Лекция 1 Информационные системы и их классификации

Основные понятия

Система (от греческого *systema* — целое, составленное из частей соединение) — это совокупность элементов, взаимодействующих друг с другом, образующих определенную целостность, единство. Приведем некоторые понятия, часто использующиеся для характеристики системы.

1. Элемент системы — часть системы, имеющая определенное функциональное назначение. Сложные элементы систем, в свою очередь состоящие из более простых взаимосвязанных элементов, часто называют подсистемами.

2. **Организация системы** — внутренняя упорядоченность, согласованность взаимодействия элементов системы, проявляющаяся, в частности, в ограничении разнообразия состояний элементов в рамках системы.

3. **Структура системы** — состав, порядок и принципы взаимодействия элементов системы, определяющие основные свойства системы. Если отдельные элементы системы разнесены по разным уровням и внутренние связи между элементами организованы только от вышестоящих к нижестоящим уровням и наоборот, то говорят об *иерархической структуре* системы. Чисто иерархические структуры встречаются практически редко, поэтому, несколько рас ширяя это понятие, под иерархической структурой обычно понимают и такие структуры, где среди прочих связей иерархические связи имеют главенствующее значение.

4. **Архитектура системы** — совокупность свойств системы, существенных для пользователя.

5. **Целостность системы** — принципиальная несводимость свойств системы к сумме свойств отдельных ее элементов (эмерджентность свойств) и, в то же время, зависимость свойств каждого элемента от его места и функции внутри системы.

Информационная система — взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели»

В Федеральном законе «Об информации, информатизации и защите информации» дается следующее определение:

«Информационная система — организационно упорядоченная совокупность документов (массивов документов) и информационных технологий, в том числе с использованием средств вычислительной техники и связи, реализующих информационные процессы»

Классификация по масштабу

По масштабу информационные системы подразделяются на следующие группы:

- одиночные;
- групповые;
- корпоративные.

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 создайся 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 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 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.

Классификация по сфере применения

По сфере применения информационные системы обычно подразделяются на четыре группы:

- системы обработки транзакций;
- системы принятия решений;
- информационно-справочные системы;

- офисные информационные системы.

Системы обработки транзакций, в свою очередь, по оперативности обработки данных, разделяются на пакетные информационные системы и оперативные информационные системы. В информационных системах организационного управлений преобладает режим оперативной обработки транзакций, для отражения *актуального* состояния предметной области в любой момент времени, а пакетная обработка занимает весьма ограниченную часть.

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* основан 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.

Классификация по способу организации

По способу организации групповые и корпоративные информационные системы подразделяются на следующие классы:

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

В любой информационной системе можно выделить необходимые функциональные компоненты, которые помогают понять ограничения различных архитектур информационных систем.

Архитектура 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.

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.

В настоящее время архитектура клиент-сервер получила признание и широкое распространение как способ организации приложений для рабочих групп и информационных систем корпоративного уровня. Подобная организация работы

повышает эффективность выполнения приложений за счет использования возможностей сервера БД, разгрузки сети и обеспечения контроля целостности данных.

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

- нижний уровень представляет собой приложения клиентов, имеющие программный интерфейс для вызова приложения на среднем уровне;
- средний уровень представляет собой сервер приложений;
- верхний уровень представляет собой удаленный специализированный сервер базы данных.

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.

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.

По характеру хранимой информации БД делятся на *фактографические* и *документальные*. Если проводить аналогию с описанными выше примерами информационных хранилищ, то фактографические БД — это картотеки, а документальные — это архивы. В фактографических БД хранится краткая информация в строго определенном формате. В документальных БД — всевозможные документы. Причем это могут быть не только текстовые документы, но и графика, видео и звук (мультимедиа).

Автоматизированная система управления (АСУ) - это комплекс технических и программных средств, совместно с организационными структурами (отдельными людьми пли коллективом), обеспечивающий управление объектом (комплексом) в производственной, научной или общественной среде.

Выделяют информационные системы управления образования (Например, кадры, абитуриент, студент, библиотечные программы). Автоматизированные системы для научных исследований (АСНИ), представляющие собой программно-аппаратные комплексы, обрабатывающие данные, поступающие от различного рода экспериментальных установок и измерительных приборов, и на основе их анализа облегчающие обнаружение новых эффектов и закономерностей. Системы автоматизированного проектирования и геоинформационные системы.

Систему искусственного интеллекта, построенную на основе высококачественных специальных знании о некоторой предметной области (полученных от экспертов - специалистов этой области), называют экспертной системой. Экспертные системы -

один из немногих видов систем искусственного интеллекта - получили широкое распространение, и нашли практическое применение. Существуют экспертные системы по военному делу, геологии, инженерному делу, информатике, космической технике, математике, медицине, метеорологии, промышленности, сельскому хозяйству, управлению, физике, химии, электронике, юриспруденции и т.д. И только то, что экспертные системы остаются весьма сложными, дорогими, а главное, узкоспециализированными программами, сдерживает их еще более широкое распространение.

Экспертные системы (ЭС) - это компьютерные программы, созданные для выполнения тех видов деятельности, которые под силу человеку-эксперту. Они работают таким образом, что имитируют образ действий человека-эксперта, и существенно отличаются от точных, хорошо аргументированных алгоритмов и не похожи на математические процедуры большинства традиционных разработок.

Дополнительный материал

Предметом изучения информатики являются информационные технологии, которые реализуются на практике в автоматизированных информационных системах (АИС) различного назначения; выступающих в качестве объекта информатики. Таким образом, АИС позволяют автоматизировать ту или иную сферу профессиональной деятельности людей за счет использования компьютерных средств и технологий. Иными словами, в качестве основных средств (инструмента) автоматизации профессиональной деятельности людей сегодня выступают средства ЭВТ и связи.

В качестве основного классификационного признака АИС целесообразно рассматривать особенности автоматизируемой профессиональной деятельности – процесса переработки входной информации для получения требуемой выходной информации, в котором АИС выступает в качестве инструмента должностного лица или группы должностных лиц, участвующих в управлении организационной системой.

В соответствии с предложенным классификационным признаком можно выделить следующие классы АИС:

- автоматизированные системы управления (АСУ);
- системы поддержки принятия решения (СППР);
- автоматизированные информационно-вычислительные системы (АИВС);
- автоматизированные системы обучения (АСО);
- автоматизированные информационно-справочные системы (АИСС).

Рассмотрим особенности каждого класса АИС и характеристики возможных видов АИС в составе каждого класса.

Автоматизированные системы управления

Автоматизированная система управления представляет собой автоматизированную информационную систему, предназначенную для автоматизации всех или большинства задач управления, решаемых коллективным органом управления (министерством, дирекцией, правлением, службой, группой управления и т.д.). В зависимости от объекта управления различают АСУ персоналом и АСУ техническими средствами (АСУП и АСУТС). АСУ является организационной и технической основой реализации рациональной технологии коллективного решения задач управления в различных условиях обстановки. В этой связи разработка рациональной технологии организационного управления является определяющим этапом создания любой АСУ.

АСУП обеспечивает автоматизированную переработку информации необходимой для управления организацией в повседневной деятельности, а также при подготовке и реализации программ развития.

АСУТС предназначены для реализации соответствующих технологических процессов. Они являются по сути перёдаточным звеном между должностными лицами осуществляющими управление техническими системами, и сами ми техническими системами.

В настоящее время АСУТС нашли широкое распространение во всех развитых государствах. Объясняется это тем, что управление существующими новейшими технологический процесс без применения АСУТС становится практически невозможным. Что касается АСУП, то в настоящее′. время такие системы широко используются в странах Запада, и непрерывно ведутся работы по созданию новых систем, в том числе – на базе достижений в области искусственного интеллекта.

Системы поддержки принятия решений

Системы поддержки принятия решений (СППР) являются достаточно новым классом АИС, теория создания которых в настоящее время интенсивно развивается.

СППР называется АИС, предназначенная для автоматизации деятельности конкретных должностных лиц при выполнении ими своих должностных (функциональных) обязанностей в процессе управления персоналом и (или) техническими средствами.

Выделяются четыре категории должностных лиц, деятельность которых отличается различной спецификой переработки информации: руководитель, должностное лицо аппарата управления, оперативный дежурный, оператор. В соответствии с четырьмя категориями должностных лиц различают и четыре вида СППР: СППР руководителя (СППР Р), СППР должностного лица аппарата управления (СППР 0), СППР оперативного дежурного (СППР Д) и СППР оператора (СППР Оп).

Автоматизированные информационно-вычислительные системы

АИВС предназначены для решения сложных в математическом отношении задач, требующих больших объемов самой разнообразной информации. Таким образом видом деятельности автоматизируемом АИВС является проведение различных (сложных и «объемных») расчетов; Эти системы используются для обеспечения научных исследований и разработок, а также как подсистемы АСУ и СППР в тех

случаях, когда выработка управленческих решений должна опираться на сложные вычисления.

В зависимости от специфики области деятельности, в которой используются АИВС, различают следующие в этих систем.

Информационно-расчетные системы

ИРС – это автоматизированная информационная система, предназначенная для обеспечения оперативных расчетов и автоматизации обмена информацией между рабочими местами в пределах некоторой организации или системы организаций. ИРС обычно сопрягается с автоматизированной системой управления и в рамках последней может рассматриваться как ее подсистема. Технической базой ИРС являются, как правило, сети больших, малых и микро-ЭВМ. ИРС имеют сетевую структуру и могут охватывать несколько десятков и даже сотен рабочих мест различных уровней иерархии. Основной сложностью при создании ИРС является обеспечение высокой оперативности расчетов и обмена информации в системе при строгом разграничении доступа должностных лиц к служебной информации.

Системы автоматизации проектирования

САПР – это автоматизированная информационная система, предназначенная для автоматизации деятельности подразделений проектной организации или коллектива специалистов в процессе разработки проектов изделий на основе применения единой информационной базы, математических и графических моделей, автоматизированных проектных и конструкторских процедур. САПР является одной из систем интегральной автоматизации производства, обеспечивающих реализацию автоматизированного цикла создания нового изделия от предпроектных научных исследований до выпуска серийного образца.

В области экономики САПР могут использоваться при проектировании экономических информационных систем и их элементов. Кроме того, технология САПР может обеспечить создание автоматизированной системы отображения обстановки на экране в процессе ведения экономических операций или в ходе деловых игр различных типов.

Проблемно-ориентированные имитационные системы ПОИС предназначены для автоматизации разработки имитационных моделей в некоторой предметной области. Например, если в качестве предметной области взять развитие автомобилестроения, то любая модель, создаваемая в этой предметной области, может включать стандартные блоки, моделирующие деятельность предприятий, поставляющих комплектующие; собственно сборочные производства; сбыт, обслуживание и ремонт автомобилей; рекламу и др. Эти стандартные блоки могут строиться с различной детализацией моделируемых процессов и различной оперативностью расчетов. Пользователь, работая с ПОИС, сообщает ей, какая модель ему нужна (т.е. что необходимо учесть при моделировании и с какой степенью точности), а ПОИС автоматически формирует имитационную модель, необходимую пользователю.

В состав программного обеспечения ПОИС входят банк типовых моделей (БТМ) предметных областей, планировщик моделей, базы данных предметных областей, а также средства диалогового общения пользователя с ПОИС.

ПОИС является достаточно сложной АИС, реализуемой, как правило, с использованием технологии искусственного интеллекта на высокопроизводительных ЭВМ.

Моделирующее центры

МЦ — автоматизированная информационная система представляющая собой комплекс готовых к использованию моделей, объединенных единой предметной областью, информационной базой и языком общения с пользователями.

МЦ, так же как и ПОИС, предназначены для обеспечения проведения исследований на различных моделях. Но в отличие от ПОИС, МЦ не обеспечивают автоматизацию] создания имитационных моделей, а Предоставляют пользователю возможность комфортной работы с готовыми моделями.

МЦ могут являться системами как коллективного, так и индивидуального использования и в принципе не требуют для своей реализации мощных ЭВМ.

Автоматизированные системы обучения

Традиционные методы обучения специалистов в различных областях профессиональной деятельности складывались многими десятилетиями, в течение которых накоплен большой опыт.

Однако, как свидетельствуют многочисленные исследования, традиционные методы обучения обладают рядом недостатков. К таким недостаткам следует отнести пассивный характер устного изложения, трудность организации активной работы студентов, невозможность учета в полной мере индивидуальных особенностей отдельных обучаемых и т.д.

Одним из возможных путей преодоления этих трудностей является создание АСО – автоматизированных информационных систем, предназначенных для автоматизации подготовки специалистов с участием или без участия преподавателя и обеспечивающих обучение, подготовку учебных курсов, управление процессом обучения и оценку его результатов. Основными видами АСО являются автоматизированные системы программированного обучения (АСПО), системы обеспечения деловых игр (АСОДИ), тренажеры и тренажерные комплексы (ТиТК).

АСПО ориентированы на, обучение в основном по теоретическим разделам курсов и дисциплин. В рамках АСПО реализуются заранее подготовленные квалифицированными преподавателями «компьютерные курсы». При этом учебный материал разделяется на порции (дозы) и для каждой порции материала указывается возможная реакция обучаемого. В зависимости от действий обучаемого и его ответов на поставленные вопросы АСПО формирует очередную дозу представляемой информации.

Наибольшую сложность при создании АСПО составляет разработка «компьютерного курса» для конкретной дисциплины. Именно поэтому в настоящее время наибольшее распространение получили «компьютерные курсы» по традиционным, отработанным в методическом плане дисциплинам (физике, элементарной математике, программированию и т.д.).

АСОДИ предназначена для подготовки и проведения деловых игр, сущность которых заключается в имитации принятия должностными лицами индивидуальных и групповых решений в различных проблемных ситуациях путем игры по заданным правилам.

В ходе деловой игры на АСОДИ возлагаются следующие задачи:

- хранение и предоставление обучаемым и руководителям игры текущей информации о проблемной среде в процессе деловой игры в соответствии с их компетенцией;
- формирование по заданным правилам реакции проблемной среды на действия обучаемых;
- обмен информацией между участниками игры (обучаемыми и руководителями игры);
- контроль и обобщение действий обучаемых в процессе деловой игры;
- предоставление руководителям игры возможности вмешательства в ход игры, например, для смены обстановки.

Технической базой АСОДИ являются высокопроизводительные ЭВМ или локальные вычислительные сети. Методологической базой АСОДИ, как правило, является имитационное моделирование на ЭВМ.

ТиТК предназначены для обучения практическим навыкам работы на конкретных рабочих местах (боевых постах). Они являются средствами индивидуального (тренажеры) и группового (тренажерные комплексы) обучения.

ТиТК являются достаточно дорогостоящими средствами обучения, а их создание требует больших затрат времени. Однако их чрезвычайно высокая эффективность при обучении таких специалистов, как летчики, водители, операторы систем управления и т.д., позволяет считать их достаточно перспективными видами АСО.

Автоматизированные информационно-справочные

АИСС — это автоматизированная информационная система, предназначенная для сбора, хранения, поиска и в дачи в требуемом виде потребителям информации справочного характера. В зависимости от характера работы с информацией различают следующие виды АИСС:

- автоматизированные архивы (АА);
- автоматизированные системы делопроизводства (АСД);
- автоматизированные справочники (АС) и картотеки (АК)
- автоматизированные системы ведения электронных карт местности (АСВЭКМ) и др.′

В настоящее время разработано большое количество разновидностей АИСС и их количество продолжает увеличиваться. АИСС создаются с использованием технологий баз данных, достаточно хорошо разработанной и получившей широкое распространение. Для создания АИС как правило, не требуется

высокопроизводительная вычислительная техника.

