I. ОБЩЕЕ ПРЕДСТАВЛЕНИЕ ОБ ИНФОРМАЦИОННОЙ СИСТЕМЕ

В уже достаточно долгой истории компьютерной индустрии можно выделить два основных направления: вычисления, а также накопление и обработка информации. Как известно, возникновение компьютеров главным образом стимулировалось необходимостью проведения массивных расчетов для создания ядерного оружия и ракетной техники. Объемы требуемых вычислений просто не позволяли произвести их в приемлемое время традиционным коллективом расчетчиков. Итак, первыми пользователями компьютеров и разработчиками компьютерных программ стали вычислительные математики.

Однако почти сразу на появление компьютеров обратили внимание бизнесмены. Как правило, в гражданском бизнесе не требуются массивные расчеты за исключением таких отраслей как, например, авиа- или автомобилестроение. В более распространенных видах гражданского бизнеса (банковское дело, биржевые операции, системы резервирования билетов или мест в гостиницах) основной проблемой всегда являлись объемы информации, которую необходимо собирать, надежно хранить и оперативно обрабатывать. Появление информационных систем, основным назначением которых является решение отмеченной проблемы, явилось ответом компьютерной индустрии на требования мира бизнеса.

1.1 Специфика и задачи информационных систем

В зависимости от конкретной области применения информационные системы могут очень сильно различаться по своим функциям, архитектуре, реализации. Однако можно выделить, по крайней мере, два свойства, которые являются общими для всех информационных систем.

Во-первых, любая информационная система предназначена для сбора, хранения и обработки информации. Поэтому в основе любой информационной системы лежит среда хранения и доступа к данным. Среда должна обеспечивать уровень надежности хранения и эффективность доступа, которые соответствуют области применения информационной системы. Заметим, что в вычислительных программных системах наличие такой среды не является обязательным. Основным требованием к программе, выполняющей численные расчеты (если, конечно, говорить о решении действительно серьезных задач), является ее быстродействие.

Во-вторых, информационные системы ориентируются на конечного пользователя, например, банковского клерка. Такие пользователи могут быть очень далеки от мира компьютеров. Для них терминал, персональный компьютер или рабочая станция представляют собой всего лишь орудие их собственной профессиональной деятельности. Поэтому информационная система обязана обладать простым, удобным, легко осваиваемым интерфейсом, который должен предоставить конечному пользователю все необходимые для его работы функции, но в то же время не дать ему возможность выполнять какие-либо лишние действия.

Конкретные задачи, которые должны решаться информационной системой, зависят от той прикладной области, для которой предназначена система. Области применения информационных приложений разнообразны: банковское дело, страхование, медицина, транспорт, образование и т.д. Трудно найти область деловой активности, в которой сегодня можно было обойтись без использования информационных систем. С другой стороны, очевидно, что, например, конкретные задачи, решаемые банковскими информационными системами, отличаются от задач, решение которых требуется от медицинских информационных систем.

Можно выделить некоторое количество задач, не зависящих от специфики прикладной области. Естественно, такие задачи связаны с общими чертами информационных систем, рассмотренных в предыдущем разделе.

Прежде всего, кажется бесспорным мнение о том, что наиболее существенной составляющей является информация, которая долго накапливается и утрата которой невосполнима. Уровень надежности и продолжительность хранения информации во многом определяются конкретными требованиями корпорации к информационной системе.

Следующей задачей, которую должно выполнять большинство информационных систем, — это хранение данных, обладающих разными структурами. Трудно представить себе более или менее развитую информационную систему, которая работает с одним однородным файлом данных. Более того, разумным требованием к информационной системе является то, чтобы она могла развиваться. Могут появиться новые функции, для выполнения которых требуются дополнительные данные с новой структурой. При этом вся накопленная ранее информация должна остаться сохранной. Теоретически можно решить эту задачу путем использования нескольких файлов внешней памяти, каждый из которых хранит данные с фиксированной структурой.

При использовании такого подхода информационная система перегружается деталями организации хранилища данных. При выполнении функций уровня пользовательского интерфейса информационной системе самой приходится выполнять выборку информации из файлов по заданному критерию. Некоторые системы управления файлами позволяют выбирать записи по простому критерию, например, по заданному значению ключа записи. В результате развития большинства таких систем в них выделился отдельный компонент, который представляет собой примитивную разновидность системы управления базами данных (СУБД). Самодельные СУБД — главный бич информационных систем.

