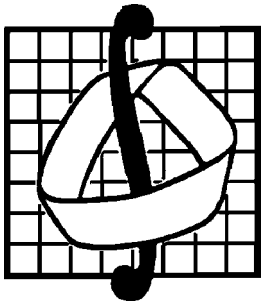
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. ЛОМОНОСОВА

Механико-математический факультет

Кафедра вычислительной математики



М.И.Кумсков

Базы данных и процессы их создания.

Введение.

Москва - 2004

ББК 32.97
Б 30
УДК 681.3

Учебное пособие содержит информационный материал, поддерживающий курс лекций «Введение в Базы Данных», который читается на механико-математическом факультете МГУ имени М.В.Ломоносова для студентов 5 курса как естественнонаучный курс. Рассмотрены вопросы визуального моделирования и логического проектирования, организации процессов создания БД и информационных систем. Описаны основы реляционной модели и языка SQL для формирования запросов к реляционной БД. Визуальное моделирование, логической структуры БД излагается в нотации UML.

Для студентов и аспирантов, начинающих работать с базами данных и информационными системами.

Рецензенты: доцент, к.ф.-м.н. К.Ю.Богачев

академик РАН Н.С.Бахвалов

Предисловие

В 1977 году Дж.Мартин[1] отмечал: «Историки будут рассматривать появление банков данных на ЭВМ и возможностей, связанных с ними, как шаг, изменивший природу эволюции общества и имеющий, возможно, большее значение, чем изобретение печатного станка. Уже сейчас, когда мы приближаемся к созданию информационно-ориентированного общества, около 20% объема национального продукта США расходуется на накопление, обработку и распространение знаний и информации во всех возможных формах…». Прогноз оказался верным - за последние 30 лет было произведено больше информации, чем за предшествующие 5000 лет.

Базы данных (БД) являются основой построения корпоративных информационных систем, которые принято относить к **фактографическим** системам. Далее будут рассматриваться только фактографические ИС, служащие для регистрации информации о событиях, важных для деятельности предприятия. Другой класс ИС называют **документальным** системами. Такие ИС работают с документами на естественном языке – с публикациями в периодической прессе, текстами законодательных актов, внутренней документацией предприятия. Иногда документальные ИС называют информационно-поисковыми системами (ИПС) или системами управления документооборотом – они предназначены для накопления и поиска документов по различным критериям на естественном языке. В последнее время к этим системам проявляется повышенный интерес. Это связано с актуальностью поиска и управления огромными объемами текстовой информации на предприятиях и в Интернете. Известно, что объем печатной информации в настоящее время удваивается каждые четыре-пять лет.

БД можно рассматривать как «промежуточный слой» программного обеспечения, расположенный между прикладными программами (или клиентскими приложениями - «клиентами») и данными, который позволяет непротиворечиво поддерживать одновременную работу многих пользователей с одними и теми же данными.

---

[1] Мартин Дж. **Организация баз данных в вычислительных системах**. 2-е издание: Пер. с англ. – М.: Мир, 1980.

Рассмотрены «логические аспекты» баз данных, которые имеют фундаментальный характер. При проектировании и создании БД возникает задача построения модели предметной области (модели «сущность-связь»), а также задача ее отображения в модель данных. Доминирующей моделью данных современных БД является реляционная модель, а стандартным языком работы с реляционными БД является непроцедурный язык SQL. Описаны основные понятия реляционной модели и правила составления простых SQL-запросов.

В курсе не рассматриваются вопросы, связанные с «физическим» уровнем организации БД, - вопросы производительности, организации «внутренних» данных, алгоритмов БД и т.п. Изложение основано на использовании следующих материалов:

Дейт К. **Введение в системы баз данных:** пер с англ. – М.: Издательский дом «Вильямс», 2000.

Хансен Г., Хансен Дж. **Базы данных: разработка и управление:** Пер с англ. – М.: Издательский дом БИНОМ, 1999.

Васкевич Д. **Стратегии клиент/сервер.** Руководство по выживанию для специалистов по реорганизации бизнеса. – К.: Диалектика, 1996.

Буч Г., Рамбо Дж., Джекобсон А. **Язык UML. Руководство пользователя.**: Пер. с англ. – М.: ДМК, 2000.

Грабер М. **Введение в SQL:** Пер. с англ. – М., ЛОРИ, 1996.

Материалы интернет сайтов *·www.citforum.ru,* **IBM Rational Software** (*www.rational.com*), **АйТи** (*www.it.ru*)

Учебное пособие предназначено для студентов пятого курса механико-математического факультета МГУ имени М.В.Ломоносова в качестве материала для подготовки по курсам «Базы Данных и процессы их создания» и «Корпоративные информационные системы». Курсы читаются автором на механико-математическом факультете МГУ с 1997 года.

Критические замечания и опечатки просьба сообщать автору на кафедру Вычислительной математики механико-математического факультета МГУ.

Ноябрь 2004 М.И.Кумсков

1. Основные понятия

Что такое База данных

Познакомимся с определениями баз данных и информационных систем. По Дейту - “Система БД – это, по сути, не что иное, как *компьютеризированная система хранения записей*. Саму же БД можно рассматривать как подобие электронной картотеки, т.е. как хранилище для некоторого набора занесенных в компьютер файлов данных… **Основная цель** системы БД – содержать информацию и предоставлять ее по требованию.

Основная функция, выполняемая СУБД – это предоставление многим пользователям возможности параллельно работать с данными, не вникая в детали на уровне аппаратного обеспечения и физической организации данных во внешней памяти.

Упомянутые в определении Дейта «компьютерные системы», обеспечивающие информационные потребности ее пользователей, принято называть *информационными системами* (ИС). Информационная система, использует два вида программного обеспечения:

1. Системное программное обеспечение (ПО) для поддержки БД, обычно называемое системой управления базой данных (СУБД).
2. Прикладное программное обеспечение (клиентские приложения ИС), которое использует средства СУБД для автоматизации конкретных бизнес задач, таких, например, как выставление счетов, анализ продаж и т.п.

Система управления базой данных (СУБД) – это системное программное обеспечение, которое выполняет следующие функции:

- Централизованное определение данных (на основе метаданных), задающих структуру БД. Используется язык определения данных – DDL (Data Definition Language)
- Одновременный доступ к данным многих пользователей;
- Разграничение прав доступа (защита данных). Используется язык контроля данных – DCL (Data Control Language)
- Обеспечение целостности (непротиворечивости) данных;

- Формирование запросов, команд обработки и извлечения данных. Используется язык преобразования данных – DML (Data Manipulation Language)
- Ориентированные на прикладного программиста возможности быстрого создания приложений.

По Саймону - «Управление базами данных – это формальная дисциплина, использующая разнообразные модели данных (иерархические, сетевые, реляционные и объектно-реляционные), в соответствии с которыми существуют и *управляются (посредством формализованных правил)* данные и их определения (метаданные). Эти правила связаны с параллельными процессами и целостностью таким образом, чтобы случайный доступ к данным и их модификация не могли иметь места. Чтобы квалифицировать среду хранения и управления данными как систему БД, она должна удовлетворять определенным условиям:

1. Данные должны иметь известный формат, который должен быть полностью определен для БД в целом, а не только для прикладной программы, которая их использует. Этот формат определяется с помощью метаданных, которые хранятся в самой БД.
2. Данные должны храниться, извлекаться и модифицироваться только с помощью специальной программы, которую принято называть *системой управления БД* (СУБД); прямой доступ к корпоративным данным из прикладных программ невозможен.
3. Данные должны быть субъектом управления транзакциями, которые обеспечивают поддержку целостности данных во время и после завершения операций над ними. Управление транзакциями становится особенно важным, когда много пользователей одновременно осуществляет доступ к данным.

Таким образом, СУБД представляет собой промежуточное системное программное обеспечение, используя которое прикладные программы работают со «своими» данными – СУБД позволяет приложениям «видеть» **только часть** данных и работать **только** с ними. Введение подобного промежуточного слоя ПО позволяет определить *логическую структуру данных*, отделив ее от *физической организации данных* при их хранении во внешней памяти. ***Данные становятся независимыми*** от приложений. Описание логической

структуры данных хранится в самой СУБД в виде ***метаданных*** и доступно как прикладным программам, так и системному ПО. Одной из основных задач СУБД является отображение логической структуры данных в физическую структуру хранения и оптимизация производительности ИС при работе с данными.

Разделение физической и логической структур данных (введение метаданных) явилось первой задачей БД, которая позволяла сделать данные ***независимыми*** от приложений и тем самым упростить разработку и модификацию прикладного ПО.

До появления БД было чрезвычайно трудно изменить способ использования данных в информационных системах. Разработчики приложений по-разному представляли данные в своих программах, и постоянно их модифицировали по мере возникновения новых задач. Модификации вызывали цепочку изменений в уже существующих программах, что обходилось весьма дорого. Появление БД позволило обеспечить **гибкость использования** данных:

во-первых, **данные стали независимы от программ**, использующих - данные могли добавляться и физически перестраиваться без изменения программ.

во-вторых, появилась возможность запрашивать и отыскивать информацию в БД без трудоемкого написания программ на обычном языке программирования - для этого стал использоваться специальный язык запросов к БД.

Представление данных

Описание данных и отношений между ними принято рассматривать на двух уровнях: на логическом и на физическом. Физическое представление определяет способы реального хранения данных во внешней памяти. За управление этим представлением отвечает администратор БД, решая вопросы оптимизации производительности. Логическое представление определяет, в каком виде данные видны программисту или пользователю. БД позволяет разделить эти два представления и обеспечить независимость логического представления от физического.

Структуру данных необходимо формально описать. Такое

описание структуры БД называют схемой. **Логическую схему**, которую видит разработчик конкретного приложения, иногда называют **внешней организацией БД** или **подсхемой БД**. Если объединить все внешние схемы, то получим **концептуальную (логическую) модель** БД, на основе которой могут быть сформированы различные внешние схемы, например, для будущих приложений. За целостность концептуальной модели БД и за ее развитие отвечает администратор данных. Иногда концептуальную модель называют общей моделью данных. При переходе к физическому представлению мы получаем **внутреннюю схему** и соответственно внутреннюю модель данных. Такие различные представления данных для БД (внешняя, концептуальная и внутренняя схемы) называют архитектурой ANSI/SPARC или трехсхемной архитектурой.

Запись

Информация в БД храниться в виде записей. Запись состоит их поименованных элементов (которые иногда называют **полями** записи или **атрибутами** записи). В навигационных БД (см. ниже) поля записи могут иметь сложную структуру, в отличие от реляционных БД, где атрибуты имеют только простой тип. Наборы записей одной структуры группируются в виде логического файла (или логического набора данных), который имеет имя. (В реляционных БД логическим файлам соответствуют поименованные таблицы данных). Логический файл можно рассматривать как электронную картотеку, а вся БД на концептуальном уровне – как набор взаимосвязанных (т.е. имеющих ссылки друг на друга) картотек. Тогда запись в картотеке – это «карточка», имеющая ссылки на другие карточки БД. Понятие записи и набора данных применимо как для логического, так и для физического представления БД.

Для идентификации записей используются ключи. **Потенциальный ключ** – это одно или несколько полей, по содержимому которых можно однозначно идентифицировать запись в наборе данных. При создании БД из всех потенциальных ключей набора данных выделяется **первичный ключ**. Для первичных ключей БД автоматически поддерживает ряд ограничений целостности: все ключи должны быть уникальны и недопустимы «пустые значения».

Роли - Кто работает с данными?

Пользователей БД можно разделить на группы, или (как иногда говорят) ввести **роли**:

1. **Прикладной программист -** отвечает за создание приложений. Прикладные программы выполняют над данными все стандартные операции - создание, чтение, обновление и удаление данных. Эти операции выполняются через соответствующие запросы к СУБД;
2. **Конечный пользователь -** использует клиентские программы, автоматизирующие его бизнес функции. Приложения выполняются на персональном компьютере пользователя, взаимодействуя с БД.
3. **Администратор БД** (системный администратор) - осуществляет техническое обслуживание СУБД. Он создает и модифицирует структуру БД, отвечает за быстродействие системы, за архивирование данных, за восстановление системы после сбоя, за техническое управление правами доступа (привилегиями).
4. **Администратор данных -** отвечает в целом за данные предприятия, за целостность их структуры, за предоставление прав доступа к данным.

Данные – это **важный актив предприятия**, и администратор данных должен «разбираться в данных» и понимать информационные потребностей различных заинтересованных лиц предприятия. Он решает, какие данные необходимо вносить в структуру БД и обеспечивает поддержку порядка при использовании данных. Администратор данных определяет ключевые правила по предоставлению прав доступа, т.е. по обеспечению политики *безопасности данных.* Таким образом, администратор данных работает **как менеджер**, а не как специалист по техническим вопросам. *Технический* специалист, ответственный **за реализацию решений *администратора данных*** – это **Администратор БД**.

Ответственности (задачи) указанных ролей, могут быть определены согласно ГОСТ 12207 или Rational Unified Process (см. ниже) для трех основных процессов жизненного цикла ИС: для процесса разработки, для процесса эксплуатации и для·процесса сопровождения.

Типы Баз Данных

При рассмотрении вычислений принято различать централизованные базы данных, расположенные на «универсальных ЭВМ» (mainframe) и БД архитектуры «клиент-сервер». Централизованные БД доминировали в 70-80-х годах, архитектура «клиент-сервер» является основной в настоящее время.

Навигационные БД

При централизованном подходе вычисления проводятся на «большой» универсальной вычислительной машине, к которой подключены терминалы. Электронные картотеки (или наборы данных) организованы в виде множества пар «Главная-Подчиненная». Запись главной картотеки содержит (физические) указатели на одну или несколько записей подчиненной картотеки. Общая организация БД на логическом уровне может быть представлена в виде **иерархии** (дерева), где узлу (вершины дерева) представляют картотеки, имеющие одинаковую структуру записей. Такие БД называются **иерархическими** - классическим представителем БД этого класса является система IMS фирмы IBM.

В 80-х годах начали использовать БД, логическая структура данных которых представляется в виде **сети -** внутри структуры данных разрешаются циклы. Такая модель данных называется **сетевой**[2]. И в сетевой, и в иерархической модели данных работа с данными состояла в «перемещении фокуса вычисления» по «графу картотек». Поэтому эти две модели данных иногда называют **навигационными** моделями, чтобы отличать их от реляционной модели.

Для определения данных и разработки прикладных программ в навигационных БД использовался (и продолжает использоваться) алгоритмический язык COBOL. Чтобы оценить масштаб современного применения навигационных БД, использующихся в «старых» системах (или в унаследованных системах - Legacy system), достаточно сказать, что число программистов, пишущих на языке COBOL, в настоящее время превышает два миллиона человек(!) во всем мире.

Доминирующая в настоящее время реляционная модель данных описана ниже.

[2] Не следует путать с сетевой организацией вычислений в рамках локальной вычислительной сети.

Сервер и клиенты

В Архитектуре "клиент-сервер" используются распределенные вычисления – на стороне пользователей (или «клиентов», т.е. на стороне прикладных программ) вычисления проводятся на ПЭВМ, которые имеют доступ к данным на сервере. **Сервер** – это логический процесс, который обеспечивает обслуживание запросов других процессов на предоставление доступа **к общему ресурсу** (например, к принтеру, к электронной почте и т.п.). Сервер не посылает данных клиенту, пока от него не поступит запрос. Сервер отвечает за управление синхронизацией обслуживания клиентов и за связи сервера с другими процессами. Таким образом, сервер – это не узел локальной вычислительной сети (ЛВС), а логический процесс, отвечающий за обработку запросов к БД.

Клиент – это процесс, посылающий серверу запрос на обслуживание. Клиент может начать взаимодействие с сервером, а сервер – нет. Как правило, клиент – это прикладная программа, выполняющаяся на рабочем месте (например, ПЭВМ) пользователя. Она обеспечивает автоматизацию бизнес-задач пользователя, например, ввод данных о заказе клиента. Клиент и сервер, как правило, выполняются на различных узлах сети[3], хотя это не обязательно - они могут физически выполняться на одной и той же машине. Технология "клиент-сервер" обеспечивает *распределенную обработку данных* – используются различные вычислители, связанные друг с другом сетью (Рисунок 1.1).

На физическом уровне СУБД может хранить данные на многих ЭВМ, - узлах сети. В этом случае говорят о *распределенной базе данных.* Клиенты могут не знать о таком распределении и рассматривать данные (на логическом уровне) как "хранимые в одном месте", т.е. на одной машине. Взаимодействие клиента с сервером осуществляется с использованием языка, ориентированного на работу с БД. Для реляционных БД таким языком является SQL. После обработки запроса сервер БД возвращает клиенту только данные, удовлетворяющие запросу.

[3] Сетью будем называть среду передачи данных между клиентом и сервером. Такой средой является, например, локальная сеть или комбинация Интернета и локальной сети.

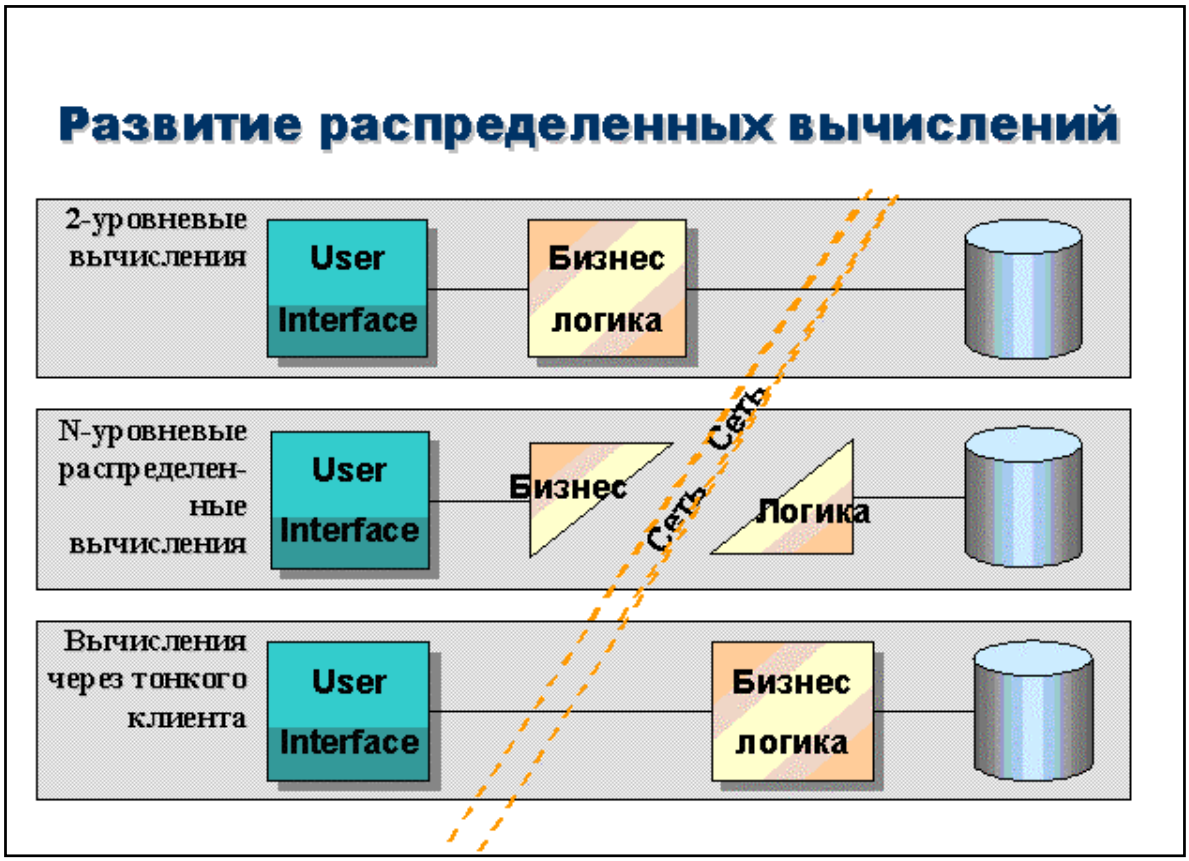


Рисунок 1.1 Распределенные вычисления в архитектуре «Клиент-сервер».

В архитектуре клиент-сервер вычисления классифицируют на три типа:

- Взаимодействие с пользователем – **интерфейсные** вычисления: получения и представление данных.
- Вычисления, согласно правилам прикладной области – **бизнес логика**.

Взаимодействия с СУБД – **работа с данными**.

Рисунок 1.1 иллюстрирует различные архитектурные решения клиент-сервер». Верхняя картинка - 2-х звенные (2-tier) вычисления, бизнес логика выполняется на клиенте («**толстый клиент**»). Средняя картинка – часть вычислений бизнес логики «вынесено» с клиента в сеть и выполняется на **сервере приложений,** который взаимодействует

с СУБД. Бизнес логика может выполняться на нескольких серверах приложений, взаимодействующих друг с другом. Это - **многозвенные** (n-tier) вычисления. Нижняя картинка – вся бизнес логика выполняется на сервере приложений в сети – «**тонкий клиент**» только принимает данные от пользователя и визуализирует (представляет в желаемой форме) результаты вычислений. Таким тонким клиентом, как правило, является интернет браузер (например, MS Internet Explorer).

Системы **файл-сервер**. При рассмотрении клиент-серверных реляционных СУБД их следует различать по важному критерию: является ли СУБД системой файл-сервер («настольные СУБД») или системой клиент-сервер (промышленные СУБД)? В настольных (или **файл-серверных системах**) «толстый клиент» проводит **все** вычисления, связанные с отработкой запроса к БД. Так, например, если в запросе используются три (соединяемые между собой) реляционные таблицы, то сервер ***передает клиенту копии этих таблиц*** по локальной сети. Отработка вычислений соединения таблиц **проводится на клиенте**. В результате на выходе может быть получен десяток записей (объемом несколько килобайт), из трех таблиц, общим объемом десятки (или сотни) мегабайт. Таким образом, файл-сервер отвечает только за централизованное хранение файлов с данными. К настольным СУБД относятся такие БД как MS Access, Paradox, FoxPro.

В промышленных СУБД все вычисления при поиске или модификации данных проводятся на (мощном) сервере. Клиенту передается только результат запроса. К промышленным СУБД относятся такие БД как: MS SQL Server, Oracle Server, IBM DB2 Server.

Транзакционные и аналитические системы

Реляционные БД (и соответствующие ИС) разделяются на два больших класса: системы для регистрации/обработки событий (операций, транзакций) и системы для аналитических расчетом (для анализа агрегированных данных). Первые системы наиболее распространены и называются **транзакционными** или **операционными** (не путать с операционными системами ЭВМ - такими как LINUX, Windows и т.п.). Транзакционные системы в литературе иногда называют **OLTP**-системами (On-Line Transaction Processing – обработка транзакций в режиме диалога).

Другой класс фактографических ИС – системы поддержки

принятия решений (**аналитические** системы). Эти системы обрабатывают большие объемы исторических данных (т.е. первичных данных, накопленных за различные периоды времени) с использованием средств прикладной статистики, прогнозирования, распознавания образов и т.п. Из первичных данных выделяется содержательная информация – зависимости, корреляции, тенденции. Требования бизнеса к скорости и качеству проведения такого анализа привели к необходимости специальной внутренней организации хранения данных в ИС. Такие системы называют **OLAP** системами (On-Line Analytical Processing).

Транзакции

Транзакцией называют последовательность операций с БД, в которой все операции должны быть успешно (без сбоев) выполнены, или отменены, если в середине транзакции произошел сбой. Результатом выполнения транзакций может быть два итога:

- Транзакция фиксирована (commit transaction)
- Произошел откат транзакции **(**rollback transaction)

Таким образом, транзакция – это логически неделимая единица работы с БД (принцип атомарности). Считается, что перед выполнением транзакции данные согласованы и целостны:

- Все ограничения **логической целостности** данных выполнены
- Состояния файлов корректно – ни один из элементов хранилища не является испорченным с точки зрения СУБД – т.е. обеспечена **физическая целостность** данных.

Назначение транзакций – обеспечить целостность и согласованность данных в течение всего времени работы информационной системы. Управление транзакциями в СУБД позволяет поддерживать непротиворечивость данных, например, при переводе денег с одного счета на другой. Если такую последовательность команд не оформить как транзакцию, то возможна ситуация, когда произойдет снятие денег с одного счета, но из-за промежуточного сбоя деньги на счет получателя не будут зачислены.

Всю логику обработки транзакций на себя берет СУБД – прикладной программист от этого освобожден. Ему требуется только

указать границы транзакции, т.е. операторы их начала и завершения. В СУБД присутствуют механизмы выявления некорректного завершения приложений, и все незавершенные транзакции таких приложений откатываются.

Так, если модификация в БД оформлена в виде транзакции, то клиент, запрашивающий (параллельно с выполнением модификации другим пользователем) данные, будет получать либо только "старые" данные (до модификации), либо только "новые" (после модификации), но он никогда не получит часть "старых" и часть "новых" данных.

Свойства транзакций

Свойство неделимости транзакции иногда называют **атомарностью** (atomicity) – транзакция рассматривается как *единая логическая работа*. Изменения данных или вносятся в БД целиком, или полностью «откатываются». Поддержка транзакций в БД – это основной механизм обеспечения целостности и непротиворечивости данных при параллельной работе многих пользователей с одними и теми же данными (иногда десятков и сотен клиентских программ одновременно).

Обеспечение транзакциями взаимной целостности данных называют свойством **согласованности** (consistency). Транзакция переводит БД из одного непротиворечивого состояния (на момент старта транзакции) в другое непротиворечивое состояние (на момент фиксации транзакции).

У пользователя, работающего в ИС, СУБД создает иллюзию, будто только он один *монопольно* работает с данными – он никому не мешает и ему никто не мешает создавать, читать, изменять и удалять данные (Create, Read, Update, Delete – CRUD – сокращение, используемое для обозначения операций клиента с БД). Такое свойство транзакций называется **изолированностью** (isolation). Оно гарантирует, что операции транзакций будут выполняться в БД логически правильно при интенсивной параллельной работе многих пользователей.

При успешном выполнении транзакции данные попадают на хранение в БД и становятся постоянными. Это свойство транзакций называют обеспечением **долговечности** (durability). Совместно все четыре свойства транзакции:

- **Атомарность** - **A**tomicity
- **Согласованность** - **C**onsistency
- **Изолированность** - **I**solation
- **Долговечность** – **D**urability

в литературе иногда называют ACID свойствами (по первым буквам английских названий свойств). Транзакцию, удовлетворяющую условиям ACID, иногда называют плоской транзакцией (flat transaction), поскольку в силу атомарности для нее нет никаких промежуточных точек фиксации.

Фиксация транзакции – это запись в БД (на постоянной основе) всех изменений, которые были произведены в процессе ее выполнения. До фиксации транзакции еще возможна отмена всех сделанных изменений и возврат БД в состояние, в котором находились данные до начала выполнения команд транзакции. Фиксация транзакции означает, что новые данные становятся ***видимыми*** другим пользователям БД.

Откат транзакции - происходит**,** если нормальное завершение транзакции невозможно, например, из-за нарушения целостности БД, или пользователь выдал специальную команду, или произошел разрыв соединения. Тогда БД возвращается в исходное состояние, все изменения в данных аннулируются. Механизм корректного исполнения отката и фиксации транзакций основан на использовании **журнала транзакций**. Чтобы иметь возможность сделать откат, СУБД должна сохранять все изменении, которые транзакция вносит в БД при выполнении CRUD команд. *В журнале транзакций сохраняются состояния строк (записей) до команды и после нее*. После этого изменения вносятся в БД. Если транзакция фиксируется, то в журнале делается соответствующая пометка. В случае отката, СУБД по журналу восстанавливает модифицированные строки до исходного состояния, отменяя все сделанные изменения.

Для корректного ведения журнала транзакций СУБД решает целый ряд проблем. Необходимо избегать потери изменений данных, когда несколько пользователей выполняют CRUD-операции над одними и теми же данными. Кроме этого, нужно исключить возможность чтения еще незафиксированных изменений. Для этого СУБД использует дисциплину совместной обработки (**сериализации**) транзакций:

Правило 1: Транзакция не может получить доступ к

«незафиксированным данным», т.е. к тем данным, в которых произведены изменения, но транзакция (в рамках которой изменялись данные) еще не зафиксирована.

Правило 2: Результат совместного выполнения транзакций должен быть эквивалентен результату их последовательного выполнения, т.е. с логической точки зрения, одна транзакция выполняется первой, а другая второй.

Сериализации транзакций реализуется через механизм блокировок.

Двухфазная фиксация транзакций

Если транзакция объединяет команды, выполняемые в одной БД, расположенной на одном сервере, то ее называют **локальной транзакцией**. Если же транзакция работает с распределенными БД, когда модифицируются записи, физически хранящиеся в разных БД на удаленных вычислительных узлах, то она называется **распределенной транзакцией.** С логической точки зрения локальные и распределенный транзакции должны обрабатываться одинаково – СУБД должна так организовать процесс выполнения распределенной транзакции, чтобы все локальные транзакции, входящие в нее, синхронно фиксировались на соответствующих узлах распределенной системы. Т.е. распределенная транзакция должна фиксироваться только в том случае, когда зафиксированы все локальные транзакции, ее составляющие. Если откатывается хотя бы одна локальная транзакция, то должен быть выполнен откат и всей распределенной транзакции.

Практическая реализация этих требований привела к введению понятия двухфазной фиксации транзакций (two phase commit) – фиксация распределенных транзакций проводится в два этапа (в две фазы).

Первый этап. СУБД, управляющая распределенной транзакцией, посылает команду «приготовиться к фиксации» всем узлам сети (серверам БД), задействованных при выполнении локальных транзакций. Сервера подтверждают свою готовность, если хотя бы от одного из серверов не получено подтверждение, то управляющий сервер проводит откат всех локальных транзакций.

Второй этап начинается, когда все локальные сервера (СУБД) готовы к фиксации своих транзакций. Управляющий сервер посылает

команду «зафиксировать транзакцию» всем локальным (распределенным) серверам. Затем он завершает фиксацию распределенной транзакции. Тем самым гарантируется синхронная фиксация распределенной транзакции на всех узлах вычислительной сети.

Восстановление после сбоя

СУБД обладает встроенными механизмами, обеспечивающими восстановление состояния информационной системы после сбоя. Для восстановления согласованного (целостного) состояния данных применяется механизм транзакций. Используется журнал транзакций, который содержит всю последовательность действий, изменяющих состояние БД. Принцип использования журнала прост:

- результаты зафиксированных транзакций должны присутствовать в восстановленной БД
- результаты незафиксированных (до сбоя) транзакций в БД должны отсутствовать

Таким образом, в БД восстанавливается последнее до сбоя согласованное состояние данных. Процесс восстановления основан на механизме отката незавершенных транзакций.

Базы данных и деятельность предприятия

При рассмотрении БД как набора электронный картотек, связанных друг с другом, возникает вопрос о составе и организации этих картотек. Представление их структуры в формальном виде приводит к понятию модели данных, которая должна отражать основные объекты деятельности предприятия[4]. В этом случае базу данных можно трактовать как «модель организации». Первоначально базы данных рассматривались как мощный технический инструмент. Позже выяснилось, что организация БД является эффективным средством понимания структуры предприятия. БД может служить моделью совокупности объектов компании как единого целого. В результате, чтобы получить «хорошую» БД, разработчики должны создать модель данных, которая является средством концептуального описания информационных потребностей предприятия.

Термин «база данных» может использоваться в различных контекстах и включает в себя ряд совершенно разных понятий. Как может трактоваться термин «*база данных»*? Существует три «уровня» интерпретации термина в различных контекстах: База Данных это -

1. **Центральное хранилище данных** (или совместно используемый механизм, предоставляющий общее хранилище взаимосвязанных и управляемых данных); БД – это структурированные данные.
2. Инструментальное средство поиска и анализа и отображения информации пользователей ; БД – это **система управления базой данных (СУБД)**.
3. Концептуальная модель (или модель для представления состояния организации, как в кратковременном, так и в долговременном аспектах). БД – это **модель сущностей предприятия**.

