant4;

…

else statementN

end;

Если при выполнении операторов, расположенных в разделе *try*, не возникает исключительная ситуация, то обращения к разделу *except* вообще не происходит.

Если в разделе *try* возникает исключительная ситуация, то управление сразу передается разделу *except*.

Раздел *except* содержит набор операторов *on … do*, определяющих реакцию на исключительные ситуации. Между ключевыми словами *on* и *do* указывается имя класса исключительной ситуации. Оператор, расположенный после слова *do*, предназначен для ее обработки.

Исключительные ситуации, возникающие во время выполнения программы, описываются в языке Object Pascal с помощью специального объектного типа Exception. На базе этого типа определен ряд дочерних классов, соответствующих наиболее типичным исключительным ситуациям.

procedure TForm1.Button1Click(Sender: TObject);

var a,b,c: double;

begin

TRY

a:=StrToInt(Edit1.Text);

b:=StrToInt(Edit2.Text);

c:=a/b;

ShowMessage ('Результат ='+FloatToStr(c));

EXCEPT on EZeroDivide do ShowMessage ('Ошибка деления на 0');

on EConvertError do ShowMessage ('Неверный формат числа');

END;

end;

**Блок try … finally**

Блок try … finally используется в тех случаях, когда необходимо выполнить некоторые действия даже в случае возникновения исключительной ситуации.

try

statement1;

statement2;

…

finally

statement3;

statement4;

…

statementN;

end;

В данной конструкции сначала выполняются операторы, расположенные в разделе *try*. Если при их выполнении не возникло исключительной ситуации, то выполняются операторы, расположенные в разделе *finally*. Если же при выполнении операторов в разделе *try* возникает исключительная ситуация, то управление сразу передается первому оператору раздела *finally*.