

Алтайский
государственный технический
университет им. И.И. Ползунова

П.И. Ананьев
М.А. Кайгородова

ОСНОВЫ БАЗ ДАННЫХ

Учебное пособие

Барнаул 2015

Министерство образования и науки
Российской Федерации

Алтайский государственный технический
университет им. И.И. Ползунова

П.И. Ананьев, М.А. Кайгородова
ОСНОВЫ БАЗ ДАННЫХ

Барнаул 2015

УДК 681.3.06(075.8)

Ананьев П.И., Кайгородова М.А. Основы баз данных, 2-е изд.: Учебное пособие/ Алт. госуд. технич. ун-т им. И.И. Ползунова. - Барнаул: 2015.- 189.- ил.

Данное учебное пособие предназначено для изучения дисциплины “Базы данных”. Здесь рассмотрены вопросы проектирования баз данных, различные подходы к их разработке и реализации.

Рецензент: Е.Н. Крючкова - профессор
кафедры Прикладной математики АлтГТУ

1 ОСНОВНЫЕ ПОНЯТИЯ

1.1 ИНФОРМАЦИЯ КАК РЕСУРС

Информационное обеспечение является составной частью более широкого понятия информационных процессов. В нормативно-правовой трактовке **информационные процессы** определяются как *«процессы создания, сбора, обработки, накопления, хранения, поиска, распространения и потребления информации»* и охватывают тем самым все сферы человеческой деятельности.

Информационное обеспечение чаще всего соотносится с организационно-управленческой и производственно-технологической сферой. Поэтому под **информационным обеспечением** будем понимать *совокупность процессов сбора, обработки, хранения, анализа и выдачи информации, необходимой для обеспечения управленческой деятельности и технологических процессов.*

Основополагающим в определении информационного обеспечения является понятие информации. Термин *информация* происходит от латинского *informatio* — *разъяснение, изложение*. Информацию в вероятностно-статистическом подходе стали трактовать как *уменьшение степени неопределенности знания о каком-либо объекте, системе, процессе или явлении, или изменение неопределенности состояния самого объекта, системы, явления, процесса*. Такую трактовку по имени ее автора, американского математика К. Э. Шеннона еще называют информацией *по Шеннону*.

Известна также и широко используется философская, или точнее говоря, общенаучная трактовка понятия информации как *изменение объема и структуры знания воспринимающей системы*. При этом под воспринимающей системой понимается не только собственно сам человек или его производные (коллектив, общество), но и, вообще говоря, любая система, например биологическая клетка, воспринимающая при рождении генетическую информацию.

Существует еще и *нормативно-правовая* трактовка понятия информации, которая используется в законодательных актах, регламентирующих информационные процессы и технологии. Так, в частности, в Федеральном законе от 27 июля 2006 г. N 149-ФЗ "Об информации, информационных технологиях и о защите информации" дается следующее определение термина «информация» — *сведения (сообщения, данные) независимо от формы их представления*.

Как представляется, в контексте рассмотрения содержания информационно-аналитической сферы наиболее подходящим является объединение общенаучной и нормативно-правовой трактовки понятия информации. Поэтому в дальнейшем **информацию** будем понимать как *изменение объема и структуры знания о некоторой*

предметной области (лица, предметы, факты, события, явления, процессы) воспринимающей системой (человек, организационная структура, автоматизированная информационная система) независимо от формы и способа представления знания.

При рассмотрении понятия информационного обеспечения в контексте обработки информации важное значение имеет понятие данных. От информации **данные** отличаются конкретной формой представления и являются некоторым ее подмножеством, определяемым целями и задачами сбора и обработки информации. Поэтому определим данные как *информацию, отражающую определенное состояние некоторой предметной области в конкретной форме представления и содержащую лишь наиболее существенные с точки зрения целей и задач сбора и обработки информации элементы образа отражаемого фрагмента действительности.* Можно выделить *неструктурированную и структурированную форму представления данных.*

В качестве примера **неструктурированной** формы можно привести:

- связный текст (т. е. документ на естественном языке — на литературном, официально-деловом и т. д.);
- графические данные в виде фотографий, картинок и прочих неструктурированных изображений.

Примерами **структурированной** формы данных являются:

- анкеты;
- таблицы;
- графические данные в виде чертежей, схем, диаграмм.

В плане оперирования с информацией в процессах ее создания (порождения), сбора, выдачи и потребления важное значение имеет понятие *документированной информации.*

Под *документированием* информации в широком смысле слова можно понимать выделение единичной смысловой части информации (данных) но некоторой предметной области в общей ее массе, обособление этой части с приданием ему самостоятельной роли (имя, статус, реквизиты и т. п.). *Процесс документирования превращает информацию в информационные ресурсы.*

Таким образом, документирование информации подводит к одному из самых фундаментальных понятии в сфере информационного обеспечения — информационным системам. Так же как и для понятий информации и документа, понятие информационной системы многогранно и имеет несколько определений и подходов. В нормативно-правовом смысле **информационная система** определяется как *«организационно упорядоченная совокупность документов (массивов документов) и информационных технологии, в том числе и с использованием средств вычислительной техники и связи,*

реализующих информационные процессы».

Опыт, практика создания и использования автоматизированных информационных систем в различных сферах деятельности позволяет дать более широкое и универсальное определение, которое полнее отражает все аспекты их сущности.

Под информационной системой понимается организованная совокупность программно-технических и других вспомогательных средств, технологических процессов и функционально-определенных групп работников, обеспечивающих сбор, представление и накопление информационных ресурсов в определенной предметной области, поиск и выдачу сведений, необходимых для удовлетворения информационных потребностей установленного контингента пользователей - абонентов системы.

Информационные системы, в которых представление, хранение и обработка информации осуществляются с помощью вычислительной техники, называются автоматизированными, или сокращенно АИС.

По характеру представления и логической организации хранимой информации АИС разделяются на *фактографические, документальные и геоинформационные.*

Фактографические АИС накапливают и хранят данные в виде множества экземпляров одного или нескольких типов структурных элементов (*информационных объектов*). Каждый из таких экземпляров структурных элементов или некоторая их совокупность отражают сведения по какому-либо факту, событию и т. д., отделенному (вычлененному) от всех прочих сведений и фактов. Структура каждого типа информационного объекта состоит из конечного набора реквизитов, отражающих основные аспекты и характеристики сведений для объектов данной предметной области. К примеру, фактографическая АИС, накапливающая сведения по лицам, каждому конкретному лицу в базе данных ставит в соответствие запись, состоящую из определенного набора таких реквизитов, как фамилия, имя, отчество, год рождения, место работы, образование и т. д. Комплектование информационной базы в фактографических АИС включает, как правило, обязательный процесс структуризации входной информации из документального источника. Структуризация при этом осуществляется через определение (выделение, вычленение) экземпляров информационных объектов определенного типа, информация о которых имеется в документе, и заполнение их реквизитов.

В **документальных** АИС единичным элементом информации является нерасчлененный на более мелкие элементы документ и информация при вводе (входной документ), как правило, не структурируется, или структурируется в ограниченном виде. Для вводимого документа могут устанавливаться некоторые формализованные позиции — дата изготовления, исполнитель, тематика и т. д.

В *геоинформационных* АИС данные организованы в виде отдельных информационных объектов (с определенным набором реквизитов), привязанных к общей электронной топографической основе (электронной карте).

Информационным ядром подсистемы представления и обработки информации фактографических АИС, или, говоря иначе, внутренним носителем знаний о предметной области является **база данных** (БД). Понятие базы данных является центральным в сфере технологий автоматизированных информационных систем.

Прежде чем подробнее рассмотреть понятие «база данных» необходимо остановиться на файловых системах, которые предшествовали базам данных (БД).

1.2 НЕДОСТАТКИ ТРАДИЦИОННЫХ ФАЙЛОВЫХ СИСТЕМ

Несмотря на появление файлов с произвольным доступом, быстро стало очевидным, что файловые системы любого типа обладают некоторыми врожденными недостатками:

- * Избыточность данных
- * Слабый контроль данных
- * Недостаточные возможности управления данными
- * Большие затраты труда программиста

Избыточность данных. Главная трудность состоит в том, что многие приложения используют свои собственные файлы данных. Таким образом, некоторые единицы данных повторяются в разных приложениях. Например, в банке одно и то же имя клиента встречается в файлах, содержащих сведения о текущих счетах, сберегательных счетах и ссудах. Более того, хотя это одно и то же имя клиента, соответствующие поля в разных файлах могут называться по-разному. Так, поле CNAME файла текущих счетов превращается в SNAME файла сберегательных счетов и в INAME файла ссуд. Одно и то же поле в разных файлах может, кроме того, иметь разную длину. Например, поле CNAME может содержать до 20 символов, а поля SNAME и INAME допускают максимальную длину 15 символов. Следствием такой избыточности данных являются лишние затраты на поддержание и хранение данных. Избыточность данных также порождает риск противоречий между разными версиями общих данных.

Предположим, что клиент изменил имя. Изменение могло быть сразу внесено в файл текущих счетов, через неделю — в файл сберегательных счетов, а в файл ссуд изменение *могло* оказаться внесенным неверно. По прошествии некоторого времени подобные расхождения могут существенно снизить качество информации, содержащейся в файлах данных. Такая несогласованность данных может также отразиться на точности отчетов. Информационные системы, использующие базы данных, позволяют избавиться от подобной избыточности данных, поскольку все приложения используют один и тот же

набор данных. Существенная информация, например, имя или адрес клиента, будет записываться в базе данных всего один раз. Таким образом, мы сможем изменять адрес или имя клиента один раз, будучи при этом уверены, что все приложения будут пользоваться согласованными данными.

Слабый контроль данных. В файловых системах отсутствует централизованный контроль на уровне элементов данных. Весьма часто один и тот же элемент данных имеет несколько имен в зависимости от того, в какие файлы он входит.

На более фундаментальном уровне всегда существует вероятность того, что разные отделы компании пользуются терминологией, не согласованной с остальными. Например, банк может вкладывать в термин *счет* один смысл применительно к сбережениям и совсем другой применительно к ссудам. И наоборот, разные слова могут иметь одинаковые значения. Страховая компания может говорить о *владельце полиса* или о *клиенте*, вкладывая в эти два термина один и тот же смысл. Система управления базами данных осуществляет централизованный контроль данных и помогает избежать недоразумений, порожденных омонимами и синонимами.

Омонимы. Разные значения одного и того же термина.

Синонимы. Термины, имеющие одно и то же значение.

Недостаточные возможности управления данными. Индексно-последовательные файлы позволяют обращаться к определенной записи по ключу, например, по идентификатору товара. Например, если мы знаем идентификатор настольной лампы, мы можем напрямую извлечь из файла PRODUCT относящуюся к ней запись. Этого достаточно до тех пор, пока нам нужна отдельная запись.

Однако предположим, что нам нужен целый ряд связанных между собой записей. Например, мы хотим найти все продажи клиенту Петрову. Допустим, что мы также хотим узнать общее число таких продаж, среднюю цену или же список товаров, купленных клиентом и кто является изготовителем этих товаров. Такую информацию будет трудно, если не невозможно, извлечь из файловой системы, поскольку файловые системы не позволяют устанавливать связь между данными разных файлов. Системы управления базами данных были специально разработаны для того, чтобы упростить связывание данных из разных файлов.

Большие затраты труда программиста. Новая прикладная программа часто требовала совершенно нового набора файлов. Даже если существующий уже файл содержал некоторые нужные данные, приложению часто требовался еще какой-либо набор элементов данных. В результате программисту приходилось перекодировать определения нужных элементов данных из существующих файлов, а также определять новые элементы данных. Таким образом, в файловой системе существовала жесткая

зависимость между программами и данными.

Что еще более важно, манипулирование данными в файлово-ориентированных языках (таких, как Кобол), было слишком сложным для создания больших приложений. Это означало, что затраты труда программиста как на создание приложения, так и на поддержание его работы, были весьма значительны.

Базы данных позволили разделить программы и данные, так что программа может быть в некотором смысле независима от деталей определения данных. Предоставляя доступ к множеству общих данных и поддерживая мощные языки управления данными, информационные системы, использующие базы данных, позволяют значительно сократить объем работ по созданию и поддержке программного обеспечения.

1.3 ИНФОРМАЦИОННЫЕ СИСТЕМЫ, ИСПОЛЬЗУЮЩИЕ БАЗЫ ДАННЫХ

Информационные системы, использующие базы данных, позволили преодолеть ограничения файловых систем. Поддерживая целостную, централизованную структуру данных, информационные системы, использующие базы данных, позволили избавиться от проблем избыточности и слабого контроля данных. Доступ к централизованной базе данных имеет вся компания и если, например, необходимо внести изменение в имя клиента, это изменение будет известно всем пользователям. Данные контролируются посредством словаря/каталога данных (data dictionary/directory, DD/D), которым в свою очередь управляет группа сотрудников компании, называемых администраторами базы данных (АБД). Новые методы обращения к данным сильно упростили процесс связывания элементов данных, что в свою очередь привело к расширению возможностей работы с данными. Все эти характеристики систем управления базами данных упрощают процесс программирования и уменьшают необходимость программной поддержки.

1.4 БАЗА ДАННЫХ

Было сделано немало попыток дать определение понятию *база данных*. В справочной литературе по информатике приводится следующее определение базы данных — *«совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ»*. Нормативно-правовая трактовка понятия базы данных представлена в законе «О правовой охране программ для ЭВМ и баз данных», согласно которому *«Базой данных является представленная в объективной форме совокупность самостоятельных материалов (статей, расчетов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной*

вычислительной машины (ЭВМ).». Для наших целей вполне подойдет следующее: *база данных — это совокупность данных, организованная с определенной целью.* Здесь слово *организованная* означает, что указанная совокупность включает данные, которые *сохраняются*, имеют определенный формат (*отформатированы*), к ним может быть обеспечен доступ (*доступны*), и они могут быть представлены потребителю информации в приемлемом виде (*репрезентативны*). Упоминанием в определении *цели* подчеркивается, что состав данных должен соответствовать некоторой задаче (или классу задач): с одной стороны, в базе не должны содержаться данные, не имеющие отношения к задаче, а с другой — должна содержаться вся информация, касающаяся задачи. Прекрасным и понятным примером базы данных может служить телефонный справочник. Он содержит данные, имеющие прямое отношение к цели его существования (в данном случае — имена абонентов), которые открывают доступ к номеру телефона. В справочнике отсутствует избыточная информация, скажем цвет аппарата абонента. То есть в этой простейшей базе содержится информация, непосредственно связанная с целью ее создания. Наиболее часто цель существования базы данных связана с деловой сферой, но в ней может содержаться и научная, военная и прочая информация, в ряде случаев не имеющая отношения к сфере бизнеса. Существуют деловые, научные, военные и прочие базы данных, причем список возможных областей человеческой деятельности, в которых находят применение данные, организованные таким образом, может быть продолжен практически неограниченно. Кроме разделения баз данных, основанного на областях их применения, можно предложить и другой подход, взяв за основу форматы включенной в них информации. Помимо текстовой и числовой это могут быть изображения, различного рода графики и схемы, аудио- и видеоинформация. В последнее время все чаще речь идет о *составных документах*, которые представляют собой связную совокупность различных видов информации.

Когда речь идет о базах данных вообще и проектировании баз данных в частности, общепринято ссылаться на цель, которой подчинена ее организация и функционирование, независимо от того, в какой именно области человеческой деятельности конкретная база используется — будь то космонавтика, биология или что-либо еще. Более того, в действительности оказывается, что именно целевая функция определяет специфику той или иной базы данных.

В прежние времена разработчики программ для автоматизированной обработки данных часто встречались с ситуацией, когда нужно было сохранять данные между прогонами программы. Эта процедура получила наименование *сохранения с восстановлением*. То есть речь идет о том, что данные должны быть сохранены в том виде, в каком они оказались на момент завершения одного прогона программы, а затем в

начале следующего прогона восстановлены, чтобы программа продолжала их обрабатывать так, будто перерыва не было. Именно эта потребность и привела со временем к появлению баз данных, о которых мы сейчас говорим. Вторая объективная потребность, которая дала дополнительный толчок этому процессу, — необходимость в накоплении и сохранении данных. Хотя в большинстве случаев для этого оказывается вполне достаточно иерархической структуры файлов и каталогов, а также возможностей системы их обслуживания, включая индексацию, базы данных открывают в этой области такие перспективы, которые недоступны системам управления файлами.

Современные базы данных как правило служат для сохранения и обработки информации, касающейся различных аспектов деятельности как подразделений (достаточно крупных или мелких, состоящих всего из нескольких человек), так и организаций. Таким образом, мы можем использовать термин *база данных уровня предприятия*, когда речь идет об информации, охватывающей деятельность предприятия в целом, или *база данных уровня подразделения*, если соответствующая информация не выходит за рамки его деятельности, или *база данных уровня рабочей группы* для самых мелких коллективов. Наиболее распространены базы данных двух последних уровней — уровня подразделения и уровня рабочей группы.

Иногда можно встретить базы данных уровня предприятия, например ведомости на выдачу зарплаты или штатное расписание, но при этом весьма небольшого объема. Фактически, когда базы данных отдельных подразделений сливаются (*интегрируются*) в одну большую базу, есть смысл говорить о *банке данных*. Базы данных меньшего объема, которые служат источником информации для баз более крупных, в последнее время довольно часто называются *операционными базами данных*. Однако в этом нет ничего нового. То, что сейчас называется операционной базой, прежде было известно под именем *производственной базы данных*, поскольку в рамках такой базы *производились* (подготавливались) данные, которые затем включались в более крупную базу. В контексте формирования банка данных можно встретить упоминание о производственной базе как об операционной, а иногда — как о *накопителе данных*. С развитием сети Internet базы данных и банки данных все чаще становятся конечными пунктами маршрута поиска информации, который начинается от Web-браузера.

Когда база данных рабочей группы интегрируется для обслуживания потребностей более высокого уровня — уровня подразделения, то, что получается в результате, часто называются *кладовой данных*. Фактически это является банком данных уровня подразделения. У вас не должен вызвать удивление тот факт, что термин банк данных, как и термин база данных, может иметь множество толкований. Тем не менее, когда несколько небольших по масштабам баз данных объединяются в одну большую, которая

может обслуживать более широкий круг запросов, чему соответствует более высокий уровень организационной структуры, результирующая объединенная база может рассматриваться как хранилище информации при том, однако, условии, что в ней оказываются данные, накопленные за достаточно длительный период времени, которые могут служить основой для принятия определенных управленческих решений, а сформировавшаяся в результате информационная среда позволяет обобщать данные и обеспечивает защиту первичных данных от искажения или модификации. Такое хранилище информации служит как бы резервуаром, в который сливаются потоки от множества питающих его продукционных баз.

Возможен и другой вариант: БД просто увеличивается в объеме, поскольку в ней оседают данные за довольно продолжительное время, и в результате она становится чем-то вроде исторического архива. Примером может служить накопление в электронном виде результатов переписей. Еще одной причиной чисто количественного роста объема БД может служить сам тип накапливаемой информации (например, если БД включает изображения) или частота ее поступления. Примером здесь может служить накопление данных спутниковой телеметрии. За такими БД закрепилось название *сверхбольшие БД*.

Совершенно естественно, что со временем, по мере повышения плотности хранения информации и удешевления необходимых для этого носителей, совершенствования методов и средств распараллеливания обработки на небольших машинах, развития технологии RAID и программных комплексов для работы с большими массивами данных, претерпевали изменения и представления о масштабах границы, отделяющей сверхбольшие БД от прочих.

На сегодняшний день эта граница лежит где-то в пределах 100 Гбайт. БД большего объема уже может претендовать на то, чтобы считаться сверхбольшей. А всего-то несколько лет назад казалось, что мыслимый предел наращивания объема — это 10 Гбайт.

Система базы данных (банк данных) состоит из базы данных; программного обеспечения общего назначения, называемого **системой управления базой данных (СУБД)**, служащего для управления базой данных; соответствующего оборудования и людей. СУБД служит средством, с помощью которого прикладные программы или пользователи работают с данными базы.

1.5 ПРЕИМУЩЕСТВА И НЕДОСТАТКИ СУБД

СУБД обладают как многообещающими потенциальными преимуществами, так и недостатками.

1.5.1 Преимущества

Преимущества систем управления базами данных перечислены в табл. 1.1.

Таблица 1.1. Преимущества систем управления базами данных

Преимущество
Контроль за избыточностью данных
Непротиворечивость данных
Больше полезной информации при том же объеме хранимых данных
Совместное использование данных
Поддержка целостности данных
Повышенная безопасность
Применение стандартов
Повышение эффективности с ростом масштабов системы
Возможность нахождения компромисса при противоречивых требованиях
Повышение доступности данных и их готовности к работе
Улучшение показателей производительности
Упрощение сопровождения системы за счет независимости от данных
Улучшенное управление параллельной работой
Развитые службы резервного копирования и восстановления

1.5.1.1 Контроль за избыточностью данных

Как уже говорилось, традиционные файловые системы неэкономно расходуют внешнюю память, сохраняя одни и те же данные в нескольких файлах. При использовании базы данных, наоборот, предпринимается попытка исключить избыточность данных за счет интеграции файлов, что позволяет исключить необходимость хранения нескольких копий одного и того же элемента информации. Однако полностью избыточность информации в базах данных не исключается, а лишь ограничивается ее степень. В одних случаях ключевые элементы данных необходимо дублировать для моделирования связей, а в других случаях некоторые данные требуется дублировать из соображений повышения производительности системы.

1.5.1.2 Непротиворечивость данных

Устранение избыточности данных или контроль над ней позволяет уменьшить риск возникновения противоречивых состояний. Если элемент данных хранится в базе только в одном экземпляре, то для изменения его значения потребуется выполнить только одну операцию обновления, причем новое значение станет доступным сразу всем пользователям базы данных. А если этот элемент данных с ведома системы хранится в базе данных в нескольких экземплярах, то такая система сможет следить за тем, чтобы копии не противоречили друг другу. К сожалению, во многих современных СУБД такой способ обеспечения непротиворечивости данных не поддерживается автоматически.

1.5.1.3 Больше полезной информации при том же объеме хранимых данных

Благодаря интеграции рабочих данных организаций на основе тех же данных можно получать дополнительную информацию. Например, в файловой системе

сотрудникам отдела контрактов неизвестны владельцы сданных в аренду объектов. Аналогично, сотрудники отдела реализации не имеют полных сведений о договорах аренды. При интеграции этих файлов в общей базе сотрудники отдела контрактов получают доступ к сведениям о владельцах, а сотрудники отдела реализации — к сведениям о договорах аренды. Теперь на основе тех же данных пользователи смогут получать больше информации.

1.5.1.4 Совместное использование данных

Файлы обычно принадлежат отдельным лицам или целым отделам, которые используют их в своей работе. В то же время база данных принадлежит всей организации в целом и может совместно использоваться всеми зарегистрированными пользователями. При такой организации работы большее количество пользователей может работать с большим объемом данных. Более того, при этом можно создавать новые приложения на основе уже существующей в базе данных информации и добавлять в нее только те данные, которые в настоящий момент еще не хранятся в ней, а не определять заново требования ко всем данным, необходимым новому приложению. Новые приложения могут также использовать такие предоставляемые типичными СУБД функциональные возможности, как определение структур данных и управление доступом к данным, организация параллельной обработки и обеспечение средств копирования/восстановления, исключив необходимость реализации этих функций со своей стороны.

1.5.1.5 Поддержка целостности данных

Целостность базы данных означает корректность и непротиворечивость хранимых в ней данных. Целостность обычно описывается с помощью *ограничений*, т.е. правил поддержки непротиворечивости, которые не должны нарушаться в базе данных. Ограничения можно применять к элементам данных внутри одной записи или к связям между записями. Например, ограничение целостности может гласить, что зарплата сотрудника не должна превышать 40 000 фунтов стерлингов в год или же что в записи с данными о сотруднике номер отделения, в котором он работает, должен соответствовать реально существующему отделению компании. Таким образом, интеграция данных позволяет АБД задавать требования по поддержке целостности данных, а СУБД применять их.

1.5.1.6 Повышенная безопасность

Безопасность базы данных заключается в защите базы данных от несанкционированного доступа со стороны пользователей. Без привлечения соответствующих мер безопасности интегрированные данные становятся более уязвимыми, чем данные в файловой системе. Однако интеграция позволяет АБД определить требуемую систему

безопасности базы данных, а СУБД привести ее в действие. Система обеспечения безопасности может быть выражена в форме имен и паролей для идентификации пользователей, которые зарегистрированы в этой базе данных. Доступ к данным со стороны зарегистрированного пользователя может быть ограничен только некоторыми операциями (извлечением, вставкой, обновлением и удалением). Например, АБД может быть предоставлено право доступа ко всем данным в базе данных, менеджеру отделения компании — ко всем данным, которые относятся к его отделению, а инспектору отдела реализации — лишь ко всем данным о недвижимости, в результате чего он не будет иметь доступа к конфиденциальным данным, таким как, зарплата сотрудников.

1.5.1.7 Применение стандартов

Интеграция позволяет АБД определять и применять необходимые стандарты. Например, стандарты отдела и организации, государственные и международные стандарты могут регламентировать формат данных при обмене ими между системами, соглашения об именах, форму представления документации, процедуры обновления и правила доступа.

1.5.1.8 Повышение эффективности с увеличением масштабов системы

Комбинируя все рабочие данные организации в одной базе данных и создавая набор приложений, которые работают с одним источником данных, можно добиться существенной экономии средств. В этом случае бюджет, который обычно выделялся каждому отделу для разработки и поддержки их собственных файловых систем, можно объединить с бюджетами других отделов (с более низкой общей стоимостью), что позволит добиться повышения эффективности при росте масштабов производства. Теперь объединенный бюджет можно будет использовать для приобретения оборудования в той конфигурации, которая в большей степени отвечает потребностям организации. Например, она может состоять из одного мощного компьютера или сети из небольших компьютеров.

1.5.1.9 Возможность нахождения компромисса для противоречивых требований

Потребности одних пользователей или отделов могут противоречить потребностям других пользователей. Но поскольку база данных контролируется АБД, он может принимать решения о проектировании и способе использования базы данных, при которых имеющиеся ресурсы всей организации в целом будут использоваться наилучшим образом. Эти решения обеспечивают оптимальную производительность для самых важных приложений, причем чаще всего за счет менее критичных.

1.5.1.10 Повышение доступности данных и их готовности к работе

Данные, которые пересекают границы отделов, в результате интеграции становятся непосредственно доступными конечным пользователям. Потенциально это повышает

функциональность системы, что, например, может быть использовано для более качественного обслуживания конечных пользователей или клиентов организации. Во многих СУБД предусмотрены языки запросов или инструменты для создания отчетов, которые позволяют пользователям вводить не предусмотренные заранее запросы и почти немедленно получать требуемую информацию на своих терминалах, не прибегая к помощи программиста, который для извлечения этой информации из базы данных должен был бы создать специальное программное обеспечение.

1.5.1.11 Улучшение показателей производительности

В СУБД предусмотрено много стандартных функций, которые программист обычно должен самостоятельно реализовать в приложениях для файловых систем. На базовом уровне СУБД обеспечивает все низкоуровневые процедуры работы с файлами, которую обычно выполняют приложения. Наличие этих процедур позволяет программисту сконцентрироваться на разработке более специальных, необходимых пользователям функций, не заботясь о подробностях их воплощения на более низком уровне. Во многих СУБД предусмотрена также среда разработки четвертого поколения с инструментами, упрощающими создание приложений баз данных. Результатом является повышение производительности работы программистов и сокращение времени разработки новых приложений (с соответствующей экономией средств).

1.5.1.12 Упрощение сопровождения системы за счет независимости от данных

В файловых системах описания данных и логика доступа к данным встроены в каждое приложение, поэтому программы становятся зависимыми от данных. Для изменения структуры данных (например, для увеличения длины поля с адресом с 40 символов до 41 символа) или для изменения способа хранения данных на диске может потребоваться существенно преобразовать все программы, на которые эти изменения способны оказать влияние. В СУБД подход иной: описания данных отделены от приложений, а потому приложения защищены от изменений в описаниях данных. Эта особенность называется *независимостью от данных*. Наличие независимости программ от данных значительно упрощает обслуживание и сопровождение приложений, работающих с базой данных.

1.5.1.13 Улучшенное управление параллельной работой

В некоторых файловых системах при одновременном доступе к одному и тому же файлу двух пользователей может возникнуть конфликт двух запросов, результатом которого будет потеря информации или утрата ее целостности. В свою очередь, во многих СУБД предусмотрена возможность параллельного доступа к базе данных и гарантируется отсутствие подобных проблем.

1.5.1.14 Развитые службы резервного копирования и восстановления

Ответственность за обеспечение защиты данных от сбоев аппаратного и программного обеспечения в файловых системах возлагается на пользователя. Так, может потребоваться каждую ночь выполнять резервное копирование данных. При этом в случае сбоя может быть восстановлена резервная копия, но результаты работы, выполненной после резервного копирования, будут утрачены, и данную работу потребуется выполнить заново. В современных СУБД предусмотрены средства снижения вероятности потерь информации при возникновении различных сбоев.

1.5.2 Недостатки

Недостатки подхода, связанного с применением баз данных, перечислены в табл. 1.2.

Таблица 1.2. Недостатки систем управления базами данных

Недостаток
Сложность
Размер
Стоимость СУБД
Дополнительные затраты на аппаратное обеспечение
Затраты на преобразование
Производительность
Более серьезные последствия при выходе системы из строя

1.5.2.1 Сложность

Обеспечение функциональности, которой должна обладать каждая хорошая СУБД, сопровождается значительным усложнением программного обеспечения СУБД. Чтобы воспользоваться всеми преимуществами СУБД, проектировщики и разработчики баз данных, администраторы данных и администраторы баз данных, а также конечные пользователи должны хорошо понимать функциональные возможности СУБД. Непонимание принципов работы системы может привести к неудачным результатам проектирования, что будет иметь самые серьезные последствия для всей организации.

1.5.2.2 Размер

Сложность и широта функциональных возможностей приводит к тому, что СУБД становится чрезвычайно сложным программным продуктом, который может потребовать много места на диске и нуждаться в большом объеме оперативной памяти для эффективной работы.

1.5.2.3 Стоимость СУБД

В зависимости от имеющейся вычислительной среды и требуемых функциональных возможностей стоимость СУБД может изменяться в очень широких пределах. Например, однопользовательская СУБД для персонального компьютера может стоить около 100 долларов. Однако большая многопользовательская СУБД для мэйнфрейма,

обслуживающая сотни пользователей, может быть чрезвычайно дорогостоящей: от 100 000 до 1 000 000 долларов. Кроме того, следует учесть ежегодные расходы на сопровождение системы, которые составляют некоторый процент от ее общей стоимости.

1.5.2.4 Дополнительные затраты на аппаратное обеспечение

Для удовлетворения требований, предъявляемых к дисковым накопителям со стороны СУБД и базы данных, может понадобиться приобрести дополнительные устройства хранения информации. Более того, для достижения требуемой производительности может понадобиться более мощный компьютер, который, возможно, будет работать только с СУБД. Приобретение другого дополнительного аппаратного обеспечения приведет к дальнейшему росту затрат.

1.5.2.5 Затраты на преобразование

В некоторых ситуациях стоимость СУБД и дополнительного аппаратного обеспечения может оказаться несущественной по сравнению со стоимостью преобразования существующих приложений для работы с новой СУБД и новым аппаратным обеспечением. Эти затраты включают также стоимость подготовки персонала для работы с новой системой, а также оплату услуг специалистов, которые будут оказывать помощь в преобразовании и запуске новой системы. Все это является одной из основных причин, по которой некоторые организации остаются сторонниками прежних систем и не хотят переходить к более современным технологиям управления базами данных. Термин *традиционная система* иногда используется для обозначения устаревших и, как правило, не самых лучших систем.

1.5.2.6 Производительность

Обычно файловая система создается для некоторых" специализированных приложений, например для оформления счетов, а потому ее производительность может быть весьма высока. Однако СУБД предназначены для решения более общих задач и обслуживания сразу нескольких приложений, а не какого-то одного из них. В результате многие приложения в новой среде будут работать не так быстро, как прежде.

1.5.2.7 Более серьезные последствия при выходе системы из строя

Централизация ресурсов повышает уязвимость системы. Поскольку работа всех пользователей и приложений зависит от готовности к работе СУБД, выход из строя одного из ее компонентов может привести к полному прекращению всей работы организации.

1.6 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Информационная система, использующая базу данных, включает в себя два вида программного обеспечения:

- Программное обеспечение общего назначения для поддержания базы данных,

обычно называемое системой управления базой данных (СУБД).

- Прикладное программное обеспечение, которое использует средства СУБД для выполнения конкретных деловых задач, таких, как выставление счетов или анализ продаж.

Прикладное программное обеспечение обычно создается сотрудниками компании для решения конкретных задач компании. Оно может быть написано на стандартном языке программирования типа Си или же на языке (обычно называемом языком четвертого поколения), входящем в комплект системы управления базой данных. Прикладные программы используют средства СУБД для обращения к данным и их обработки, создавая отчеты или документы, необходимые для работы компании.

1.6.1 Понятие о СУБД

Система управления базой данных (СУБД) определяется как «совокупность, программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами».

В настоящее время развитие СУБД как специального вида программного обеспечения приводит к более широким функциям СУБД. Ввиду этого в расширенном толковании СУБД можно определить как *программный комплекс, обеспечивающий функционирование БД. Она играет роль кладовщика при данных, отвечает за их сохранность, безопасность, целостность, взаимное соответствие и обеспечивает доступ пользователей к данным.* В ведении СУБД находится *словарь данных*, который иногда называют *каталогом системы*. В этом словаре содержатся сведения обо всем, что хранится в БД, — наименованиях, структуре, размещении и типах. Сведения такого характера представляют собой *метаданные*, т.е. *данные о данных*. Весь жизненный путь некоторого фрагмента данных — от создания до уничтожения — отражается в словаре данных так же, как и информация о его логической и физической структуре. Администратор БД должен поддерживать тесную связь со словарем данных, который будет служить ему все время существования базы.

1.6.2 Безопасность данных

Безопасность — главный предмет забот при создании, тестировании и сопровождении БД. Вопрос о том, нужны ли средства обеспечения безопасности, сам по себе некорректен. Можно ставить только вопрос, насколько они должны быть развиты. Как правило, СУБД предлагают несколько специфических механизмов повышения уровня безопасности в дополнение к тем, что уже обеспечиваются операционной системой и системой поддержки сети. Наиболее распространенными из них являются обязательная регистрация и последующее распознавание пользователей до предоставления им доступа к тем или иным разделам данных.

Помимо этого, практически стандартного для СУБД приема, существует и множество других — разделение пользователей на группы с различными привилегиями доступа, поддержка профилей и пр. Все эти средства преследуют одну цель — обеспечить последовательное проведение в жизнь определенной политики.

1.6.3 Целостность данных

Под термином *целостность* понимается взаимная согласованность отдельных фрагментов данных и их корректность. *Согласованность* означает, что все порции данных в БД должны быть единообразно смоделированы и включены в систему. Квалифицировать данные как корректные можно в том случае, если они достоверны, точны и значимы.

Один из способов обеспечения целостности, который принят повсеместно на вооружение в СУБД, — блокирование постороннего доступа к данным в процессе их обработки. Как правило, данные блокируются на уровне страницы БД или строки.

Другой прием, также способствующий сохранению целостности данных, — тиражирование изменений фрагмента данных, который в результате хранится в нескольких местах. И последнее, на что следует обратить внимание, — необходимость держать под неусыпным контролем данные, которые вводятся в систему или редактируются, и следить за их соответствием требованиям спецификаций (например, чтобы данные не выходили за пределы заданного диапазона).

Если следовать рекомендованной и опробованной на практике технологии моделирования и организации данных, СУБД автоматически будет надежно поддерживать целостность данных на протяжении всего жизненного цикла БД. Следует помнить, что данные, не увязанные между собой, — мусор, и только объединенные и взаимно увязанные данные являются информацией. Таким образом, целостность является неременным условием превращения разрозненных данных в нечто, представляющее ценность для потребителей информации.

Обязательным требованием для большинства СУБД является обеспечение многопользовательского доступа к данным. Отсюда vyplывает проблема *согласованности* операций нескольких пользователей с БД. Проблема эта состоит в том, что если два или более пользователей обратятся к одному фрагменту данных в одно и то же время, СУБД должна гарантированно обеспечить сохранение целостности и достоверности данных.

Методы, которые при этом используются, достаточно прозрачны, но тем не менее для их программной реализации требуются определенные усилия. Действительно, предоставить возможность двум пользователям одновременно взглянуть на одни и те же данные, не изменяя их, не сложно.

Но СУБД придется серьезно взяться за дело, если один пользователь изменяет фрагмент данных в то самое время, когда другой их просматривает или более того — также пытается изменить. В этом случае СУБД должна создать несколько копий этого фрагмента данных, согласовать все внесенные в копии изменения и в конце концов занести в БД один откорректированный вариант, удовлетворяющий всех участников операции.

Выше уже упоминалось об одном из аспектов технологии параллельного доступа — блокировке. Вообще говоря, чем более мелкий фрагмент данных может быть адресован механизмом блокировки, тем лучше обеспечивается параллелизм при многопользовательском доступе к БД — т.е. тем больше пользователей могут с ней работать, не мешая друг другу.

В подавляющем большинстве БД наименьшей порцией данных, которую можно вовлечь в процедуру блокировки, является строка. Более крупный фрагмент — блок или страница БД. Блокировка на уровне строки лучше обеспечивает доступ со случайным распределением по всему массиву данных, а блокировка на уровне страницы или блока более продуктивна при достаточно продолжительных операциях с близкими по смыслу данными.

Таким образом, обеспечение параллельного доступа и сохранение целостности — две очень тесно связанные между собой функции СУБД.

1.6.4 Общение пользователя с БД

Совершенно очевидно, что поддержка живого общения пользователя с БД, является одной из важнейших функций СУБД. Каким образом может пользователь обратиться к БД? Посредством *языка доступа (access language)* или *языка запросов (query language)*. В последние годы доминирующее положение среди подобного класса языков занял *SQL — Structured Query Language* (язык структурированных запросов). Особенно это касается наиболее распространенного класса СУБД — систем управления *реляционными БД (RDBMS — Relational Database Management System)*. Обращение пользователя к БД так или иначе осуществляется через СУБД; именно она воспринимает операторы языка запросов — SQL или ему подобного. Администратор БД использует язык запросов для формирования и обслуживания БД, а пользователь — для доступа к данным, их просмотра и модификации.

1.7 КОМПОНЕНТЫ СУБД

СУБД является весьма сложным видом программного обеспечения. Компонентную структуру СУБД практически невозможно обобщить, поскольку она очень сильно различается в разных системах. Однако при изучении систем баз данных полезно представлять себе ее обобщенную структуру в виде набора из нескольких компонентов и

определенных связей между ними. Рассмотрим одну из возможных архитектур СУБД.

СУБД состоит из нескольких программных компонентов (модулей), каждый из которых предназначен для выполнения специфической операции. Некоторые функции СУБД могут поддерживаться используемой операционной системой. Однако в любом случае операционная система предоставляет только базовые службы, и СУБД всегда представляет собой надстройку над ними. Таким образом, при проектировании СУБД следует учитывать особенности интерфейса между создаваемой СУБД и конкретной операционной системой.

Основные программные компоненты среды СУБД представлены на рис. 1.1. На этой схеме также показано, как СУБД взаимодействует с другими программными компонентами, например с такими, как пользовательские запросы и методы доступа (т.е. методы управления файлами, используемые при сохранении и извлечении записей с данными).

На рис. 1.1 показаны следующие программные компоненты среды СУБД.

- **Процессор запросов.** Это основной компонент СУБД, который преобразует запросы в последовательность низкоуровневых команд для диспетчера базы данных.

- **Диспетчер базы данных.** Этот компонент взаимодействует с запущенными пользователями прикладными программами и запросами. Диспетчер базы данных принимает запросы и проверяет внешние и концептуальные схемы для определения тех концептуальных записей, которые необходимы для удовлетворения требований запроса. Затем диспетчер базы данных вызывает диспетчер файлов для выполнения поступившего запроса. Компоненты диспетчера базы данных показаны на рис. 1.2.

- **Диспетчер файлов.** Манипулирует предназначенными для хранения данных файлами и отвечает за распределение доступного дискового пространства. Он создает и поддерживает список структур и индексов, определенных во внутренней схеме. Если используются хешированные файлы, то в его обязанности входит и вызов функций хеширования для генерации адресов записей. Однако диспетчер файлов не управляет физическим вводом и выводом данных непосредственно, а лишь передает запросы соответствующим методам доступа, которые считывают данные в системные буферы или записывают их оттуда на диск (или в кэш).

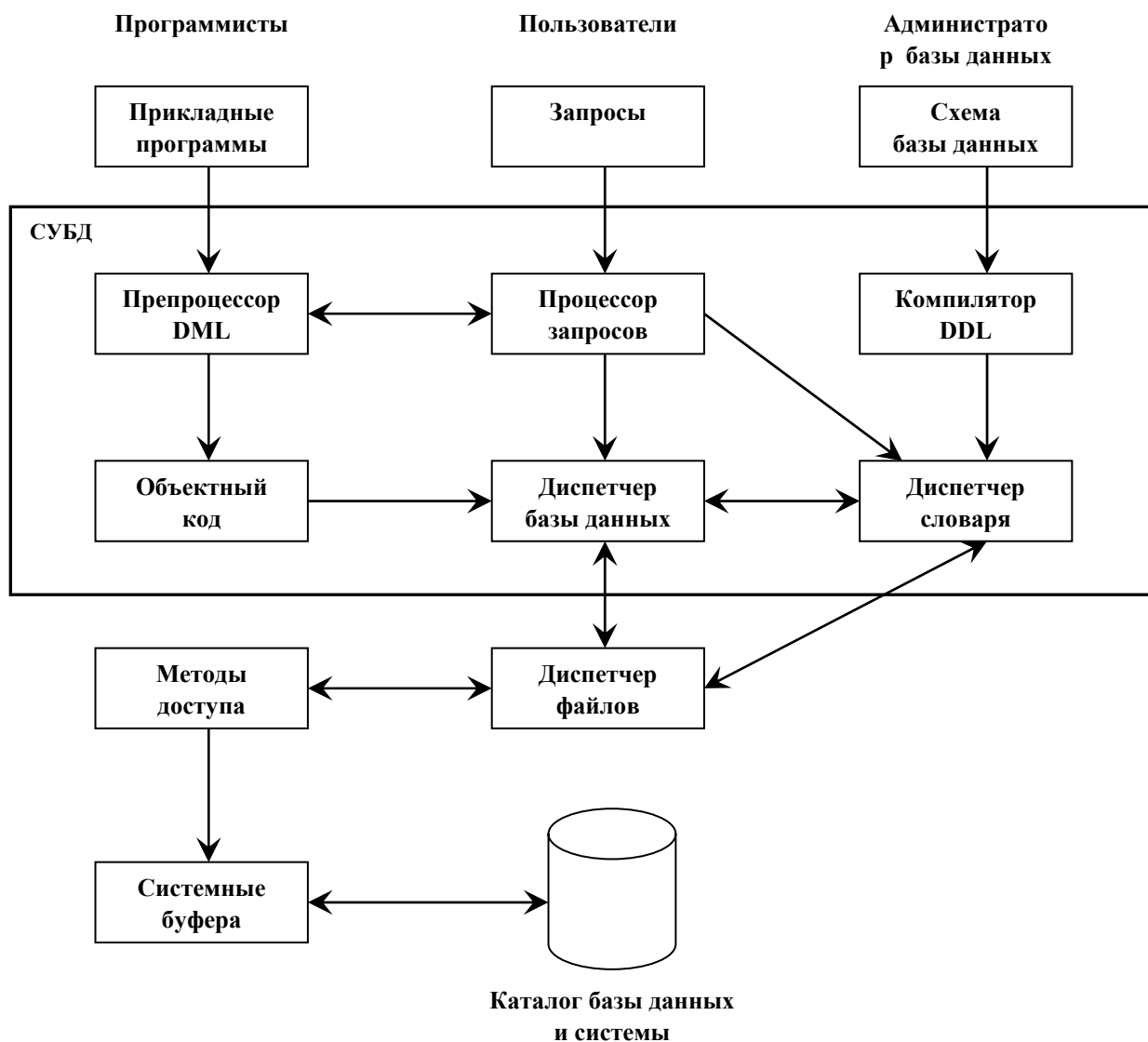


Рис. 1.1. Основные компоненты типичной системы управления базами данных

- **Препроцессор языка DML.** Этот модуль преобразует внедренные в прикладные программы DML-операторы в вызовы стандартных функций базового языка. Для генерации соответствующего кода препроцессор языка DML должен взаимодействовать с процессором запросов.
- **Компилятор языка DDL.** Компилятор языка DDL преобразует DDL-команды в набор таблиц, содержащих метаданные. Затем эти таблицы сохраняются в системном каталоге, а управляющая информация — в заголовках файлов с данными.
- **Диспетчер словаря.** Диспетчер словаря управляет доступом к системному каталогу и обеспечивает работу с ним. Системный каталог доступен большинству компонентов СУБД.