Таким образом, на верхнем уровне БД – это *модель*, представляющая хранимые (иногда говорят «устойчивые» - persistent) объекты конкретной организации. Понимание того, как построить эту модель правильно, является центральным моментом в создании ИС как совокупности приложений БД. Модели данных позволяют ставить и решать вопрос о том, как соотносятся потребности подразделений предприятия и ее информация (данные о хранимых объектах).

[4] Термин «предприятие» является обобщением для представления организаций, фирм, компаний, корпораций и т.п.

Общие тенденции развития ИС состоят в следующем:

- Распределенность и децентрализация ресурсов управления информацией;
- Неоднородность компонентов ИС;
- Использование стандартов;
- Использование моделирования бизнес процессов и ИС.

Распределенность и децентрализация. «Разукрупнение» компьютеров стало причиной появления небольших и мощных компьютерных систем. Такие вычислительные системы (настольных персональных компьютеров (ПК), рабочие станции, сервера) связываются друг с другом в сеть с помощью коммуникационных средств, например, локальных вычислительных сетей (Local Area Networks, LAN). Это позволяет иметь б***о***льшие вычислительные мощности, чем при использовании универсальных систем.

Неоднородность. Важным требованием в архитектуре «клиент-сервер» является возможность совместной работы разных СУБД от различных изготовителей. Унаследованные системы (Legacy system) – это, как правило, системы навигационных БД, которые управляют «старыми» данными предприятия. Они должны использовать информацию совместно с реляционными БД, а также с БД подразделений и «настольными БД», использующиеся на ПК.

Использование стандартов. Совместная работа (иногда говорят интероперабельность) в неоднородных средах основана на использовании стандартов. Пользователи должны иметь возможность замены аппаратных средств и/или программного обеспечения в системах, без необходимости отказа от интерфейсов, на которые опираются их приложения.

Использование моделирования бизнес процессов и ИС. На протяжении 70-80 годов при применении информационных систем в коммерческой деятельности наблюдалась разобщенность между структурой ИС и сущностями реального мира, для поддержки которых они разрабатывались. Примерами такого разобщения могут служить известные в 80-х годах проблемы:

1. Имелись приложения, которые требовали многократного повторения

ввода данных и одновременного использования значительного объема информации на бумажных носителях;

2. Процесс разработки ИС был основан на методологиях и моделях, которые требовали сложного отображения между концептуальным проектом и реализацией приложений.

ИС и Базы данных – исторический аспект

Движущими силами развития ИС и БД являлись две тенденции. Во-первых, выгода их использования в крупных коммерческих организациях (или корпорациях) и, во-вторых, стремительное развитие средств вычислительной техники. Кратко рассмотрим «как это было»[5]. До начала 80-х гг. роль компьютеров была ясной и понималась однозначно. Большие универсальные машины (мэйнфреймы - mainframe) являлись инструментом для решения задач бизнеса крупных корпораций. Существовали большие поставщики, такие как IBM и DEC, которые имели свои системы, свою стратегию и свои прикладные решения. Эти компании четко определяли организационные структуры, профессиональный состав персонала и консультантов на всех этапах применения технических и программных средств.

В 60-70-е годы компьютеры не столько *изменяли* что-либо в организациях, сколько предохраняли от этого. Большие организации, утопавшие до этого в бумагах, обнаружили, что использование информационных технологий позволяет им повысить эффективность существующих структур. Вместо того чтобы изменять существовавшие бизнес-процессы, компьютеры делали их более «быстрыми». Одно из утверждений тех времен гласит: процесс в бизнесе не должен преобразовываться до тех пор, пока он эффективно функционирует; "не *меняй* процесс - сделай его более *быстрым*".

Компьютеры 60-х годов не вызвали фундаментальных *изменений* в организациях бизнеса, они сыграли важную роль в *сохранении функциональной структуры* организаций того времени в первоначальном виде. Компьютер делал возможной сложную централизованно управляемую бюрократическую деятельность. Автоматизировались все существующие "негибкости" бюрократии.

---

[5] ·Васкевич Д. Стратегии клиент/сервер. Руководство по выживанию для специалистов по реорганизации бизнеса. – К.: Диалектика, 1996.

Возможности вычислительной техники значительно изменились в период с 1968 по 1972 гг. Базы данных, терминалы, сети и магнитные диски - все появилось практически в одно время. В результате почти все существовавшие до тех пор приложения устарели, и возникла потребность в целом классе новых приложений.

В начале 70-х годов в издании Harvard Business Review была описана концепция Баз данных, подключенных и управляемых с использованием терминалов - менеджерам предлагалось проведение следующих мероприятий:

- Рассматривать информацию как **ключевой корпоративный ресурс;**
- Построить **целостную и интегрированную БД**, охватывающую все сферы деятельности компании;
- Обеспечить **оперативный ввод и обновление** информации в БД, сразу, как только информация поступает в компанию.

С течением времени выяснилось, что:

1. Базы данных, функционирующие в реальном масштабе времени, могут сделать их предприятия **более конкурентоспособными**. Если информация доступна, оперативна и организована в БД, то это дает возможность компании работать эффективней.
2. Создание современных ИС, работающих в оперативном режиме, является **исключительно дорогостоящим делом**.

Одной из задач, с которой сталкивается руководители различных уровней, является **владение информацией,** позволяющей знать, что же в действительности происходит на предприятии. БД дает возможность задавать вопросы, позволяющие решить насущные проблемы, в рамках информационной системы предприятия.

Десятилетие 80-х - это время появления и развития персональных компьютеров, которые изменили взгляды пользователей на доступность вычислительных ресурсов и сформировали потребность в совершенно новом классе приложений и компьютерных систем. Фокус компьютерной индустрии переместились из сферы самих компьютеров и БД в сферу **процессов проектирования** и построения приложений. 80-е годы были периодом перехода процесса разработки приложений для бизнеса из области искусства в инженерную дисциплину. В ходе

этого процесса сформировались два важных результата: методология и CASE[6]-средства.

Методологию можно рассматривать как формальное предписание, руководство, описывающее процесс построения программных средств. В основе методологии лежит «простая» идея: проектирование ИС и БД может являться формальным процессом, который можно изучать и совершенствовать. Поставщики **CASE-систем** предложили инструментальные средства для автоматизации не только самого процесса проектирования, но и связанных с ним методологий. К началу 90-х годов CASE-инструменты занимали сегмент индустрии программного обеспечения, оцениваемый миллиардами долларов и предназначенный для предсказуемого построения более качественных приложений.

80-е годы можно назвать десятилетием персонального компьютера. В 1990г. в мире насчитывалось 30 тыс. больших универсальных компьютеров - самое большое количество, которое когда-либо может повториться снова. В том же году число персональных компьютеров составляло свыше 30 млн.

До второй половины 80-х годов большинство людей имели лишь очень ограниченный доступ к вычислительным ресурсам. В 1985 году уже миллионы пользователей были вовлечены в «персональное использование» текстовых редакторов и электронных таблиц - MultiMate, VisiCalc, Lotus 1-2-3, dBase, WordStar. Эти приложения были значительно проще в освоении и применении по сравнению с приложениями, используемыми на универсальных ЭВМ. Эти приложения позволяли пользователям думать по-новому, поскольку они могли **настраиваться под их конкретные потребности**. Персональные компьютеры 80-х не могли использоваться для построения реальных приложений для бизнеса. Поскольку профессионалы компьютерных подразделений предприятий пренебрежительно отнеслись к возможностям первых персональных компьютеров, то в течение 80-х годов выросло новое поколение «других» профессионалов. Вчерашние

[6] CASE – Computer Aided Software Engineering – инструментальные средства, поддерживающие работы в рамках той или иной методологии и автоматизирующие работы по проектированию ИС и БД. См.: Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. – М.: Финансы и статистика, 1998.

программисты VisiCalc, Lotus 1-2-3, BASIC и C, стали "гуру" в области технологии клиент/сервер, специалистами по интеграции таких систем.

Результаты использования ПК были очевидны: легкий в применении графический интерфейс пользователя, фактически неограниченных доступ к ресурсам компьютера с индивидуального рабочего места и новое представление о компьютерах и приложениях, выполняемых на них. Затраты на информационные технологии к концу 80-х годов **утроилась** (по сравнению с 1980г). Большинство компаний достигли отметки, когда они тратили на компьютеры больше, чем могли себе позволить. 90-е годы лучше всего охарактеризовать как десятилетие, когда сфера управления окончательно убедилась, что она не может больше мириться с ростом затрат на информационные технологии (ИТ). В целом ***затраты на ИТ вышли из-под контроля***.

80-90-е гг. характеризовались тесным сближением структур БД и тех предприятий, которые они поддерживают. Появились:

- Объектно-ориентированные технологии в программировании и управлении данными. Это позволило разработчикам говорить с заказчиками и пользователями **на одном языке** и упростить связь между концептуальным проектом и реализацией ИС.

- Управление потоками работ (workflow management), при котором определяется последовательность исполняемых задач, как пользователей, так и ИС приложений в рамках единого бизнес процесса. Системы, основанные на потоках работ, дают возможность моделировать «естественный путь осуществления производственной деятельности».

Вопросы

1. Можно ли создавать ИС без использования СУБД? Опишите преимущества и недостатки такого проектного решения.

2. Что такое и чем различаются логическое и физическое представление данных? Какова роль БД в их «различении»?

3. Что такое концептуальная модель БД?

4. Что такое метаданные, и для чего они используются в СУБД?

5. Опишите основные этапы развития БД и информационных систем.

6. Чем иерархическая модель данных отличается от сетевой?

7. Что такое сервер? Чем клиент-серверные СУБД отличаются от файл-серверных систем?

8. Что такое транзакции. Для чего они используются в БД?

9. Опишите ACID свойства транзакции.

10. Дайте определения, что такое ключ? Какие бывают ключи?

11. Что такое сервер приложений? Для чего он используется в ИС? Возможно ли построение информационной системы без сервера приложений?

12. Что такое многозвенная архитектура ИС?

2. Реляционная модель

Реляционная модель была предложена в 1970 году И.Ф.Коддом (E.F.Codd)[7], работавшим в исследовательской лаборатории IBM. Последующие десять лет эта модель интенсивно развивалась в университетах и научных организациях. Созданные в это десятилетие прототипы реляционных СУБД отличались невысокой производительностью, чтобы иметь коммерческий успех при использовании в корпоративных ИС. Ситуация существенно изменилась, когда появились более мощные ЭВМ и были разработаны новые методы физического хранения и доступа к данным.

Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных может быть сведено к совокупности отношений.

Наименьшая единица данных реляционной модели – это отдельное *атомарное* значение данных. Основными понятиями реляционных баз данных являются тип данных, домен, атрибут, кортеж, первичный ключ и отношение. Понятие *тип данных* в реляционной модели аналогично типу данных в языках программирования. Данные могут быть определены как символьные, числовые, битовые строки, "деньги", дата, время, временн***о***й интервал. *Доменом* называется именованное множество атомарных значений одного и того же типа (возможно с введением некоторых ограничений), т.е. это аналог подтипа. Домен можно рассматривать как область допустимых значений атрибута. *Атрибут* – это именованное значение.

Отношения - это именованное множество пар «атрибут-домен» (или, более строго, - «имя атрибута» - «имя домена»). *Степень отношения* – это число его атрибутов. Отношение степени один называют унарным отношением, степени два – бинарным, …, степени n – n-арным. Атрибуты не упорядочены - для ссылки на значение атрибута в кортеже используется имя атрибута. Значения атрибутов являются *атомарными*, т.е. среди значений домена не могут содержаться составные данные, такие, например, как списки, структуры или наборы значений.

[7] Codd E.F. A Relational Model of Data for Large Shared Data Banks. / Comm. ACM v.13, n.6, June 1970.

Различают описание отношения (схема или заголовок отношения) и его «содержимое» (тело отношения). *Заголовок* (схема) определяет фиксированное множество атрибутов A_1, A_2, ..., A_n, и взаимно однозначное соответствие между атрибутами A_i и определяющими их доменами D_i (i=1,2,...,n). *Тело* состоит из меняющегося во времени множества *кортежей*, где каждый кортеж состоит в свою очередь из множества пар «атрибут-значение» (A_i : V_i), (i=1,2,...,n), V_i является значением из домена D_i, который связан с атрибутом A_i. Таким образом, кортеж - это набор именованных значений заданного типа, или «строка в таблице».

Упорядоченность кортежей отсутствует – это свойство отношения является следствием определения отношения как множества кортежей. *Мощность отношения* – это текущее число его кортежей. Мощность отношения может изменяться в отличие от его степени.

Поскольку отношение – это множество, то никакие два кортежа не могут быть «дубликатами» друг друга. Пусть R – отношение с атрибутами A_1, A_2, ..., A_n. Множество атрибутов K=(A_i, A_j, ..., A_k) отношения R называется **ключом R,** если по К всегда можно однозначно идентифицировать кортеж в R. Один из ключей (например, состоящий из минимального набора атрибутов) принимается за его **первичный ключ**. Другие возможные ключи, если они есть, называются **альтернативными** ключами. Иногда первичным ключом называют «минимальный» (по числу атрибутов) ключ. Часто на практике для определения первичного ключа искусственно вводят специальный атрибут для идентификации кортежа (идентификатор объекта).

Неформальным представлением отношения в реляционной базе данных является **таблица**, заголовком которой является схема отношения, а строками - кортежи отношения. Имена атрибутов определяют имена столбцов таблицы. Иногда говорят "столбец таблицы", имея в виду "атрибут отношения". Будем использовать эту терминологию, которой придерживаются в большинстве коммерческих реляционных СУБД.

Реляционная база данных - это набор таблиц, имена которых совпадают с именами отношений в схеме БД. Основные структурные понятия реляционной модели данных (Таблица 2.1) имеют простую интуитивную интерпретацию.

Таблица 2.1. Сводка терминов реляционной модели

Неформальное определение	***Реляционный термин***
Таблица	Отношение (нет порядка кортежей)
Строка таблицы (запись)	Кортеж
Столбец (элемент записи)	Атрибут
Область определения элемента записи	Домен

Реляционная Алгебра

Кодд предложил инструмент для работы с отношениями – реляционную алгебру[8]. Каждая операция этой алгебры использует одно или несколько отношений в качестве своих операндов и создает новое отношение. Реляционная алгебра состоит из следующих девяти операций:

1) Объединение – теоретико-множественная операция
2) Пересечение – теоретико-множественная операция
3) Разность – теоретико-множественная операция
4) Произведение – реляционная операция
5) Проецирование – реляционная операция
6) Выбор – реляционная операция
7) Соединение – реляционная операция
8) Деление – реляционная операция
9) Присвоение – «программистская» операция

Первые три операции соответствуют операциям теории множеств, и они выполняются над отношениями как над множествами. При этом отношения, участвующие в теоретико-множественных операциях должны быть *«совместимы по объединению»*, т.е. они должны иметь одинаковую степень и атрибуты с «совместимыми» доменами. Для реляционных таблиц совместимость по объединению означает наличие одинакового числа и порядка столбцов, совместимых по типу. Такие столбцы реляционных таблиц называют *эквивалентными*.

[8] См. Мейер М. Теория реляционных баз данных: Пер. с англ. – М.: Мир, 1987.

Последняя операция – присвоение – это «стандартная» операция любого языка программирования, задающее имя величине (объекту).

Операция *произведение* создает новое отношение, соответствующее декартовому произведению кортежей отношений, участвующих в операции. Атрибуты отношения результата формируются как объединение атрибутов отношений аргументов - степень результата – это сумма степеней аргументов, а мощность – произведение мощностей аргументов. На основе этой операции строится операция соединения, описанная ниже.

Использование реляционных операций позволяет "разрезать" или "склеивать" отношения, соответствующие таблицам реляционной БД. Основными реляционными операциями являются операции проецирования, выбора и соединения.

1. Операция **проецирования** – отношение $R(A_1,A_2,...,A_k)$ проецируется на атрибуты A_1 и A_2:

$$\pi_{A_1 A_2}(R(A_1, A_2,..., A_k)) = R_2(A_1, A_2)$$

В результате проекции получается новое отношение $R_2(A_1,A_2)$, содержащее только заданные в операции атрибуты, при этом повторяющиеся кортежи удаляются. Степень нового отношения равна числу атрибутов, на которые выполняется проекция.

Эту операцию можно рассматривать как операцию «вычеркивания» столбцов из таблицы реляционной БД.

2. Операция σ выбора (**селекции**):

$$\sigma_{A_1=Rome,A_2=500}(R(A_1,..., A_k)) = R_2(A_1,..., A_k)$$

В результате селекции получается новое отношение $R_2(A_1,...,A_k)$, содержащее только кортежи, удовлетворяющие условию выбора. Степень отношения результата не меняется.

3. Операция **соединения –** два отношения R_1 и R_2 соединяются по атрибутам A_1 и B_1, принадлежащим одному домену:

$$R_1(A_1,\ldots,A_k)\underset{A_1=B_1}{\triangleright\triangleleft}R_2(B_1,\ldots,B_m)=R_3(A_1,\ldots,A_k,B_2,\ldots,B_m)$$

В результате соединения получается новое отношение $R_3(A_1,\ldots,A_k,B_2,\ldots,B_m)$ степени (k+m-1), содержащее кортежи, с объединением атрибутов обеих отношений (без повторения общего атрибута), удовлетворяющие следующему правилу:

Отношение-результат R_3 содержит декартово произведение множеств кортежей R_1 и R_2, из которого удалены все кортежи, значения атрибутов которых не удовлетворяет условию соединения (т.е. все кортежи, у которых $A_1 \neq B_1$).

Соединения, построенные на условии равенства атрибутов, называются *эквисоединениями*. Однако при определении соединения можно использоваться не только условие равенства, но и любое отношение «тета», допустимое на области определения соответствующих атрибутов: ($A_1\ \theta\ B_1$). Такое соединение называют «тета-соединение».

Операция соединения является **основной** операцией реляционной модели – можно сказать, что на ней основан весь фундамент реляционных БД. В связи с этим можно дать следующее неформальное определение реляционной СУБД:

1. Все данные представлены в БД в виде плоских таблиц.
2. СУБД поддерживает три реляционных оператора:
   - оператор выбора,
   - оператор проецирования
   - оператор соединения

Действительно, использование оператора соединения позволяет получить из нескольких «*базовых*» таблиц (т.е. из таблиц, явно хранящихся в БД) одну «производную» таблицу. Оператор проектирования позволяет выбрать из этой производной таблицы нужные атрибуты (столбцы), а оператор выбора – нужные кортежи (строки).

Подводим итог - чтобы считаться реляционной СУБД должна:

1. Представлять данные в виде плоских таблиц.

2. Поддерживать логическую структуру данных независимо от их физического представления.

3. Использовать непроцедурный язык высокого уровня для структурирования данных, выполнения запросов и модификации таблиц. (На практике для этого используют SQL).

4. Поддерживать основные реляционные операции (выбор, проектирования и соединения), а также такие теоретико-множественные операции как объединение, пересечение и дополнение.

5. Поддерживать виртуальные таблицы (**view** - представления), предоставляя альтернативный способ просмотра таблиц.

6. Различать в таблицах «неизвестные значения» (**null** или нулевые значения) и пропуски в данных («пропущенные значения»).

7. Обеспечивать поддержку целостности, авторизации, транзакций и восстановления данных.

Нормальные Формы

Для проверки корректности проекта реляционной БД ее таблицы должны быть нормализованы, т.е. находиться в третьей нормальной форме.

Таблица реляционной БД находится в **первой нормальной форме** (1НФ) – если все ее атрибуты атомарны (соответствуют базовым типам данных) и среди атрибутов «нет содержательного повторения атрибутов».

Таблица реляционной БД находится во **второй нормальной форме (**2НФ), если она находится в 1НФ, и нет атрибутов, зависящих от части первичного ключа. (Условие 2НФ относится к таблицам, имеющим составной первичный ключ).

Таблица реляционной БД находится в **третьей нормальной форме (3**НФ), если она находится в 2НФ, и все атрибуты зависят только от первичного ключа («сильная» формулировка 3НФ). Классическая (слабая) формулировка 3НФ: среди атрибутов таблицу нет транзитивной зависимости между атрибутами.

Язык SQL

SQL (Structured Query Language – *структурированный язык запросов*), – это универсальный язык высокого уровня для создания, модификации и управления данными в реляционных БД. SQL – это один из нескольких языков, который «вырос» из идеи использования реляционной модели данных для организации СУБД. В настоящее время SQL практически полностью доминирует в мире реляционных БД. Производители реляционных СУБД, первоначально использующие другие языки, сегодня переориентировались на SQL.

SQL впервые был выпущен на рынок компанией Oracle в 1979 году. В 1981 году IBM объявила о своем первом, основанном на SQL, программном продукте, SQL/DS. О создании собственных реляционных СУБД заявили компании Relational Technology и ряд других. К 1989 году насчитывалось более 25 различных SQL подобных СУБД, работающих на различных типах универсальных ЭВМ и ПЭВМ.

SQL до 1986 года не имел официального стандарта. В 1986 году стандарт SQL был совместно опубликован Американским институтом стандартов (ANSI – American National Standards Institute) и Международной организацией по стандартизации (ISO – International Standards Organization). В 1989 и 1992 годах стандарт языка SQL пересматривался. В настоящее время действует стандарт SQL 92, который в основном представляет собой расширенное множество спецификаций стандарта SQL 89.

SQL относятся к **непроцедурным языкам** высокого уровня. Так, на процедурном языке Си нужно записать шаг за шагом все инструкции, необходимые для выполнения задания. В отличие от языка Си, SQL просто декларирует, что нужно получить, а исполнение (т.е. как получить) возлагается на СУБД. При этом СУБД рассматривается как «черный ящик», и что происходит внутри него – пользователя БД не должно беспокоить. Его должно интересовать только получение правильного ответа на запрос к БД. Ограничивая пользователя в

вопросах исполнения операторов, SQL позволяет упростить сами операции и обеспечивает гибкость при реализации баз данных и внутренней оптимизации операций.

Запрос к БД на SQL состоит из набора предложений (или «секций» запроса) с помощью которых указывается, какие данные необходимо получить и на основе каких условий. С помощью единственного запроса на SQL можно (1) соединить несколько таблиц, получив промежуточную (временную) таблицу, а затем вырезать из нее (2) требуемые строки и (3) столбцы (селекция и проекция).

Имеются два типа SQL: Интерактивный и Вложенный. Большей частью, обе формы работают одинаково, но используются различно. **Интерактивный SQL** используется для функционирования непосредственно в БД, чтобы производить вывод для использования его заказчиком. В этой форме SQL, когда вводится команду, она сейчас же выполнится и результат можно увидеть немедленно. **Вложенный SQL** состоит из команд SQL помещенных внутри приложений, которые обычно написаны на алгоритмическом языке.
В этом разделе будет представлен SQL в интерактивной форме. Все что верно относительно интерактивного SQL в основном применимо и к вложенной форме.

SQL используется для **манипулирования данными** (**data manipulation**) – т.е. для выборки и модификации, для **определения данных** (**data definition**) и **администрирования данных** (**data control**). Принято выделять функциональные категории команд SQL:

- DML (Язык Манипулирования Данными) - это набор команд, которые определяют значения, представленные в таблицах;
- DDL (Язык Определения Данных) - так называемый Язык Описания Схемы, состоит из команд, которые создают объекты (таблицы, индексы, представления, и так далее) в базе данных;
- DCD (Язык Управления Данными) состоит из средств, которые определяют права (привилегии) пользователя для выполнения определенных действий в БД.

Это не различные языки, а команды SQL, сгруппированные по их функциям. Имеется две разновидности операций по манипулированию

данными – **выборка данных** (**data retrieval**) и модификация данных (**data modification**). *Выборка* – это поиск в БД необходимых данных, а модификация означает добавление, удаление или изменение данных. Операции по выборке (называемые запросами) осуществляют поиск в БД и отображают его результат. Во всех таких запросах SQL используется ключевое слово SELECT. Именно запросы будут детально рассмотрены в курсе. Все три операции по выборке данных – проектирование, выбор и соединение, - записываются с использование SELECT.

Различные типы запросов на SQL будем показывать на очень простой учебной реляционной БД[9] ("SCO" – Salespeople – Customers – Orders). БД "SCO" является минимально достаточной, чтобы легко отслеживать логику операторов, и достаточно полной, чтобы иллюстрировать главные понятия и практику использования SQL. – См. Таблица 2.2- Таблица 2.4.

Таблица 2.2. Таблица «Salespeople» (Продавцы) учебной БД "SCO"

snum	sname	city	comm
1001	Peel	London	.12
1002	Serres	SanJose	.13
1004	Motika	London	.11
1007	Rifkin	Barcelona	.15
1003	Axelrod	NewYork	.10
. . . .	. . . .	. . . . .	. . . . .
. . .	. . . . .	. . . . .	. . . . .

- **snum** – уникальный номер, назначенный каждому продавцу("номер служащего") – первичный ключ;
- **sname** – имя продавца;
- **city** – расположение продавца(город);
- **comm** – комиссионные продавца в десятичной форме.

[9] Грабер М. Введение в SQL: Пер. с англ. – М., ЛОРИ, 1996.

Таблица 2.3. Таблица «Customers» (Заказчики или Покупатели) учебной БД "SCO"

cnum	cname	city	rating	snum
2001	Hoffman	London	100	1001
2006	Clemens	London	100	1001
2008	Cisneros	SanJose	300	1007
2007	Pereira	Rome	100	1004
.....	. . . . .	. . . . .	. . . . .	. .

- **cnum** - уникальный номер Заказчика – первичный ключ;
- **cname -** имя заказчика;
- **city –** место расположения заказчика (город);
- **rating** - код рейтинга, т.е. уровень предпочтения данного заказчика перед другими. Более высокий номер указывают на большее предпочтение (рейтинг);
- **SNUM** - номер продавца, назначенного заказчику (из таблицы Продавцов) – внешний ключ.

Таблица 2.4. Таблица «Orders» (Заказы или Покупки) учебной БД "SCO"

onum	amt	odate	cnum	snum
3001	18.69	10/03/1999	2008	1007
3003	767.19	10/03/1999	2001	1001
3009	1713.23	10/04/1999	2002	1003
3007	75.75	10/04/1999	2004	1002
3010	1309.95	10/06/1999	2004	1002
3011	9891.88	10/06/1999	2006	1001
. . .	. . . . .	. . . .	. . . . .	. . . .

- **onum -** уникальный номер покупки (заказа) – *первичный ключ*;
- **amt –** величина покупки (заказа) в условной денежной единице;
- **odate -** дата покупки (заказа);
- **cnum -** номер заказчика, сделавшего покупку (из таблицы Заказчиков) – *внешний ключ;*
- **snum** - номер продавца, оформившего продажу (заказ) (из таблицы Продавцов) – *внешний ключ.*

Команда SELECT

Создание запроса. Запросы - наиболее часто используемый аспект **SQL**. Запрос - команда, которую вы даете вашей программе базы данных, и которая сообщает ей, чтобы она вывела определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера. Запросы обычно рассматриваются как часть языка **DML**. Все запросы в **SQL** состоят из одиночной команды. Структура этой команды обманчиво проста, потому что нужно расширять ее, чтобы выполнить сложные оценки и обработки данных. Эта команда называется - **SELECT**(**ВЫБОР**). Упрощенный синтаксис оператора **SELECT** имеет следующий вид:

SELECT <*список атрибутов «производной (выходной)» таблицы*>
FROM <*список таблиц, участвующих в запросе*>
WHERE <*условия поиска и/или условия соединения таблиц (если в запросе участвует несколько таблиц)*>;

В самой простой форме команда **SELECT** извлекает информацию из таблицы. Например, можно написать запрос, который выводит полную таблицу Продавцов, т.е. все столбцы (атрибуты) и все строки (кортежи), в следующем виде:

```
SELECT snum, sname, city, comm
    FROM  Salespeople;
```

Ключевое слово **SELECT** указывает, что эта команда - запрос. Далее следует **snum, sname** - это список атрибутов таблицы, которые выбираются запросом.

В предложении **FROM** – запроса указываются имена таблиц, используемых в качестве источника информации. В данном примере - это имя одной таблицы Продавцов (**Salespeople**).

Точка с запятой используется во всех *интерактивных* (т.е. выполняемых в диалоговом режиме) командах **SQL,** чтобы сообщать серверу БД, что запись команды завершена и она готова к выполнению.

Если в запросе нужно выбрать все атрибуты таблицы, то можно

использовать сокращение «звездочка (*)». Например, предыдущий запрос можно было бы переписать в следующем (более компактном) виде:

```
SELECT *
    FROM Salespeople;
```

Удаление Избыточных Данных

Особенностью **SQL** запросов, отличающих их от операций реляционной алгебры (а именно от операции проецирования), является необходимость **явного** указания, что следует удалять повторяющиеся строки таблицы-результаты. Для этого используется ключевое слово **DISTINCT.** Это аргумент, который ставится сразу после ключевого слова **SELECT** и позволяет устранять повторяющиеся значения строк из результата запроса. Предположим, что нужно узнать, какие продавцы в настоящее время имеют записи о продажах в таблице Заказов, - имеем следующий запрос:

Запрос: *Вывести номера продавцов, которые хотя бы один раз оформили заказ.*

```
SELECT snum
    FROM Orders;
```

В этом запросе будет выведено столько строк (значений атрибута **snum** с повторениями), сколько их хранит таблица **Orders**. Для получения списка номеров продавцов без повторений, запрос нужно видоизменить следующим образом, добавив ключевое слово **DISTINCT**:

```
SELECT DISTINCT snum
    FROM Orders;
```

Задание условий в запросах

Ключевое слово **WHERE** задает предложение команды **SELECT**, которое позволяет устанавливать ограничение, на выбираемые строки. Для этого в запросе используются предикаты (условия) - команда **SELECT** извлекает только те строки из таблицы, для которой утверждение предиката верно.

Запрос: *вывести имена и параметр «комиссионные» для всех продавцов в Лондоне.*

```
SELECT sname, comm
        FROM Salespeople
        WHERE city = 'London';
```

Когда предложение **WHERE** присутствует в запросе, сервер БД просматривает всю таблицу и проверяет для каждой строки условие предиката. Атрибут рейтинга покупателя (**rating)** таблицы **Customers** предназначен для объединения заказчиков в группы «предпочтения». Используем это числовое поле для выбора такой группы.

Запрос*: Показать всех покупателей, имеющих рейтинг 100.*

```
SELECT *
        FROM Customers
        WHERE rating = 100;
```

Одиночные кавычки не используются для задания чисел в предикатах.

Упражнение 1

1. Напишите запрос, выводящий номер заказа, его размер, и дату заказа.

2. Напишите запрос, который вывел бы все строки из таблицы покупателей, для которых номер продавца = **1001**.

3. Вывести таблицу продавцов в следующем виде: **city, sname, snum, comm**.

4. Напишите команду **SELECT**, которая выводит рейтинг (**rating**) и имя каждого заказчика из города «**San Jose**».

5. Напишите запрос, который показывает номера (**snum**) всех продавцов из таблицы заказов без повторений.

Операторы сравнения

Операторы сравнения указывают на тип сравнения между двумя значениями в предикате.

Операторы сравнения **SQL** приведены в таблице:

Символ	**Отношение**
=	Равный
>	Больше чем
<	Меньше чем
>=	Больше чем или равно
<=	Меньше чем или равно
<>	Не равно

Эти операторы имеют стандартные значения для числовых значений. Для строковых значений их результат зависит от кодировки строки: **ASCII** или **EBCDIC**. **SQL** сравнивает символьные значения в лексикографическом порядке кодов символов как это определено в соответствующей кодировке.