До сих пор мы говорили о тех функциях информационной системы, которые требуют выборки данных из внешнего хранилища, например, производят отчеты. Но откуда берутся данные во внешнем хранилище? Каким образом поддерживается соответствие хранимой информации состоянию предметной области? Конечно, для этого должны существовать дополнительные функции информационной системы, которые обеспечивают ввод, обновление и удаление данных.

Если говорить о групповых или корпоративных информационных системах, то их наличие предполагает возможность работы с системой с нескольких рабочих мест. Некоторые из конечных пользователей изменяют содержимое базы данных (вводят, обновляют, удаляют данные). Другие выполняют операции, связанные с выборкой из базы данных. Третьи делают и то, и другое. Вся проблема состоит в том, что такая коллективная работа должна производиться согласованно и желательно, чтобы согласованность действий обеспечивалась автоматически. Под согласованностью действий мы понимаем то, что оператор, формирующий отчеты, не сможет воспользоваться данными, которые начал, но еще не закончил формировать другой оператор. Оператор формирующий данные, не сможет выполнить операцию над данными, которыми пользуется другой оператор, начавший, но не закончивший формировать отчет. Оператор, желающий обновить или удалить данные, не сможет выполнить операцию до тех пор, пока не закончится аналогичная операция над теми же данными, которую ранее начал, но еще не закончил другой оператор. При поддержке согласованности действий все результаты, получаемые от информационной системы, будут соответствовать согласованному состоянию базы данных, т.е. будут достоверны и непротиворечивы.

Подобные рассуждения вызвали появления понятия **классической транзакции**. Будем понимать под целостным состоянием базы данных информационной системы такое ее

состояние, которое соответствует требованиям прикладной области (или, вернее, требованиям модели прикладной области, на основе которой проектировалась информационная система). Тогда классической транзакцией называется последовательность операций изменения базы данных и/или выборки из базы данных, воспринимаемая СУБД как атомарное действие. Это означает, что при успешном завершении транзакции СУБД гарантирует наличие в базе данных результатов всех операций изменения, произведенных при выполнении транзакции. Условием успешного завершения транзакции является то, что база данных находится в целостном состоянии. Если это условие не выполняется, то СУБД производит полный откат транзакции, ликвидируя в базе данных результаты всех операций изменения, произведенных при выполнении транзакции. Тем самым, легко видеть, что база данных будет находиться в целостном состоянии при начале любой транзакции и останется в целостном состоянии после успешного завершения любой транзакции.

Все развитые СУБД поддерживают понятие транзакции. Если информационная система базируется на СУБД такого класса, то для обеспечения согласованности действий параллельно работающих конечных пользователей достаточно при проектировании системы правильно связать операции информационной системы с транзакциями СУБД. Это относится уже к области проектирования, и мы подробно обсудим проблемы в следующих частях курса.

В корпоративных информационных системах по естественным причинам часто возникает потребность в распределенном хранении общей базы данных. Например, разумно хранить некоторую часть информации как можно ближе к тем рабочим местам, в которых она чаще всего используется. По этой причине при построении информационной системы приходится решать задачу согласованного управления распределенной базой данных (иногда применяя методы репликации данных). При однородном построении распределенной базы данных (на основе однотипных серверов баз данных) эту задачу обычно удается решить на уровне СУБД (большинство производителей развитых СУБД поддерживает средства управления распределенными базами данных). Если же система разнородна (т.е. для управления отдельными частями распределенной базы данных используются разные серверы), то приходится прибегать к использованию вспомогательных инструментальных средств интеграции разнородных баз данных типа мониторов транзакций.

Традиционным методом организации информационных систем является двухзвенная архитектура "клиент-сервер" (рисунок 1.1). В этом случае вся прикладная часть информационной системы выполняется на рабочих станциях системы (т.е. дублируется), а на стороне сервера(ов) осуществляется только доступ к базе данных. Если логика прикладной части системы достаточно сложна, то такой подход порождает проблему "толстого" клиента. Каждая рабочая станция должна обладать достаточным набором ресурсов, чтобы быть в состоянии произвести прикладную обработку данных, поступающих от пользователя и/или из базы данных. Для того, чтобы клиенты могли быть "тощими", а зачастую и для повышения общей эффективности системы, все чаще применяются трехзвенные архитектуры "клиент-сервер" (рисунок 1.2). В этой архитектуре, кроме клиентской части системы и сервера(ов) базы данных, вводится промежуточный сервер приложений. На стороне клиента выполняются только интерфейсные действия, а вся логика обработки информации поддерживается в сервере приложений.