Простота Создания АИСС и высокий положительный эффект от их использования определили их активное пользование во всех сферах профессиональной (в том числе и управленческой) деятельности.

В процессе развития автоматизированных информационно-поисковых систем сформировались три вида информационного обслуживания ДОКУМЕНТАЛЬНОЕ, ФАКТОГРАФИЧЕСКОЯ И КОНЦЕПТОГРАФИЧЕСКОЕ. Каждому из этих видов соответствует своя информационная система.

ДОКУМЕНТАЛЬНАЯ система, в течении уже многих веков обеспечивала информационное обслуживание общества в целом и различных его институтов, в том числе науки и техники.

Сущность документального обслуживания заключается в том, что информационные потребности членов общества удовлетворяются путем предоставления им первичных документов, необходимые сведения из которых потребители извлекают сами. Обычно грамотное документальное обслуживание осуществляется в два этапа: сначала потребителю предоставляется некоторая совокупность релевантных (релевантность - смысловое соответствие содержания документа информационному запросу {смысловое соответствие между двумя текстами}) его запросу вторичных документов (этот этап называется библиографическим), а затем, после отбора потребителем из этой совокупности определенного числа уже пертинентных (пертинентность - соответствие содержания документа информационной потребности конкретного специалиста) документов, ему предоставляют сами документы (этот этап называется библиотечным обслуживанием). Таким образом, потребность в информации при документальном обслуживании удовлетворяется опосредовано, через первичный документ.

В отличии от документального обслуживания ФАКТОГРАФИЧЕСКОЕ предполагает удовлетворение информационных потребностей непосредственно, т.е. путем представления потребителям самих сведений (отдельных данных, фактов, концепций). Эти сведения, также релевантные запросам потребителей, предварительно извлекаются информационными работниками из первичных документов и после определенной их обработки (оформления) представляются потребителям. Следует уточнить само понятие "фактографическая информация". ФАКТОГРАФИЧЕСКАЯ ИНФОРМАЦИЯ следует понимать сведения не только фактического характера, но и теоретического, предположительного, оценочного характера, т.е. включать и факты, и концепции, все то, что может быть объектом извлечения из текста, описания на определенном информационном языке, хранения и поиска в той или иной информационной системе.

Если в случае документального и фактографического обслуживания потребителю информации предоставляются документы или сведения, извлеченные из информационного потока, так сказать, в "натуральном" виде, то при КОНЦЕПТОГРАФИЧЕСКОМ обслуживании все это

(документы и сведения) подвергаются интерпретации, оценке, обобщению со стороны информационного работника. В результате такой интерпретации формулируется так называемая ситуативная информация, содержащая в себе оценку рассматриваемых сведений, тенденций и перспективы развития отдельных научных и технических направлений, рекомендаций и пр. По этой причине под концептографическим обслуживанием можно также понимать формулирование и доведения до потребителей ситуативной информации, в явном виде не содержащейсяв анализируемых источниках, а полученной в результате информационно-логического и концептографического анализа некоторой совокупности сообщений. Другими словами, в случае концептографического обслуживания потребителю представляются не только сведения о документе или сами сведения из документа, но и некоторая дополнительная информация, привнесенная информационным работником в процессе их интерпретации.

Все виды информационного обслуживания функционируют на основе своих специфичных рядов вторичных документов. По сути дела каждая из разновидностей обслуживания сводиться к созданию своего ряда вторичных документов и доведению их до потребителя различными средствами и в различных режимах информационного обслуживания.

Существенное повышение эффективности информационных систем в настоящих условиях, когда открыты возможности внедрения в информационный процесс высокопроизводительных технических средств, может быть достигнута за счет их автоматизации. Появление автоматизированных информационных систем - результат объективного процесса, обусловленного научно-технической революцией. Эти системы, интегрируя информацию, обеспечивают комплексное решение задач управления.

Лекция №2 Модели жизненного цикла информационных систем

Понятие жизненного цикла является одним из базовых понятий методологии проектирования информационных систем. Жизненный цикл информационной системы представляет собой непрерывный процесс, начинающийся! с момента принятия решения о создании информационной системы и заканчивается в момент полного изъятия ее из эксплуатации.

Стандарт ISO/IEC 12207 определяет структуру жизненного цикла, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания информационной системы. Согласно данному стандарту структура жизненного цикла основывается на трех группах процессов:

основные процессы жизненного цикла (приобретение, поставка, разработка, эксплуатация, сопровождение);

+ вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, вери-

фикация, аттестация, оценка, аудит, разрешение проблем);

+ организационные процессы (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого жизненного цикла, обучение).

Основные процессы жизненного цикла

Среди основных процессов жизненного цикла наибольшую важность разработка, эксплуатация и сопровождение. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными; полученными на предыдущем этапе, и результатами.

Разработка

Разработка информационной системы включает в себя все работы по разработке информационного программного обеспечения и его компонентов в соответствии с заданными требованиями. Разработка информационного программного обеспечения также включает:

+ оформление проектной и эксплуатационной документации;

+ подготовку материалов, необходимых для проведения тестирования тайных программных продуктов;

+ разработку материалов, необходимых для организации обучения персонала.

Разработка является одним из важнейших процессов жизненного цикла информационной системы и, как правило, включает в себя стратегическое планирование, анализ, проектирование и реализацию (программирование).

Эксплуатация

Эксплуатационные работы можно подразделить на подготовительные и основные. К подготовительным относятся:

+ конфигурирование базы данных и рабочих мест пользователей;

+ обеспечение пользователей эксплуатационной документацией;

+ обучение персонала.

+ Основные эксплуатационные работы включают;

+ непосредственно эксплуатацию;

+ локализацию проблем и устранение причин их возникновения;

+ модификацию программного обеспечения;

+ подготовку предложений по совершенствованию системы;

+ развитие и модернизацию системы.

Сопровождение

Службы технической поддержки играют весьма заметную роль в жизни любой корпоративной информационной системы. Наличие квалифицированного технического обслуживания на этапе эксплуатации информационной системы является необходимым условием для решения поставленных перед ней задач. При-

чем ошибки обслуживающего персонала могут приводить к явным или скрытым финансовым потерям сопоставимым со стоимостью самой информационной системы

Вспомогательные процессы

Среди вспомогательных процессов одно из главных мест занимает управление конфигурацией. Это один из вспомогательных процессов, поддерживающих основные процессы жизненного цикла информационной системы, прежде всего процессы разработки и сопровождения. При разработке проектов сложных информационных систем, состоящих из многих компонентов, каждый из которых может разрабатываться независимо и, следовательно, иметь несколько вариантов реализации и/или несколько версий одной реализации, возникает проблема учета их связей и функций, создания единой структуры и обеспечения развития всей системы. Управление конфигурацией позволяет организовывать, систематически учитывать и контролировать внесение изменений в различные компоненты информационной системы на всех стадиях ее жизненного цикла.

Организационные процессы

Управление проектом связано с вопросами планирования и организации работ, создания коллективов разработчиков и контроля за сроками и качеством выполняемых работ. Техническое и организационное обеспечение проекта включает: О выбор методов и инструментальных средств для реализации проекта;

+ определение методов описания промежуточных состояний разработки;

+ разработку методов и средств испытаний созданного программного обеспечения;

+ обучение персонала.

Обеспечение качества проекта связано с проблемами верификации, тестирования компонентов информационной системы.

Верификация — это процесс определения соответствия текущего состояния разработки, достигнутого на данном этапе, требованиям этого этапа.

Проверка — это процесс определения соответствия параметров разработки исходным требованиям. Проверка отчасти совпадает с тестированием, проводится для определения различий между действительными и ожидавшимися результатами и оценки соответствия характеристик информационной системы исходным требованиям.

Модели ЖЦ

Под моделью жизненного цикла понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении жизненного цикла. Модель жизненного цикла

зависит от специфики информационной системы и специфики условий, в которых последняя создается и функционирует

К настоящему времени наибольшее распространение получили следующие основные модели жизненного цикла:

Задачная модель;

каскадная модель (или системная) (70-85 г.г.);

спиральная модель (настоящее время).

Задачная модель

При разработке системы "снизу-вверх" от отдельных задач ко всей системе (задачная модель) единый поход к разработке неизбежно теряется, возникают проблемы при информационной стыковке отдельных компонентов. Как правило, по мере увеличения количества задач трудности нарастают, приходится постоянно изменять уже существующие программы и структуры данных. Скорость развития системы замедляется, что тормозит и развитие самой организации. Однако в отдельных случаях такая технология может оказаться целесообразной:

Крайняя срочность (надо чтобы хоть как-то задачи решались; потом придется все сделать заново);

Эксперимент и адаптация заказчика (не ясны алгоритмы, решения нащупываются методом проб и ошибок).

Общий вывод: достаточно большую эффективную информационной системы таким способом создать невозможно.

Каскадная модель

В ранних не очень больших по объему однородных информационных систем каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем (рис. 1). Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Положительные стороны применения каскадного подхода заключаются в следующем:

на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;

выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

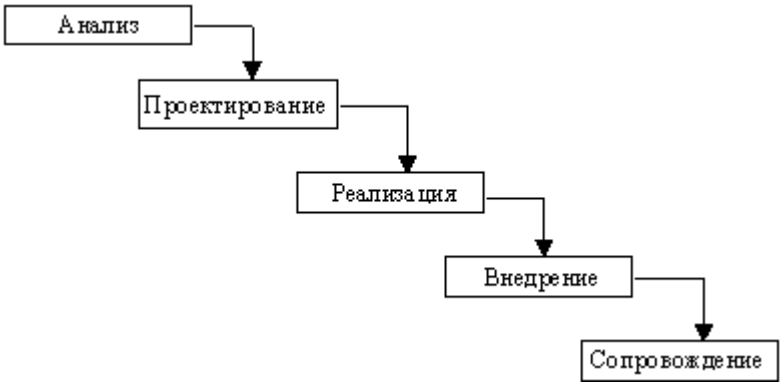

Рис. 1. Каскадная схема разработки

Каскадный подход хорошо зарекомендовал себя при построении информационных систем, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания систем никогда полностью не укладывался в такую жесткую схему. В процессе создания постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания программного обеспечения принимал следующий вид (рис. 2):

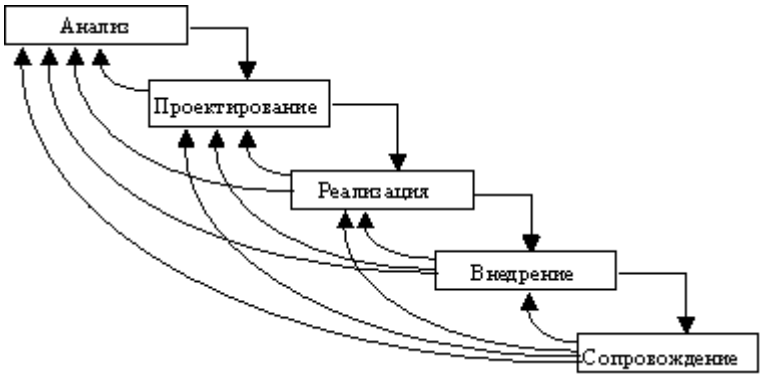

Рис. 1.2. Реальный процесс разработки ПО по каскадной схеме

Основным недостатком каскадного подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к информационным системам "заморожены" в виде технического задания на все время ее создания. Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания программного обеспечения, пользователи получают систему, не удовлетворяющую их потребностям. Модели (как функциональные, так и информационные) автоматизируемого объекта могут устареть одновременно с их утверждением. Сущность системного подхода к разработке ИС заключается в ее декомпозиции (разбиении) на автоматизируемые

функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. Таким образом, данная модель основным достоинством имеет системность разработки, а основные недостатки - медленно и дорого.

Спиральная модель

Для преодоления перечисленных проблем была предложена спиральная модель жизненного цикла (рис. 3), делающая упор на начальные этапы жизненного цикла: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента или версии программного обеспечения, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом, углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача - как можно быстрее показать пользователям системы работоспособный продукт, тем самым, активизируя процесс уточнения и дополнения требований.

Основная проблема спирального цикла - определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

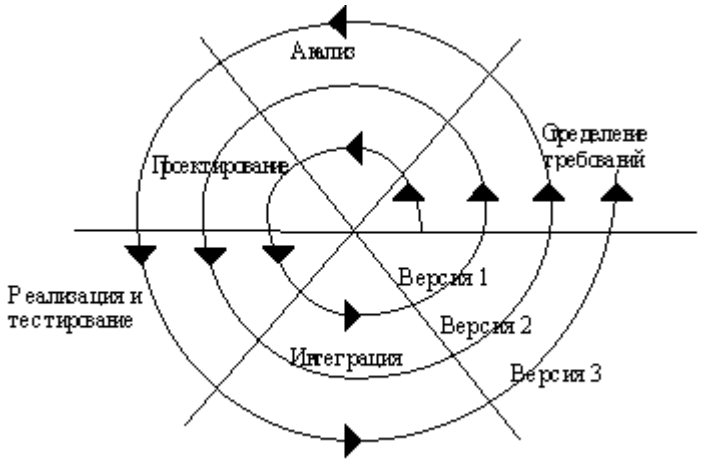

Рис 3. Спиральная модель ЖЦ ИС

Одним из возможных подходов к разработке программного обеспечения в рамках спиральной модели жизненного цикла является получившая в последнее время широкое распространение методология быстрой разработки приложений RAD (Rapid Application Development). Под этим термином обычно понимается процесс разработки программного обеспечения, содержащий 3 элемента:

небольшую команду программистов (от 2 до 10 человек);

короткий, но тщательно проработанный производственный график (от 2 до 6 мес.);

повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Жизненный цикл программного обеспечения по методологии RAD состоит из четырех фаз:

фаза определения требований и анализа;

фаза проектирования;

фаза реализации;

фаза внедрения.

Лекция 3. Функции СУБД

Традиционных возможностей файловых систем оказывается недостаточно для построения даже простых информационных систем. Выявлено несколько потребностей, которые не покрываются возможностями систем управления файлами: поддержание логически согласованного набора файлов; обеспечение языка манипулирования данными; восстановление информации после разного рода сбоев; реально параллельная работа нескольких пользователей. Можно считать, что если прикладная информационная система опирается на некоторую систему управления данными, обладающую этими свойствами, то эта система управления данными является *системой управления базами данных (СУБД)*.

Основные функции СУБД

Более точно, к числу функций СУБД принято относить следующие:

Непосредственное управление данными во внешней памяти

Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для убыстрения доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но подчеркнем, что в развитых СУБД

пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то, как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

Управление буферами оперативной памяти

СУБД обычно работают с БД значительного размера; по крайней мере, этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

Заметим, что существует отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предположении, что в будущем объем оперативной памяти компьютеров будет настолько велик, что позволит не беспокоиться о буферизации. Пока эти работы находятся в стадии исследований.

Управление транзакциями

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Таким образом, поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД (если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД.

То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД (на самом деле, это несколько идеализированное представление, поскольку в некоторых случаях пользователи многопользовательских СУБД могут ощутить присутствие своих коллег).

С управлением транзакциями в многопользовательской СУБД связаны важные понятия *сериализации транзакций* и *сериального плана выполнения смеси транзакций*. Под сериализаций параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. Сериальный план выполнения смеси транзакций - это такой план, который приводит

к сериализации транзакций. Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Существует несколько базовых алгоритмов сериализации транзакций. В централизованных СУБД наиболее распространены алгоритмы, основанные на синхронизационных захватах объектов БД. При использовании любого алгоритма сериализации возможны ситуации конфликтов между двумя или более транзакциями по доступу к объектам БД. В этом случае для поддержания сериализации необходимо выполнить откат (ликвидировать все изменения, произведенные в БД) одной или более транзакций. Это один из случаев, когда пользователь многопользовательской СУБД может реально (и достаточно неприятно) ощутить присутствие в системе транзакций других пользователей.