***Запрос**: вывести всех заказчиков с рейтингом (**rating**) выше 200.*

```
SELECT *
        FROM Customers
        WHERE rating > 200;
```

Булевские Операторы

Основные булевские операторы поддерживаются в **SQL** выражениях: **AND**, **OR**, и **NOT**, - логическое «**И**», «**ИЛИ**» и «**Отрицание**». Другие, более сложные, логические операторы (например, "исключающий или") могут быть сформированы из этих трех основных операторов.

Запрос: *Вывести всех заказчиков в городе «Berlin», которые имеют рейтинг выше 200:*

```
SELECT *
        FROM Customers
        WHERE city = ‘Berlin’ AND rating > 200;
```

Другие примеры (прочитайте запрос самостоятельно):

```
SELECT *
        FROM Customers
        WHERE city = ‘San Jose’ OR rating > 200;
```

```
SELECT *
        FROM Customers
        WHERE city = ‘London’ OR NOT rating > 200;
```

```
SELECT *
        FROM Customers
        WHERE NOT city = ‘Moscow’ OR rating > 200;
```

Упражнение 2

1. Напишите запрос, который выдает все заказы размером больше чем 1,000.

2. Напишите запрос, который выводит атрибуты **sname** и **city** для всех продавцов в Лондоне с комиссионными выше .10 .

3. Вывести всех покупателей с рейтингом =< 100, если они не находятся в Риме.

4. Что будет выведено в результате следующего запроса? - (сформулируйте запрос):

```
SELECT *
        FROM Orders
        WHERE (amt < 1000 OR
                (odate = 10/03/2003 AND cnum > 2017 ));
```

5. Как можно проще переписать такой запрос?

```
SELECT snum, sname, city, comm
        FROM Salespeople
        WHERE ( comm > + .12 OR comm < .14 );
```

Оператор IN

Оператор **IN** определяет набор (список) значений, в который должно быть включено значение атрибута.

***Запрос**: Найти всех продавцов, которые размещены в городе Barcelona или в городе London.*

Запись запроса без использования оператора **IN**:

```
SELECT *
        FROM Salespeople
        WHERE city = 'Barcelona' OR city = 'London';
```

Более простой способ записи этого же запроса использует оператор **IN**:

```
SELECT *
        FROM Salespeople
        WHERE city IN (‘Barcelona’, ‘London’);
```

Оператор IN определяет набор значений с помощью списка имен, заключенных в круглые скобки и отделенных запятыми. Когда список числовые значения, а не строки, то одиночные кавычки опускаются.

***Запрос**: Найти всех заказчиков, обслуживаемых продавцами, имеющими номера snum = 1001, 1007 или 1004:*

```
SELECT *
        FROM Customers
        WHERE cnum IN (1001, 1007, 1004);
```

Можно использовать NOT с IN:

```
SELECT *
        FROM Salespeople
        WHERE city NOT IN (‘London’, 'San Jose’);
```

или

```
SELECT *
        FROM Salespeople
        WHERE NOT city IN (‘London’, 'San Jose’);
```

Оператор BETWEEN

Оператор **BETWEEN** похож на оператор **IN**. Однако, в отличие от определения списка (как это делает **IN**), оператор **BETWEEN** определяет диапазон значений. Вводится ключевое слово **BETWEEN** с начальным значением, ключевое слово-разделитель **AND** и затем конечное значение. **BETWEEN** «чувствителен» к порядку, т.е. первое значение в предложении должно быть меньше второго.

Запрос*: Найти продавцов, имеющих комиссионные в интервале .10 и .12 (включая границы интервала):*

```
SELECT *
        FROM Salespeople
        WHERE comm BETWEEN .10 AND .12;
```

Для оператора **BETWEEN** значения **границы** (в предыдущем запросе - 0.10 и 0.12) попадают в запрос.

Оператор **BETWEEN** может работать с символьными полями в терминах ASCII, т.е. можно использовать **BETWEEN**, чтобы выбирать ряд значений из упорядоченных по алфавиту значений.

Запрос*: Выбрать всех заказчиков, чьи имена попадают в алфавитный диапазон “A-G”:*

```
SELECT *
        FROM Customers
        WHERE cname BETWEEN 'A' AND 'G';
```

Оператор LIKE

Оператор **LIKE** применим только к полям типа **CHAR** или **VARCHAR**, с которыми он используется, чтобы находить подстроки. При задании условий используются «групповые символы» (**wildcards)**. Имеются два типа групповых символов используемых с оператором **LIKE**:

1. Символ подчеркивания («_») - замещает любой одиночный символ. Например, 'b_t' будет соответствовать словам 'bat' или 'bit', но не будет соответствовать 'brat'.

2. Знак процента («%») - замещает последовательность любого (в том числе нулевого) числа символов. Например, '%p%t' будет соответствовать словам 'put', 'posit', или 'opt', но не 'spite'.

***Запрос**: Найти всех заказчиков, чьи имена начинаются с символа «G»:*

```
SELECT *
        FROM Customers
        WHERE cname LIKE 'G%';
```

Использование оператора **LIKE** удобно, если вы ищете, например, имя, но не помните, как они точно пишутся. Предположим, что вы не уверены, как записано по буквам имя одного из ваших продавцов «Peal» или «Peel». Вы можете использовать групповые символы, чтобы найти все возможные имена:

```
SELECT *
        FROM Salespeople
        WHERE sname LIKE 'P _ _ l %';
```

Групповой символ «%» необходим в конце строки условия, поскольку, если длина поля sname больше чем число символов в имени Peel, то при хранении оно дополняется пробелами до конца поля. Групповой символ ' % ' - просто соответствует этим пробелам.

Работа с NULL значениями

В таблице могут храниться записи, которые не имеют значений для некоторого атрибута, например, потому, что информация отсутствует, или это поле просто не заполнялось. **SQL** учитывает такой вариант, позволяя вводить значение **NULL** (пустой) в поле, вместо конкретного значения. Когда значение поля равно **NULL**, это означает, что СУБД специально промаркировала это поле как не имеющее значения для этой строки. Это отличается от назначения полю «нуля» или пробела. **NULL** не имеет типа данных, поэтому оно может помещаться в атрибут любого типа. Тем ни менее, **NULL** в **SQL** часто упоминается как нуль.

Предположим, что в БД нужно ввести данные о новом покупателе, которому еще не назначен продавец. Информацию о покупателе можно ввести со значением **NULL** в поле snum и заполнить это поле позже, когда продавец будет назначен.

Так как NULL указывает на отсутствие значения, вы не можете знать, каков будет результат сравнения с использованием NULL. Когда NULL сравнивается со значением, даже с другим таким же NULL, результат будет «неизвестен».

Выражение типа 'city = NULL' или 'city IN (NULL)' будет неизвестно, независимо от значения city. Следует делать различия между, например, значениями «неверно» и «неизвестно» - т.е. между строками, содержащими значения столбцов, которые не соответствуют условию предиката, и строками, которые содержат **NULL**. SQL имеет

специальный оператор **IS**, который используется с ключевым словом **NULL**, для записи условий на значения **NULL**.

***Запрос**: Найти все записи в о покупателях с NULL значениями в поле **city**:*

```
SELECT *
        FROM Customers
        WHERE city IS NULL;
```

***Запрос**: Найти все записи в о покупателях, поле **city** которых не пусто:*

```
SELECT *
        FROM Customers
        WHERE city NOT NULL;
```

или

```
        FROM Customers
        WHERE NOT city IS NULL;
```

Упражнение 3

1. Напишите запрос, который выводит все заказы, сделанные 3 и 4 октября 2003 с использование оператора **IN** и оператора **BETWEEN**.

2. Напишите запрос, который выводит всех покупателей, чьи имена начинаются с буквы, попадающей в диапазон от «A» до «G».

3. Напишите запрос, который выберет всех покупателей, чьи имена начинаются с буквы «С».

4. Напишите запрос, который выберет все заказы, имеющие нулевые значения или пропущенные значения **NULL** в поле **amt** (сумма заказа).

Агрегатные функции

Запросы могут вычислять обобщенное значение поля - это делается с помощью агрегатных функций:

- **COUNT(<поле>)** – выдает число не-NULL значений поля результирующей таблицы.
- **SUM(<поле>)** - вычисляет арифметическую сумму всех выбранных значений поля.
- **AVG(<поле>)** - вычисляет усреднение всех выбранных значений данного поля.
- **MAX(<поле>)** - вычисляет наибольшее из всех выбранных значений данного поля.
- **MIN(<поле>)** - вычисляет наименьшее из всех выбранных значений данного поля.

Агрегатные функции используются в предложении **SELECT** запроса, - они используют имя поля как аргумент. С функциями **SUM** и **AVG** могут использоваться только числовые поля. С функциями **COUNT**, **MAX** и **MIN** могут использоваться и числовые, и строковые атрибуты.

***Запрос**: найти сумму всех покупок*

```
SELECT SUM (amt)
    FROM Orders;
```

Нахождение среднего значения суммы - это похожая операция:

```
```

Функция **COUNT** считает число значений в данном столбце, или число строк в таблице. Когда она считает значения столбца, ее следует использовать с оператором **DISTINCT**, чтобы не подсчитывать повторения.

***Запрос**: Найти число продавцов, номера которых представлены в таблице Заказов:*

```
FROM Orders;
```

Использование **COUNT** со строками. Чтобы подсчитать общее число строк в таблице, используют функцию **COUNT** со звездочкой вместо имени поля, как, например, в следующем примере:

```
SELECT COUNT (*)
```

Функция «**COUNT** со звездочкой» подсчитывает все строки, включая «дубликаты».

Предложение GROUP BY

Предложение **GROUP BY** позволяет определять группу строк в терминах заданного атрибута и применять агрегатные функции к группам. Возможно формировать группы строк и применять к ним

агрегатные функции в одном предложении **SELECT**.

***Запрос**: найти наибольшее значение заказа, оформленного каждым продавцом.*

```
        FROM Orders
        GROUP BY snum;
```

Каждая группа состоит из всех строк с одним и тем же значением поля **snum**, и функция **MAX** применяется ***отдельно для каждой такой группы***. Таким образом, поле, к которому применяется **GROUP BY**, имеет по определению, константное значение на каждую группу. Можно использовать **GROUP BY** с несколькими полями:

***Запрос**: вывести наибольшее значение заказа, оформляемого каждым продавцом каждый день.*

```
SELECT snum, odate, MAX (amt)
    FROM Orders
    GROUP BY snum, odate;
```

Предложение HAVING

Предположим, что в предыдущем запросе нужно было бы вывести только те строки, которые имеют максимальный размер заказа, значение которого выше 3000. НЕЛЬЗЯ использовать агрегатную функцию в предложении **WHERE**, поскольку предикаты формируют условия в терминах одной строки, а агрегатные функции относятся к ***группе строк***. Это означает, что нельзя написать что-нибудь подобно следующему:

Внимание - ОШИБКА!

```
SELECT snum, odate, MAX (amt)
        FROM Orders
        WHERE MAX (amt) > 3000
        GROUP BY snum, odate;
```

Этот запрос не правилен. Чтобы увидеть максимальную стоимость заказов выше 3000, нужно использовать предложение **HAVING**. Предложение **HAVING** определяет критерии, используемые ***для задания условий на группы***, также как предложение **WHERE** задает условия для индивидуальных строк.

Правильной командой будет следующая:

```
SELECT snum, odate, MAX (amt)
        FROM Orders
        GROUP BY snum, odate
        HAVING MAX (amt) > 3000;
```

Аргументы в предложении **HAVING** подчиняются тем же правилам, что и в предложении **SELECT**, состоящего из команд, использующих **GROUP BY**. Они должны иметь одно значение на группу вывода. Следующая команда неправильна:

Внимание - ОШИБКА!

```
SELECT snum, MAX (amt)
        FROM Orders
        GROUP BY snum
        HAVING odate = 10/03/1988;
```

Поле **odate** не может быть использовано в предложении **HAVING**, потому что оно имеет ***больше чем одно значение на группу*** вывода, формируемую атрибутом **snum**. Чтобы избегать такой ошибки, предложение **HAVING** должно ссылаться только на агрегаты и поля выбранные **GROUP BY**.

Правильная запись предыдущего запроса:

```
SELECT snum, MAX (amt)
        FROM Orders
        WHERE odate = 10/03/1990
        GROUP BY snum;
```

HAVING может использовать в своих условиях аргументы, которые имеют только ***одно значение на группу вывода***.

Запрос*: Вывести наибольшие заказы для продавцов с номерами 1002 и 1007:*

```
SELECT snum, MAX (amt)
        FROM Orders
        GROUP BY snum
        HAVING snum B (1002,1007);
```

Упорядочение вывода

Таблицы содержат неупорядоченные (по определению отношения) наборы строк, и поэтому данные запроса, не обязательно появляются в какой-то определенной последовательности. Чтобы упорядочить результат запроса используют предложение **ORDER BY**. Эта команда упорядочивает вывод согласно значениям столбцов, указанных в предложении **ORDER BY**. Порядок сортировки можно определять по возрастанию (ключевое слово **ASC**) или по убыванию (ключевое слово **DESC**) для каждого столбца, указанного в предложении. По умолчанию установлено возрастание.

***Запрос**: Вывести таблицу Заказов, упорядоченную по номеру заказчика в обратном порядке (т.е. по убыванию номеров):*

```
SELECT *
        FROM Orders
        ORDER BY cnum DESC;
```

Можно дополнительно отсортировать таблицу с помощью второго столбца, например, с помощью поля **amt**, внутри упорядочения по полю **cnum**.

```
SELECT *
        FROM Orders
        ORDER BY cnum DESC, amt DESC;
```

Во всех случаях столбцы, которые упорядочиваются в предложении **ORDER BY**, должны быть указаны в предложении **SELECT**. Это обязательное требование. Следующая команда, например, будет запрещена:

Внимание – ОШИБКА!

```
SELECT cname, city
        FROM Customers
        ORDER BY cnum;
```

Так как поле cnum не присутствует в выводе запроса, предложение **ORDER BY** «не сможет найти его», чтобы использовать для упорядочения вывода.

Предложение **ORDER BY** может использоваться совместно с предложением **GROUP BY** для упорядочения групп - **ORDER BY** всегда записывается ***последней*** строкой запроса.

```
SELECT snum, odate, MAX (amt)
        FROM Orders
        GROUP BY snum, odate
        ORDER BY snum;
```

По умолчанию используется сортировка по возрастанию.

Упорядочение вывода по номеру столбца. Вместо имени столбца, можно использовать его порядковый номер (в выходной таблице) для указания поля сортировки. Другими словами, поле, упомянутое в предложении **SELECT** первым, для **ORDER BY** - это поле 1, независимо от того каким по порядку оно стоит в хранимой таблице. Например, вы можете использовать следующую команду, чтобы вывести поля **sname, comm** таблицы Продавцов, упорядоченные по убыванию значения комиссионных

```
SELECT sname, comm
            FROM Salespeople
            GROUP BY 2 DESC;
```

Упражнение 4

1. Напишите запрос, который сосчитал бы сумму покупок на 3 Октября 2003г.

2. Напишите запрос, который сосчитал бы число различных не-NULL значений поля city в таблице Заказчиков.

3. Напишите запрос, который выбрал бы наименьшее значение заказа для каждого заказчика.

5. Напишите запрос, который выбрал бы высший рейтинг (покупателя) для каждого города.

6. Напишите запрос, который сосчитал бы для каждого дня число покупателей, делающих заказы.

7. Напишите запрос, который выводил бы список заказчиков в нисходящем порядке. Вывод поля rating должен сопровождаться именем заказчика и его номером.

8. Напишите запрос, который бы выводил общие заказы на каждый день и располагал результаты в убывающем порядке.

Соединение таблиц

Одна из важных особенностей запросов **SQL** - это использование связей между таблицами и способность выводить информацию в запросе, используя эти связи для ***соединения таблиц***. **Соединение** является основной операцией в реляционных базах данных, которая формирует производные таблицы для «содержательных» запросов.

Таблицы, участвующие в соединении, задаются списком в предложении **FROM**. Предикаты (условия запроса) могут ссылаться на любые столбцы производной таблицы.

Имена таблиц и столбцов. Полное имя столбца состоит из имени таблицы, отделяемой точкой от собственно имени столбца. Например:

Salespeople. snum
Salespeople. city
Orders.odate

В предыдущих запросах можно было опускать имена таблиц, потому что запрос проводился только с одной таблицей. Когда делается запрос со многими таблицами, можно опускать имена таблиц, если все ее столбцы выходной таблицы имеют ***различные имена***.

В нашей БД мы имеем две таблицы со столбцами **city**. Если нужно связать эти столбцы (т.е. использовать соединение по этим столбцам), мы должны указать имена **Salespeople.city** или **Customers.city**, чтобы **SQL** мог их различать.

Запрос*: Сопоставить каждому продавцу его заказчиков в городе, в котором они живут, и вывести все пары продавцов и заказчиков для общего города.*

```
SELECT Customers.cname, Salespeople.sname, Salespeople.city
FROM Salespeople, Customers
WHERE Salespeople.city = Customers.city;
```

Так как поле **city** имеется и в таблице Продавцов и таблице

Заказчиков, имена таблиц должны использоваться как префиксы.

Соединение таблиц через соответствующие первичные и внешние ключи. Такое соединение очень часто используется и называется ***естественным соединением***.

Запрос: Показать имена всех заказчиков и соответствующих им имена продавцов, (т.е. которые их обслуживают)

```
SELECT Customers.cname, Salespeople.sname
        FROM Customers, Salespeople
        WHERE Salespeople.snum = Customers.snum;
```

Это - пример естественного соединения. Соединения, которые используют условие равенства, называются – эквисоединения (**соединениями по равенству)**. Соединения по равенству - это наиболее общий вид соединения, но имеются и другие варианты. Можно использовать любой из операторов сравнения в соединении – такие соединения называют «**тета-соединениями**». Вот пример такого вида соединения.

Запрос*: Вывести все комбинации имени продавца и имени заказчика так, что первый предшествует последнему в алфавитном порядке, а последний имеет рейтинг меньше чем 200.*

```
SELECT sname, cname
        FROM Salespeople, Customers
        WHERE sname < cname AND rating < 200;
```

В запросе можно соединять сразу несколько таблиц.

***Запрос**: Найти все заказы покупателей, которые находятся в городах, не совпадающих с городами, где находятся их продавцы.*

```
SELECT onum, cname, Orders.cnum, Orders.snum
        FROM Salespeople, Customers, Orders
        WHERE Customers.city < > Salespeople.city
        AND Orders.cnum = Customers.cnum
        AND Orders.snum = Salespeople.snum;
```

Объединение таблицы с собой. Можете задать соединение таблицы с собой, как объединение двух копий одной и той же таблицы. Таблица на самом деле не копируется, но **SQL** выполняет команду так, как если бы это было сделано. Другими словами, это соединение - такое же, как и любое другое соединение между двумя таблицами, за исключением того, что в данном случае обе таблицы идентичны.

Псевдонимы (алиасы). Синтаксис команды для объединения таблицы с собой, тот же что и для объединения многочисленных таблиц, в одном экземпляре. Когда соединяется таблица с собой, все «повторяемые» имена столбца, содержат префиксы имени таблицы. Чтобы ссылаться к этим столбцам внутри запроса, нужно иметь два различных имени для этой таблицы. Это можно сделать с помощью определения ***временных имен,*** называемых - переменными диапазона, переменными корреляции или просто – псевдонимами. Они определяются в предложении **FROM** – записывается имя таблицы, и через пробел, указывается ее ***псевдоним для данного запроса***.

***Запрос**: Найти все пары заказчиков, имеющих один и тот же рейтинг:*

```
SELECT first.cname, second.cname, first.rating
        FROM Customers  first, Customers  second
        WHERE first.rating = second.rating;
```

Запрос выполняется так, как если бы соединялись две таблицы называемые 'первая' и 'вторая'. Псевдонимы разрешают обработать одну таблицу как две независимых таблицы. Псевдонимы **first** и **second** были определены в предложении **FROM**. Псевдонимы существуют локально только для данной команды. Когда запрос завершен, псевдонимы, используемые в нем, больше не имеют значения.

Устранение Избыточности. Наш вывод в предыдущем запросе имеет два значение для каждой комбинации пары имен, причем второй раз в обратном порядке (например, "**John, Monica, 100**" и "**Monica, John, 100**"). Простой способ избежать этого повторения состоит в том, чтобы одно имя предшествовало другому в алфавитном порядке. Это делает предикат асимметричным, поэтому те же самые значения в обратном порядке не будут выбраны повторно:

```
SELECT first.cname, second.cname, first.rating
        FROM Customers first, Customers second
        WHERE first.rating = second.rating
        AND first.cname < second.cname;
```

Следующий запрос объединяет таблицу Пользователей с собой: чтобы *найти все пары заказчиков обслуживаемых одним продавцом*. В то же самое время, этот запрос объединяет таблицу заказчика с таблицей продавцов:

```
SELECT sname, Salespeople.snum, first.cname, second.cname
FROM Customers first, Customers second, Salespeople
WHERE first.snum = second.snum
 AND Salespeople.snum = first.snum
AND first.cnum < second.cnum;
```

Упражнение 5

1. Напишите запрос, который бы вывел список номеров заказов, сопровождающихся именем покупателя, который сделал эти заказы.

2. Напишите запрос, который бы выдавал имена продавца и заказчика для каждого заказа после его номера.

3. Напишите запрос, который бы выводил всех заказчиков, обслуживаемых продавцом с комиссионными выше 12% . Выведите имя заказчика, имя продавца, и ставку комиссионных продавца.

4. Напишите запрос, который вычислил бы сумму комиссионных продавца для каждого заказа покупателя с рейтингом выше 100. (Сумма комиссионных равна размеру заказы, умноженному на величину комиссионных продавца, оформившего заказ)

5. Напишите запрос, который бы вывел все пары продавцов, живущих в одном и том же городе. Исключите комбинации продавцов с самим собой, а также дубликаты строк, выводимых «в обратным порядке» (т.е. «Иванов-Петров» и «Петров-Иванов»).

6. Напишите запрос, который вывел бы все пары номеров заказов данного заказчика, имя заказчика, и исключал дубликаты из вывода, как в предыдущем запросе.

7. Напишите запрос, который вывел бы имена (**cname**) и города (**city**) всех заказчиков с такой же оценкой (**rating**), что у покупателя с номером 2001.

Подзапросы

С помощью SQL вы можете «вкладывать» один запрос («внутренний») внутрь другого запроса («внешнего»). Как правило, внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса.

Предположим, что мы знаем имя продавца: **Monika**, но не знаем значение его поля **snum**, и хотим извлечь все заказы этого продавца из таблицы Заказов:

```
SELECT *
        FROM Orders
        WHERE snum =
                (SELECT snum
                FROM Salespeople
                WHERE sname = 'Monika');
```

Чтобы оценить внешний (основной) запрос, **SQL** сначала должен оценить внутренний запрос (или подзапрос) внутри предложения WHERE. Единственной найденной строкой будет **snum = 1004**. **SQL** помещает его в предикат основного запроса вместо самого подзапроса, так чтобы предикат выглядел как «**WHERE snum = 1004**»

Основной запрос затем выполняется, как обычно. Подзапрос должен выбрать **один и только один** столбец, а тип данных этого столбца должен совпадать с тем значением, с которым он будет сравниваться в предикате. Если наш подзапрос возвратит более одного значения, это будет указывать на ошибку в наших данных.

Предикаты, включающие подзапросы, используют выражение в виде **«<скаляр><оператор сравнения> <подзапрос>»**, которое нельзя «переворачивать».

Нельзя, например, записывать запрос так:

Внимание - ОШИБКА!

```
SELECT *
        FROM Orders
        WHERE
                (SELECT DISTINCT snum
                FROM Orders
                WHERE cnum = 2001)
         = snum;
```

Функция, которая ВСЕГДА выдает одиночное значение для любого числа строк, - это агрегатная функция. Любой запрос, использующий одиночную функцию агрегата без предложения GROUP BY, будет выдавать одиночное значение, которое можно использовать в предикате основного запроса.

***Запрос**: Вывести все заказы, имеющие сумму покупок выше средней на 4-е Октября 2003:*

```
SELECT *
            FROM Orders
            WHERE amt >
              (SELECT AVG (amt)
                    FROM Orders
                    WHERE odate = 04/10/2003);
```

Подзапросы с использованием оператора **IN**. Можно записывать подзапросы, которые выдают несколько строк, если используется оператор **IN** (операторы **BETWEEN** и **LIKE** не могут использоваться с подзапросами). Когда вы используете **IN** с подзапросом, **SQL** формирует список значения по результату подзапроса.

***Запрос**: С использованием оператора **IN**, и подзапроса найти все Заказы для продавцов из Лондона:*

```
SELECT *
        FROM Orders
        WHERE snum IN
                (SELECT snum
                FROM Salespeople
                WHERE city = 'London');
```

Для получения того же результата можно использовать соединение:

```
SELECT onum, amt, odate, cnum, Orders.snum
FROM Orders, Salespeople
WHERE Orders.snum = Salespeople.snum
AND Salespeople.city = 'London';
```

Используя оператор **IN (**вместо оператора равенства «=»), можно устранить ограничение, состоящее в том, что подзапрос должен возвращать единственное значение:

```
SELECT onum, amt, odate
        FROM Orders
        WHERE snum IN
                (SELECT DISTINCT snum
                FROM Orders
                WHERE cnum = 2001);
```

***Запрос**: Вывести значение параметра «комиссионные» для всех продавцов, обслуживающих заказчиков в Лондоне:*

```
SELECT comm
        FROM Salespeople
        WHERE snum IN
                (SELECT snum
                FROM Customers
                WHERE city = 'London');
```

Суть всех подзапросов в том, что се они выдают **одиночный** столбец. Это обязательное условие на подзапрос, поскольку результат сравнивается со значением одного атрибута. Так, например, **«SELECT *»** не может использоваться в подзапросе.

Подзапросы в предложении **HAVING**. Эти подзапросы могут использовать свои собственные агрегатные функции, если они не выдают множественных значений:

Запрос: Подсчитать число заказчиков с рейтингом (**rating**) выше среднего в городе «SanJose».

```
SELECT rating, COUNT (DISTINCT cnum)
        FROM Customers
        GROUP BY rating
        HAVING rating >
                (SELECT AVG (rating)
                FROM Customers
                WHERE city = " San Jose');
```

Связанные подзапросы

В подзапросах можно «обратиться» к внутреннему запросу в предложении внешнего запроса **FROM**, сформировав – связанный (или зависимый) подзапрос. В этом случае подзапрос выполняется неоднократно, по одному разу для каждой строки таблицы основного запроса.

Запрос: Найти всех покупателей в заказах на 3-е октября 2003

```
SELECT *
        FROM Customers outer
        WHERE 03/10/2003 IN
                (SELECT odate
                FROM Orders inner
                WHERE outer.cnum = inner.cnum);
```

Здесь "внутренний"(**inner**) и "внешний"(**outer**), это псевдонимы. Так как значение в поле **cnum** внешнего запроса меняется, внутренний запрос должен выполняться отдельно ***для каждой строки внешнего запроса***. Строка внешнего запроса, для которого внутренний запрос каждый раз будет выполнен, называется - текущей *строкой-кандидатом*. Процедура вычисления зависимого подзапроса имеет следующий вид:

1. Выбрать строку из таблицы, именованную во внешнем запросе. Это будет *текущая строка-кандидат*.
2. Сохранить значения полей из строки-кандидата в псевдониме с именем в предложении FROM внешнего запроса.
3. Выполнить подзапрос. Везде , где есть ссылка на поля внешнего запроса, использовать значения текущей строки-кандидата. (Использование значения из строки- кандидата внешнего запроса в подзапросе называется ***внешней ссылкой***).
4. Оценить предикат внешнего запроса на основе результатов подзапроса, выполняемого в шаге 3. Если предикат верен, то строка-кандидат сохраняется для вывода, иначе – отбрасывается.
5. Повторить процедуру для следующей строки-кандидата таблицы, пока все строки таблицы не будут проверены.

Можно было бы решить задачу, используя запрос с соединением следующего вида:

```
SELECT *
        FROM Customers first, Orders second
        WHERE first.cnum = second.cnum
        AND second.odate = 03/10/2003;
```

***Запрос**: Вывести имена и номера всех продавцов, которые имеют более одного заказчика.*

```
SELECT snum, sname
        FROM Salespeople main
        WHERE 1 <
                (SELECT COUNT (*)
                FROM Customers
                WHERE snum = main.snum);
```

Сравнение таблицы с собой. Можно использовать зависимый подзапрос, основанный на той же самой таблице, что и основной запрос.

***Запрос**: Найти все заказы со значениями размера заказа выше среднего для их заказчиков:*

```
SELECT *
        FROM Orders outer
        WHERE amt >
                (SELECT AVG amt
                FROM Orders inter
                WHERE inner.cnum = outer.cnum);
```

Зависимые подзапросы в предложении **HAVING**. Когда

используется зависимый подзапрос в предложении **HAVING**, нужно ограничивать внешние ссылки к позициям, которые могли бы непосредственно использоваться в самом предложении **HAVING**. Предложение **HAVING** может использовать только агрегатные функции, которые указаны в их предложении **SELECT** или поля, используемые в их предложении **GROUP BY**.

***Запрос**: Найти сумму заказов (**amt**), сгруппировав их по датам, и удалив все даты, где сумма была бы, по крайней мере, на 2000 больше максимальной суммы:*

```
SELECT odate, SUM (amt)
        FROM Orders a
        GROUP BY odate
        HAVING SUM (amt) >
                (SELECT 2000 + MAX (amt)
                        FROM Orders b
                        WHERE a.odate = b.odate);
```

Подзапрос вычисляет значение **MAX(amt)** для всех строк с той же самой датой, что и у текущей агрегатной группы основного запроса. Это выполнено с использованием предложения **WHERE**. Сам подзапрос не должен использовать предложения **GROUP BY** или **HAVING.**

Упражнение 6

1. Напишите запрос, который бы использовал подзапрос для получения всех Заказов для заказчика с именем **Cisneros**. Предположим, что вы не знаете номера этого заказчика, указываемого в поле **cnum**.

2. Напишите запрос, который вывел бы имена и рейтинги всех заказчиков, чьи заказы превышают усредненные величины их заказов.

3. Напишите запрос, который бы выводил общую сумму всех покупок для каждого продавца, у которого эта общая сумма больше, чем сумма наибольшего заказа.

4. Напишите команду SELECT, использующую связанный подзапрос, которая выводит имена и номера всех заказчиков с максимальными для их городов рейтингами.

5. Напишите два запроса, которые выводят всех продавцов (по их имени и номеру), имеющих в своих городах заказчиков, которых они не обслуживают. Один запрос - с использованием соединения и один - со связанным подзапросом.

3. Методология создания ИС/БД

В создании ИС участвует не один и не два разработчика, а целая проектная команда. При этом число реляционных таблиц БД может достигать несколько сотен. Для успешного создания таких систем нужна методология, проверенная на практике и опирающаяся на мощные и удобные в использовании инструментальные CASE средства. Ниже описана такая методология – Rational Unified Process[10].

Проблема сложности

На практике при разработке информационных систем нередко бывают такие ситуации, когда разработчики становятся заложниками своего успеха. Например, небольшая группа разработала и сдала БД с набором простых приложений. Заказчик доволен и хочет развивать систему – заключается новый договор, в группу набираются новые люди. Разработчики расширяют систему, заказчику пока все нравится. В команде проекта занято уже не 5-7 сотрудников, а человек 20–30, но работы организованы «по-старому». В команде есть лидеры, которые ведут разработку в целом и держат все архитектурные особенности проекта у себя в голове. В какой-то момент проект начинает «пробуксовывать». Появляются факторы, часто незаметные при выполнении небольших проектов.

Первый из них — **сложность**. «Подкрадывается» она неожиданно. Например, система начинает работать неустойчиво, - в некоторые моменты она вдруг перестает реагировать на действия пользователя (т.е. «виснет»). Попытки найти и исправить ошибку приводят к тому, что возникают новые ошибки в самых неожиданных местах.