Ниже перечислены основные программные компоненты, входящие в состав Диспетчера базы данных.

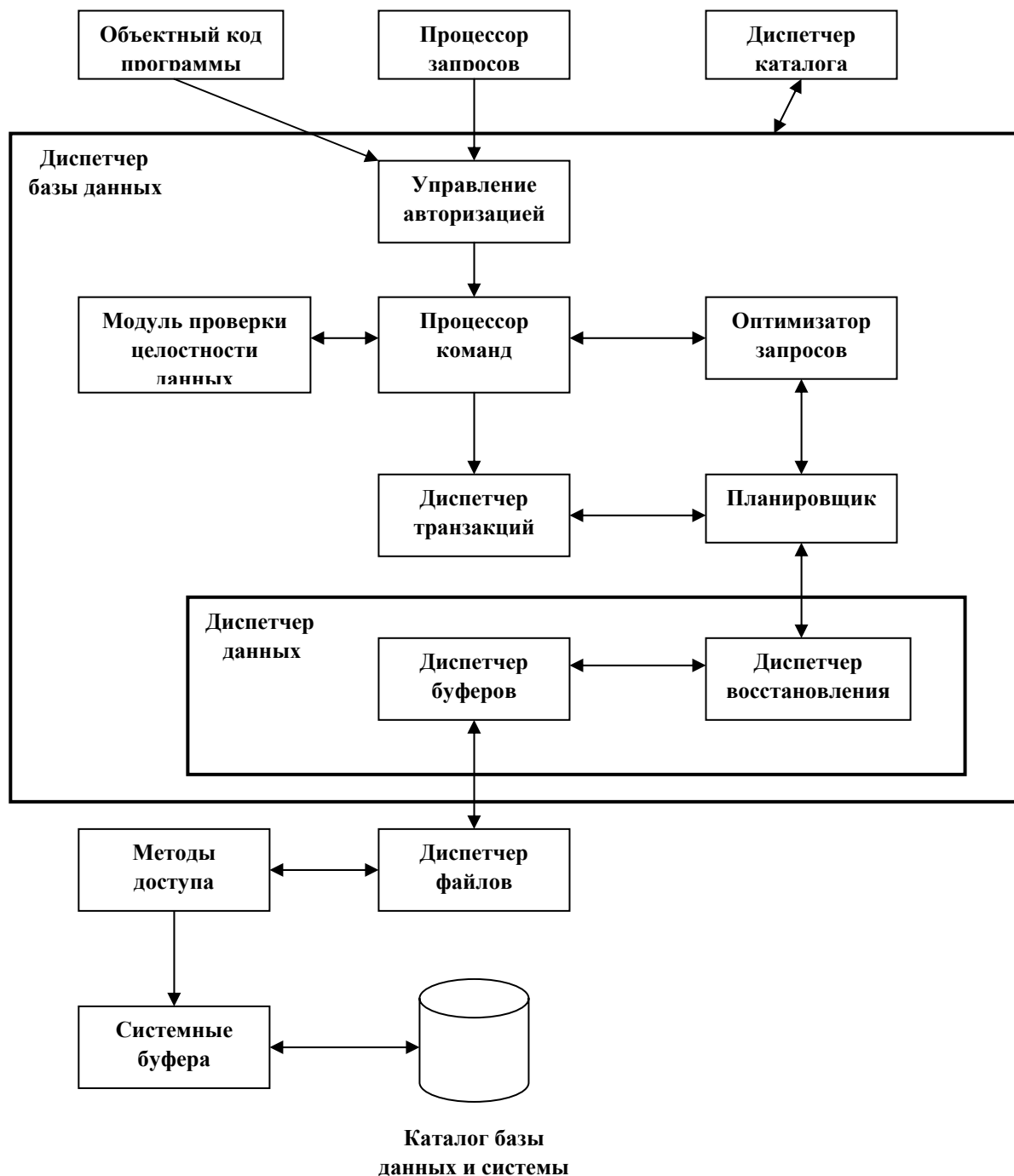


Рис.1.2. Компоненты, диспетчера базы данных

- **Модуль контроля прав доступа.** Этот модуль проверяет наличие у данного пользователя полномочий для выполнения затребованной операции.
- **Процессор команд.** После проверки полномочий пользователя для выполнения затребованной операции управление передается процессору команд.
- **Средства контроля целостности.** В случае операций, которые изменяют содержимое базы данных, средства контроля целостности выполняют проверку того, удовлетворяет ли затребованная операция всем установленным ограничениям поддержки целостности данных (например, требованиям, установленным для ключей).

- **Оптимизатор запросов.** Этот модуль определяет оптимальную стратегию выполнения запроса. Более подробно оптимизация запросов рассматривается в главе 20.
- **Диспетчер транзакций.** Этот модуль осуществляет требуемую обработку операций, поступающих в процессе выполнения транзакций.
- **Планировщик.** Этот модуль отвечает за бесконфликтное выполнение параллельных операций с базой данных. Он управляет относительным порядком выполнения операций, затребованных в отдельных транзакциях.
- **Диспетчер восстановления.** Этот модуль гарантирует восстановление базы данных до непротиворечивого состояния при возникновении сбоев. В частности, он отвечает за фиксацию и отмену результатов выполнения транзакций.
- **Диспетчер буферов.** Этот модуль отвечает за перенос данных между оперативной памятью и вторичным запоминающим устройством — например, жестким диском или магнитной лентой. Диспетчер восстановления и диспетчер буферов иногда (в совокупности) называют *диспетчером данных*, а сам диспетчер буферов — *диспетчером кэша*.

Для воплощения базы данных на физическом уровне помимо перечисленных выше модулей нужны некоторые другие структуры данных. К ним относятся файлы данных и индексов, а также системный каталог. Группой DAFTG (Database Architecture Framework Task Group) была предпринята попытка стандартизации СУБД и в 1986 году предложена некоторая эталонная модель. Назначение эталонной модели заключается в определении концептуальных рамок для разделения предпринимаемых попыток стандартизации на более управляемые части и указания взаимосвязей между ними на очень широком уровне.

1.8 МОДЕЛИ ДАННЫХ НА ОСНОВЕ ЗАПИСЕЙ

В модели на основе записей база данных состоит из нескольких записей фиксированного формата, которые могут иметь разные типы. Каждый тип записи определяет фиксированное количество полей, каждое из которых имеет фиксированную длину. Существуют три основных типа логических моделей данных на основе записей: *реляционная модель данных* (relational data model), *сетевая модель данных* (network data model) и *иерархическая модель данных* (hierarchical data model). Иерархическая и сетевая модели данных были созданы почти на десять лет раньше реляционной модели данных, потому их связь с концепциями традиционной обработки файлов более очевидна.

1.8.1 Реляционная модель данных

Реляционная модель данных основана на понятии *математических отношений*. В реляционной модели данные и связи представлены в виде таблиц, каждая из которых имеет несколько столбцов с уникальными именами. В табл. 1.3 и 1.4 показан пример

реляционной схемы некоторой части проекта, содержащей сведения об отделениях компании и персонале организации. Например, из табл. 1.4 видно, что сотрудник John White работает менеджером с годовой зарплатой 30000 фунтов стерлингов в отделении компании с номером (branchNo) B005, который, согласно данным из табл. 1.3, расположен по адресу 22 Deer Rd, London. Здесь важно отметить, что между отношениями Staff и Branch существует следующая связь: сотрудник *работает* в отделении компании. Однако между этими двумя отношениями нет явно заданной связи; ее существование можно заметить, только зная, что атрибут branchNo в отношении Staff эквивалентен атрибуту branchNo в отношении Branch.

Таблица 1.3. Пример описания сущности Branch в реляционной схеме

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Таблица 1.4. Пример описания сущности Staff в реляционной схеме

StaffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Обратите внимание, что в реляционной модели данных единственное требование состоит в том, чтобы база данных с точки зрения пользователя выглядела как набор таблиц. Однако такое восприятие относится только к логической структуре базы данных. Оно не относится к физической структуре базы данных, которая может быть реализована с помощью разнообразных структур хранения.

1.8.2 Сетевая модель данных

В сетевой модели данные представлены в виде коллекций *записей*, а связи — в виде *наборов*. В отличие от реляционной модели, связи здесь явным образом моделируются наборами, которые реализуются с помощью указателей. Сетевую модель можно представить как граф с записями в виде *узлов* графа и наборами в виде его *ребер*. На рис. 1.3 показан пример сетевой схемы для тех же наборов данных, которые показаны в табл. 1.3 и 1.4. Самой популярной сетевой СУБД является система IDMS/R фирмы Computer Associates.

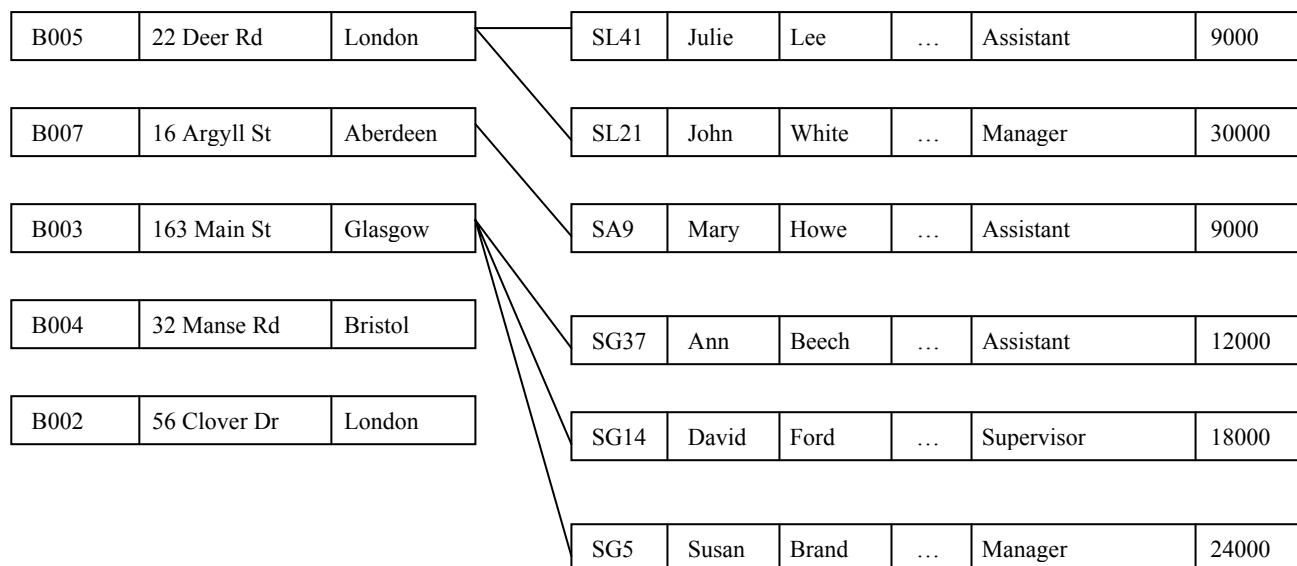


Рис.1.3. Пример фрагмента сетевой схемы

1.8.3 Иерархическая модель данных

Иерархическая модель является ограниченным подтипом сетевой модели. В ней данные также представлены как коллекции *записей*, а связи — как *наборы*. Однако в иерархической модели узел может иметь только одного родителя. Иерархическая модель может быть представлена как древовидный граф с записями в виде узлов (которые также называются *сегментами*) и множествами в виде ребер. На рис. 1.4 приведен пример иерархической схемы для тех же наборов данных, которые показаны в табл. 1.3 и 1.4. Самой распространенной иерархической СУБД является система IMS корпорации IBM, хотя она обладает также некоторыми другими неиерархическими чертами.

Основанные на записях (логические) модели данных используются для определения общей структуры базы данных и высокоуровневого описания ее реализации. Их основной недостаток заключается в том, что они не дают адекватных средств для явного указания ограничений, накладываемых на данные. В то же время в объектных моделях данных отсутствуют средства указания их логической структуры, но за счет предоставления пользователю возможности указать ограничения для данных они позволяют в большей мере представить семантическую суть хранимой информации.

Большинство современных коммерческих систем основано на реляционной модели, тогда как самые первые системы баз данных создавались на основе сетевой или иерархической модели. При использовании последних двух моделей от пользователя требуется знание физической организации базы данных, к которой он должен осуществлять доступ, в то время как при работе с реляционной моделью независимость от данных обеспечивается в значительно большей степени. Следовательно, если в реляционных системах для обработки информации в базе данных принят декларативный

подход (т.е. они указывают, *какие* данные следует извлечь), то в сетевых и иерархических системах — навигационный подход (т.е. они указывают, *как* их следует извлечь).

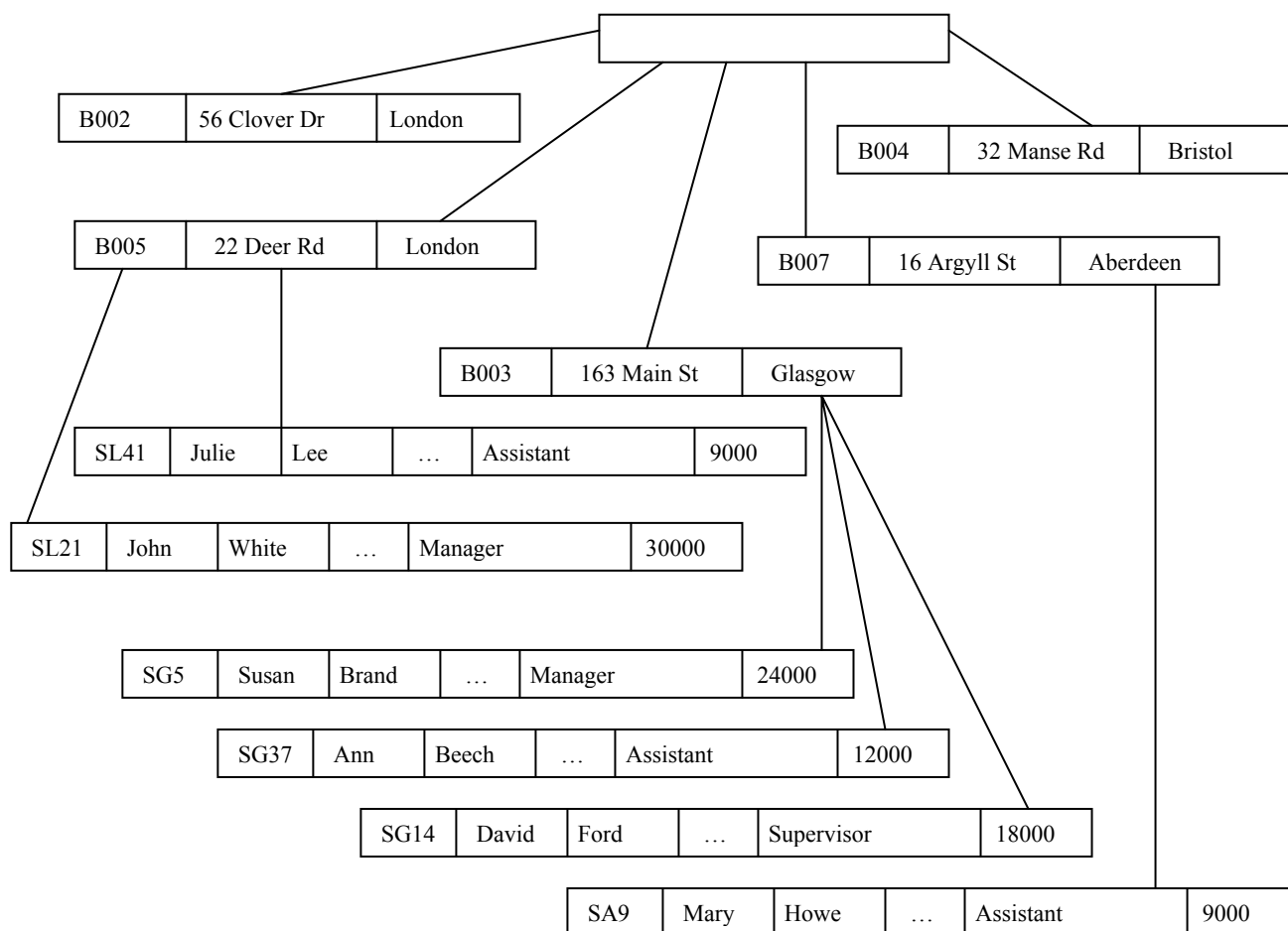


Рис. 1.4. Пример фрагмента иерархической схемы

1.9 ИСТОРИЯ РАЗВИТИЯ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Е.Ф. Кодд (E.F. Codd) в 1970 году сформулировал концепцию *реляционной модели* (*relational model*) БД. Ранее, до появления на рынке реляционной БД DB2, доминирующее положение занимали *иерархические* (*hierarchic*) и *сетевые* (*network*) модели. Широко известным примером СУБД первого типа является IMS фирмы IBM, а второй — IDMS. До появления последних, БД строились на базе *файлов с последовательным доступом* (*flat files*). Для разработки программ доступа к таким БД использовались языки третьего поколения. До сих пор еще встречаются построенные по этому принципу специализированные системы, которые функционируют на больших ЭВМ (*mainframes*) и миникомпьютерах. Группой *Database Task Group (DTBG)* был разработан стандарт *CODASYL* на такого рода БД. Этот стандарт охватывал сетевые БД, сформированные на основе языка COBOL, а IDMS представляла собой реализацию этого стандарта одной из фирм. Но начиная с 70-ых годов доминирующее положение на рынке занимает класс

систем управления реляционными БД, типичными представителями которого являются программные продукты фирм *Oracle*, *Sybase*, *Informix* и *Ingres*.

В настоящее время на передний план выходят объектно-ориентированные СУБД, для которых имеется достаточно много секторов рынка приложений — интегрированные системы автоматизированного проектирования и управления технологическими процессами (САПР/АСУТП, в английской транскрипции — CAD/CAM), системы планирования производства и инжиниринга, мультимедийные системы и т.д. ООСУБД отличаются способностью работать с данными самых различных типов, наличием мощных средств обработки. В борьбе за потребителя фирмы-изготовители реляционных СУБД разработали универсальные серверы, обладающие множеством объектно-ориентированных и мультимедийных функций, включая работу с различными типами данных — текстовыми, звуковыми, статическими и динамическими изображениями. Примером может служить Universal Server фирмы *Oracle*. Кроме того, серверное ядро БД можно нарастить подключением средств работы с типами данных, *определенными пользователем (user-defined)*, и *расширяемыми (extensible)* типами. Эти возможности включены и в Oracle8. Реляционные СУБД с такими функциями можно уже рассматривать как гибридные. Далее идут *многомерные БД (Multi Dimensional Databases — MDD)*, которые также находят свою нишу на рынке. БД подобного класса предлагают усложненный механизм индексации для приложений с множеством переменных, которые нуждаются в сложном многомерном доступе к данным. Это практически невозможно реализовать в традиционных СУБД. Фирмы-производители СУБД, стремясь хоть как-нибудь приблизиться к требованиям MDD, предлагают потребителям программное обеспечение с более развитыми возможностями индексного доступа при помощи специальных технологий, в частности, с использованием *битовых индексов (bitmapped indexes)*. Примером такого рода продукции может служить Express фирмы *Oracle*.

1.10 СТРАТЕГИЧЕСКОЕ ПЛАНИРОВАНИЕ БАЗЫ ДАННЫХ

Переход от состояния, когда данные разрознены и находятся в личном пользовании, к совместному использованию данных значительно легче описать на словах, чем выполнить. Для достижения успеха необходимо, чтобы данные воспринимались как корпоративный ресурс, и на проектирование, реализацию и использование одной или более базы данных потребуется затратить некоторое количество других корпоративных ресурсов. Существенный элемент этого процесса — планирование базы данных.

Планирование базы данных — стратегическая попытка определить информационные потребности организации на продолжительный период времени в будущем. Успешный план базы данных будет предшествовать проектам базы данных и их реализации, чтобы удовлетворить, потребности организации в информации.

1.10.1 Необходимость планирования базы данных

Планирование базы данных определяется информационными потребностями организации, которые, в свою очередь, зависят от бизнес-плана компании, как показано на рис. 1.5. Например, корпорация составляет стратегический бизнес-план на ближайшие пять лет. Выполнение поставленных в плане задач зависит от доступности определенных видов информации. Информация может быть получена только в том случае, если ресурсы данных, намеченные в плане базы данных, имеют место. Это диктует необходимость проектов создания новых баз данных или совершенствования старых.

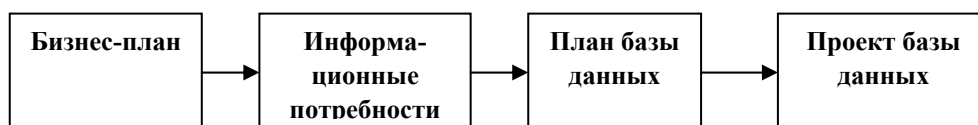


Рис. 1.5. Операционные базы данных являются непосредственным следствием бизнес-планов

Планирование базы данных обладает существенными достоинствами. Некоторые преимущества формального плана информационных ресурсов:

- Он отражает текущее понимание менеджерами информационных ресурсов.
- Он определяет и уточняет требования к ресурсам, помогая убедиться в том, что ресурсы будут доступны.
- Он определяет возможности эффективного управления ресурсами, включая сотрудничество между разными отделами организации.
- Он определяет планы действий для достижения намеченных целей.
- Он может обеспечить мощный стимул и понимание направления развития работниками всех уровней, сосредоточить их усилия, повысить производительность и заставить их в полной мере почувствовать себя частью предприятия.

1.10.2 Проект плана базы данных

Стратегическое планирование базы данных инициируется руководителями высшего уровня. Они распределяют ресурсы и назначают сотрудников, которые будут принимать участие в проекте. По решению управляющих, участники проекта получают все ресурсы, необходимые для успешного создания проекта.

Участники проекта должны обладать большим опытом работы с информационной системой, а также в других областях деятельности компании. Рекомендуется включать в группу четырех постоянных работников, двоих из числа работающих с информационной системой и двоих, знакомых с большинством остальных областей деятельности организации. Все участники проекта должны быть квалифицированными и уважаемыми специалистами, поскольку их работа окажет сильное влияние на деятельность компании на протяжении многих лет. Если они не имеют опыта проведения исследований, следует

пригласить консультанта со стороны, который будет работать в качестве советника по методологии исследования. Руководителем проекта, однако, не должен становиться такой консультант; им должен быть постоянный сотрудник компании, по возможности, начальник администрации базы данных.

На протяжении работы над проектом его участники взаимодействуют с руководителями всех групп пользователей. Главные конечные пользователи определяют основные процессы, виды деятельности и объекты, используемые в ручной или автоматической обработке информации. Участники проекта синтезируют эти данные в корпоративную информационную модель, которая становится частью подробного плана базы данных.

Для того чтобы проект выполнял поставленные перед ним задачи, его разработка не должна длиться более шести месяцев. За это время руководству должен быть представлен отчет, охватывающий планы как минимум на ближайшие пять лет. Этот отчет должен содержать анализ следующих вопросов:

- Информационные потребности различных отделов организации.
- Информационные потребности руководства разного уровня.
- Информационные потребности филиалов организации.
- Информационная модель, отвечающая этим потребностям.
- Предполагаемый объем данных в различных регионах в изучаемый период.
- Предварительные оценки стоимости наращивания возможностей системы.
- Рекомендации по подробному проектированию новой системы или расширения старой с календарными планами.

Главные конечные пользователи определяют процессы, деятельность и объекты

Разработчики проекта синтезируют бизнес-модель и определяют информационную структуру,

Разработчики плана на этом этапе не должны стремиться к детализированной информационной модели. Подробные модели будут разрабатываться при последующем проектировании базы данных. Вместо этого, как отмечает Джеймс, разработчики должны определить стабильные элементы информационной структуры организации — элементы, которые, вероятно, останутся прежними при организационных переменах. По прошествии шести месяцев информационная модель должна быть завершена примерно на 90 процентов. Это означает, что большая часть ее элементов определена, так что модель будет иметь ценность для стратегического планирования.

1.11 ЖИЗНЕННЫЙ ЦИКЛ БАЗЫ ДАННЫХ (ЖЦБД)

Как уже упоминалось выше, система базы данных является фундаментальным

компонентом более широкого понятия — информационной системы организации. Следовательно, жизненный цикл приложений баз данных неразрывно связан с жизненным циклом информационной системы.

Рис. 1.6. Этапы жизненного цикла приложений баз данных

пользователей жизненный цикл может оказаться не очень сложным. Однако он может стать чрезвычайно сложным при проектировании среднего или крупного приложения базы данных, с десятками и даже тысячами пользователей, сотнями запросов и прикладных программ.

Таблица 1.3. Основные действия, выполняемые на каждом этапе жизненного цикла приложения базы данных

Этап	Описание
Планирование разработки базы данных	Планирование наиболее эффективного способа реализации этапов жизненного цикла системы
Определение требований к системе	Определение диапазона действий и границ приложения базы данных, состава его пользователей и областей применения
Сбор и анализ требований пользователей	Сбор и анализ требований пользователей из всех возможных областей применения
Проектирование базы данных	Полный цикл разработки включает концептуальное, логическое и физическое проектирование базы данных
Выбор целевой СУБД (необязательный этап)	Выбор наиболее подходящей СУБД для приложения базы данных
Разработка приложений	Определение пользовательского интерфейса и прикладных программ, которые используют и обрабатывают данные в базе данных
Создание прототипов (необязательный этап)	Создание рабочей модели приложения базы данных, которая позволяет разработчикам или пользователям представить и оценить окончательный вид и способы функционирования системы
Реализация	Создание внешнего, концептуального и внутреннего определений базы данных и прикладных программ
Преобразование и загрузка данных	Преобразование и загрузка данных (и прикладных программ) из старой системы в новую
Тестирование	Приложение базы данных тестируется с целью обнаружения ошибок, а также его проверки на соответствие всем требованиям, выдвинутым пользователями
Эксплуатация и сопровождение	На этом этапе приложение базы данных считается полностью разработанным и реализованным. Впредь вся система будет находиться под постоянным наблюдением и соответствующим образом поддерживаться. В случае необходимости в функционирующее приложение могут вноситься изменения, отвечающие новым требованиям. Реализация этих изменений проводится посредством повторного выполнения некоторых из перечисленных выше этапов жизненного цикла

1.12. ТРЕХУРОВНЕВАЯ АРХИТЕКТУРА ANSI-SPARC

Первая попытка создания стандартной терминологии и общей архитектуры СУБД была предпринята в 1971 году группой DBTG. Группа DBTG признала необходимость использования двухуровневого подхода, построенного на основе использования системного представления, т.е. *схемы* (schema), и пользовательских представлений, т.е. *подсхем* (subschema). Сходные терминология и архитектура были предложены в 1975 году Комитетом планирования стандартов и норм SPARC (Standards Planning and Requirements Committee) Национального института стандартизации США (American National Standard Institute — ANSI), ANSI/X3/SPARC. Комитет ANSI/SPARC признал необходимость использования трехуровневого подхода к созданию системного каталога. В этих

материалах отражены предложения, которые были сделаны организациями *Guide and Share*, состоящими из пользователей продуктов корпорации IBM, и опубликованы за несколько лет до этого. Основное внимание в них было сконцентрировано на необходимости воплощения независимого уровня для изоляции программ от особенностей представления данных на более низком уровне. Хотя модель ANSI/SPARC не стала стандартом, она все еще представляет собой основу для понимания некоторых функциональных особенностей СУБД.

В данном случае для нас наиболее фундаментальным моментом в этих и последующих отчетах исследовательских групп является определение трех уровней абстракции, т.е. трех различных уровней описания элементов данных. Эти уровни формируют *трехуровневую архитектуру*, которая охватывает *внешний, концептуальный и внутренний* уровни, как показано на рис. 1.7. Уровень, на котором данные воспринимаются пользователями, называется *внешним уровнем* (external level), тогда как СУБД и операционная система воспринимают данные на *внутреннем уровне* (internal level). *Концептуальный уровень* (conceptual level) представления данных предназначен для *отображения* внешнего уровня на внутренний и обеспечения необходимой *независимости* друг от друга.

Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее физического представления. Ниже перечислено несколько причин, по которым желательно выполнить такое разделение.

- Каждый пользователь должен иметь возможность обращаться к одним и тем же данным, реализуя свое собственное представление о них. Каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей.
- Пользователи не должны непосредственно иметь дело с такими подробностями физического хранения данных в базе, как индексирование и хеширование. Иначе говоря, взаимодействие пользователя с базой не должно зависеть от особенностей хранения в ней данных.
- Администратор базы данных (АБД) должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления.
- Внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации, как переключение на новое устройство хранения.
- АБД должен иметь возможность изменять концептуальную структуру базы

данных без какого-либо влияния на всех пользователей.

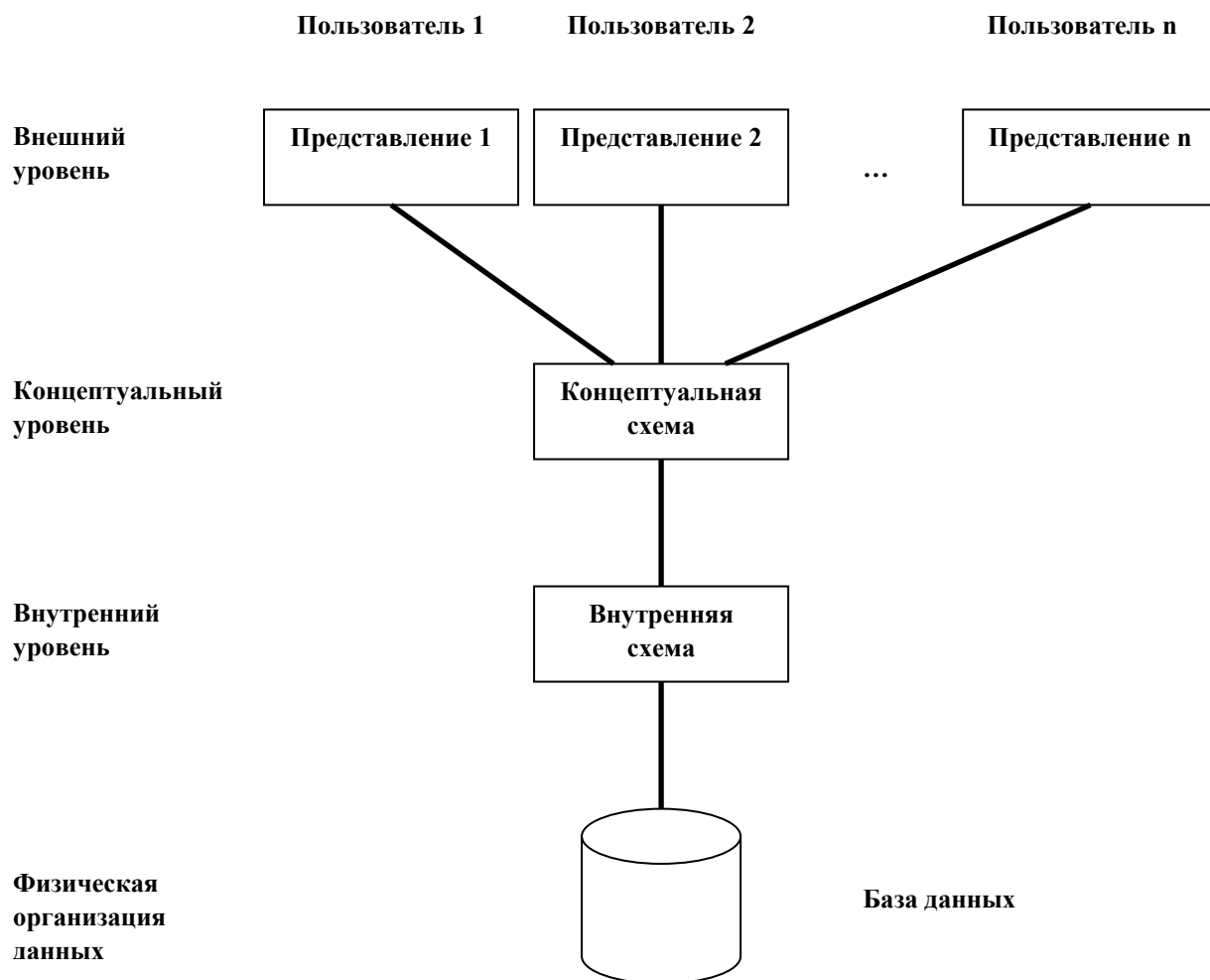


Рис. 1.7. Трехуровневая архитектура ANSI-SPARC

1.12.1. Внешний уровень

Внешний уровень – представление базы данных с точки зрения пользователей. Этот уровень описывает ту часть базы данных, которая относится к каждому пользователю.

Внешний уровень состоит из нескольких различных внешних представлений базы данных. Каждый пользователь имеет дело с представлением "реального мира", выраженным в наиболее удобной для него форме. Внешнее представление содержит только те сущности, атрибуты и связи "реального мира", которые интересны пользователю. Другие сущности, атрибуты или связи, которые ему неинтересны, также могут быть представлены в базе данных, но пользователь может даже не подозревать об их существовании.

Помимо этого, различные представления могут по-разному отображать одни и те же данные. Например, один пользователь может просматривать даты в формате (день, месяц, год), а другой — в формате (год, месяц, день). Некоторые представления могут включать производные или вычисляемые данные, которые не хранятся в базе данных как

таковые, а создаются по мере надобности. Например, можно было бы организовать просмотр данных о возрасте сотрудников. Однако вряд ли стоит хранить эти сведения в базе данных, поскольку в таком случае их пришлось бы ежедневно обновлять. Вместо этого в базе данных хранятся даты рождения сотрудников, а возраст вычисляется средствами СУБД при обнаружении соответствующей ссылки. Представления могут также включать комбинированные или производные данные из нескольких объектов.

1.12.2. Концептуальный уровень

Концептуальный уровень – обобщающее представление базы данных. Этот уровень описывает то, какие данные хранятся в базе данных, а также связи, существующие между ними.

Промежуточным уровнем в трехуровневой архитектуре является концептуальный уровень. Этот уровень содержит логическую структуру всей базы данных (с точки зрения АБД). Фактически это полное представление требований к данным со стороны организации, которое не зависит от любых соображений относительно способа их хранения. На концептуальном уровне представлены следующие компоненты:

- все сущности, их атрибуты и связи;
- накладываемые на данные ограничения;
- семантическая информация о данных;
- информация о мерах обеспечения безопасности и поддержки целостности данных.

Концептуальный уровень поддерживает каждое внешнее представление, в том смысле, что любые доступные пользователю данные должны содержаться (или могут быть вычислены) на этом уровне. Однако этот уровень не содержит никаких сведений о методах хранения данных. Например, описание сущности должно содержать сведения о типах данных атрибутов (целочисленный, действительный или символьный) и их длине (количестве значащих цифр или максимальном количестве символов), но не должно включать сведений об организации хранения данных, например об объеме занятого пространства в байтах.

1.12.3. Внутренний уровень

Внутренний уровень – физическое представление базы данных в компьютере. Этот уровень описывает, как информация храниться в базе данных.

Внутренний уровень описывает физическую реализацию базы данных и предназначен для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. Он содержит описание структур данных и организации отдельных файлов, используемых для хранения данных на запоминающих

устройствах. На этом уровне осуществляется взаимодействие СУБД с методами доступа операционной системы (вспомогательными функциями хранения и извлечения записей данных) с целью размещения данных на запоминающих устройствах, создания индексов, извлечения данных и т.д. На внутреннем уровне хранится следующая информация:

- распределение дискового пространства для хранения данных и индексов;
- описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных);
- сведения о размещении записей;
- сведения о сжатии данных и выбранных методах их шифрования.

Ниже внутреннего уровня находится *физический уровень* (physical level), который контролируется операционной системой, но под управлением СУБД. Однако функции СУБД и операционной системы на физическом уровне не вполне четко разделены и могут варьироваться от системы к системе. В одних СУБД используются многие предусмотренные в данной операционной системе методы доступа, тогда как в других применяются только самые основные и реализована собственная файловая организация. Физический уровень доступа к данным ниже СУБД состоит только из известных операционной системе элементов (например, указателей на то, как реализовано последовательное распределение и хранятся ли поля внутренних записей на диске в виде непрерывной последовательности байтов).

1.12.4. Схемы, отображения и экземпляры

Общее описание базы данных называется *схемой базы данных*. Существуют три различных типа схем базы данных, которые определяются в соответствии с уровнями абстракции трехуровневой архитектуры. На самом высоком уровне имеется несколько *внешних схем* или *подсхем*, которые соответствуют разным представлениям данных. На концептуальном уровне описание базы данных называют *концептуальной схемой*, а на самом низком уровне абстракции — *внутренней схемой*. Концептуальная схема описывает все сущности, атрибуты и связи между ними, с указанием необходимых ограничений поддержки целостности данных. На нижнем уровне находится внутренняя схема, которая является полным описанием внутренней модели данных. Она содержит определения хранимых записей и методов представления, описания полей данных, сведения об индексах и выбранных схемах хеширования. Для каждой базы данных существует только одна концептуальная и одна внутренняя схема.

СУБД отвечает за установление соответствия между этими тремя типами схем, а также за проверку их непротиворечивости. Иначе говоря, СУБД должна обеспечивать, чтобы каждую внешнюю схему можно было вывести на основе концептуальной схемы.

Для установления соответствия между любыми внешней и внутренней схемами СУБД должна использовать информацию из концептуальной схемы. Концептуальная схема связана с внутренней схемой посредством *концептуально внутреннего отображения*. Оно позволяет СУБД найти фактическую запись или набор записей на физическом устройстве хранения, которые образуют *логическую запись* в концептуальной схеме, с учетом любых ограничений, установленных для операций, выполняемых над данной логической записью. Оно также позволяет обнаружить любые различия в именах объектов, именах атрибутов, порядке следования атрибутов, их типах данных и т.д. Наконец, каждая внешняя схема связана с концептуальной схемой с помощью *концептуально внешнего отображения*. С его помощью СУБД может отображать имена пользовательского представления на соответствующую часть концептуальной схемы.

Важно различать описание базы данных и саму базу данных. Описанием базы данных является *схема базы данных*. Схема создается в процессе проектирования базы данных, причем предполагается, что она изменяется достаточно редко. Однако содержащаяся в базе данных информация может меняться часто — например, всякий раз при вставке сведений о новом сотруднике или новом объекте сдаваемой в аренду недвижимости. Совокупность информации, хранящейся в базе данных в любой определенный момент времени, называется *состоянием, базы данных*. Следовательно, одной и той же схеме базы данных может соответствовать множество ее различных состояний. Схема базы данных иногда именуется *содержанием* базы данных, а ее состояние — *детализацией*.

1.13 АРХИТЕКТУРА МНОГОПОЛЬЗОВАТЕЛЬСКИХ СУБД

В этом разделе мы познакомимся с различными типовыми архитектурными решениями, используемыми при реализации многопользовательских СУБД, а именно со схемами обычной телеобработки, файловым сервером и технологией "клиент/сервер".

1.13.1 Телеобработка

Традиционной архитектурой многопользовательских систем раньше считалась схема, получившая название *телеобработки*, при которой один компьютер с единственным процессором был соединен с несколькими терминалами, как показано на рис. 1.8. При этом вся обработка выполнялась с помощью единственного компьютера, а присоединенные к нему пользовательские терминалы были типичными "неинтеллектуальными" устройствами, не способными функционировать самостоятельно. С центральным процессором терминалы были связаны с помощью кабелей, по которым они посылали сообщения пользовательским приложениям (через подсистему управления обменом данными операционной системы). В свою очередь, пользовательские приложения обращались к необходимым службам СУБД. Таким же образом сообщения

возвращались назад на пользовательский терминал. К сожалению, при такой архитектуре основная и чрезвычайно большая нагрузка возлагалась на центральный компьютер, который должен был выполнять не только действия прикладных программ и СУБД, но и значительную работу по обслуживанию терминалов (например, форматирование данных, выводимых на экраны терминалов).

В последние годы был достигнут существенный прогресс в разработке высокопроизводительных персональных компьютеров и составленных из них сетей. При этом во всей индустрии наблюдается заметная тенденция к *децентрализации* (downsizing), т.е. замене дорогих мейнфреймов более эффективными, с точки зрения эксплуатационных затрат, сетями персональных компьютеров, позволяющими получить такие же, если не лучшие, результаты. Эта тенденция привела к появлению следующих двух типов архитектуры СУБД: технологии файлового сервера и технологии "клиент/сервер".

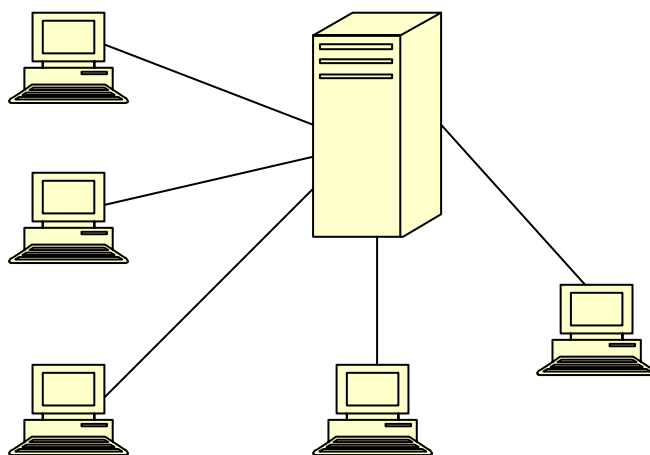


Рис. 1.8. Топология архитектуры телеобработки

1.13.2. Файловый сервер

В среде файлового сервера обработка данных распределена в сети, обычно представляющей собой локальную вычислительную сеть (ЛВС). Файловый сервер содержит файлы, необходимые для работы приложений и самой СУБД. Однако пользовательские приложения и СУБД размещены и функционируют на отдельных рабочих станциях, и обращаются к файловому серверу только по мере необходимости получения доступа к нужным им файлами, как показано на рис. 1.9.