Журнализация

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журнализуются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). Грубо говоря, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала. В некоторых СУБД так и делают, но в большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу).

При мягком сбое во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту сбоя, и могут отсутствовать объекты, модифицированные транзакциями, которые к моменту сбоя успешно завершились (по причине использования буферов оперативной памяти, содержимое которых при мягком сбое пропадает). При соблюдении протокола WAL во внешней памяти журнала должны гарантированно находиться записи, относящиеся к операциям модификации обоих видов объектов. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций. Для того чтобы этого добиться, сначала производят откат незавершенных транзакций (undo), а потом повторно воспроизводят (redo) те операции завершенных транзакций, результаты которых не отображены во внешней памяти. Этот процесс содержит много тонкостей, связанных с общей организацией управления буферами и журналом. Более подробно мы рассмотрим это в соответствующей лекции.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после жесткого сбоя является достаточно длительным.

Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые *языками баз данных*. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - *язык определения схемы* БД *(SDL - Schema Definition Language)* и *язык манипулирования данными (DML - Data Manipulation Language).* SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language). Перечислим основные функции реляционной СУБД, поддерживаемые на "языковом" уровне (т.е. функции, поддерживаемые при реализации интерфейса SQL).

Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Специальные операторы языка SQL позволяют определять так называемые представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

Наконец, авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в

специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

Типовая организация современной СУБД

Естественно, организация типичной СУБД и состав ее компонентов соответствует рассмотренному нами набору функций. Напомним, что мы выделили следующие основные функции СУБД:

- управление данными во внешней памяти;
- управление буферами оперативной памяти;
- управление транзакциями;
- журнализация и восстановление БД после сбоев;
- поддержание языков БД.

Логически в современной реляционной СУБД можно выделить наиболее внутреннюю часть - ядро СУБД (часто его называют Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения, набор утилит. В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала. Как можно было понять из первой части этой лекции, функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и проверенным протоколам. Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL (или в подсистеме поддержки выполнения таких программ) и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. При использовании архитектуры "клиент-сервер" ядро является основной составляющей серверной части системы.

Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу. Основной проблемой реляционных СУБД является то, что языки этих систем (а это, как правило, SQL) являются непроцедурными, т.е. в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не является процедурой, а лишь описывает в некоторой форме условия совершения желаемого действия (вспомните примеры из первой лекции). Поэтому компилятор должен решить, каким образом выполнять оператор языка прежде, чем произвести программу. Применяются достаточно сложные методы оптимизации операторов, которые мы подробно рассмотрим в следующих лекциях. Результатом компиляции является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением подсистемы поддержки времени выполнения,

представляющей собой, по сути дела, интерпретатор этого внутреннего языка.

Наконец, в отдельные утилиты БД обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

Лекция 4. Экспертные системы

Систему искусственного интеллекта, построенную на основе высококачественных специальных знании о некоторой предметной области (полученных от экспертов - специалистов этой области), называют экспертной системой. Экспертные системы - один из немногих видов систем искусственного интеллекта получили широкое распространение и практическое применение. Существуют экспертные системы по военному делу, геологии, инженерному делу, информатике, космической технике, математике, медицине, метеорологии, промышленности, сельскому хозяйству, управлению, физике, химии, электронике, юриспруденции и т.д. И только то, что экспертные системы остаются весьма сложными, дорогими, а главное, узкоспециализированными программами, сдерживает их еще более широкое распространение.

От других программ экспертные системы отличаются по следующим признакам:

компетентность - в конкретной предметной области экспертная система должна достигать того же уровня, что и эксперты - люди, при этом она должна пользоваться теми же эвристическими приемами, также глубоко и широко отражать предметную область;

символьные рассуждения - знания, на которых основана экспертная система, представляют в символьном виде понятия реального мира, рассуждения также происходят в виде преобразований символьных наборов;

глубина - экспертиза должна решать глубокие, нетривиальные задачи, отличающиеся сложностью либо в плане сложности знаний, которые экспертная система использует, либо в плане их обилия, это не позволяет использовать полный перебор вариантов как метод решения задачи и заставляет прибегать к эвристическим, творческим, неформальным методам;

самосознание - экспертная система должна включать в себя механизм объяснения того, каким образом она приходит к решению задачи.

Экспертные системы, выполняющие интерпретацию, как правило, используют информацию от датчиков для описания ситуации. Например, это может быть интерпретация показаний измерительных приборов на химическом заводе для определения состояния процесса. Интерпретирующие системы имеют дело не с четкими символьными представлениями проблемной ситуации, а непосредственно с

реальными данными. Они сталкиваются с затруднениями, которых нет у систем других типов, потому что им приходится обрабатывать информацию зашумленную, недостаточную, неполную, ненадежную или ошибочную. Им необходимы специальные методы регистрации характеристик непрерывных потоков данных, сигналов пли изображений и методы их символьного представления.

Интерпретирующие экспертные системы могут обрабатывать разнообразные виды данных. Например, системы анализа сцен и распознавания речи, используя естественную информацию (в одном случае визуальные образы), анализируют их характеристики и понимают их смысл. Интерпретация в области химии использует данные дифракции спектрального анализа или ядерного магнитного резонанса для определения структуры веществ. Интерпретирующая система в геологии используета зондирование - измерение проводимости горных пород, чтобы определить подповерхностные геологические структуры интерпретирующие системы используют показания (например, значения температуры, пульса, кровяного давления), диагноз или тяжесть заболевания. Наконец, в военном деле системы используют данные от радаров, радиосвязи и сонарных устройств оценить ситуацию и идентифицировать цели.

Экспертные системы, осуществляющие прогноз, определяют вероятностные условия заданных ситуаций. Примерами служат прогноз ущерба урожаю от вида вредных насекомых, оценивание спроса на нефть на мировом рынке прогнозирование места возникновения следующего вооруженного конфликта по данным разведки. Системы прогнозирования иногда используют моделирование, т.е. программы, которые отражают взаимосвязи в реальном мире, чтобы сгенерировать ситуации, которые могут возникнуть при тех или иных входных данных.

Экспертные системы выполняют Диагностирование, используя описания ситуации, поведения или знания о конструкции компонентов, чтобы установить вероятные причины неправильно функционирования диагностируемой системы. Примерами служат определение причин заболевания по симптомам, наблюдаемым у пациентов; локализация неисправностей в электронных схемах и определение неисправных компонентов в системе охлаждения ядерных реакторов. Диагностические системы часто являются консультантами, которые не только ставят диагноз, но и помогают в отладке. Они могут взаимодействовать с пользователем, чтобы оказать помощь при поиске неисправностей, а затем предложить порядок действий по их устранению. Медицина представляется вполне естественной областью для диагностирования, и действительно, в медицинской области было разработано больше диагностических систем, чем в любой другой отдельно взятой предметной области. Однако в настоящее время многие диагностические системы разрабатываются для приложений к инженерному делу и компьютерным системам.

Экспертные системы, выполняющие проектирование, разрабатывают конфигурации объектов с учетом набора ограничений, присущих проблеме. Примерами могут служить генная инженерия, разработка СБИС и синтез сложных органических молекул.

Экспертные системы, занятые планированием, проектируют действия; они определяют полную последовательность действий, прежде чем начнется их выполнение.

Примерами могут служить создание плана применения последовательности химических реакций к группам атомов с целью синтеза сложных органических соединений или создание плана воздушного нападения, рассчитанного на несколько дней, с целью нейтрализации определенного фактора боеспособности врага.

Экспертные системы, выполняющие наблюдение, сравнивают действительное поведение с ожидаемым поведением системы. Примерами могут служить слежение за показаниями измерительных приборов в ядерных реакторах с целью обнаружения аварийных ситуаций или оценка данных мониторинга больных, помещенных в блоки интенсивной терапии. Наблюдающие экспертные системы подыскивают наблюдаемое поведение, которое подтверждает их ожидания относительно нормального поведения или их предположения о возможных отклонениях. Наблюдающие экспертные системы по самой своей природе должны работать в режиме реального времени и осуществлять зависящую как от времени, так и от контекста интерпретацию поведения наблюдаемого объекта.

Экспертные системы, выполняющие обучение, подвергают диагностике, «отладке» и исправлению (коррекции) поведение обучаемого. В качестве примеров приведем обучение студентов отысканию неисправностей в электрических цепях, обучение военных моряков обращению с двигателем на корабле и обучение студентов-медиков выбору антимикробной терапии. Обучающие системы создают модель того, что обучающийся знает и как он эти знания применяет к решению проблемы. Системы диагностируют и указывают обучающемуся его ошибки, анализируя модель и строя планы исправлений указанных ошибок. Они исправляют поведение обучающихся, выполняя эти планы с помощью непосредственных указаний обучающимся.

Экспертные системы, осуществляющие управление, адаптивно руководят поведением системы в целом. Примером служит управление производством и распределением компьютерных систем. Управляющие экспертные системы должны включать наблюдающие компоненты, чтобы отслеживать поведение объекта на протяжении времени, но они могут нуждаться и в других компонентах для выполнения любых пли всех из уже рассмотренных типов задач: интерпретации, прогнозирования, диагностики, проектирования, планирования, отладки, ремонта и обучения. Типичная комбинация задач состоит из наблюдения, диагностики, отладки, планирования и прогноза.

Лекция №5. Теория реляционных баз данных

Модель данных — совокупность структур данных и операций по их обработке. С помощью модели данных можно наглядно представить структуру объектов и установленные между ними связи. Для терминологии моделей данных характерны понятия «элемент данных» и «правила связывания». Элемент данных описывает любой набор данных, а правила связывания определяют алгоритмы взаимосвязи элементов данных. К настоящему времени разработано множество различных моделей данных, но на практике используется три основных. Выделяют иерархическую, сетевую и реляционную модели данных. Соответственно говорят об иерархических, сетевых и реляционных СУБД.

О Иерархическая модель данных. Иерархически организованные данные встре-

чаются в повседневной жизни очень часто. Например, структура высшего учебного заведения — это многоуровневая иерархическая структура. Иерархическая (древовидная) БД состоит из упорядоченного набора элементов. В этой модели исходные элементы порождают другие элементы, причем эти элементы в свою очередь порождают следующие элементы. Каждый порожденный элемент имеет только один порождающий элемент.

Организационные структуры, списки материалов, оглавление в книгах, планы проектов и многие другие совокупности данных могут быть представлены в иерархическом виде. Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя.

Основным недостатком данной модели является необходимость использования той иерархии, которая была заложена в основу БД при проектировании. Потребность в постоянной реорганизации данных (а часто невозможность этой реорганизации) привели к созданию более общей модели — сетевой.

О Сетевая модель данных. Сетевой подход к организации данных является расширением иерархического подхода. Данная модель отличается от иерахической тем, что каждый порожденный элемент может иметь более одного порождающего элемента. ■

Поскольку сетевая БД может представлять непосредственно все виды связей, присущих данным соответствующей организации, по этим данным можно перемещаться, исследовать и запрашивать их всевозможными способами, то есть сетевая модель не связана всего лишь одной иерархией. Однако для того чтобы составить запрос к сетевой БД, необходимо достаточно глубоко вникнуть в ее структуру (иметь под рукой схему этой БД) и выработать механизм навигации по базе данных, что является существенным недостатком этой модели БД.

О Реляционная модель данных. Основная идея реляционной модели данных заключается в том, чтобы представить любой набор данных в виде двумерной таблицы. В простейшем случае реляционная модель описывает единственную двумерную таблицу, но чаще всего эта модель описывает структуру и взаимоотношения между несколькими различными таблицами.

Реляционная модель данных

Итак, целью информационной системы является обработка *данных* об *объектах* реального мира, с учетом *связей* между объектами. В теории БД данные часто называют *атрибутами, а* объекты — *сущностями.* Объект, атрибут и связь — фундаментальные понятия И.С.

Объект (или сущность) — это нечто существующее и *различимое,* то есть объектом можно назвать то «нечто», для которого существуют название и способ отличать один подобный объект от другого. Например, каждая школа — это объект. Объектами являются также человек, класс в школе, фирма, сплав, химическое соединение и т. д. Объектами могут быть не только материальные предметы, но и более абстрактные понятия, отражающие реальный мир. Например, события, регионы, произведения искусства; книги (не как полиграфическая продукция, а как

произведения), театральные постановки, кинофильмы; правовые нормы, философские теории и проч.

Атрибут (или *данное)* — это некоторый показатель, который характеризует некий объект и принимает для конкретного экземпляра объекта некоторое числовое, текстовое или иное значение. Информационная система оперирует наборами объектов, спроектированными применительно к данной предметной области, используя при этом конкретные *значения атрибутов* (данных) тех или иных объектах. Например, возьмем в качестве набора объектов классы в школе. Число учеников в классе — это данное, которое принимает числовое значение (у одного класса 28, у другого— 32). Название класса — это данное, принимающее текстовое значение (у одного — 10А, у другого — 9Б и т. д.).

Развитие реляционных баз данных началось в конце 60-х годов, когда появились первые работы, в которых обсуждались возможности использования при проектировании баз данных привычных и естественных способов представления данных — так называемых табличных даталогических моделей.

Основоположником теории реляционных баз данных считается сотрудник фирмы IBM доктор Э. Кодд, опубликовавший 6 июня 1970 г. статью *A Relational Model of Data for Large-Shared Data Banks* (Реляционная модель данных для больших коллективных банков данных). В этой статье впервые был использован термин «реляционная модель данных. Теория реляционных баз данных, разработанная в 70-х годах в США доктором Э. Коддом, имеет под собой мощную математическую основу, описывающую правила эффективной организации данных. Разработанная Э. Коддом теоретическая база стала основой для разработки теории проектирования баз данных.

Э. Кодд, будучи математиком по образованию, предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он доказал, что любой набор данных можно представить в виде двумерных таблиц особого вида, известных в математике как «отношения».

Реляционной считается такая база данных, в которой все данные представлены для пользователя в виде прямоугольных таблиц значений данных, и все операции над базой данных сводятся к манипуляциям с таблицами.

Таблица состоит из *столбцов (полей)* и *строк (записей);* имеет имя, уникальное внутри базы данных. *Таблица* отражает *тип объекта* реального мира *(сущность),* а каждая ее *строка— конкретный объект.* Каждый столбец таблицы — это совокупность значений конкретного атрибута объекта. Значения выбираются из множества всех возможных значений атрибута объекта, которое называется *доменом (domain).*

В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементам данных. Если при вычислении логического условия относительно элемента данных в результате получено значение «истина», то этот элемент принадлежит домену. В простейшем случае домен определяется как допустимое потенциальное множество значений одного типа. Например, совокупность дат рождения всех сотрудников составляет «домен дат рождения», а

имена всех сотрудников составляют «домен имен сотрудников». Домен дат рождения имеет тип данных, позволяющий хранить информацию о моментах времени, а домен имен сотрудников должен иметь символьный тип данных.

Если два значения берутся из одного и того же домена, то можно выполнять сравнение этих двух значений. Например, если два значения взяты из домена дат рождения, то можно сравнить их и определить, кто из сотрудников старше. Если же значения берутся из разных доменов, то их сравнение не допускается, так как, по всей вероятности, оно не имеет смысла. Например, из сравнения имени и даты рождения сотрудника ничего определенного не выйдет.

Каждый столбец (поле) имеет имя, которое обычно записывается в верхней части таблицы. При проектировании таблиц в рамках конкретной СУБД имеется возможность выбрать для каждого поля его *тип,* то есть определить набор правил по его отображению, а также определить те операции, которые можно выполнять над данными, хранящимися в этом поле. Наборы типов могут различаться у разных СУБД.