Известно, что в плохо структурированных программах (где вопросам проектирования архитектуры не уделялось должного внимания), исправление ошибки в 50% случаев приводит к появлению новой. В результате сопровождение и развитие системы становится не устойчивым, и приходится принимать решение: систему больше не модернизировать, потому что нет ни времени исправлять целый «веер» вновь возникающих ошибок.

Второй фактор - производительность: система нормально

[10] Кролл П., Крачтен Ф. «Rational Unified Process- это легко: Руководство по RUP для практиков» - Пер. с англ., М., КУДИЦ-Образ - 2004,

работает, но с какого-то момента начинает сильно отставать в производительности. Например, сделали информационную систему на «настольной» СУБД Access (т.е. использовали файл-серверную архитектуру). Все получилось замечательно: данные вносятся легко и быстро, все «летает». Но файл-серверная система плохо масштабируется - при возрастании размеров таблиц, числа одновременно подключенных пользователей резко возрастает сетевой трафик, по сети «гоняются» одни и те же таблицы… Система начинает «виснуть» сначала раз в неделю, потом - несколько раз в день, что, в конце концов, требует принятия кардинальных решений по пересмотру архитектуры системы.

Таким образом, часто причина бед кроется в несоответствии возросшей сложности создаваемой информационной системы и уровня организации работ в проекте. Сложность БД постоянно увеличивается в прямой пропорции с усложнением автоматизируемого бизнеса. Корпоративная ИС должна обеспечивать оперативный доступ к актуальной информации и иметь с возможность быстрой адаптации к изменению бизнес-правил, юридических и организационных аспектов работы.

Для разработчиков сложность часто имеет «латентный характер»: до определенной поры удается работать «по-старому». Но наступает момент, когда обнаруживается, что структура системы усложнились настолько, что без применения специальных средств и технологий больше обходиться нельзя. В качестве технологической «оснастки» должны выступать стандарты, четкое распределение обязанностей и ответственности, формализация процессов и операций при создании системы и структуры БД.

Другая сторона возможных неудач развивающихся ИТ-проектов заключается в том, что при управлении программистскими проектами возникают специфические ИТ-риски[11]. Хрестоматийный пример - закон Брукса, не вполне очевидный для тех, кто раньше не участвовал в разработке ИС: *добавление людей в «горящий» ИТ-проект только замедляет его выполнение*[12]:

«Программные проекты чаще проваливаются из-за нехватки

[11] Риск – это возможное событие с отрицательными последствиями для проекта. Один из принципов управления проектами состоит в знании «своих рисков» и минимизации их влияния на успех проекта.

[12] Брукс Ф. Мифический человеко-месяц или как создаются программные системы: Пер. с англ. - М., Символ-Плюс, 2001.

календарного времени, чем по всем остальным причинам вместе взятым. Почему эта причина неудач столь распространена?

Во-первых, слабо развиты наши методы оценок. В сущности, они отражают молчаливое и совершенно неверное предположение, что все будет идти хорошо.

Во-вторых, наши методы оценки ошибочно путают достигнутый прогресс с затраченными усилиями, неявно допуская, что скорость выполнения проекта пропорциональна количеству занятых в нем сотрудников.

В-третьих, поскольку менеджеры программных проектов не уверены в своих оценках, им часто недостает вежливого упрямства.

В-четвертых, выполнение графика работ слабо контролируется. Типовые опробованные в других инженерных дисциплинах методы считаются радикальными нововведениями при разработке программного обеспечения.

В-пятых, при обнаружении отставания от графика естественной и общепринятой реакцией является увеличение числа разработчиков. Это все равно, что тушить пламя бензином. В результате дела идут значительно хуже. Чем сильнее пламя, тем больше нужно бензина, и в итоге этот путь приводит к катастрофе…»

Среди классических «специальных» рисков при управлении разработкой программных проектов можно выделить следующие риски:

- неправильный выбор архитектуры на ранних этапах;
- появление потока запросов на изменения;
- систематическое изменение требований;
- необходимость постоянных переделок продукта и документации.

Таким образом, при создании информационных систем необходимо «правильно» организовать процесс разработки (иногда говорят, необходимо знать ключевые риски проекта). Под словом «правильно» обычно подразумевается организация работ, позволяющая не допускать известные и характерные ошибки аналогичных проектов.

Результатами «неправильной» организации проекта являются следующие «симптомы»:

1. Проект выходит из графика и бюджета, оценки времени и денег,

необходимые для завершения разнородны и субъективны.

2. Разработчики и заказчики разговаривают на «разных языках», что затрудняет отображение «неформальных» требований в формальную спецификацию системы. (Разработчики говорят – «Заказчик что-то хочет, только сказать об этом не может»). В результате трудно работать с меняющимися требованиями, приводящим к постоянным переделкам приложений и доработке структуры БД.
3. Ошибки требований и ошибки в проектных решениях остаются не обнаруженными до начала системной интеграции, системного тестирования или опытной эксплуатации.
4. Подсистемы трудно интегрировать из-за отсутствия общего архитектурного контекста проекта.
5. Отсутствует (или не соответствует коду) внутренняя документация ИС, т.е. описание ее основных архитектурных решений. Каждый разработчик знает «свою» часть, но всю ИС в целом не знает никто. Сопровождение и развитие такой системы чрезвычайно трудоемко и ведет к дальнейшему росту “хаотичности” структуры.
6. ИС работает ненадежно, ее производительность раздражает пользователей.
7. Этот список можно продолжить – дополните его самостоятельно.

Для успешного создания ИС необходима методология, опирающиеся на мощные, простые и удобные в использовании инструментальные CASE-средства. Осуществление сложных проектов в приемлемые сроки с высоким качеством невозможно без применения инженерных методов (иногда говорят - CASE-технологий).

Для грамотной постановки процесса разработки необходимо в первую очередь ответить на следующие вопросы:

- Как преодолеть разрыв между постановкой и реализацией задачи?
- Как обеспечить прозрачность проекта и его хода для заказчика и подрядчиков?
- Как автоматизировать достоверный учет и отчетность о ходе работ?
- Как сформировать и оценивать метрики проекта для постепенного роста качества планирования и точности оценки проектов?
- Как оценить качество результата?

Главная проблема работы в программистском проекте - это организация «правильных» коммуникаций как внутри команды

разработчиков, так и их коммуникаций с заинтересованными лицами. Поэтому кроме индивидуальных средств работы (компиляторы, отладчики…) у программистов должны быть средства, поддерживающие такие коммуникации.

В основе инженерных методов создания ИС положено «пошаговое» описание типичных работ. Методология определяет стадии и этапы проекта (phases), его контрольные точки (milestones), роли и состав работ. Для каждого этапа определяются:

- Последовательность работ и задач
- Рекомендации по их выполнению
- Состав получаемых документов и материалов (артефактов)
- Порядок контроля и проверки качества
- Инструментальные средства автоматизированной поддержки работ

Методология Rational Unified Process

Известны различные методологии: PJM (Oracle), MSF (Microsoft), RUP (IBM Rational.Software). Одной из ведущих методологией является Rational Unified Process (RUP)[13]. RUP (при успешном внедрении) обеспечивает прозрачность и управляемость процесса и позволяет создавать ИС в соответствии с требованиями заказчика на момент сдачи системы.

RUP – это оформленная в виде web-сайта электронная энциклопедия «правильной» (или образцовой – framework) методологии, которая описывает основные процессы создания ИС:

1. моделирование производственных процессов,
2. управление требованиями,
3. анализ и проектирование,
4. реализация,
5. тестирование,
6. развертывание,
7. управление изменениями и конфигурациями,
8. управление проектом,

---

[13] Крачтен Ф. Введение в Rational Unified Process.: Пер. с англ. – М.: Вильямс, 2002.
Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения.: Пер. с англ. – СПб: Питер, 2002.

9. поддержка среды разработки.

В соответствии с RUP все эти процессы, адаптируемые в организации для успешной реализации проекта разработки ИС, разделяются на ряд обязательных стадий (фаз), каждая из которых может состоять из несколько итераций.

Итеративная разработка – один из шести основополагающих принципов методологии Rational (Рисунок 3.1) по созданию программных систем. RUP дает разработчикам четкие инструкции по реализации этих принципов, среди которых также – эффективное управление требованиями, визуальное моделирование, использование компонентных архитектур, контроль качества на всем протяжении жизненного цикла приложения, контроль изменений, вносимых в ИС.

Рисунок 3.1 Лучшие принципов работы в проекте.

Особенности процесса RUP состоят в следующем:

1. Используется **итеративная разработка.** Функциональность ИС поэтапно «выращивается» до получения «зрелой» системы. Это позволяет улучшать понимание проблем и потребностей заказчика через создание последовательного набора прототипов ИС. В каждый следующий прототип добавляется новая функциональность, и исправляются текущие замечания заказчика. Возможно успешно управлять рисками проекта. Итеративный подход предполагает выполнение работ по контролю требований и управлению изменениями.

2. Используется **модель сценариев использования** (use-case model)[14] для представления функциональных требований к ИС. **Сценарий использования (**Use-case) – это ключевое понятие RUP, которое является единицей организации работ в проекте (т.е. это единица планирования, проектирования, отладки и тестирования ИС). Если в проекте функциональные требования к ИС сформулированы не в виде сценариев использования, то полноценно работать на основе RUP в проекте вы не сможете. Таким образом, модель сценариев использования является основой для эффективного планирования работ и управлению проектом, без его использования эффективность работы по RUP существенно снижается.

3. Основан на создании и итерационном развитии **визуальных моделей ИС**. Фокус внимания перемещен в проекте от создания бумажных документов, на разработку UML моделей. Наиболее важными артефактами являются модель сценариев использования, дизайн-модель, модель компонентов, модель тестирования (Рисунок 3.2). Необходимая рабочая документация генерируется автоматически по текущим моделям. Процесс анализа и проектирования (построения моделей) использует объектную технологию.

4. На первых этапах проекта основное внимание уделяется практической проверке **устойчивости выбранной архитектуры ИС**. Только после стабилизации интерфейсов основных подсистем начинается их программирование. Это позволяет сократить уровень перепроектирования и переделок кода, проводить планирование разработки ИС на основе выделения программных компонент[15].

---

[14] Модель вариантов использования или модель прецедентов

[15] Компоненты - это нетривиальные модули или подсистемы, которые реализуют заданный набор интерфейсов. Они могут быть повторно использованы в различных проектах в рамках известных компонентных технологий (CORBA, COM/DCOM, JavaBeans, J2EE)

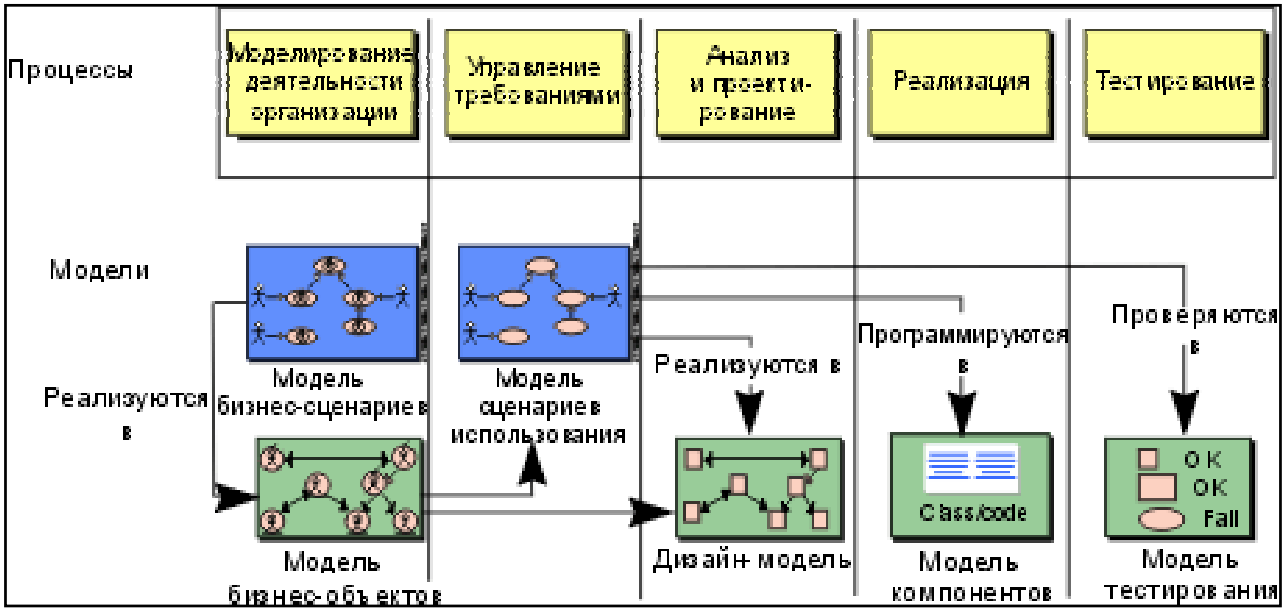


Рисунок 3.2. Модели и их использование в различных процессах RUP.

5. **Инструментальные CASE-средства** автоматизируют основные работы процесса, включая управление требованиями, изменениями и конфигурацией.

Рисунок 3.4 представляет структуру RUP в двух измерениях. По горизонтальной оси слева направо течет время проекта, по вертикали представлены основные и вспомогательные процессы. Динамические аспекты процесса выражаются в таких понятиях, как фазы, итерации и контрольные точки. Название фаз (или стадий проекта) при переводе можно дать в соответствии с этапами, определенными ГОСТом 34.[16] (

Таблица 3.1):

[16] Стандарт ГОСТ 34.602-89. «Информационная Технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы»

Таблица 3.1. Название фаз (стадий) RUP

Название фазы RUP	Варианты перевода	Название стадии по ГОСТ 34
Inception	Анализ, Предпроектное обследование, Начальная	Техническое задание
Elaboration	Проработка проекта, Развитие	Технический проект
Construction	Построение системы	Рабочий проект
Transition	Передача в эксплуатацию	Ввод в действие

Схема выполнения работ (Рисунок 3.4) иллюстрирует, как с течением времени смещаются акценты от фазы к фазе: величина «горба» условно иллюстрирует трудоемкость работ по соответствующему процессу. Рисунок 3.5. иллюстрирует основные артефакты Rational Unified Process и схему зависимостей между ними. Состав процессов описывается в терминах работ, задачи, артефактов и ролей (Рисунок 3.3).

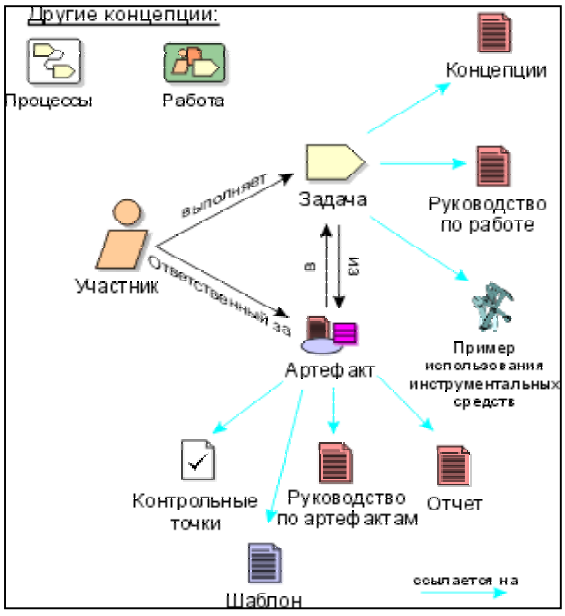


Рисунок 3.3. Основные элементы описания процессов - работа, задач, артефакты, роли

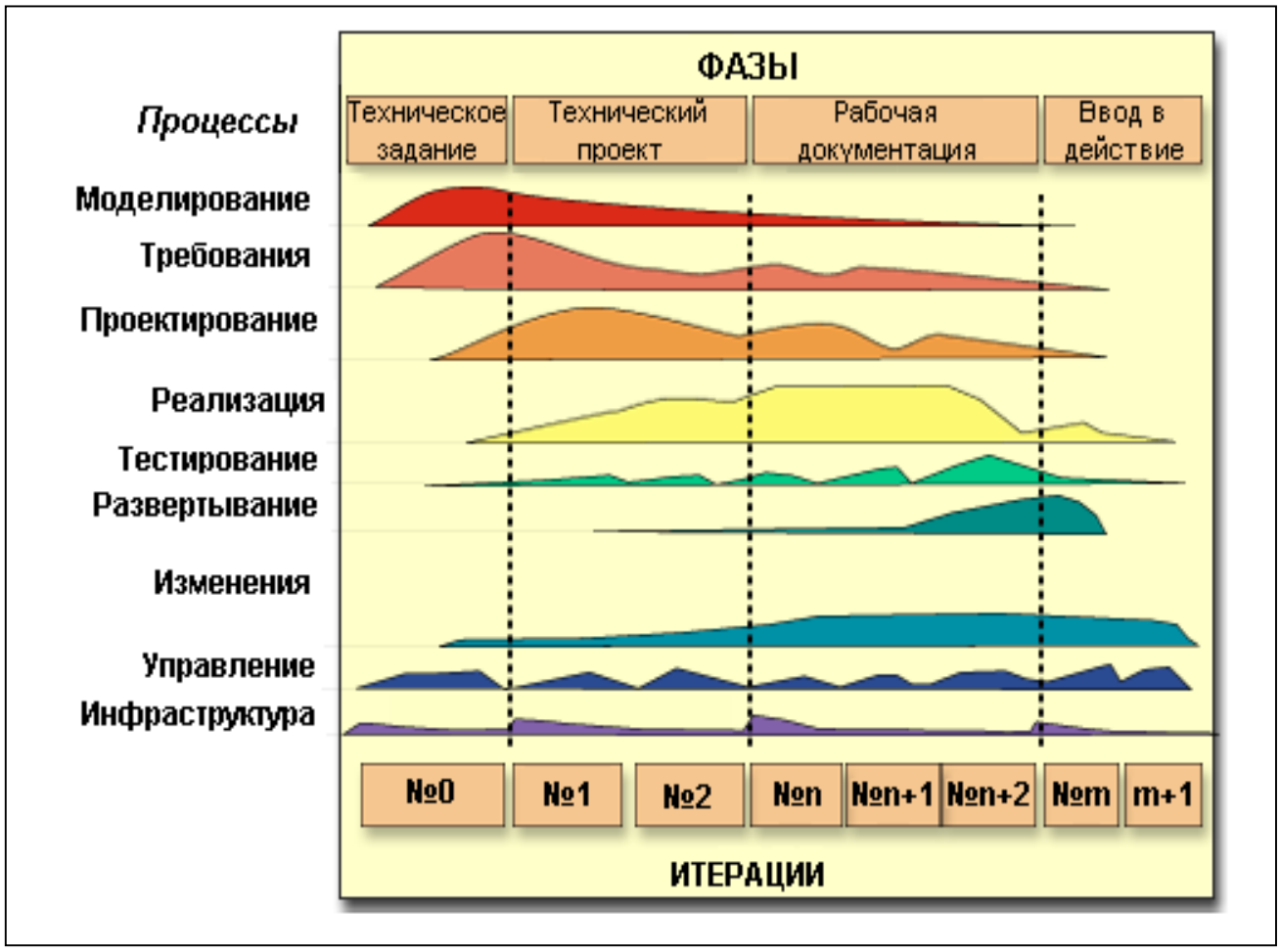


Рисунок 3.4. Архитектура RUP. Время проекта «течет» слева направо.

Основные процессы «дисциплины» создания ИС *(сверху вниз)**:***

- **Моделирование производственных процессов**
- **Управление требованиями**
- **Анализ и проектирование**
- **Реализация**
- **Тестирование**
- **Развертывание**
- **Управление изменениями и конфигурациями**
- **Управление проектом**
- **Поддержка среды разработки**

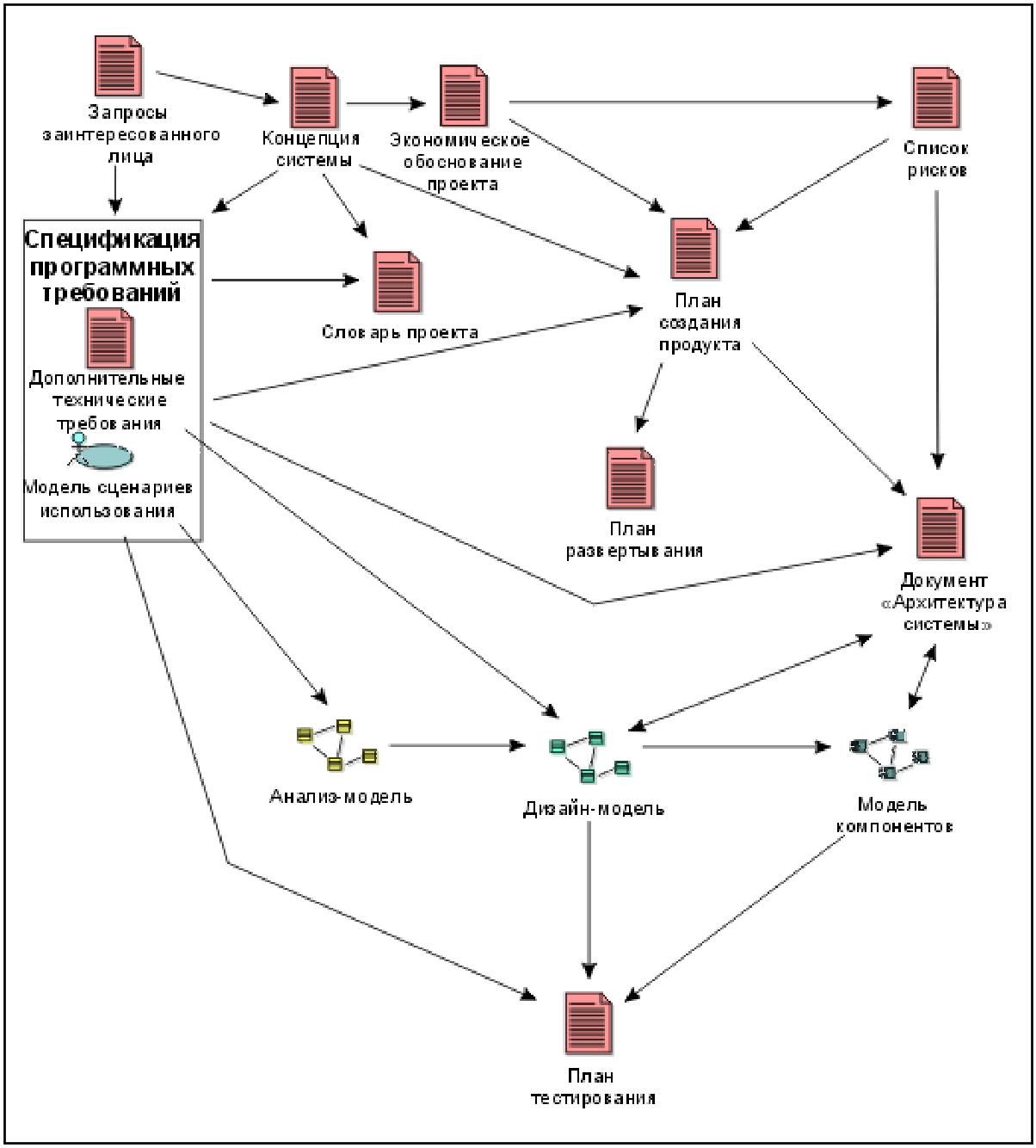


Рисунок 3.5. RUP: Основные артефакты процесса и схема зависимости между ними.

Стадии процесса RUP

Рассмотрим назначение и задачи четырех стадий (фаз) проекта.

Стадия "Техническое Задание"

Стадия **"Техническое Задание" (Inception)** проводится, чтобы обеспечить согласованность ожиданий заинтересованных лиц проекта на всех этапах жизненного цикла разработки ИС.

Цели стадии:

- Определить функции ИС и граничные условия, включая концепцию и критерии успешности проекта;
- Выявить основные сценарии использования системы;
- Определить вариант архитектуры ИС;
- Оценить стоимость и составить общий план работ;
- Оценить потенциальные риски проекта;
- Подготовить инфраструктуру работы по проекту.

Основные задачи стадии "Техническое Задание»

- Формулировка содержания проекта, включающая требования к параметрам, на основе которых будут сформулированы критерии приемки конечного продукта (в начале проекта нужно определить критерии его успешности).
- Подготовка технико-экономического обоснования проекта
- Выбор варианта архитектуры и оценка возможностей выполнения проекта
- Подготовка инфраструктуры - среды поддержки разработки проекта, выбор инструментальных средств.

Рисунок 3.6 иллюстрирует роли, выполняемые работы и создаваемые артефакты.

При успешном завершении стадии «Техническое Задание»:

- Определены функции ИС и ее масштаб, включая концепцию, критерии приемки системы, предполагаемые возможности ИС;
- Выявлены основные сценарии использования системы, которые послужат основой при выборе направления проектирования;

- Предложен вариант архитектуры, выбранный на основе ключевых сценариев использования системы;
- Проведена оценка полной стоимости работ и составлен план работ по проекту с более детальной проработкой для следующей стадии Технический проект (ТП);
- Проведена оценка потенциальных рисков;
- Подготовлен репозиторий для работы над проектом.

Создаются первые версии следующих основных материалов (артефактов RUP):

- Документ «Концепция системы» (**Vision**);
- Документ «Экономическое обоснование проекта» (**Business Case**);
- Документ «Словарь проекта» (**Glossary**);
- Документ «Список рисков» (**Risks**);
- План разработки продукта (**Software Development Plan**);
- План следующей итерации (**Iteration Plan**);
- Модель сценариев использования (первая версия - **Software Requirements**).

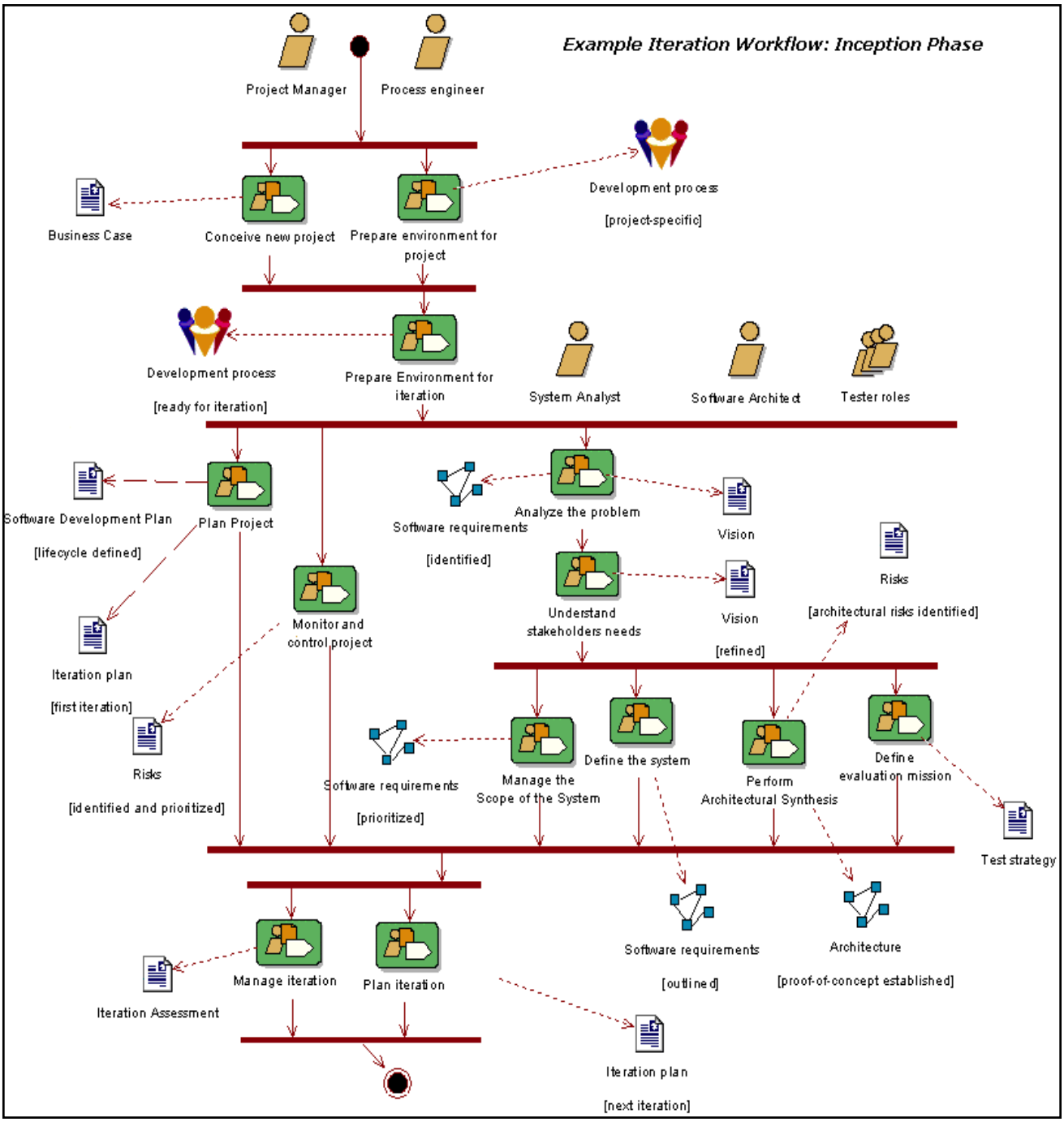


Рисунок 3.6. RUP: Роли, работы и артефакты итерации стадии «Техническое Задание» (Inception).

Стадия «Технический проект»

На стадии **«Технический проект»** (**Elaboration**) проводится проверка выбранной архитектуры системы, которая определяется основными требованиями. На стадии возможно создание несколько прототипов ИС с различными архитектурными подходами.

Цели стадии:

- Определить и стабилизировать базовую архитектуру, концепцию ИС и ее основные требования – как функциональные, так и нефункциональные;
- Обеспечить поддержку среды разработки.

Основные задачи стадии:

- Определение, обоснование и формирование базовой версии (baseline) архитектуры ИС;
- Детализация концепции системы (Vision), выявление наиболее критичных факторов, влияющих на архитектурные и проектные решения;
- Создание детального плана итерации для стадии "Рабочий проект";
- Уточнение деталей процесса разработки.

Рисунок 3.7 иллюстрирует роли, выполняемые работы и создаваемые артефакты. При успешном завершении стадии:

- Обоснована архитектура системы, требования и планы;
- Снижены риски так, что возможно прогнозировать затраты и построить план-график работ;
- Определена и утверждена базовая версия архитектуры, рассмотрены ключевые сценарии использования (use case);
- Получены прототипы программных компонент.

Созданы следующие основные артефакты:

- Релиз системы (**Build**);
- Документ "Архитектура системы" (**Architecture**)
- Дизайн-модель, включая модель данных (**Design Model**);
- План тестирования;
- План следующей итерации (**Iteration Plan**);
- Оценка итерации (**Iteration Assessment**).