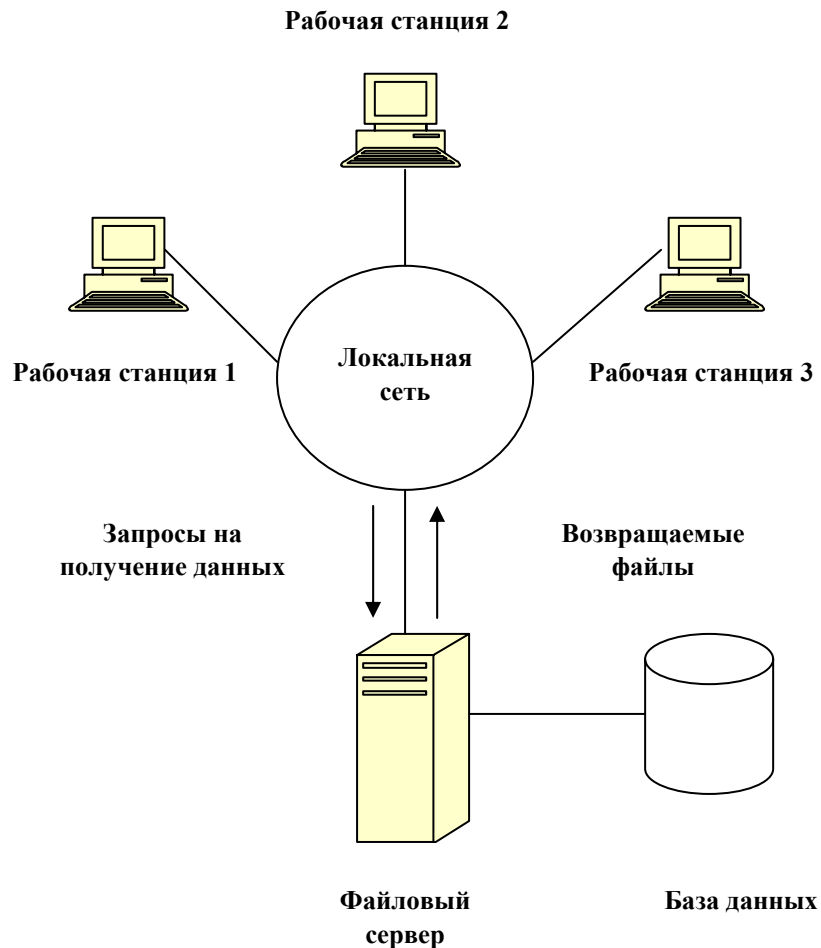


Рис.1.9. Архитектура с использованием файлового сервера

Таким образом, файловый сервер функционирует просто как совместно используемый жесткий диск. СУБД на каждой рабочей станции посылает запросы файловому серверу по всем необходимым ей данным, которые хранятся на диске файлового сервера. Такой подход характеризуется значительным сетевым трафиком, что может привести к снижению производительности всей системы в целом.

Архитектура с использованием файлового сервера обладает следующими основными недостатками.

1. Большой объем сетевого трафика.
2. На каждой рабочей станции должна находиться полная копия СУБД.
3. Управление параллельной работой, восстановлением и целостностью усложняется, поскольку доступ к одним и тем же файлам могут осуществлять сразу несколько экземпляров СУБД.

1.13.3. Технология "клиент/сервер"

Технология "клиент/сервер" была разработана с целью устранения недостатков, имеющих в первых двух подходах. В этой технологии используется способ взаимодействия программных компонентов, при котором они образуют единую систему.

Как видно из самого названия, существует некий *клиентский* процесс, требующий определенных ресурсов, а также *серверный* процесс, который эти ресурсы предоставляет. При этом совсем не обязательно, чтобы они находились на одном и том же компьютере. На практике принято размещать сервер на одном узле локальной сети, а клиенты — на других узлах. На рис. 1.10 показана архитектура типа "клиент/сервер".

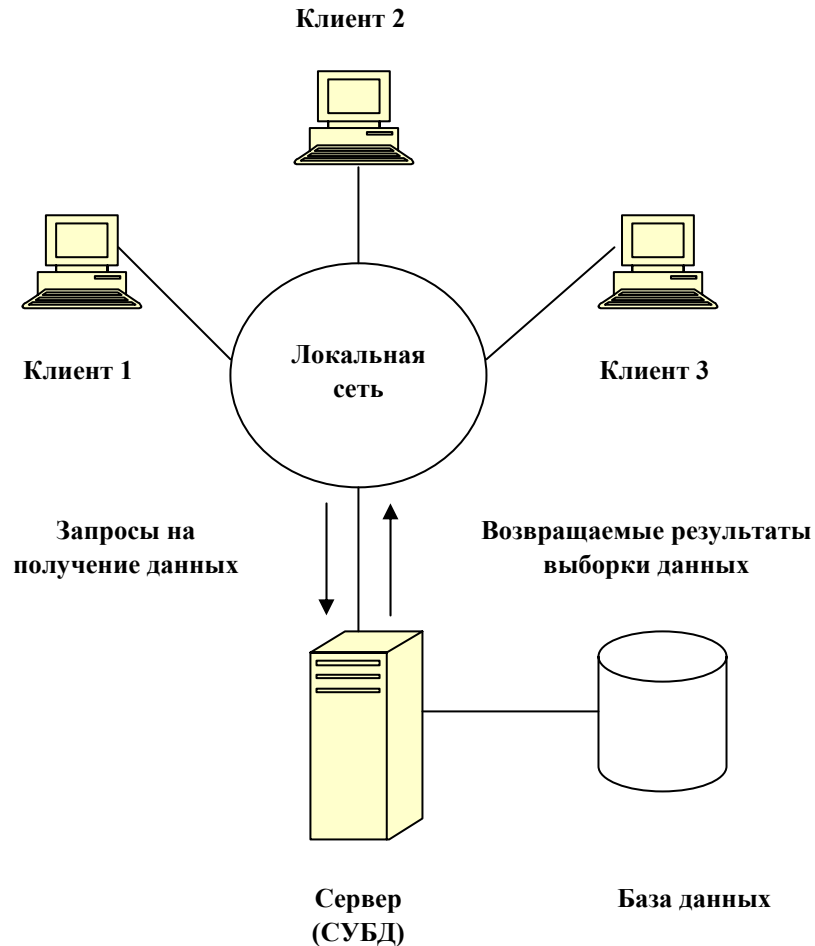


Рис.1.10. Общая схема построения систем с архитектурой «клиент/сервер»

В контексте базы данных клиент управляет пользовательским интерфейсом и логикой приложения, действуя как сложная рабочая станция, на которой выполняются приложения баз данных. Клиент принимает от пользователя запрос, проверяет синтаксис и генерирует запрос к базе данных на языке SQL или другом языке базы данных, который соответствует логике приложения. Затем он передает сообщение серверу, ожидает поступления ответа и форматирует полученные данные для представления их пользователю. Сервер принимает и обрабатывает запросы к базе данных, а затем передает полученные результаты обратно клиенту. Такая обработка включает проверку полномочий клиента, обеспечение требований целостности, поддержку системного каталога, а также выполнение запроса и обновление данных. Помимо этого, поддерживается управление параллельной работой и восстановлением. Выполняемые клиентом и сервером операции приведены в табл. 1.4.

Таблица 1.4. Функции, выполняемые участниками взаимодействия в среде "клиент/сервер"

Клиент	Сервер
Управляет пользовательским интерфейсом	Принимает и обрабатывает запросы к базе данных со стороны клиентов
Принимает и проверяет синтаксис введенного	Проверяет полномочия пользователей пользователем запроса
Выполняет приложение	Гарантирует соблюдение ограничений целостности
Генерирует запрос к базе данных и передает серверу	Выполняет запросы/обновления и его возвращает результаты клиенту
Отображает полученные данные пользователю	Поддерживает системный каталог Обеспечивает параллельный доступ к базе данных Обеспечивает управление восстановлением

Этот тип архитектуры обладает приведенными ниже преимуществами.

- Обеспечивается более широкий доступ к существующим базам данных.
- Повышается общая производительность системы. Поскольку клиенты и сервер находятся на разных компьютерах, их процессоры способны выполнять приложения параллельно. При этом настройка производительности компьютера с сервером упрощается, если на нем выполняется только работа с базой данных.
- Стоимость аппаратного обеспечения снижается. Достаточно мощный компьютер с большим устройством хранения нужен только серверу — для хранения и управления базой данных.
- Сокращаются коммуникационные расходы. Приложения выполняют часть операций на клиентских компьютерах и посылают через сеть только запросы к базе данных, что позволяет существенно сократить объем пересылаемых по сети данных.
- Повышается уровень непротиворечивости данных. Сервер может самостоятельно управлять проверкой целостности данных, поскольку все ограничения определяются и проверяются только в одном месте. При этом каждому приложению не приходится выполнять собственную проверку.
- Эта архитектура весьма естественно отображается на архитектуру открытых систем.

1.14 ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ

Процесс проектирования базы данных состоит из трех основных этапов: концептуальное, логическое и физическое проектирование.

1.14.1 Концептуальное проектирование базы данных

Концептуальное проектирование базы данных - процесс создания модели используемой на предприятии информации, не зависящей от любых физических аспектов ее представления.

Первый этап процесса проектирования базы данных называется *концептуальным проектированием* базы данных. Он заключается в создании концептуальной модели

данных для анализируемой части предприятия. Эта модель данных создается на основе информации, записанной в спецификациях требований пользователей. Концептуальное проектирование базы данных абсолютно не зависит от таких подробностей ее реализации, как тип выбранной целевой СУБД, набор создаваемых прикладных программ, используемые языки программирования, тип выбранной вычислительной платформы, а также от любых других особенностей физической реализации.

При разработке концептуальная модель данных постоянно подвергается тестированию и проверке на соответствие требованиям пользователей. Созданная концептуальная модель данных предприятия является источником информации для этапа логического проектирования базы данных.

1.14.2 Логическое проектирование базы данных

Логическое проектирование базы данных – процесс создания модели используемой на предприятии информации на основе выбранной модели организации данных, но без учета типа целевой СУБД и других физических аспектов реализации.

Второй этап проектирования базы данных называется *логическим проектированием* базы данных. Его цель состоит в создании логической модели данных для исследуемой части предприятия. Концептуальная модель данных, созданная на предыдущем этапе, уточняется и преобразуется в логическую модель данных. Логическая модель данных учитывает особенности выбранной модели организации данных в целевой СУБД (например, реляционная модель).

Если концептуальная модель данных не зависит от любых физических аспектов реализации, то логическая модель данных создается на основе выбранной модели организации данных целевой СУБД. Иначе говоря, на этом этапе уже должно быть известно, какая СУБД будет использоваться в качестве целевой — реляционная, сетевая, иерархическая или объектно-ориентированная. Однако на этом этапе игнорируются все остальные характеристики выбранной СУБД, например, любые особенности физической организации ее структур хранения данных и построения индексов.

В процессе разработки логическая модель данных постоянно тестируется и проверяется на соответствие требованиям пользователей. Для проверки правильности логической модели данных используется метод *нормализации*. Нормализация гарантирует, что отношения, выведенные из существующей модели данных, не будут обладать избыточностью данных, способной вызвать нарушения в процессе обновления данных после их физической реализации. Помимо всего прочего, логическая модель данных должна обеспечивать поддержку всех необходимых пользователям транзакций.

Созданная логическая модель данных является источником информации для этапа физического проектирования и обеспечивает разработчика физической базе данных

средствами поиска компромиссов, необходимых для достижения поставленных целей, что очень важно для эффективного проектирования. Логическая модель данных играет также важную роль на этапе эксплуатации и сопровождения уже готовой системы. При правильно организованном сопровождении поддерживаемая в актуальном состоянии модель данных позволяет точно и наглядно представить любые вносимые в базу данных изменения, а также оценить их влияние на прикладные программы и использование данных, уже имеющихся в базе.

1.14.3 Физическое проектирование базы данных

Физическое проектирование базы данных – процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах; на этом этапе рассматриваются основные отношения, организация файлов и индексов, предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.

Физическое проектирование является третьим и последним этапом создания проекта базы данных, при выполнении которого проектировщик принимает решения о способах реализации разрабатываемой базы данных. Во время предыдущего этапа проектирования была определена логическая структура базы данных (которая описывает отношения и ограничения в рассматриваемой прикладной области). Хотя эта структура не зависит от конкретной целевой СУБД, она создается с учетом выбранной модели хранения данных, например реляционной, сетевой или иерархической. Однако, приступая к физическому проектированию базы данных, прежде всего необходимо выбрать конкретную целевую СУБД. Поэтому физическое проектирование неразрывно связано с конкретной СУБД. Между логическим и физическим проектированием существует постоянная обратная связь, так как решения, принимаемые на этапе физического проектирования с целью повышения производительности системы, способны повлиять на структуру логической модели данных.

Как правило, основной целью физического проектирования базы данных является описание способа физической реализации логического проекта базы данных. В случае реляционной модели данных под этим подразумевается следующее:

- создание набора реляционных таблиц и ограничений для них на основе информации, представленной в глобальной логической модели данных;
- определение конкретных структур хранения данных и методов доступа к ним, обеспечивающих оптимальную производительность СУБД;
- разработка средств защиты создаваемой системы.

Этапы концептуального и логического проектирования больших систем следует

отделять от этапов физического проектирования. На это есть несколько причин.

- Они связаны с совершенно разными аспектами системы, поскольку отвечают на вопрос, *что* делать, а не *как* делать.
- Они выполняются в разное время, поскольку понять, *что* надо сделать, следует прежде, чем решить, *как* это сделать.
- Они требуют совершенно разных навыков и опыта, поэтому требуют привлечения специалистов различного профиля.

Проектирование базы данных — это итерационный процесс, который имеет свое начало, но не имеет конца и состоит из бесконечного ряда уточнений. Его следует рассматривать прежде всего как процесс познания. Как только проектировщик приходит к пониманию работы предприятия и смысла обрабатываемых данных, а также выражает это понимание средствами выбранной модели данных, приобретенные знания могут показать, что требуется уточнение и в других частях проекта. Особо важную роль в общем процессе успешного создания системы играет концептуальное и логическое проектирование базы данных. Если на этих этапах не удастся получить полное представление о деятельности предприятия, то задача определения всех необходимых пользовательских представлений или обеспечения защиты базы данных становится чрезмерно сложной или даже неосуществимой. К тому же может оказаться затруднительным определение способов физической реализации или достижения приемлемой производительности системы. С другой стороны, способность адаптироваться к изменениям является одним из признаков удачного проекта базы данных. Поэтому вполне имеет смысл затратить время и энергию, необходимые для подготовки наилучшего возможного проекта.

1.14.4. Проектирование транзакций

Прежде чем перейти к описанию проектирования транзакций, рассмотрим определение понятия транзакции.

Транзакция – одно или последовательность действий, выполняемых одним и тем же пользователем (или прикладной программой), которые получают доступ к базе данных или изменяют ее содержимое.

Транзакции представляют такие события реального мира, как, например, регистрация предлагаемого для сдачи в аренду объекта недвижимости, прием на работу нового сотрудника или же регистрация нового клиента и сдача в аренду объекта недвижимости. Все эти транзакции должны обращаться к базе данных с той целью, чтобы хранимые в ней данные всегда соответствовали текущей ситуации в реальном мире, а также для удовлетворения информационных потребностей пользователей.

Транзакция может состоять из нескольких операций, подобных, например,

переводу денег с одного счета на другой. Однако с точки зрения пользователя эти операции представляют собой единое задание. А с точки зрения проектировщика СУБД каждая транзакция переводит базу данных из одного непротиворечивого состояния в другое. СУБД обеспечивает непротиворечивость данных в базе даже в случае возникновения сбоя. Кроме того, СУБД гарантирует, что после завершения транзакции все внесенные ею изменения будут надежно сохранены в базе данных (без необходимости выполнения другой транзакции для устранения ошибок, возникших при выполнении первой транзакции). Если по какой-либо причине транзакция не будет завершена, СУБД гарантирует, что все внесенные ею изменения будут отменены. В примере с банковским переводом денег это значит, что если деньги сняты (дебетованы) с одного счета и сбой транзакции произошел во время их внесения (кредитования) на другой счет, то СУБД отменит дебет первого счета. Если операции дебета и кредита поместить в отдельные транзакции, то сразу после дебетования первого счета и завершения транзакции это изменение отменить будет нельзя, разве что только путем запуска другой транзакции с кредитованием этого счета на снятую сумму.

Цель проектирования транзакций заключается в определении и документировании высокоуровневых характеристик всех транзакций, которые должны будут выполняться в разрабатываемой базе данных, в том числе:

- данные, которые используются транзакцией;
- функциональные характеристики транзакции;
- выходные данные, формируемые транзакцией;
- степень важности транзакции для пользователей;
- предполагаемая интенсивность использования.

Эту работу следует выполнить еще на начальной стадии проектирования, что позволит обеспечить поддержку всех требуемых транзакций со стороны логической модели данных. Существуют три основных типа транзакций: транзакции извлечения, обновления и смешанные транзакции.

- *Транзакции извлечения* используются для выборки некоторых данных с целью отображения их на экране или подготовки отчета. Примером транзакции извлечения является поиск и отображение подробных сведений об объекте недвижимости (по заданному номеру объекта).

- *Транзакции обновления* используются для вставки новых, удаления старых или же изменения уже существующих записей базы данных. Примером транзакции обновления является внесение в базу подробных сведений о новом объекте недвижимости.

- *Смешанные транзакции* включают как операции извлечения, так и операции обновления данных. Примером смешанной транзакции является поиск и отображение подробных сведений об объекте недвижимости (по заданному номеру объекта), с последующим изменением месячной арендной платы.

1.15 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Приведите общенаучное и нормативно-правовое определения информации.
2. Дайте определение данных.
3. Какие данные можно отнести к структурированным? Приведите примеры.
4. Сформулируйте определение информационной системы.
5. Какие типы АИС вам известны?
6. Назовите недостатки традиционных файловых систем.
7. Что представляют собой база данных, СУБД?
8. Перечислите преимущества и недостатки СУБД.
9. Охарактеризуйте основные компоненты СУБД.
10. Какие основные модели данных на основе записей вы знаете?
11. Выполните сравнительный анализ известных вам моделей данных.
12. В чем заключается стратегическое планирование баз данных?
13. Опишите каждый уровень архитектуры ANSI-SPARC.
14. Что представляет собой технология «клиент/сервер»?
15. Перечислите основные этапы проектирования базы данных.

1.16 УПРАЖНЕНИЯ

Задание: Выполнить описание предметной области «Зоопарк».

В описании должны быть четко представлены процессы, протекающие в указанной предметной области. Такое описание дает разработчику возможность представить потребности заказчика и реализовать программу, которая бы им удовлетворяла. Кроме этого в описании должны быть сформулированы вопросы, на которые будет отвечать разработанная программа.

Рассмотри пример такого описания.

Служащих зоопарка можно подразделить на несколько категорий: ветеринары, уборщики, дрессировщики, строители-ремонтники, работники администрации. Каждая из перечисленных категорий работников имеет уникальные атрибуты-характеристики, определяемые профессиональной направленностью. За каждым животным ухаживает определенный круг служащих, причем только ветеринарам, уборщикам и дрессировщикам разрешен доступ в клетки к животным.

В зоопарке обитают животные различных климатических зон, поэтому часть животных на зиму необходимо переводить в отапливаемые помещения. Животных можно подразделить на хищников и травоядных. При расселении животных по клеткам необходимо учитывать не только потребности данного вида, но и их совместимость с животными в соседних клетках (нельзя рядом селить, например, волков и их добычу - различных копытных).

Для кормления животных необходимы различные типы кормов: растительный, живой, мясо и различные комбикорма. Растительный корм это фрукты и овощи, зерно и сено. Живой корм - мыши, птицы, корм для рыб. Для каждого вида животных рассчитывается свой рацион, который в свою очередь варьируется в зависимости от возраста, физического состояния животного и сезона. Таким образом у каждого животного в зоопарке имеется меню на каждый день, в котором указывается количество и время кормлений в день, количество и вид пищи (обезьянам необходимы фрукты и овощи, мелким хищникам - хорькам, ласкам, совам, некоторым кошачьим, змеям - надо давать мышей). У зоопарка имеются поставщики кормов для животных. Каждый поставщик специализируется на каких-то конкретных видах кормов. Часть кормов зоопарк может производить сам: запастись сеном, разводить мышей и т.д.

Ветеринары должны проводить медосмотры, следить за весом, ростом, развитием животного, ставить своевременно прививки и заносить все эти данные в карточку, которая заводится на каждую особь при ее появлении в зоопарке. Больным животным назначается лечение и при необходимости их можно изолировать в стационаре.

При определенных условиях (наличие пары особей, подходящих по возрасту, физическому состоянию) можно ожидать появления потомства. Потомство от данной пары животных при достижении ими положенного возраста можно либо оставить в зоопарке, создав для них подходящие условия содержания, либо обменяться с другими зоопарками или просто раздать в другие зоопарки - по решению администрации.

Виды запросов в информационной системе:

1. Получить список и общее число служащих зоопарка, либо служащих данной категории полностью, по продолжительности работы в зоопарке, по половому признаку, возрасту, размеру заработной платы.
2. Получить перечень и общее число служащих зоопарка, ответственных за указанный вид животных либо за конкретную особь за все время пребывания животного в зоопарке, за указанный период времени.
3. Получить список и общее число служащих зоопарка, имеющих доступ к указанному виду животных либо к конкретной особи.
4. Получить перечень и общее число всех животных в зоопарке либо

животных указанного вида, живших в указанной клетке все время пребывания в зоопарке, по половому признаку, возрасту, весу, росту.

5. Получить перечень и общее число нуждающихся в теплом помещении на зиму, полностью животных только указанного вида или указанного возраста.

6. Получить перечень и общее число животных, которым поставлена указанная прививка, либо переболевших некоторой болезнью, по длительности пребывания в зоопарке, половому признаку, возрасту, признаку наличия и количеству потомства.

7. Получить перечень всех животных, совместимых с указанным видом, либо только тех животных, которых необходимо переселить, или тех, которые нуждаются в теплом помещении.

8. Получить перечень и общее число поставщиков кормов полностью, либо поставляющих только определенный корм, поставлявших в указанный период, по количеству поставляемого корма, цене, датам поставок.

9. Получить перечень и объем кормов, производимых зоопарком полностью, либо только тех кормов, в поставках которых зоопарк не нуждается (обеспечивает себя сам).

10. Получить перечень и общее число животных полностью, либо указанного вида, которым необходим определенный тип кормов, в указанном сезоне, возрасте или круглый год.

11. Получить полную информацию (рост, вес, прививки, болезни, дата поступления в зоопарк или дата рождения, возраст, количество потомства) о всех животных, или о животных только данного вида, о конкретном животном, об особи, живущей в указанной клетке.

Задания:

1. Модель для университета. Сколько преподавателей работает на математическом факультете? Их фамилии? Какие предметы они преподают?

2. Модель для университета. Какие студенты специализируются в истории? В английском?

3. Модель для университета. Кто из преподавателей читает социологические курсы? Какие курсы они читают? Каким группам студентов?

4. Модель для университета. Сколько студентов, чьей специальностью является немецкий язык, официально зарегистрированы на усиленной программе? Кто является преподавателем каждого из них?

5. Модели для торговой фирмы. Какие товары имеют продажную цену более 200 рублей? Какие из них имеют закупочную цену менее 150 рублей? Какие товары

произведены на Москве? Кто их изготовители?

6. Модели для торговой фирмы. Кто из продавцов продал товары ценой более 200 рублей? Даты этих продаж? Какова базовая зарплата этих продавцов?

7. Модели для банка. Какой процент обладателей текущих счетов банка составляют его служащие?

8. Модели для банка. Сколько кассиров имеют в банке сберегательные счета? Сколько менеджеров? Сколько кассиров не имеют таких счетов?

9. Модели для банка. Кто из менеджеров, имеющих в банке сберегательные счета, руководит служащими, имеющими в банке сберегательные счета?

10. Выведите концептуальную модель данных из следующего отчета

Консультационная служба				
Отчет о специализации консультантов				
Фамилия	№ страховки	Дата приема	Код специальности	Описание специальности
Иванов	539-88-4242	22/11/2000	A	Обучение пользователей
			B	Ввод данных
			O	Преобразование файлов
Петров	560-43-1111	8/11/1999	C	Программирование
			D	Преобразование файлов
			F	Системное проектирование
Сидоров	524-33-8119	7/3/1990	A	Обучение пользователей
			C	Программирование
			D	Системный анализ
			F	Системное проектирование

11. Сколько студентов занимается по программе Физика 201? Какие предметы изучает Иванов? Сколько раз Петров изучал Бухгалтерский Учет 201, когда и кто был его преподавателем и какие оценки он получил?

12. База данных должна давать ответы на вопросы по истории Европы. Создайте отдельную модель данных для указанной задачи.

Сколько королей Пруссии носили имя Фредерик? В какие годы они жили и в какие — правили? Управляли ли они на протяжении своей жизни какими-либо еще странами? Управлялись ли в XVII веке какие-либо европейские страны женщинами? Если да, то какие?

13. База данных должна давать ответы на вопросы по истории Европы.

Правил ли дед Марии-Антуанетты какой-либо страной? Какой и когда? Кто была ее мать? Были ли случаи, когда правители двух разных стран женились между собой? Сколько детей Генриха VIII стали королями Англии? Кто были их матери?

14. Какие станции транслируют программы «Бэтмэн»? Повторяла ли компания Brick Wall в этом году какие-либо серии Косби-шоу за 1988 год? Показывали ли они пятую серию? Когда и какая станция его транслировала?

15. Репортажи о скольких бейсбольных матчах Brick Wall показала за последний год? Когда они транслировали встречи между командами «Dodgers» и «Mets»? Матчи какой команды показывались больше всего? Как насчет футбольных матчей? Баскетбольных? Теннисных? Турниров по гольфу? Других видов спорта? Был ли показан хоть один теннисный матч с участием Штеффи Граф? Когда и какой станцией?

16. Какие коммерческие объявления Brick Wall показала более трех раз в течение одного часа на одной станции? Когда это было? В течение какого часа, какого числа и на какой станции? Какую плату Brick Wall назначила за трансляцию каждого из этих коммерческих сообщений?

17. Создайте модель базы данных, отвечающей на юридические вопросы. В каких делах высказывались мнения по Разделу 411.3с федерального кодекса? В каких судах? Были ли они отвергнуты? Какой раздел федерального кодекса был затронут в процессе Блэка против Вильямса?

18. Создайте модель базы данных, отвечающей на юридические вопросы. Какие юридические фирмы представляли General Continental в судах в течение последних десяти лет? Какие дела разбирались; какая сторона выиграла; каков был размер вознаграждения? Какие фирмы представляли противную сторону? Какие еще крупные компании представляли эти юридические фирмы в процессах в то же самое время?

19. В процессе работы над проектом для страховой компании один из аналитиков консультационной фирмы создал отчет, оценивающий производительность труда операторов, вводящих данные. Этот отчет выдает число транзакций каждого типа, введенных каждым клерком в каждый день месяца. Выведите концептуальную модель данных, которая могла бы быть использована в качестве основы для указанного отчета.

СТРАХОВАЯ КОМПАНИЯ				
МЕСЯЧНЫЙ ОТЧЕТ О ПРОИЗВОДИТЕЛЬНОСТИ ТРУДА СЛУЖАЩИХ				
За месяц по 31 марта				
Неслужащего	Фамилия	Дата	Тип транзакции	Количество
3855	Иванов	3/1	Новый полис	15
			Взнос	75
			Требование	22
		3/2	Новый полис	18

			Взнос	53
			Требование	25
3921	Петров	3/1	Взнос	45
			Изменение	83
			Требование	10
		3/2	Новый полис	8
			Взнос	63
			Изменение	35

20. Авиакомпания хочет получать ответы на подобные вопросы о своих самолетах: «Сколько посадочных мест в Боинге 727? Сколько у него двигателей? Какой средний возраст Боингов 727 нашего авиапарка? Кто главный механик, ответственный за обслуживание самолета номер 1388? Какая компания создала этот самолет?»

21. Администрация в большом городе должна отслеживать имеющееся у нее компьютерное оборудование. Она также хочет получать ответы на вопросы о моделях компьютеров. Создайте модель данных, отвечающую на следующие вопросы:

Какой максимальный объем памяти возможен у IBM PC? У PC-XT и PC-AT? Каков максимальный объем памяти у Macintosh II? У кого из наших служащих в кабинете есть IBM PC? У кого стоит компьютер с серийным номером 4538842? Какова его оперативная память?

22. Построить модель для ответов на вопросы по теме: “Издание учебной литературы для ВУЗов”, отвечающую на следующие вопросы: определить ВУЗы, в которых есть заданная специальность и определить план приема на указанную специальность по каждому ВУЗу. Какие учебники готовятся для студентов указанной специальности? Для каких специальностей может быть использовано в учебном процессе указанное издание? Определить размер тиража, если задается количество экземпляров издания на одного студента, когда оно включено в список основной и дополнительной литературы. Какое количество учебной литературы должно быть отправлено в указанный ВУЗ.

23. Построить модель для отображения результатов самостоятельной работы студента в течение семестра с целью организации учебного процесса, его совершенствования и выдачи справочной информации студенту и в управляющие инстанции, отвечающую на следующие вопросы: какой график выполнения самостоятельных работ по указанной дисциплине? Какие задания на самостоятельную работу у указанного студента? Как выполняет самостоятельные работы указанный студент? Какие студенты имеют результаты по указанной контрольной торчке?

24. Создать модель для помощи разработчику СУБД или лицу, которое использует СУБД, отвечающую на следующие вопросы: какие характеристики есть у указанной СУБД (вид используемой модели данных, вид техники, временные характеристики,

обеспечение секретности)? Какую литературу можно использовать при изучении указанной СУБД? В каких организациях внедрена и работает данная СУБД? Какие организации занимаются разработкой ПО с использованием указанной СУБД?

25. Сбербанк. Сведения о вкладчиках филиала Сбербанка: номер Лицевого счета, паспортные данные, сумма вклада, категория вклада, дата последней операции.

26. Сессия. Сводная ведомость группы по итогам сессии: фамилия с инициалами, номер зачетки, совокупность оценок по зачетам и экзаменам.

27. Склад. Инвентарная ведомость наличия товаров на складе: наименование товара, единица измерения, цена одной единицы, количество.

28. Магазин. Учетная ведомость наличия товаров в коммерческом магазине: наименование товара, количество (штук) - сдано и осталось, цена одной штуки.

29. Ломбард. Наименование каждого хранимого товара, оценочная стоимость; сумма, выданная сдатчику; дата сдачи, оговоренный срок хранения.

30. Коммерческий вестник. Наименование товара или услуги, единица измерения, количество, цена, наименование продавца, условия оплаты, условия поставки-отгрузки.

31. Автосалон. Марка предлагаемого автомобиля, цвет, технические характеристики, фирма-изготовитель, дата выпуска, пробег, цена.

32. Справочник аудиторий. Номер аудитории, корпус, вместимость, особенности (например, нет доски), спецоборудование (например, лингафонный кабинет).

33. Кадры сотрудников. Паспортные данные сотрудника, образование, степень, звание, специальность, подразделение, должность, оклад.

34. Кадры студентов. Паспортные данные, группа, адрес родителей, изучаемый язык, размер стипендии, наличие места в общежитии (или потребность в нем).

35. Расписание занятий. Номер группы, номер недели, день недели, номер пары и все данные о занятии на этой паре.

36. Каталог библиотеки. Инвентарный номер, выходные данные книги, цена, факт наличия или кому выдана.

37. Расписание автобусов. Номер маршрута, направление, время отправления, расстояние, марка автобуса, цена билета, количество проданных билетов.

38. Расписание самолетов. Номер рейса, время вылета и прибытия на конечный пункт, тип самолета, периодичность полетов, цена билета, пункты промежуточной посадки.

39. Домоуправление. Адрес, тип квартиры, площадь, степень благоустройства, форма собственности, отношение к капремонту и сносу, паспортные данные проживающих.

40. Горжилуправление. Адрес дома, количество квартир, общая и жилая площадь, год постройки, состояние, год последнего ремонта.
41. Справочник селекционера. Наименование сорта какой-либо культуры, автор, родительские сорта, урожайность, характеристики плодов, морозоустойчивость, устойчивость к вредителям и болезням, наличие в селекционном фонде.
42. Справочник ГАИ. Марка, цвет, заводской и бортовой номера, дата выпуска, дата последнего техосмотра транспортного средства, паспортные данные владельца.
43. Справочник автолюбителя. Марка, фирма, год начала серийного выпуска и технические характеристики различных автомобилей.
44. Справочник покупателя. Наименование и номер магазина, адрес, телефоны, характер специализации, форма собственности, средний объем товарооборота.
45. Справочник предприятий. Наименование, адрес, руководитель предприятия, выпускаемая продукция, потребляемое сырье, статус (форма собственности), численность работающих.
46. Справочник абитуриента. Наименование вуза, адрес, список факультетов, конкурс прошлого года по каждому факультету.
47. Справочник службы быта. Наименование фирмы, адрес, профиль (вид оказываемых услуг), статус (форма собственности), разряд (категория цен).
48. Справочник кутилы (рестораны и кафе). Название, адрес, наценочная категория, особенности кухни, часы работы, вид музыкально-эстрадной обслуживания.
49. Биржа труда-1. Справочник безработных: паспортные данные, профессия, образование, место и должность последней работы, причина увольнения, семейное положение, жилищные условия, адрес.
50. Биржа труда-2. Справочник вакансий: предприятие, должность, зарплата, жилищные условия, требования к специалисту.
51. Справочник знахаря. Наименование болезни, симптомы, возможные последствия, рекомендуемые лекарства, снадобья и процедуры.
52. Справочник видеомана. База видеофильмов: название, студия, жанр, год выпуска, режиссер, исполнители главных ролей, краткое содержание, субъективная оценка фильма.
53. Бюро знакомств. База клиентов: пол, возраст, рост, вес, знак Зодиака, материально-жилищное положение, профессия, хобби, требования к будущему партнеру.
54. Домашняя библиотека. Автор, название книги, год и место издания, раздел библиотеки (специальная литература, домашнее хозяйство, хобби, беллетристика и так далее), год и место приобретения.

55. Справочник фаната. База спортсменов: паспортные и антропологические данные, гражданство, происхождение, вид спорта, клуб или команда, данные о последнем рекорде или победах и так далее.

56. Справочник радиолюбителя. База паспортных данных транзисторов для других деталей: марка, характеристики, предельно допустимые условия эксплуатации, цена и так далее.

57. Записная книжка. Фамилии, адреса, телефоны знакомых и родственников, характер знакомства, дата рождения и так далее.

58. Справочник коммерческих банков. Наименование, адрес, статус (форма собственности), условия хранения средств на лицевом счете (годовые проценты на разных типах вклада).

59. Справочник начальника тюрьмы. Паспортные данные заключенных, статья, срок, дата заключения под стражу, место в тюремной иерархии, сведения о родственниках, особенности характера.

60. Справочник командира. Список подчиненных военнослужащих: паспортные данные, адрес родителей, гражданская профессия, звание и дата его получения, должность, подразделение, форма службы (срочная, кадровая, контрактная и так далее), период службы (для срочнослужащих), особенности характера и отношение к службе.

61. Картотека Интерпола. Данные по каждому зарегистрированному преступнику: фамилия, имя, кличка, рост, цвет волос и глаз, особые приметы, гражданство, место и дата рождения, последнее место жительства, знание языков, преступная профессия, последнее дело и так далее. Преступные и мафиозные группировки (данные о подельниках). Выборка по любому подмножеству признаков. Перенос "завязавших" в архив; удаление - только после смерти.

62. Справочник покупателя. База торговых точек города: название, адрес и телефоны, специализация, форма собственности, время работы. Выбор магазинов произвольному шаблону.

63. Генеалогическое дерево. Паспортные данные членов некоторого родового клана; ссылки на детей (или на родителей). Поиск всех потомков или всех предков для указанного лица.

64. Администратор гостиницы. База номеров: класс, число мест. База гостей: паспортные, данные, даты приезда и отъезда, номер. Поселение гостей: выбор подходящего номера (при наличии свободных мест), регистрация, оформление квитанции. Отъезд: выбор всех постояльцев, отъезжающих сегодня, освобождение места или оформление задержки с выпиской дополнительной квитанции.

65. Справочник меломана. База групп и исполнителей; база песен; база дисков с перечнем песен (в виде ссылок). Выбор всех песен заданной группы; всех дисков, где встречается заданная песня.

66. Ежедневник. База намечаемых мероприятий - дата, время, место проведения. Автоматическое напоминание ближайшего дела: по текущей дате и времени; удаление вчерашних дел либо перенос на будущее. Просмотр дел на завтра, послезавтра и так далее.

67. Терминология. База определений какой-либо науки: вводимый термин, его толкование (определение), ссылки на используемые термины. Возможность просмотра всей цепочки от заданного термина до первичных понятий.

68. Шеф-повар. База рецептов блюд: раскладка, рецепт приготовления. База продуктов на складе: наименование, цена, количество. Формирование меню на день (на заданное число персон). Проверка достаточности запасов; формирование расходной накладной на склад, корректировка запасов.

69. Справочник лекаря. База болезней: название, симптомы, процедуры, перечень рекомендуемых лекарств с указанием требуемого количества. База медикаментов на складе: название, количество, взаимозаменяемость. Формирование рецепта после осмотра больного, проверка наличия лекарств, корректировка запасов.

70. Справочник фирм. Название, адрес и телефоны, первое лицо, статус (форма собственности), сырье, продукция. Выбор по произвольному шаблону.

71. Обмен жилья. База предложений по обмену: район, площадь, планировка и так далее; требования к вариантам обмена. Регистрация клиентов, выбор подходящих вариантов, удаление при состоявшемся обмене или отказе.

72. Справочник почтовой индексации. Республика, область (край), район, населенный пункт, почтовый индекс. Поиск по любой совокупности полей (кроме последнего); иерархическая связь между полями (обратите внимание, что Новомосковск есть и в Тульской, и в Днепропетровской областях).

73. Купи-продай. База продавцов: наименование товара, объем партии при оптовой продаже, цена, условия продажи-отгрузки, форма оплаты, контактный адрес или телефон, примечание (например, "посредников прошу не беспокоиться"). База покупателей: наименование товара, объем покупки, приемлемая цена и форма оплаты, контактный адрес или телефон, примечание. Поиск и регистрация вариантов с той и другой стороны; формирование объявлений для печати, удаление в архив после купли-продажи (возможно, один из клиентов остается неудовлетворенным), полное удаление при отказе от услуг.

74. Успеваемость. База студентов: фамилия, имя, отчество, группа. База предметов: название, форма контроля (совокупность зачетов и экзаменов по семестрам). Ввод

результатов очередной сессии в сводную ведомость, пополнение/исправление после пересдач. Формирование списка задолжников.

75. Классификация до Дарвину- База растений и/или животных с указанием царства, класса, типа, семейства, рода, вида. Иерархическая организация классификации (например, просмотр только тех семейств, которые входят в данный тип). Отличительные признаки, по которым ведется классификация. Просмотр произвольной ветви дерева.

76. Рынок компьютеров. База фирм-продавцов: название, адрес, телефоны. База компьютеров с их характеристиками; база комплектующих и расходных материалов. Регистрация наличия на фирмах разных моделей, расходных и комплектующих с указанием цены (в рублях или СКВ). Корректировка данных по рекламным объявлениям. Поиск подходящих вариантов.

1.17 ТЕСТЫ

1. Совокупность процессов сбора, обработки, хранения, анализа и выдачи информации, необходимой для обеспечения управленческой деятельности и технологических процессов, называется:

- 1) информационной системой
- 2) информационным обеспечением
- 3) информационным процессом

2. Процессы создания, сбора, обработки, накопления, хранения, поиска, распространения и потребления информации - это

- 1) информационная система
- 2) информационное обеспечение
- 3) информационные процессы

3. Изменение объема и структуры знания воспринимающей системы - это:

- 1) общенаучная трактовка понятия "информация"
- 2) нормативно-правовая трактовка понятия "информация"
- 3) вероятностно-статистическая трактовка понятия "информация"

4. Связный текст, графические данные, анкеты - это:

- 1) неструктурированная форма данных
- 2) структурированная форма данных
- 3) нет правильного ответа

5. База данных - это

- 1) совокупность данных
- 2) совокупность данных, организованная в виде таблиц
- 3) совокупность данных, организованная с определенной целью

6. Система базы данных состоит из:

- 1) Базы данных
- 2) Базы данных и СУБД
- 3) Базы данных, СУБД и оборудования
- 4) Базы данных, СУБД, оборудования и людей
- 5) Базы данных, СУБД, оборудования, людей и помещения

Ответ на тест:

1-2; 2-3; 3-1; 4-3; 5-3; 6-4.

2 ПОСТРОЕНИЕ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ

Процесс определения требований и концептуального проектирования требует выяснения информационных требований пользователей и представления их в виде хорошо сконструированной модели. Для того чтобы выполнить эту задачу, необходимо внимательно рассмотреть природу требований пользователей и точное значение их логического представления.

2.1 РЕАЛЬНОСТЬ И МОДЕЛИ

Модель — это представление реальности, отражающее лишь избранные детали. Например, рассмотрим такую бухгалтерскую операцию, как внесение некоторой суммы на текущий счет. Учетный отдел банка желает сохранить некоторые детали (например, номер счета, сумма вклада, время, дата, номер кассира) и опустить некоторые другие (любезности, которыми кассир обменялся с клиентом, количество посетителей в банке, длина очереди, музыка, звучавшая по внутреннему радио, погода на улице и т.д.). реальность содержит мириады деталей, но учетный отдел сочтет большинство из них не имеющими отношения к операции. Таким образом, представление учетного отдела об операции будет содержать лишь те детали, которые будут признаны существенными.

Разумеется, некоторые детали, признанные несущественными одним пользователем, могут оказаться чрезвычайно важными для других пользователей. Представьте, например, что вы создаете базу данных для ресторана быстрого обслуживания. Погода может оказаться существенным аспектом реальности для менеджера, поскольку в холодный день продается совсем не то, что в жаркий. В результате менеджер решит отслеживать погодные изменения и соответствующий состав заказов. Длина очереди может оказаться еще одним важным аспектом реальности для менеджера, поскольку на основе этой информации он определит необходимое число работников на раздаче и минимизирует время ожидания для клиентов. Таким образом, у разных пользователей могут быть разные модели реальности.