Рис. 1.1. Традиционная двухзвенная архитектура "клиент-сервер"

Рис. 1.2. Трехзвенная архитектура "клиент-сервер" с выделенным сервером приложений

Заметим, что некоторые черты трехзвенности могут присутствовать и в двухзвенной архитектуре. Если, например, используемый сервер баз данных поддерживает развитый механизм хранимых процедур, то можно перебросить некоторую часть логики приложения на сторону баз данных. Заметим, что механизм хранимых процедур недостаточно полно специфицирован в стандарте языка SQL.

И наконец, еще один класс задач относится к обеспечению удобного и соответствующего целям информационной системы пользовательского интерфейса. На первый взгляд эта задача кажется не очень существенной. Можно полагать, что если информационная система обеспечивает полный набор функций и ее интерфейс обеспечивает доступ к любой из этих функций, то конечные пользователи должны быть удовлетворены. На самом деле, это не так. Пользователи часто судят о качестве системы в целом, исходя из качества ее интерфейса. Более того, эффективность использования системы зависит от качества интерфейса.

1.2 Проблемы построения ИС

Самой первой проблемой является проблема проектирования. Нельзя начинать техническую разработку, не имея тщательно проработанного проекта. Если начинать с решения наиболее очевидных задач, не обращая внимания на потенциально существующие, то такая система будет непрерывно находиться в стадии разработки и переделки.

Проектирование проводится в несколько стадий:

1. Первой стадией проектирования должен быть анализ требований корпорации. Для этого на основе экспертных запросов необходимо выявить все актуальные и потенциальные потребности корпорации, которые должны удовлетворяться проектируемой информационной системой, понять какие потоки данных существуют внутри корпорации, оценить объемы информации, которые должны поддерживаться и обрабатываться информационной системой. Эта стадия, как правило, носит неформальный характер, хотя, конечно, очень важно сохранить полученную информацию, поскольку она должна входить в документацию системы. Существуют CASE-средства верхнего уровня, которые помещают полученные данные в общий репозиторий проекта и позволяют использовать их на следующих стадиях проектирования.

2. Следующая стадия проектирования — выработка концептуальной схемы базы данных, которая будет лежать в основе информационной системы. Сначала придется выбрать систему *нотаций*, в которой будет представляться концептуальная схема (правильнее было бы сказать — выбрать концептуальную модель). Таких нотаций существует великое множество, и хотя практически все они диаграммные, они отличаются одна от другой. Концептуальное представление базы данных должно сохраняться как часть документации информационной системы на все время ее существования и будет использоваться при ее сопровождении и развитии.

3. Вырабатывается общая реляционная схема базы данных

   На этой же стадии необходимо решить, какие таблицы будут реально хранимыми, а какие — представляемыми (view).

4. Теперь необходимо определиться с архитектурой системы. В частности, очень важно решить, какой будет база данных — централизованной или распределенной (другими

словами будет ли использоваться только один сервер баз данных или их будет несколько). Если принимается решение о распределенном характере базы данных, то необходимо произвести соответствующую декомпозицию набора определений схемы базы данных. (Заметим, что принятие решения об архитектуре системы возможно и до выработки общей реляционной схемы базы данных. Тогда декомпозиция производится на уровне концептуальной схемы, а затем для каждой отдельной части концептуальной схемы создается реляционная схема в терминах языка SQL.)

Наиболее простым случаем декомпозиции является тот, когда образующиеся разделы базы данных логически автономны. В терминах концептуальной схемы это означает, что между разделенными сущностями отсутствуют прямые или транзитивные (через другие сущности) связи. В терминах реляционной схемы: ни в одном разделе не присутствует таблица, ссылающаяся на таблицу, которая располагается в другом разделе (рисунки 1.3 и 1.4). Если требование логической автономности компонентов распределенной базы данных выполнено, то дальнейшее проектирование можно производить для каждого компонента независимо.