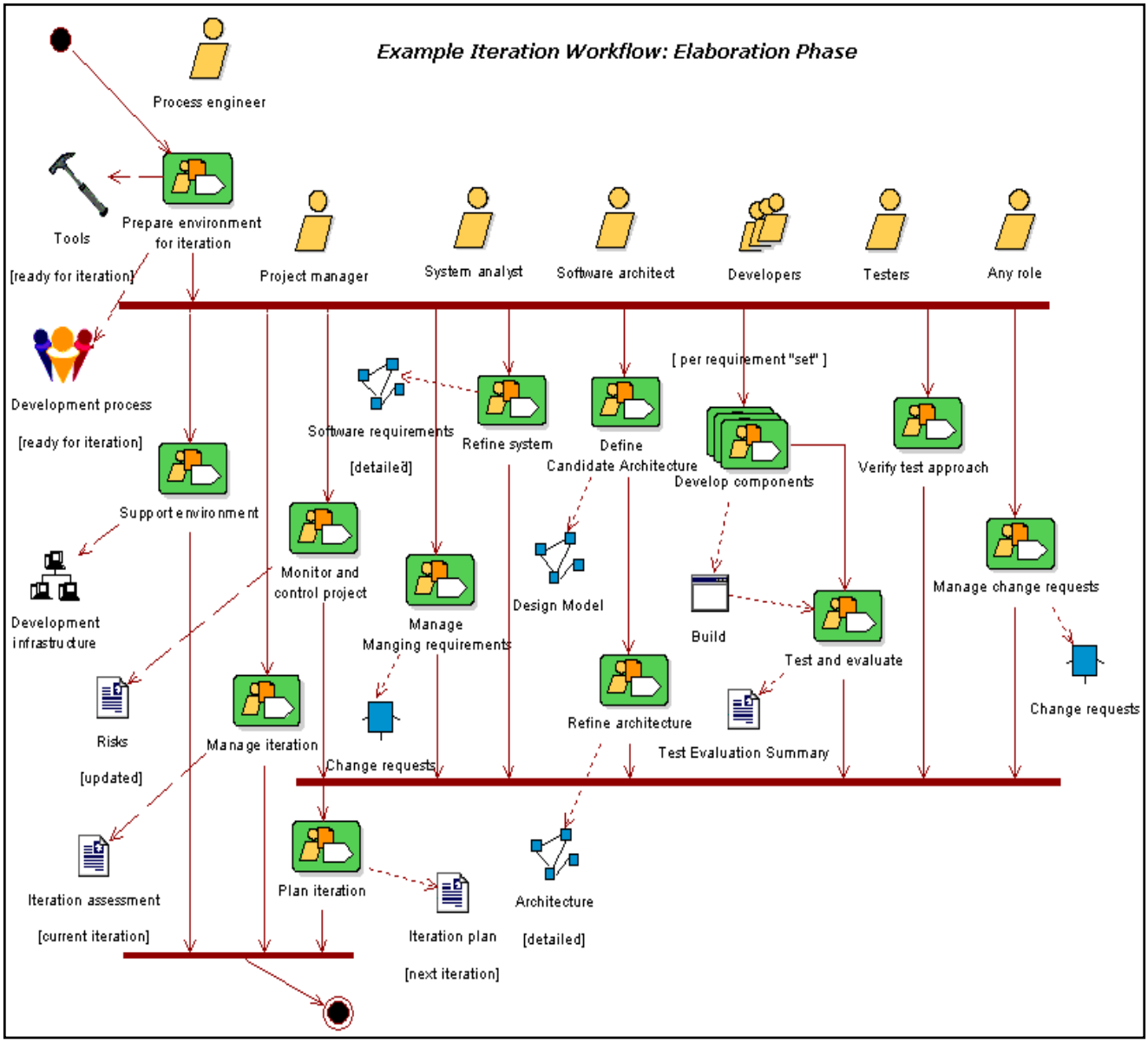


Рисунок 3.7. RUP: Роли, работы и артефакты итерации стадии «Технический Проект» (Elaboration).

Стадия «Рабочий проект»

На стадии **«Рабочий проект» (Construction)** проводится построение информационной системы на основе утвержденной архитектуры. Стадия представляет собой процесс «изготовления» (программирования), где основное внимание уделяется управлению ресурсами для оптимизации стоимости, сроков выполнения работ и качества.

Цели стадии:

- Минимизировать стоимости разработки за счет оптимизации ресурсов и исключения переделок;
- Завершить анализ, проектирование и компонентное тестирование;
- Провести системное тестирование ИС;
- Разработать ИС, готовую для развертывания у пользователей;

Основные задачи стадии:

- Управление ресурсами и оптимизация процесса разработки ИС;
- Завершение разработки и тестирования компонент, удовлетворяющей заданным критериям качества;
- Оценка бета-релиза ИС в соответствии с критериями приемки, изложенными в концепции.

Рисунок 3.8 иллюстрирует роли, выполняемые работы и создаваемые артефакты. При успешном завершении стадии «Рабочий Проект»:

- Проведено бета-тестирование ИС;
- Завершены процессы анализа, проектирования и тестирования;
- Разработан дистрибутив ИС, готовый для передачи пользователям;

Созданы следующие основные артефакты:

- Бета версия Системы (**Product**);
- Документация пользователя (**End-User Support Material**);
- План следующей итерации (**Iteration Plan**);
- Оценка итерации (**Iteration Assessment**);
- Документ «Сводные результаты тестирования» (**Test Evaluation Summary**);
- Запросы на изменение (**Change Requests**).

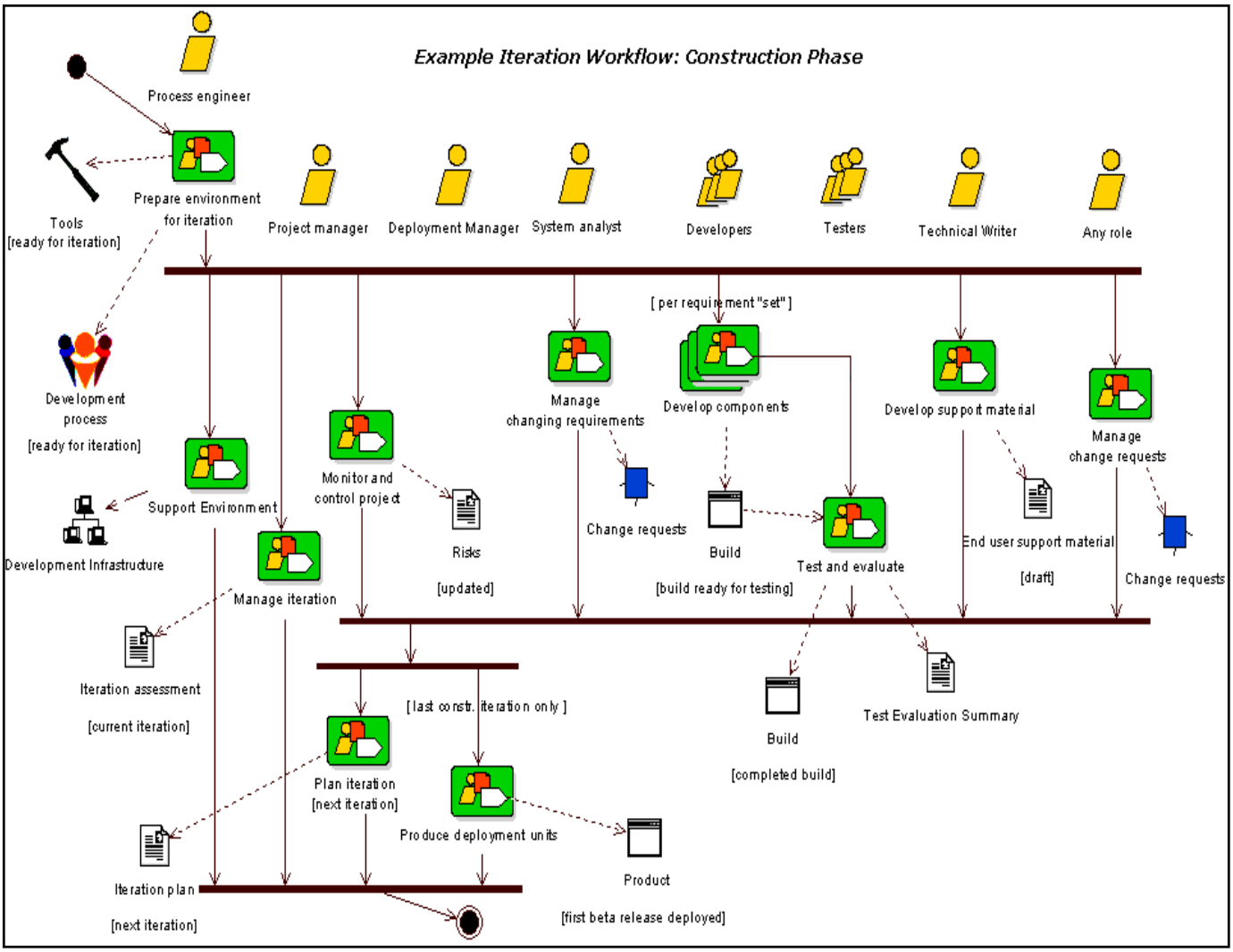


Рисунок 3.8. RUP: Роли, работы и артефакты итерации стадии «Рабочий Проект» (Construction).

Стадия «Ввод в действие»

На стадии **«Ввод в действие» (Transition)** основное внимание уделяется работам по «продвижению» ИС к Заказчику (к конечным пользователям). Стадия может занять несколько итераций. Поддерживается обратная связь с пользователями и заинтересованными лицами. В конце стадии все поставленные цели должны быть достигнуты.

Цели стадии:
Провести тестирование ИС на стороне Заказчика;
Конвертировать операционные данные Заказчика в БД новой ИС;
Обучить пользователей;
Подготовить дистрибутив, при необходимости провести тиражирование продукта.

Основные задачи стадии:
Выполнение план развертывания ИС;
Детализация документации для конечных пользователей;
Системное тестирование ИС на стороне заказчика;

- Проведение приемо-сдаточных испытаний ИС.

Рисунок 3.9 иллюстрирует роли, выполняемые работы и создаваемые артефакты.

При успешном завершении стадии «Ввод в действие»:

- Достигнуты цели проекта;
- Завершен процесс поставки Заказчику системы и документации.

Созданы следующие основные артефакты:

- Выходной релиз системы (**Product**);
- Инсталляционные артефакты;
- Документация пользователя (**End-User Support Material**);
- Оценка результатов проекта (**Status Assessment**)

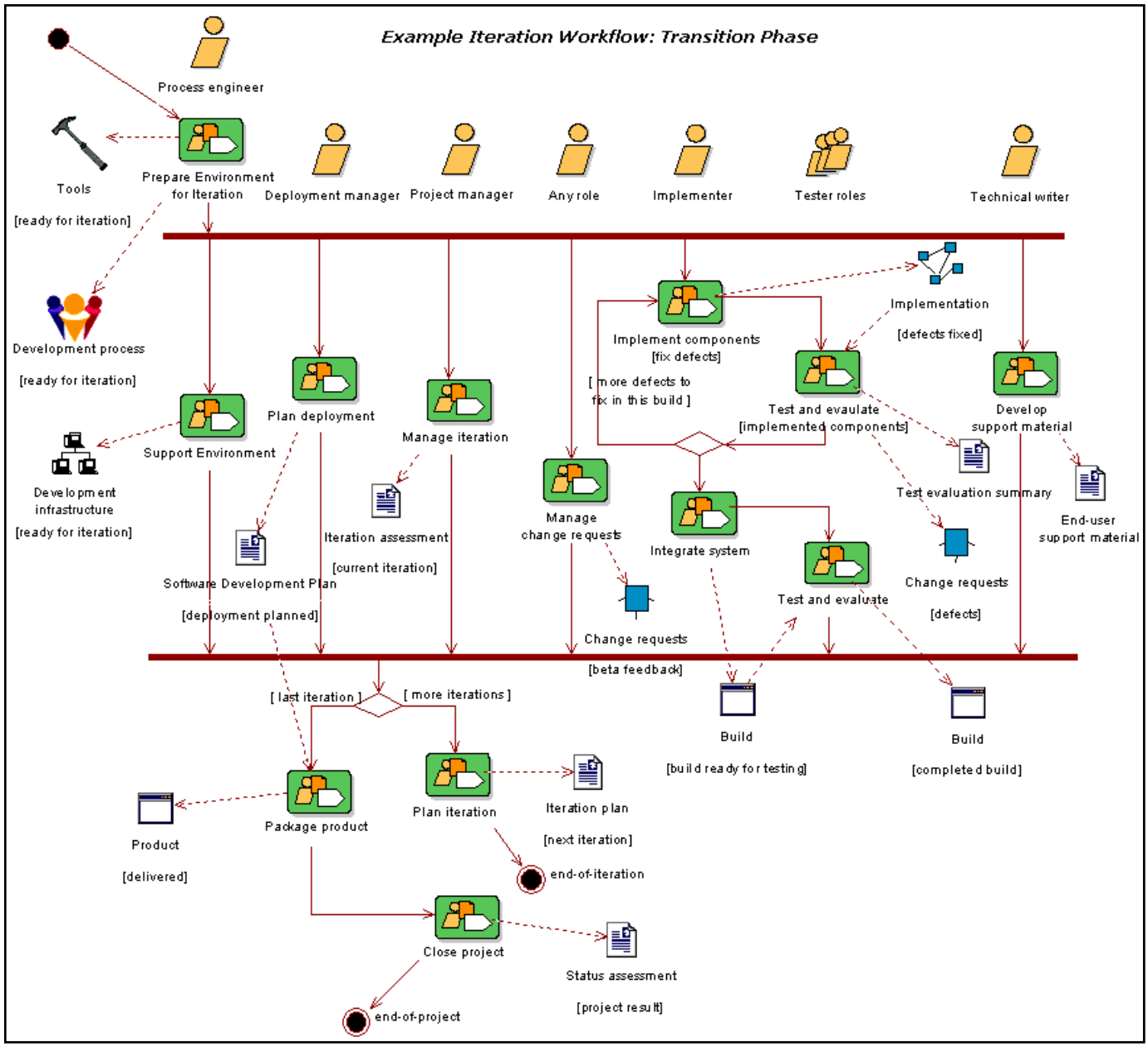


Рисунок 3.9. RUP: Роли, работы и артефакты итерации стадии «Ввод в действие» (Transition).

Внедрение RUP

При внедрении RUP ее процессы должны быть адаптированы и к особенностям самой организации-разработчика, и к специфике выполняемых ею конкретных проектов. И первое, что нужно любой организации, — это правильная постановка работы с требованиями. С требований нужно начинать постановку методологии потому, что ошибки в них самые распространенные и наиболее дорогостоящие. Далее необходимо ставить управление изменениями, управление проектом. Так мы можем достичь второго уровня CMM (Capability Maturity Model for Software[17]), показывающего уровень технологической зрелости организации, при которой базовые процессы разработки в полной мере планируются и контролируются[18].

Для настройки и публикации описания методологии (в виде Web-сайта) на основе RUP, можно использовать Rational Process Workbench (RPW - Рисунок 3.10), который предназначен для тех случаев, когда в RUP необходимо внести значительные изменения.

RPW является первым инструментом визуального моделирования процессов, использующим UML. Визуальное моделирование процессов повышает уровень абстракции, облегчая понимание и изменение процессов. Взаимосвязанность процессов обеспечена метамоделью RUP. RPW поддерживает три основные задачи моделирования процессов:

- определение процесса
- описание процесса
- представление процесса

В качестве основы для определения процесса берется модель RUP. Изменение и расширение базовой модели проводится с помощью Rational Rose. Визуализация связей между элементами процесса показывает, например, какие артефакты задействованы в процессе и какие роли отвечают за их создание. Библиотека элементов процесса содержит текстовую информацию о каждом элементе в модели процесса. Библиотека содержит все текстовые страницы RUP, а RPW — необходимые шаблоны для создания новых страниц описания.

[17] См. www.sei.cmu.edu/cmm/cmm.html

[18] Для оценки уровня зрелости по CMM используется международный стандарт ISO-15504. См. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504 – CMM) /Пер. с англ. – М.: Книга и бизнес, 2001.

На последнем этапе — этапе представления процесса — RPW генерирует описание процессов, включающее текст и графику в виде веб-сайта, соединяя модели процессов и библиотеку описаний в единое целое.

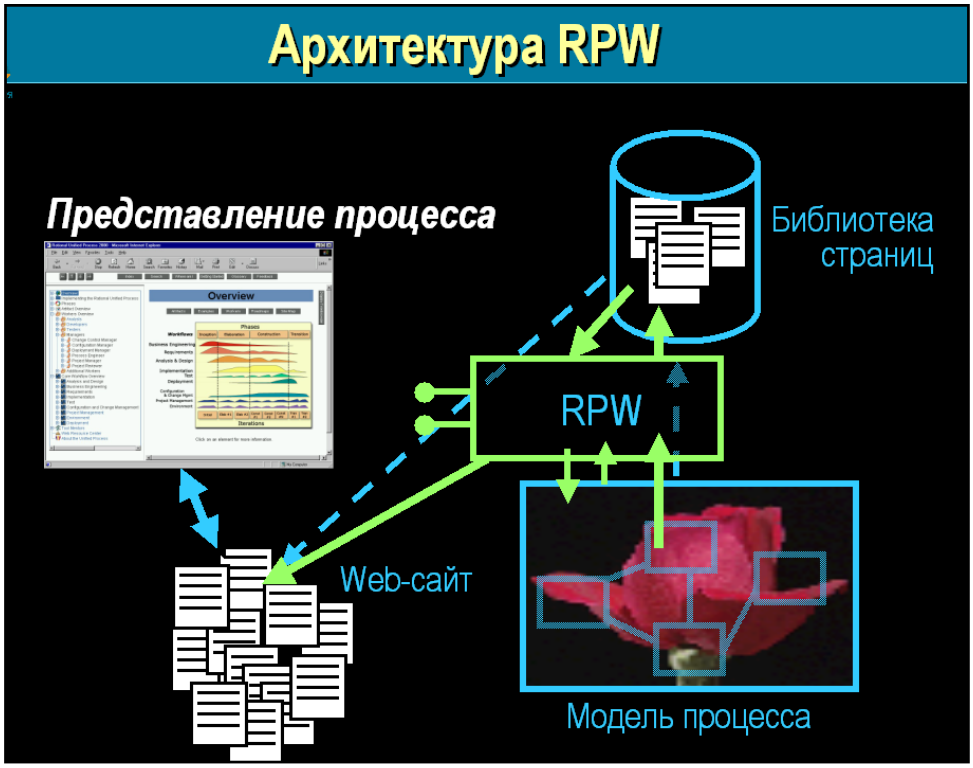


Рисунок 3.10. Архитектура Rational Process Workbench - инструмента визуального моделирования процессов, использующих UML

В RUP описана методика внедрения процессов разработки в организации (Рисунок 3.11). Кроме того, RUP интересен еще и тем, что он инвариантен к тому, какой инструмент будет использован для управления требованиями, для управления конфигурацией, визуального моделирования. Например, для управления требованиями можно (но не обязательно) использовать RequisitePro, входящий в Rational Suite.

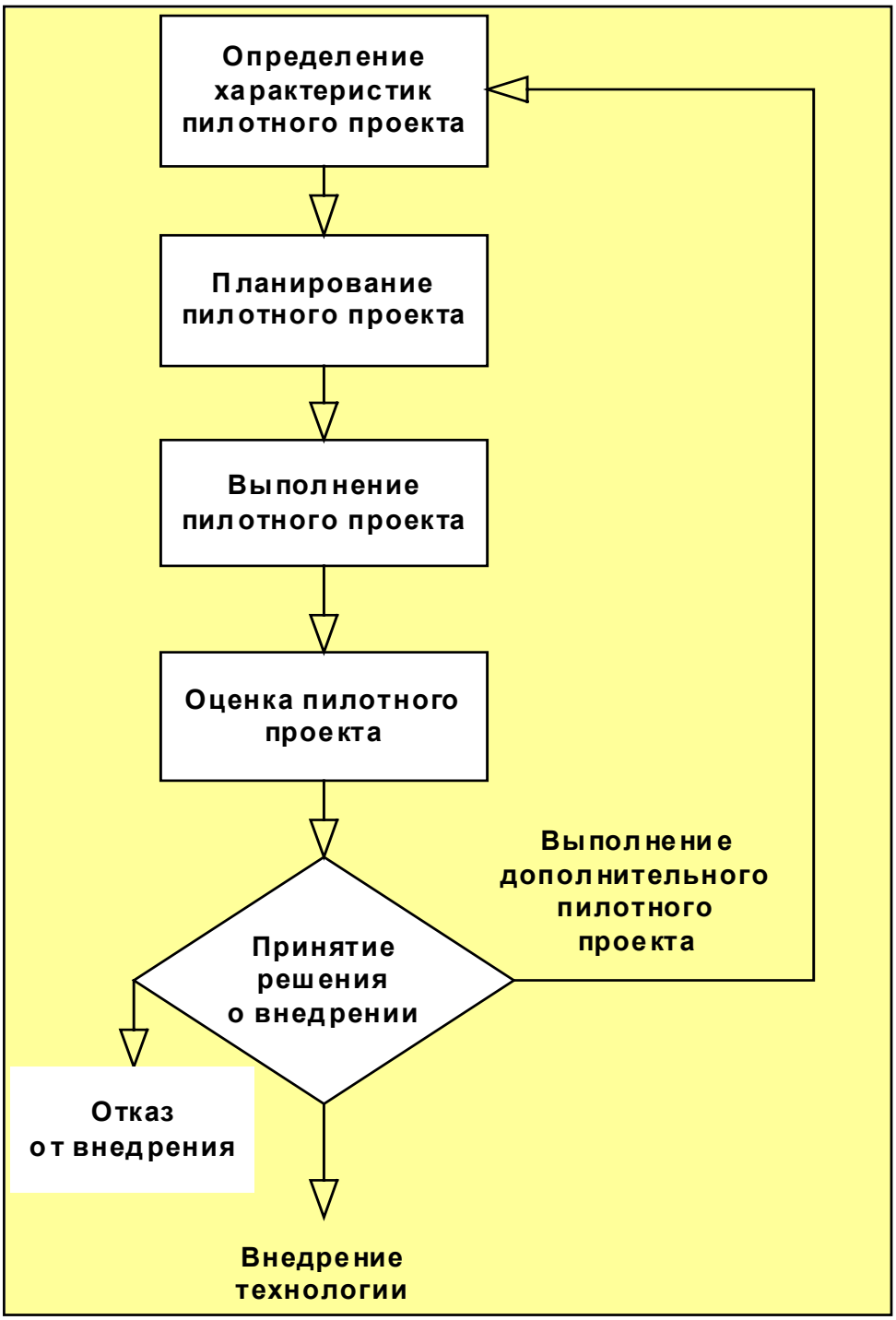


Рисунок 3.11. Этапы пилотного проекта внедрения процессов RUP.

Жизненный цикл ИС - ГОСТ 12207 и RUP

Рассмотрим соответствие российского стандарта ГОСТ 12207, «Информационная технология. Процессы Жизненного Цикла Программных Средств» и методологии Rational Unified Process. Стандарт ГОСТ 12207 введен в действие в России летом 2000 года. Он определяет работы, которые могут выполняться в жизненном цикле программных средств. Эти работы распределены по пяти основным, восьми вспомогательным и четырем организационным процессам. Каждый процесс жизненного цикла разделен на набор работ; каждая работа разделена на набор задач.

Основные процессы жизненного цикла ГОСТ 12207

- Процесс заказа, (выполняется на стороне **заказчика** ПС)
- Процесс поставки, (на стороне **поставщика** ПС)
- Процесс разработки (на стороне **разработчика** ПС)
- Процесс эксплуатации (на стороне **оператора** ПС)
- Процесс сопровождения (выполняется **персоналом сопровождения**)

Вспомогательные процессы жизненного цикла ГОСТ 12207

- Процесс документирования
- Процесс управления конфигурацией
- Процесс обеспечения качества
- Процесс верификации
- Процесс аттестации
- Процесс совместного анализа
- Процесс аудита
- Процесс решения проблем

Организационные процессы жизненного цикла ГОСТ 12207

- Процесс управления
- Процесс создания инфраструктуры
- Процесс усовершенствования
- Процесс обучения

Особенность стандарта 12207 состоит в том, что он имеет общий (т.е. так называемый «рамочный») характер. Он не определяет состав выходных документов каждой задачи, не определяет роли, ответственные за создание документов в процессе и т.п. Практическое использование стандарта 12207 из-за этого сильно затруднено, поскольку требует определения конкретной технологии, с одной стороны, удовлетворяющей стандарту.

Для простоты сравнения объединим участвующие стороны по ГОСТ 12207, и рассмотрим двух основных участника – Заказчика и Разработчика. «Заказчик» объединяет **заказчика** ПС, **оператора** ПС, **персонал сопровождения**. «Разработчик» объединяет **поставщика и разработчика** ПС. В этой терминологии RUP описывает работы, выполняемые на стороне разработчика. Оказывается, что реализация большинства процессов стандарта 12207, *выполняемых на стороне разработчика*, детально описана в RUP (Таблица 3.2).

Таблица 3.2. Соответствия задач *процесса разработки* 12207 и процессов RUP

Задачи процесса разработки 12207	Процессы RUP
1 Подготовка процесса разработки	Управление проектом
2 Анализ требований к системе	Управление требованиями (к системе)
3 Проектирование системной архитектуры	Анализ и проектирование (системы)
4 Анализ требований к ПС	Управление требованиями (ПС)
5 Проектирование программной архитектуры	Анализ и проектирование (ПС)
6 Техническое проектирование ПС	Анализ и проектирование (ПС)
7 Программирование и тестирование ПС	Реализация ПС
8 Сборка программных средств	Управление конфигурациями ПС
9 Квалификационные испытания ПС	Тестирование (ПС)
10 Сборка системы	Развертывание
11 Квалификационные испытания системы	Развертывание
12 Ввод в действие ПС	Развертывание
13 Обеспечение приемки ПС	Управление проектом

Таблица 3.2 иллюстрирует, что основной процесс разработки ИС представлен в стандарте 12207 довольно «крупными мазками» - задачи этого процесса покрываются восемью процессами RUP, при этом три процесса RUP (Управление требованиями, Анализ и проектирование, Реализация) полностью покрыты процессом разработки 12207.

Рассмотрим соответствие RUP **вспомогательным процессам** жизненного цикла по 12207.

Процесс документирования (12207). «Встроен» во все задачи всех ролей. Для основных документов определены шаблоны документов. Документирование в RUP поддерживается средством автоматизации Rational SoDA – определены схемы генерации основных отчетов и составных документов.

Процесс управления конфигурацией(12207) - «…применение административных и технических процедур на всем протяжении жизненного цикла программных средств для: обозначения, определения и установления состояния (базовой линии) программных объектов в системе; управления изменениями и выпуском объектов; описания и сообщения о состояниях объектов и заявок на внесение изменений в них; обеспечения полноты, совместимости и правильности объектов; управления хранением, обращением и поставкой объектов». Явно определен в RUP как самостоятельный процесс «Конфигурационное управление и управление изменениями» с соответствующим описанием ролей, задач, входных и выходных материалов (артефактов).

Процесс обеспечения качества(12207). Поддерживается в RUP на двух уровнях – на уровне качества материалов и на уровне «качества процесса». Качество создаваемых материалов для каждого процесса RUP обеспечивается введение ролей «рецензентов».

Качество процесса поддерживается в процессе управления проектом и может контролироваться за счет введения и использования метрик проекта.

Процесс верификации(12207) определяет, «что программные продукты функционируют в полном соответствии с требованиями или условиями, реализованными в предшествующих работах»**.** В RUP полностью покрывается процессом «Тестирование» (включая функциональное и нагрузочное тестирования»).

Процесс аттестации(12207) определяет «полноту соответствия установленных требований, созданной системы или программного продукта их функциональному назначению». В RUP покрывается задачами процесса «Развертывание».

Процесс совместного анализа(12207), т.е. «оценки состояний и, при необходимости, результатов работ (продуктов) по проекту». В RUP покрывается задачами процесса «Управление проектом».

Процесс аудита(12207), т.е. «определения соответствия требованиям, планам к условиям договора». В RUP покрывается задачами процесса «Управления проектом»

Процесс решения проблем(12207) т.е. «анализа и решения проблем (включая обнаруженные несоответствия), независимо от их происхождения или источника, которые обнаружены в ходе выполнения разработки, эксплуатации, сопровождения или других процессов». В RUP покрывается процессами «Конфигурационное управление и управление изменениями» и процессом «Управление проектом».

И, наконец, рассмотрим соответствие RUP организационным процессам жизненного цикла по 12207.

Процесс управления(12207) «состоит из общих работ и задач, которые могут быть использованы любой стороной, управляющей соответствующим процессом(ами). Администратор отвечает за управление продуктом, проектом, работами и задачами соответствующего процесса(ов), таких как: заказ, поставка, разработка, эксплуатация, сопровождение или вспомогательные процессы». В RUP покрывается процессом «Управление проектом» (полностью).

Процесс создания инфраструктуры(12207) т.е. «установления и обеспечения (сопровождения) инфраструктуры, необходимой для любого другого процесса. Инфраструктура может содержать технические и программные средства, инструментальные средства, методики, стандарты и условия для разработки, эксплуатации или сопровождения». В RUP покрывается процессом «Поддержка среды разработки» (полностью).

Процесс усовершенствования(12207) т.е. «установления, оценки, измерения, контроля и улучшения любого процесса жизненного цикла программных средств». Явно в RUP не описан.

Процесс обучения(12207) т.е. «обеспечения первоначального и продолженного обучения персонала». Явно в RUP не описан.

Подготовка учебных материалов в RUP предусмотрена как задача процесса «Развертывание».

Таким образом, RUP можно рассматривать как «детализированную» реализации задач, определяемых стандартом 12207 по основным процессам, выполняемых на стороне разработчика и поставщика, а также практически всех задач вспомогательных и организационных процессов 12207. Конкретное определение и реализация процесса создания ПС на основе RUP происходит, как правило, в ходе выполнения соответствующего пилотного проекта, в рамках которого одновременно решаются задачи настройки средств автоматизации процесса и настройки репозитория проекта.

Вопросы

1. Что определяет ГОСТ 12207? Нужно ли его использовать при разработке базы данных?

2. Опишите основные фазы (стадии) выполнения проекта по RUP.

3. Чем первая фаза проекта (Inception или «Техническое Задание») отличается от второй фазы (Elaboration или «Технический Проект»)

4. Что такое артефакты? Для чего они используются? Приведите примеры.

5. Перечислите 6 лучших принципов (Best Practice) работы в проекте создания ИС. Дайте краткую характеристику каждому принципу.

6. Опишите особенности внедрения методологии в проект на примере RUP.

7. Что такое риски проекта? Зачем нужна их идентификация?

4. Визуальное моделирование

Для моделирования данных и структуры ИС используются различные подходы и визуальные нотации. Мы будем рассматривать визуальные модели на UML[19]. Его важной особенностью является **единство подхода,** как при моделировании данных, так и при визуальном проектировании архитектуры ИС.

Визуальное моделирование позволяет решать 4 важные задачи[20]:

2. Визуализировать систему в ее текущем состоянии («as is») и требуемом в будущем состоянии («to be»);
3. Определить структурные аспекты системы («статику») и ее поведение во времени («динамику»);
4. Накапливать образцы (шаблоны) для конструирования аналогичных систем;
5. Документировать в модели принимаемые решения.

Использование визуального моделирования важно для сложных систем – иначе разработчики (и заказчики) не смогут ее воспринять как единое целое.

Опыт использования моделирования позволяет сформулировать четыре основных принципа:

1. Выбор типа модели определяет **подходы к решению** проблемы и влияет на то, как будет выглядеть решение.

2. Каждая модель должна иметь **разные уровни детализации**, каждый из которых представляет различный уровень абстракций, как проблемы, так и предлагаемых решений.

3. Лучшие модели те, что **ближе к реальности**. Этому принципу хорошо удовлетворяет объектно-ориентированный подход к проектированию и построению системы – объектам прикладной области соответствуют программные объекты системы.

---

[19] См. Фаулер М., Скотт К. **UML в кратком изложении. Применение стандартного языка объектного моделирования**: Пер. с англ. – М.: Мир, 1999. и Фаулер М., Скотт К. **UML. Основы. Краткое руководство по унифицированному языку моделирования**: 2-е издание / Пер. с англ. – СПб: Символ-Плюс, 2002.

[20] Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя.: Пер. с англ. – М.: ДМК, 2000.

4. Одной модели недостаточно. Наилучший подход при разработке нетривиальной системы – использовать **несколько моделей**, почти независимых друг от друга.