База данных воплощает модель реальности. СУБД управляет базой данных, позволяя каждому пользователю записывать, извлекать и обрабатывать данные, составляющие модель реальности. Манипулируя данными различным образом, пользователи могут извлекать из них информацию, необходимую для успешной работы предприятия. Таким образом, модели являются мощным средством, помогающим избавиться от несущественных деталей и понять реальные требования конкретных пользователей.

Отобразить. Ассоциировать элементы из одной области с элементами другой области.

Моделирование реальности во многом похоже на решение ситуационной задачи. В

обоих случаях вам нужно просеять детали и создать «правильную» модель части реальности. Это означает, что вы должны ассоциировать или **отразить** элементы реальности в элементы модели. Если процесс отображения выполнен должным образом, то моделью можно воспользоваться для решения задачи. Если нет, то модель не может послужить источником правильного решения.

2.2 КРИТЕРИИ ОЦЕНКИ МОДЕЛИ ДАННЫХ

Оптимальная модель данных должна удовлетворять критериям, перечисленным в табл. 2.1. Однако иногда эти критерии несовместимы, поэтому приходится идти на некоторый компромисс.

Таблица 2.1. Критерии оценки модели данных

Критерий	Описание
Структурная достоверность	Соответствие способу определений и организации информации на данном предприятии
Простота	Удобство изучения модели как профессионалами в области разработки информационных систем, так и обычными пользователями
Выразительность	Способность представлять различия между данными, связи между данными и ограничения
Отсутствие избыточности	Исключение излишней информации, т.е. любая часть данных должна быть представлена только один раз
Способность к совместному использованию	Отсутствие принадлежности к какому-то особому приложению или технологии и, следовательно, возможность использования модели во многих приложениях и технологиях
Расширяемость	Способность развиваться и включать новые требования с минимальным воздействием на работу уже существующих приложений
Целостность	Согласованность со способом использования и управления информацией внутри предприятия
Схематическое представление	Возможность представления модели с помощью наглядных схематических обозначений

2.3 КОНЦЕПТУАЛЬНЫЕ МОДЕЛИ

Объектно-ориентированная модель. Модель, представляющая категории реального мира в виде объектов.

Семантическая модель. Модель, отражающая значения реальных категорий и отношений.

Методология моделирования данных, которую мы будем изучать и использовать, может быть названа **объектно-ориентированной**, поскольку она представляет компьютерное отображение категорий реального мира в виде «объектов», обладающих определенными «удостоверениями личности» и атрибутами и находящихся в некоторых отношениях, а не в виде записей традиционной файловой системы. Объектно-

ориентированное представление более точно отражает логическую сущность реальных приложений, чем представления, основанные на записях. По этой причине нашу методологию также можно назвать **семантической**, поскольку она позволяет эффективно отображать *значения* реальных вещей в конструкции модели.

Объектно-ориентированное или семантическое моделирование?

Объектно-ориентированные базы данных явились результатом сближения двух областей исследований: семантического моделирования данных и объектно-ориентированных языков. Эти области исследований развивались независимо, но в восьмидесятые годы начали сливаться, оказав существенное влияние на базы данных.

Семантическое моделирование данных изначально возникло с целью повышения эффективности и точности проектирования баз данных (Hull and King, 1987). Методы семантического моделирования оказались применимы ко многим пользовательским проблемам и легко преобразуемы в реализационные модели, основанные на записях: сетевые, иерархические и реляционные. Категорно-относительная (К-О) модель Чена стала наиболее популярной семантической моделью, о ней часто идет речь в книгах, посвященных моделированию данных и проектированию баз данных.

В то время как занимающиеся семантическим моделированием данных, в первую очередь уделяли внимание структуре данных, разработчики объектно-ориентированных языков программирования главным образом интересовались поведением объектов данных. Иными словами, они искали способы манипуляции данными, которые можно было бы сосредоточить в манипуляционных возможностях языка (запросах, вычислениях, обновлении данных). Структура данных рассматривалась во вторую очередь.

Сближение этих двух областей произошло, когда исследователи начали применять понятия объектно-ориентированных языков к семантическим структурам данных. В результате появилось понятие объектно-ориентированной базы данных. На этом стыке дисциплин преобладает объектно-ориентированная терминология, поэтому мы говорим об объектах, а не о категориях, как это принято в семантической терминологии. Кроме того, объектно-ориентированные языки выделяют несколько понятий, отсутствующих в исходной К-О модели Чена: «удостоверение личности» объекта, иерархия надобъектов и подобъектов, наследование. Таким образом, мы пользуемся методологией, скомбинированной из К-О модели Чена и понятий объектно-ориентированного моделирования. К-О модель формирует базис нашей концептуальной модели данных, а объектно-ориентированное моделирование вносит некоторые существенных усовершенствования.

2.4 КЛАССЫ И ОБЪЕКТЫ

Объект можно неформально определить как осязаемую реальность, проявляющую четко выделяемое поведение. С точки зрения восприятия человеком объектом может быть:

- * осязаемый и (или) видимый предмет;
- * нечто, воспринимаемое мышлением;
- * нечто, на что направлена мысль или действие.

Таким образом, мы расширили неформальное определение объекта новой идеей: **объект моделирует часть окружающей действительности и таким образом существует во времени и пространстве.** Термин *объект* в программном обеспечении впервые был введен в языке Simula и применялся для моделирования реальности.

Объектами реального мира не исчерпываются типы объектов, интересные при проектировании программных систем. Другие важные типы объектов вводятся на этапе проектирования, и их взаимодействие друг с другом служит механизмом отображения поведения более высокого уровня. Это приводит нас к более четкому определению: «Объект представляет собой конкретный опознаваемый предмет, единицу или сущность (реальную или абстрактную), имеющую четко определенное функциональное назначение в данной предметной области». В еще более общем плане *объект* может быть определен как нечто, имеющее четко очерченные границы.

Существуют такие объекты, для которых определены явные концептуальные границы, но сами объекты представляют собой неосязаемые события или процессы. Например, химический процесс на заводе можно трактовать как объект, так как он имеет четкую концептуальную границу, взаимодействует с другими объектами посредством упорядоченного и распределенного во времени набора операции и проявляет хорошо определенное поведение.

Объекты могут быть осязаемыми, но иметь размытые физические границы: реки, туман или толпы людей. Подобно тому, как взявший в руки молоток начинает видеть во всем окружающем только гвозди, проектировщик с объектно-ориентированным мышлением начинает воспринимать весь мир в виде объектов. Разумеется, такой взгляд несколько упрощен, так как существуют понятия, явно не являющиеся объектами. К их числу относятся атрибуты, такие, как время, красота, цвет, эмоции. Однако, потенциально все перечисленное - это свойства, присущие объектам.

Полезно понимать, что объект — это нечто, имеющее четко определенные границы, но этого недостаточно, чтобы отделить один объект от другого или дать оценку

качества абстракции. Можно дать следующее определение:

Объект обладает состоянием, поведением и идентичностью; структура и поведение схожих объектов определяет общий для них класс; термины “экземпляр класса” и “объект” взаимозаменяемы.

Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств.

К числу свойств объекта относятся присущие ему или приобретаемые им характеристики, черты, качества или способности, делающие данный объект самим собой. Например, для лифта характерным является то, что он сконструирован для поездок вверх и вниз, а не горизонтально. Перечень свойств объекта является, как правило, статическим, поскольку эти свойства составляют неизменяемую основу объекта. Мы говорим «как правило», потому что в ряде случаев состав свойств объекта может изменяться. Примером может служить робот с возможностью самообучения. Робот первоначально может рассматривать некоторое препятствие как статическое, а затем обнаруживает, что это дверь, которую можно открыть. В такой ситуации по мере получения новых знаний изменяется создаваемая роботом концептуальная модель мира.

Все свойства имеют некоторые значения. Эти значения могут быть простыми количественными характеристиками, а могут ссылаться на другой объект. Состояние лифта может описываться числом 3, означающим номер этажа, на котором лифт в данный момент находится. Состояние торгового автомата описывается в терминах других объектов, например, имеющихся в наличии напитков. Конкретные напитки — это самостоятельные объекты, отличные от торгового автомата (их можно пить, а автомат нет, и совершать с ними иные действия).

Таким образом, мы установили различие между объектами и простыми величинами: простые количественные характеристики (например, число 3) являются «постоянными, неизменными и непреходящими», тогда как объекты существуют во времени, изменяются, имеют внутреннее состояние, преходящи и могут создаваться, уничтожаться и разделяться.

Тот факт, что всякий объект имеет состояние, означает, что всякий объект занимает определенное пространство (физически или в памяти компьютера).

Что такое поведение. Объекты не существуют изолированно, а подвергаются воздействию или сами воздействуют на другие объекты.

Поведение - это то, как объект действует и реагирует; поведение выражается в терминах состояния объекта и передачи сообщений.

Иными словами, поведение объекта - это его наблюдаемая и проверяемая извне

деятельность. Операцией называется определенное воздействие одного объекта на другой с целью вызвать соответствующую реакцию.

Что такое класс?

Понятия класса и объекта настолько тесно связаны, что невозможно говорить об объекте безотносительно к его классу. Однако существует важное различие этих двух понятий. В то время как объект обозначает конкретную сущность, определенную во времени и в пространстве, класс определяет лишь абстракцию существенного в объекте. Таким образом, можно говорить о классе «Млекопитающие», который включает характеристики, общие для всех млекопитающих. Для указания на конкретного представителя млекопитающих необходимо сказать «это — млекопитающее» или «то — млекопитающее».

В общепонятных терминах можно дать следующее определение класса: группа, множество или вид с общими свойствами или общим свойством, разновидностями, отличиями по качеству, возможностями или условиями. В контексте объектно-ориентированного анализа дадим следующее определение класса:

Класс — это некое множество объектов, имеющих общую структуру и общее поведение.

Любой конкретный объект является просто экземпляром класса. Что же не является классом? Объект не является классом, хотя в дальнейшем мы увидим, что класс может быть объектом. Объекты, не связанные общностью структуры и поведения, нельзя объединить в класс, так как по определению они не связаны между собой ничем, кроме того, что все они объекты.

Важно отметить, что классы, как их понимают в большинстве существующих языков программирования, необходимы, но не достаточны для декомпозиции сложных систем. Некоторые абстракции так сложны, что не могут быть выражены в терминах простого описания класса. Например, на достаточно высоком уровне абстракции графический интерфейс пользователя, база данных или система учета как целое, это явные объекты, но не классы. Лучше считать их некими совокупностями (кластерами) сотрудничающих классов.

2.5 КЛАССИФИКАЦИЯ

Определение классов и объектов — одна из самых сложных задач объектно-ориентированного проектирования. Опыт показывает, что эта работа обычно содержит в себе элементы открытия и изобретения. С помощью открытий распознаются ключевые понятия и механизмы, которые образуют словарь предметной области. С помощью изобретения конструируются обобщенные понятия, а также новые механизмы, которые определяют

правила взаимодействия объектов. Поэтому открытие и изобретение — неотъемлемые части успешной классификации. Целью классификации является нахождение общих свойств объектов. При классификации, в одну группу объединяются объекты, имеющие одинаковое строение или одинаковое поведение.

Классификация затрагивает многие аспекты объектно-ориентированного проектирования. Она помогает определить иерархии обобщения, специализации и агрегации. Найдя общие формы взаимодействия объектов, вводятся механизмы, которые станут фундаментом реализации проекта.

2.5.1 Трудности классификации

Разумная классификация — трудная проблема. Разумная классификация — работа интеллектуальная и лучший способ ее ведения — последовательный, итеративный процесс. Это становится очевидным при анализе разработки таких программных продуктов, как графический интерфейс, стандарты баз данных и языки программирования четвертого поколения. В разработке программного обеспечения развитие какой-либо абстракции часто следует общей схеме. В начале проблема решается как-нибудь, для каждого частного случая. По мере накопления опыта некоторые решения оказываются более удачными, чем другие, и возникает род фольклора, переходящего от человека к человеку. Удачные решения изучаются более систематически, они программируются и анализируются. Это позволяет развить модели, осуществить их автоматическую реализацию, и разработать теорию, обобщающую найденное решение. Это в свою очередь поднимает практику на более высокий уровень и позволяет взяться за еще более сложную задачу, к которой, в свою очередь, мы подходим как-нибудь, тем самым начиная новый виток спирали.

Итеративный подход к классификации накладывает соответствующий отпечаток и на процедуру конструирования иерархии классов и объектов при разработке сложного программного обеспечения. На практике обычно за основу берется какая-то определенная структура классов, которую постепенно совершенствуют.

И только на поздней стадии разработки, когда уже получен некоторый опыт использования такой структуры, можно критически оценить качество получившейся классификации. Основываясь на полученном опыте, можно создать новый подкласс из уже существующих (вывод), или разделить большой класс на много маленьких (факторизация), или, наконец, слить несколько существующих в один (композиция). Возможно, в процессе разработки будут найдены новые общие свойства, ранее не замеченные, и можно определить новые классы (абстракция).

Сложность классификации можно объяснить это двумя причинами. Во-первых, отсутствием «совершенной» классификации, хотя, естественно, одни классификации

лучше других. Существует столько способов деления мира на объектные системы, сколько человек принимается за эту задачу. Любая классификация зависит от точки зрения субъекта. Во-вторых, разумная классификация требует изрядной доли творческого озарения. Иногда ответ очевиден, иногда он — дело вкуса, а бывает, что все зависит от умения заметить главное.

2.5.2 Идентификация классов и объектов. Классический и современный подходы.

Со времен Платона проблема классификации занимала умы бесчисленных философов, лингвистов, когнитивистов, математиков. Поэтому было бы разумно изучить накопленный опыт и применить его в объектно-ориентированном проектировании. Исторически известны только три подхода:

- классическая категоризация
- концептуальная кластеризация
- теория прототипов.

2.5.2.1 Классическая категоризация.

В классическом подходе все вещи, обладающие данным свойством или совокупностью свойств, формируют некоторую категорию. Причем наличие этих свойств является необходимым и достаточным условием, определяющим категорию. Например, холостые люди — это категория: каждый человек или холост, или женат, и этот признак достаточен для решения вопроса, к какой категории принадлежит тот или иной индивидуум. С другой стороны, высокие люди не определяют категории, если, конечно, мы специально не уточним критерий, позволяющий четко отличать высоких людей от невысоких.

Классическая категоризация пришла к нам от Платона и Аристотеля. Принципы классификации, предложенные Аристотелем нашли свое отражение в известной игре «Птица рыба-зверь».

Принципы классической категоризации отражены в современной теории развития ребенка. После первого года жизни ребенок осознает существование объектов и затем начинает приобретать навыки их классификации, вначале пользуясь базовыми категориями, такими, как собаки, кошки и игрушки. Позднее ребенок осознает, с одной стороны более общие (животные), а с другой стороны, более частные категории (колли, овчарки).

Таким образом, классический подход в качестве критерия похожести объектов использует родственность их свойств. В частности, объекты можно разбивать на непересекающиеся множества в зависимости от наличия или отсутствия некоторого признака. Лучшими являются такие наборы свойств, элементы которых мало взаимодействуют между собой. Этим объясняется всеобщая любовь к таким критериям

как размер, цвет, форма и материал. Так как эти критерии не пересекаются, про какой-нибудь предмет можно утверждать что он большой, серый, круглый и деревянный. Вообще говоря, свойства не обязательно должны быть измеряемыми, в качестве их можно использовать наблюдаемое поведение. То обстоятельство, что птицы летают, а рыбы нет, позволяет отличить орла от форели.

Какие конкретно свойства надо принимать во внимание? Это зависит от обстановки. Например, цвет автомобиля надо зафиксировать в задаче учета продукции автомобилестроительного завода, но он не интересен программе, управляющей уличным светофором. Вот почему нет абсолютного критерия классификации, одна и та же структура классов может подходить для одной задачи и не годиться для другой.

Пример с высокими и низкими людьми показывает, что классическая категоризация не всегда работает. Естественные категории не четко отграничены друг от друга. Большинство птиц летает, но не все. Стул может быть деревянным, металлическим или пластмассовым, а количество ног у него целиком зависит от прихоти конструктора. Практически невозможно перечислить определяющие свойства естественной категории, так, чтобы не было исключений. Это, действительно, коренные пороки классической категоризации, которые и попытались исправить в современных подходах.

2.5.2.2 Концептуальная кластеризация.

Это более современный вариант классического подхода. Он возник из попыток формального представления знаний. При таком подходе сначала формируются концептуальные описания классов (кластеров объектов), а затем классифицируются сущности в соответствии с этими описаниями. Например, возьмем понятие «патриотическая песня». Это именно понятие, а не признак или свойство, поскольку степень патриотичности песни едва ли можно измерить. Но если можно утверждать, что песня скорее про патриотизм, чем про что-то другое, то ее можно поместить в эту категорию.

Концептуальную кластеризацию можно связать с теорией нечетких (многозначных) множеств, в которой объект может принадлежать к нескольким категориям одновременно с разной степенью точности. Концептуальная кластеризация делает в классификации абсолютные суждения, основываясь на наилучшем согласии.

2.5.2.3 Теория прототипов.

Классическая категоризация и концептуальная кластеризация — достаточно выразительные методы, вполне пригодные для проектирования сложных программных систем. Но все же есть ситуации, в которых эти методы не работают. Рассмотрим более современный метод классификации, *теорию прототипов*.

Существуют некоторые абстракции, которые не имеют ни четких свойств, ни четкого определения. Существуют категории (например, игры), которые не соответствуют классически образцам, так как нет признаков, свойственных всем играм. По этой причине их можно объединить так называемой семейной схожестью. У категории игр нет четкой границы. Категорию можно расширить и включить новые виды игр при условии, что они напоминают уже известные игры. Вот почему этот подход называется *теорией прототипов*: класс определяется одним объектом-прототипом, и новый объект можно отнести к классу при условии, что он наделен существенным сходством с прототипом.

Рассмотрим классификацию на основе прототипов к проблеме стульев. Можно считать мягкий пуф, парикмахерское кресло и складной стул стульями не потому, что они удовлетворяют некоторому фиксированному набору признаков прототипа, но потому, что они имеют достаточное фамильное сходство с прототипом. Не требуется никакого общего набора свойств прототипа, которое годилось бы и для пуфика и для парикмахерского кресла, но они оба — стулья, так как каждый из них в отдельности похож на прототипный стул, пусть даже каждый по-своему. Свойства, определяемые при взаимодействии с объектом (свойства взаимодействия), являются главными при определении семейного сходства.

Понятие свойств взаимодействия — центральное для теории прототипов. В концептуальной кластеризации объекты группируются в соответствии с различными концепциями. В теории прототипов классификация объектов производится по степени их сходства с конкретным прототипом.

2.5.2.4 Применение классических и новых теорий.

Разработчику, озабоченному постоянно меняющимися требованиями к системе и вечно сражающемуся с напряженным планом при ограниченных ресурсах, предмет этого обсуждения может показаться далеким от реальности. В действительности, три рассмотренных подхода к классификации имеют непосредственное отношение к объектно-ориентированному проектированию.

На практике классы и объекты идентифицируются сначала по свойствам, важным в данной ситуации, то есть стараются выделить и отобрать структуры и типы поведения с помощью словаря предметной области. Потенциально возможных абстракций, как правило, очень много. Если таким путем не удалось построить приемлемой структуры

классов, пробуют концептуальный подход. В этом случае в центре внимания - поведение объектов, когда они взаимодействуют друг с другом. Наконец, пробуют выделить прототипы и ассоциировать с ними объекты.

Эти три способа классификации составляют теоретическую основу объектно-ориентированного подхода к выделению классов.

Рассмотрим несколько проверенных практикой подходов к выделению классов.

2.5.2.5 Классические подходы.

Разные ученые находят различные источники классов и объектов, согласующихся с требованиями предметной области. Эти подходы называются *классическими*, поскольку они опираются на классическую категоризацию.

Таблица 2.2

Осязаемые предметы	Автомобили, телеметрические данные, датчики давления
Роли	Мать, учитель, политик
События	Посадка, прерывание, запрос
Взаимодействие	Заем, встреча, пересечение

Таблица 2.3

Люди	Человеческие существа, выполняющие некоторые функции
Места	Области, связанные с людьми или предметами
Предметы	Осязаемый материальный объект или группа объектов
Организации	Формально организованная совокупность людей, ресурсов, оборудования, которая имеет определенную цель и существование которой в целом не за висит от индивидуумов
Концепции	Принципы и идеи, сами по себе неосязаемые, но предназначенные для организации деятельности и/или общения, или же для наблюдения за ними
События	Нечто случающееся с чем-то в заданное время или последовательно

Таблица 2.4

Структуры	Отношения «целое-часть» и «общее-частное»
Другие системы	Внешние системы, с которыми взаимодействует приложение
Устройства	Устройства, с которыми взаимодействует приложение
События	Происшествия, которые должны быть запомнены
Разыгрываемые роли	Роли, которые выполняют пользователи, работающие с приложением
Места	Здания, офисы и другие места, существенные для работы приложения
Организационные структуры	Группы, к которым принадлежат пользователи. Единицы

На более высоком уровне абстракции можно ввести понятие предметной области, которая в сущности является логически связанной группой классов, относящейся к

высокоуровневым функциям системы

2.5.2.6 Анализ поведения.

В то время как классические подходы концентрируют внимание на осязаемых элементах предметной области, другая школа сосредотачивается на динамическом поведении как на первоисточнике объектов и классов. Это напоминает концептуальную кластеризацию, рассмотренную выше: классы формируются, основываясь на группах объектов, демонстрирующих сходное поведение.

Можно предложить понятие ответственности объекта, под которыми следует понимать его знания и умения. Ответственность — это способ выразить цель объекта и его место в системе. Ответственность объекта есть совокупность всех услуг, которые он может предоставлять по всем его контрактам. То есть, вместе объединяются те объекты, которые имеют сходные ответственности и строится иерархия классов, в которой каждый подкласс, выполняя обязательства суперкласса, привносит свои дополнительные услуги.

2.5.2.7 Анализ предметной области.

Иногда в поисках полезных и уже доказавших свою работоспособность идей полезно обратиться сразу ко всем приложениям в рамках данной предметной области, как, например, ведение историй болезни пациентов, торговля ценными бумагами, разработка компиляторов или системы управления ракетами. Если возникли проблемы в середине разработки, анализ какой-нибудь узкой предметной области может помочь, указав на ключевые абстракции, оказавшиеся полезными в сходных системах. Выделим следующие этапы в анализе области:

- Построение скелетной модели предметной области при консультациях с экспертами в этой области.
- Изучение существующих в данной области систем и представление результатов в стандартном виде.
- Определение сходства и различий между системами при участии экспертов.
- Уточнение общей модели для приспособления к нуждам конкретной системы.

Анализ области можно вести относительно аналогичных приложений (вертикально) или относительно аналогичных частей одного и того же приложения (горизонтально). Например, начиная проектировать систему учета пациентов, имеет смысл рассмотреть уже имеющиеся подобные системы, чтобы понять, какие ключевые абстракции и механизмы, использованные в них, будут полезны, а какие нет. Аналогично система бухгалтерского учета должна представлять различные виды отчетов. Если считать отчеты некой предметной областью, ее анализ может привести разработчика к пониманию ключевых абстракций и механизмов, которые обслуживают все виды отчетов. Полученные таким образом классы и объекты представляют собой множество ключевых

абстракций и механизмов, отобранных с учетом цели исходной задачи: создания системы отчетов. Поэтому окончательный проект будет проще.

В роли эксперта часто выступает просто пользователь системы, например, инженер или диспетчер. Он не обязательно должен быть программистом, но должен быть близко знаком с исследуемой проблемой и разговаривать на языке этой проблемы.

Обычно для начального уяснения проблемы достаточно короткой встречи экспертов и разработчиков. Анализ прикладной области лучше всего вести шаг за шагом.

2.5.2.8 Анализ вариантов.

По отдельности классический подход, поведенческий подход и изучение предметной области, рассмотренные выше, сильно зависят от индивидуальных способностей и опыта аналитика. Для большинства реальных проектов одновременное применение всех трех подходов неприемлемо, так как процесс анализа становится недетерминированным и непредсказуемым.

Анализ вариантов — это подход, который можно успешно сочетать с первыми тремя, делая их применение более упорядоченным.

Коротко говоря, этот вид анализа можно начинать вместе с анализом требований. В этот момент пользователи, эксперты и разработчики перечисляют сценарии, наиболее существенные для работы системы (пока не углубляясь в детали). Затем они тщательно прорабатывают сценарии, раскладывая их по кадрам. При этом они устанавливают, какие объекты участвуют в сценарии, каковы обязанности каждого объекта и как они взаимодействуют в терминах операций. Тем самым группа разработчиков вынуждена четко распределить области влияния абстракций. Далее набор сценариев расширяется, чтобы учесть исключительные ситуации и вторичное поведение. В результате появляются новые или уточняются существующие абстракции.

2.5.2.9 Неформальное описание.

Радикальная альтернатива классическому анализу была предложена в чрезвычайно простом методе Аббота. Согласно этому методу надо описать задачу или ее часть на простом русском языке, а потом подчеркнуть существительные и глаголы. Существительные — кандидаты на роль классов, а глаголы могут стать именами операций. Подход Аббота полезен, так как он прост и заставляет разработчика заниматься словарем предметной области. Однако он весьма приблизителен и непригоден для сколько-нибудь сложных проблем. Человеческий язык — ужасно неточное средство выражения, потому список объектов и операций зависит от умения разработчика записывать свои мысли. Тем более, что для многих существительных можно найти соответствующую глагольную форму и наоборот.

2.5.3 Ключевые абстракции и механизмы

2.5.3.1 Поиск и выбор ключевых абстракций.

Ключевая абстракция — это класс или объект, который входит в словарь проблемной области. Самая главная ценность ключевых абстракций заключена в том, что они определяют границы нашей проблемы: выделяют то, что входит в нашу систему и поэтому важно для нас, и устраняют лишнее. Задача выделения таких абстракций специфична для проблемной области. Правильный выбор объектов зависит от назначения приложения и степени детальности обрабатываемой информации.

Определение ключевых абстракций включает в себя два процесса: открытие и изобретение. Разработчик открывает абстракции, слушая специалистов по предметной области: если эксперт про нее говорит, то эта абстракция обычно действительно важна. Изобретая, разработчик создает новые классы и объекты, не обязательно являющиеся частью предметной области, но полезные при проектировании или реализации системы.

Наиболее мощный способ выделения ключевых абстракций — сводить задачу к уже известным классам и объектам.

2.5.3.2 Уточнение ключевых абстракций.

Определив кандидатов на роли ключевых абстракций, разработчик должен оценить их по критериям. Определив новые абстракции, разработчик должен найти их место в контексте уже существующих классов и объектов. Не стоит пытаться делать это строго сверху вниз или снизу вверх. Нет особой необходимости строить иерархию классов, начиная с самого верхнего класса, и потом дополнять ее подклассами. Чаще создается несколько независимых иерархий, осознаются их общие черты и выделяются один или несколько суперклассов. Требуется несколько проходов вверх и вниз по иерархии, чтобы создать приемлемую модель.

Трудно сразу расположить классы и объекты на правильных уровнях абстракции. Иногда, найдя важный класс, можно передвинуть его вверх в иерархии классов, тем самым увеличивая степень повторности использования кода. Это называется *продвижением класса*. Программисты часто легкомысленно относятся к правильному наименованию классов и объектов, но на самом деле очень важно отразить в обозначении классов и объектов сущность описываемых ими предметов. При идентификации одного только объекта нужно придумать имена: для него и для его класса.

2.6 ОСНОВЫ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ

Главными элементами концептуальной модели данных являются *объекты* и *отношения*. Объекты часто представляют в виде *существительных*, а отношения — в виде *глаголов*.

2.6.1 Объекты

Объекты представляют вещи, которые пользователи считают важными в моделируемой нами части реальности. Примерами объектов могут быть люди, автомобили, деревья, дома, молотки и книги. Это конкретные объекты. Концептуальными объектами будут компании, навыки, организации, проекты товаров, деловые операции, штатное расписание.

Из предыдущего не ясно, называется ли объектом конкретная вещь (отдельный человек, конкретный автомобиль, конкретный банк) или *множество* вещей (все люди, все автомобили, все банки). Во избежание двусмысленности мы будем пользоваться термином *объектное множество (класс)* для обозначения множества вещей одного типа и объект-элемент для обозначения одного члена (одного элемента) объектного множества. Как показано на рисунке, будем изображать объектные множества в виде прямоугольников, а объекты-элементы — в виде точек. Имя объектного множества пишется заглавными буквами в единственном числе. Так «ЧЕЛОВЕК» — имя объектного множества, представляющего людей. Строчными буквами («человек») обозначается элемент из объектного множества ЧЕЛОВЕК. Мы пишем «человек в ЧЕЛОВЕК», чтобы обозначить, что человек является элементом объектного множества ЧЕЛОВЕК.

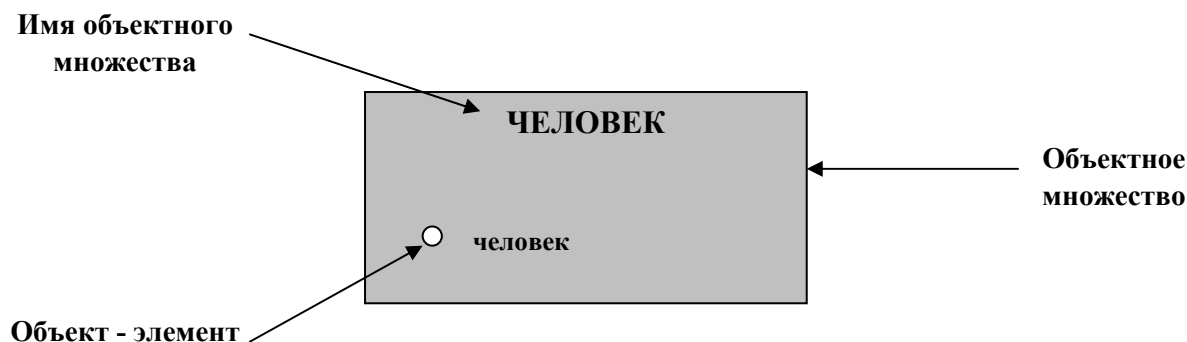


Рис. 2.1. Объектное множество

Объектные множества бывают **лексическими** и **абстрактными**. Элементы лексических объектных множеств можно напечатать, тогда как элементы абстрактных объектных множеств напечатать нельзя. Так, например, ИМЯ будет лексическим объектным множеством, поскольку его элементами являются имена, то есть строки символов, которые можно напечатать. ДАТА, КОЛИЧЕСТВО и НОМЕР-СТРАХОВКИ

также являются примерами лексических объектных множеств, так как даты, количества и номера страховок также можно распечатать.

Лексическое объектное множество. Объектное множество, состоящее из элементов, которые можно напечатать.

Абстрактное объектное множество. Объектное множество, состоящее из элементов, которые нельзя напечатать.

С другой стороны, ЧЕЛОВЕК является абстрактным объектным множеством, поскольку человека напечатать нельзя. Несмотря на то, что человека можно представить лексическим объектом, таким как имя или номер страховки, мы настаиваем на том, что человек *не есть* его имя или номер страховки. Например, имя или номер страховки может поменяться, но человек останется тем же самым. Таким образом, для того чтобы более точно моделировать реальность, мы будем различать абстрактные объектные множества и лексические.

В компьютерной реализации концептуальной модели элементы лексических объектов будут представлены в виде строк символов. Элементы абстрактных объектов будут представлены внутренними номерами, не имеющими смысла вне системы. Внутренний номер иногда называют *«идентификатор объекта»* или **суррогатным ключом**, так как он представляет и однозначно определяет абстрактный объект-элемент реального мира,

Суррогатный ключ. *«Идентификатор»* абстрактного объекта-элемента в компьютерной системе; вне системы смысла не имеет.

Допустим, что некто по имени Иванов Иван Иванович является элементом объектного множества ЧЕЛОВЕК. В компьютерной реализации этого объектного множества Иванов будет обозначаться некоторым суррогатным ключом, допустим, «13948226». Его имя (Иванов Иван Иванович), номер страховки, дата рождения, рост, вес и другая информация будет записана в виде лексических данных и связана в базе данных с суррогатным ключом, представляющим его. Пользователи увидят только эти лексические данные. Они никогда не встретят число 13948226 в связи с Ивановым. Но система будет пользоваться этим суррогатным ключом, ассоциированным с Ивановым, во всех возможных многочисленных отношениях в базе данных.

Суррогатные ключи решают проблемы, связанные с традиционными типами ключей. Например, во многих системах очень сложно изменить значение ключа. Номер страховки часто используется в качестве ключа для однозначного обращения к информации о человеке. Что происходит, если номер страховки введен неверно? Поскольку этот номер имеет важное значение *вне* системы, его *необходимо* исправить. Но это может привести к огромному количеству проблем в базе данных, так как на значение

номера страховки могут быть многочисленные ссылки. Этой проблемы можно избежать путем использования суррогатных ключей, поскольку они определяются системой и не имеют смысла вне ее. Если номер страховки Иванова Ивана Ивановича введен неверно, мы просто исправим его. Это изменение не повлияет больше ни на что в базе данных, так как больше ничего не ссылается на номер страховки. Вместо этого во всех ссылках на Иванова используется суррогатный ключ.

2.6.2 Конкретизация и обобщение

Некоторые объектные множества содержатся внутри других объектных множеств. Например, МУЖЧИНА (множество мужчин) содержится внутри множества ЧЕЛОВЕК. Это означает, что каждый мужчина (элемент множества МУЖЧИНА) является также человеком (элементом множества ЧЕЛОВЕК). Аналогично, множество ЖЕНЩИНА содержится внутри множества ЧЕЛОВЕК (ЧЕЛОВЕК). **Конкретизация** -объектное множество, являющееся подмножеством другого объектного множества.

Обобщение - объектное множество, являющееся надмножеством другого объектного множества (содержащее его).

ЧЕЛОВЕК, с другой стороны, является **обобщением** или надмножеством множества МУЖЧИНА (и множества ЖЕНЩИНА). Графическое изображение конкретизации/обобщения представлено на рисунке. U-образный символ обозначает направление включения. Верхняя часть U «открывается» в сторону большего или объемлющего множества. Если мы поместим множества рядом, то U будет лежать на боку, указывая на множество ЧЕЛОВЕК. Если сделать прямоугольник ЧЕЛОВЕК нижним, а МУЖЧИНА — верхним, то U нужно перевернуть. Мы также могли бы изобразить прямоугольник МУЖЧИНА внутри прямоугольника ЧЕЛОВЕК, но поскольку в одном объектном множестве может содержаться несколько других, такая диаграмма получится слишком загроможденной.



Рис. 2.2. Отношение конкретизации/обобщения

Представим себе мужчину по фамилии Иванов. Тогда Иванов является также человеком. Это представлено графически на рисунке 2.2.

Обратите внимание, что две точки обозначают одного и того же человека. Одна

точка представляет его как элемент множества ЧЕЛОВЕК, а вторая — как элемент множества МУЖЧИНА. На самом деле это один объект. Он просто показан принадлежащим двум разным объектным множествам. Мы вскоре покажем важность такого представления.

2.6.3 Отношения

Отношение. Связь между элементами двух объектных множеств.

Отношение связывает два объектных множества. Графически мы представляем отношение между двумя объектными множествами в виде соединяющего их отрезка (при желании дополненного ромбом).

Составное объектное множество. Отношение, рассматриваемое как объектное множество.

В качестве примера рассмотрим два множества служащих компании: ИНСПЕКТОР и РАБОЧИЙ. Мы определим элементы множества РАБОЧИЙ как тех служащих компании, которые не контролируют работу других служащих. Множество ИНСПЕКТОР состоит из тех служащих, которые контролируют рабочих. Отношение КОНТРОЛИРУЕТ (обратите внимание, что это глагол) связывает каждого инспектора с рабочими, которых он контролирует.

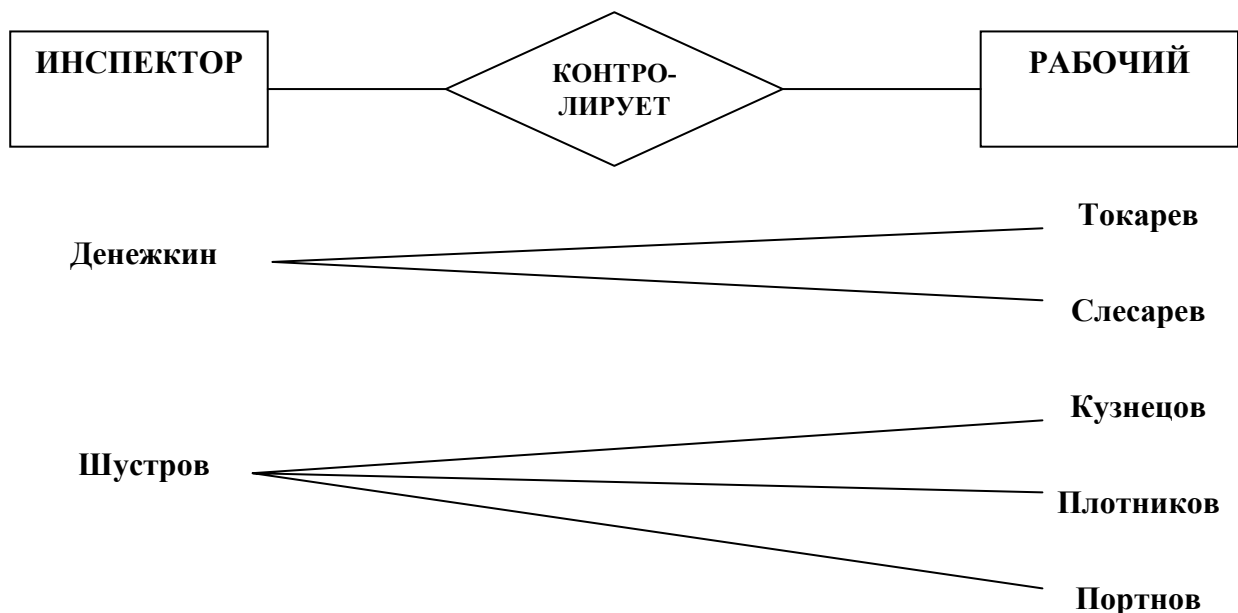


Рис. 2.3. Представление отношения КОНТРОЛИРУЕТ

2.6.4 Мощность

Мощность - максимальное количество элементов одного объектного множества, связанных с одним элементом другого объектного множества.

Хотя обычно интересует максимальная мощность, иногда полезно определять и минимальную мощность.

Некоторые отношения не имеют конкретного значения максимальной мощности. Например, инспектор контролирует как минимум одного рабочего, возможно, больше. Такую мощность обозначают $1,^*$, где «1» обозначает минимальную мощность, а «*» просто обозначает «много». С другой стороны, если допускается, что каждого данного рабочего контролирует один и только один инспектор, то мощность в обратном направлении будет $1,1$.

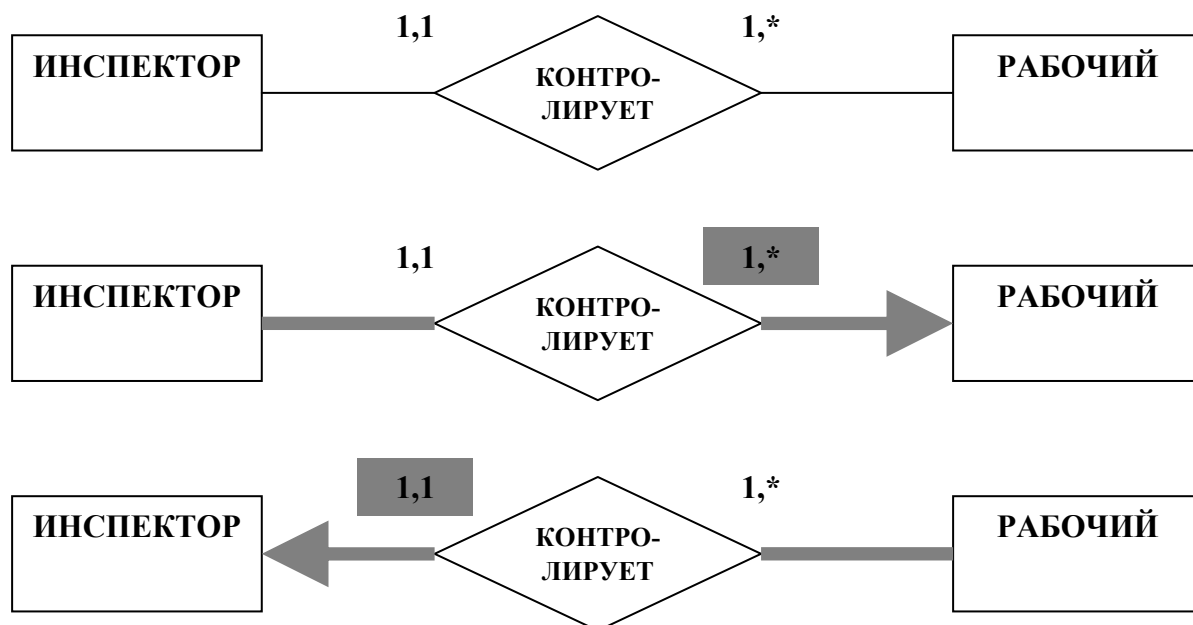


Рис.2 .4. Мощность отношения КОНТРОЛИРУЕТ

Данного рабочего контролирует ровно один инспектор

Мощность отношения конкретизации (или включения) всегда одна и та же. Каждый элемент объемлющего множества связан с одним или нулем элементов подмножества, а каждый элемент подмножества связан ровно с одним элементом объемлющего множества.



Рис. 2.5. Мощность отношения конкретизации/обобщения

Максимальная мощность является значительно более важным понятием, чем минимальная. Поэтому для упрощения диаграмм необходимо указывать минимальную мощность только тогда, когда это необходимо. За исключением отношения включения, опущенную минимальную мощность можно полагать равной нулю.

Максимальная мощность в одном из направлений, равная одному, соответствует

математическому понятию функции, которая устанавливает соответствие один-к-одному или много-к-одному между множествами. Поэтому отношение, имеющее максимальную мощность в одном из направлений, равную одному, называется **функциональным** в этом направлении. Отношение между рабочим и инспектором на рисунке является функциональным в направлении от рабочего к инспектору. Это означает, что, зная рабочего, мы можем однозначно определить его инспектора. Это отношение не является функциональным в обратном направлении, поскольку инспектор может контролировать нескольких рабочих.

Функциональное отношение. Отношение, максимальная мощность которого как минимум в одном направлении равна одному.

Если максимальная мощность отношения в обоих направлениях равна одному, то она называется отношением **один-к-одному**. Если максимальная мощность в одном направлении равна одному, а в другом — многим, то отношение называется отношением **один-ко-многим**. И, наконец, если максимальная мощность в обоих направлениях равна многим, то отношение называется отношением **много-ко-многим**. В таблице 2.2 приведены характеристики трех основных мощностей отношений.

Таблица 2.5. Три основные мощности отношений

Мощность	Обозначение	Пример
Один-к-одному	1:1 или 1-1	У мужа есть одна жена. У жены есть один муж (мощность отношения один-к-одному)
Один-ко-многим	1:* или 1-*	Служащий работает в одном отделе. В отделе работает много служащих (Мощность отношения один-ко-многим)
Много-ко-многим	*:* или * - *	Студент посещает много курсов. Курс слушает много студентов (Мощность отношения много-ко-многим)

2.6.5 Атрибуты

Атрибут. Функциональное отношение объектного множества с другим объектным множеством.