Рис. 1.3. Распределенная база данных с логически автономными разделами

Рис. 1.4. Распределенная база данных с логически неавтономными разделами

В чем, собственно, состоит проблема распределенных баз данных с логически неавтономными разделами? Все дело в том, что любая связь, отраженная в схеме базы данных (между сущностями в концептуальной модели или между таблицами в реляционной модели), соответствует ограничению целостности, которое впоследствии должно сохраняться в базе данных и поддерживаться СУБД. В настоящее время общая технология организации распределенных баз данных (даже однородных, основанных на SQL) отсутствует. Некоторые производители решают эту задачу путем расширения функций сервера, и тогда, вообще говоря, в базе данных могут присутствовать ограничения целостности, содержащие ссылки на таблицы, которые располагаются в других разделах. В других системах задача управления распределенной базой данных решается на стороне клиента. В этом случае сервер, управляющий разделом распределенной базы данных ничего не знает о существовании других разделов и не может поддерживать ограничения целостности, включающие ссылки на объекты "чужих" разделов. Тогда целостность распределенной базы данных приходится поддерживать за счет написания явного кода в приложении. Другими словами, если невозможно произвести декомпозицию схемы общей базы данных в набор схем логически независимых разделов, то снова необходимо учитывать возможности используемого серверного продукта и двигаться дальше соответствующим образом.

5. Следующая стадия проектирования состоит в дополнении реляционных схем разделов распределенной базы данных определениями общих ограничений целостности, триггеров и хранимых процедур. На каждом из этих компонентов схемы следует остановиться отдельно. Начнем с общих ограничений целостности.

   Язык SQL-92 позволяет определить ограничения целостности, относящиеся к общему состоянию базы данных и включающие ссылки на произвольное число таблиц. Семантика таких ограничений целостности может быть существенно шире, чем

ограничения, задаваемые связями **1** к **n** и даже **n** к **m** (**один-ко-многим** и **многие-ко-многим**). Поэтому часто ограничения общего вида не выводятся автоматически из концептуальной схемы базы данных и их приходится добавлять к реляционной схеме вручную. Для того, чтобы понять, какие ограничения общего вида должны быть включены в реляционные схемы разделов, приходится возвращаться к документу, содержащему анализ требований корпорации. Задача проектировщика состоит в том, чтобы, с одной стороны, выявить все необходимые ограничения целостности и, с другой стороны, не перегрузить базу данных необязательными ограничениями. Конечно, если предполагается использование распределенной базы данных, то опять придется учитывать возможности сервера по части ссылок на объекты "чужих" разделов. Это может повлиять на выбор ограничений целостности и/или повлечь создание новой декомпозиции общей базы данных на разделы.

Триггеры

Триггер — это хранимая процедура без параметров, содержащая оператор(ы) изменения базы данных и вызываемая сервером баз данных автоматически при совершении некоторого события

Под *событием* понимается выполнение определенного рода операторов изменения базы данных.

Более точно, при определении триггера указывается вид события, возбуждающего триггер, условие его срабатывания и набор операторов, которые должны быть выполнены при выполнении условия. Имеется несколько практических областей применения механизма триггеров. Во-первых, триггеры позволяют поддерживать логическую целостность базы данных в тех случаях, когда пользовательская транзакция не может не нарушить эту целостность.

Вторая область применения триггеров — автоматический мониторинг действий конечных пользователей. Приведем очень простой пример. Начальник отдела кадров хочет знать, сколько операций в день выполняет его клерк. Тогда он должен сказать об этом на стадии анализа требований, и в результате, например, будет создана таблица

```
СТАТИСТИКА (ТЕКУЩАЯ_ДАТА, ЧИСЛО_ОПЕРАЦИЙ)
```

и триггер вида

```
ON INSERT, DELETE FROM СОТРУДНИКИ
IF EXISTS (SELECT * FROM СТАТИСТИКА WHERE ТЕКУЩАЯ_ДАТА = TODAY)
   UPDATE СТАТИСТИКА SET
      NEW (ЧИСЛО_ОПЕРАЦИЙ) = OLD (ЧИСЛО_ОПЕРАЦИЙ) + 1
   WHERE ТЕКУЩАЯ_ДАТА = TODAY
ELSE
   INSERT INTO СТАТИСТИКА (TODAY, 1)
```

Здесь необходимо сделать два замечания.

Во-первых, в распространенном на сегодня стандарте SQL-92 механизм триггеров не специфицирован. В СУБД, которые поддерживают этот механизм, соответствующие языковые средства и их семантика различаются. Поэтому на текущий момент использование механизма триггеров неявно влечет сильную привязку к конкретному производителю.

Во-вторых, как и в случае определения представлений и ограничений целостности, при определении триггеров необходимо учитывать распределенный характер базы данных и возможности сервера ссылаться на "чужие" таблицы.