Имя поля должно быть уникальным в таблице, однако различные таблицы могут иметь поля с одинаковыми именами. Любая таблица должна иметь, по крайней мере, одно поле; поля расположены в таблице в соответствии с порядком следования их имен при ее создании. В отличие от полей, строки не имеют имен; порядок их следования в таблице не определен, а количество логически не ограничено.

Так как строки в таблице не упорядочены, невозможно выбрать строку по ее позиции — среди них не существует «первой», «второй», «последней». Любая таблица имеет один или несколько столбцов, значения в которых однозначно идентифицируют каждую ее строку. Такой столбец (или комбинация столбцов) называется *первичным ключом (primary key).* Часто вводят искусственное поле, предназначенное для нумерации записей в таблице. Таким полем, например, может быть его порядковый, который сможет обеспечить уникальность каждой записи в таблице. Ключ должен обладать следующими свойствами.

Уникальностью. В каждый момент времени никакие два различных кортежа отношения не имеют одинакового значения для комбинации входящих в ключ атрибутов. То есть в таблице не может быть двух строк, имеющих одинаковый идентификационный номер или номер паспорта.

Минимальностью. Ни один из входящих в ключ атрибутов не может быть исключен из ключа без нарушения уникальности. Это означает, что не стоит создавать ключ, включающий и номер паспорта, и идентификационный номер. Достаточно использовать любой из этих атрибутов, чтобы однозначно идентифицировать кортеж. Не стоит также включать в ключ неуникальный атрибут, то есть запрещается использование в качестве ключа комбинации идентификационного номера и имени служащего. При исключении имени служащего из ключа все равно можно уникально идентифицировать каждую строку.

Каждое отношение имеет, по крайней мере, один возможный ключ, поскольку совокупность всех его атрибутов удовлетворяет условию уникальности — это следует из самого определения отношения.

Один из возможных ключей произвольно выбирается в *качестве первичного ключа.* Остальные возможные ключи, если они есть, принимаются за *альтернативные ключи.* Например, если в качестве первичного ключа выбрать идентификационный номер, то номер паспорта будет альтернативным ключом.

Взаимосвязь таблиц является важнейшим элементом реляционной модели данных. Она поддерживается *внешними ключами (foreign key).*

При описании модели реляционной базы данных для одного и того же понятия часто употребляют различные термины, что зависит от уровня описания (теория или практика) и системы (Access, SQL Server, dBase). В табл. 2.3 приведена сводная информация об используемых терминах.

Таблица 2.3. Терминология баз данных

Теория БД	**Реляционные БД**	**SQL Server**
Отношение (Relation)	Таблица (Table)	Таблица (Table)
Кортеж (Tuple)	Запись (Record)	Строка (Row)
Атрибут (Attribute)	Поле (Field)	Столбец или колонка (Column)

Реляционные базы данных

Реляционная база данных — это совокупность отношений, содержащих всю информацию, которая должна храниться в базе данных. То есть база данных представляет набор таблиц, необходимых для хранения всех данных. Таблицы реляционной базы данных логически связаны между собой.Требования к проектированию реляционной базы данных в общем виде можно свести к нескольким правилам.

- Каждая таблица имеет уникальное в базе данных имя и состоит из однотипных строк.
- Каждая таблица состоит из фиксированного числа столбцов и значений. В одном столбце строки не может быть сохранено более одного значения. Например, если есть таблица с информацией об авторе, дате издания, тираже и т. д., то в столбце с именем автора не может храниться более одной фамилии. Если книга написана двумя и более авторами, придется использовать дополнительные таблицы.
- Ни в какой момент времени в таблице не найдется двух строк, дублирующих друг друга. Строки должны отличаться хотя бы одним значением, чтобы была возможность однозначно идентифицировать любую строку таблицы.
- Каждому столбцу присваивается уникальное в пределах таблицы имя; для него устанавливается конкретный тип данных, чтобы в этом столбце размещались однородные значения (даты, фамилии, телефоны, денежные суммы и т. д.).
- Полное информационное содержание базы данных представляется в виде явных значений самих данных, и такой метод представления является единственным. Например, связь между таблицами осуществляется на основе хранимых в

соответствующих столбцах данных, а не на основе каких-либо указателей, искусственно определяющих связи.

О При обработке данных можно свободно обращаться к любой строке или любому столбцу таблицы. Значения, хранимые в таблице, не накладывают никаких ограничений на очередность обращения к данным. Описание столбцов,

Нормализация и ее необходимость

При проектировании структуры базы данных заказчик часто предоставляет разработчику описание форм и бланков, существующих в бумажном виде. Поэтому, прежде чем приступать к проектированию таблиц для БД, необходимо выяснить цели проектирования. К ним относятся:

О возможность хранить все необходимые данные в БД;

О исключение избыточности данных;

О необходимость свести количество хранимых таблиц к .минимуму.

При простом переносе полей бумажных форм в таблицы базы данных неизбежно возникнет ряд проблем — даже для простых двумерных структур приходится изменять состав полей

(В дальнейшем при проектировании базы данных эта универсальная таблица может быть разбита на несколько таблиц, имеющих более простую структуру и связанных друг с другом.

Нормализация таблиц — это формальный аппарат ограничений на формирование таблиц, описывающий разбиение таблиц на две или более частей и обеспечивающий применение лучших методов добавления, изменения и удаления данных; или еще проще — процесс представления данных в виде простых двумерных таблиц, который позволяет устранить дублирование этих данных и обеспечивает непротиворечивость хранимых в базе данных. Таким образом, окончательной целью нормализации является получение такого проекта базы данных, в котором любая часть информации хранится лишь в одном месте, то есть исключается избыточность информации. Это делается не столько с целью экономии места (в некоторых случаях нормализованные таблицы занимают больше места, чем ненормализованные), сколько для исключения возможности противоречий в хранимых данных. Если исходить из структуры данных, то нормализацией называется процесс превращения сетевой или иерархической структуры данных в реляционную.

Основой процесса нормализации является предложенный Е. Коддом в рамках реляционной теории аппарат, называемый *нормализацией отношений.* Им выделено три формы нормальных отношений, которые в дальнейшем были доработаны, и предложен механизм перехода от формы к форме, а кроме того было добавлено еще три специальных формы. Итого, существует шесть форм „ нормальных отношений. Но, как правило, необходимо и достаточно привести базу данных к третьей нормальной форме.

Таблица считается *нормализованной на определенном уровне,* когда она удовлетворяет условиям, накладываемым соответствующей *формой нормализации.*

Процесс нормализации представляет собой последовательное изменение структуры таблиц до тех пор, пока она не будет удовлетворять требованиям последней формы нормализации. Существуют следующие шесть форм нормализации: О первая нормальная форма (First Normal Form, INF);

О вторая нормальная форма (Second Normal Form, 2NF);

О третья нормальная форма (Third Normal Form, 3NF);

О нормальная форма Бойса - Кодда (Brice - Codd Normal Form, BCNF);

О четвертая нормальная форма (fourth Normal Form, 4NF);

О пятая нормальная форма, или нормальная форма проекции-соединения (Fifth Normal Form, 5NF или PJ/NF).

При описании нормальных форм используется несколько понятий.

О *Функциональной зависимостью* между полями *A и B* называется зависимость, при которой каждому значению *A* в любой момент времени соответствует *единственное* значение *B* из всех возможных. Примером функциональной зависимости может служить связь реки и моря, так как одна река впадает в единственное море и с течением времени эта связь не меняется.

О *Полной функциональной зависимостью* между составным полем *A* и полем *B* называется зависимость, при которой поле *B* зависит функционально от поля *A* и не зависит функционально от любого подмножества поля *A*.

О *Многозначная функциональная зависимость.* Поле *A* однозначно определяет поле *B,* если для каждого значения поля *A* существует хорошо определенное множество соответствующих значений поля *B.* Например, если рассматривать таблицу предметов и оценок учеников в школе, то поле с оценкой имеет хорошо определенное множество допустимых значений (1, 2, 3, 4, 5). Кроме того, количество предметов в школе также ограничено.

О *Транзитивная функциональная зависимость* между полями *A* и *C* наблюдается в том случае, если поле *B* функционально зависит от поля *A* и поле *C* функционально зависит от поля B. В то же время не существует функциональной зависимости поля A от поля B.

О Несколько полей *взаимно независимы,* если ни одно из них не является функционально зависимым от другого поля.

О *Неключевым* полем таблицы называется каждое поле, не входящее в состав первичного ключа.

Первая нормальная форма

Таблица находится в *первой нормальной форме* тогда, когда она не содержит повторяющихся полей и составных значений полей (то есть каждое поле должно содержать одно значение, а не их комбинацию).

Вторая нормальная форма

Таблица находится во *второй нормальной форме,* если она удовлетворяет требованиям первой нормальной формы и все ее поля, не входящие в первичный ключ, связаны

полной функциональной зависимостью с первичным ключом, то есть любое не ключевое поле однозначно идентифицируется полным набором ключевых полей.

Итак, таблица, находящаяся во второй нормальной форме, должна удовлетворять следующим правилам:

О таблица должна содержать данные об одном типе объектов;

О каждая таблица должна содержать одно поле или несколько полей, образующих уникальный идентификатор (или первичный ключ) для каждой строки;

О все поля, не имеющие ключа, должны определяться полным уникальным идентификатором данной таблицы.

Если таблица имеет простой первичный ключ, состоящий только из одного

Третья нормальная форма

Таблица находится в *третьей нормальной форме,* если она удовлетворяет определению второй нормальной формы и ни одно из ее неключевых полей функционально не зависит от любого другого неключевого поля. Можно сказать, что таблица находится в третьей нормальной форме, если она находится во второй нормальной форме и каждое неключевое поле нетранзитивно зависит от первичного ключа.

Требование третьей нормальной формы сводится к тому, чтобы все нёключевые поля зависели *только* от первичного ключа и не зависели друг от друга. Другими словами, нужно иметь возможность изменять значение любого неключевого поля, не изменяя значения любого другого поля базы данных. Это требование исключает любое поле, значения в котором получаются как результат вычислений, использующих значения других полей.

Лекция 6. Распределенные базы данных

Основная задача систем управления распределенными базами данных состоит в обеспечении средства интеграции локальных баз данных, располагающихся в некоторых узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных как к единой базе данных.

При этом должны обеспечиваться:

- простота использования системы;
- возможности автономного функционирования при нарушениях связности сети или при административных потребностях;
- высокая степень эффективности.

Возможны однородные и неоднородные распределенные базы данных. В однородном случае каждая локальная база данных управляется одной и той же СУБД. В неоднородной системе локальные базы данных могут относиться даже к разным моделям данных. Сетевая интеграция неоднородных баз данных - это актуальная, но

очень сложная проблема. Многие решения известны на теоретическом уровне, но пока не удается справиться с главной проблемой - недостаточной эффективностью интегрированных систем.

Заметим, что более успешно практически решается промежуточная задача - интеграция неоднородных SQL-ориентированных систем. Понятно, что этому в большой степени способствует стандартизация языка SQL и общее следование производителей СУБД принципам открытых систем.

Мы ограничимся рассмотрением проблем однородных распределенных СУБД на примере System R*.

Распределенная система управления базами данных System R

Основную цель проекта можно сформулировать следующим образом: обеспечить средства интеграции локальных баз данных System R, располагающихся в узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных так, как если бы они были централизованы. При этом должны обеспечиваться:

- легкость использования системы;
- возможности автономного функционирования при нарушениях связности сети или при административных потребностях;
- высокая степень эффективности.

Для решения этих проблем было необходимо принять ряд проектных решений, касающихся декомпозиции исходного запроса, оптимального выбора способа выполнения запроса, согласованного выполнения транзакций, обеспечения синхронизации, обнаружения и разрешения распределенных тупиков, восстановления состояния баз данных после разного рода сбоев узлов сети.

Легкость использования системы достигается за счет того, что пользователи System R (разработчики прикладных программ и конечные пользователи) остаются в среде языка SQL, т.е. могут продолжать работать в тех же внешних условиях, что и в System R (и SQL/DS и DB2). Возможность использования SQL основывается на обеспечении System R прозрачности местоположения данных. Система автоматически обнаруживает текущее местоположение упоминаемых в запросе пользователя объектов данных; одна и та же прикладная программа, включающая предложения SQL, может быть выполнена в разных узлах сети. При этом в каждом узле сети на этапе компиляции запроса выбирается наиболее оптимальный план выполнения запроса в соответствии с расположением данных в распределенной системе.

Обеспечению автономности узлов сети в System R уделяется очень большое внимание. Каждая локальная база данных администрируется независимо от других. Возможны автономное подключение новых пользователей, смена версии автономной части системы и т.д. Система спроектирована таким образом, что в ней не требуются централизованные службы именования объектов или обнаружения тупиков. В индивидуальных узлах не требуется наличие глобального знания об операциях, выполняющихся в других узлах сети; работа с доступными базами данных может

продолжаться при выходе из строя отдельных узлов сети или линий связи.

Высокая степень эффективности системы является одним из наиболее ключевых требований к распределенным системам управления базами данных вообще и к System R в частности. Для достижения этой цели используются два основных приема.

Во-первых, в System R выполнению запроса предшествует его компиляция. В ходе этого процесса производится поиск употребляемых в запросе имен объектов баз данных в распределенном каталоге и замена имен на внутренние идентификаторы; проверка прав доступа пользователя, от имени которого производится компиляция, на выполнение соответствующих операций над базами данных и выбор наиболее оптимального глобального плана выполнения запроса, который затем подвергается декомпозиции и по частям рассылается в соответствующие узлы сети, где производится выбор оптимальных локальных планов выполнения компонентов запроса и происходит генерация модулей доступа в машинных кодах. В результате множество действий производится на стадии компиляции до реального выполнения запроса. Обработанная посредством прекомпилятора System R прикладная программа, включающая предложения SQL, может в дальнейшем выполняться много раз без дополнительных накладных расходов. Использование распределенного каталога, распределенная компиляция и оптимизация запросов являются наиболее интересными и оригинальными аспектами проекта System R.

Вторым средством повышения эффективности системы является возможность перемещения удаленных отношений в локальную базу данных. Диалект SQL, используемый в System R, включает предложение MIGRATE TABLE, при выполнении которого указанное отношение переносится в локальную базу данных. Это средство, находящееся в распоряжении пользователей, конечно, в ряде случаев может помочь добиться более эффективного прохождения транзакций. Естественно, как и для всех операций, операция MIGRATE по отношению к указанному отношению доступна не любому пользователю, а лишь тем, которые обладают соответствующим правом.

Прежде, чем перейти к более детальному изложению наиболее интересных аспектов реализации System R, упомянем некоторые средства, которые разработчики этой системы предполагали реализовать на начальной стадии проекта, но которые реализованы не были (причем некоторые из них, видимо, и не будут никогда реализованы). Предполагалось иметь в системе средства горизонтального и вертикального разделения отношений распределенной базы данных, средства дублирования отношений в нескольких узлах с поддержкой согласованности копий и средства поддержания мгновенных снимков состояния баз данных в соответствии с заданным запросом.

Как и в случае разделенных отношений, кроме существенных проблем поддержания согласованности копий, проблемой является и разумное использование копий, наличие которых должно было бы учитываться оптимизатором.

Создание мгновенного снимка состояния баз данных в соответствии с заданным запросом на выборку должно было производиться с использованием новой конструкции SQL.