На самом деле, **атрибут** объекта — просто функциональное отношение объектного множества этого объекта к другому объектному множеству. Так, два атрибута представлены в виде отношений на рис. 2.6. Однако иногда удобно представлять атрибуты более простым образом, как на рис. 2.7.

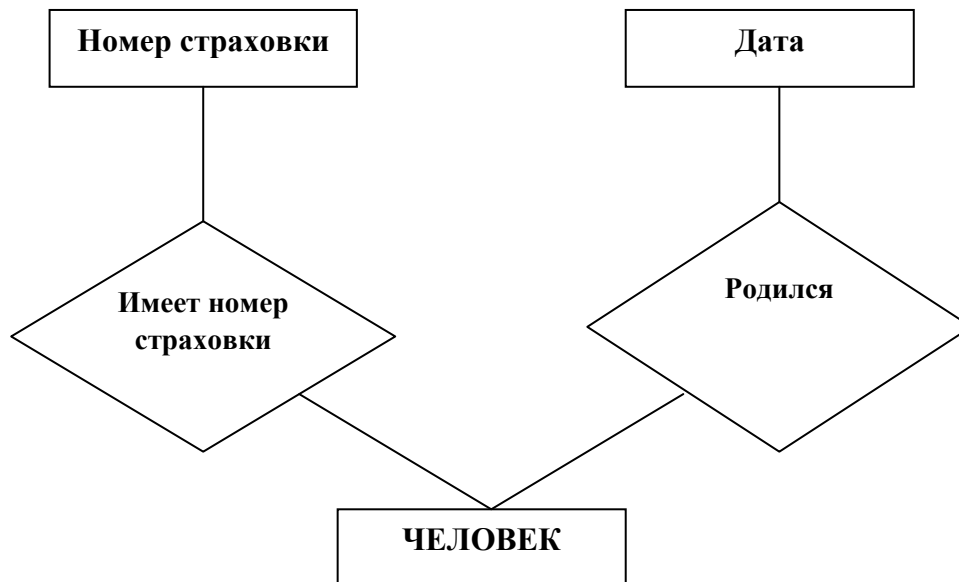


Рис. 2.6. Атрибуты, показанные как отношения

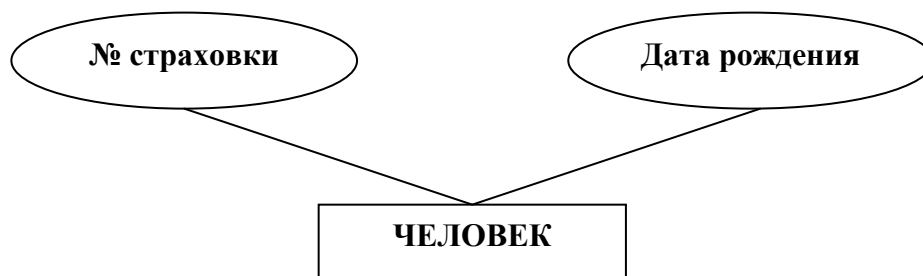


Рис. 2.7. Обозначение атрибутов

Существует еще более простой способ представления объектных множеств, представленный на рисунке 2.8.

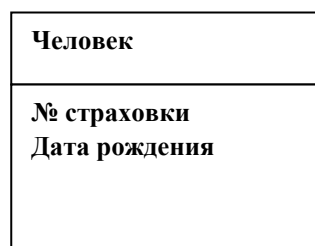


Рис. 2.8. Представление объектных множеств с указанием атрибутов

Обратите внимание, что имя отношения и имя объекта в некотором смысле комбинируются, давая вместе имя атрибута, особенно ДАТА-РОЖДЕНИЯ. Таким образом, мы видим, что такая запись атрибутов является просто краткой записью отношений. Обычно такой краткой записью можно пользоваться, если мы не собираемся использовать атрибут как объект, участвующий в других отношениях.

При нормальном использовании атрибуты являются функциональными отношениями в направлении от объекта к атрибуту. Это означает, что значение атрибута однозначно определено для каждого элемента объекта. Например, у каждого человека есть ровно одна дата рождения и ровно один номер страховки. Максимальная мощность

отношения со стороны атрибута в таком отношении всегда равна одному, поэтому ее можно опустить. Если для некоторого элемента объектного множества значение некоторого атрибута не определено, то этот атрибут имеет пустое значение для элемента объектного множества.

Важно понимать, что атрибуты нужно в концептуальном плане отделять от объектов, которые они описывают. Значения атрибутов могут часто меняться, в то время как описываемый ими объект остается тем же самым. Так, у человека может измениться рост, вес и цвет волос, но это будет тот же самый человек. Это не означает, что значения всех атрибутов меняются. В действительности, часто необходимо найти атрибуты, значения которых не меняются, поскольку их можно использовать в качестве ключей.

Каждый атрибут связан с набором значений, который называется *доменом*. Домен определяет все потенциальные значения, которые могут быть присвоены атрибуту. Например, количество комнат в объекте недвижимости может находиться в пределах от 1 до 15 для каждого экземпляра сущности. Следовательно, набор допустимых значений для атрибута `rooms` (Количество комнат) сущности `PropertyForRent` можно определить как набор целых чисел от 1 до 15.

Различные атрибуты могут совместно использовать один и тот же домен. Например, атрибуты `address` (Адрес) типов сущностей `Branch` (Отделение компании), `PrivateOwner` (Владелец объекта недвижимости) и `BusinessOwner` (Владелец делового предприятия) разделяют один и тот же домен, который включает все возможные адреса. Домены также могут представлять собой комбинацию, состоящую из нескольких других доменов. Например, домен для атрибута `address` сущности `Branch` состоит из таких подчиненных доменов, как `street`, `city` и `postcode`.

Домен атрибута `name` (Имя) определить труднее, поскольку он состоит из множества всех возможных имен. Очевидно, что это — символьная строка, но она может включать не только буквы, но также дефисы или другие специальные символы. Полностью разработанная модель данных включает домены каждого атрибута, присутствующего в ER-модели.

Атрибуты подразделяются на *простые* и *составные*, *однозначные* и *многозначные*, а также *производные*.

Простой атрибут - атрибут, состоящий из одного компонента с независимым существованием.

Простые атрибуты не могут быть разделены на более мелкие компоненты. Примером простых атрибутов является атрибут `position` (Должность) или `salary` (Зарплата) сущности `Staff`. Простые атрибуты иногда называют *элементарными*.

Составной атрибут – атрибут, состоящий из нескольких компонентов, каждый из

которых характеризуется независимым существованием.

Некоторые атрибуты могут быть разделены на более мелкие компоненты, которые характеризуются независимым существованием. Например, атрибут address (Адрес) сущности Branch, представляющей отделение компании, со значением '163 Main St, Glasgow, Gil 9QX' может быть разбит на отдельные атрибуты street('163 Main St'), city('Glasgow') и postcode('Gil 9QX').

Решение о моделировании атрибута address в виде простого атрибута или разбиении его на атрибуты street, city и postcode зависит от того, как рассматривается атрибут address в пользовательском представлении — как единое целое или как набор отдельных компонентов.

Однозначный атрибут – атрибут, который содержит одно значение для каждого экземпляра сущности определенного типа.

Большинство атрибутов являются однозначными. Например, для каждого отдельного экземпляра сущности Branch всегда имеется единственное значение в атрибуте номера отделения компании (branchNo), скажем, 'BOO3'. Поэтому атрибут branchNo является однозначным.

Многозначный атрибут – атрибут, который содержит несколько значений для каждого экземпляра сущности определенного типа.

Некоторые атрибуты могут иметь несколько значений для каждого экземпляра сущности. Например, сущность Branch может иметь несколько значений для атрибута telNo (Номер телефона отделения компании). Допустим, что отделение номер 'BOO3' имеет номера телефонов '095-339-2178' и '095-339-4439'. Следовательно, атрибут telNo в этом случае будет многозначным. Многозначный атрибут допускает присутствие определенного количества значений (возможно, в заданных пределах, определяющих максимальное и минимальное количество). Например, атрибут telNo отделения компании может иметь от одного до трех значений. Иными словами, любое отделение компании должно иметь не меньше одного номера телефона и не больше трех номеров телефонов.

Производный атрибут – атрибут, который представляет значение, производное от значения связанного с ним атрибута или некоторого множества атрибутов, принадлежащих некоторому (не обязательно данному) типу сущности.

Некоторые атрибуты могут быть связаны с определенной сущностью. Например, значение атрибута duration (Срок действия) сущности Lease (Договор аренды) вычисляется на основе атрибутов rentStart (Начало срока аренды) и rentFinish (Конец срока аренды), которые также относятся к типу сущности Lease. Атрибут duration является производным атрибутом, значение которого вычисляется на основании значений атрибутов rentStart и rentFinish.

В некоторых случаях значение атрибута является производным от многих экземпляров сущности одного и того же типа. Например, атрибут totalStaff (Общее количество Сотрудников) сущности типа Staff может быть вычислен на основе подсчета общего количества экземпляров сущности Staff.

Производные атрибуты могут также создаваться в форме ассоциаций атрибутов сущностей различных типов. Например, рассмотрим атрибут deposit (Задаток) сущности типа Lease. Значение атрибута deposit рассчитывается как удвоенное значение ежемесячной платы за аренду данного объекта недвижимости. Следовательно, значение атрибута deposit сущности Lease является производным от атрибута rent сущности типа PropertyForRent.

Ключ — это значение, которое однозначно определяет элемент объектного множества. В реализации концептуальной базы данных каждый человек из объектного множества ЧЕЛОВЕК получит суррогатный ключ для идентификации этого человека внутри базы данных.

Конкретизация/обобщение и атрибуты. Если объект является конкретизацией другого объекта, то тогда конкретизированный объект наследует все атрибуты и отношения обобщенного объекта. ЖЕНАТЫЙ ЧЕЛОВЕК, например, является конкретизацией объекта ЧЕЛОВЕК. Поэтому у состоящего в браке человека есть имя, номер страховки, адрес и т.д. просто потому, что он является человеком. Объект ЖЕНАТЫЙ ЧЕЛОВЕК наследует эти атрибуты от объекта человек. Кроме того, у конкретизированного объекта могут быть свои собственные атрибуты. Например, СУПРУГ будет атрибутом объекта ЖЕНАТЫЙ ЧЕЛОВЕК, но не объекта ЧЕЛОВЕК.

Наследование. Свойство объектного подмножества обладать всеми атрибутами объемлющего множества.

Конкретизированные объекты наследуют не только атрибуты, но и все отношения. ЧЕЛОВЕК связан с объектом КОМПАНИЯ отношением РАБОТАЕТ-В. ЖЕНАТЫЙ ЧЕЛОВЕК, будучи конкретизацией объекта ЧЕЛОВЕК, также связан с объектом КОМПАНИЯ отношением РАБОТАЕТ-В. Наследование атрибутов и отношений является важной идеей, поскольку она позволяет определять подмножества объектных множеств, обладающих своими собственными атрибутами и отношениями и сохраняющие все атрибуты и отношения объемлющего множества. Это дает возможность более точно моделировать реальность, чем без идеи наследования.

2.7 МОДЕЛИРОВАНИЕ КОНЦЕПТУАЛЬНЫХ И ФИЗИЧЕСКИХ ОБЪЕКТОВ

Хотя составные объекты и отношения высокого порядка являются очень полезными средствами решения широкого круга проблем моделирования данных, есть некоторые проблемы, достаточно сложные аспекты которых вполне можно решить базовыми методами.

Рассмотрим примеры **концептуальных и физических объектных множеств**. Например, ТИП МАТЕРИАЛА и ТИП БРИГАДЫ строительной компании «Премьер» примеры концептуальных объектных множеств, поскольку их элементы соответствуют *типам* предметов, а не конкретным физическим представителям этих типов. Типом материала будет словосочетание «доска 2x4x10 дюймов», а не конкретная такая доска. *Типом бригады* будет слово «кровельщики» или «электрики», тогда как конкретная *бригада* будет «кровельщики здания на ул. Анатолия, 200».

Концептуальное объектное множество. Объектное множество; элементами которого являются абстрактные понятия.

Физическое объектное множество. Объектное множество, элементами которого являются физические предметы.

Часто важно различать концептуальные объектные множества и **физические объектные множества**, которые им соответствуют, поскольку в одной и той же модели данных может оказаться необходимым представлять оба типа объектов. Проиллюстрируем это следующим примером.

Студент звонит в библиотеку и спрашивает:

СТУДЕНТ: У вас есть «Ночной дозор» Лукьяненко?

БИБЛИОТЕКАРЬ (вводит запрос в компьютерный каталог); Нет.

С: А «Дневной дозор»?

Б (вводит второй запрос): Нет.

С: А сколько всего у вас книг Лукьяненко?

Б (вводит третий допрос): Двенадцать.

С: Действительно? А какие?

Б: У нас есть «Сумеречный дозор», копия 1, «Сумеречный дозор», копия 2, «Сумеречный дозор», копия 3, и так далее до копии номер 12.

С: Но это же одна и та же книга! У вас не двенадцать книг Лукьяненко, а только одна.

Б: Нет, не одна и та же. Одна из них — издательства «Альфа-книга», другая — перевод на английский, третья — на французский, одна — подарочный вариант и так

далее.

С: Да, но на самом деле это одна и та же книга. Независимо от того, как она издана, это все равно «Сумеречный дозор». На самом деле у вас только одна книга Лукьяненко.

Этот диалог, никогда не мог состояться, поскольку ни один библиотекарь не станет так отвечать. Однако он служит для того чтобы указать на серьезную проблему естественного языка, которым люди пользуются при обычном общении. Что в этом примере подразумевается под *книгой*? Студент и библиотекарь подразумевают под книгой разные вещи. Один смысл слова «книга» (в котором его употребляет студент) — нечто концептуальное, имеющее множество разных физических версий. Таким образом, для студента «Сумеречный дозор» — это и в самом деле одна и та же книга, независимо от инвентарного номера, независимо от того, на английском она или на французском языке, полное это издание или сокращенное. Библиотекарь же употребляет слово «книга» (по крайней мере, сначала) в другом смысле: книга — это нечто физическое, что мы можем подержать в руках, пролистать и положить на полку. Библиотеке необходимо вести учет всех *физических* книг, которые в ней есть, независимо от того, первая это копия данной *концептуальной* книги или двенадцатая.

При проектировании базы данных должна быть возможность различать эти значения. В некоторых случаях пользователи будут ссылаться на *концептуальный объект*, который является абстрактной или обобщенной версией объекта. В других случаях пользователи будут ссылаться на *физический объект*, или конкретный элемент концептуального объекта. Если необходимо, чтобы база данных удовлетворяла потребности всех пользователей, то должно быть зафиксировано различие концептуального и физического. Нужно решить, на какие вопросы пользователи будут требовать ответов от системы. После того как будет выяснено, какая информация нужна, можно решать, как моделировать данные. В идеальном варианте нужно удовлетворить сторонников всех точек зрения, включая *и* студента, *и* библиотекаря.

2.8 ПРИМЕРЫ

2.8.1 Пример построения концептуальной модели по документу

Рассмотрим формирование концептуальной модели на основании документа. В качестве примера документа возьмем накладную.

НАКЛАДНАЯ № 234 от 20.01.2005

Продавец: ООО «Рога и копыта»

Покупатель: ООО «Минотавр»

№п/п	Товар	Ед	Кол-во	Цена	Сумма
1.	Рога	шт	23	456.34	10495.82
2.	Копыта	шт	5	34.58	172.90
	Итого				10668.72

В накладной можно выделить две части. Первая часть содержит общую информацию о накладной, такую как номер и дата накладной, а также продавец и покупатель.

НАКЛАДНАЯ № 234

от 20.01.2005

Продавец: *ООО «Рога и копыта»*

Покупатель: *ООО «Минотавр»*

Вторая часть содержит информацию о товарах, которые отпускаются по данной накладной.

№п/п	Товар	Ед	Кол-во	Цена	Сумма
1.	Рога	шт	23	456.34	10495.82
2.	Копыта	шт	5	34.58	172.90
	Итого				10668.72

Так как информация в рассмотренных частях содержится различная, то и для их представления надо будет использовать два класса. Назовем их «Шапка накладной» и «Строки накладной». Какие классы можно выделить дополнительно? В общей части накладной можно выделить продавца и покупателя. Так как продавец всегда должен быть один, то выделять для него отдельный класс нет необходимости. Поэтому выделим только класс «Покупатель». Но так как система обычно содержит информацию не только о накладных на продажу, то информация об организации-покупателе может быть востребована и в других местах. Поэтому данный класс лучше назвать «Организация» и использовать его везде, где фигурирует какая-либо организация, а не только в накладных.

Теперь рассмотрим список товаров. Здесь содержится следующая информация: «Номер по порядку», «Товар», «Единица измерения», «Количество», «Цена», «Сумма». «Номер по порядку» может являться свойством класса «Строки накладной», но обычно никого не интересует порядок следования строк в накладной и этот номер не храниться, а формируется при печати накладной. «Товар» в данном случае является одним из важнейших классов и обязательно должен быть в концептуальной модели. «Единица измерения» в зависимости от решаемой задачи может быть выделена в качестве класса, а может являться свойством класса «Товар». Для того чтобы построить модель, наиболее отвечающую формальным требованиям, выделим «Единицу измерения» в качестве класса. «Количество» будет свойством класса «Строки накладной», т.к. это простое значение, оно не может быть классом. Тоже можно сказать о «Цене». Но она может быть как свойством класса «Строки накладной», так и свойством класса «Товар». Куда отнести это свойство зависит от организации процесса продажи в данной фирме. Если каждый товар имеет фиксированную цену и по ней продается всем покупателям, то «Цена» должна быть свойством класса «Товар». Если один и тот же товар продается разным покупателям по разной цене, то ее надо зафиксировать и «Цена» будет свойством класса «Строки накладной», но при этом в классе «Товар» тоже может быть свойство «Цена»,

которое будет содержать рекомендуемую цену товара. Рассмотрим последний показатель «Сумма». Этот показатель вычисляемый и следовательно храниться не должен.

В заключение построим связи между классами и определим их мощность. Класс «Организация» должен быть связан с классом «Шапка накладной», т.к. этот показатель изначально фигурировал в верхней части накладной. Для одной организации может быть выписано много накладных, но накладная выписывается для одной организации. Следовательно мощность будет один-ко-многим. «Шапка накладной» связана с классом «Строки накладной», т.к. только вместе они описывают представленный документ. С одним объектом класса «Шапка накладной» может быть связано несколько объектов класса «Строки накладной». Строка накладной не может относиться к двум различным накладным. Следовательно мощность связи между классами «Шапка накладной» и «Строки накладной» будет один-ко-многим. Класс «Товар» связан с классом «Строки накладной». При этом один товар может фигурировать во многих строках, а в строке может быть только один товар. Следовательно мощность будет один-ко-многим. Точно также связаны классы «Единица измерения» и «Товар». Полученная модель представлена на рисунке 2.9.

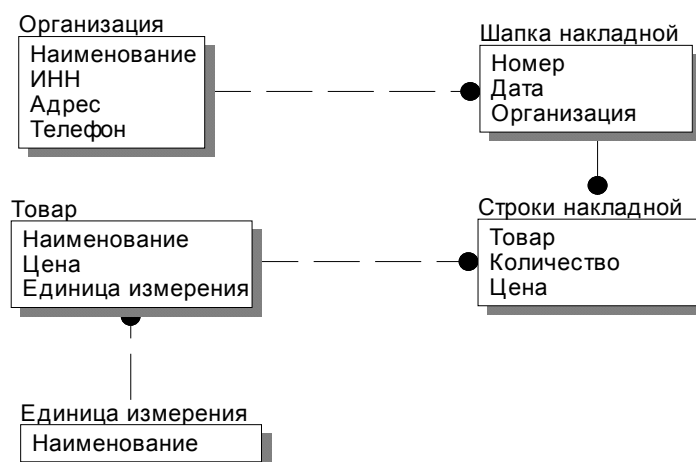


Рис. 2.9. Пример концептуальной модели, описывающей накладную

2.8.2 Пример построения концептуальной модели по описанию предметной области

В качестве примера рассмотрим некоторую компанию, которая занимается продажей пищевых продуктов. Ниже приводится примерная информация, доступная разработчику подобных программ.

Компания импортирует определенные виды пищевых товаров и обеспечивает ими специализированные магазины по всему миру.

Типичный процесс деятельности такого дистрибьютора товаров приведен ниже:

- 1. На складе создаются запасы товаров.**
- 2. Клиент звонит в отдел продаж.**
- 3. Клиент делает заказ на определенные виды товаров.**

4. Заказ принимается и передается в отдел перевозок.

5. В отделе перевозок заказ комплектуется, упаковывается и отправляется заказчику.

6. По мере необходимости производится пополнение запасов на складе.

Рассмотрим работу аналитика над нашим проектом. Перед началом работы он должен составить для себя список основных задач:

- Составить список всех абстрактных существительных, применяемых для описания системы
- Повторно рассмотреть составленный список, выделив в нем возможные классы
- Перечислить свойства каждого класса
- Объединяя классы, составить эскиз системы
- Встретиться с руководством фирмы, уточнить и пополнить информацию
- Дополнить полученной информацией свойства классов
- Разработать окончательную модель системы

Программист-аналитик встречается со следующим списком предметов, характеризующих систему и являющихся кандидатами на роль классов:

- Компания
- Центральный офис
- Склад
- Товар
- Продавец
- Служащий отдела доставки
- Заказы
- Клиенты
- Поставщик
- Работник
- Начальник
- Количество товара
- Цена

Приведенный выше список можно сократить по следующим причинам:

- Склад и Центральный офис являются просто местами положения и поэтому не имеют отношения к классам.
- Количество товара представляет собой число единиц товара и может быть использовано как одно из свойств **Товара**.
- Аналогично, **Цена** является не более чем свойством **Товара**.

- **Начальник**, возможно, будет использоваться как **свойство Работника**.

После устранения "лишних" классов можно начинать поиск различий между оставшимися. Сразу видны две группы: **Работник и Корпорация. Продавец и Служащий** отдела доставки относятся к **Работнику** как подклассы. **Поставщик, Компания и Клиент** являются по сути своей корпорациями.

- После определения основных классов требуется более подробная информация о каждом из классов.

Для каждого класса нужно определить свойства. Значимым свойством клиента (можно даже сказать, определяющим) является имя его компании. Что делает клиент? Он звонит продавцу, делает заказ, состоящий из одного или более товаров, и затем получает этот заказ от поставщика.

Определяющим свойством класса **Продавец** является опять-таки имя. Продавец получает заказ от клиента, записывает его, и затем отправляет в отдел доставки.

Служащий отдела доставки определяется своим именем. Он получает заказ, комплектует его из имеющихся товаров и отправляет перевозчику.

Перевозчик, характеризуемый именем, получает заказ от служащего отдела доставки и доставляет клиенту.

Товар определяется названием или кодом товара, а также именем компании, которая его поставляет. Он заказывается покупателем, выбирается служащим отдела доставки и помещается в заказ.

Поставщик товара характеризуется именем, получает заказы на партии товара от **Компании** и доставляет ей товар.

Определив свойства классов, довольно легко определить и взаимосвязи между ними.

В результате описанных выше действий получается эскиз системы, ее черновой набросок. Он представляет собой информацию, доступную аналитику на текущий момент, и является основой для дальнейшей деятельности. На данном этапе аналитик может задать множество вопросов. Например, как организация определяет такую вещь, как товар? Эскиз также отражает правильность понимания аналитиком основных проблем. Он, однако, должен быть дополнен информацией об основных правилах ведения бизнеса и о способах решения конфликтных ситуаций.

Рассмотрим повторно модель нашего предприятия.

Дополнение деталей к проекту. Некоторая дополнительно полученная информация просто позволяет добавить детали, полный перечень которых будет составлен только на этапе разработки системы. Так, заказ должен характеризоваться названием компании и датой. Заказ создается продавцом, а комплектуется в отделе

продаж. Класс **Корпорации** должен быть дополнен свойствами адреса, контактного лица, телефона, факса и т.д. Класс **Работник** получает свойства фотографии работника и различных заметок. Для **Товара** нужны такие свойства, как цена, единица, поле для иностранных языков, признак прекращения поставки.

Добавление свойств, отражающих правила ведения бизнеса. В некоторых ситуациях понимание аналитиком правил ведения бизнеса выражается в добавлении к классам дополнительных свойств. Так, из разговоров с работниками компании можно выяснить, что требуется поддержка некоторого минимального количества товара на складе, при достижении которого требуется пополнение запасов (то есть составление заказа поставщику). Поэтому к классу **Товар** нужно добавить свойство "минимальное количество".

Класс **Клиентов** получает свойства максимального и минимального объема покупок, а также скидки. Соответственно, класс **Заказов** дополняется методами применения скидки клиента к заказу и проверки на максимум и минимум заказанных товаров.

Добавление новых классов. Аналитик также должен рассмотреть новые классы, которые ранее были несущественны. Определить, является ли новое свойство отдельным классом, можно следующим способом. Свойство можно сделать отдельным классом, если оно имеет отношение один-ко-многим с существующими классами. Например, аналитик обнаруживает, что товары классифицируются по категориям. Отдельная категория, например, "консервированные фрукты" относится ко многим конкретным товарам. В этом случае категория становится отдельным классом, а товар с ней связывается отношением: "Каждый товар имеет свою категорию"

Стало очевидно, что компании требуется система безопасности с уровнями доступа. Поэтому к классу **Работников** было добавлено свойство "уровень доступа". Но, поскольку оно связано с работником отношением один-ко-многим, его можно также выделить в отдельный класс.

Кроме того у **Заказа** должны быть **Строки**, которые указывают какой **Товар** включен в **Заказ**.

Рассмотрим полученную концептуальную модель, представленную на рисунке 2.10. Эта модель построена с учетом того, что при дальнейшей работе нам придется отказаться от иерархии классов, поэтому свойства базовых классов были перенесены в производные или в базовый класс добавлены свойства всех производных.



Рис. 2.10. Пример концептуальной модели для компании, которая занимается продажей пищевых продуктов.

2.9 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет собой модель?
2. Что подразумевается под действием «отобразить»?
3. Перечислите основные критерии оценки модели данных.
4. Сформулируйте определение концептуальной модели.
5. Для чего необходимо выделять классы и объекты?
6. Приведите примеры классификаций.
7. В чем отличие лексического объектного множества от абстрактного?
8. Что такое «конкретизация» и «обобщение»?
9. Дайте определение отношения, мощности отношения и приведите примеры различных отношений.
10. Что вы понимаете под атрибутом?
11. В чем состоит отличие концептуального объектного множества от физического?

2.10 УПРАЖНЕНИЯ

Задание: построить концептуальную модель для предметной области «Зоопарк».

Из описания, представленного в первой главе, можно выделить следующие классы: должность, служащий, животное, сведения о прививках, прививки, местожительство, рацион, корм, поставщик, класс, вид, совместимость. Это не полный набор классов, но приведенный список моделирует основную часть предметной области. Для

окончательного построения модели необходимо определить атрибуты и связи между классами. В результате получаем модель, представленную на рисунке 2.11.

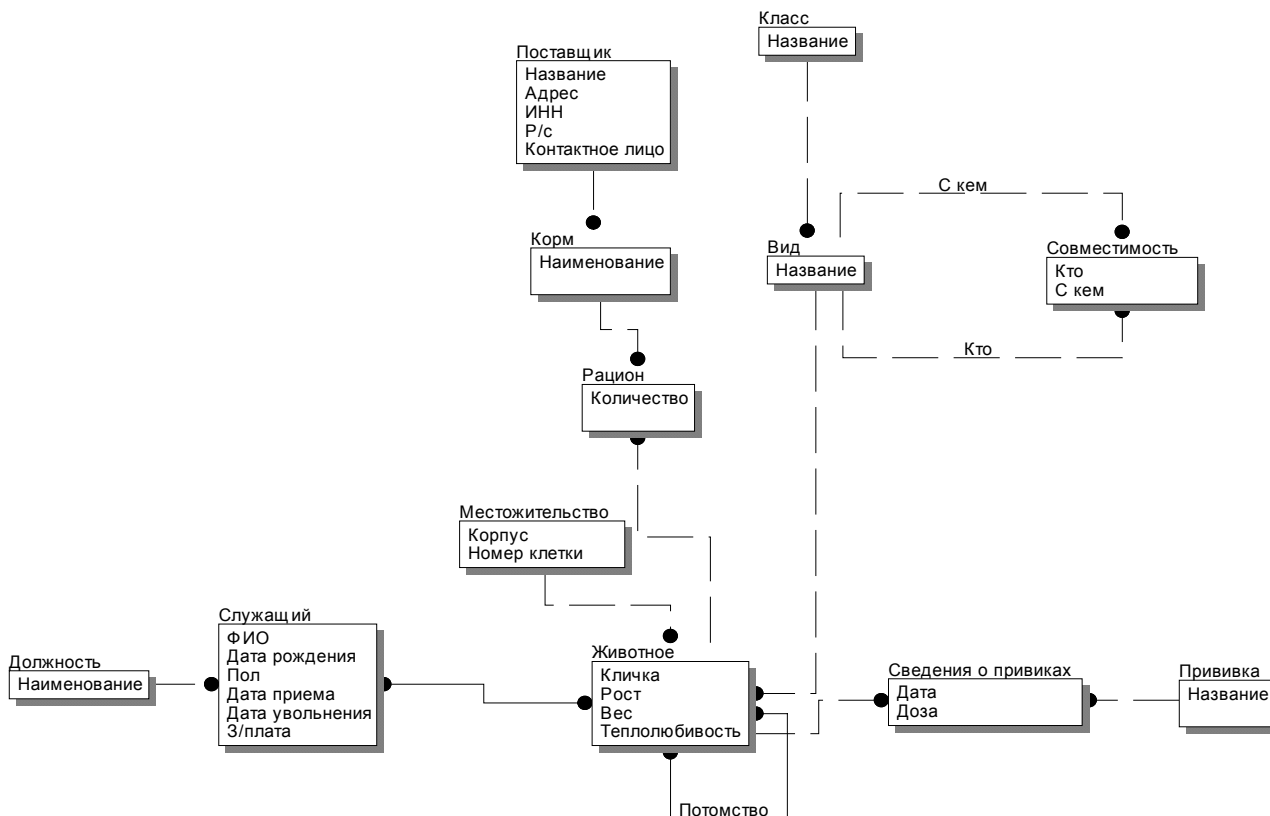


Рис. 2.11. Концептуальная модель, описывающая зоопарк.

Задание: построить концептуальную модель для предметной области, описанной в первом модуле.

2.11 ТЕСТЫ

1. Главными элементами концептуальной модели являются

- 1) классы
- 2) объекты
- 3) объекты и отношения
- 4) концепции
- 5) множества

2. Класс - это

- 1) Некое множество объектов
- 2) Некое множество объектов, имеющих общую структуру
- 3) Некое множество объектов, имеющих общее поведение
- 4) Некое множество объектов, имеющих общую структуру и поведение

3. Объектные множества бывают

- 1) лексическими
- 2) абстрактными

- 3) семантическими
- 4) лексическими и абстрактными
- 5) лексическими и семантическими

4. Укажите число уровней в архитектуре БД

- 1) 1
- 2) 2
- 3) 3
- 4) 4
- 5) 5

5. Мощность - это

- 1. Максимальное количество элементов одного объектного множества, связанных с одним элементом другого объектного множества.
- 2. Максимальное количество элементов одного объектного множества, связанных с максимальным количеством элементов другого объектного множества.
- 3. Минимальное количество элементов одного объектного множества, связанных с максимальным количеством элементов другого множества.
- 4. Нет правильного ответа

6. Мощность один-к-одному означает, что

- 1) Мин. мощность отношения в обоих направлениях равна одному
- 2) Макс. мощность отношения в обоих направлениях равна одному
- 3) Макс. мощность отношения в одном из направлений равна одному
- 4) Нет правильного ответа.

1-3;

2-4;

3-4;

4-3;

5-1;

6-2.

3 РЕЛЯЦИОННАЯ МОДЕЛЬ

3.1 ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННОЙ МОДЕЛИ

3.1.1 Основные определения

Реляционная модель основана на математическом понятии *отношения*, физическим представлением которого является таблица. Реляционная модель предусматривает организацию данных исключительно в виде таблиц. Для потребителя, проектировщика БД, администратора БД и конечного пользователя данные представляются совершенно одинаково — в виде таблиц. Следовательно, таблица — базовый языковый объект реляционной модели.

Таблица представляет собой множество именованных *атрибутов*, или столбцов, и множество *записей* (*кортежей*), или строк. Очень часто столбец называется *полем* таблицы (*field*). Пересечение строки и столбца образуют *ячейку* таблицы (*cell*). Набор допустимых значений столбца — *домен* (*domain*) — характеризуется определенным типом данных, например *символьным* или *целым*. Строки же и представляют собой данные. Например:

Производитель	Модель	Цена
Toyota	Camry	\$25000
Honda	Accord	\$23000
Ford	Taurus	\$20000
Volkswagen	Passat	\$20000

Существует различная терминология описания основных понятий реляционной модели. Соответствие между альтернативными вариантами терминов и объектной моделью приведено в таблице 3.1

Таблица 3.1. Альтернативные варианты терминов в реляционной модели и их соответствие объектной модели.

Официальные термины	Альтернативный вариант1	Альтернативный вариант 2	Объектная модель
Отношение	Таблица	Файл	Класс
Кортеж	Строка	Запись	Объект
Атрибут	Столбец	Поле	Свойство

Реляционная модель предъявляет к таблице определенные требования.

- Данные в ячейках таблицы должны быть структурно неделимыми. Каждая ячейка может содержать *только одну* порцию данных. Это свойство часто определяется как *принцип информационной неделимости*. Недопустимо, чтобы в ячейке таблицы реляционной модели содержалось более одной порции данных, что иногда именуется *информационным кодированием* (*information coding*). Примером может служить идентификационный номер транспортного средства (Vehicle Identification Number — VIN). Если записать его в одну ячейку, то будет нарушен принцип неделимости информации,

поскольку в ячейке окажутся такие *разделяемые* данные, как наименование производителя, модели, сведения о местонахождении предприятия-изготовителя и т.д. Хотя на практике решение о том, следовать ли жестко требованиям теории, почти всегда принимает проектировщик, нужно учитывать, что нарушение этих требований может затем отрицательно сказаться на обеспечении целостности данных.

- Данные в одном столбце должны быть одного типа.
- Каждый столбец должен быть уникальным (недопустимо дублирование столбцов).
- Столбцы размещаются в произвольном порядке.
- Строки (записи) размещаются в таблице также в произвольном порядке.
- Столбцы имеют уникальные наименования.

Помимо собственно таблиц и их свойств, реляционная модель имеет еще и собственные операции. Эти операции позволяют выполнять некоторые действия над подмножествами столбцов и/или строк, объединять (сливать) таблицы и выполнять другие операции над множествами. Чрезвычайно важно отметить, что все эти операции используют таблицы в качестве исходных данных; результатом операций также являются таблицы. В настоящее время стандартным языком для работы СУБД, является SQL, средства которого позволяют выполнять все эти операции.

В таблице не должно быть повторяющихся строк. Поэтому необходимо иметь возможность уникальной идентификации каждой отдельной строки таблицы по значениям одного или нескольких столбцов (называемых реляционными ключами). Рассмотрим терминологию, используемую для обозначения реляционных ключей.

Суперключ – столбец или множество столбцов, которое единственным образом идентифицирует строку данной таблицы.

Суперключ однозначно обозначает каждую строку в таблице. Но суперключ может содержать дополнительные столбцы, которые необязательны для уникальной идентификации строки, поэтому рассмотрим суперключи, состоящие только из тех столбцов, которые действительно необходимы для уникальной идентификации строк.

Потенциальный ключ – суперключ, который не содержит подмножества, также являющегося суперключом данного отношения.

Потенциальный ключ К для данной таблицы R обладает двумя свойствами.

- **Уникальность.** В каждой строке таблицы R значение ключа К единственным образом идентифицирует эту строку.
- **Неприводимость.** Никакое допустимое подмножество ключа К не обладает свойством уникальности.

Таблица может содержать несколько потенциальных ключей. Если ключ состоит из нескольких столбцов, то он называется составным ключом. Для идентификации потенциального ключа требуется знать смысл используемых столбцов в «реальном мире»; только это позволит обоснованно принять решение о возможности существования значений-дубликатов.

Первичный ключ — потенциальный ключ, который выбран для уникальной идентификации строк внутри таблицы.

Поскольку таблица не содержит строк-дубликатов, всегда можно уникальным образом идентифицировать каждую ее строку. Это значит, что таблица всегда имеет первичный ключ. В худшем случае все множество столбцов может использоваться как первичный ключ, но обычно, чтобы различить строки, достаточно использовать несколько меньшее подмножество столбцов. Потенциальные ключи, которые не выбраны в качестве первичного ключа, называются *альтернативными ключами*.

Чаще всего в реальных системах в качестве первичного ключа используют суррогатный ключ. Это позволяет избежать проблем, связанных с изменением значения первичного ключа.

Пустое значение — указывает, что значение столбца в настоящий момент неизвестно или неприемлемо для этой строки.

Пустое значение (которое условно обозначается как NULL) следует рассматривать как логическую величину «неизвестно». NULL не следует понимать как нулевое численное значение или заполненную пробелами текстовую строку. Нули и пробелы представляют собой некоторые значения, тогда как ключевое слово NULL обозначает отсутствие какого-либо значения.

Внешний ключ — это столбец или подмножество столбцов одной таблицы, который может служить в качестве первичного ключа для другой таблицы. Говорят также, что внешний ключ одной таблицы является *ссылкой* на первичный ключ другой таблицы.

И последнее, на чем необходимо остановиться в этом кратком обзоре свойств реляционной модели, — два фундаментальных правила целостности: правило *целостности объектов* (*entity integrity rule*) и правило *ссылочной целостности* (*referential integrity rule*).

Правило целостности объектов утверждает, что первичный ключ не может быть полностью или частично пустым, т.е. иметь значение NULL.

Правило ссылочной целостности гласит, что внешний ключ может быть либо пустым (иметь значение NULL), либо соответствовать значению первичного ключа, на который он ссылается.

Таким образом, система управления реляционной БД — это СУБД, которая удовлетворяет сформулированным выше фундаментальным требованиям реляционной модели. Однако с реляционными СУБД, разработанными и введенными в эксплуатацию в конце 70-ых — начале 80-ых годов, произошло следующее: была предпринята попытка внедрить средства языка SQL в *нереляционные* системы, а затем их попросту назвали реляционными. Стали они реляционными или нет — здесь критерием могут служить двенадцать правил полноты реляционных систем, сформулированных Коддом, за которыми так и закрепилось наименование *правил Кодда*.

3.1.2 Правила Кодда

Двенадцать правил Кодда определяют требования к реляционным СУБД.

1. **Явное представление данных.** *Информация должна быть представлена в виде данных, хранящихся в ячейках.* Как упоминалось выше, идентификационный номер транспортного средства VIN как содержимое единого столбца противоречит этому правилу.

2. **Гарантированный доступ к данным.** *К каждому элементу данных должен быть обеспечен доступ с использованием комбинации имени таблицы, первичного ключа строки и имени столбца.* Например, возможность обратиться к столбцу, используя массивы или указатели, противоречит этому правилу.

3. **Полная обработка неопределенных значений.** *Неопределенные значения Null, отличные от любого определенного значения, должны поддерживаться для всех типов данных при выполнении любых операций.* Например, если Null рассматривается как число 0 для неопределенного элемента числового типа и как пробел для неопределенного элемента символьного типа, это будет противоречить сформулированному выше правилу. Null должен рассматриваться только в качестве пропущенного, неопределенного элемента данных и никак иначе. Если же необходимо, чтобы незадаанные элементы имели некоторое заранее предопределенное значение, потребитель должен иметь возможность задавать *значения по умолчанию (default)*.

4. **Доступ к описанию БД в терминах реляционной модели.** *Словарь данных активной БД должен сохраняться в форме таблицы, и СУБД должна поддерживать доступ к нему при помощи стандартных языковых средств доступа к таблицам.* Нарушением этого правила является сохранение словаря данных в виде файлов операционной системы.

5. **Полнота подмножества языка.** *Язык манипулирования данными и язык определения данных должны поддерживать все операции доступа к данным и быть единственным средством такого доступа, кроме, возможно, операций низшего уровня*

(см. правило 12). Нарушением этого правила является возможность вмешательства в содержимое файла, хранящего таблицу, средствами, не являющимися операторами языка SQL. Кроме того, см. *правило 12*.

6. Возможность обновления представлений. *Все представления, подлежащие обновлению, должны быть доступны для этого.* Противоречит этому правилу ситуация, когда система позволяет объединить три таблицы и сформировать соответствующее представление, но не позволяет это представление обновить.

7. Наличие высокоуровневого языка манипулирования данными. *Операции вставки, обновления и удаления должны применяться к таблице в целом.* В настоящее время это правило удовлетворяется практически всеми СУБД.

8. Физическая независимость данных. *Прикладные программы не должны зависеть от используемых способов хранения данных на носителях и методов обращения к ним.* Если файл, в котором хранятся данные таблицы, перемещается на другой диск или переименовывается, это никак не должно сказываться на приложениях.

9. Логическая независимость данных. *Прикладные программы не должны зависеть от логических ограничений.* Если таблица разделяется на две, то представление должно обеспечивать слияние обеих частей с тем, чтобы это никак не сказывалось на приложениях.

10. Независимость контроля целостности. *Все необходимое для поддержания целостности данных должно храниться в словаре данных.* Ограничения на первичные ключи, внешние ключи, пределы допустимого диапазона значений данных, переключатели и тому подобные данные должны храниться в словаре данных.

11. Дистрибутивная независимость. *Реляционная БД должна быть переносима и способна к распространению.* Это правило представляет собой расширение *правила 8* в том смысле, что БД должна быть переносимой не только в пределах системы (локально переносимой), но и по сети, объединяющей множество систем (удаленно переносимой).

12. Согласование языковых уровней. Если реляционная СУБД допускает низкоуровневый язык доступа (элемент доступа — запись), последний не должен совершать операций, противоречащих требованиям правил безопасности и поддержания целостности данных, которые соблюдаются языком более высокого уровня. Средства, обеспечивающие такие низкоуровневые операции, как формирование резервных копий или загрузки, не должны игнорировать существующие в системе правила опознавания пользователей и их привилегий доступа, блокировок и ограничений. Однако зачастую разработчики СУБД пренебрегают этим правилом во имя повышения скорости. В таком случае забота о безопасности и целостности БД ложится в основном на плечи

администратора БД, что может рассматриваться в качестве приемлемого компромисса между теорией и практикой. Примером может служить отмена и восстановление ограничений при загрузке сверхбольших БД.