Хранимые процедуры

Наконец, в базе данных могут храниться *хранимые процедуры*. Интересно, что в стандарте SQL-92 вообще не встречается термин "хранимая процедура". В стандарте специфицированы два способа взаимодействия прикладной программы с сервером баз данных. Первый, наиболее часто используемый способ состоит в встраивании операторов языка SQL в программу, написанную на одном из традиционных языков программирования. В самом стандарте определены правила встраивания SQL в программы, которые написаны на языках Си, Паскаль, Фортран, Ада и т.д. Второй способ основан на специфицированном в стандарте "языке модулей SQL". С использованием этого языка можно определить модуль, содержащий несколько процедур, каждая из которых соответствует некоторому параметризованному оператору SQL. В прикладной программе содержатся не операторы SQL, а лишь вызовы процедур (с указанием фактических параметров) из модуля SQL, с которым эта прикладная программа связана (правила связывания в стандарте не определены). Заметим, что стандарт не обязывает следовать каким-то конкретным правилам при реализации встроенного SQL или языка модулей. Посмотрим, что же делается в реализациях.

Что касается встроенного SQL, то во всех реализациях прикладная программа, содержащая операторы SQL, подвергается прекомпиляции с помощью соответствующего программного продукта, который входит в состав программного обеспечения сервера баз данных. В результате прекомпиляции всегда формируется текст прикладной программы на используемом языке программирования, уже не содержащий напрямую текста на языке SQL. Каждое вхождение оператора SQL в исходный текст программы заменяется на вызов некоторой процедуры или функции в синтаксисе включающего языка программирования. Основная разница между разными реализациями состоит в том, что из себя представляет эта процедура или функция. В большинстве систем (Oracle, Informix, Sybase, CA-OpenIngres) такая процедура представляет собой обращение к клиентской части СУБД с передачей в качестве параметра текста оператора на языке SQL. Вся обработка оператора, включая его грамматический разбор и выработку процедурного плана производится сервером во время выполнения прикладной программы (конечно, при повторном выполнении оператора сервер использует уже подготовленный план). Однако имеется и другой подход, исторически используемый компанией IBM в ее серии продуктов DB2. В DB2 прекомпилятор, выбрав текст очередного оператора SQL, сам обращается к серверу с запросом на компиляцию оператора. Сервер выполняет грамматический разбор и строит процедурный план выполнения оператора, сохраняя этот план в базе данных и привязывая к соответствующей прикладной программе. В качестве ответа от сервера прекомпилятор получает идентификатор, указание которого серверу повлечет выполнение соответствующего плана (грубо говоря, имя удаленной процедуры). Тогда в тексте прикладной программы появится вызов процедуры-переходника (stub), которая жестко привязана к соответствующей удаленной процедуре. Другими словами, более распространен метод динамической компиляции встроенного SQL (при выполнении прикладной программы), но имеются примеры реализаций со статической компиляцией встроенных операторов SQL (при прекомпиляции текста прикладной программы со встроенным SQL) (таблица 1.1.). В этом курсе мы не будем рассматривать сравнительные преимущества и недостатки динамического и статического подходов.

Таблица 1.1.

Характеристики динамической и статической схем обработки встроенного SQL

Вид подхода	Прекомпиляция	Выполнение
Динамическая схема	На стороне клиента	Компиляция и выполнение на стороне сервера
Статическая схема	На стороне клиента, компиляция на стороне сервера	Выполнение на стороне сервера

Идея же процедур SQL, определяемых на языке модулей, повлекла внедрение во многие реализации механизма хранимых процедур. Хранимая процедура — это именованная, параметризованная конструкция, определяемая на языке SQL или некотором его расширении, встраиваемая в прикладную программу, компилируемая на стороне сервера во время обработки текста процедуры прекомпилятором. Выполняемый или интерпретируемый код хранимой процедуры сохраняется в базе данных, а сама она может быть вызвана из любой прикладной программы авторизованного пользователя (того, кто получил привилегию на выполнение этой процедуры) с указанием фактических параметров (рисунок 1.5). Пожалуй, механизмы хранимых процедур возникли прежде всего для того, чтобы обеспечить некоторое подобие статической компиляции встроенных операторов SQL в СУБД, которые поддерживают динамическую схему. Компания Oracle пошла дальше, разработав процедурное расширение SQL - язык PL/SQL. С использованием этого языка можно отсылать на сервер компоненты логики приложения. (Похоже, что в следующем стандарте языка SQL - SQL-3 - появится нечто похожее на PL/SQL.) Как мы отмечали выше, фактически это можно трактовать как первое приближение к трехзвенной организации информационной системы. Как правило, развитые СУБД поддерживают язык модулей SQL, но язык хранимых процедур обычно шире и не стандартизован. Поэтому с хранимыми процедурами нужно обращаться осторожно, если вы не хотите попасть в полную зависимость от конкретного производителя. Кроме того, в случае использования распределенной базы данных следует внимательно отслеживать возможности ссылок на таблицы, размещающиеся в разных разделах.