DEFINE SNAPSHOT <snapshot-name> (<attribute-list>)

AS <query>

REFRESHED EVERY <period>

При выполнении предложения фактически производится выполнение указанного в нем запроса на выборку, а результирующее отношение сохраняется под указанным в предложении именем в локальной базе данных в том узле, в котором выполняется предложение. После этого мгновенный снимок периодически обновляется в соответствии с запомненным запросом.

Можно обновить мгновенный снимок, не дожидаясь истечения временного интервала, указанного в определении, путем выполнения предложения REFRESH SNAPSHOT <snapshot-name>.

Разумное использование мгновенных снимков более реально, чем использование разделенных отношений и копированных отношений, поскольку их можно в некотором смысле рассматривать как материализованные представления базы данных. Имя мгновенного снимка можно было бы использовать прямо в запросе на выборку там, где можно использовать имена базовых отношений или представлений. Большие проблемы связаны с обновлением отношений через их мгновенные снимки, поскольку в момент обновления содержимое мгновенного снимка может расходиться с текущим содержимым базового отношения.

По отношению к мгновенным снимкам проблем поддержания согласованного состояния мгновенного снимка и базовых отношений не существует, поскольку автоматическое согласование не требуется. Что же касается разделенных отношений и раскопированных отношений, то для них эта проблема общая и достаточно трудная. Во-первых, согласование разделов и копий вызывает существенные накладные расходы при выполнении операций модификации хранимых отношений. Для этого требуется выработка и соблюдение специальных протоколов модификации.

Во-вторых, введение копированных отношений обычно производится не столько для увеличения эффективности системы, сколько для увеличения доступности данных при нарушении связности сети. В системах, в которых применяется этот подход, при нарушении связности сети работа с распределенной базой данных обычно продолжается только в одной из образовавшихся подсетей. При этом для выбора подсети используются алгоритмы голосования; решение принимается на основе учета количества связных узлов сети. Применяются и другие подходы, но все они очень дорогостоящие, а самое главное, они плохо согласуются с базовым подходом System R по поводу выбора способа выполнения запроса на стадии его компиляции. Поэтому, как нам кажется, в System R никогда не будут реализованы средства, позволяющие тем или иным способом поддерживать копии отношений в нескольких узлах сети.

Именование объектов и организация распределенного каталога

Напомним прежде всего, что полное имя отношения (базового или представления) в базе данных System R имеет вид имя-пользователя.имя-отношения, где имя-пользователя идентифицирует пользователя - создателя отношения, а имя-отношения

- это то имя, которое было указано в предложениях CREATE TABLE или CREATE VIEW. В запросах можно указывать либо это полное имя отношения, либо его локальное имя. Во втором случае при компиляции используются стандартные правила дополнения локального имени до полного с использованием в качестве составляющей имя-пользователя идентификатора пользователя, от имени которого выполняется компиляция.

В System R используется развитие этого подхода. Системное имя отношения включает четыре компонента: идентификатор пользователя-создателя отношения; идентификатор узла сети, в котором выполнялась операция создания отношения; локальное имя отношения, присвоенное ему при создании; идентификатор узла, в котором отношение располагалось непосредственно после своего создания (напомним, что отношение может перемещаться из одного узла в другой при выполнении операции MIGRATE).

В запросе на SQL можно использовать системные имена объектов, но разрешается использовать и короткие локальные имена (либо локальное имя, квалифицированное именем пользователя). При этом возможны две интерпретации локального имени. Оно может интерпретироваться как часть системного имени, и в этом случае по умолчанию дополняется до системного, исходя из идентификатора узла, в котором производится компиляция, и идентификатора пользователя, от имени которого она производится (если имя пользователя не указано явно). Вторая возможная интерпретация локального имени заключается в рассмотрении его как имени ранее определенного синонима системного имени.

Распределенная компиляция запросов

Как мы уже отмечали, запросы на языке SQL до своего реального выполнения подвергаются компиляции. Как и в случае System R компиляция запроса может производиться на стадии прекомпиляции прикладной программы, написанной на традиционном языке программирования (PL/1, Cobol, ассемблер) с включением предложений SQL, или в динамике выполнения транзакции при выполнении предложения PREPARE.

Будем называть главным узлом тот узел сети, в котором инициирован процесс компиляции предложения SQL, и дополнительными узлами - те узлы, которые вовлекаются в этот процесс в ходе его выполнения. На самом грубом уровне процесс компиляции можно разбить на следующие фазы:

1. В главном узле производится грамматический разбор предложения SQL с построением внутреннего представления запроса в виде дерева. На основе информации из локального каталога главного узла и удаленных каталогов дополнительных узлов производится замена имен объектов, фигурирующих в запросе, на их системные идентификаторы.

2. В главном узле генерируется глобальный план выполнения запроса, в котором учитывается лишь порядок взаимодействий узлов при реальном выполнении запроса. Для выработки глобального плана используется расширение техники оптимизации, применяемой в System R. Глобальный план отображается в преобразованном соответствующим образом дереве запроса.

3. Если в глобальном плане выполнения запроса участвуют дополнительные узлы, производится его декомпозиция на части, каждую из которых можно выполнить в одном узле (например, локальная фильтрация отношения в соответствии с заданным в условии выборки предикате ограничения). Соответствующие части запроса (во внутреннем представлении) рассылаются в дополнительные узлы.

4. В каждом узле, участвующем в глобальном плане выполнения запроса (главном и дополнительных) выполняется завершающая стадия выполнения компиляции. Эта стадия включает, по существу, две последние фазы процесса компиляции запроса в System R: оптимизацию и генерацию машинных кодов. Производится проверка прав пользователя, от имени которого производится компиляция, на выполнение соответствующих действий; происходит обработка представлений базы данных (здесь имеются тонкости, связанные с тем, что представления могут включать удаленные отношения; ниже мы еще остановимся на этом, а пока будем считать, что в запросе употребляются только имена базовых отношений); осуществляется локальная оптимизация обрабатываемой части запроса в соответствии с имеющимися индексами; наконец, производится генерация кода.

Интегрированные или федеративные системы и мультибазы данных

Направление интегрированных или федеративных систем неоднородных БД и мульти-БД появилось в связи с необходимостью комплексирования систем БД, основанных на разных моделях данных и управляемых разными СУБД.

Основной задачей интеграции неоднородных БД является предоставление пользователям интегрированной системы глобальной схемы БД, представленной в некоторой модели данных, и автоматическое преобразование операторов манипулирования БД глобального уровня в операторы, понятные соответствующим локальным СУБД. В теоретическом плане проблемы преобразования решены, имеются реализации.

При строгой интеграции неоднородных БД локальные системы БД утрачивают свою автономность. После включения локальной БД в федеративную систему все дальнейшие действия с ней, включая администрирование, должны вестись на глобальном уровне. Поскольку пользователи часто не соглашаются утрачивать локальную автономность, желая тем не менее иметь возможность работать со всеми локальными СУБД на одном языке и формулировать запросы с одновременным указанием разных локальных БД, развивается направление мульти-БД. В системах мульти-БД не поддерживается глобальная схема интегрированной БД и применяются специальные способы именования для доступа к объектам локальных БД. Как правило, в таких системах на глобальном уровне допускается только выборка данных. Это позволяет сохранить автономность локальных БД.

Как правило, интегрировать приходится неоднородные БД, распределенные в вычислительной сети. Это в значительной степени усложняет реализацию. Дополнительно к собственным проблемам интеграции приходится решать все проблемы, присущие распределенным СУБД: управление глобальными

транзакциями, сетевую оптимизацию запросов и т.д. Очень трудно добиться эффективности.

Как правило, для внешнего представления интегрированных и мульти-БД используется (иногда расширенная) реляционная модель данных. В последнее время все чаще предлагается использовать объектно-ориентированные модели, но на практике пока основой является реляционная модель. Поэтому, в частности, включение в интегрированную систему локальной реляционной СУБД существенно проще и эффективнее, чем включение СУБД, основанной на другой модели данных.

Лекция 7. Язык реляционных баз данных SQL

SEQUEL/SQL СУБД System R

Язык для взаимодействия с БД SQL появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (Structered English Query Language) только частично отражает суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле уже являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; возможности определения структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов. В языке отсутствовали средства синхронизации доступа к объектам БД со стороны параллельно выполняемых транзакций: с самого начала предполагалось, что необходимую синхронизацию неявно выполняет СУБД.

Рассмотрим эти свойства языка немного более подробно.

Запросы и операторы манипулирования данными

Как известно, двумя фундаментальными языками запросов к реляционным БД являются языки реляционной алгебры и реляционного исчисления. При всей своей строгости и теоретической обоснованности эти языки редко используются в современных реляционных СУБД в качестве средств пользовательского интерфейса. Запросы на этих языках трудно формулировать и понимать. SQL представляет собой некоторую комбинацию реляционного исчисления кортежей и реляционной алгебры, причем до сих пор нет общего согласия, к какому из классических языков он ближе. При этом возможности SQL шире, чем у этих базовых реляционных языков, в частности, в общем случае невозможна трансляция запроса, сформулированного на SQL, в выражение реляционной алгебры, требуется некоторое ее расширение.

Существенными свойствами подъязыка запросов SQL являются возможность простого формулирования запросов с соединениями нескольких отношений и использование вложенных подзапросов в предикатах выборки. Вообще говоря, одновременное наличие обоих средств избыточно, но это дает пользователю при формулировании запроса возможность выбора более понятного ему варианта.

Существенной особенностью SQL является возможность указания в запросе потребности группирования отношения-результата по указанным полям с поддержкой условий выборки на всю группу целиком. Такие условия выборки могут содержать агрегатные функции, вычисляемые на группе. Эта возможность SQL главным образом отличает этот язык от языков реляционной алгебры и реляционного исчисления, не содержащих аналогичных средств.

Еще одним отличием SQL является необязательное удаление кортежей-дубликатов в окончательном или промежуточных отношениях-результатах. Строго говоря, результатом оператора выборки в языке SQL является не отношение, а мультимножество кортежей. В тех случаях, когда семантика запроса требует наличия отношения, уничтожение дубликатов производится неявно.

Операторы манипулирования данными UPDATE и DELETE построены на тех же принципах, что и оператор выборки данных SELECT. Набор кортежей указанного отношения, подлежащих модификации или удалению, определяется входящим в соответствующий оператор логическим выражением, которое может включать сложные предикаты, в том числе и с вложенными подзапросами.

В операторе вставки кортежа(ей) в указанное отношение заносимый кортеж может задаваться как в литеральной форме, так и с помощью внутреннего подоператора выборки.

Операторы определения и манипулирования схемой БД

В число операторов определения схемы БД SQL System R входили операторы создания и уничтожения постоянных и временных хранимых отношений (CREATE TABLE и DROP TABLE) и создания и уничтожения представляемых отношений (CREATE VIEW и DROP VIEW). В языке и в реализации System R не запрещалось использовать операторы определения схемы в пределах транзакции, содержащей операторы выборки и манипулирования данными. Допускалось, например, использование операторов выборки и манипулирования данными, в которых указываются отношения, не существующие в БД к моменту компиляции оператора. Конечно, эта возможность существенно усложняла реализацию и требовалась по существу очень редко.

Оператор манипулирования схемой БД ALTER TABLE позволял добавлять указываемые поля к существующим отношениям. В описании языка определялось, что выполнение этого оператора не должно приводить к недействительности ранее откомпилированных операторов над отношением, схема которого изменяется, и что значения вновь определенных полей в существующих кортежах отношения становятся неопределенными.

Определения ограничений целостности и триггеров

Язык SQL System R включал очень мощные средства контроля и поддержания целостности БД. Средства контроля базировались на аппарате ограничений целостности (ASSERTIONS). Фактически, ограничение целостности - это логическое выражение, вычисляемое над текущим состоянием БД, ложность которого соответствует нецелостному состоянию БД. Логическое выражение ограничения целостности могло содержать любой допустимый в языке предикат.

Более точно, ограничения целостности делились на два класса: проверяемые после выполнения оператора манипулирования данными и проверяемые при завершении транзакции или при выполнении специального оператора INFORCE INTEGRITY. Типы предикатов, которые можно использовать в операторах определения ограничений целостности разных классов, различаются. В операторах первого класса проверяется, фактически, текущий кортеж, с которым производится манипулирование. Во втором случае проверяются указанные в ограничении целостности отношения, т.е. все их кортежи. Различается и определяемая в языке реакция системы на нарушения ограничений целостности разных классов. В первом случае нарушение ограничения целостности приводит к откату транзакции в точку, непосредственно предшествующую операции манипулирования данными, выполнение которого вызвало нарушение ограничения целостности. Во втором случае ограничение приводит к полному откату транзакции к ее началу.

Очень важным механизмом, определенным в языке SQL System R, является механизм триггеров. В контексте System R этот механизм рассматривался главным образом как средство автоматического поддержания целостности БД. При определении триггера указывалось условие проверки его применимости (имя отношения и тип операции манипулирования данными), условие применимости триггера (логическое выражение, построенное по правилам, близким к правилам для ограничений целостности первого класса) и действие, которое должно быть выполнено над БД в случае истинности условия применимости. Такое действие могло быть выражено с помощью произвольного оператора манипулирования данными. Во время выполнения действия могли срабатывать другие триггеры и т.д.

Механизмы ограничений целостности и триггеров System R являлись очень мощными и общими, но реализация их очень трудна и накладна (как уже отмечалось, триггеры так и не были реализованы в System R). Дополнительную сложность в реализации создавал тот факт, что допускалось (по крайней мере не запрещалось языком) определение ограничений целостности и триггеров в пределах той же транзакции, в которой выполняются операторы манипулирования данными. При наиболее полной реализации требовалось бы большое число дополнительных действий во время выполнения транзакции. Кроме того, в ряде случаев отсутствие зафиксированной семантики соответствующих конструкций языка приводило к неоднозначному пониманию выполнения транзакций.

Представления базы данных

В языке допускалось использование хранимых отношений БД и представляемых отношений. Наиболее удачным решением было использование для определения представлений общего аппарата операторов выборки. Любой оператор выборки может быть использован для определения представления.

В языке отсутствуют какие-либо ограничения по поводу использования представлений: в любом операторе SQL, в котором допускается использование имени хранимого отношения, допускается и использование имени представления. В SQL ичего не говорится о рекомендуемом способе реализации доступа к представлениям, но при любом способе эффект должен быть таким, как если бы выполнить полную материализацию представления до выполнения оператора.

Массу проблем, исследований и предложений породила потенциальная возможность выполнения операторов манипулирования данными над представлениями. Понятно, что эта возможность легко реализуема для простых представлений, но в более сложных случаях не только реализация, но и семантика операций становится нетривиальной.

Определение управляющих структур

Внесение в реляционный язык, каким является SQL, явных операторов порождения и уничтожения структур физического уровня, поддерживающих эффективное выполнение запросов к БД, явилось в SQL чисто прагматическим решением, обеспечивающим возможность всех видов работ с БД с помощью одного языка.

В SQL System R упоминаются два вида таких структур: индексы и связи (links). Индекс в его абстрактном языковом представлении - это инвертированный файл, обеспечивающий доступ к кортежам соответствующего отношения на основе заданных значений одного или нескольких столбцов, составляющих ключ индекса. Операторы языка позволяли создавать и уничтожать индексы, но никаким образом не давали возможности явно указать на необходимость использования существующего индекса при выполнении оператора выборки, решение об этом возлагалось на реализацию.

С помощью оператора определения индекса можно было выразить два дополнительных утверждения, касающихся логической схемы отношения и физической структуры его хранения. Использование при определении индекса ключевого слова UNIQUE означало, что ключ этого индекса является возможным ключом соответствующего отношения. Фактически это означает наличие дополнительного механизма определения ограничения целостности отношения. Один из индексов для данного отношения мог быть определен с ключевым словом CLUSTERING. Это означает требование физической кластеризации во внешней памяти кортежей отношения с равными или близкими значениями ключа индекса.