UML – стандартизированный язык моделирования

Unified Modeling Language (UML) — это стандартная нотация визуального моделирования программных систем, принятая консорциумом Object Managing Group (OMG) осенью 1997 г. На сегодняшний день она поддерживается многими объектно-ориентированными CASE-продуктами. При этом была стандартизирована (унифицирована) только нотация, т.е. семантика самого языка - невозможно унифицировать технологию разработки программных систем. Каждая компания может использовать собственную методологию, основанную на UML.

UML – это унифицированное визуальное средство (язык) для:

- определения,
- представления,
- проектирования и
- документирования

программных систем, организационно-экономических систем, технических систем и других систем различной природы

Языку UML уделяется в настоящее время большое внимание. Дело в том, что в последнее время наблюдается общее повышение интереса ко всем аспектам, связанным с разработкой сложных программных приложений. Для многих компаний корпоративное программное обеспечение и базы данных (БД) представляют стратегическую ценность. Существует высокая заинтересованность в разработке и проверке методов и подходов, позволяющих автоматизировать процесс создания сложных ИС. Известно, что систематическое использование таких методов дает возможность значительно улучшить качество, сократить стоимость и время поставки ИС. Сегодня к этим методам можно отнести:

- компонентную технологию разработки ИС;
- визуальное программирование;
- использование образцов (patterns) при проектировании ИС;
- визуальное представление проекта (визуальное моделирование, CASE-средства).

Визуальные модели широко используются в существующих технологиях управления проектированием систем, сложность, масштабы и функциональность которых постоянно возрастают. В практике эксплуатации ИС все время приходится решать такие задачи, как физическое перераспределение вычислений и данных, обеспечение параллелизма вычислений, безопасности доступа к ИС и устойчивости к сбоям, репликация БД, оптимизация балансировки нагрузки ИС и т. п.

Построить модель корпоративной ИС до ее программной разработки или до начала проведения архитектурной реконструкции столь же важно, как подготовить проектные чертежи, прежде чем приступить к строительству большого здания. Хорошая модель ИС позволяет наладить взаимодействие между заказчиками, пользователями и командой разработчиков, обеспечивает ясность представления архитектурных решений, и позволяет "охватить" систему во всей ее полноте. Разрабатываемые системы становятся все более сложными, соответственно возрастает и актуальность использования "хороших" методов их моделирования. Язык моделирования, как правило, включает в себя:

- элементы модели — фундаментальные концепции моделирования и их семантику;
- нотацию — визуальное представление элементов моделирования;
- принципы использования — методологию, т.е. правила применения элементов в рамках построения тех или иных типов моделей ИС.

Создавая визуальные модели, проектная команда может разрешить ряд типичных проблем. Во-первых, и это главное, благодаря технологии визуального моделирования становится возможным работать со **сложными** и очень сложными системами и проектами. Не имеет значения, преобладает ли в проекте "техническая сложность" (статическая) или "сложность управления" (динамическая). Важно, что общая сложность программных систем возрастает по мере создания новых версий. И в какой-то момент наступает "эффект критической массы" — дальнейшее развитие (сопровождение) ИС становится невозможным, поскольку уже никто не представляет в целом, что и почему происходит. В результате теряется управляемость проектом. Внешней причиной или толчком возникновения этого неприятного эффекта может послужить, например, увольнение ведущего

программиста или системного аналитика.

Во-вторых, визуальные модели позволяют содержательно организовать общение между заказчиками и разработчиками. Шутка о том, что "заказчик чего-то хочет, но точно не знает, чего именно", часто и с завидным постоянством, оказывается былью. А если на начальном этапе работы над проектом ИС заказчик думает, что точно знает, чего хочет, то, как правило (и об этом свидетельствует богатый опыт), его требования изменяются в ходе выполнения проекта. С одной стороны, аппетит приходит во время еды, а с другой — высокая динамика бизнеса объективно заставляет менять требования к разрабатываемой (или поддерживаемой) ИС.

Само по себе визуальное моделирование не является "серебряной пулей", способной раз и навсегда решить все проблемы, однако его использование существенно облегчает путь к достижению таких целей, как повышение качества программного продукта, сокращение стоимости проекта, поставка системы в запланированные сроки.

Как «разбивать» сложную систему на части?

При проектировании сложной ИС ее разбивают на части (подсистемы), каждая из которых затем рассматривается отдельно. Возможны два различных способа такого разбиения: структурное (или функциональное) разбиение и объектная (компонентная) декомпозиция.

Функциональная, или алгоритмическая декомпозиция: структура системы описывается в терминах иерархии ее функций и передачи информации между отдельными функциональными элементами, а поведение системы описывается в терминах последовательности выполнения процедур, реализующих алгоритмы функций

Объектная декомпозиция: структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами

Суть функционального разбиения хорошо отражена в известной формуле: программа = данные + алгоритмы.

При функциональной декомпозиции программной системы ее структура может быть описана блок-схемами, узлы которых представляют собой "обрабатывающие центры" (функции), а связи между узлами описывают движение данных.

Объектное разбиение, (иногда называемое компонентным, — что нашло отражение в специальном термине "разработка, основанная на компонентах" Component Based Development — CBD), — опирается на иной принцип декомпозиции — система разбивается на "активные сущности" (объекты или компоненты), которые взаимодействуют друг с другом, обмениваясь сообщениями и находясь в отношении «микро-клиент—микро-сервер». Сообщения, которые может принимать объект, определены в его интерфейсе. В этом смысле посылка сообщения "объекту-серверу" эквивалентна вызову соответствующего метода объекта.

Если при проектировании информационная система разбивается на объекты (компоненты), то ее визуальное моделирование может осуществляться с помощью UML. Если используется функциональная декомпозиция ИС, то можно использовать и UML, но лучше применять другие (специализированные структурные) нотации и методологии. Нужно ли программировать в объектах (компонентах) — отдельная тема для обсуждения. Споры по этому поводу относятся к разряду "религиозных войн"[21]. Есть убежденные сторонники в обоих лагерях. Уместно заметить, что все современные средства программирования используют библиотеки компонентов, позволяющих повторно применять отлаженный программный код, что значительно облегчает сборку программных приложений. Такое "сборочное программирование" стало возможным за счет объектного подхода и привело к изменению квалификационной оценки программистов за рубежом. Теперь считается, что программист — это тот, кто умеет программировать в компонентах, т. е. это «сборщик из компонент», не писатель программного кода, как еще иногда принято считать.

UML предоставляет выразительные средства для создания визуальных моделей, которые одинаково понимаются всеми участниками проекта, и являются инструментом коммуникации в рамках проекта. UML не зависит от объектно-ориентированных (ОО) языков программирования (он может отображаться в любой из них) и не

[21] Это нашло отражение во фразе: Террорист отличается от методологиста тем, что с террористом можно (иногда) договориться.

зависит от используемой методологии разработки проекта. UML является открытым и обладает средствами расширения базового ядра –

Рисунок 4.1 иллюстрирует пример введения новых модельных элементов, расширяющих нотация (с использованием стереотипов) для описания бизнес процессов. На UML можно содержательно описывать классы, объекты и компоненты, соответствующие различным предметным областям.

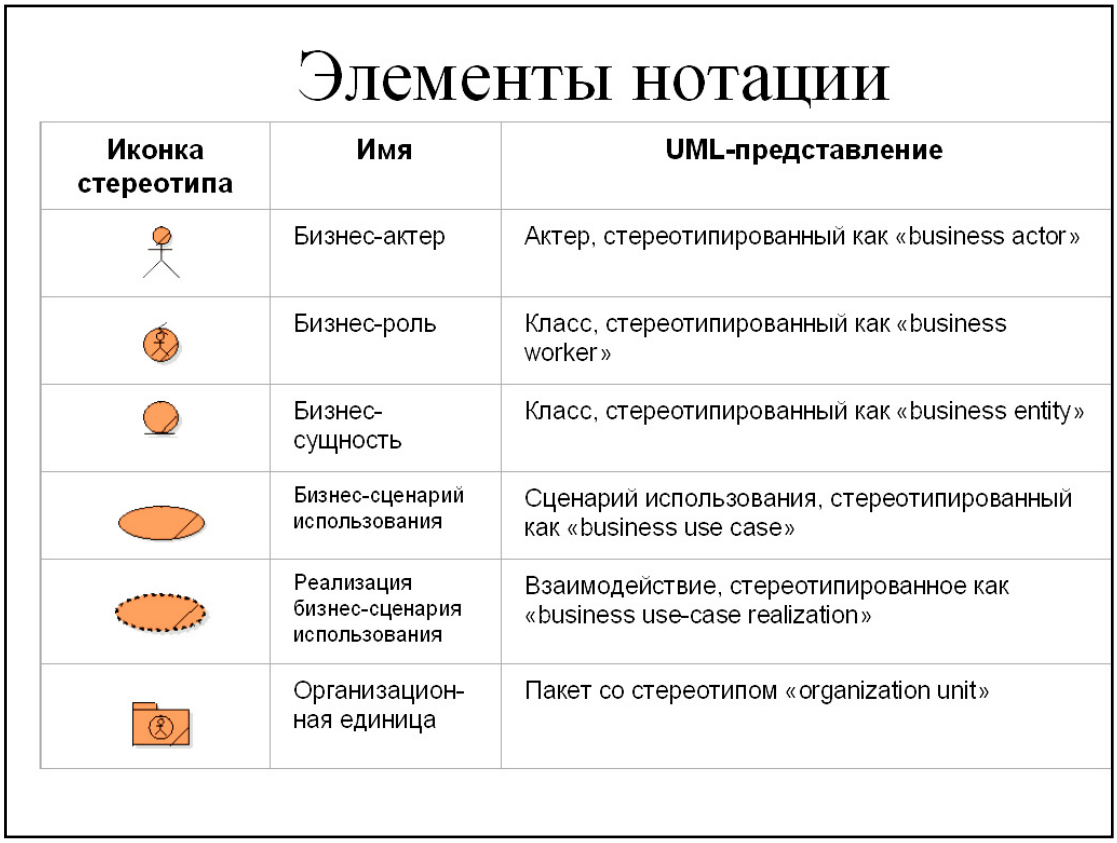


Рисунок 4.1. Элементы нотации UML, используемые при моделировании бизнес процессов

Что можно «рисовать» на UML?

При визуальном моделировании на UML используются восемь видов диаграмм, каждая из которых может содержать элементы определенного типа:

- Диаграммы сценариев использования (**Usecase diagrams**)
- Диаграммы классов (**Class diagrams**)
- Диаграммы взаимодействия:
  - Диаграммы последовательности (**Sequence diagrams**)
  - Диаграммы кооперации (**Collaboration diagrams**)
- Диаграммы состояний (**Statechart diagrams**)
- Диаграммы деятельности (**Activity diagrams**)
- Диаграммы компонент (**Component diagrams**)
- Диаграммы размещения (**Deployment diagrams**)

Типы допустимых элементов и отношений между ними зависят от вида диаграмм. Для иллюстрации диаграмм UML будем использовать постановку задачи «**Регистрация студентов на курсы**», описанную в **приложении**.

Диаграммы сценариев использования (Usecase diagrams — см.

Рисунок 4.2) описывают функциональность ИС, которая будет видна пользователям системы – ее актерам или действующим лицам. Каждая «бизнес-функция» пользователя изображается в виде "сценария использования" (Usecase), который определяет типичное взаимодействие пользователя с системой, которое:

- описывает видимую пользователем функцию;
- может представлять различные уровни детализации;
- обеспечивает достижение конкретной цели, важной для пользователя.

Сценарий использования рисуется как овал, связанный с типичными пользователями, называемыми актерами (actors). Актер, представляющий человека-пользователя, характеризуется ролью в данном сценарии использования (иногда говорят в прецеденте или в варианте использования). На диаграмме изображается только актер (как роль), однако реальных пользователей (или экземпляров актера), выступающих в данной роли по отношению к ИС, может быть много. Список всех прецедентов фактически определяет функциональные требования к ИС, с помощью которых может быть сформулировано ТЗ.

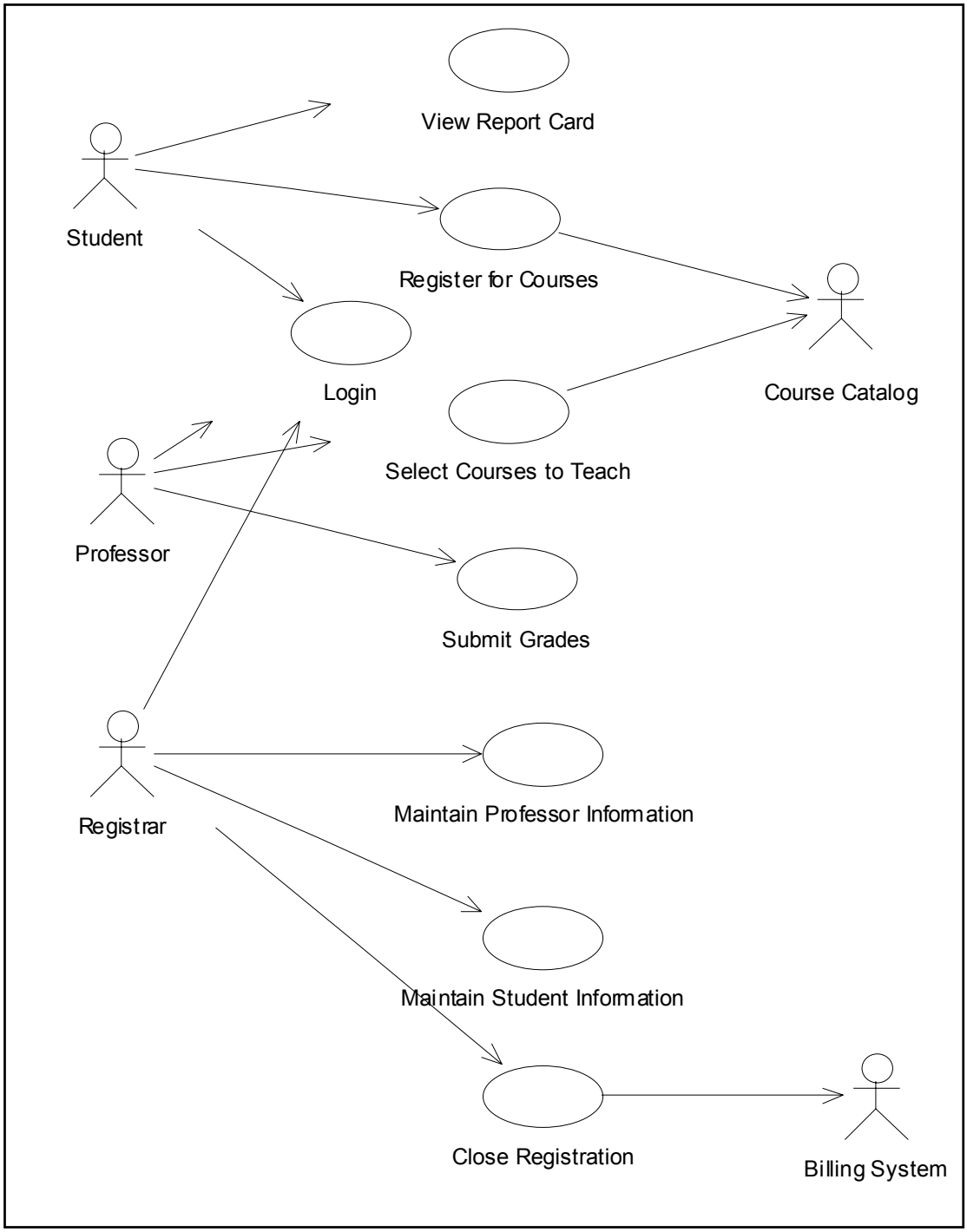


Рисунок 4.2. Диаграмма сценариев использования (Usecase diagram) для задачи – «Регистрация студентов на курсы».

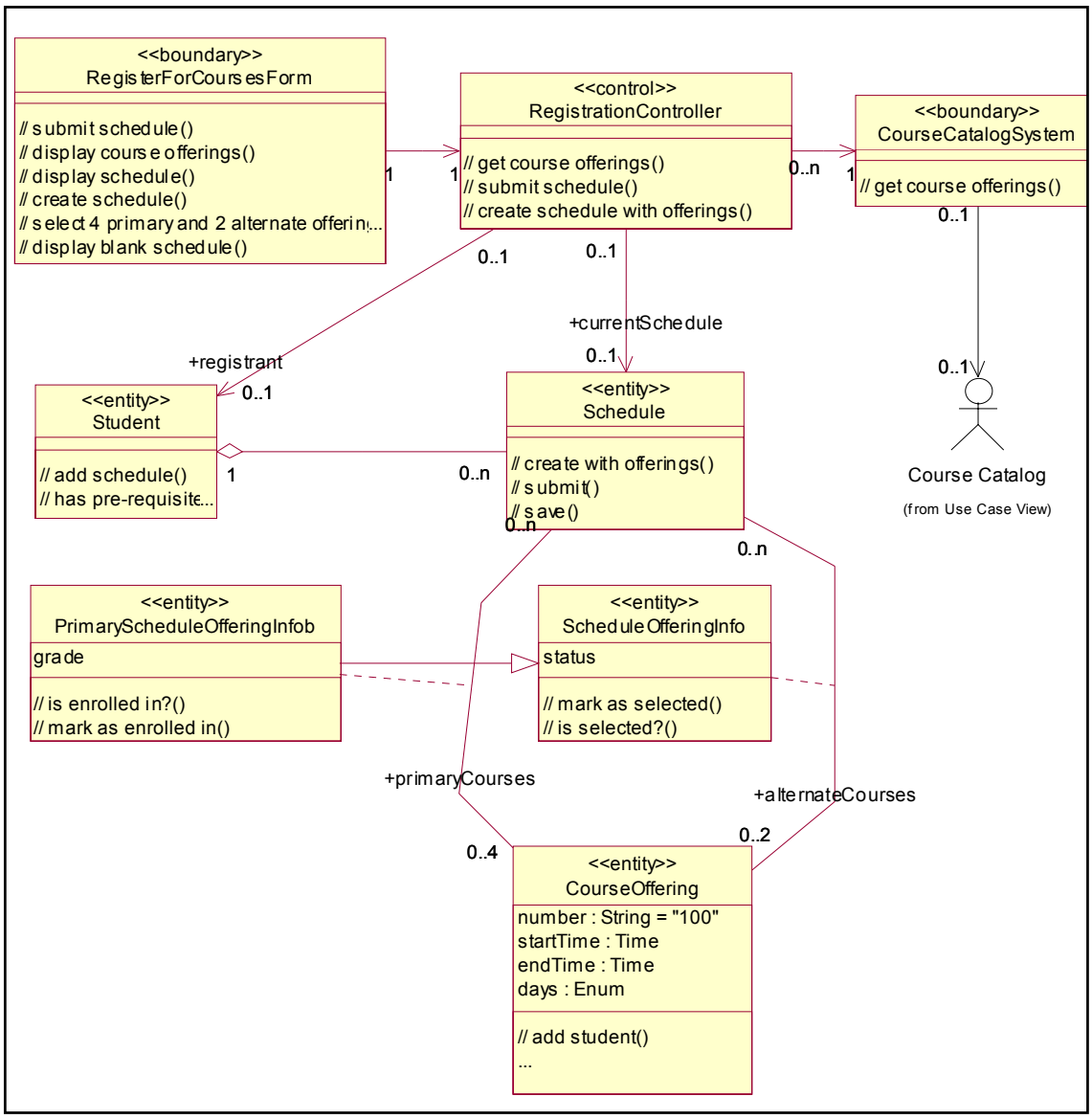


Рисунок 4.3 Диаграмма классов, описывающая классы-участники реализации сценария использования «Зарегистрироваться на курс» (Register for Courses).

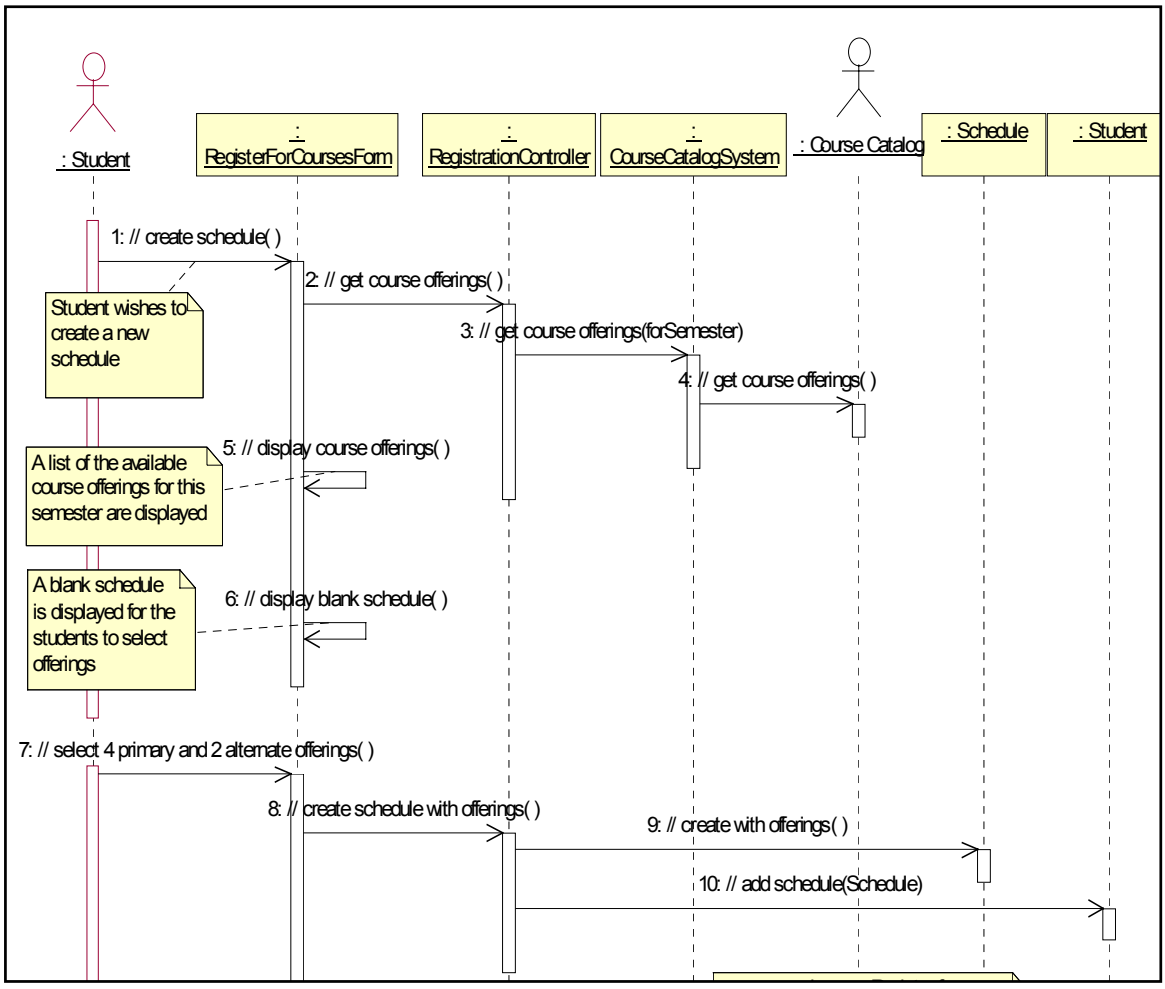


Рисунок 4.4. Диаграмма последовательности (sequence diagram), описывающая реализацию сценария использования «Зарегистрироваться на курс» (Register for Courses). Соответствует диаграмме сотрудничества - Рисунок 4.5.

Диаграмма классов (

Рисунок 4.3) описывают статическую структуру классов и их отношений. Эти диаграммы могут использоваться для описания "предметной области" на концептуальном уровне. С другой стороны, на детальном уровне (уровне спецификаций и уровне реализаций) диаграммы определяют структуру программных классов. Они используются для генерации каркасного программного кода на заданном языке программирования, а также для генерации SQL DDL-предложений, определяющих логическую структуру реляционных таблиц БД.

Для описания динамики применяются:

- диаграммы взаимодействия (interaction diagrams), которые делятся на диаграммы последовательности (sequence diagrams - Рисунок 4.4) и диаграммы сотрудничества (collaboration diagrams - Рисунок 4.5).
- диаграммы состояний (statechart diagrams - Рисунок 4.6);
- диаграммы активностей (activity diagrams - Рисунок 4.7);

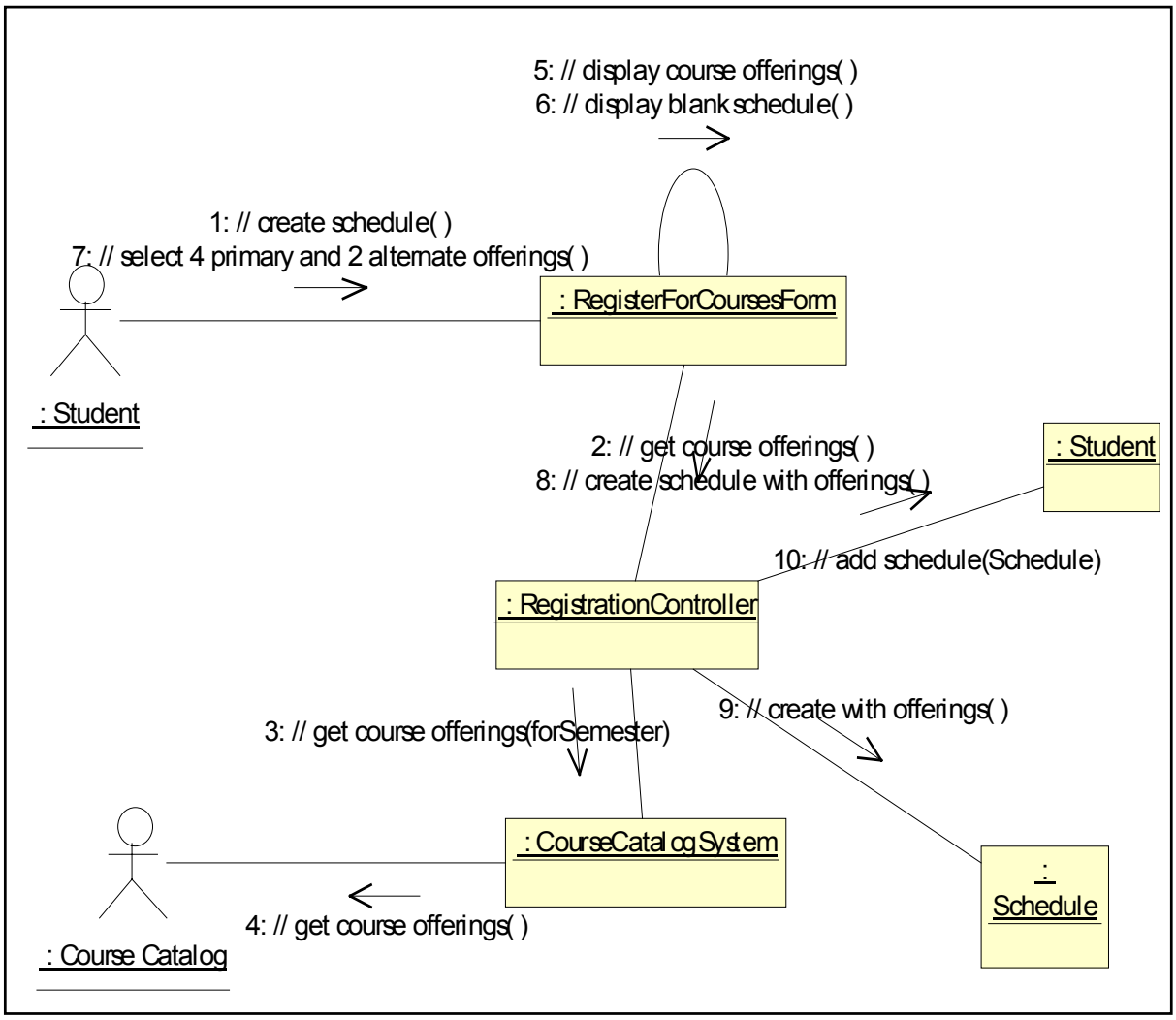


Рисунок 4.5 Диаграмма сотрудничества (Collaboration diagram), описывающая реализацию сценария использования «Зарегистрироваться на курс» (Register for Courses). Соответствует диаграмме последовательности - Рисунок 4.4

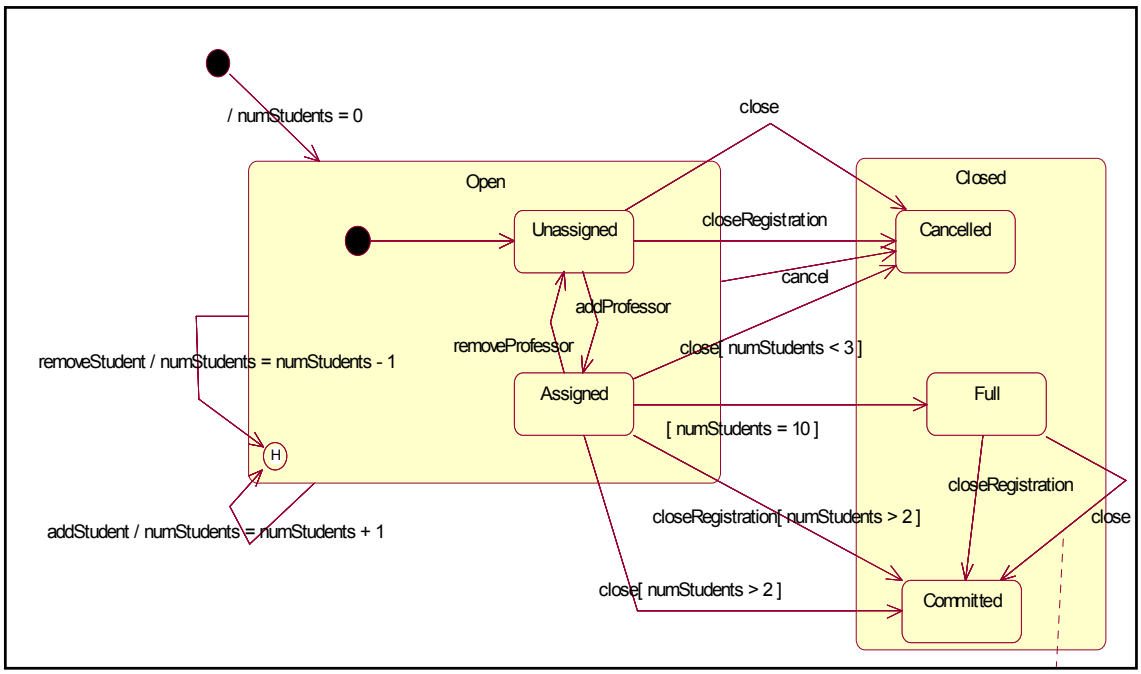


Рисунок 4.6. Диаграмма состояний (Statechart diagrams) объекта «Занятие по курсу» (Course Offering)

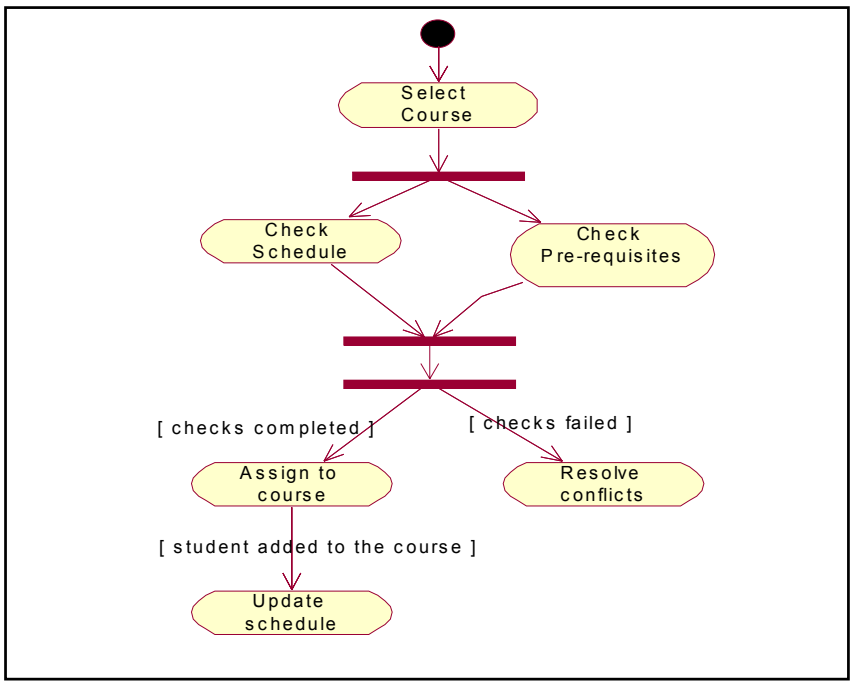


Рисунок 4.7. Диаграмма активности (Activity diagrams) для сценария использования «Регистрация на курсы»

Компонентные диаграммы (component diagrams) и диаграмм развертывания (deployment diagrams) описывают архитектурные аспекты ИС физического уровня.