СУБД, удовлетворяющая всем фундаментальным требованиям (т.е. соответствующая обеим частям определения, обладающая шестью перечисленными свойствами, поддерживающая реляционные операции и не противоречащая двум правилам целостности), а также всем двенадцати правилам Кодда, может квалифицироваться как система управления реляционными БД. Все это Кодд суммировал в *правиле 0: для того чтобы систему можно было квалифицировать как реляционную СУБД, она должна использовать исключительно реляционные функции для управления базой данных.*

3.2 НОРМАЛИЗАЦИЯ

3.2.1 Основные определения

Нормализация представляет собой процесс дальнейшего совершенствования реляционной модели. Она выполняется после создания приближенной модели и предназначена для повышения уровня ее структурной организации. В основе нормализации лежит определенный математический аппарат, базирующийся на концепции *функциональной зависимости*.

Говорят, что один или множество столбцов функционально зависят от одного или множества столбцов X , если данное множество значений для X определяет единственное множество значений для Y . Утверждение " Y функционально зависит от X " равносильно утверждению " X определяет Y ", которое записывается в форме $X \Rightarrow Y$.

Функциональная зависимость. Поле B таблицы функционально зависит от поля A той же таблицы в том и только в том случае, когда в любой заданный момент времени для каждого из различных значений поля A обязательно существует только одно из различных значений поля B . Отметим, что здесь допускается, что поля A и B могут быть составными.

Полная функциональная зависимость. Поле B находится в полной функциональной зависимости от составного поля A , если оно функционально зависит от A и не зависит функционально от любого подмножества поля A .

Наиболее очевидным примером может быть первичный ключ таблицы реляционной модели, который однозначно определяет строку этой таблицы. Однако могут существовать и другие зависимости, в которые не входят первичные ключи. Главная цель нормализации — избавить реляционную таблицу от зависимостей, не связанных с первичными ключами.

Ниже приведены три основных довода в пользу нормализации, которые обычно приводятся в литературе, посвященной анализу и проектированию БД.

- **Обеспечение целостности.** Одного этого довода уже вполне достаточно, чтобы

взять на себя труд позаботиться о нормализации. Данные сохраняют корректность и достоверность, поскольку в результате нормализации они будут сохраняться только в одном месте. Другими словами, нормализация должна привести к исключению избыточности данных. В противном случае придется следить за синхронными изменениями всех копий данных, что потребует включения в систему дополнительно к стандартным средствам СУБД еще и очень сложных прикладных программ. С этой ситуацией уже много раз сталкивались пользователи существующих систем.

- **Создание формальной модели, как можно более независимой от специфики приложения.** Другими словами, нормализация способствует тому, что реляционная модель опирается на *данные*, а не на *процессы* их обработки. На практике это означает, что структура БД остается неизменной даже при изменении процессов обработки. Требования приложений не должны сказываться на логической структуре БД. (Однако они должны учитываться при проектировании *физической* структуры системы, как будет показано в дальнейшем.)

- **Снижение требований к объему памяти.** Непосредственным следствием этого является повышение скорости поиска данных. Если не обращать внимания на внешние ключи, то полная нормализация приводит к исключению избыточности данных в проектируемой системе. Естественно, что избыточность всегда связана с дополнительными объемами памяти для хранения избыточных данных. Кроме того, чем больший массив данных просматривается при поиске нужной информации, тем продолжительней поиск и, соответственно, тем ниже производительность системы.

3.2.2 Пять нормальных форм

Правила Кодда, как их теперь принято называть, очень просты и немногочисленны, но весьма строги. случае применения к таблицам с данными каждое правило описывает следующий уровень соответствия требованиям теории реляционных баз данных и различные степени нормализации. Как будет показано в дальнейшем, существует пять различных уровней нормализации, но ни одна из реляционных СУБД до сих пор не предоставляет поддержки для всех пяти нормальных форм. Это происходит из-за жестких требований в отношении производительности. Суть дела в том, что в полностью нормализованной базе данных для выполнения запроса вам потребуется соединить столь много таблиц, что производительность такой системы не сможет удовлетворить практических запросов.

Проведение нормализации базы данных состоит в устранении избыточности данных и выявлении функциональной зависимости.

Всякая нормализованная таблица автоматически считается таблицей в *первой нормальной форме*, сокращенно *1НФ*. Таким образом, строго говоря, "нормализованная" и "находящаяся в 1НФ" означают одно и то же. Однако на практике термин "нормализованная" часто используется в более узком смысле – "полностью нормализованная", который означает, что в проекте не нарушаются никакие принципы нормализации.

Теперь в дополнение к 1НФ можно определить дальнейшие уровни нормализации – *вторую нормальную форму (2НФ)*, *третью нормальную форму (3НФ)* и т.д. По существу, таблица находится в 2НФ, если она находится в 1НФ и удовлетворяет, кроме того, некоторому дополнительному условию, суть которого будет рассмотрена ниже. Таблица находится в 3НФ, если она находится в 2НФ и, помимо этого, удовлетворяет еще другому дополнительному условию и т.д.

Таким образом, каждая нормальная форма является в некотором смысле более ограниченной, но и более *желательной*, чем предшествующая. Это связано с тем, что "(N+1)-я нормальная форма" не обладает некоторыми непривлекательными особенностями, свойственным "N-й нормальной форме". Общий смысл дополнительного условия, налагаемого на (N+1)-ю нормальную форму по отношению к N-й нормальной форме, состоит в исключении этих непривлекательных особенностей.

3.2.3 Первая нормальная форма.

Таблица представлена в первой нормальной форме (1НФ) тогда и только тогда, когда все ее столбцы содержат только неделимые (в том смысле, что их дальнейшее разложение невозможно) значения и в ней отсутствуют повторяющиеся группы (столбцов) в пределах одной строки.

Первое правило реляционной модели состоит в том, что ни один столбец не должен содержать два или более значений, т.е. реляционная модель базы данных запрещает повторяющиеся поля.

Ни одну из реляционных СУБД не удовлетворяет только 1НФ, поскольку в этом случае требуется определять большое число полей, многие из которых остаются в основном пустыми. Альтернативный подход, при котором записи могут иметь переменную длину, практически не применяется из-за больших затрат ресурсов на реализацию алгоритма, определяющего, где кончается одна запись и начинается следующая.

Метод, который можно использовать для приведения таблицы к первой нормальной форме, состоит в том, чтобы разбить ее данные на две связанные таблицы:

Если таблица находится в 1НФ, то отображение ее данных и построение к ней

запросов не составляет никакой сложности. Однако это правило еще не препятствует дублированию данных в строках. Следствием будет ввод в таблицу большого количества данных, хранить которые нет никакой необходимости. Избыточные данные могут послужить причиной проблем целостности и снижения эффективности при внесении изменений, поэтому подобных решений при проектировании баз данных необходимо избегать.

3.2.4 Вторая нормальная форма

Для определения второй нормальной формы необходимо ввести концепцию *функциональной зависимости*. Это зависимость, связывающая атрибуты в одной таблице с единственным значением в другой таблице. Функциональную зависимость для таблиц А и В принято обозначать как $A \Rightarrow B$. Это понятие подводит "на один шаг" к родственной концепции объединения таблиц в отношения типа **1:1** или **1:M**. Приведем определение второй нормальной формы.

Таблица представлена во второй нормальной форме (2НФ) тогда и только тогда, когда она представлена в 1НФ и каждый неключевой атрибут полностью определяется первичным ключом. Атрибут полностью определяется первичным ключом, если он находится в правой части выражения, описывающего функциональную зависимость, а левую часть этого выражения представляет первичный ключ или какое-либо выражение, которое может быть вычислено на его основе с использованием транзитивных свойств функциональной зависимости.

Под транзитивными свойствами функциональной зависимости понимается следующее: если существует отношение между двумя атрибутами в одном направлении, то между этими же атрибутами существует отношение в противоположном направлении.

Избыточность 1НФ. может послужить причиной возникновения проблем и при удалении или добавлении *Аномалия вставки* может проявиться в том случае, когда при добавлении в таблицу с первичным ключом какой-либо записи вы вынуждены поместить в поле первичного ключа либо пустое, либо уже существующее значение. Подобное действие нарушает основные требования реляционной теории и потребует от вас (как минимум) заново сформировать поле первичного ключа этой таблицы и все связанные с ним ключевые поля в других таблицах, что вызовет значительные потери в производительности. Аналогично, *аномалия удаления* проявляется в тех случаях, когда у вас возникает необходимость удалить запись и провести реорганизацию ваших таблиц. Если впоследствии потребуется восстановить удаленную запись, то вынуждены будете вновь подвергнуть обработке весь набор таблиц.

Цель разработки 2НФ состоит в устранении этих проблем.

В этих таблицах, представленных в 2НФ, аномалия удаления возникнуть не может.

3.2.5 Третья нормальная форма

В общем случае 1НФ и 2НФ рассматриваются как промежуточные ступени в процессе нормализации базы данных. Большая часть СУБД ориентирована на достижение следующей степени нормализации, именуемой третьей нормальной формой (3НФ). Дальнейшая нормализация выделяет элементы информации в отдельные таблицы, так что они не будут потеряны при удалении в исходных таблицах. Приведем определение третьей нормальной формы.

Таблица представлена в третьей нормальной форме (3НФ), если она удовлетворяет определению 2НФ и не одно из неключевых полей не зависит функционально от любого другого неключевого поля.

Таблица находится в нормальной форме Бойса-Кодда (БКНФ), если и только если любая функциональная зависимость между его полями сводится к полной функциональной зависимости от возможного ключа.

Это определение — всего лишь оригинальный способ выразить необходимость представления системы связанных таблиц в таком виде, чтобы атрибуты каждой таблицы непосредственно определялись первичным ключом этой таблицы.

Уместно сделать несколько замечаний о недостатках, присущих даже таблицам, представленным в 3НФ. Существуют варианты, когда имеет смысл разделить таблицу на более мелкие таблицы, если часть представленных в ней данных непостоянна и часто обновляется, а остальные данные пассивны и изменяются в редких случаях. Также есть смысл объединить таблицы, когда необходимо обеспечить высокую скорость реакции на запрос. Можно даже пойти на дублирование данных в таблицах, если это позволит снизить затраты на обработку запросов, хотя формально не следовало бы этого делать.

Иногда приходится отступить от принципа полной нормализации данных проекта. Это может быть вызвано следующими причинами:

- * временем выполнения запросов;
- * временем проведения обновлений;
- * общим необходимым объемом хранилища данных;
- * аномалиями удаления, которые могут вызвать потерю целостности данных.

3.2.6 Четвертая нормальная формы

Прежде чем закончить рассмотрение правил Кодда, вам будет предложен краткий обзор двух последних правил реляционных баз данных. Эти два правила предназначены для устранения еще двух аномалий, называемых *многозначная зависимость* и *объединяющая зависимость*. Многозначная зависимость определяется следующим образом.

В таблице X существует многозначная зависимость $A \Rightarrow B$, если в этой таблице можно обнаружить ситуации, в которых пара строк содержит дублирующееся значение A и одновременно существуют другие пары строк, полученные путем перестановки значений B , присутствующих в первой паре.

Прежде всего, для существования многозначной зависимости требуется существование пар строк. A и B могут быть как отдельными атрибутами, так и объединением некоторого набора атрибутов. Тривиальная многозначная зависимость для $A \Rightarrow B$ существует в случае, когда B является подмножеством A или A объединяет $B=XS$ (более крупная таблица содержит исходную таблицу).

Многозначная зависимость. Поле A многозначно определяет поле B той же таблицы, если для каждого значения поля A существует хорошо определенное множество соответствующих значений B .

Таблица 3.2. Обучение

Дисциплина	Преподаватель	Учебник
Информатика	Шипилов П.А.	Форсайт Р. Паскаль для всех
Информатика	Шипилов П.А.	Уэйт М. и др. Язык Си
Информатика	Голованевский Г.Л.	Форсайт Р. Паскаль для всех
Информатика	Голованевский Г.Л.	Уэйт М. и др. Язык Си
...

Для примера рассмотрим таблицу "Обучение". В ней есть многозначная зависимость "Дисциплина-Преподаватель": дисциплина (в примере Информатика) может читаться несколькими преподавателями (в примере Шипиловым и Голованевским). Есть и другая многозначная зависимость "Дисциплина-Учебник": при изучении Информатики используются учебники "Паскаль для всех" и "Язык Си". При этом Преподаватель и Учебник не связаны функциональной зависимостью, что приводит к появлению избыточности (для добавление еще одного учебника придется ввести в таблицу две новых строки). Дело улучшается при замене этой таблицы на две: (Дисциплина-Преподаватель и Дисциплина-Учебник).

Существование многозначной зависимости порождает аномалию обновления. Четвертая нормальная форма устраняет нетривиальную многозначную зависимость в таблице посредством создания меньших таблиц. Процесс нормализации представляет

собой создание как можно большего числа все более мелких таблиц в целях сокращения избыточности данных. Определим четвертую нормальную форму.

Таблица X представлена в 4НФ тогда и только тогда, когда она представлена в БКНФ и для любой многозначной зависимости $A \twoheadrightarrow B$ в этой таблице можно сказать, что A является первичным ключом таблицы X .

3.2.7 Пятая нормальная формы

При любой декомпозиции таблицы на две других таблицы полученные таблицы обладают свойствами соединения без потерь. Это означает, что полученные таблицы можно снова соединить и получить прежнюю таблицу в исходном виде. Однако бывают случаи, когда требуется выполнить декомпозицию. Таблицы более чем на две таблицы. В таких (достаточно редких) случаях возникает необходимость учитывать зависимость соединения без потерь, которая устраняется с помощью пятой нормальной формы (5НФ).

Зависимость соединения без потерь – свойство декомпозиции, которое гарантирует отсутствие фиктивных строк при восстановлении первоначальной таблицы с помощью операций естественного соединения.

При разбиении таблиц с помощью операций проекции используемый метод определяется совершенно точно. В частности следует позаботиться о том, чтобы при обратном соединении полученных таблиц можно было восстановить исходную таблицу. Такая декомпозиция называется декомпозицией с соединением без потерь, поскольку при ее выполнении сохраняются все данные исходной таблицы, а также исключается создание дополнительных фиктивных строк.

Таблицы находятся в пятой нормальной форма (5НФ) тогда и только тогда, когда она представлена в 4НФ и не содержит зависимостей соединения. Таблица R с подмножеством столбцов A, B, \dots, Z удовлетворяет зависимости соединения, если и только если каждое допустимое значение R равно соединению его проекций на подмножество A, B, \dots, Z .

3.2.8 Алгоритм нормализации (приведение к 3НФ)

Итак, алгоритм нормализации (т.е. алгоритм приведения отношений к 3НФ) описывается следующим образом.

Шаг 1 (Приведение к 1НФ). На первом шаге задается одно или несколько отношений, отображающих понятия предметной области. По модели предметной области (не по внешнему виду полученных отношений!) выписываются обнаруженные функциональные зависимости. Все отношения автоматически находятся в 1НФ.

Шаг 2 (Приведение к 2НФ). Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводим декомпозицию этих отношений на несколько отношений следующим образом: те атрибуты, которые зависят от части сложного ключа выносятся в отдельное отношение вместе с этой частью ключа. В исходном отношении остаются все ключевые атрибуты:

Исходное отношение: .

$$R(K_1, K_2, A_1, \dots, A_n, B_1, \dots, B_m).$$

Ключ: $\{K_1, K_2\}$ - сложный.

Функциональные зависимости:

$$\{K_1, K_2\} \rightarrow \{A_1, \dots, A_n, B_1, \dots, B_m\} \text{ - зависимость всех атрибутов от ключа отношения.}$$

$$\{K_1\} \rightarrow \{A_1, \dots, A_n\} \text{ - зависимость некоторых атрибутов от части сложного ключа.}$$

Декомпозированные отношения:

$$R_1(K_1, K_2, B_1, \dots, B_m) \text{ - остаток от исходного отношения. Ключ } \{K_1, K_2\}$$

$R_2(K_1, A_1, \dots, A_n)$ - атрибуты, вынесенные из исходного отношения вместе с частью сложного ключа. Ключ K_1 .

Шаг 3 (Приведение к 3НФ). Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов других неключевых атрибутов, то проводим декомпозицию этих отношений следующим образом: те неключевые атрибуты, которые зависят других неключевых атрибутов выносятся в отдельное отношение. В новом отношении ключом становится детерминант функциональной зависимости:

$$\text{Исходное отношение: } R(K, A_1, \dots, A_n, B_1, \dots, B_m).$$

Ключ: K .

Функциональные зависимости:

$$K \rightarrow \{A_1, \dots, A_n, B_1, \dots, B_m\} \text{ - зависимость всех атрибутов от ключа отношения.}$$

$\{A_1, \dots, A_n\} \rightarrow \{B_1, \dots, B_m\}$ - зависимость некоторых неключевых атрибутов от других неключевых атрибутов.

Декомпозированные отношения:

$$R_1(K, A_1, \dots, A_n) \text{ - остаток от исходного отношения. Ключ } K.$$

$R_2(A_1, \dots, A_n, B_1, \dots, B_m)$ - атрибуты, вынесенные из исходного отношения вместе с детерминантом функциональной зависимости. Ключ $\{A_1, \dots, A_n\}$.

Замечание. На практике, при создании логической модели данных, как правило, не следуют прямо приведенному алгоритму нормализации. Опытные разработчики обычно

сразу строят отношения в 3НФ. Кроме того, основным средством разработки логических моделей данных являются различные варианты ER-диаграмм. Особенность этих диаграмм в том, что они сразу позволяют создавать отношения в 3НФ. Тем не менее, приведенный алгоритм важен по двум причинам. Во-первых, этот алгоритм показывает, какие проблемы возникают при разработке слабо нормализованных отношений. Во-вторых, как правило, модель предметной области никогда не бывает правильно разработана с первого шага. Эксперты предметной области могут забыть о чем-либо упомянуть, разработчик может неправильно понять эксперта, во время разработки могут измениться правила, принятые в предметной области, и т.д. Все это может привести к появлению новых зависимостей, которые отсутствовали в первоначальной модели предметной области. Тут как раз и необходимо использовать алгоритм нормализации хотя бы для того, чтобы убедиться, что отношения остались в 3НФ и логическая модель не ухудшилась.

3.2.9 Анализ критериев для нормализованных и ненормализованных моделей данных.

Сравнение нормализованных и ненормализованных моделей

Соберем воедино результаты анализа критериев, по которым мы хотели оценить влияние логического моделирования данных на качество физических моделей данных и производительность базы данных.

Таблица 3.3. Сравнительный анализ нормальных форм

Критерий	Отношения слабо нормализованы (1НФ, 2НФ)	Отношения сильно нормализованы (3НФ)
Адекватность базы данных предметной области	ХУЖЕ (-)	ЛУЧШЕ (+)
Легкость разработки и сопровождения базы данных	СЛОЖНЕЕ (-)	ЛЕГЧЕ (+)
Скорость выполнения вставки, обновления, удаления	МЕДЛЕННЕЕ (-)	БЫСТРЕЕ (+)
Скорость выполнения выборки Данных	БЫСТРЕЕ (+)	МЕДЛЕННЕЕ (-)

Как видно из таблицы, более сильно нормализованные отношения оказываются лучше спроектированы (три плюса, один минус). Они больше соответствуют предметной области, легче в разработке, для них быстрее выполняются операции модификации базы данных. Правда, это достигается ценой некоторого замедления выполнения операций выборки данных.

У слабо нормализованных отношений единственное преимущество - если к базе данных обращаться только с запросами на выборку данных, то для слабо нормализованных отношений такие запросы выполняются быстрее.

Это связано с тем, что в таких отношениях уже как бы произведено соединение отношений и на это не тратится время при выборке данных.

Таким образом, выбор степени нормализации отношений зависит от характера запросов, с которыми чаще всего обращаются к базе данных.

3.2.10 OLTP и OLAP-системы

Можно выделить некоторые классы систем, для которых больше подходят сильно или слабо нормализованные модели данных.

Сильно нормализованные модели данных хорошо подходят для так называемых OLTP-приложений (On-Line Transaction Processing (OLTP)- оперативная обработка транзакций). Типичными примерами OLTP-приложений являются системы складского учета, системы заказов билетов, банковские системы, выполняющие операции по переводу денег, и т.п. Основная функция подобных систем заключается в выполнении большого количества коротких транзакций. Сами транзакции выглядят относительно просто, например, "снять сумму денег со счета А, добавить эту сумму на счет В". Проблема заключается в том, что, во-первых, транзакций очень много, во-вторых, выполняются они одновременно (к системе может быть подключено несколько тысяч одновременно работающих пользователей), в-третьих, при возникновении ошибки, транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции (не должно быть ситуации, когда деньги сняты со счета А, но не поступили на счет В). Практически все запросы к базе данных в OLTP-приложениях состоят из команд вставки, обновления, удаления. Запросы на выборку в основном предназначены для предоставления пользователям возможности выбора из различных справочников. Большая часть запросов, таким образом, известна заранее еще на этапе проектирования системы. Таким образом, критическим для OLTP-приложений является скорость и надежность выполнения коротких операций обновления данных. Чем выше уровень нормализации данных в OLTP-приложении, тем оно, как правило, быстрее и надежнее. Отступления от этого правила могут происходить тогда, когда уже на этапе разработки известны некоторые часто возникающие запросы, требующие соединения отношений и от скорости выполнения которых существенно зависит работа приложений. В этом случае можно пожертвовать нормализацией для ускорения выполнения подобных запросов.

Другим типом приложений являются так называемые OLAP-приложения (On-Line Analytical Processing (OLAP) - оперативная аналитическая обработка данных). Это

обобщенный термин, характеризующий принципы построения систем поддержки принятия решений (Decision Support System - DSS), хранилищ данных (Data Warehouse), систем интеллектуального анализа данных (Data Mining). Такие системы предназначены для нахождения зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками потенциальных покупателей), для проведения анализа "что если...". OLAP-приложения оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми из электронных таблиц или из других источников данных. Такие системы характеризуются следующими признаками:

Добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложения).

- Данные, добавленные в систему, обычно никогда не удаляются.
- Перед загрузкой данные проходят различные процедуры "очистки", связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления для одних и тех же понятий, данные могут быть некорректны, ошибочны.
- Запросы к системе являются нерегламентированными и, как правило, достаточно сложными. Очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса.
- Скорость выполнения запросов важна, но не критична.

Данные OLAP-приложений обычно представлены в виде одного или нескольких гиперкубов, измерения которого представляют собой справочные данные, а в ячейках самого гиперкуба хранятся собственно данные. Например, можно построить гиперкуб, измерениями которого являются: время (в кварталах, годах), тип товара и отделения компании, а в ячейках хранятся объемы продаж. Такой гиперкуб будет содержать данных о продажах различных типов товаров по кварталам и подразделениям. Основываясь на этих данных, можно отвечать на вопросы вроде "у какого подразделения самые лучшие объемы продаж в текущем году?", или "каковы тенденции продаж отделений Юго-Западного региона в текущем году по сравнению с предыдущим годом?"

Физически гиперкуб может быть построен на основе специальной многомерной модели данных (MOLAP - Multidimensional OLAP) или построен средствами реляционной модели данных (ROLAP - Relational OLAP).

Возвращаясь к проблеме нормализации данных, можно сказать, что в системах OLAP, использующих реляционную модель данных (ROLAP), данные целесообразно хранить в виде слабо нормализованных отношений, содержащих заранее вычисленные

основные итоговые данные. Большая избыточность и связанные с ней проблемы тут не страшны, т.к. обновление происходит только в момент загрузки новой порции данных. При этом происходит как добавление новых данных, так и пересчет итогов.

3.2.11 Нормализация на практике

В этом последнем разделе мы рассмотрим на примере, как воплощается требование атомарности данных в *первую нормальную форму* (1НФ). Но сначала напомним ее определение.

Отсутствие повторяющихся групп. Это аналогично требованию того, чтобы на пересечении строки и столбца (в ячейке таблицы) находилось единственное значение, которое должно быть атомарным.

STATE ABBREV	SPOP	CITY	LPOP	CPOP	PCTINC
North Carolina NC	5 млн.	Burlington	40 тыс.	44 тыс.	10%
		Raleigh	200 тыс.	222 тыс.	11%
Vermont VT	4 млн.	Burlington	60 тыс.	67,2 тыс.	12%
New York NY	17 млн.	Albany	500 тыс.	540 тыс.	8%
		New York City	14 млн.	14,7 млн.	5%
		White Plains	100 тыс.	106 тыс.	6%

Для того чтобы получить 1НФ, нужно заменить группы, подразумевающие повторение, данными из строк, расположенных непосредственно перед ними.

STATE	ABBREV	SPOP	CITY	LPOP	CPOP	PCTINC
North Carolina	NC	5 млн.	Burlington	40 тыс.	44 тыс.	10%
North Carolina	NC	5 млн.	Raleigh	200 тыс.	222 тыс.	11%
Vermont	VT	4 млн.	Burlington	60 тыс.	67,2 тыс.	12%
New York	NY	17 млн.	Albany	500 тыс.	540 тыс.	8%
New York	NY	17 млн.	New York City	14 млн.	14,7 млн.	5%
New York	NY	17 млн.	White Plains	100 тыс.	106 тыс.	6%

Для нормализации более высоких порядков нам потребуется *неключевой* столбец. Определение такого столбца довольно очевидно — это *столбец, который не входит в состав ни одного потенциального ключа*. Столбцы нашей таблицы можно разделить на две группы: те, что *могут претендовать на роль первичного ключа*, и *остальные*. Столбцы, которые входят в группу *остальные*, являются *неключевыми*.

Определение второй нормальной формы (2НФ) выглядит следующим образом: **отсутствие частичной зависимости.** *Каждый неключевой столбец зависит от полного первичного ключа, то есть от всех столбцов, из которых он состоит (если первичный ключ составной).* Таблица не удовлетворяет этому определению. Информация о городе не зависит от информации о штате. В частности, столбцы CITY, LPOP, CPOP и PCTINC не зависят от столбца названия штата STATE. Следовательно, нужно разбить ее на две таблицы. Смысл этого разбиения в том, что теперь штаты и города становятся отдельными

объектами, хотя и связанными, и будут представлены отдельными таблицами.

STATE	ABBREV	SPOP
North Carolina	NC	5 млн.
Vermont	VT	4 млн.
New York	NY	17 млн.

CITY	ABBREV	LPOP	CPOP	PCTINC
Burlington	NC	40 тыс.	44 тыс.	10%
Raleigh	NC	200 тыс.	222 тыс.	11%
Burlington	VT	60 тыс.	67,2 тыс.	12%
New York City	NY	14 млн.	14,7 млн.	5%
Albany	NY	500 тыс.	540 тыс.	8%
White Plains	NY	100 тыс.	106 тыс.	6%

Определение третьей нормальной формы (ЗНФ) выглядит следующим образом:

отсутствие транзитивной зависимости. Ни один неключевой столбец не должен зависеть от другого неключевого столбца. Можно сказать, что таблица находится в ЗНФ, если все ее неключевые столбцы зависят только от ключа. Если после удаления повторяющихся групп неключевой столбец зависит от ключа, мы имеем таблицу 2НФ. А если сохраняется зависимость от ключа, то получится ЗНФ. Наша таблица городов не удовлетворяет этому требованию, поскольку столбец PCTINC (относительный прирост) зависит от столбцов CPOP (текущее количество жителей) и LPOP (количество жителей в предыдущем году). То есть этот столбец является функцией от двух других. Такой столбец называется *производным (derived)*. Но заметим, что все три столбца являются неключевыми. Само собой напрашивается решение: исключить из таблицы столбец PCTINC и вычислять соответствующее значение "по ходу", то есть во время работы с таблицей. При частом обращении можно использовать для этого *представление (view)*. В таблице штатов столбец SPOP (население штата) зависит от столбца ABBREV (аббревиатура штата), поскольку последний является потенциальным ключом, хотя и не первичным. В результате мы получим три таблицы ЗНФ.

ABBREV	SPOP
NC	5 млн.
VT	4 млн.
NY	17 млн.

STATE	ABBREV
North Carolina	NC
Vermont	VT
New York	NY

CITY	ABBREV	LPOP	SPOP
Burlington	NC	40 тыс.	44 тыс.
Raleigh	NC	200 тыс.	222 тыс.
Burlington	VT	60 тыс.	67,2 тыс.
New York City	NY	14 млн.	14,7 млн.
Albany	NY	500 тыс.	540 тыс.
White Plains	NY	100 тыс.	106 тыс.

Определение нормальной форма Бойса-Кодда (НФБК) выглядит следующим образом: **отсутствие инверсной частичной зависимости**. Ни первичный ключ, ни какая-либо его часть не должны зависеть от неключевого атрибута. Поскольку мы использовали прямое определение неключевого столбца, то полученные ЗНФ-таблицы удовлетворяют требованиям НФБК.

В теории нормализации рассматриваются и нормальные формы более высоких порядков — четвертая, пятая и т.д. Применительно к базам данных уровни нормализации выше пятого встречаются крайне редко и представляют собой, скорее, чисто теоретический интерес. 4НФ применяются к *многозначным зависимостям (multivalued dependence)*, а пятая — к *объединенным зависимостям (join dependence)*.

3.3 ПЕРЕВОД ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ МОДЕЛИ В РЕЛЯЦИОННУЮ

3.3.1 Сравнительный анализ классов и отношений

При подготовке к преобразованию всей модели данных уместно уточнить связь между объектной моделью и реляционной конструкцией. На уровне отдельных классов каждый класс ставится в соответствие с таблицей.

Переход от объектно-ориентированной к реляционной модели не делается прямо "в лоб". В объектной модели отсутствует множество деталей, которые должны быть точно определены в реляционной. Класс может быть преобразован в группу таблиц, а группа классов — в одну таблицу, и т.д. Более точно было бы говорить о классах как о группах таблиц.

3.3.2 Определение первичных ключей

Каждая таблица должна иметь первичный ключ, гарантирующий, что каждая строка таблицы может быть однозначно определена. Первичный ключ может состоять из одного или более столбцов таблицы, для чего существует целый ряд теоретических и практических причин. База данных моделирует реальный мир, таблица является аналогом класса, а запись таблицы представляет отдельный объект класса. База данных должна иметь средства для идентификации объектов. Если она не позволяет отличить два разных объекта, значит, она неточно соответствует реальности. Аналогичным образом, каждый отдельный объект может быть представлен только одним способом. Если разрешить повторяющиеся записи, то модель будет представлять несколько копий одного и того же, что противоречит теории.

Реляционная модель в качестве ключевых значений позволяет использовать реальные атрибуты объектов, например, имя человека. На практике программисты часто испытывают недостаток таких признаков. Так, для таблицы людей нельзя использовать имя в качестве ключа, поскольку оно не является уникальным. Не подходит и номер паспорта: существует масса дубликатов и подделок, у детей паспортов нет, люди могут отказаться сообщать номера, и паспорта имеют смысл только внутри одной страны. Поэтому в качестве идентификатора человека обычно используют выработанное компьютером значение.

Аналитик принимает решение об использовании искусственных ключей после получения информации о максимальных значениях полей. В подавляющем большинстве случаев должны быть использованы выработанные системой значения.

В требования к системе должны входить значения верхних пределов. Лучше всего выбирать такой размер полей, чтобы можно было записывать значения, от 10 до 100 раз превышающие используемые в настоящий момент — для возможности роста и развития.

3.3.3 Переход от классов к взаимосвязанным таблицам

После определения первичных ключей для каждой таблицы они могут быть использованы для связывания таблиц в базу данных.

В объектно-ориентированном анализе отношения между классами выражаются через иерархии, контейнеры и ассоциации. В реляционной модели они выражаются только через совместно используемые ключи, поскольку это исключительно модель данных.

Создание связей между таблицами — довольно абстрактный процесс. Но, поскольку от него зависит вся сущность разрабатываемой конструкции, этот шаг очень важен.

Отношения между таблицами имеют две основных характеристики — вид и тип. Вид отношения может быть: "составная часть", "ассоциация" или "подкласс". Эти виды

были использованы на стадии анализа системы для описания взаимоотношений между классами.

При переходе от объектной к реляционной модели важны обе характеристики отношений. Они позволяют определить, что именно означает конкретное отношение и как классы соответствуют таблицам.

3.3.4 Отношения много-ко-многим.

Это — классическая проблема для реляционного моделирования. Трудность заключается в том, что реляционная модель не имеет непосредственной поддержки отношений много-ко-многим.

Решение состоит в создании промежуточной таблицы. Первичный ключ этой таблицы должен состоять из двух внешних ключей — ключа таблицы 1 и ключа таблицы 2.

При этом сохраняется третья нормальная форма данных, и модель соответствует системе. Правда, достигается это ценой обслуживания дополнительной таблицы, но запросы SQL будут работать с ней быстро, да и создание форм не представляет особой сложности. Пример такого преобразования показан на рисунке 3.1.

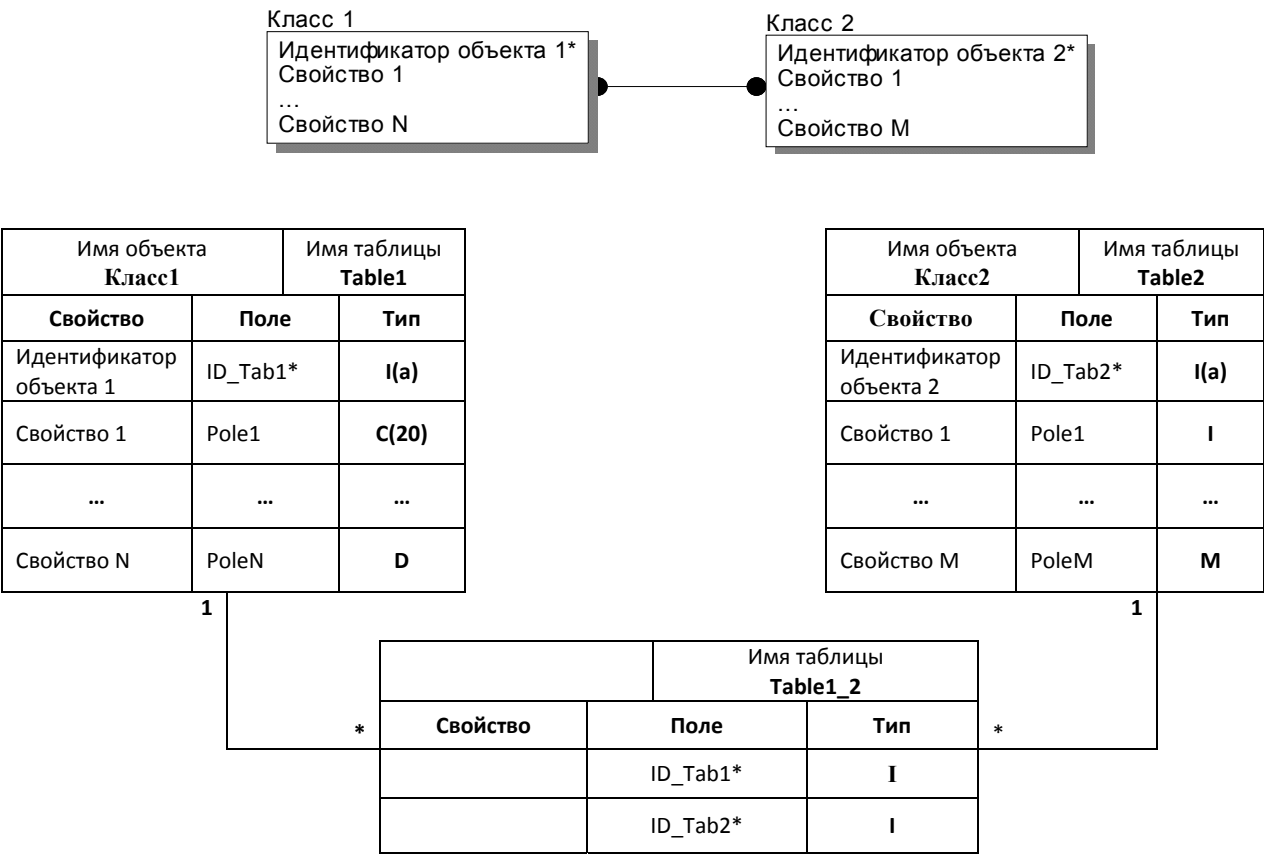


Рис 3.1. Преобразование связи много-ко-многим

3.3.5 Создание столбцов/полей таблицы из свойств классов

Следующий этап работы над проектом предполагает переход от свойств классов к столбцам таблицы. При этом требуется определить типы и размеры всех столбцов, а также добавить специальные столбцы, например, для адреса. Использование этих дополнительных и требуемых только для конкретной реализации проекта данных было описано выше.

Объектно-ориентированный анализ позволяет увидеть нужные данные и этим внести порядок в программу. Он обеспечивает простой и логичный переход к реляционной конструкции.

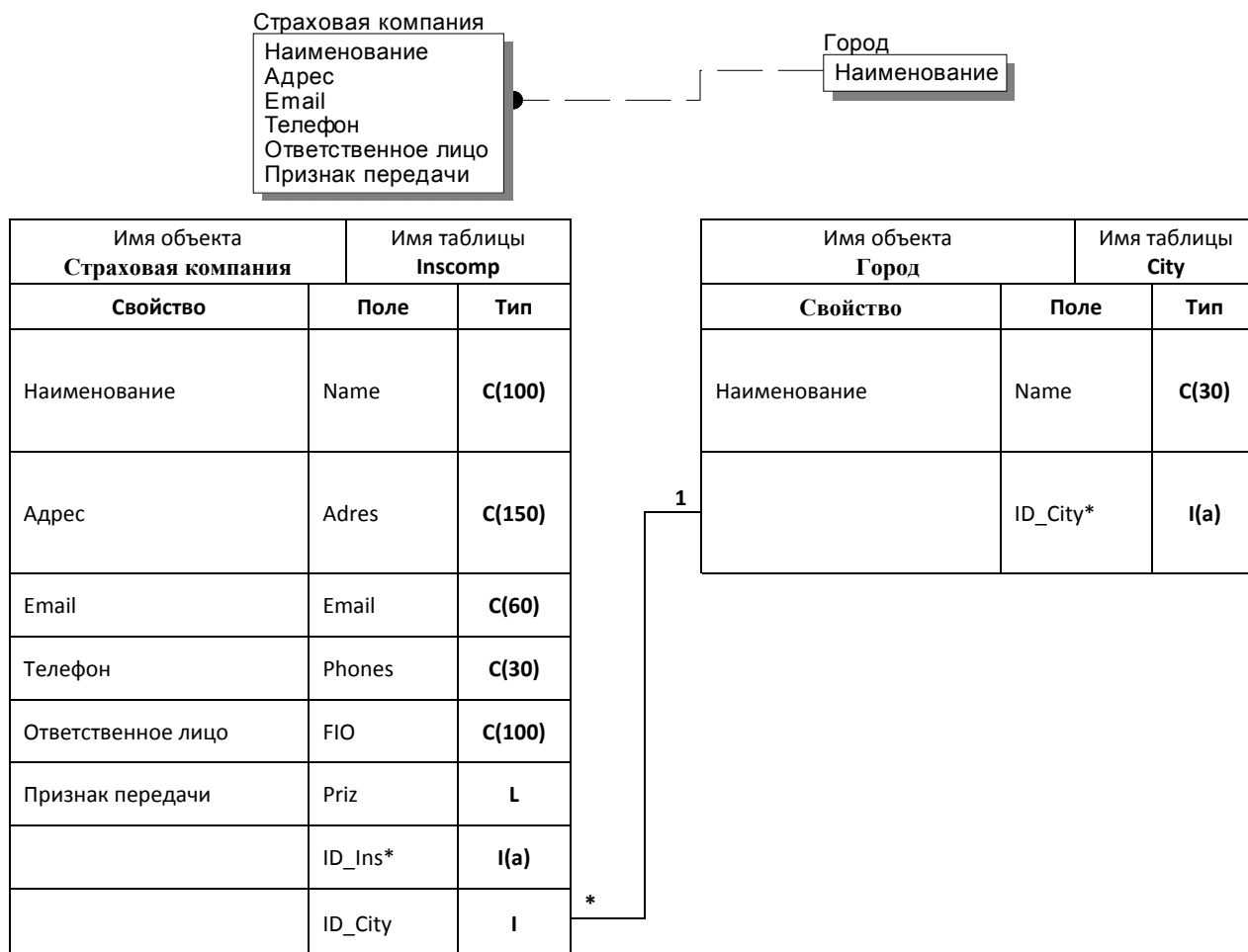


Рис. 3.2. Переход от объектной модели к реляционной

3.3.6 Реализация методов

Методы сравнительно просто реализуются в виде процедур.

3.4 РЕЛЯЦИОННАЯ АЛГЕБРА

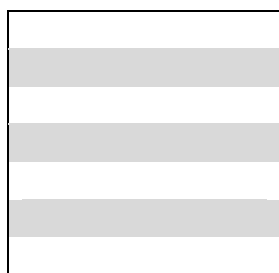
Реляционная алгебра — это теоретический язык операций, позволяющих создавать на основе одного или нескольких отношений другое отношение без изменения самих исходных отношений. Таким образом, оба операнда и результат являются отношениями, поэтому результаты одной операции могут применяться в другой операции. Это позволяет создавать вложенные выражения реляционной алгебры (по аналогии с тем, как создаются вложенные арифметические выражения), но при любой глубине вложенности результатом

является отношением. Такое свойство называется *замкнутостью*. Оно подчеркивает то, что применение любого количества операций реляционной алгебры к отношениям не приводит к созданию иных объектов, кроме отношений, точно так же, как результатами арифметических операций с числами являются только числа.

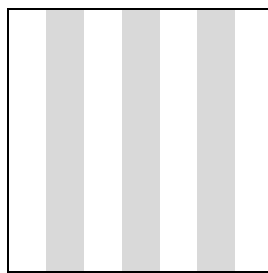
Реляционная алгебра является языком последовательного использования отношений, в котором все кортежи, возможно, даже взятые из разных отношений, обрабатываются одной командой, без организации циклов.

Существует несколько вариантов выбора операций, которые включаются в реляционную алгебру. Первоначально Кодд предложил восемь операций, но впоследствии к ним были добавлены и некоторые другие. Пять основных операций реляционной алгебры, а именно *выборка* (selection), *проекция* (projection), *декартово произведение* (cartesian product), *объединение* (union) и *разность множеств* (set difference), выполняют большинство действий по извлечению данных, которые могут представлять для нас интерес. На основании пяти основных операций можно также вывести дополнительные операции, такие как операции *соединения* (join), *пересечения* (intersection) и *деления* (division), которые могут быть выражены в терминах пяти основных операций. Результаты этих операций схематически показаны на рис. 3.3.

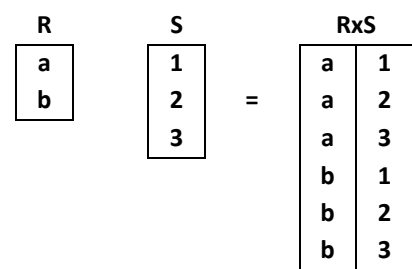
Операции выборки и проекции являются *унарными*, поскольку они работают с одним отношением. Другие операции работают с парами отношений, и поэтому их называют *бинарными* операциями. В приведенных ниже определениях R и S — это два отношения, определенные на атрибутах $A = (a_1, a_2, \dots, a_N)$ и $B = (b_1, b_2, \dots, b_M)$ соответственно.