Операторы определения связи позволяли в стиле сетевой модели данных организовать во внешней памяти списки кортежей указанного отношения. Как и в случае индексов, операторы позволяли создавать и уничтожать такие списки, но не давали возможности явно указать на необходимость использования существующих списков при выполнении операторов выборки. Большая трудоемкость поддержания списков при выполнении операторов манипулирования данными и трудность выполнения оценок стоимости их использования при выполнении операторов выборки привели к тому, что механизм связей исчез из языка уже на поздней стадии проекта System R. С тех пор этот механизм, насколько нам известно, не появлялся ни в одном варианте SQL.

Авторизация доступа к отношениям и их полям

Существенной особенностью языка SQL, появившейся в нем с самого начала, является обеспечение защиты доступа к данным средствами самого языка. Основная идея такого подхода состоит в том, что по отношению к любому отношению БД и любому столбцу отношения вводится предопределенный набор привилегий. С каждой транзакцией неявно связывается идентификатор пользователя, от имени

которого она выполняется (способы связи и идентификации пользователей не фиксируются в языке и определяются в реализации).

После создания нового отношения все привилегии, связанные с этим отношением и всеми его столбцами, принадлежат только пользователю-создателю отношения. В число привилегий входит привилегия передачи всех или части привилегий другому пользователю, включая привилегию на передачу привилегий. Технически передача привилегий осуществляется при выполнении оператора SQL GRANT. Существует также привилегия изъятия всех или части привилегий у пользователя, которому они ранее были переданы. Эта привилегия также может передаваться. Технически изъятие привилегий происходит при выполнении оператора SQL REVOKE.

Проверка полномочности доступа к данным происходит на основе информации о полномочиях, существующих во время компиляции соответствующего оператора SQL. Подобно тому, что мы отмечали в связи с ограничениями целостности и триггерами, в SQL System R отсутствовали какие-либо ограничения по поводу использования операторов GRANT и REVOKE. Это приводило к существенным техническим затруднениям в реализации, а иногда к неоднозначному пониманию поведения.

Долгое время подход к защите данных от несанкционированного доступа принимался практически без критики, однако в связи с распространяющимся использованием реляционных СУБД в нетрадиционных приложениях все чаще раздается критика. Если, например, в системе БД должна поддерживаться многоуровневая защита данных, соответствующую систему полномочий весьма трудно, а иногда и невозможно построить на основе средств SQL.

Точки сохранения и откаты транзакции

В SQL существовали два специальных оператора для установки так называемых точек сохранения транзакции и для отката транзакции к ранее установленной точке сохранения. В литературе, относящейся к System R, обсуждение этих возможностей практически не содержится, из чего неявно следует, что они не были реализованы.

Прямолинейная реализация этого механизма не вызывает особых технических затруднений, но и не очень полезна, потому что после выполнения частичного отката транзакции для успешного продолжения работы прикладной программы потребовалось бы и восстановить ее состояние в соответствующей точке, а это никак не поддерживается. Понятно, что при более тщательной проработке должны быть увязаны механизмы точек сохранения и контроля целостности. Например, было бы естественно, чтобы при выполнении оператора ENFORCE INTEGRITY, если какие-либо ограничения целостности нарушаются, происходил автоматический откат транзакции к ближайшей точки сохранения, в которой нарушения целостности БД не было. Это значительно усложнило бы реализацию, но было бы очень полезно. Аналогично, можно было бы использовать механизм точек сохранения при автоматических откатах транзакций по причине возникновения

Лекция 8. Case средства разработки информационных систем

Обзор некоторых CASE-систем.

Список производителей CASE - инструментов и ряд полезных ссылок можно найти по адресу http://sunny.aha.ru/~belikov/index.htm, вопросам использования CASE посвящена русскоязычная конференция news://fido7.su.dbms.case/, в Internet также доступна книга Вендрова А.М. CASE-технологии. Современные методы и средства проектирования информационных систем..

Power Designer компании Sybase.

В состав Power Designer входят следующие модули:

- *Process Analyst* - средство для функционального моделирования, поддерживает нотацию Йордона - ДеМарко, Гейна - Сарсона и несколько других. Имеется возможность описать элементы данных (имена, типы, форматы), связанные с потоками данных и хранилищами данных. Эт элементы передаются на следующий этап проектирования, причем хранилища данных могут быть автоматически преобразованыв сущности.

- *Data Analyst* - инструмент для построения модели "сущность-связь" и автоматической генерации на ее основе реляционной структуры. Исходные данные для модели "сущность-связь" могут быть получены из DFD-моделей, созданных в модуле Process Analyst. В ER-диаграммах допускаются только бинарные связи, задание атрибутов у связей не поддерживается. Поддерживаются диалекты языка SQL примерно для 30 реляционных СУБД, при этом могут быть сгенерированы таблицы, представления, индексы, триггеры и т.д. В результате порождается SQL-сценарий (последовательность команд CREATE), выполнение которого создает спроектированную схему базы данных. Имеется также возможность установить соединение с СУБД через интерфейс ODBC. Другие возможности: автоматическая проверка правильности модели, расчет размера базы данных, реинжиниринг (построение модельных диаграмм для уже существующих баз данных) и т.д.

- *Application Modeler* - инструмент для автоматической генерации прототипов программ обработки данных на основе реляционных моделей, построенных в Data Analyst. Может быть получен код для Visual Basic, Delphi, а также для таких систем разработки в архитектуре "клиент-сервер" как PowerBuilder, Uniface, Progress и др. Генерация кода осуществляется на основе шаблонов, соответственно управлять генерацией можно за счет изменения соответствующего шаблона.

Ознакомительную версию Power Designer, в которой заблокированы функции сохранения построенных моделей, можно получить с российского web-сервера комании Sybase.

Silverrun компании Silverrun Technologies Ltd.

CASE-система Silverrun состоит из следующих инструментов:

- *BPM* - построение DFD-диаграмм. Поддерживает нотации Йордона-ДеМарко, Гейна - Сарсона, Уорда-Меллора и многие другие. Данный инструмент позволяет автоматически проверить целостность построенной модели, причем список критериев проверки определяется пользователем (например: отсутствие имен у элементов модели, потоки данных типа "хранилище - хранилище" или "внешняя сущность - внешняя сущность" и т.д.)

- *ERX* - построение диаграмм "сущность-связь". Поддерживаются не только бинарные связи, но и связи более высоких порядков, имеется возможность определения атрибутов у связей. Построенные ER-модели с помощью внешней утилиты могут быть сконвертированы в реляционный структуры (в той версии, с которой я работал, при этом, к сожалению, терялись атрибуты связей).

- *RDM* - инструмент реляционного моделирования, позволяет генерировать SQL-скрипты для создания таблиц и индексов примерно для 25 целевых СУБД.

Следует отметить, что компания Silverrun Technologies Ltd является не только разработчиком CASE - инструментария, но также создала собственную методологию создания информационных систем, получившую название Datarun. Эта методология включает описание всех этапов жизненного цикла информационной системы, перечень и последовательность работ, требования к содержанию и оформлению документов и многое другое.

Ознакомительную версию Silverrun, можно скачать с сервера комании Argussoft. В этой версии имеются ограничения на количество элементов в создаваемых моделях.

BPWin и ERWin компании LogicWorks.

LogickWorks выпускает два взаимнодополняющих инструмента проектирования информационных систем:

- *BPWin* - функциональное моделирование на основе методологии IDEF0. Допускается также использовние нотации IDEF3 и DFD в нотации Йордона - ДеМарко. Имеется возможность экспорта построенных моделей в системы функционально-стоимостного анализа (ABC - Activity Based Costing) и информационного моделирования ERWin.

- *ERWin* - средство информационного моделирования, используется нотация IDEF1X. Поддерживаются свыше 20 целевых СУБД, имеется возможность генерации прототипов прикладных программ для Visual Basic, Delphi и т.д.

Designer/2000 компании Oracle.

Данный продукт компании Oracle, возможно, наиболее полно поддерживает все этапы создания приложений обработки данных. Однако, следует оговориться, что, в отличие от других средств, он поддерживает практически одну целевую СУБД - Oracle Server (имеется еще возможность генерации скриптов на ANSI SQL). То же самое касается и средств создания пользовательсокго интерфейса. Хотя возможна

генерация прототипов программ для языков Visual Basic, C, Java, полностью все возможности Designer/2000 реализуются только при использовании его вместе со средством разработки Oracle Developer/2000.

В состав Oracle Designer/2000 включены следующие модули:

- инструментарий анализа предметной области:
  - *Process Modeler* - средство анализа деловой активности организации. Позволяет создать модель структуры органнизации и привязать к этой модели функции, выполняемые в различных подразделениях, и информационные потоки между функциями. Содержит элементы бизнес-анализа.
  - *Dataflow Diagrammer* - в этом инструменте на базе DFD - диаграм детализуются функции, описаные в Process Modeler. Используется нотация Йордона - Де Марко.
  - *Function Hierarchy Diagrammer* - этот модуль автоматически выстраивает иерархии функций, определенных в двух предыдущих инструментах, имеется также возможность создавать прототипы функций.
  - *Entity Relationships Diagrammer* - инструмент моделирования данных (диаграмы "сущность-связь"), которыми оперируют функции, определенные в Dataflow Diagrammer. Используется нотация Баркера.
  - *Matrix Diagrammer* - инструмент для исследования связей между функциями и данными.
- генераторы структур:
  - *Database Wizard* - генерирует реляционные структуры из ER-диаграм.
  - *Application Wizard* - генерирует иерархию программных модулей конечного приложения обработки данных на основе иерархии функций. При этом может одновременно генерироваться несколько взаимосвязанных подсистем для различных подразделений одной организации. Во время генерации автоматически обнаруживаются одинаковые с точки зрения использования информационных объектов функции, которые согут быть объединены в одном модуле.
- инструментарий проектирования приложения:
  - *Data Diagrammer* - инструмент для доработки реляционных структур данных на основе нотации Баркера
  - *Module Structure Diagrammer* - инструмент для управления структурой программных модулей готового приложения. Здесь определяются типы модулей (меню, экранная форма, отчет) и их иерархии вызовов.

- *Module Data Diagrammer* - средство для проектирования экранного интерфейса программного модуля на основе используемых им данных. Позволяет без программирования весьма гибко управлять внешним видом и поведением генерируемого модуля.

- генераторы данных и кода:
  - *Server Generator* - генерирует базу данных на основе реляционных моделей.
  - *генераторы кода* - на основе моделей, построенных в Module Data Diagrammer, позволяет создать исходный код для Visual Basic, C, Java а также инструментов среды Oracle Developer/2000 (Oracle Forms, Oracle Reports). В последнем случае возможна циклическая доработка приложения: в сгенерированный прототип приложения в Developer/2000 вносятся изменения, которые видны для Designer/2000 и не теряются при повторной перегенерации.
  - *Preferences Navigator* - средство управления предпочтениями при генерации программных модулей. Плзволяет устанавливать многочисленные опции (например, внешний вид элементов экранного интерфейса) как для проекта в целом, так и для каждого модуля в отдельности.

Также в Oracle Designer/2000 имеется ряд других инструментов: утилита для интерактивного выполнения SQL-запросов, средства управления проектом и т.д.

Язык визуального моделирования (UML)

13 января 1997 года вышла версия 1.0 нового объединенного языка объектно-ориентированного моделирования Unified Modeling Language, созданного по запросу Object Management Group (OMG) - организации, ответственной за принятие стандартов в области объектных технологий и баз данных. После обсуждения, версия 1.1 UML в сентябре 1997 года представлена на голосование в OMG. Мир информационных технологий ждет результатов голосования, но формальности здесь уже не так важны, поскольку этот язык объектно-ориентированного моделирования уже фактически стал стандартом. Разработку UML поддержали и уже используют в качестве стандартов такие гранды рынка информационных технологий, как Microsoft, IBM, Hewlett-Packard, Oracle, DEC, Sybase, Logic Works и множество других.

Начиная с середины 60-х годов и до недавнего времени, широкое распространение получили структурные методологии анализа, проектирования и разработки информационных систем, которые характеризуются искусственным разделением (часто неоптимальным) системы на подсистемы, а также слабой взаимосвязью процессов и данных, присутствующих в системе. В отличие от них, объектные технологии, ориентированные на тесную взаимосвязь процессов и данных системы, позволяют программным системам быть более надежными, легко реализуемыми и

устойчивыми к изменениям. Кроме того, такая философия моделирования наиболее соответствует общим концепциям поведения систем реального мира.

Несмотря на явное преимущество объектно-ориентированных технологий анализа и проектирования перед структурными, их распространение было незначительным, поскольку ни один из методов не давал единой и цельной объектной модели системы. Каждый метод хорошо освещал одну или несколько сторон реальной системы, оставляя в тени множество других, не менее важных сторон. Кроме того, отсутствие единого стандарта очень мешало широкому распространению объектно-ориентированных методов при разработке программного обеспечения.

В течение 1994-96 годов создатели трех наиболее распространенных методологий - Гради Буч (BOOCH), Джим Рамбо (OMT - Object Modeling Technique) и Айвар Якобсон (OOSE - Object Oriented Software Engineering) объединили свои усилия под эгидой **Rational Software Corporation** на создание единого языка моделирования, который объединил бы все существенные и успешные разработки в данной области и стал бы стандартом языка объектного моделирования. Грандиозный труд, в котором наряду с Rational участвовали представители множества компаний, таких, как **Microsoft, IBM, Hewlett-Packard, Oracle, DEC, Unisys, IntelliCorp, Platinum Technology** и нескольких сотен других завершился созданием в январе 1997 года версии 1.0 Объединенного Языка Моделирования - Unified Modeling Language (UML), которая после бурного обсуждения в течение 1997 года превратилась в сентябре в версию 1.1 и была передана в OMG для принятия UML в качестве отраслевого стандарта расширяемого языка объектного моделирования. OMG - некоммерческая международная организация, в которую входят более 600 ведущих мировых компаний и отвечающая за принятие стандартов в области информационных технологий. Теперь же пришло время для стандарта расширяемого языка визуального моделирования, и, учитывая огромный успех UML в мире, мало кто сомневается в его скором принятии.

UML может быть применен на всех этапах жизненного цикла анализа бизнес-систем и разработки приложений. Различные виды диаграмм, поддерживаемые UML, и богатейший набор возможностей представления определенных аспектов системы делает UML универсальным средством описания как программных, так и деловых систем.

Диаграммы дают возможность представить систему (как деловую, так и программную) в таком виде, чтобы ее можно было легко перевести в программный код.

Кроме того ,UML специально создавался для оптимизации процесса разработки программных систем, что позволяет увеличить эффективность реализации программных систем в несколько раз и заметно улучшить качество конечного продукта.

Несмотря на свою молодость, UML уже прекрасно зарекомендовал себя на множестве успешных программных проектов. Средства автоматической

кодогенерации позволяют переводить модели на языке UML в исходный код объектно-ориентированных языков программирования, что еще более ускоряет процесс разработки.

Практически все мировые производители CASE-средств заявили о реализации поддержки UML в ближайших версиях своих продуктов. Но уже сегодня существуют множество CASE-средств, автоматизирующих процесс анализа и проектирования в UML (Rational Rose, Paradigm Plus, Select Enterprise, Microsoft Visual Modeler for Visual Basic и др.), поддерживающих множество языков программирования, таких, как C++, Java, Delphi, Power Builder, Visual Basic, Centura, Forte, Ada, Smalltalk, а также позволяющих осуществлять генерацию базы данных для большинства из существующих SQL-серверов. Модели, разработанные в UML, позволяют значительно упростить процесс кодирования и направить усилия программистов непосредственно на реализацию системы.