Порядок использования UML-диаграмм упрощенно можно представить следующим образом. Вначале для ИС определяется ее внешняя функциональность — выделяются все актеры и все прецеденты. Отношения между ними изображаются на диаграммах сценариев использования. Дальнейшая работа над проектом "управляется сценариями использования".

Для каждого сценария строится описание его динамики в виде набора диаграмм взаимодействия и диаграмм сотрудничества; при этом определяются объекты, которые задействованы в его реализации. Затем строятся диаграммы классов, которые определяют статическую структуру, описывающую отношения соответствующих объектов друг с другом. Поведение классов со сложной динамикой реагирования на события определяется на диаграмме состояний. Размещение объектов по программным модулям описывается в компонентных диаграммах, а самих программных модулей в сети — в диаграммах распределения.

Инструментальная поддержка UML - Rational Rose

IBM Rational Rose[22] — CASE-средство визуального моделирования — является простым и интегрированным решением для разработки ИС различной сложности, включая Интернет-решения - Рисунок 4.8.

Основные свойства Rational Rose:

- Построение моделей в нотациях UML
- Возможность описания системы на разных уровнях детализации для различных заинтересованных лиц
- Генерация кодов на языках C++, Java, Visual Basic, PowerBuilder, CORBA IDL
- Генерация описаний баз данных на SQL (Data Modeler)

[22] Приставка IBM перед названием продукта появилась в 2003 году после покупки фирмы Rational Software «голубым гигантом» - корпорацией IBM в декабре 2002 года.

- Возможность восстановления моделей готовых приложений путем обратного проектирования
- Возможность повторного использования проектных решений
- Поддержка групповой разработки
- Интеграция с другими инструментальными средствами

Фактически Rational Rose является стандартом среди инструментов объектно-ориентированного проектирования приложений. С помощью Rational Rose можно визуализировать, изменять и рецензировать модель ИС. Все участники проекта — аналитики, специалисты по моделированию, разработчики Вашей организации — могут использовать Rational Rose для определения архитектуры приложения и общения между собой.

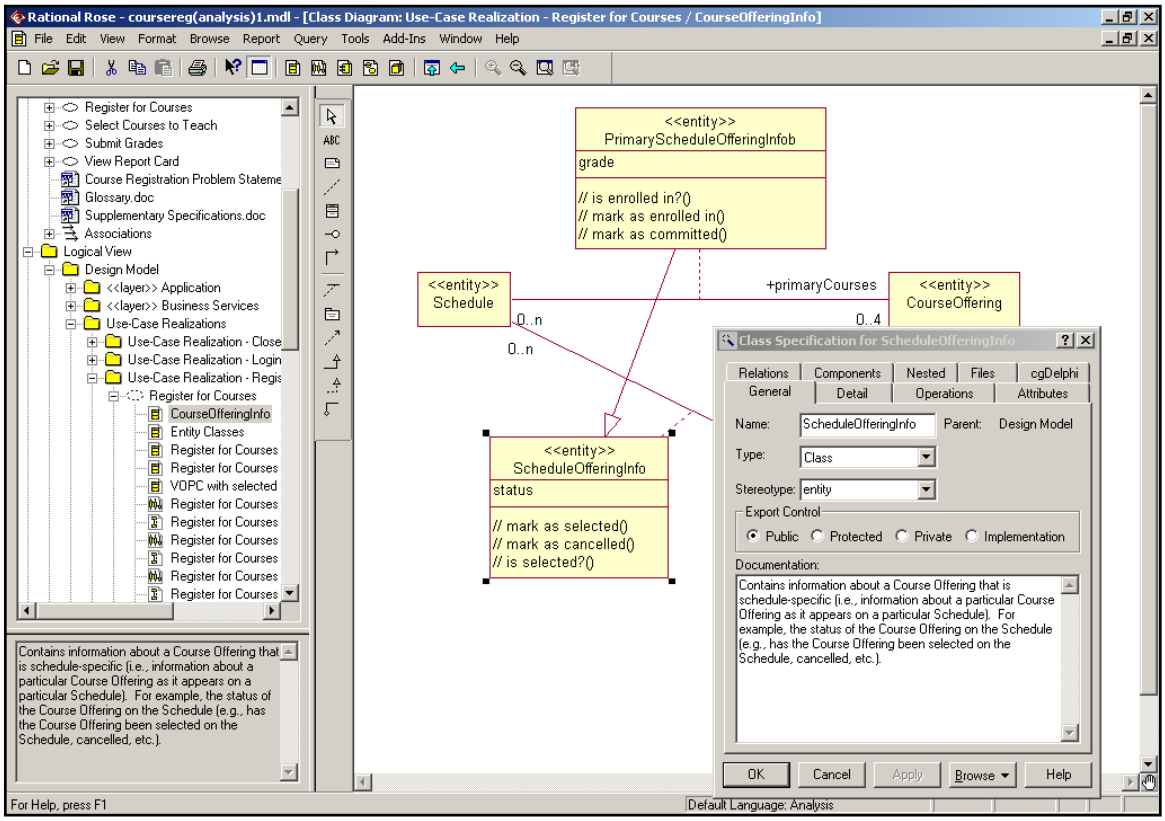


Рисунок 4.8. Экранный интерфейс Rational Rose.

Для бизнес-аналитиков и системных аналитиков —Rational Rose совместно с Rational RequisitePro позволяет визуализировать и моделировать бизнес-процессы и системные требования. Эти модели могут быть использованы совместно с остальными членами команды проекта. Архитекторы могут представлять структуру ИС с разных точек зрения. Для специалистов по БД и аналитиков данных — Rational Rose является общим инструментом, который обеспечивает поддержку логической схемы БД, генерацию схем и итерационную разработку структуры БД - поддерживается определение связей, таблиц, ключевых полей, индексов.

Средство позволяет визуализировать, понять и уточнить требования и архитектуру перед началом создания кода. Позволяя представлять пользовательский интерфейс отдельно от бизнес-логики и данных, Rational Rose дает возможность контролировать ход проектирования ИС. Использование одного инструмента на протяжении всего жизненного цикла разработки помогает строить "правильную" систему.

Визуальное моделирование логической структуры БД

Логическую структуру БД принято представлять в виде модели «сущность-связь» (ER-модели, Entity-Relationship). Рассмотрим простой пример, и будем строить ER-модель на UML с использованием диаграмм классов.

Постановка задачи «Автоматизация поликлиники». Районный отдел здравоохранения принял решение разработать ИС для учета заболеваемости в районе, загруженности медицинского персонала в поликлиниках и в районной больнице. Информационная система должна поддерживать ответы на следующие запросы (за текущий период):

- Сколько пациентов обратилось в поликлиники района?
- По каждому типу болезни, сколько новых диагнозов поставлено?
- Сколько больничных листов выдано по данному типу заболевания?
- Сколько койко-мест свободно (занято) в каждом отделении районной больнице на данное число?
- Какова загруженность врачей данной специальности по поликлиникам района?

В поликлиниках ведется запись пациентов к врачам-специалистам, ИС должна поддерживать эту работу регистратуры. На приеме у врача пациенту ставится диагноз по его жалобе. На приеме пациенту может быть выписан (открыт) или закрыт больничный лист[23], также может быть выписано направление на лечение в районную (областную) больницу. По результату приема пациента медицинская сестра в регистратуре (по документам, предоставленным врачом, в конце рабочего дня) или сам врач (в своем кабинете) вносит в ИС данные о приеме данного пациента.

По направлению на лечение проводится прием в больницу, пациент размещается в соответствующем (диагнозу) отделении больницы, ему выделяется больничная койка (койко-место). После прохождения лечения пациент выписывается из больницы и направляется в поликлинику, где ему закрывают (или продлевают) больничный лист. ИС должна регистрировать поступление и выписку пациентов по отделениям больницы.

Необходимо построить ER модель для данной задачи.

Построение ER модели можно выполнить, «отрабатывая» следующие шаги:

1-й шаг. Из постановки задачи выявить информационные объекты (сущности);

2-ой шаг. Выявленные объекты разделить на объекты-«предметы» (объекты-«роли») и объекты-события. Как правило, именно объекты-события интересны с точки зрения «учета и контроля» характеристик бизнес процессов;

3-й шаг. Для каждого объекта-события:

Выявить его участников из числа объектов-«предметов»;

Визуально изобразить[24] результат шага 3.1 для данного объекта-события в виде диаграммы классов UML. Сущности (объекты) представлять классами, отношения между ними – ассоциациями;

---

[23] Больничный лист – документ строгого учета, освобождающий человека от работы по болезни. После выздоровления пациента врач закрывает больничный лист, который сдается по месту работы как основание отсутствия сотрудника на работе.

[24] С использованием инструментального CASE средства – например, IBM Rational Rose.

4-й шаг. Построить **«сводную» диаграмму**, содержащую все классы (сущности) и все их взаимосвязи. Это шаг выполняется (в Rational Rose) автоматически.

5-й шаг. Провести реструктуризацию объектов/связей.

6-й шаг. Расставить множественность - Для каждого конца ассоциации **определяем множественность**, отвечая на вопрос: «сколько объектов может быть связано с данным конкретным объектом»? Некоторые варианты расстановок значения множественности (иногда говорят - кардинальности):

- «1:1» - отношение «один к одному»;
- «1:*» (или 1:N) отношение «один к многим»;
- «N:M» отношение «много к многим» - как правило, нуждается в дальнейшем уточнении.

7-й шаг. Провести **рецензирование** и обсуждение модели, разослав ее заинтересованным лицам. Полученную модель можно опубликовать (в Rational Rose) в виде веб-сайта (во внутренней локальной сети) и выслать заинтересованным лицам указатель на этот сайт.

Для нашего примера:

1. Выявляем информационные объекты:

пациент, запись к врачу, прием у врача, врач-специалист, специальность врача, диагноз, тип болезни, больничный лист (Б/Л), открытие Б/Л, закрытие Б/Л, поликлиника, больница, отделение больницы, направление в больницу, поступление в больницу, выписка из больницы.

2. Разделяем объекты на объекты-«предметы» и объекты-события:

Объекты-«предметы»: пациент, врач-специалист, специальность врача, диагноз, тип болезни, больничный лист (Б/Л), поликлиника, больница, отделение больницы - Рисунок 4.9
Объекты-«события»: запись к врачу, прием у врача, открытие Б/Л, закрытие Б/Л, направление в больницу, поступление в больницу, выписка из больницы - Рисунок 4.10

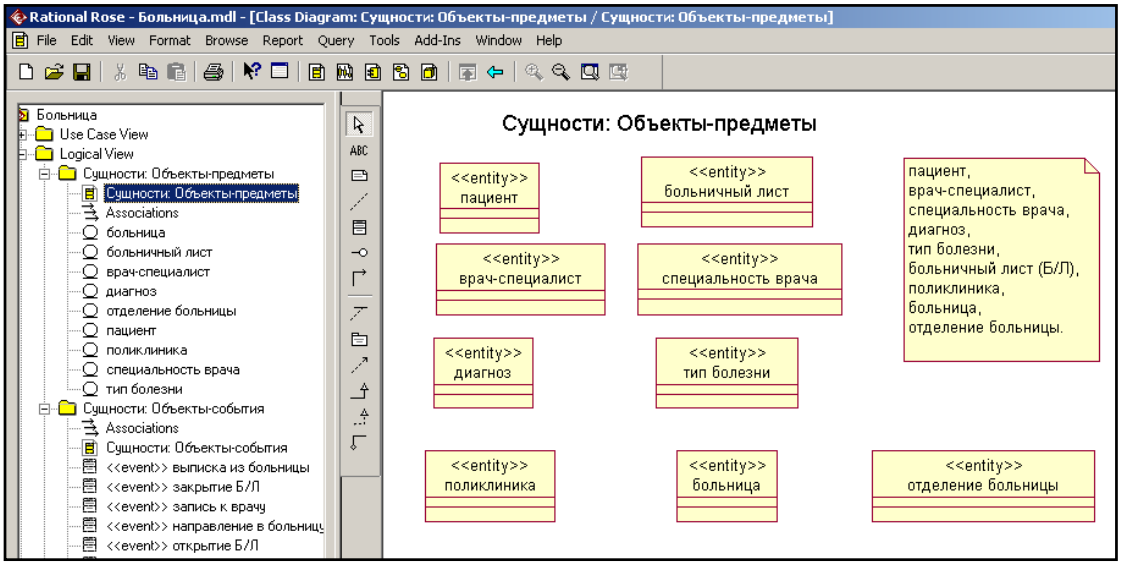


Рисунок 4.9 Объекты-«предметы» задачи «Автоматизация поликлиники».

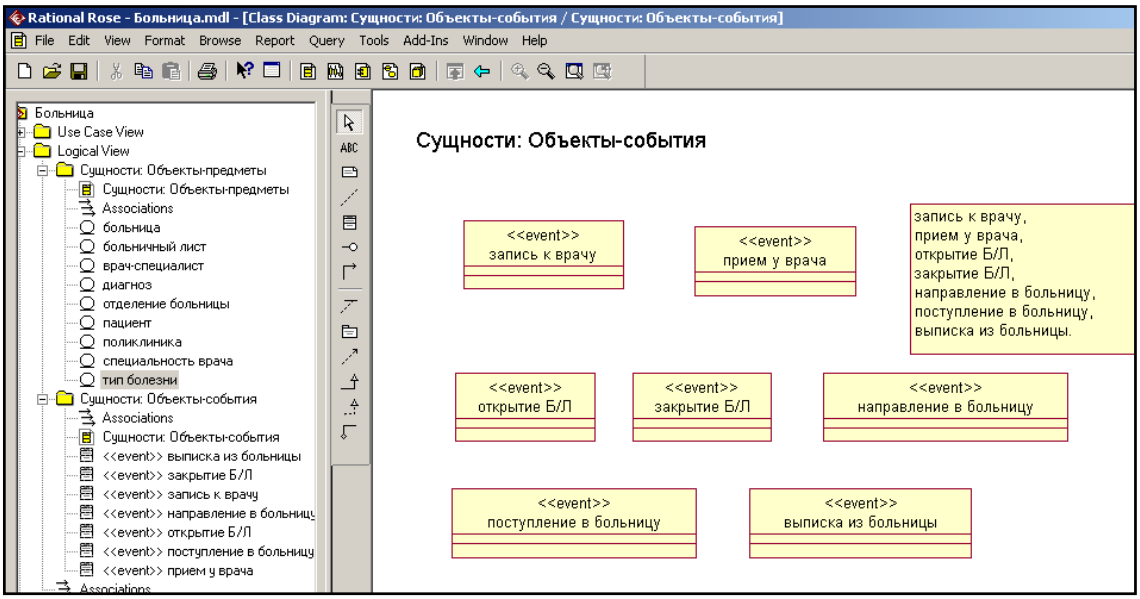


Рисунок 4.10. Объекты-«события» задачи «Автоматизация поликлиники».

Для каждого объекта-события выявляем его «участников» из числа объектов-«предметов» и строим диаграмму классов. Для этого отвечаем на вопрос: «Какую информацию (какие сущности-предметы) должно знать событие?»

Запись к врачу. Событие связано с (1) пациентом, который записывается, (2) – с врачом и с поликлиникой - Рисунок 4.11

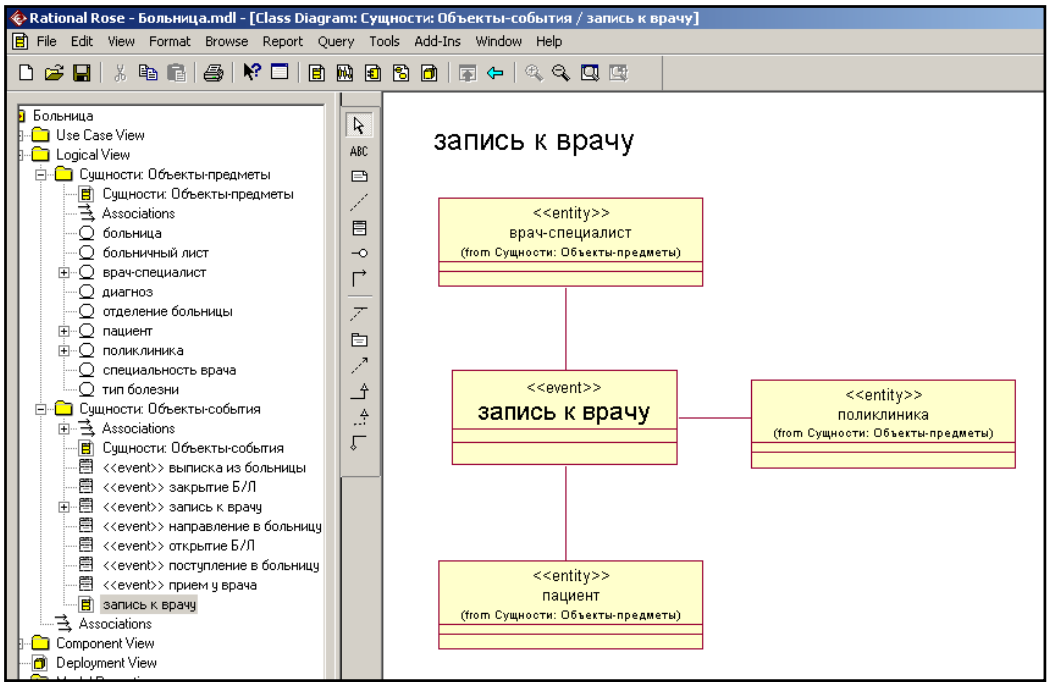


Рисунок 4.11. Событие «Запись к врачу».

В дальнейшем эта диаграмма будет нуждаться в реорганизации, поскольку, врач связан с поликлиникой (Рисунок 4.12), и событие «запись к врачу» будет связано с поликлиникой через врача.

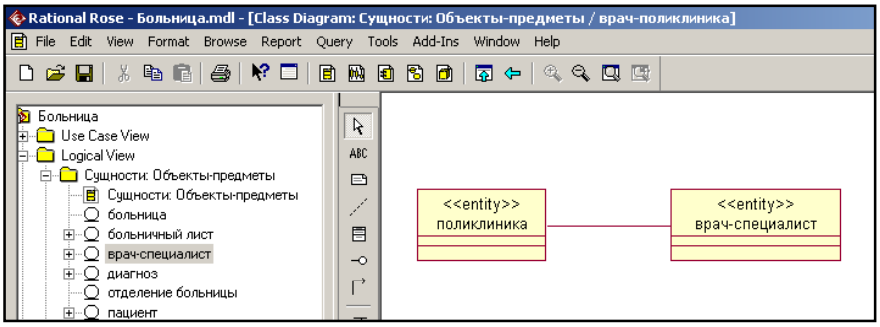


Рисунок 4.12. Связь врача с поликлиникой.

Прием у врача. Событие связано с (1) пациентом, (2) – с врачом, (3) – с диагнозом, на приеме может быть выписан/продлен/закрыт больничный лист, т.е. (4) – с больничным листом - Рисунок 4.13.

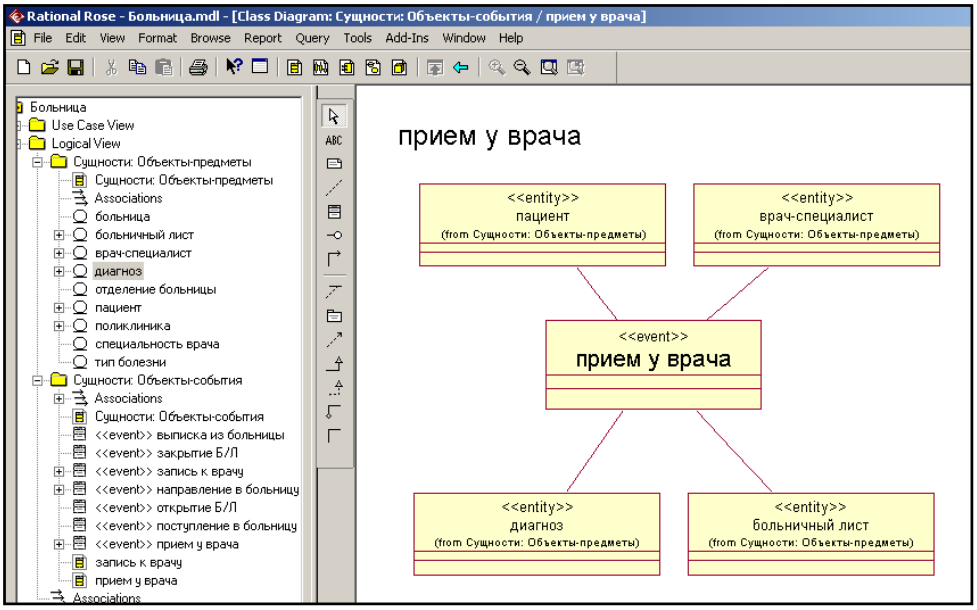


Рисунок 4.13. Событие «Прием у врача».

События открыть/закрыть больничный лист оказываются содержательно связаны с событием «Прием у врача». Принимаем решение – отдельно эти события («открыть/закрыть Б/Л») не будем «расписывать» в визуальном виде.

Связь двух событий – на приеме может быть принято решение о госпитализации пациента, т.е. произойдет событие «направление в больницу» -Рисунок 4.14.

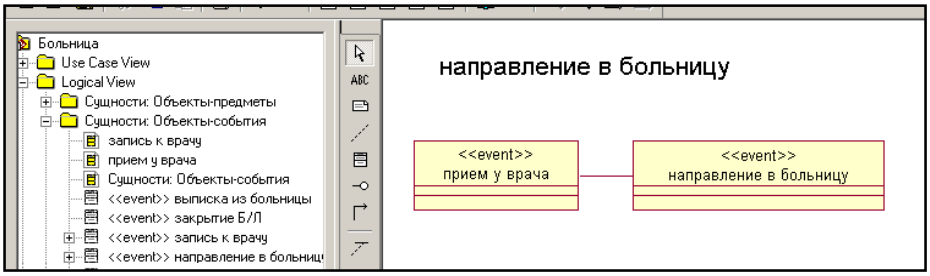


Рисунок 4.14. Связь двух событий «Прием у врача» и «Направление в больницу».

Поступление в больницу. Событие связано с пациентом, с отделением больницы и с диагнозом - Рисунок 4.15.

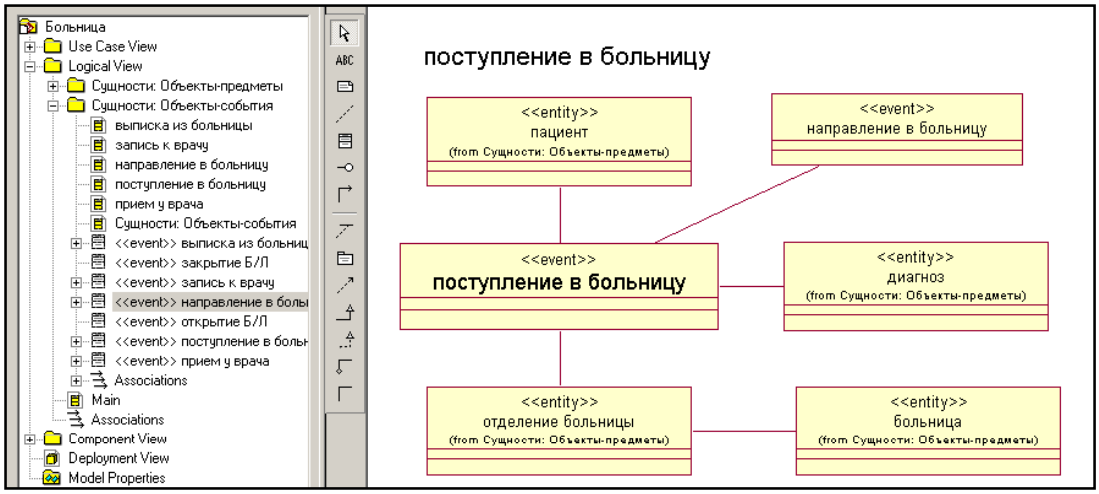


Рисунок 4.15. Событие «Поступление в больницу».

Выписка из больницы. Событие связывается с «поступлением в больницу» и через него связывается с пациентом, диагнозом и отделением - Рисунок 4.16.

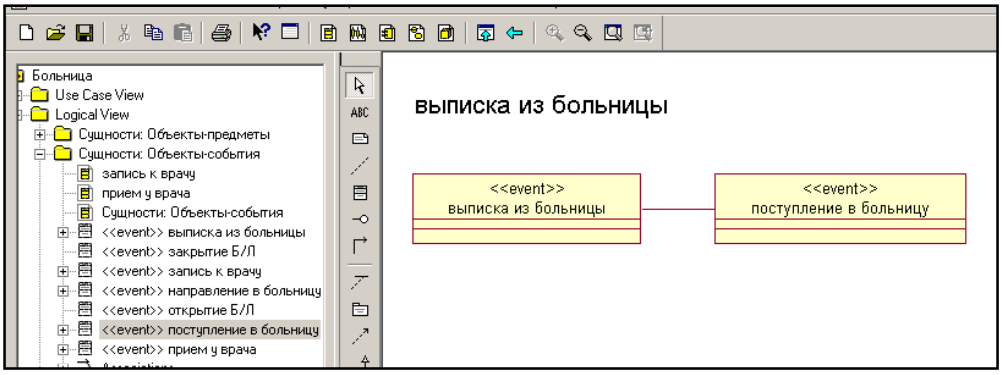


Рисунок 4.16. Событие «Выписка из больницы».

4-й шаг. Автоматически строим «сводную» диаграмму в Rational Rose - Рисунок 4.17.

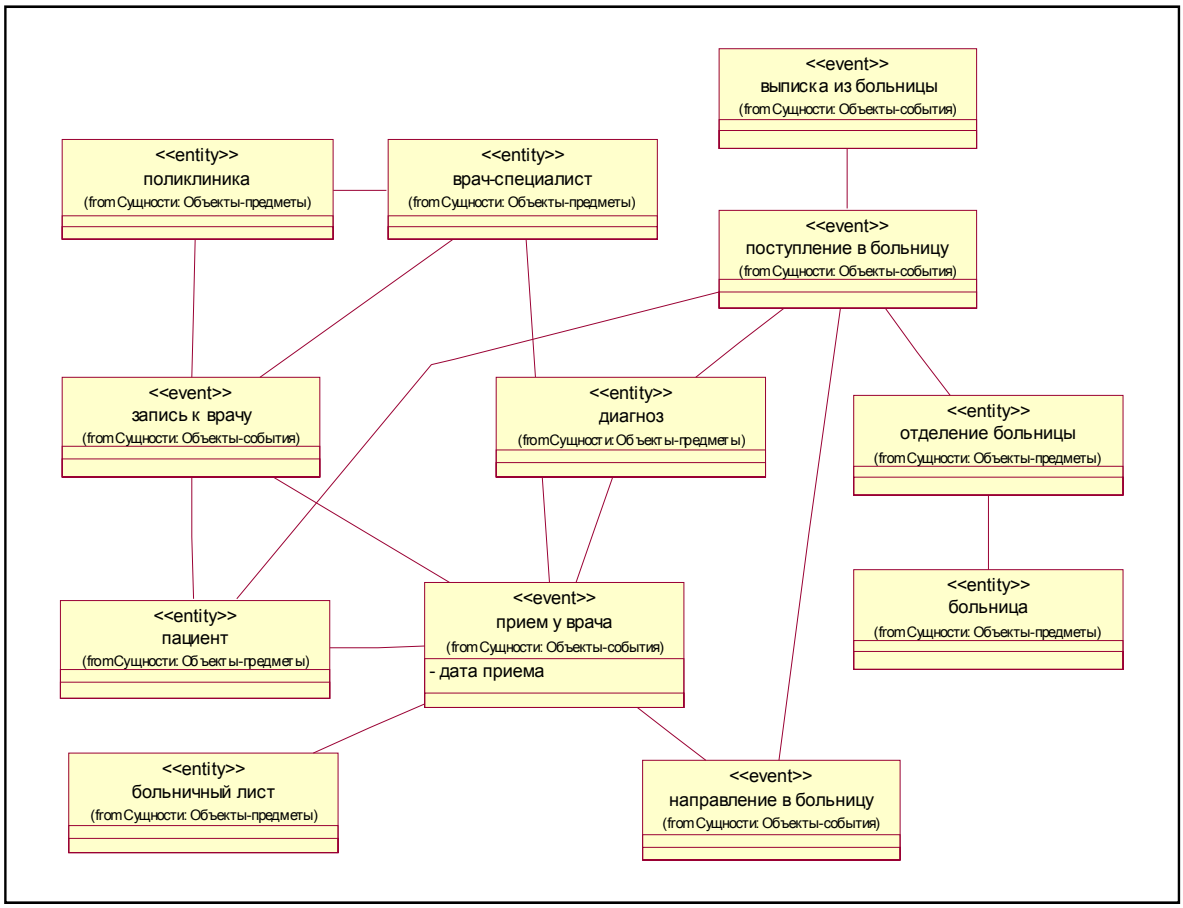


Рисунок 4.17. Автоматически сгенерированная в Rational Rose «сводная» диаграмма классов – до реструктуризации.

5-й шаг. Проводим реструктуризацию объектов/связей. **Снимаем связь** «запись к врачу» - «поликлиника», т.к. она идентифицируется через врача-специалиста. «Запись к врачу» и «прием у врача» - очень «близкие» события, имеющие похожую структуру связей. **Объединим эти две сущности,** и будем считать, что «прием у врача» имеет атрибут «состояние» с двумя значениями: «запись» (прима еще не было) и «прием» (прием у врача уже состоялся). Для диагноза использует дополнительную классификацию (сущность) «тип болезни», а для врача, – классификацию, – «специальность».Рисунок 4.18 изображает результат реструктуризации в виде сводной диаграммы классов.