а) выборка



б) Проекция



в) Декартово произведение

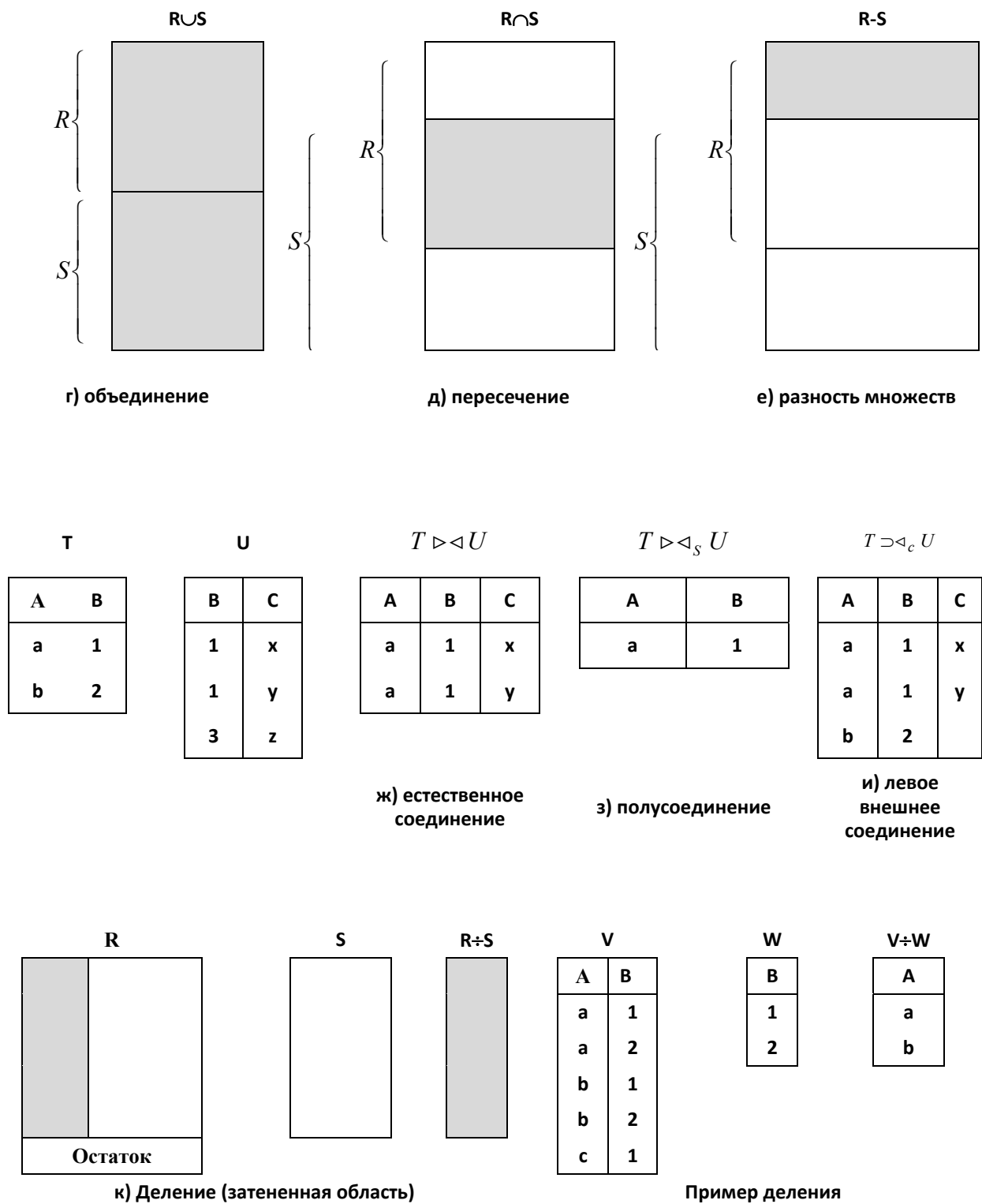


Рис.3.3. Схематическое представление результатов операций реляционной алгебры

3.4.1. Унарные операции

Начнем описание операций реляционной алгебры с изучения двух унарных операций: выборки и проекции.

Выборка (или ограничение)

$\sigma_{\text{предикат}}(R)$. Операция выборки применяется к одному отношению R и определяет

результатирующее отношение, которое содержит только те кортежи (строки) из отношения R, которые удовлетворяют заданному условию (предикату).

Пример операции выборки: составьте список всех сотрудников с зарплатой, превышающей 10000 фунтов стерлингов.

$\sigma_{\text{salary} > 10000}(\text{Staff})$

Здесь исходным отношением является отношение Staff, а предикатом — выражение salary > 10000. Операция выборки определяет новое отношение, содержащее только те кортежи отношения Staff, в которых значение атрибута salary превышает 10000 фунтов стерлингов. Результат выполнения этой операции показан в табл. 3.4. Более сложные предикаты могут быть созданы с помощью логических операций AND, OR или NOT.

Таблица 3.4. Результат выполнения операции выборки из отношения Staff кортежей с атрибутом salary > 10000

StaffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

Проекция

$\Pi_{a_1, \dots, a_n}(R)$. Операция проекции применяется к одному отношению R и определяется новое отношение, содержащие вертикальное подмножество отношения R, создаваемое посредством извлечения значений указанных атрибутов и исключения из результатов строк-дубликатов.

Пример операции проекции: Создайте ведомость зарплаты всех сотрудников компании с указанием только атрибутов staffNo, fName, lName и salary.

$\Pi_{\text{staffNo}, \text{fName}, \text{lName}, \text{salary}}(\text{Staff})$

В этом примере операция проекции определяет новое отношение, которое будет содержать только атрибуты staffNo, fName, lName и salary отношения Staff, размещенные в указанном порядке. Результат выполнения этой операции показан в табл. 3.5.

Таблица 3.5. Проекция отношения Staff по атрибутам staffNo, fName, lName, salary

staffNo	fName	lName	Salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

3.4.2. Операции с множествами

Операции, выборки и проекции предусматривают извлечение информации только из одного отношения. Тем не менее на практике часто возникает необходимость получить данные с помощью нескольких отношений. Ниже в этом разделе рассматриваются бинарные операции реляционной алгебры, начиная с операций объединения, вычисления разности, пересечения и декартова произведения.

Объединение

$R \cup S$. Объединение двух отношений R и S определяет новое отношение, которое включает все кортежи, содержащиеся только в R , только в S , одновременно в R и S , причем все дубликаты кортежей исключены. При этом отношения R и S должны быть совместимыми по объединению.

Если R и S включают, соответственно, I и J кортежей, то объединение этих отношений можно получить, собрав все кортежи в одно отношение, которое может содержать не более $(I + J)$ кортежей. Объединение возможно, только если схемы двух отношений совпадают, т.е. состоят из одинакового количества атрибутов, причем каждая пара соответствующих атрибутов имеет одинаковый домен. Иначе говоря, отношения должны быть *совместимыми по объединению*. Отметим, что в определении совместимости по объединению не указано, что атрибуты должны иметь одинаковые имена. В некоторых случаях для получения двух совместимых по объединению отношений может быть использована операция проекции.

Пример операции объединения. Создайте список всех городов, в которых имеется отделение компании или объект недвижимости.

$$\Pi_{\text{city}}(\text{Branch}) \cup \Pi_{\text{city}}(\text{PropertyForRent})$$

Для создания совместимых по объединению отношений сначала следует применить операцию проекции, чтобы выделить из отношений `Branch` и `PropertyForRent` столбцы с атрибутами `city`, исключая в случае необходимости дубликаты. Затем для комбинирования полученных промежуточных отношений следует использовать операцию объединения. Результат выполнения всех этих действий приведен в таблице 3.6.

Таблица 3.6. Объединение, основанное на атрибуте `city` отношений `Branch` и `PropertyForRent`

city
London
Aberdeen
Glasgow
Bristol

Разность

R-S. Разность двух отношений R и S состоит из кортежей, которые имеются в отношении R, но отсутствуют в отношении S. Причем отношения R и S должны быть совместимыми по объединению.

Пример операции разности

Создайте список всех городов, в которых есть отделение компании, но нет объектов недвижимости, сдаваемых в аренду.

$$\Pi_{\text{city}}(\text{Branch}) - \Pi_{\text{city}}(\text{PropertyForRent})$$

В данном случае аналогично предыдущему примеру следует создать совместимые по объединению отношения Branch и PropertyForRent, выполнив их проекцию по атрибуту city. Затем для комбинирования полученных новых отношений следует использовать операцию разности. Результат выполнения этих операций показан в таблице 3.7.

Таблица 3.7. Разность, основанная на атрибуте city отношений Branch и PropertyForRent

City
Bristol

Пересечение

$R \cap S$. Операция пересечения определяет отношение, которое содержит кортежи, присутствующие как в отношении R, так и в отношении S. Отношения R и S должны быть совместимыми по объединению

Пример операции пересечения. Создайте список всех городов, в которых есть отделение компании, а также по меньшей мере один объект недвижимости, сдаваемый в аренду.

$$\Pi_{\text{city}}(\text{Branch}) \cap \Pi_{\text{city}}(\text{PropertyForRent})$$

Как и в предыдущем примере, следует создать совместимые по объединению отношения Branch и PropertyForRent, выполнив их проекцию по атрибуту city. Затем для комбинирования полученных новых отношений следует использовать операцию пересечения. Результат выполнения этих операций показан в таблице 3.8.

Таблица 3.8. Пересечение, основанное на атрибуте city отношений Branch и PropertyForRent

City
Aberdeen
London
Glasgow

Пересечение можно сформулировать и на основе операции разности множеств:

$$R \cap S = R - (R - S)$$

Декартово произведение.

R x S. Операция декартова произведения определяет новое отношение, которое является результатом конкатенации (т.е. сцепления) каждого кортежа из отношения R с каждым кортежем из отношения S.

Операция декартова произведения применяется для умножения двух отношений. Умножением двух отношений называется создание другого отношения, состоящего из всех возможных пар кортежей обоих отношений. Следовательно, если одно отношение имеет I кортежей и N атрибутов, а другое — J кортежей и M атрибутов, то их декартово произведение будет содержать $(I \times J)$ кортежей и $(N + M)$ атрибутов. Исходные отношения могут содержать атрибуты с одинаковыми именами. В таком случае имена атрибутов будут содержать названия отношений в виде префиксов для обеспечения уникальности имен атрибутов в отношении, полученном как результат выполнения операции декартова произведения.

Пример операции декартова произведения. Создайте список всех арендаторов, которые осматривали объекты недвижимости, с указанием сделанных ими комментариев.

Имена арендаторов хранятся в отношении Client, а сведения о выполненных ими осмотрах — в отношении Viewing. Чтобы получить список арендаторов и комментарии об осмотренной ими недвижимости, необходимо объединить эти два отношения:

$(\Pi_{\text{clientNo}, \text{fName}, \text{lName}}(\text{Client})) \times (\Pi_{\text{clientNo}, \text{propertyNo}, \text{comment}}(\text{Viewing}))$

Результаты выполнения этой операции показаны в таблице 3.9. В таком виде это отношение содержит больше информации, чем необходимо. Например, первый кортеж этого отношения содержит разные значения атрибута clientNo. Для получения искомого списка необходимо для этого отношения произвести операцию выборки с извлечением тех кортежей, для которых выполняется равенство $\text{Client.clientNo} = \text{Viewing.clientNo}$. Полностью эта операция выглядит так, как показано ниже.

$\sigma_{\text{client.clientNo} = \text{Viewing.clientNo}}(\Pi_{\text{clientNo}, \text{fName}, \text{lName}}(\text{Client})) \times (\Pi_{\text{clientNo}, \text{propertyNo}, \text{comment}}(\text{Viewing}))$

Результат выполнения этой операции показан в таблице 3.10. Комбинация декартова произведения и выборки может быть сведена к одной операции *соединения*.

Таблица 3.10. Декартово произведение сокращенного варианта отношений Client и Viewing (использованы только некоторые атрибуты)

Client.client No	fName	lName	Viewing.client No	property No	Comment
CR76	John	Kay	CR56	PA14	Too small (Слишком мала)
CR76	John	Kay	CR76	PG4	Too remote (Слишком далеко)
CR76	John	Kay	CR56	PG4	
CR76	John	Kay	CR62	PA14	No Dining room (Нет отдельной столовой)
CR76	John	Kay	CR56	PG36	
CR56	Aline	Stewart	CR56	PA14	Too small (Слишком мала)
CR56	Aline	Stewart	CR76	PG4	Too remote (Слишком далеко)
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR62	PA14	No Dining room (Нет отдельной столовой)
CR56	Aline	Stewart	CR56	PG36	
CR74	Mike	Ritchie	CR56	PA14	Too small (Слишком мала)
CR74	Mike	Ritchie	CR76	PG4	Too remote (Слишком далеко)
CR74	Mike	Ritchie	CR56	PG4	
CR74	Mike	Ritchie	CR62	PA14	No Dining room (Нет отдельной столовой)
CR74	Mike	Ritchie	CR56	PG36	
CR62	Mary	Tregear	CR56	PA14	Too small (Слишком мала)
CR62	Mary	Tregear	CR76	PG4	Too remote (Слишком далеко)
CR62	Mary	Tregear	CR56	PG4	
CR62	Mary	Tregear	CR62	PA14	No Dining room (Нет отдельной столовой)
CR62	Mary	Tregear	CR56	PG36	

Таблица 3.11. Ограниченное декартово произведение сокращенного варианта отношений Client и Viewing (использованы только некоторые атрибуты)

Client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	Too remote (Слишком далеко)
CR56	Aline	Stewart	CR56	PA14	Too small (Слишком мала)
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	No dining room (Нет отдельной столовой)

Декомпозиция сложных операций

Операции реляционной алгебры могут в конечном итоге стать чрезвычайно сложными. Для упрощения такие операции можно разбить на ряд меньших операций реляционной алгебры и присвоить имена результатам промежуточных выражений. Для

присваивания имен результатам операции реляционной алгебры используется операция присваивания, обозначенная стрелкой влево (\leftarrow). Выполняемое при этом действие аналогично операции присваивания в языке программирования; в данном случае результат выражения справа от знака операции \leftarrow присваивается выражению, находящемуся слева от знака операции. В частности, в предыдущем примере выполняемые операции можно представить следующим образом:

```
TempViewing(clientNo, propertyNo, comment)  $\leftarrow$   $\Pi_{\text{clientNo}, \text{propertyNo}, \text{comment}}$ (Viewing)
TempClient(clientNo, fName, lName)  $\leftarrow$   $\Pi_{\text{clientNo}, \text{fName}, \text{lName}}$ (Client)
Comment(clientNo, fName, lName, vclientNo, propertyNo, comment)  $\leftarrow$  TempClient x
TempViewing
Result  $\leftarrow$   $\sigma_{\text{clientNo}=\text{vclientNo}}$ (Comment)
```

Еще один вариант состоит в использовании операции переименования ρ (обозначается греческой буквой "ро"), который позволяет присвоить имя результатам операции реляционной алгебры. Операция переименования позволяет также задать произвольное имя для любого из атрибутов нового отношения.

$\rho_s(E)$ или $\rho_{s(a_1, \dots, a_n)}(E)$. Операция переименования позволяет присвоить новое имя S выражению E , а также дополнительно переименовать атрибуты как a_1, a_2, \dots, a_n .

3.4.3. Операции соединения

Как правило, пользователей интересует лишь некоторая часть всех комбинаций кортежей декартова произведения, которая удовлетворяет заданному условию. Поэтому вместо декартова произведения обычно используется одна из самых важных операций реляционной алгебры — операция соединения. В результате ее выполнения на базе двух исходных отношений создается некоторое новое отношение. Операция соединения является производной от операции декартова произведения, так как она эквивалентна операции выборки из декартова произведения двух операндов-отношений тех кортежей, которые удовлетворяют условию, указанному в предикате соединения в качестве формулы выборки. С точки зрения эффективной реализации в реляционных СУБД эта операция является одной из самых сложных и часто оказывается одной из основных причин, вызывающих проблемы с производительностью, свойственные всем реляционным системам.

Ниже перечислены различные типы операций соединения, которые несколько отличаются друг от друга и могут быть в той или иной степени полезны.

- Тета-соединение (theta join).
- Соединение по эквивалентности (equijoin), которое является частным видом тета-соединения.

- Естественное соединение (natural join).
- Внешнее соединение (outer join).
- Полусоединение (semijoin).

Тета-соединение

$R \bowtie_F S$. Операция тета-соединения определяет отношение, которое содержит кортежи из декартова произведения отношений R и S , удовлетворяющие предикату F . Предикат F имеет вид $R.a_i \Theta S.b_i$, где вместо Θ может быть указана одна из операций сравнения ($<$, $<=$, $>$, $>=$, $=$ или \neq).

Обозначение тета-соединения можно переписать на основе базовых операций выборки и декартова произведения, как показано ниже.

$$R \bowtie_F S = \sigma_F(R \times S)$$

Так же как и в случае с декартовым произведением, степенью тета-соединения называется сумма степеней операндов-отношений R и S . Если предикат F содержит только операцию сравнения по равенству ($=$), то соединение называется *соединением по эквивалентности* (equi-join).

Пример операции соединения по эквивалентности.

Создайте список всех арендаторов, которые осматривали объект недвижимости, с указанием их имен и сделанных ими комментариев.

Результат можно получить с помощью операции соединения по эквивалентности.

$$(\Pi_{clientNo, fName, lName}(Client)) \bowtie_{Client.clientNo=Viewing.clientNo} (\Pi_{clientNo, propertyNo, comment}(Viewing))$$

или

$$Result \leftarrow TempClient \bowtie_{TempClient.clientNo=TempViewing.clientNo} TempViewing$$

Результат этих операции показан в таблице 3.11.

Естественное соединение

$R \bowtie S$. Естественным соединением называется соединение по эквивалентности двух отношений R и S , выполненное по всем общим атрибутам x , из результатов которого исключается по одному экземпляру каждого общего атрибута.

Степенью естественного соединения называется сумма степеней операндов-отношений R и S за вычетом количества атрибутов x .

Пример операции естественного соединения.

Создайте список всех арендаторов, которые осматривали объекты недвижимости, с указанием их имен и сделанных ими комментариев.

В предыдущем примере для составления этого списка использовалось соединение по эквивалентности, но в нем присутствовали два атрибута $clientNo$. Для удаления одного из этих атрибутов можно воспользоваться операцией естественного соединения.

$$(\Pi_{clientNo, fName, lName}(Client)) \bowtie (\Pi_{clientNo, propertyNo, comment}(Viewing))$$

или

$$Result \leftarrow TempClient \bowtie TempViewing$$

Результат этих операций показан в таблице 3.12.

Таблица 3.12. Естественное соединение сокращенного варианта отношений Client и Viewing (использованы только некоторые атрибуты)

PttrNo	FName	lName	PropertyNo	comment
CR76	John	Kay	PG4	Too remote (Слишком далеко)
CR56	Aline	Stewart	PA14	Too small (Слишком мала)
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA14	No dining room (Нет отдельной столовой)

Внешнее соединение

При соединении двух отношений часто возникает такая ситуация, что для кортежа одного отношения не находится соответствующий кортеж в другом отношении. Иначе говоря, в столбцах соединения оказываются несовпадающие значения. Но иногда может потребоваться, чтобы строка из одного отношения была представлена в результатах соединения, даже если в другом отношении нет совпадающего значения. Эта цель может быть достигнута с помощью внешнего соединения.

$R \supset \bowtie S$. Левым внешним соединением называется соединение, при котором в результирующее отношение включается также кортежи отношения R, не имеющие совпадающих значений в общих столбцах отношения S.

Для обозначения отсутствующих значений во втором отношении используется значение NULL. Внешнее соединение применяется в реляционных СУБД все чаще, к тому же в настоящее время для его выполнения предусмотрена операция, которая включена в новый стандарт SQL. Преимуществом внешнего соединения является то, что после его выполнения сохраняется исходная информация, т.е. внешнее соединение сохраняет кортежи, которые были бы исключены при использовании других типов соединения.

Пример левого внешнего соединения.

Создайте отчет о ходе проведения осмотров объектов недвижимости.

В данном случае необходимо создать отношение, состоящее как из перечня осматриваемых клиентами объектов недвижимости (с приведением их комментариев по этому поводу), так и перечня объектов недвижимости, которые еще не осматривались. Это можно сделать с помощью следующего внешнего соединения.

$$(\Pi_{clientNo, street, city}(PropertyFor Rent)) \supset \bowtie Viewing$$

Результат этой операции показан в таблице 3.13.

Таблица 3.13. Левое внешнее соединение отношений PropertyForRent и Viewing

PropertyNo	Street	City	clientNo	viewDate	comment
PA14	16 Holhead	Aberdeen	CR56	24-May-01	Too small (Слишком мала)
PA14	16 Holhead	Aberdeen	CR62	14-May-01	No dining room (Нет отдельной столовой)
PL94	6 Argyll St	London	NULL	NULL	NULL
PG4	6 Lawrence St	Glasgow	CR76	20-Apr-01	Too remote (Слишком далеко)
PG4	6 Lawrence St	Glasgow	CR56	26-May-01	
PG36	2 ManorRd	Glasgow	CR56	28-Apr-01	
PG21	18 Dale Rd	Glasgow	NULL	NULL	NULL
PG16	5 Novar Dr	Glasgow	NULL	NULL	NULL

Строго говоря, в примере показано *левое внешнее соединение*, поскольку в результирующем отношении содержатся все кортежи левого отношения. Существует также *правое внешнее соединение*, называемое так потому, что в результирующем отношении содержатся все кортежи правого отношения. Кроме того, существует и *полное внешнее соединение*, в результирующее отношение которого помещаются все кортежи из обоих отношений и в котором для обозначения несовпадающих значений кортежей используются значения NULL.

Полусоединение

$R \triangleright_F S$. Операция полусоединения определяет отношение, содержащие те кортежи отношения R, которые входят в соединение отношений R и S.

Преимущество полусоединения заключается в том, что оно позволяет сократить количество кортежей, которые нужно обработать для получения соединения. Это особенно полезно при вычислении соединений в распределенных системах. Операцию полусоединения можно сформулировать и с помощью операций проекции и соединения:

$$R \triangleright_F S = \Pi_A(R \triangleright \triangleleft_F S)$$

Здесь A — это набор всех атрибутов в отношении R. В действительности это — тета-полусоединение, причем следует отметить, что существуют также полусоединения по эквивалентности и естественные полусоединения.

Пример операция полусоединения.

Создайте отчет, содержащий полную информацию обо всех сотрудниках, работающих в отделении компании, расположенном в городе 'Glasgow'.

Если нас интересуют только атрибуты отношения Staff, то мы можем использовать следующую операцию полусоединения, которая приводит к созданию отношения, приведенного в таблице 3.14.

$$Staff \triangleright_{staff.branchNo=branch.branchNo \text{ and } branch.city='Glasgow'} Branch$$

Таблица 3.14. Результат полусоединения отношений Staff и Branch

staffNo	fName	lName	position	sex	DOB	salary	branch No
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

3.4.4. Деление

Операция деления может применяться в случае запросов особого типа, которые довольно часто встречаются в приложениях баз данных. Предположим, что отношение R определено на множестве атрибутов A, а отношение S — на множестве атрибутов B, причем $B \subseteq A$ (т.е. B является подмножеством A). Пусть $C=A-B$, т.е. C является множеством атрибутов отношения R, которые не являются атрибутами отношения S. Тогда определение операции деления будет выглядеть следующим образом.

$R \div S$. Результатом операции деления является набор кортежей отношения R, определенных на множестве атрибутов C, которые соответствуют комбинации всех кортежей отношения S.

Эту операцию можно представить и с помощью других основных операций:

$T_1 \leftarrow \Pi_c(R)$

$T_2 \leftarrow \Pi_c((S \times T_1) - R)$

$T \leftarrow T_1 - T_2$

Пример операции деления отношений

Создайте список всех арендаторов, которые осматривали объекты недвижимости с тремя комнатами.

Для решения поставленной задачи сначала следует с помощью операции выборки выполнить поиск всех трехкомнатных объектов недвижимости, а затем посредством операции проекции получить отношение, содержащее номера только этих объектов недвижимости. После этого нужно применить приведенную ниже операцию деления и получить новое отношение, как показано в таблицах 3.15-3.17.

$(\Pi_{\text{clientNo,propertyNo}}(\text{Viewing})) \div (\Pi_{\text{propertyNo}}(\sigma_{\text{rooms}=3}(\text{PropertyForRent})))$

Таблица 3.15. Результат применения операции проекции $\Pi_{\text{clientNo}, \text{propertyNo}}(\text{Viewing})$ к отношению Viewing

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36

Таблица 3.16. Результат применения операции выборки $\Pi_{\text{propertyNo}}(\sigma_{\text{rooms}=3}(\text{PropertyForRent}))$ к отношению PropertyForRent

ClientNo
PG4
PG36

Таблица 3.17. Результат применения операции деления к результатам двух предыдущих операций

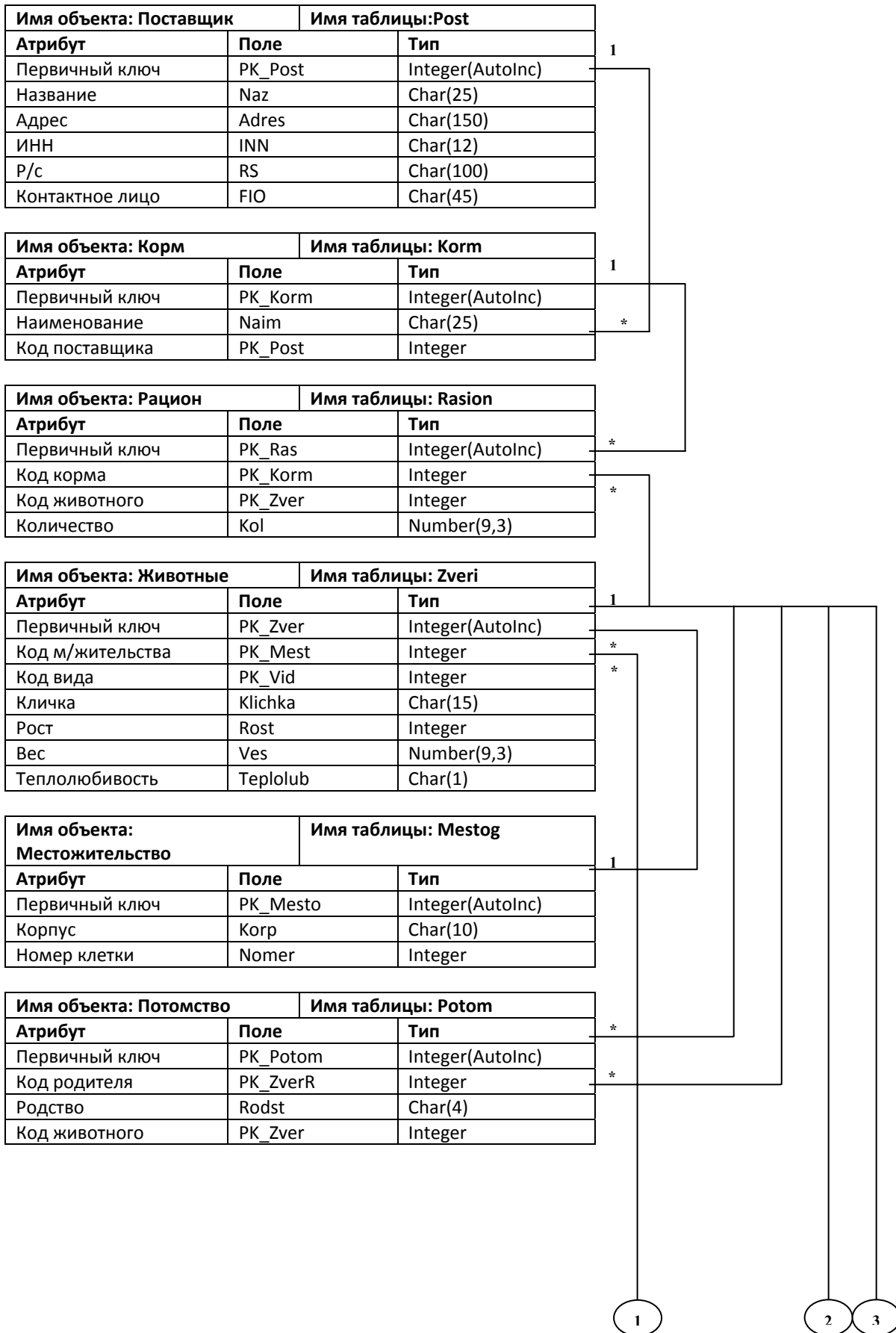
ClientNo
CR56

3.5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие свойства имеет реляционная модель?
2. Что является базовым языковым объектом реляционной модели?
3. Что представляют собой кортеж и домен?
4. Какие требования к таблице предъявляет реляционная модель?
5. Назовите правила целостности.
6. Дайте определение первичного ключа в реляционной модели.
7. В чем заключается отличие внешнего ключа от потенциального?
8. Сколько существует правил Кодда? Для чего они используются?
9. Какая СУБД является реляционной?
10. С какой целью выполняют нормализацию таблиц?
11. Сколько нормальных форм вы знаете? Дайте определение 2НФ и 3НФ.
12. В силу каких причин иногда отступают от полной нормализации данных проекта?
13. В чем состоит отличие OLTP и OLAP систем?
14. Каким образом переводят объектно-ориентированную модель в реляционную?
15. Каким образом в реляционной модели избавляются от связи много-ко-многим?
16. Перечислите операции над множествами.
17. Перечислите операции соединения.

3.6 УПРАЖНЕНИЯ

Задание: Перевести концептуальную модель, описывающую зоопарк, в реляционную.



Название	Naz	Char(15)
----------	-----	----------

Имя объекта: Совместимость		Имя таблицы: Sovm
Атрибут	Поле	Тип
Первичный ключ	PK_Sovm	Integer(AutoInc)
Код_Кто	PK_Vid	Integer
Код_С_кем	PK_Vid	Integer

Задание: Перевести концептуальную модель, полученную во втором модуле, в реляционную. В качестве СУБД в дальнейшем будет использоваться Visual FoxPro.

3.7 ТЕСТЫ

1. Реляционная модель предусматривает организацию данных исключительно в виде
 - 1) множеств
 - 2) массивов
 - 3) списков
 - 4) таблиц
 - 5) нет правильного ответа
2. Кортеж - это
 - 1) атрибут
 - 2) ячейка
 - 3) столбец
 - 4) запись
 - 5) нет правильного ответа
3. Домен - это набор
 - 1) допустимых записей
 - 2) допустимых значений столбца
 - 3) любых записей
 - 4) любых значений столбца
 - 5) нет правильного ответа
4. Реляционная модель включает
 - 1) таблицы
 - 2) операции над таблицами
 - 3) таблицы и операции над ними
5. Нормализация представляет собой
 - 1) процесс совершенствования реляционной модели
 - 2) процесс слияния таблиц
 - 3) процесс слияния столбцов
 - 4) процесс разбиения строк
 - 5) нет правильного ответа
6. Укажите пункт, который не выполняется при использовании нормализованных таблиц
 - 1) Обеспечение целостности
 - 2) Создание формальной модели, как можно более независимой от специфики приложения
 - 3) Снижение времени на разработку приложения
 - 4) Снижение требований к объему памяти

5) Все пункты выполняются

Ответы на тест:

1-4; 2-4; 3-2; 4-3; 5-1; 6-3.

4 СРЕДСТВА АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ БД

4.1 МОДЕЛЬ «СУЩНОСТЬ-СВЯЗЬ»

В предыдущих главах была рассмотрена методика построения объектной модели и реляционной на ее основе. Это не единственный способ построения модели БД. Существуют и другие способы проектирования базы данных.

Одна из наиболее сложных проблем проектирования базы данных связана с тем, что проектировщики, программисты и конечные пользователи, как правило, рассматривают данные и их назначение по-разному. Разработанный проект позволит удовлетворить все требования пользователей только при том условии, что и проектировщики, и пользователи придут к единому пониманию того, как работает данная конкретная организация. Чтобы добиться полного понимания характера данных и способов их использования в организации, необходимо применять в процессе обмена информацией между специалистами общую модель, которая не усложнена техническими подробностями и не допускает двойных толкований. Одним из примеров модели такого типа является модель "сущность-связь" (Entity-Relationship model, или ER-модель). ER-моделирование представляет собой нисходящий подход к проектированию базы данных, который начинается с выявления наиболее важных данных, называемых сущностями (entities), и связей (relationships) между данными, которые должны быть представлены в модели. Затем в модель вносятся дополнительные сведения, например, указывается информация о сущностях и связях, называемая *атрибутами* (attributes), а также все ограничения, относящиеся к сущностям, связям и атрибутам. ER-модель может быть представлена различными способами. Одним из способов представления является методология IDEF1X.

4.2 МЕТОДОЛОГИЯ IDEF1X

Метод IDEF1, разработанный Т.Рэмей (T.Ramey), также основан на подходе П.Чена и позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. В настоящее время на основе совершенствования методологии IDEF1 создана ее новая версия - методология IDEF1X. IDEF1X разработана с учетом таких требований, как простота изучения и возможность автоматизации. IDEF1X-диаграммы используются рядом распространенных CASE-средств (в частности, ERwin, Design/IDEF).

Объекты модели называются сущностями (Entity). Каждая сущность является множеством подобных индивидуальных объектов, называемых экземплярами. Каждый экземпляр индивидуален и должен отличаться от всех остальных экземпляров. Атрибут выражает определенное свойство объекта. С точки зрения БД сущности соответствует таблица, экземпляру сущности - строка в таблице, а атрибуту - колонка таблицы.

Построение модели предполагает определение сущностей и атрибутов,

Сущность в методологии IDEF1X является независимой от идентификаторов или просто независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями. Сущность называется зависимой от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности.

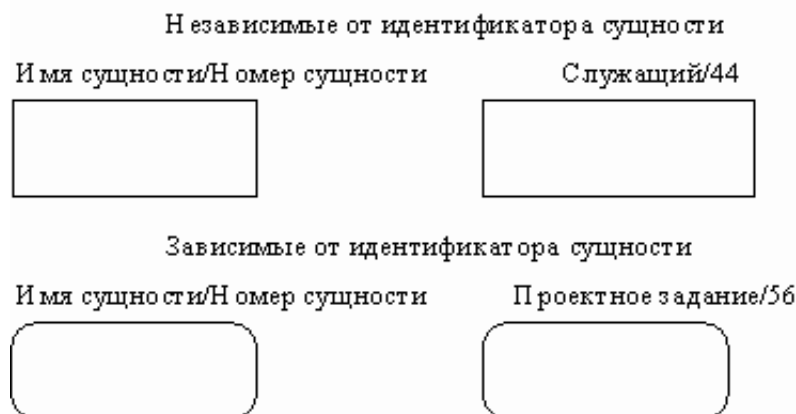


Рис 4.1. Изображение независимых и зависимых сущностей

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой "/" и помещаемые над блоком. Связь является логическим соотношением между сущностями. Каждая связь должна именоваться глаголом или глагольной фразой.

Связь может дополнительно определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). В IDEF1X могут быть выражены следующие мощности связей:

- каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Если экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем, то связь называется **идентифицирующей**, в противном случае - **неидентифицирующей**.

Связь изображается линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка.

Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией. Сущность-потомок в идентифицирующей связи является

зависимой от идентификатора сущностью. Сущность-родитель в идентифицирующей связи может быть как независимой, так и зависимой от идентификатора сущностью (это определяется ее связями с другими сущностями).

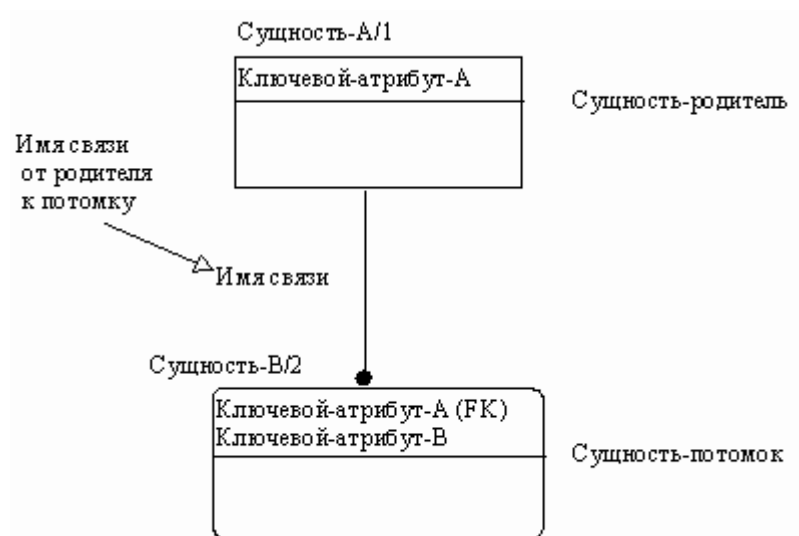


Рис 4.2. Пример идентифицирующей связи

Пунктирная линия изображает неидентифицирующую связь. Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей связи.

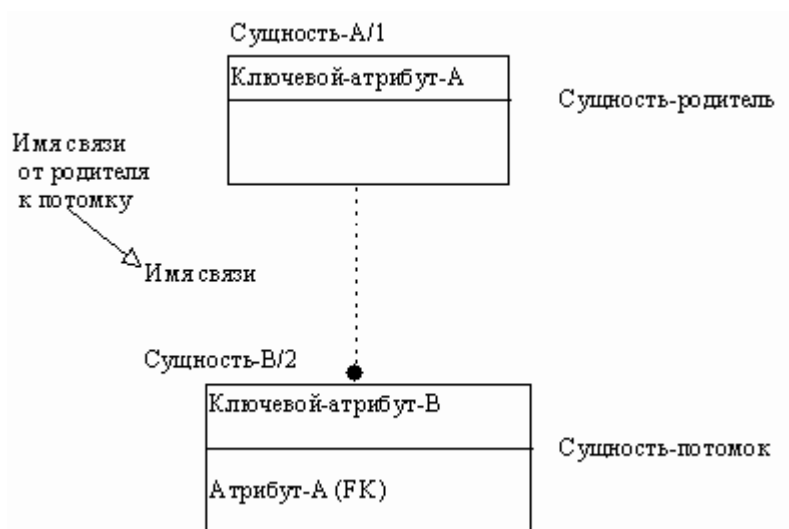


Рис. 4.3. Пример неидентифицирующей связи

Отдельные свойства сущностей называются *атрибутами*. Например, сущность Staff (Персонал) может быть описана с помощью атрибутов staffNo (Табельный номер работника), name (Имя), position (Должность) и salary (Зарплата). Атрибуты содержат значения, которые описывают каждый экземпляр сущности и составляют основную часть информации, сохраняемой в базе данных.

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой.

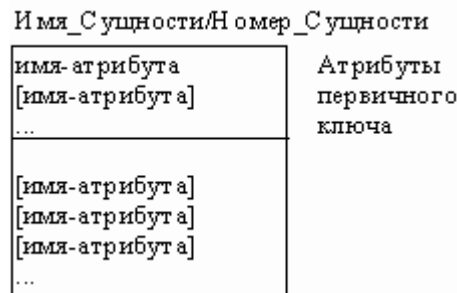


Рис 4.4. Пример определения атрибутов.

Сущности могут иметь также внешние ключи (Foreign Key), которые могут использоваться в качестве части или целого первичного ключа или неключевого атрибута. Внешний ключ изображается с помощью помещения внутрь блока сущности имен атрибутов, после которых следуют буквы FK в скобках.

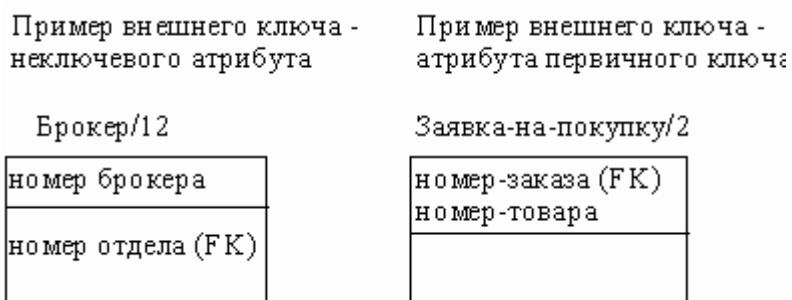


Рис. 4.5. Перенос атрибутов

4.3 ПРОБЛЕМЫ ER-МОДЕЛИРОВАНИЯ

Существуют проблемы, которые принято называть *дефектами соединения* (connection trap). Они обычно возникают «вследствие неправильной интерпретации смысла некоторых связей». Рассмотрим два основных типа дефектов соединения: *дефект типа "разветвление"* (fan trap) и *дефект типа "разрыв"* (chasm trap), а также способы их выявления и устранения в создаваемых ER-моделях. В общем случае для выявления дефектов соединения необходимо убедиться в том, что смысл каждой связи определен четко и ясно. При недостаточном понимании сути установленных связей может быть создана модель, которая не будет являться истинным представлением реального мира.

4.3.1. Дефекты типа "разветвление"

Дефект типа "разветвление" имеет место в том случае, когда модель отображает связь между типами сущностей, но путь между отдельными сущностями этого типа определен неоднозначно.

Дефект типа "разветвление" возникает в том случае, когда две или несколько связей типа 1:* исходят из одной сущности. Потенциальный дефект типа "разветвление" показан на рис. 4.6, на котором две связи типа 1:* (Has и Operates) исходят из одной и той же сущности Division.



Рис. 4.6. Пример дефекта типа "разветвление"

На основании этой модели можно сделать вывод, что один отдел (Division) может состоять из нескольких отделений компании (Branch) и в нем может работать многочисленный штат сотрудников. Проблемы начинаются при попытках выяснить, в каком отделении компании работает каждый из сотрудников отдела.

Неспособность дать точный ответ на поставленный вопрос является результатом дефекта типа «разветвление», связанного с неправильной интерпретацией связей между сущностями Staff, Division и Branch. Устранить эту проблему можно путем перестройки ER-модели для представления правильного взаимодействия этих сущностей таким образом, как показано на рис. 4.7.

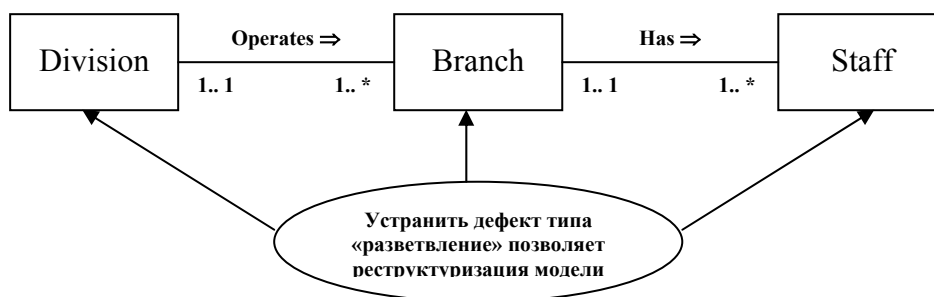


Рис. 4.7. Пример переработки ER-модели (рис. 4.6) с целью устранения дефекта типа "разветвление"

Если проверить эту структуру на уровне отдельных связей *Operates* к *Has*, можно убедиться, что теперь легко дать однозначный ответ на поставленный выше вопрос.

4.3.2. Дефекты типа "разрыв"

Дефект типа «разрыв» появляется в том случае, когда в модели предполагается наличие связи между типами сущностей, но не существует пути между отдельными сущностями этих типов.