Диаграммы повышают сопровождаемость проекта и облегчают разработку документации к программной системе.

UML необходим:

- руководителям проектов, которые управляют распределением задач и контролем за проектом;
- проектировщикам информационных систем, которые разрабатывают технические задания для программистов;
- бизнес-аналитикам, обследующим реальную систему и проводящим инжиниринг и реинжиниринг бизнеса компании;
- программистам, которые реализуют модули информационной системы.

При модификации системы объектный подход позволяет легко включать в систему новые объекты и исключать устаревшие без существенного изменения ее жизнеспособности. Использование построенной модели при модификациях системы дает возможность устранить нежелательные последствия изменений, поскольку они не ломают устоявшейся структуры системы, а только изменяют поведение объектов.

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем. Язык UML одновременно является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения. Этот язык вобрал в себя наилучшие качества методов программной инженерии, которые с успехом использовались на протяжении последних лет при моделировании больших и сложных систем.

Язык UML основан на некотором числе базовых понятий, которые могут быть изучены и применены большинством программистов и разработчиков, знакомых с методами объектно-ориентированного анализа и проектирования. При этом базовые понятия могут комбинироваться и расширяться таким образом, что специалисты

объектного моделирования получают возможность самостоятельно разрабатывать модели больших и сложных систем в самых различных областях приложений.

Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса объектно-ориентированного анализа и проектирования в частности. Выбор выразительных средств для построения моделей сложных систем предопределяет те задачи, которые могут быть решены с использованием данных моделей. При этом одним из основных принципов построения моделей сложных систем является принцип абстрагирования, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. При этом все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

Другим принципом построения моделей сложных систем является принцип многомодельности. Этот принцип представляет собой утверждение о том, что никакая единственная модель не может с достаточной степенью адекватности описывать различные аспекты сложной системы. Применительно к методологии ООАП это означает, что достаточно полная модель сложной системы допускает некоторое число взаимосвязанных представлений (views), каждое из которых адекватно отражает некоторый аспект поведения или структуры системы. При этом наиболее общими представлениями сложной системы принято считать статическое и динамическое представления, которые в свою очередь могут подразделяться на другие более частные представления.) феномен сложной системы как раз и состоит в том, что никакое ее единственное представление не является достаточным для адекватного выражения всех особенностей моделируемой системы.

Еще одним принципом прикладного системного анализа является принцип иерархического построения моделей сложных систем. Этот принцип предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений. При этом исходная или первоначальная модель сложной системы имеет наиболее общее представление (метапредставление). Такая модель строится на начальном этапе проектирования и может не содержать многих деталей и аспектов моделируемой системы.

Язык UML предназначен для решения следующих задач:

1. Предоставить в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, специально предназначенный для разработки и документирования моделей сложных систем самого различного целевого назначения.

Речь идет о том, что важным фактором дальнейшего развития и повсеместного использования методологии ООАП является интуитивная ясность и понятность основных конструкций соответствующего языка моделирования. Язык UML включает в себя не только абстрактные конструкции для представления метамоделей систем, но и целый ряд конкретных понятий, имеющих вполне определенную семантику. Это позволяет языку UML одновременно достичь не только

универсальности представления моделей для самых различных приложений, но и возможности описания достаточно тонких деталей реализации этих моделей применительно к конкретным системам.

Практика системного моделирования показала, что абстрактного описания языка на некотором метауровне недостаточно для разработчиков, которые ставят своей целью реализацию проекта системы в конкретные сроки. В настоящее время имеет место некоторый концептуальный разрыв между общей методологией моделирования сложных систем и конкретными инструментальными средствами быстрой разработки приложений. Именно этот разрыв по замыслу OMG и призван заполнить язык UML.

Отсюда вытекает важное следствие - для адекватного понимания базовых конструкций языка UML важно не только владеть некоторыми навыками объектно-ориентированного программирования, но и хорошо представлять себе общую проблематику процесса разработки моделей систем. Именно интеграция этих представлений образует новую парадигму ООАП, практическим следствием и центральным стержнем которой является язык UML.

Снабдить исходные понятия языка UML возможностью расширения и специализации для более точного представления моделей систем в конкретной предметной области.

Хотя язык UML является формальным языком - спецификаций, формальность его описания отличается от синтаксиса как традиционных формально-логических языков, так и известных языков программирования. Разработчики из OMG предполагают, что язык UML как никакой другой может быть приспособлен для конкретных предметных областей. Это становится возможным по той причине, что в самом описании языка UML заложен механизм расширения базовых понятий, который является самостоятельным элементом языка и имеет собственное описание в форме правил расширения.

В то же время разработчики из OMG считают крайне нежелательным переопределение базовых понятий языка по какой бы то ни было причине. Это может привести к неоднозначной интерпретации их семантики и возможной путанице. Базовые понятия языка UML не следует изменять больше, чем это необходимо для их расширения. Все пользователи должны быть способны строить модели систем для большинства обычных приложений с использованием только базовых конструкций языка UML без применения механизма расширения. При этом новые понятия и нотации целесообразно применять только в тех ситуациях, когда имеющихся базовых понятий явно недостаточно для построения моделей системы.

Язык UML допускает также специализацию базовых понятий. Речь идет о том, что в конкретных[7] приложениях пользователи должны уметь дополнять имеющиеся базовые понятия новыми характеристиками или свойствами, которые не противоречат семантике этих понятий в языке UML.

3. Описание языка UML должно поддерживать такую спецификацию моделей, которая не зависит от конкретных языков программирования и инструментальных средств проектирования программных систем.

Речь идет о том, что ни одна из конструкций языка UML не должна зависеть от особенностей ее реализации в известных языках программирования. То есть, хотя отдельные понятия языка UML и связаны с последними очень тесно, их жесткая интерпретация в форме каких бы то ни было конструкций программирования не может быть признана корректной. Другими словами, разработчики из OMG считают необходимым свойством языка UML его контекстно-программную независимость.

С другой стороны, язык UML должен обладать потенциальной возможностью реализации своих конструкций на том или ином языке программирования. Конечно, в первую очередь имеются в виду языки, поддерживающие концепцию ООП, такие как C++, Java, Object Pascal. Именно это свойство языка UML делает его современным средством решения задач моделирования сложных систем. В то же время, предполагается, что для программной поддержки конструкций языка UML могут быть разработаны специальные инструментальные CASE-средства. Наличие последних имеет принципиальное значение для широкого распространения и использования языка UML.

4. Описание языка UML должно включать в себя семантический базис для понимания общих особенностей ООАП.

Говоря об этой особенности, имеют в виду самодостаточность языка UML для понимания не только его базовых конструкций, но что не менее важно - понимания общих принципов ООАП. В этой связи необходимо отметить, что поскольку язык UML не является языком программирования, а служит средством для решения задач объектно-ориентированного моделирования систем, описание языка UML должно по возможности включать в себя все необходимые понятия для ООАП. Без этого свойства язык UML может оказаться бесполезным и невостребованным большинством пользователей, которые не знакомы с проблематикой ООАП сложных систем.

С другой стороны, какие бы то ни было ссылки на дополнительные источники, содержащие важную для понимания языка UML информацию, по мнению разработчиков из OMG, должны быть исключены. Это позволит избежать неоднозначного толкования принципиальных особенностей процесса ООАП и их реализации в форме базовых конструкций языка UML.

5. Поощрять развитие рынка объектных инструментальных средств.

Более 800 ведущих производителей программных и аппаратных средств, усилия которых сосредоточены в рамках OMG, видят перспективы развития современных информационных технологий и основу своего коммерческого успеха в широком продвижении на рынок инструментальных средств, поддерживающих объектные технологии. Говоря же об объектных технологиях, разработчики из OMG имеют в виду, прежде всего, совокупность технологических решений CORBA и UML. С этой точки зрения языку UML отводится роль базового средства для описания и документирования различных объектных компонентов CORBA.

6. Способствовать распространению объектных технологий и соответствующих понятий ООАП.

Эта задача тесно связана с предыдущей и имеет с ней много общего. Если исключить из рассмотрения рекламные заявления разработчиков об исключительной гибкости и мощности языка UML, а попытаться составить объективную картину возможностей этого языка, то можно прийти к следующему заключению. Следует признать, что усилия достаточно большой группы разработчиков были направлены на интеграцию в рамках языка UML многих известных техник визуального моделирования, которые успешно зарекомендовали себя на практике (см. главу 2). Хотя это привело к усложнению языка UML по сравнению с известными нотациями структурного системного анализа, платой за сложность являются действительно высокая гибкость и изобразительные возможности уже первых версий языка UML. В свою очередь, использование языка UML для решения всевозможных практических задач будет только способствовать его дальнейшему совершенствованию, а значит и дальнейшему развитию объектных технологий и практики ООАП.

7. Интегрировать в себя новейшие и наилучшие достижения практики ООАП.

Язык UML непрерывно совершенствуется разработчиками, и основой этой работы является его дальнейшая интеграция с современными модельными технологиями. При этом различные методы системного моделирования получают свое прикладное осмысление в рамках ООАП. В последующем эти методы могут быть включены в состав языка UML в форме дополнительных базовых понятий, наиболее адекватно и полно отражающие наилучшие достижения практики ООАП.

Чтобы решить указанные выше задачи, за свою недолгую историю язык UML претерпел определенную эволюцию. В результате описание самого языка UML стало нетривиальным, поскольку семантика базовых понятий включает в себя целый ряд перекрестных связей с другими понятиями и конструкциями языка. В связи с этим так называемое линейное или последовательное рассмотрение основных конструкций языка UML стало практически невозможным, т. к. одни и те же понятия могут использоваться при построении различных диаграмм или представлений. В то же время каждое из представлений модели обладает собственными семантическими особенностями, которые накладывают отпечаток на семантику базовых понятий языка в целом.

Использование SilverRun

Методология

Планирование и разработка комплексных информационных систем невозможны без тщательно обдуманного методологического подхода. Какие этапы необходимо пройти, какие методы и модели использовать, как организовать контроль за продвижением проекта и качеством выполнения работ - эти вопросы решаются методологиями программной инженерии. Методологий существует много, и главное в них - единая дисциплина работы на всех этапах жизненного цикла системы. Если учитываются все критические задачи и контролируется их решение, качество создаваемых систем значительно возрастает. При этом, в общем случае, не важно, какие конкретно методы были выбраны для решения этих задач.

Для различных классов систем используются свои методы разработки. Они определяются как типом создаваемой системы, так и средствами реализации. Вероятно, самыми распространенными по объемам разработок являются информационные системы бизнес-класса. Практически в каждой организации имеются специалисты, разрабатывающие или сопровождающие информационные системы. Спецификация этих систем в большинстве случаев состоит из двух основных компонентов: функционального и информационного. По способу сочетания этих компонентов подходы к представлению информационных систем можно разбить на два основных типа - структурный и объектно-ориентированный. Разумеется, объектно-ориентированные методы также являются структурными в прямом понимании этого слова. Но исторически в программной инженерии этот термин закрепился за рядом дисциплин: структурное программирование, структурный дизайн, структурный анализ. В структурных технологиях функциональная и информационная модели строятся отдельно, чаще всего в виде диаграмм потоков данных и диаграмм "сущность-связь". Объектно-ориентированные технологии рассматривают информацию неотъемлемо от процедур ее обработки. Модели объектно-ориентированных технологий описывают структуру, поведение и реализацию систем в терминах классов объектов.

Объектно-ориентированные технологии доминируют в области создания операционных систем, средств разработки и исполнения приложений, систем реального времени. Концепция объекта помогает бороться с быстро растущей сложностью систем. Кроме того, взаимодействующие электронные устройства, как и элементы программ, естественно представляются объектами.

В области создания бизнес-систем лидируют структурные технологии, так как они максимально приспособлены для взаимодействия с заказчиками и пользователями, не являющимися специалистами в области информационных технологий. А анализ опыта разработок информационных систем показал, что активное привлечение пользователей на этапах выявления требований и постановки задачи является критическим фактором успеха крупных проектов. При разработке систем бизнес-класса основные усилия затрачиваются именно на понимание и спецификацию требований пользователя, а для реализации используются покупные средства разработки приложений (чаще всего языки четвертого поколения) и системы управления базами данных (чаще всего реляционные).

В терминах вышесказанного, место системы SILVERRUN в технологиях программной инженерии можно определить следующим образом: это CASE-система верхнего уровня, предназначенная для инструментальной поддержки структурных методологий создания информационных систем бизнес-класса. Таким образом, эта система может быть использована специалистами, занимающимися анализом и моделированием деятельности предприятий, разработчиками информационных систем, администраторами баз данных.

Средства управления проектом

Для контроля выполнения всех предусмотренных выбранной методологией действий используются специальные средства. В них расписываются все шаги проектной деятельности, и для каждого шага назначаются ответственные за его

выполнение специалисты. Обычно средства управления проектом дают возможность прогнозировать сроки и стоимость выполнения проекта с учетом сложности задачи, количества и квалификации персонала. В системе регистрируются план-графики работ и выполнение заданий. На основании полученных в процессе работы данных система может пересчитать прогноз стоимости и длительности проекта. Также обычно имеется возможность учитывать результаты предыдущих проектов, подстраивая таким образом планирование под темпы работы конкретного коллектива. Использование средств управления проектом позволяет значительно усилить контроль проектной деятельности и гарантировать выполнение всех предусмотренных методологией действий.

CASE-система верхнего уровня

Как уже упоминалось, самой критической фазой жизненного цикла информационной системы является анализ требований, включающий спецификацию правил работы и разработку архитектуры систем. Программист может не знать правила растаможивания товаров или проведения финансовых операций. Но он должен создать систему, обеспечивающую корректное выполнение этих действий. Это возможно только при наличии их детальных спецификаций. А если на этапе проектирования был сделан неправильный выбор архитектуры системы, может значительно снизиться эффективность ее использования. Подобные ошибки невозможно компенсировать даже высоким качеством реализации. Правильно решить такие задачи можно, только работая совместно с пользователями. Не случайно тесное взаимодействие с пользователем и полная спецификация требований стоят на первом месте в списке факторов, определяющих успех или крах проекта. CASE-системы верхнего уровня предназначены для автоматизированной поддержки создания спецификаций, с одной стороны, понятных пользователю, с другой стороны, содержащих технические детали для разработчиков. Эти системы позволяют постепенно переходить от формирования бизнес-требований к генерации реализующих эти требования приложений. Ниже будет показано, как это осуществляется в системе SILVERRUN.

Средства поддержки проектирования систем

При разработке архитектуры системы "клиент-сервер" бывает необходимо проверить работоспособность принятого решения. Для этой цели предназначены специализированные средства, например эмуляторы систем распределенной обработки. С помощью таких средств можно оценить требуемую пропускную способность каналов связи и мощность рабочих станций. При больших масштабах, характерных для распределенных систем, потери, возникающие вследствие построения неработоспособной архитектуры, могут быть очень велики. Поэтому затрата времени и средств на предварительный анализ выбранной архитектуры чаще всего бывает оправдана.

Средства управления разработкой приложений

Коллективная разработка больших информационных систем значительно упрощается при использовании автоматизированных систем ее поддержки. Трекеры

ошибок и системы контроля версий позволяют синхронизировать работу удаленных групп разработчиков и создавать многоверсионные системы.

Языки разработки приложений четвертого поколения

Для массового создания информационных систем критична эффективность и относительная простота средств разработки приложений. Этим требованиям удовлетворяют языки четвертого поколения (4GL). Высокая скорость разработки, возможность повторного использования и хорошая переносимость созданных на этих языках приложений сделали языки четвертого поколения наиболее используемыми средствами разработки систем бизнес-класса.