Рис. 1.5. Подготовка и использование хранимой процедуры

Логическое проектирование базы данных информационной системы на этом можно считать законченным. Мы можем приступать к следующим стадиям. По крупному, их осталось две: физическое проектирование базы данных; проектирование и разработка интерфейсов и обрабатывающей части прикладной системы. Эти две стадии могут выполняться параллельно.

Физическое проектирование ИС

Физическое проектирование включает два основных шага, первый из которых, как правило, не зависит от особенностей выбранного серверного SQL-ориентированного продукта, а второй зависит, причем критически. На первом шаге этой стадии определяется набор требуемых индексов. Для того, чтобы правильно выбрать этот набор, необходимо еще раз

тщательно проанализировать требования корпорации и оценить, какие запросы будут выполняться наиболее часто. Это непростая задача, но нужно учитывать, что создание индекса на большой заполненной таблице (когда информационная система уже функционирует) требует значительного времени. Нужно также иметь в виду, что операторы определения (создания) и уничтожения индексов не определены в стандарте SQL. Хотя имеется негласное соглашение между производителями SQL относительно вида этих операторов, они все же могут различаться в разных системах. Кроме того, следует иметь в виду, что в большинстве систем индексы автоматически создаются для полей таблицы, которые объявлены первичным, возможными или внешними ключами.

Второй шаг состоит в определении областей внешней памяти, в которых будут храниться фрагменты базы данных. По этому поводу вообще невозможно дать какие-либо общие рекомендации, поскольку вопрос непосредственного размещения данных во внешней памяти является специфическим для каждой конкретной СУБД. Если нет человека, имеющего опыт администратора данной системы, единственный выход состоит во внимательном изучении руководства администратора. Обычно эти руководства содержат по меньшей мере описание того, как СУБД работает с внешней памятью. Конечно, окончательное решение остается на совести проектировщика.

Целесообразно выполнять работы, связанные с физическим проектированием, совместно со специалистом, который в дальнейшем будет выполнять функции администратора базы данных. Если на стадии физического проектирования такой специалист еще не назначен, необходимо тщательно документировать детали физического проекта.

Разработка интерфейса

Параллельно с физическим проектированием базы данных информационной системы может проводиться проектирование и разработка интерфейса системы и ее обрабатывающей части. В принципе, к этому моменту уже должно быть ясно, что вы хотите от интерфейса и какие функции должна выполнять система. Так что основной проблемой этой стадии является выбор инструментальных средств, которые позволят достаточно быстро произвести достаточно эффективную реализацию. Мы недаром два раза употребили слово "достаточно". Нужно понять, что важнее для корпорации — как можно быстрее ввести информационную систему в действие или добиться требуемого уровня эффективности.

Понятно, что в любом случае в наше время неразумно программировать интерфейсные функции вручную. Нужно использовать имеющиеся полуфабрикаты. Выбор определяется вкусом и привычками разработчика, а также общей ориентацией проекта. Например, если с самого начала проект был ориентирован на использование продуктов компании Oracle и ведется в интегрированной среде Designer/Developer 2000, то было бы странно покидать эту среду при разработке интерфейсов.

Что же касается обрабатывающих частей информационной системы, то все зависит от того, что они должны делать. Имеется масса примеров информационных систем, в которых вся обработка данных состоит в преобразовании их форматов при вводе и выводе, формировании отчетов и т.д. Такие функции можно фактически не программировать. Но если требуется более сложная обработка данных, то потребуется явное программирование, и тогда уже неважно, на каком языке это программирование будет вестись.

Последнее замечание этого раздела. Мы описали одну из возможных схем проектирования и разработки информационной системы, представив весь процесс в виде последовательно выполняемых стадий. Если проект основывается на некотором интегрированном CASE-средстве с единым репозитарием, то эти стадии, вообще говоря, могут

перемешиваться. Возможности использования интегрированных инструментальных средств при проектировании, сопровождении и реинжиниринге информационных систем будут рассмотрены в следующих частях курса.