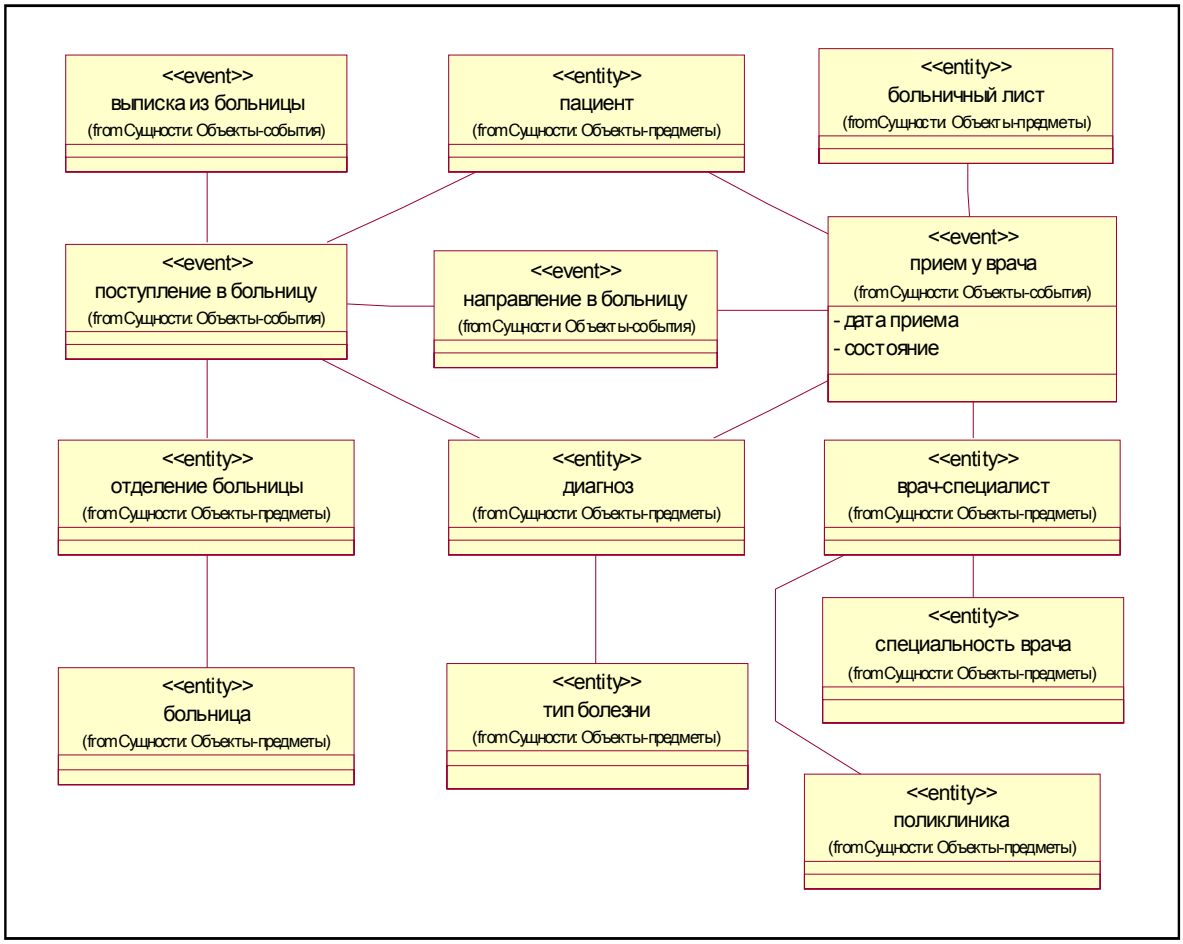


Рисунок 4.18. «Сводная» диаграмма классов (модель «сущность-связь»)– после реструктуризации и до идентификации множественности ассоциаций.

6-й шаг. Расставляем множественность ассоциаций.

Например, пациент (конкретный) может быть связан со многими приемами у врача; конкретный прием у врача связан ровно с одним пациентом. Аналогично можно расставить множественность у ассоциации «Врач-специалист» - «приемами у врача». Рисунок 4.19 изображает результат выполнения этого шага в виде сводной диаграммы классов.

7-й шаг. Проводим рецензирование и обсуждение модели с заинтересованными лицами, включая разработчиков ИС «Автоматизация поликлиники».

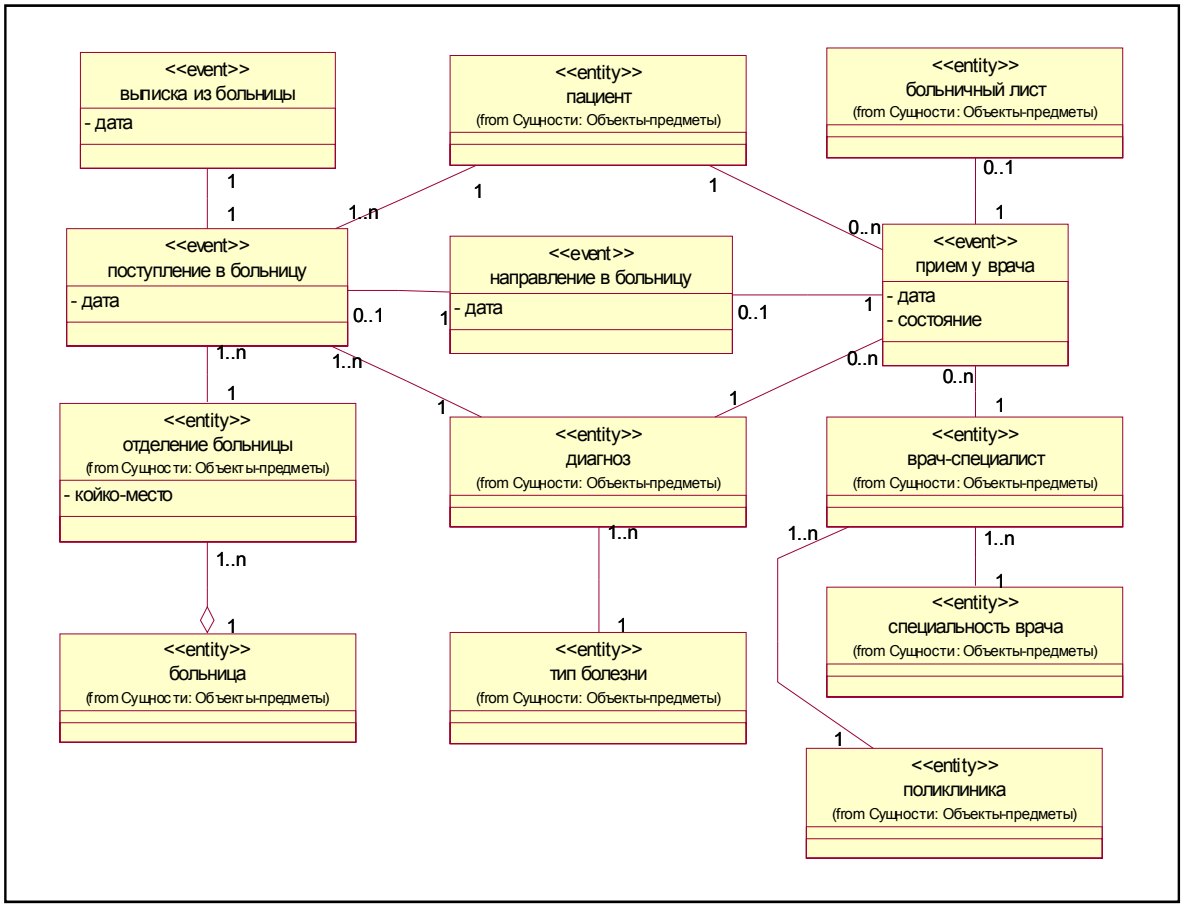


Рисунок 4.19. «Сводная» диаграмма классов (модель «сущность-связь») задачи «Автоматизация поликлиники».

Вопросы

1. Каковы основные принципы «борьбы со сложностью» при разработке системы?

2. Чем отличается структурная декомпозиция системы от объектно-ориентированной?

3. Создайте диаграмму сценариев использования для задачи «Расчет платежной ведомости» (см. Приложение 2)

3. Постройте модель «сущность-связь» для задачи «Расчет платежной ведомости» (см. Приложение 2)

5. Заключение

Подведем итоги. Термин «базы данных» может использоваться на различных уровнях и в различных контекстах и включает в себя:

- **Центральное хранилище данных** (или совместно используемый механизм, предоставляющий общее хранилище взаимосвязанных и управляемых данных); БД – это структурированные данные.
- Инструментальное средство поиска и анализа и отображения информации пользователей ; БД – это **система управления базой данных (СУБД)**.
- Концептуальная модель (или модель для представления состояния организации, как в кратковременном, так и в долговременном аспектах). БД – это **модель информационных сущностей предприятия**.

СУБД берет на себя обеспечение ряда ключевых функций, о которых «могут не заботиться» прикладные программисты (СУБД «инкапсулирует» эти функции):

- ***Параллельный (одновременный) доступ*** к данным многих прикладных программ ИС. Число одновременных подключений к БД может достигать нескольких сотен или даже тысяч. СУБД обеспечивает иллюзию того, что каждая прикладная программа работает с ***общими данными*** независимо от других. Как правило, прикладные программы ИС обращаются не ко всем данным БД, а к некоторой ее части, доступной данному приложению (или данному пользователю). Обеспечение и разграничение такого доступа к общим данным – важная функция СУБД.
- ***Безопасность: Разграничение прав доступа к данным.*** СУБД ведет список зарегистрированных пользователей данной БД, включая описание подмножества данных, «видимых» для данного пользователя и права доступа к этим данным (право читать, обновлять, создавать новые записи, удалять существующие записи). Как видим, этот список – маленькая внутренняя «системная» БД (метаданные).
- ***Надежность: Восстановление данных после сбоев ИС.*** СУБД имеет специальные программные средства для архивирования данных и их оперативного восстановления после сбоев.

Ответственность за периодическое архивирование данных лежит на администраторе БД. Он же и восстанавливает БД после сбоя.

- ***Непротиворечивость: Обеспечение целостности данных.*** В результате независимого и одновременного доступа к одним и тем же данным возможно появление ошибок в данных. Для их предотвращения на этапе проектирования данных определяются правила поддержки целостности. Одна из важных задач СУБД – автоматически следить за их выполнением, тем самым, поддерживая непротиворечивость (или целостность) данных.

Использование СУБД приводит к концепции ***централизованного управления*** при использовании данных. *С логической точки* зрения на предприятии имеется единое компьютерное хранилище данных, структура которых единообразно представлена в ***корпоративной СУБД*** в виде метаданных. Все вновь создаваемые приложения используют подмножество общих данных. Централизованный подход к использованию данных на предприятии имеет ряд существенных преимуществ:

1. Возможность сокращения избыточности данных. Например, данные по клиентам организации единообразно ведутся в одном наборе данных («таблице») СУБД, которой пользуется для решения своих задач различные подразделения предприятия: отдел продаж, бухгалтерия, отдел снабжения и т.д.
2. Унификация данных (соблюдение корпоративных стандартов). Все данные единообразно описаны и одинаково представляются для визуализации и /или обмена данными.
3. Возможность общего и одновременного доступа к данным.
4. Разграничение доступа для обеспечения безопасности.

Общие тенденции развития БД/ИС состоят в следующем:

- Распределенность и децентрализация ресурсов управления информацией;
- Неоднородность компонентов ИС;
- Использование стандартов;
- Использование моделирования бизнес процессов и ИС.

6. Приложения

Приложение 1.

Модель «сущность-связь» Учебной БД

Рисунок **2** иллюстрирует модель «сущность-связь» Учебной БД,

Рисунок **3** - логическую схему Учебной БД, автоматически сгенерированная в Rational Rose, а Рисунок 4 - набор DDL операторов SQL (в стандарте ANSI SQL 92) для создания схема Учебной БД, автоматически сгенерированный в Rational Rose.

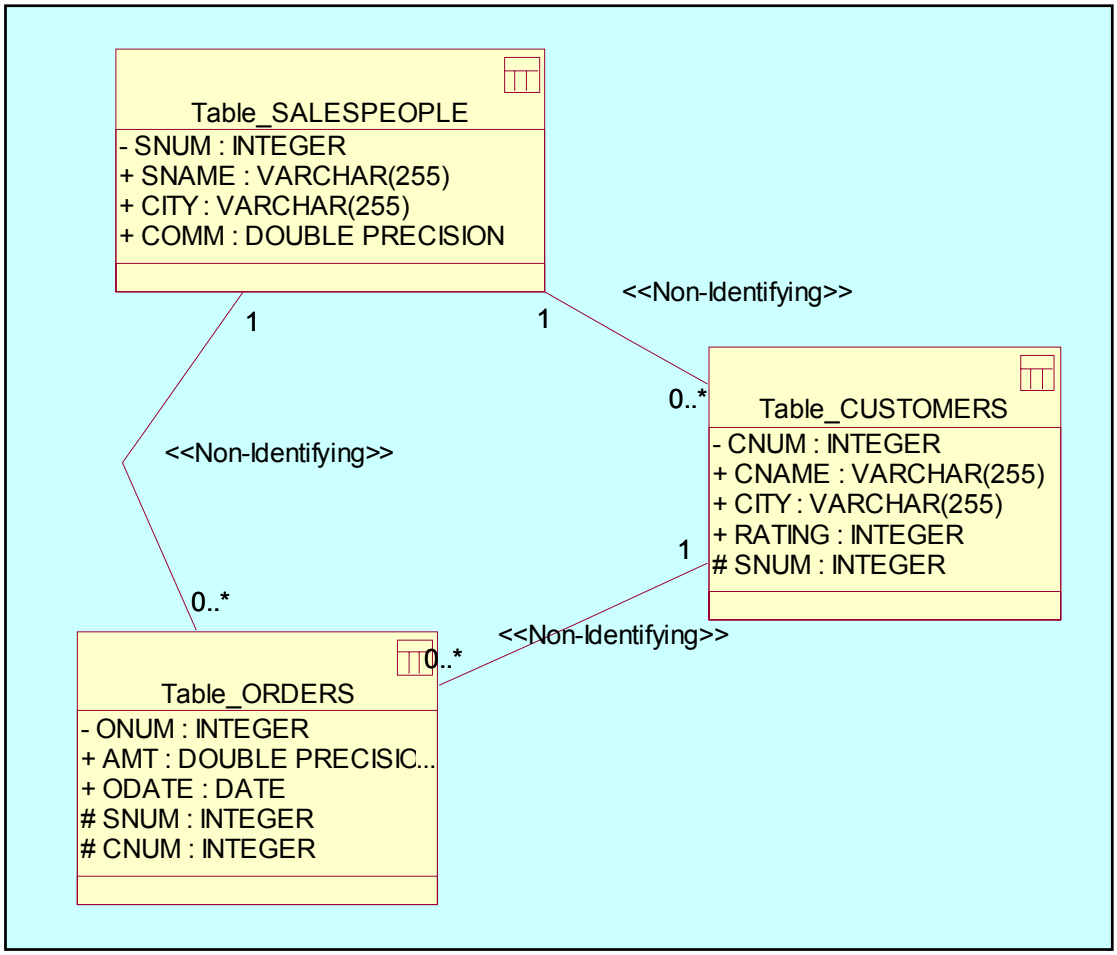


Рисунок 2. Диаграмма классов – модель «сущность-связь» Учебной “SCO” БД.

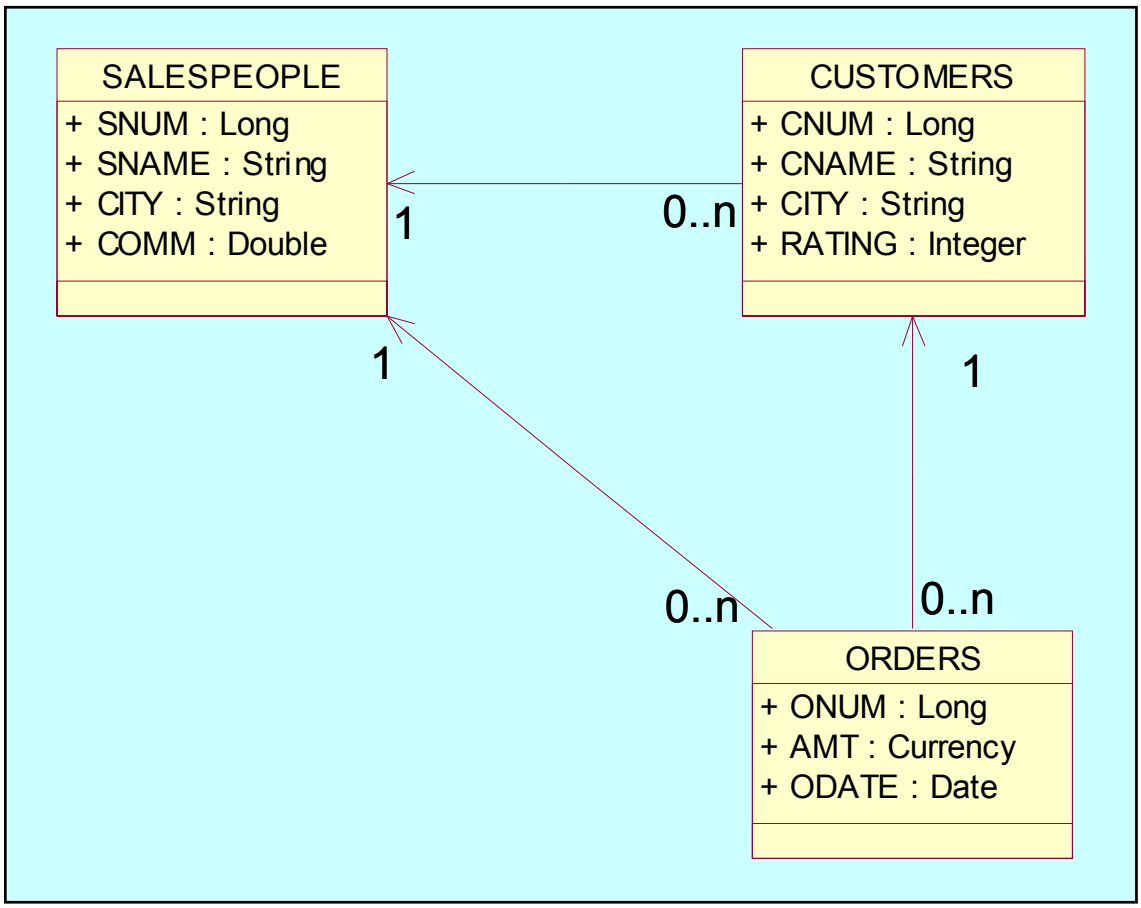


Рисунок 3. Логическая схема Учебной “SCO” БД, автоматически созданная в Rational Rose.

```
CREATE TABLE Table_SALESPEOPLE (
        SNUM INTEGER NOT NULL,
        SNAME VARCHAR ( 255 ) NOT NULL,
        CITY VARCHAR ( 255 ) NOT NULL,
        COMM DOUBLE PRECISION NOT NULL,
        CONSTRAINT TC_Table_SALESPEOPLE12 UNIQUE (SNUM),
        CONSTRAINT PK_Table_SALESPEOPLE6 PRIMARY KEY (SNUM)
        );
CREATE TABLE Table_ORDERS (
        ONUM INTEGER NOT NULL,
        AMT DOUBLE PRECISION NOT NULL,
        ODATE DATE NOT NULL,
        SNUM INTEGER NOT NULL,
        CNUM INTEGER NOT NULL,
        CONSTRAINT PK_Table_ORDERS4 PRIMARY KEY (ONUM),
        CONSTRAINT TC_Table_ORDERS14 UNIQUE (ONUM)
        );
CREATE TABLE Table_CUSTOMERS (
        CNUM INTEGER NOT NULL,
        CNAME VARCHAR ( 255 ) NOT NULL,
        CITY VARCHAR ( 255 ) NOT NULL,
        RATING INTEGER NOT NULL,
        SNUM INTEGER NOT NULL,
        CONSTRAINT TC_Table_CUSTOMERS13 UNIQUE (CNUM),
        CONSTRAINT PK_Table_CUSTOMERS5 PRIMARY KEY (CNUM)
        );
CREATE INDEX TC_Table_CUSTOMERS9 ON Table_CUSTOMERS (SNUM);
ALTER TABLE Table_ORDERS ADD CONSTRAINT FK_Table_ORDERS4 FOREIGN KEY
(CNUM) REFERENCES Table_CUSTOMERS (CNUM)  ON DELETE NO ACTION ON UPDATE
NO ACTION;

ALTER TABLE Table_ORDERS ADD CONSTRAINT FK_Table_ORDERS6 FOREIGN KEY
(SNUM) REFERENCES Table_SALESPEOPLE (SNUM)  ON DELETE NO ACTION ON
UPDATE NO ACTION;

ALTER TABLE Table_CUSTOMERS ADD CONSTRAINT FK_Table_CUSTOMERS5 FOREIGN
KEY (SNUM) REFERENCES Table_SALESPEOPLE (SNUM)  ON DELETE NO ACTION ON
UPDATE NO ACTION;
```

Рисунок 4. Набор DDL операторов SQL (в стандарте ANSI SQL 92) для создания схема Учебной “SCO” БД, автоматически сгенерированный в Rational Rose.

Приложение 2.

Постановка задачи «Регистрация студентов на курсы»

Вам поручено развитием новой системы регистрации студентов. Колледж хотел бы, чтобы новая система типа «клиент-сервер» заменила старую систему, разработанную на универсальной ЭВМ. Новая система позволит слушателям регистрироваться на курсы и просматривать свои зачетные ведомости на персональных компьютерах, присоединенных к локальной сети (LAN) университетского городка. Профессора будут иметь доступ к системе, чтобы подтвердить преподавание курсов, а также записывать отметки (экзаменационные оценки) студентов.

Из-за уменьшения в федеральном финансировании колледж не может позволить себе заменять всю старую систему одновременно. Колледж будет поддерживать существующую базу данных каталога курса, где собрана вся информация о курсах. Эта база данных - Ingress реляционная база данных выполняющаяся на DEC VAX. Колледж вложил капитал в открытый интерфейс SQL, который позволяет иметь доступ этой базе данных от серверов Unix колледжа. Производительность унаследованной системы довольно бедна, так что новая система должна обеспечить быстрый доступ к данным унаследованной системы. Новая система будет читать информацию о курсах из унаследованной базы данных, но не будет модернизировать ее. Офис колледжа продолжит поддержку информации о курсах через другую систему.

В начале каждого семестра слушатели могут запрашивать каталог курсов, содержащий список семинарских занятий в течение семестра. Информация о каждом курсе, такая как профессор, кафедра, и пререквизиты курса будет включена в каталог, чтобы помочь слушателям принять информированное решение.

Новая система позволит слушателям выбирать четыре семинарских занятия в течение текущего семестра. Кроме того, каждый слушатель должен указать два альтернативных выбора в случае, если слушатель не может быть назначен на первичный курс. Семинарские занятия будут иметь максимум 10 слушателей и минимума 3 слушателя. Семинарское занятие с меньшим, чем три числом слушателей будет отменено. В течение каждого семестра, есть период времени, что слушатели могут изменять свой график занятий. Слушатели должны имеют доступ система, чтобы добавлять или удалять курсы из своего расписания (графика занятий). Как только процесс регистрации закончен, система регистрации посылает информацию в систему выписки счетов, так что слушатель мог оплатить свое обучение в данном семестре. Если курс заполняется в течение фактического процесса регистрации, слушатель должен быть уведомлен об этом перед передачей в систему своего графика занятий для обработки.

В конце семестра, слушатель должен иметь доступ к системе, чтобы просмотреть свою электронную зачетную ведомость. Так как студенческие отметки – конфиденциальная информация, система должна использовать дополнительные меры безопасности, чтобы предотвратить несанкционированный доступ к ней.

Профессора должны иметь доступ к диалоговой системе, чтобы указать курс, который они будут преподавать. Они также должны иметь возможность просматривать список слушателей, записанных на их семинарские занятия. Кроме того, профессора должны иметь возможность делать запись отметок слушателей по каждому курсу.

Постановка задачи «Расчет платежной ведомости»

Вам поручено создать новую систему расчета платежной ведомости, чтобы заменить существующую систему, которая является безнадежно устаревшей. Асте нуждается в новой системе, чтобы позволить служащим записывать информацию об отработанном времени и автоматически производить расчет зарплаты, основанный на числе отработанных часов и общей суммы продаж (для уполномоченных служащих).

Служащие будут вводить карту учета времени (табель), вводить информацию о продажах, изменять свой метод оплаты и создавать различные отчеты. Система будет работать на индивидуальных рабочих местах служащих повсюду во всей компании. По соображениям безопасности и аудита, служащие могут иметь доступ и редактировать только их собственные карты учета времени и данные о продажах. Система должна хранить информацию обо всех служащих компании и должна платить каждому служащему правильное количество денег, вовремя и методом, который они выбрали. Система должна использовать унаследованную базу данных, - Project Management Database (), которая содержит всю информацию относительно открытых проектов и о номерах проектов. Новая система должна работать с PMDB, - она будет только читать ее, но не изменять информацию, хранимую в PMDB.

Некоторые служащие работа на почасовой основе, т.е. оплачиваются на основе почасовой нормы. Они представляют в систему карты учета времени, которые содержат запись даты, и число часов, отработанных по конкретному номеру проекта. Почасовые служащие получают оплату каждую пятницу.

Некоторые служащие имеют фиксированный заработок. Хотя они имеют фиксированный заработок, они также предоставляют карты учета времени, которые содержат запись даты, и отработанные часы. В результате система может рассчитывать общее число часов, отработанных по конкретному номеру проекта. Они получают оплату каждый последний рабочий день месяца.

Некоторые из штатных служащих также получают комиссионные, основанные на их продажах. Они представляют в систему данные о продажах, которые отражают дату и размер продажи. Норма комиссии определена для каждого служащего и может составлять 10%, 15%, 25 %, или 35 %.

Одно из наиболее нужных свойств новой системы – отчеты служащего. Служащие будут запросить систему для выдачи числа отработанных часов, общее количество всех часов, оплаченных по номеру данного проекта, полная плата, полученная за год до настоящего времени, оставшееся время отпуска, и т.д. Служащие могут выбирать свой метод оплаты. Фирма может отправить их чеки с зарплатой по почтовому адресу, который они указали. Они могут выбрать прямое перечисление денег (депозит) на счет банка по их выбору. Служащий может получать чек с зарплатой непосредственно в офисе.

Администратор должен поддерживать информацию обо всех служащих. Администратор отвечает за добавление новых служащих, удаление служащих и изменения информации служащего (имя, адрес, классификация статуса оплаты (почасовый, штатный, уполномоченный)), также может формировать административные отчеты.

Расчет платежной ведомости должен автоматически запускаться каждый пятницу и в последний рабочий день месяца, - для оплаты работы соответствующих служащих в эти дни. Новая система должна быть разработана так, чтобы расчет платежной ведомости всегда производился автоматически без потребности в каком-либо ручном вмешательстве.

Приложение 3

Ответы к упражнениям по SQL

Упражнение 1.

```
1. SELECT onum, amt, odate
        FROM Orders;

2. SELECT *
        FROM Customers
        WHERE snum = 1001;

3 SELECT city, sname, snum, comm
        FROM Salespeople;

4. SELECT rating, cname
        FROM Customers
        WHERE city = 'SanJose';

5. SELECT DISTINCT snum
        FROM Orders;
```

Упражнение 2.

```
1. SELECT * FROM Orders WHERE amt > 1000;

2. SELECT sname, city
        FROM Salespeople
        WHERE city = 'London' AND comm > .10;

3. SELECT *
        FROM Customers
        WHERE rating > 100 OR city = 'Rome';
```

или

```
SELECT *
        FROM Customers
        WHERE NOT rating < = 100 OR city = 'Rome';
```

или

```
SELECT *
        FROM Customers
        WHERE NOT (rating < = 100 AND city < > 'Rome');
```

4. Выводятся заказы, размер которых меньше 1000, или заказы, которые сделаны 10 марта 2003 года покупателями, имеющими значение номера больше 2017.

5. SELECT *
FROM Salespeople;

Упражнение 3.

1. SELECT *
FROM Orders
WHERE odate IN (10/03/1990, 10/04/1990);
и

SELECT *
FROM Orders
WHERE odate BETWEEN 10/03/1990 AND 10/04,1990;

2. SELECT *
FROM Customers
WHERE cname BETWEEN 'A' AND 'H';

3. SELECT *
FROM Customers
WHERE cname LIKE 'C%';

5. SELECT *
FROM Orders
WHERE amt = 0 AND (amt IS NULL);
или
SELECT *
FROM Orders
WHERE NOT (amt = 0 OR amt IS NULL);

Упражнение 4.

1. SELECT COUNT(*)
FROM Orders
WHERE odate = 10/03/1990;

2. SELECT COUNT (DISTINCT city)
FROM Customers;

```
3. SELECT cnum, MIN (amt)
     FROM Orders
     GROUP BY cnum;

4. SELECT MIN (cname)
    FROM Customers
    WHERE cname LIKE 'G%';

5. SELECT city, MAX (rating)
     FROM Customers
     GROUP BY city;

6. SELECT odate, count (DISTINCT snum)
    FROM Orders
    GROUP BY odate;

7. SELECT rating, cname, cnum
    FROM Customers
    ORDER BY rating DESC;
8. SELECT odate, SUM (amt)
     FROM Orders
     GROUP BY odate
     ORDER BY 2 DESC;
```

Упражнение 5.

```
1. SELECT onum, cname
     FROM Orders, Customers
     WHERE Customers.cnum = Orders.cnum;

2. SELECT onum, cname, sname
     FROM Orders, Customers, Salespeople
     WHERE Customers.cnum = Orders.cnum
       AND Salespeople.snum = Orders.snum;

3. SELECT cname, sname, comm
     FROM Salespeople, Customers
     WHERE Salespeople.snum = Customers.snum
       AND comm > .12;

4. SELECT onum, comm * amt
     FROM Salespeople, Orders, Customers
     WHERE rating > 100
       AND Orders.cnum = Customers.cnum
       AND Orders.snum = Salespeople.snum;
```

```
5. SELECT first.sname, second.sname
     FROM Salespeople first, Salespeople second
     WHERE first.city = second.city
       AND first.sname < second.sname;

6. SELECT cname, first.onum, second.onum
     FROM Orders first, Orders second, Customers
     WHERE first.cnum = second.cnum
       AND first.cnum = Customers.cnum
       AND first.onum < second.onum;

7. SELECT a.cname, a.city
     FROM Customers a, Customers b
     WHERE a.rating = b.rating
       AND b.cnum = 2001;
```

Упражнение 6

```
1. SELECT *
     FROM Orders
     WHERE cnum =
     (SELECT cnum
        FROM Customers
        WHERE cname = 'Cisneros');
```

или

```
   SELECT *
     FROM Orders
     WHERE cnum IN
      (SELECT cnum
         FROM Customers
         WHERE cname = 'Cisneros');

2. SELECT DISTINCT cname, rating
     FROM Customers, Orders
     WHERE amt >
      (SELECT AVG (amt)
         FROM Orders)
      AND Orders.cnum = Customers.cnum;
```

3. SELECT snum, SUM (amt)
FROM Orders
GROUP BY snum
HAVING SUM (amt) >
(SELECT MAX (amt)
FROM Orders);

4. SELECT cnum, cname
FROM Customers outer
WHERE rating =
(SELECT MAX (rating)
FROM Customers inner
WHERE inner.city = outer.city);

5.а. Решение с помощью связанного подзапроса:
SELECT snum, sname
FROM Salespeople main
WHERE city IN
(SELECT city
FROM Customers inner
WHERE inner.snum < > main.snum);

5.б. Решение с помощью соединения:
SELECT DISTINCT first.snum, sname
FROM Salespeople first, Customers second
WHERE first.city = second.city
AND first.snum < > second.snum;

Приложение 4

Сокращения

Сокращение	Значение
CASE-средство	Computer-Aided Software Engineering – программные система, поддерживающая автоматизированное проектирование программ
ER-модель	«Entity-Relationship» модель или модели «сущность-связь»
RPW	Rational Process Workbench
RUP	Rational Unified Process
SQL	Structured Query Language
UML	Unified Modeling Language
БД	База Данных
ГОСТ	Государственный Стандарт
ИС	Информационная Система
ПО	Программное обеспечение
ПС	Программное средство
ПЭВМ	Персональная ЭВМ, персональный компьютер
СММ	Capability Maturity Model
СУБД	Система управления Базой Данных
ТЗ	Техническое Задание

Содержание

Кумсков Михаил Иванович

Базы данных и процессы их создания.
Введение.

М:Изд-во механико-математического факультета МГУ, 2003.

Подписано в печать 9.12.2004
Формат 60 x 90 1/6 Объем 7,5 п.л.
Заказ 17 Тираж 300 экз.

Издательство ЦПИ при механико-математическом факультете МГУ, г. Москва , Воробьевы горы.
Лицензия на издательскую деятельность ИД № 04059 от 30.02.200

Отпечатано на типографском оборудовании механико-математического факультета