SILVERRUN имеет возможность передать информацию в репозитории (или словари данных) наиболее распространенных средств создания приложений. При этом экономятся значительные трудозатраты разработчиков. Например, генератор приложений SILVERRUN для PowerBuilder может генерировать из модели данных до 75% кода приложения.

Лекция 9. Методологии и технологии проектирования ИС

Общие требования к методологии и технологии

Методологии, технологии и инструментальные средства проектирования (CASE-средства) составляют основу проекта любой ИС. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают выполнение процессов ЖЦ.

Технология проектирования определяется как совокупность трех составляющих:

- пошаговой процедуры, определяющей последовательность технологических операций проектирования;
- критериев и правил, используемых для оценки результатов выполнения технологических операций;
- нотаций (графических и текстовых средств), используемых для описания проектируемой системы.

Технологические инструкции, составляющие основное содержание технологии, должны состоять из описания последовательности технологических операций, условий, в зависимости от которых выполняется та или иная операция, и описаний самих операций.

Технология проектирования, разработки и сопровождения ИС должна удовлетворять следующим общим требованиям:

- технология должна поддерживать полный ЖЦ ПО;
- технология должна обеспечивать гарантированное достижение целей разработки ИС с заданным качеством и в установленное время;
- технология должна обеспечивать возможность выполнения крупных проектов в виде подсистем (т.е. возможность декомпозиции проекта на

составные части, разрабатываемые группами исполнителей ограниченной численности с последующей интеграцией составных частей). Опыт разработки крупных ИС показывает, что для повышения эффективности работ необходимо разбить проект на отдельные слабо связанные по данным и функциям подсистемы. Реализация подсистем должна выполняться отдельными группами специалистов. При этом необходимо обеспечить координацию ведения общего проекта и исключить дублирование результатов работ каждой проектной группы, которое может возникнуть в силу наличия общих данных и функций;

- технология должна обеспечивать возможность ведения работ по проектированию отдельных подсистем небольшими группами (3-7 человек). Это обусловлено принципами управляемости коллектива и повышения производительности за счет минимизации числа внешних связей;

- технология должна обеспечивать минимальное время получения работоспособной ИС. Речь идет не о сроках готовности всей ИС, а о сроках реализации отдельных подсистем. Реализация ИС в целом в короткие сроки может потребовать привлечения большого числа разработчиков, при этом эффект может оказаться ниже, чем при реализации в более короткие сроки отдельных подсистем меньшим числом разработчиков. Практика показывает, что даже при наличии полностью завершенного проекта, внедрение идет последовательно по отдельным подсистемам;

- технология должна предусматривать возможность управления конфигурацией проекта, ведения версий проекта и его составляющих, возможность автоматического выпуска проектной документации и синхронизацию ее версий с версиями проекта;

- технология должна обеспечивать независимость выполняемых проектных решений от средств реализации ИС (систем управления базами данных (СУБД), операционных систем, языков и систем программирования);

- технология должна быть поддержана комплексом согласованных CASE-средств, обеспечивающих автоматизацию процессов, выполняемых на всех стадиях ЖЦ.

Реальное применение любой технологии проектирования, разработки и сопровождения ИС в конкретной организации и конкретном проекте невозможно без выработки ряда стандартов (правил, соглашений), которые должны соблюдаться всеми участниками проекта. К таким стандартам относятся следующие:

- стандарт проектирования;
- стандарт оформления проектной документации;
- стандарт пользовательского интерфейса.

Стандарт проектирования должен устанавливать:

- набор необходимых моделей (диаграмм) на каждой стадии проектирования и степень их детализации;

- правила фиксации проектных решений на диаграммах, в том числе: правила именования объектов (включая соглашения по терминологии), набор атрибутов для всех объектов и правила их заполнения на каждой стадии, правила оформления диаграмм, включая требования к форме и размерам объектов, и т. д.;
- требования к конфигурации рабочих мест разработчиков, включая настройки операционной системы, настройки CASE-средств, общие настройки проекта и т. д.;
- механизм обеспечения совместной работы над проектом, в том числе: правила интеграции подсистем проекта, правила поддержания проекта в одинаковом для всех разработчиков состоянии (регламент обмена проектной информацией, механизм фиксации общих объектов и т.д.), правила проверки проектных решений на непротиворечивость и т. д.

Стандарт оформления проектной документации должен устанавливать:

- комплектность, состав и структуру документации на каждой стадии проектирования;
- требования к ее оформлению (включая требования к содержанию разделов, подразделов, пунктов, таблиц и т.д.),
- правила подготовки, рассмотрения, согласования и утверждения документации с указанием предельных сроков для каждой стадии;
- требования к настройке издательской системы, используемой в качестве встроенного средства подготовки документации;
- требования к настройке CASE-средств для обеспечения подготовки документации в соответствии с установленными требованиями.

Стандарт интерфейса пользователя должен устанавливать:

- правила оформления экранов (шрифты и цветовая палитра), состав и расположение окон и элементов управления;
- правила использования клавиатуры и мыши;
- правила оформления текстов помощи;
- перечень стандартных сообщений;
- правила обработки реакции пользователя.

Методология RAD

Одним из возможных подходов к разработке ПО в рамках спиральной модели ЖЦ является получившая в последнее время широкое распространение методология быстрой разработки приложений RAD (Rapid Application Development). Под этим термином обычно понимается процесс разработки ПО, содержащий 3 элемента:

- небольшую команду программистов (от 2 до 10 человек);

- короткий, но тщательно проработанный производственный график (от 2 до 6 мес.);
- повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Команда разработчиков должна представлять из себя группу профессионалов, имеющих опыт в анализе, проектировании, генерации кода и тестировании ПО с использованием CASE-средств. Члены коллектива должны также уметь трансформировать в рабочие прототипы предложения конечных пользователей.

Жизненный цикл ПО по методологии RAD состоит из четырех фаз:

- фаза анализа и планирования требований;
- фаза проектирования;
- фаза построения;
- фаза внедрения.

На фазе анализа и планирования требований пользователи системы определяют функции, которые она должна выполнять, выделяют наиболее приоритетные из них, требующие проработки в первую очередь, описывают информационные потребности. Определение требований выполняется в основном силами пользователей под руководством специалистов-разработчиков. Ограничивается масштаб проекта, определяются временные рамки для каждой из последующих фаз. Кроме того, определяется сама возможность реализации данного проекта в установленных рамках финансирования, на данных аппаратных средствах и т.п. Результатом данной фазы должны быть список и приоритетность функций будущей ИС, предварительные функциональные и информационные модели ИС.

На фазе проектирования часть пользователей принимает участие в техническом проектировании системы под руководством специалистов-разработчиков. CASE-средства используются для быстрого получения работающих прототипов приложений. Пользователи, непосредственно взаимодействуя с ними, уточняют и дополняют требования к системе, которые не были выявлены на предыдущей фазе. Более подробно рассматриваются процессы системы. Анализируется и, при необходимости, корректируется функциональная модель. Каждый процесс рассматривается детально. При необходимости для каждого элементарного процесса создается частичный прототип: экран, диалог, отчет, устраняющий неясности или неоднозначности. Определяются требования разграничения доступа к данным. На этой же фазе происходит определение набора необходимой документации.

После детального определения состава процессов оценивается количество функциональных элементов разрабатываемой системы и принимается решение о разделении ИС на подсистемы, поддающиеся реализации одной командой разработчиков за приемлемое для RAD-проектов время - порядка 60 - 90 дней. С использованием CASE-средств проект распределяется между различными командами (делится функциональная модель). Результатом данной фазы должны быть:

- общая информационная модель системы;
- функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков;
- точно определенные с помощью CASE-средства интерфейсы между автономно разрабатываемыми подсистемами;
- построенные прототипы экранов, отчетов, диалогов.

Все модели и прототипы должны быть получены с применением тех CASE-средств, которые будут использоваться в дальнейшем при построении системы. Данное требование вызвано тем, что в традиционном подходе при передаче информации о проекте с этапа на этап может произойти фактически неконтролируемое искажение данных. Применение единой среды хранения информации о проекте позволяет избежать этой опасности.

В отличие от традиционного подхода, при котором использовались специфические средства прототипирования, не предназначенные для построения реальных приложений, а прототипы выбрасывались после того, как выполняли задачу устранения неясностей в проекте, в подходе RAD каждый прототип развивается в часть будущей системы. Таким образом, на следующую фазу передается более полная и полезная информация.

На фазе построения выполняется непосредственно сама быстрая разработка приложения. На данной фазе разработчики производят итеративное построение реальной системы на основе полученных в предыдущей фазе моделей, а также требований нефункционального характера. Программный код частично формируется при помощи автоматических генераторов, получающих информацию непосредственно из репозитория CASE-средств. Конечные пользователи на этой фазе оценивают получаемые результаты и вносят коррективы, если в процессе разработки система перестает удовлетворять определенным ранее требованиям. Тестирование системы осуществляется непосредственно в процессе разработки.

После окончания работ каждой отдельной команды разработчиков производится постепенная интеграция данной части системы с остальными, формируется полный программный код, выполняется тестирование совместной работы данной части приложения с остальными, а затем тестирование системы в целом. Завершается физическое проектирование системы:

- определяется необходимость распределения данных;
- производится анализ использования данных;
- производится физическое проектирование базы данных;
- определяются требования к аппаратным ресурсам;
- определяются способы увеличения производительности;
- завершается разработка документации проекта.

Результатом фазы является готовая система, удовлетворяющая всем согласованным требованиям.

На фазе внедрения производится обучение пользователей, организационные изменения и параллельно с внедрением новой системы осуществляется работа с существующей системой (до полного внедрения новой). Так как фаза построения достаточно непродолжительна, планирование и подготовка к внедрению должны начинаться заранее, как правило, на этапе проектирования системы. Приведенная схема разработки ИС не является абсолютной. Возможны различные варианты, зависящие, например, от начальных условий, в которых ведется разработка: разрабатывается совершенно новая система; уже было проведено обследование предприятия и существует модель его деятельности; на предприятии уже существует некоторая ИС, которая может быть использована в качестве начального прототипа или должна быть интегрирована с разрабатываемой.

Следует, однако, отметить, что методология RAD, как и любая другая, не может претендовать на универсальность, она хороша в первую очередь для относительно небольших проектов, разрабатываемых для конкретного заказчика. Если же разрабатывается типовая система, которая не является законченным продуктом, а представляет собой комплекс типовых компонент, централизованно сопровождаемых, адаптируемых к программно-техническим платформам, СУБД, средствам телекоммуникации, организационно-экономическим особенностям объектов внедрения и интегрируемых с существующими разработками, на первый план выступают такие показатели проекта, как управляемость и качество, которые могут войти в противоречие с простотой и скоростью разработки. Для таких проектов необходимы высокий уровень планирования и жесткая дисциплина проектирования, строгое следование заранее разработанным протоколам и интерфейсам, что снижает скорость разработки.

Методология RAD неприменима для построения сложных расчетных программ, операционных систем или программ управления космическими кораблями, т.е. программ, требующих написания большого объема (сотни тысяч строк) уникального кода.

Не подходят для разработки по методологии RAD приложения, в которых отсутствует ярко выраженная интерфейсная часть, наглядно определяющая логику работы системы (например, приложения реального времени) и приложения, от которых зависит безопасность людей (например, управление самолетом или атомной электростанцией), так как итеративный подход предполагает, что первые несколько версий наверняка не будут полностью работоспособны, что в данном случае исключается.

Оценка размера приложений производится на основе так называемых функциональных элементов (экраны, сообщения, отчеты, файлы и т.п.) Подобная метрика не зависит от языка программирования, на котором ведется разработка. Размер приложения, которое может быть выполнено по методологии RAD, для хорошо отлаженной среды разработки ИС с максимальным повторным использованием программных компонентов, определяется следующим образом:

< **1000** функциональных элементов	один человек

1000-4000 функциональных элементов	одна команда разработчиков
> 4000 функциональных элементов	4000 функциональных элементов на одну команду разработчиков

В качестве итога перечислим основные принципы методологии RAD:

- разработка приложений итерациями;
- необязательность полного завершения работ на каждом из этапов жизненного цикла;
- обязательное вовлечение пользователей в процесс разработки ИС;
- необходимое применение CASE-средств, обеспечивающих целостность проекта;
- применение средств управления конфигурацией, облегчающих внесение изменений в проект и сопровождение готовой системы;
- необходимое использование генераторов кода;
- использование прототипирования, позволяющее полнее выяснить и удовлетворить потребности конечного пользователя;
- тестирование и развитие проекта, осуществляемые одновременно с разработкой;
- ведение разработки немногочисленной хорошо управляемой командой профессионалов;
- грамотное руководство разработкой системы, четкое планирование и контроль выполнения работ.

Структурный подход

Сущность структурного подхода к разработке ИС заключается в ее декомпозиции (разбиении) на автоматизируемые функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы "снизу-вверх" от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов.

Все наиболее распространенные методологии структурного подхода базируются на ряде общих принципов . В качестве двух базовых принципов используются следующие:

- принцип "разделяй и властвуй" - принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;

- принцип иерархического упорядочивания - принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к непредсказуемым последствиям (в том числе и к провалу всего проекта). Основными из этих принципов являются следующие:

- принцип абстрагирования - заключается в выделении существенных аспектов системы и отвлечения от несущественных;

- принцип формализации - заключается в необходимости строгого методического подхода к решению проблемы;

- принцип непротиворечивости - заключается в обоснованности и согласованности элементов;

- принцип структурирования данных - заключается в том, что данные должны быть структурированы и иерархически организованы.

В структурном анализе используются в основном две группы средств, иллюстрирующих функции, выполняемые системой и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм), наиболее распространенными среди которых являются следующие:

- SADT (Structured Analysis and Design Technique) модели и соответствующие функциональные диаграммы;

- DFD (Data Flow Diagrams) диаграммы потоков данных;

- ERD (Entity-Relationship Diagrams) диаграммы "сущность-связь".

На стадии проектирования ИС модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру ПО, структурные схемы программ и диаграммы экранных форм.

Перечисленные модели в совокупности дают полное описание ИС независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

Методология функционального моделирования SADT

Методология SADT разработана Дугласом Россом и получила дальнейшее развитие в работе. На ее основе разработана, в частности, известная методология IDEF0 (Icam DEFinition), которая является основной частью программы ICAM (Интеграция компьютерных и промышленных технологий), проводимой по инициативе ВВС США.

Методология SADT представляет собой совокупность методов, правил и процедур,

предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. Основные элементы этой методологии основываются на следующих концепциях:

- графическое представление блочного моделирования. Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описываются посредством интерфейсных дуг, выражающих "ограничения", которые в свою очередь определяют, когда и каким образом функции выполняются и управляются;
- строгость и точность. Выполнение правил SADT требует достаточной строгости и точности, не накладывая в то же время чрезмерных ограничений на действия аналитика. Правила SADT включают:
- ограничение количества блоков на каждом уровне декомпозиции (правило 3-6 блоков);
- связность диаграмм (номера блоков);
- уникальность меток и наименований (отсутствие повторяющихся имен);
- синтаксические правила для графики (блоков и дуг);
- разделение входов и управлений (правило определения роли данных).
- отделение организации от функции, т.е. исключение влияния организационной структуры на функциональную модель.

Методология SADT может использоваться для моделирования широкого круга систем и определения требований и функций, а затем для разработки системы, которая удовлетворяет этим требованиям и реализует эти функции. Для уже существующих систем SADT может быть использована для анализа функций, выполняемых системой, а также для указания механизмов, посредством которых они осуществляются.