Дефект типа "разрыв" может возникать, если существует одна или несколько связей с минимальной кратностью, равной нулю (которая обозначает необязательное участие), и эти связи составляют часть пути между взаимосвязанными сущностями. На рис. 4.8 потенциальный дефект типа "разрыв" показан на примере связей между сущностями Branch, Staff и PropertyForRent.

Рассмотрев эту модель, можно сделать вывод, что одно отделение компании имеет много сотрудников, которые работают со сдаваемыми в аренду объектами. Однако не все сотрудники непосредственно работают с объектами и не все сдаваемые в аренду объекты недвижимости в каждый конкретный момент находятся в ведении кого-либо из

сотрудников компании. В данном случае проблема возникает, когда необходимо выяснить, какие объекты недвижимости приписаны к тому или иному отделению компании.



Рис. 4.8. Пример дефекта типа "разрыв"

Попробуем ответить на следующий вопрос: "Какое отделение компании отвечает за работу с объектом под номером 'PA14'? К сожалению, на данный вопрос нельзя дать ответ, если этот объект в текущий момент не связан ни с одним из сотрудников, работающих в каком-либо из отделений компании. Неспособность дать ответ на заданный вопрос рассматривается как утрата информации (поскольку известно, что любой объект недвижимости *должен* быть приписан к какому-то отделению компании) в результате которой и возникает дефект типа "разрыв". Кратность сущности Staff и PropertyForRent в связи Oversees имеет минимальное значение, равное нулю, а это означает, что некоторые объекты недвижимости не могут быть связаны с отделением компании с помощью информации о сотрудниках. Поэтому для разрешения этой проблемы следует ввести недостающую связь *Offers* между сущностями Branch и PropertyForRent. ER-модель, показанная на рис. 4.9, отображает истинные связи между этими сущностями. Такая структура гарантирует, что всегда будут известны объекты недвижимости, связанные с каждым отделением компании, включая объекты недвижимости которые в данный момент не поручены никому из сотрудников этой компаний.



Рис. 4.9. ER-модель, представленная на рис. 4.8, после переработки с целью устранения дефекта типа "разрыв"

4.4 СОЗДАНИЕ МОДЕЛИ ДАННЫХ С ПОМОЩЬЮ

Toad Data Modeler Freeware

4.4.1 Отображение модели данных в Toad Data Modeler Freeware

4.4.1.1. Физическая и логическая модель данных

Toad Data Modeler Freeware имеет два уровня представления модели - логический и физический. **Логический уровень** - это абстрактный взгляд на данные, на нем данные представляются так, как выглядят в реальном мире, и могут называться так, как они называются в реальном мире, например "Постоянный клиент", "Отдел" или "Фамилия сотрудника". Объекты модели, представляемые на логическом уровне, называются сущностями и атрибутами (подробнее о сущностях и атрибутах будет рассказано ниже). Логическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД.

Физическая модель данных, напротив, зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация о всех объектах БД. Поскольку стандартов на объекты БД не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД. Следовательно, одной и той же логической модели могут соответствовать несколько разных физических моделей. Если в логической модели не имеет значения, какой конкретно тип данных имеет атрибут, то в физической модели важно описать всю информацию о конкретных физических объектах - таблицах, колонках, индексах, процедурах и т. д. Разделение модели данных на логические и физические позволяет решить несколько важных задач.

Документирование модели. Многие СУБД имеют ограничение на именование объектов (например, ограничение на длину имени таблицы или запрет использования специальных символов - пробела и т. п.). Зачастую разработчики ИС имеют дело с нелокализованными версиями СУБД. Это означает, что объекты БД могут называться короткими словами, только латинскими символами и без использования специальных символов (т. е. нельзя назвать таблицу предложением - только одним словом). Кроме того, проектировщики БД нередко злоупотребляют "техническими" наименованиями, в результате таблица и колонки получают наименования типа *RTD_324* или *CUST_A12* и т. д. Полученную в результате структуру могут понять только специалисты (а чаще всего только авторы модели), ее невозможно обсуждать с экспертами предметной области. Разделение модели на логическую и физическую позволяет решить эту проблему. На физическом уровне объекты БД могут называться так, как того требуют ограничения СУБД. На логическом уровне можно этим объектам дать синонимы - имена более понятные неспециалистам, в том числе на кириллице и с использованием специальных символов. Например, таблице *CUST_A12* может соответствовать сущность **Постоянный клиент**. Такое соответствие позволяет лучше задокументировать модель и дает возможность обсуждать структуру данных с экспертами предметной области.

Масштабирование. Создание модели данных, как правило, начинается с создания логической модели. После описания логической модели, проектировщик может выбрать необходимую СУБД и Toad Data Modeler Freeware автоматически создаст соответствующую физическую модель. На основе физической модели Toad Data Modeler Freeware может сгенерировать системный каталог СУБД или соответствующий SQL-скрипт. Этот процесс называется прямым проектированием. Тем самым достигается масштабируемость - создав одну логическую модель данных, можно сгенерировать физические модели под любую поддерживаемую Toad Data Modeler Freeware СУБД. С другой стороны, Toad Data Modeler Freeware способен по содержимому системного каталога или SQL-скрипту воссоздать физическую и логическую модель данных. На основе полученной логической модели данных можно сгенерировать физическую модель для другой СУБД и затем сгенерировать ее системный каталог. Следовательно, Toad Data Modeler Freeware позволяет решить задачу по переносу структуры данных с одного сервера на другой. Например, можно перенести структуру данных с Oracle на Informix (или наоборот) или перенести структуру dbf-файлов в реляционную СУБД, тем самым облегчив решение по переходу от файл-серверной к клиент-серверной ИС. Для того чтобы извлечь выгоды от перехода на клиент-серверную технологию, структуру данных следует модифицировать.

Для переключения между логической и физической моделью данных служит список выбора в левой части панели инструментов Toad Data Modeler Freeware (рис. 4.10).

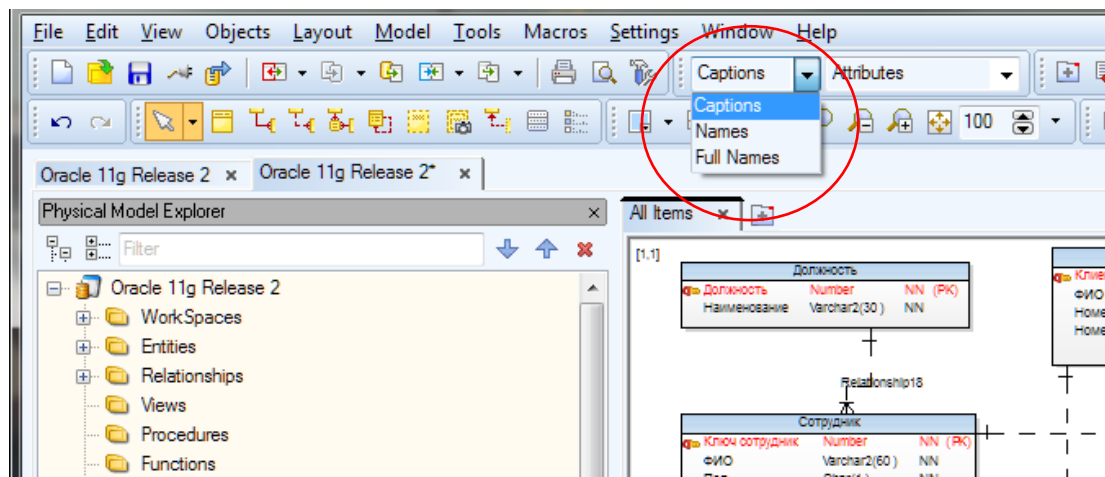


Рис. 4.10. Переключение между логической и физической моделью

При переключении, если физической модели еще не существует, она будет создана автоматически.

4.4.1.2 Интерфейс Toad Data Modeler Freeware. Уровни отображения модели

Интерфейс выполнен в стиле Windows-приложений, достаточно прост и интуитивно понятен.

Палитра инструментов имеет:

- кнопку указателя (Pointer) - в этом режиме можно установить фокус на каком-либо объекте модели;
- кнопку внесения сущности (Entity) - для внесения сущности нужно щелкнуть левой кнопкой мыши по кнопке внесения сущности и один раз по свободному пространству на модели. Повторный щелчок приведет к внесению в модель еще одной новой сущности. Для редактирования сущностей или других объектов модели необходимо перейти в режим указателя;
- кнопки создания связей: идентифицирующую, "многие-ко-многим" и неидентифицирующую и другие;
- кнопку внесения текстового блока (text objects). С ее помощью можно внести текстовый комментарий в любую часть графической модели.

Имеется возможность изменить шрифт и цвет для всех объектов модели или для какой-либо отдельной категории объектов. Для этого служат кнопки Font и Setup Colors.

4.4.2. Создание логической модели данных

4.4.2.1. Уровни логической модели

Различают три уровня логической модели, отличающихся по глубине представления информации о данных:

- диаграмма сущность-связь (Entity Relationship Diagram, ERD);
- модель данных, основанная на ключах (Key Based model, KB);
- полная атрибутивная модель (Fully Attributed model, FA).


Диаграмма сущность-связь представляет собой модель данных верхнего уровня. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Такая диаграмма не слишком детализирована, в нее включаются основные сущности и связи между ними, которые удовлетворяют основным требованиям, предъявляемым к ИС. Диаграмма сущность-связь может включать связи многие-ко-многим и не включать описание ключей. Как правило, ERD используется для презентаций и обсуждения структуры данных с экспертами предметной области.

Модель данных, основанная на ключах, - более подробное представление данных. Она включает описание всех сущностей и первичных ключей и предназначена для представления структуры данных и ключей, которые соответствуют предметной области.

Полная атрибутивная модель - наиболее детальное представление структуры данных: представляет данные в третьей нормальной форме и включает все сущности, атрибуты и связи.

4.4.2.2. Сущности и атрибуты

Основные компоненты диаграммы Toad Data Modeler Freeware - это сущности, атрибуты и связи. Построение модели данных предполагает определение сущностей и атрибутов, т. е. необходимо определить, какая информация будет храниться в конкретной сущности или атрибуте. Сущность можно определить как объект, событие или концепцию, информация о которых должна сохраняться. Сущности должны иметь наименование с четким смысловым значением, именоваться существительным в единственном числе, не носить "технических" наименований и быть достаточно важными для того, чтобы их моделировать. Именованная сущность в единственном числе облегчает в дальнейшем чтение модели. Фактически имя сущности дается по имени ее экземпляра. Примером может быть сущность ***Зверь*** (но не *Звери*!) с атрибутами ***Ключ зверя, Кличка и Место***. На уровне физической модели ей может соответствовать таблица ***Zver*** с колонками ***PK_Zver, Klichka и PK_Mesto***.

Для внесения сущности в модель необходимо на уровне логической модели - "кликнуть" по кнопке сущности на панели инструментов , затем "кликнуть" по тому месту на диаграмме, где необходимо расположить новую сущность. Щелкнув правой кнопкой мыши по сущности и выбрав из всплывающего меню пункт Edit, можно вызвать диалог Edit, в котором определяются имя, описание и комментарии сущности.

Каждый атрибут хранит информацию об определенном свойстве сущности, а каждый экземпляр сущности должен быть уникальным. Атрибут или группа атрибутов, которые идентифицируют сущность, называется первичным ключом. Для описания атрибутов следует, "кликнув" правой кнопкой по сущности, выбрать в появившемся меню пункт Edit. Появляется диалог Edit (рис. 4.11).

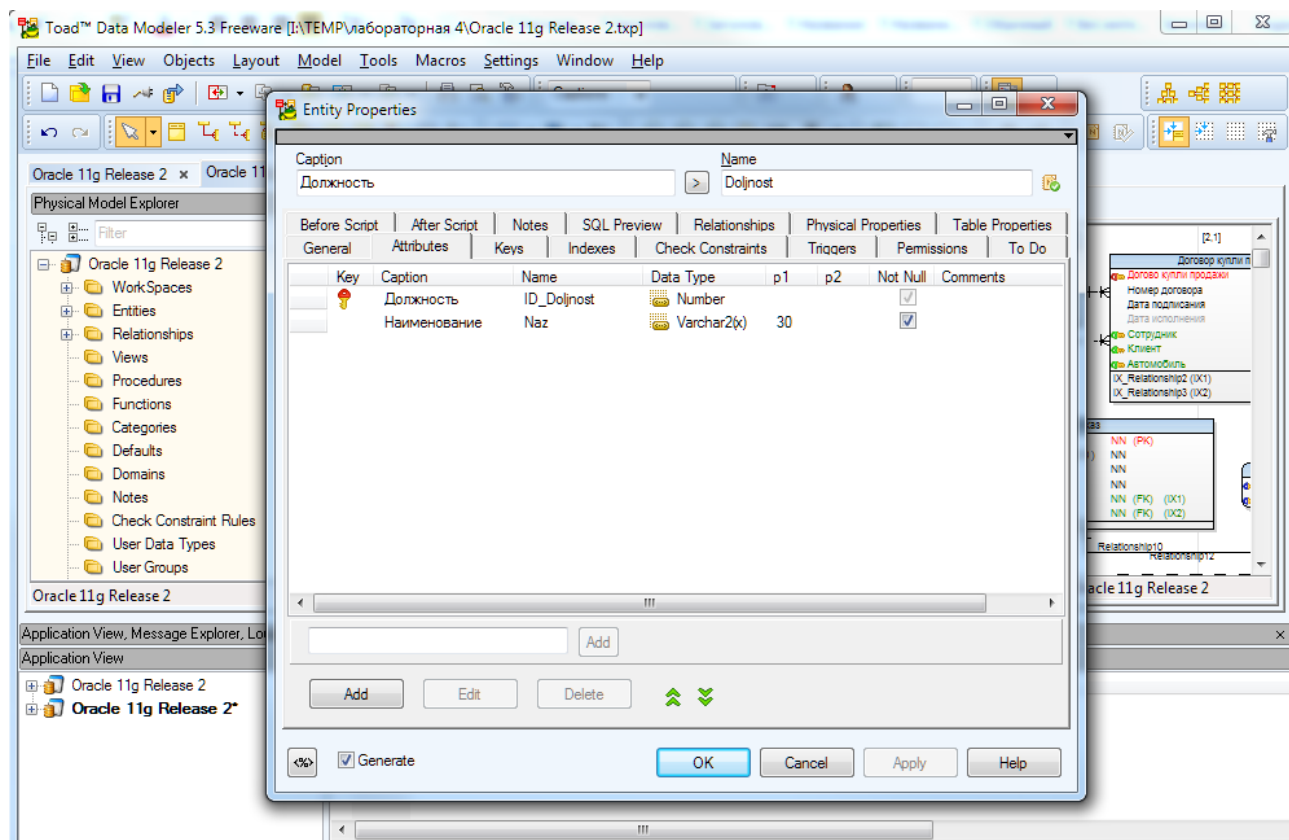


Рис. 4.11. Диалог Edit Entity

Если щелкнуть по кнопке Add, а потом Edit то в появившемся диалоге Attribute (рис. 4.12) можно указать имя атрибута, имя соответствующей ему в физической модели колонки и домен. Домен атрибута будет использоваться при определении типа колонки на уровне физической модели.

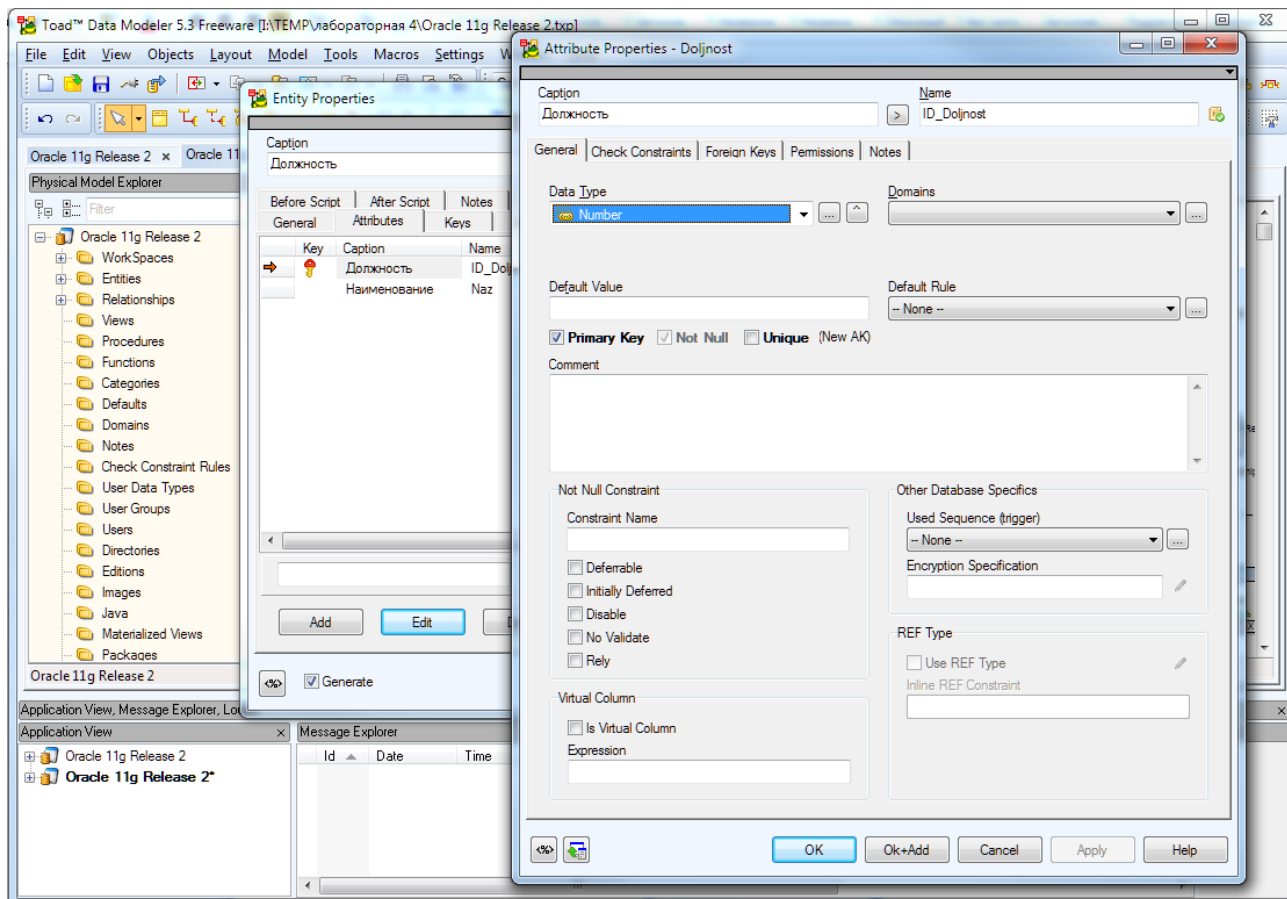


Рис. 4.12. Диалог Attribute

Для атрибутов первичного ключа необходимо сделать пометку в поле Key.

При установлении связей между сущностями атрибуты первичного ключа родительской сущности мигрируют в качестве внешних ключей в дочернюю сущность.

Имя сущности показывается над прямоугольником, изображающим сущность, список атрибутов сущности - внутри прямоугольника. Список разделен горизонтальной чертой, выше которой расположены атрибуты первичного ключа, ниже - неключевые атрибуты. Очень важно дать атрибуту правильное имя. Атрибуты должны именоваться в единственном числе и иметь четкое смысловое значение. Соблюдение этого правила позволяет частично решить проблему нормализации данных уже на этапе определения атрибутов. Например, создание в сущности *Зверь* атрибута *Прививки зверя* противоречит требованиям нормализации, поскольку атрибут должен быть атомарным, т.е. не содержать множественных значений. Согласно синтаксису IDEF1X имя атрибута должно быть уникально в рамках модели (а не только в рамках сущности!). По умолчанию при попытке внесения уже существующего имени атрибута Toad Data Modeler Freeware переименовывает его. Например, если атрибут *Комментарий* уже существует в модели, другой атрибут (в другой сущности) будет назван *Комментарий/2*, затем *Комментарий3* и т. д.

4.4.2.3. Связи

Имя связи выражает некоторое ограничение или бизнес-правило и облегчает чтение диаграммы, например:

- Каждая ПРИВИКА <ставится> ЗВЕРЮ;
- Каждый ЗВЕРЬ <получает> ПРИВИВКУ;

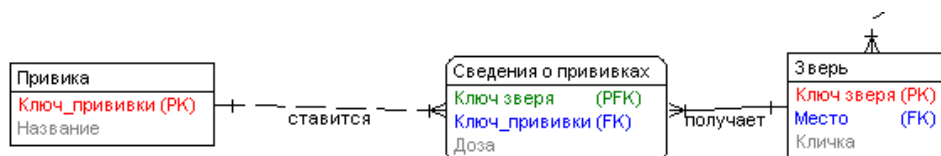


Рис.4.13. Имя связи

Связи показывают, какие именно прививки и какому зверю поставлены. По умолчанию имя связи на диаграмме не показывается. Для отображения имени следует в пункте меню View выбрать пункт Display Relationship Names.

На логическом уровне можно установить идентифицирующую связь один-ко-многим, связь многие-ко-многим и неидентифицирующую связь один-ко-многим (соответственно это кнопки слева направо в палитре инструментов).

Когда рисуется идентифицирующая связь, Toad Data Modeler Freeware автоматически преобразует дочернюю сущность в зависимую. Экземпляр зависимой сущности определяется только через отношение к родительской сущности, т.е. информация о заказе не может быть внесена и не имеет смысла без информации о клиенте, который его размещает. При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи называется миграцией атрибутов. В дочерней сущности новые атрибуты помечаются как внешний ключ - (PFK).




В дальнейшем, при генерации схемы БД, атрибуты первичного ключа получают признак NOT NULL, что означает невозможность внесения записи в таблицу **Сведения о прививках** без информации о животном.

При установлении неидентифицирующей связи дочерняя сущность остается независимой, а атрибуты первичного ключа родительской сущности мигрируют в состав неключевых компонентов родительской сущности. Неидентифицирующая связь служит для связывания независимых сущностей. Например, экземпляр сущности **Сотрудник** может существовать безотносительно к какому-либо экземпляру сущности **Отдел**, т.е. сотрудник может работать в организации, не числясь в каком-либо отделе.

Для создания новой связи следует:

- установить курсор на нужной кнопке в палитре инструментов (идентифицирующая или неидентифицирующая связь) и нажать левую кнопку мыши;
- щелкнуть сначала по родительской, а затем по дочерней сущности.

Форму линии связи можно изменить. Для этого нужно захватывать мышью нужную линию связи и переносить ее с места на место, пока линия не начнет выглядеть лучше.

В палитре инструментов кнопка  соответствует *идентифицирующей* связи, кнопка  связи *многие-ко-многим* и кнопка  соответствуют *неидентифицирующей* связи.

Для редактирования свойств связи следует "кликнуть" правой кнопкой мыши по связи и выбрать на контекстном меню пункт Edit.

В закладке **Type** появившегося диалога можно задать мощность, имя и тип связи

Мощность связи (Cardinality) - служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней.

Различают четыре типа мощности:

- общий случай, когда одному экземпляру родительской сущности соответствуют 0, 1 или много экземпляров дочерней сущности не помечается каким-либо символом;
- символом **P** помечается случай, когда одному экземпляру родительской сущности соответствуют 1 или много экземпляров дочерней сущности (исключено нулевое значение);
- символом **Z** помечается случай, когда одному экземпляру родительской сущности соответствуют 0 или 1 экземпляр дочерней сущности (исключены множественные значения).

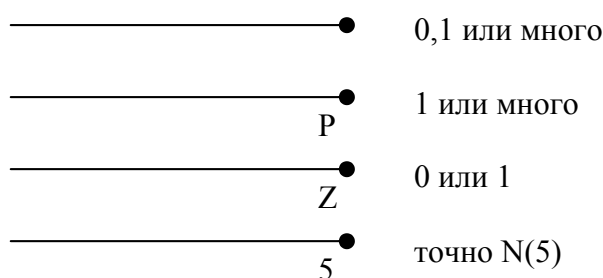


Рис.4.15. Обозначения мощности

Цифрой помечается случай точного соответствия, когда одному экземпляру родительской сущности соответствует заранее заданное число экземпляров дочерней сущности.

По умолчанию символ, обозначающий мощность связи, не показывается на диаграмме. Для отображения имени следует в контекстном меню, которое появляется,

если щелкнуть левой кнопкой мыши по любому месту диаграммы, не занятому объектами модели, выбрать пункт Display Options/Relationship и затем включить опцию Cardinality.

Имя связи (Name) - фраза, характеризующая отношение между родительской и дочерней сущностями.

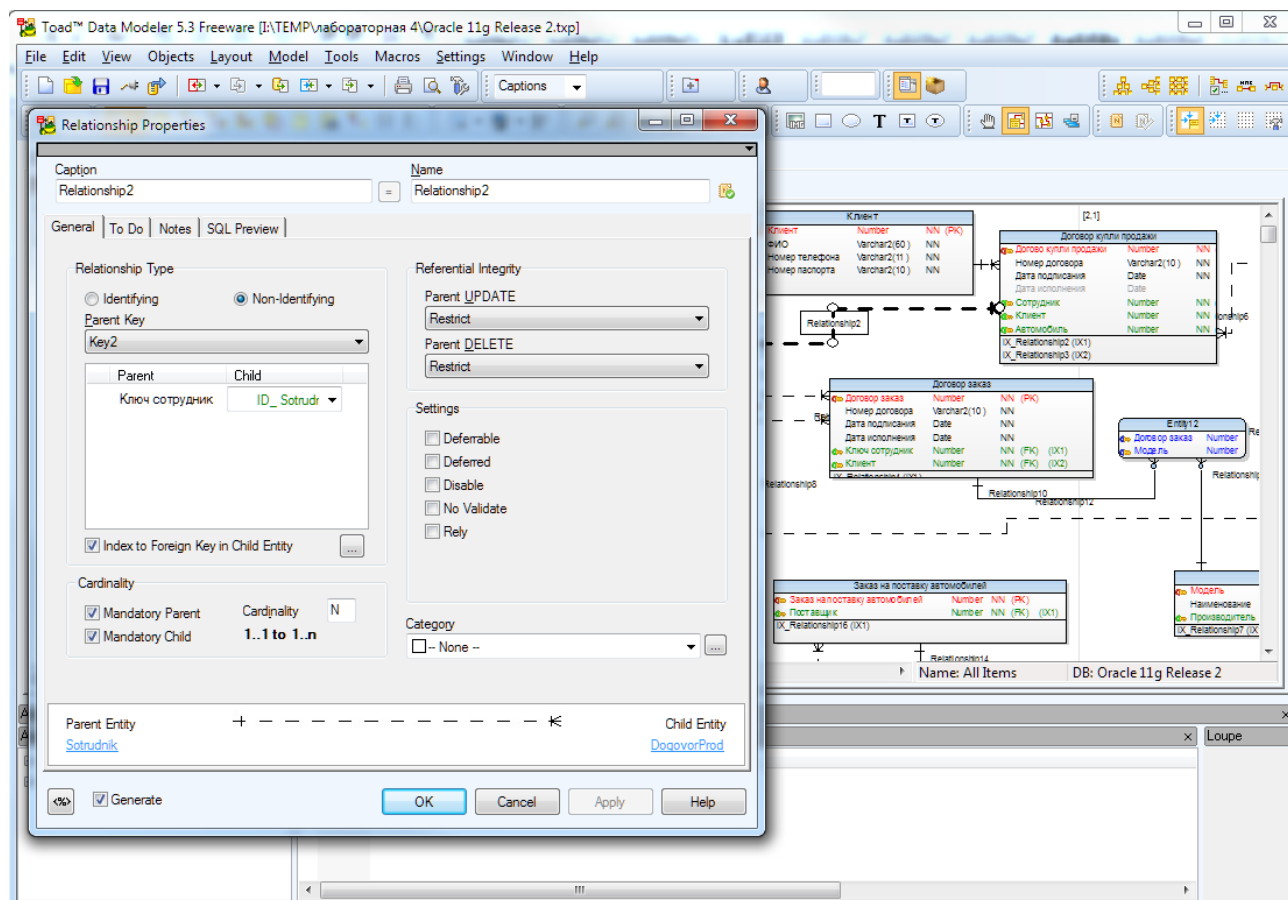


Рис.4.16. Диалог Relationship

Тип связи (идентифицирующая/неидентифицирующая). Для неидентифицирующей связи можно указать обязательность (Nulls). В случае обязательной связи (No Nulls) при генерации схемы БД атрибут внешнего ключа получит признак NOT NULL, несмотря на то что внешний ключ не войдет в состав первичного ключа дочерней сущности. В случае необязательной связи (Nulls Allowed) внешний ключ может принимать значение NULL. Необязательная неидентифицирующая связь помечается прозрачным ромбом со стороны родительской сущности.

Необходимо задать правила ссылочной целостности.

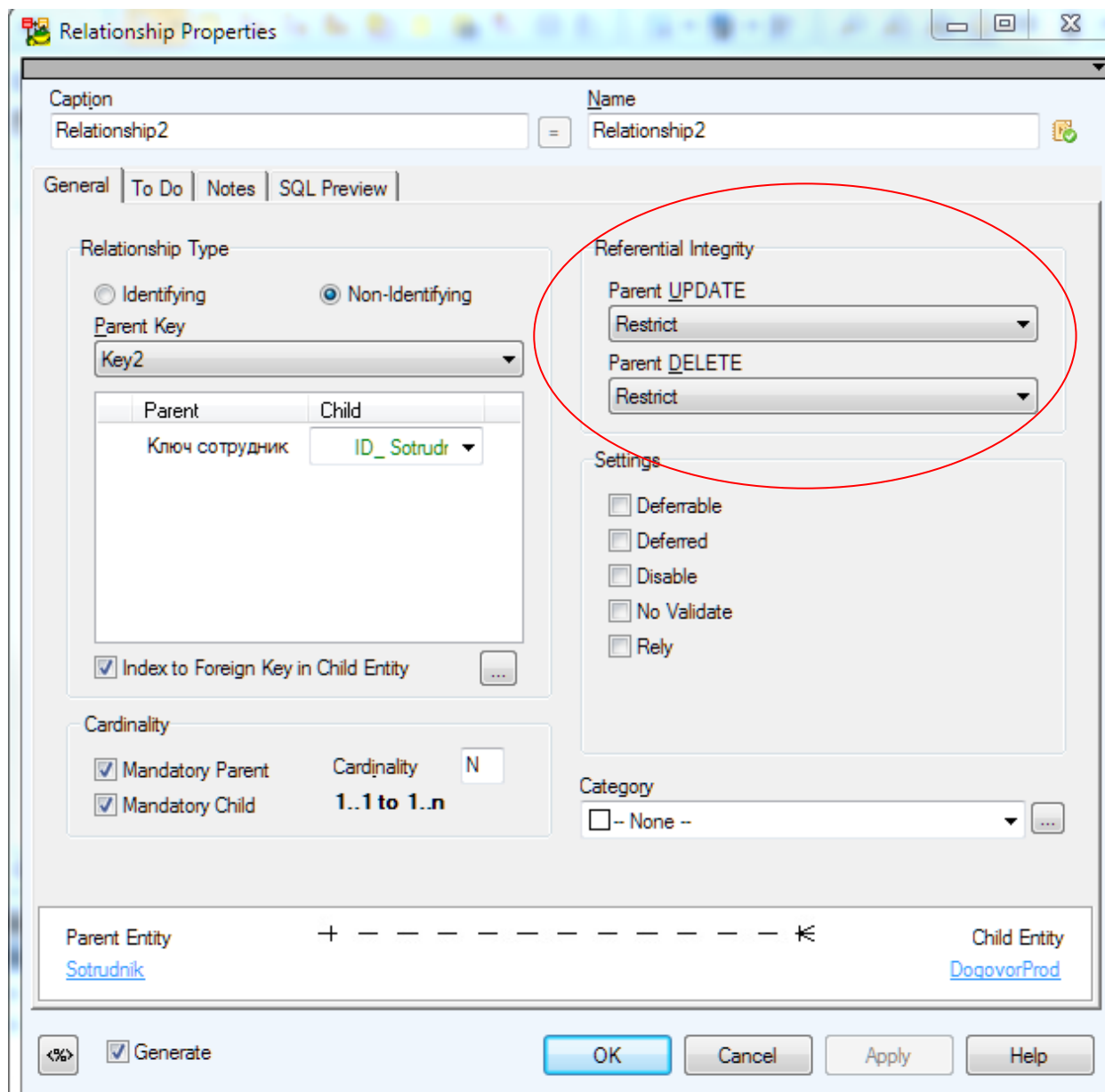


Рис.4.17. Ссылочная целостность в окне диалога Relationship

Триггеры представляют собой программы, выполняемые всякий раз при выполнении команд вставки, замены или удаления (INSERT, UPDATE или DELETE).

Триггеры и хранимые процедуры - это именованные блоки кода SQL, которые заранее откомпилированы и хранятся на сервере для того, чтобы быстро производить выполнение запросов, валидацию данных и выполнять другие часто вызываемые функции.

Хранение и выполнение кода на сервере позволяет создавать код только один раз, а не в каждом приложении, работающем с БД, что экономит время при написании и сопровождении программ. При этом гарантируется, что целостность данных и бизнес-правила поддерживаются независимо от того, какое именно клиентское приложение обращается к данным. Триггеры и хранимые процедуры не требуется пересылать по сети из клиентского приложения, что значительно снижает сетевой трафик.

Хранимой процедурой называется именованный набор предварительно откомпилированных команд SQL, который может вызываться из клиентского приложения или из другой хранимой процедуры.

Триггером называется процедура, которая выполняется автоматически как реакция на событие. Таким событием может быть вставка, изменение или удаление строки в существующей таблице. Триггер сообщает СУБД, какие действия нужно выполнить при выполнении команд SQL INSERT, UPDATE или DELETE для обеспечения дополнительной функциональности, выполняемой на сервере.

Триггер ссылочной целостности - особый вид триггера, используемый для поддержания целостности между двумя таблицами, которые связаны между собой. Если строка в одной таблице вставляется, изменяется или удаляется, то триггер ссылочной целостности (RI-триггер) сообщает СУБД, что нужно делать с теми строками в других таблицах, у которых значение внешнего ключа совпадает со значением первичного ключа вставленной (измененной, удаленной) строки.

Правила удаления управляют тем, что будет происходить в БД при удалении строки. Аналогично правила вставки и обновления управляют тем, что будет происходить с БД, если строки изменяются или добавляются.

Toad Data Modeler Freeware автоматически присваивает каждой связи значение ссылочной целостности, устанавливаемой по умолчанию, прежде чем добавить ее в диаграмму. Режимы RI, присваиваемые Toad Data Modeler Freeware по умолчанию приведены в табл. 4.1.

Таблица 4.1. Значения RI, присваиваемые в Toad Data Modeler Freeware по умолчанию

	Идентифицирующая связь	Неидентифицирующая связь (Nulls Allowed)	Неидентифицирующая связь (No Nulls)	Категориальная связь
Parent Delete Возможные режимы	RESTRICT, CASCADE, NONE	RESTRICT, CASCADE, NONE, SET NULL, SET DEFAULT	RESTRICT, CASCADE, NONE, SET DEFAULT	RESTRICT, CASCADE, NONE
Parent Delete Режимы по умолчанию	RESTRICT	SET NULL	RESTRICT	CASCADE
Parent Insert Возможные режимы	RESTRICT, CASCADE, NONE	RESTRICT, CASCADE, NONE, SET NULL, SET DEFAULT	RESTRICT, CASCADE, NONE, SET DEFAULT	RESTRICT, CASCADE, NONE
Parent Insert Режимы по умолчанию	NONE	NONE	NONE	NONE
Parent Update Возможные режимы	RESTRICT, CASCADE, NONE	RESTRICT, CASCADE, NONE, SET NULL, SET DEFAULT	RESTRICT, CASCADE, NONE, SET DEFAULT	RESTRICT, CASCADE, NONE
Parent Update Режимы по умолчанию	RESTRICT	SET NULL	RESTRICT	CASCADE

В большинстве случаев можно рекомендовать следующие установки для реализации ссылочной целостности. Для неидентифицирующей связи они приведены на рисунке 4.18, а для идентифицирующей - на рисунке 4.19.

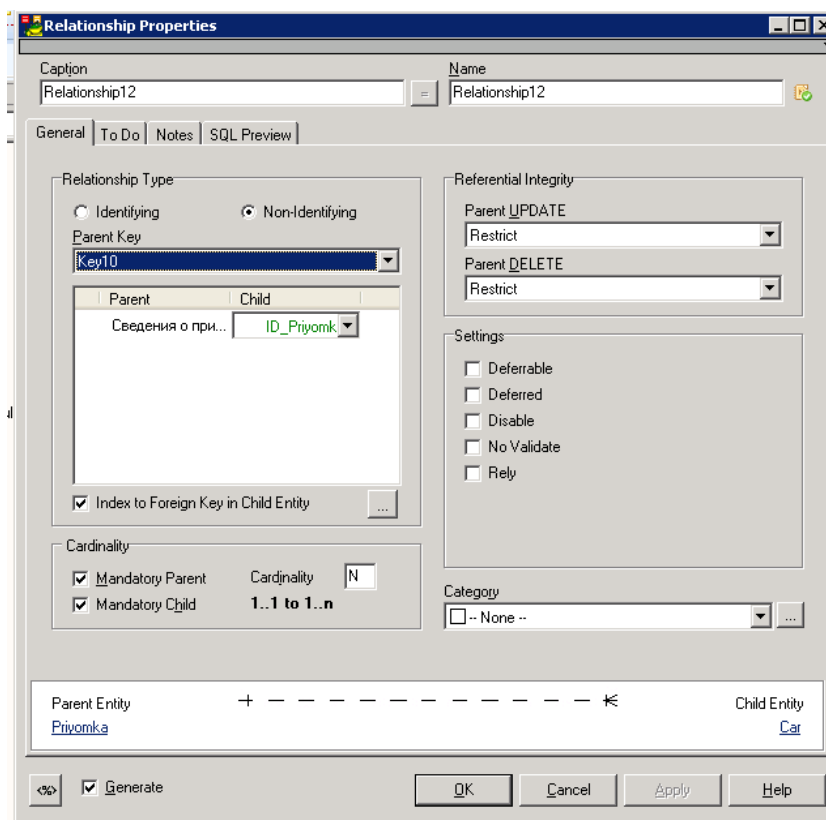


Рис. 4.18. Значения RI для неидентифицирующей связи

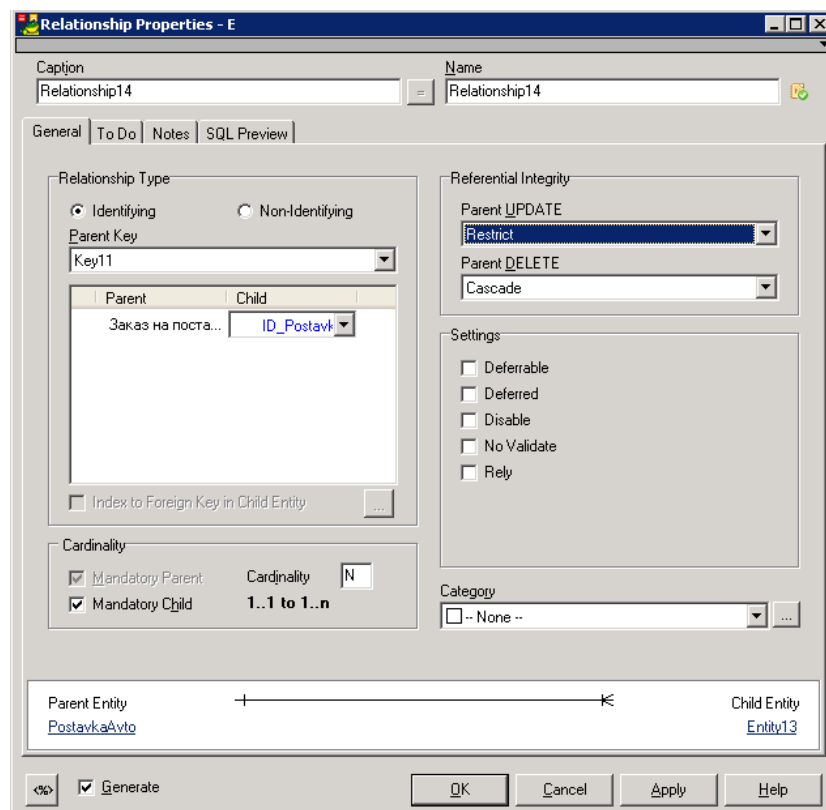



Рис. 4.19. Значения RI для идентифицирующей связи

Связь многие-ко-многим возможна только на уровне логической модели данных. На рис. 4.20 показан пример связи многие-ко-многим. Зверь может получить много прививок, одна и та же прививка может быть поставлена нескольким животным. Такая связь обозначается сплошной линией с двумя точками на концах. Для внесения связи следует установить курсор на кнопку  в палитре инструментов, щелкнуть сначала по одной, а затем по другой сущности.

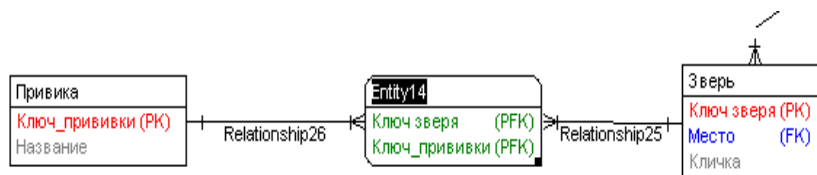


Рис.4.20. Связь многие-ко-многим

Toad Data Modeler Freeware автоматически преобразует связь многие-ко-многим, добавляя новую таблицу и устанавливая две новые связи один-ко-многим от старых к новой таблице (рис. 4.20). При этом имя новой таблице присваивается автоматически.

Такого решения проблемы связи многие-ко-многим не всегда оказывается достаточно. В примере таблица **Entity14** имеет смысл списка прививок конкретного зверя, поэтому ее следует переименовать согласно бизнес-логике в **Сведения о прививках**. Один и тот же зверь может получать различные дозы лекарства в разные периоды, поэтому для того, чтобы идентифицировать конкретную прививку, необходимо в таблицу **Сведения о прививках** добавить дополнительные колонки, например Доза (рис. 4.21).

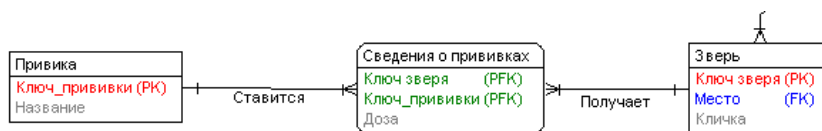


Рис. 4.21. Дополнение модели при разрешении связи многие-ко-многим

4.4.2.4. Типы сущностей

Как было указано выше, связи определяют, является ли сущность независимой или зависимой. Различают несколько типов зависимых сущностей.

Характеристическая - зависимая дочерняя сущность, которая связана только с одной родительской и по смыслу хранит информацию о характеристиках родительской сущности.



Рис .4.22. Пример характеристической сущности "Зверь "

Ассоциативная - сущность, связанная с несколькими родительскими сущностями. Такая сущность содержит информацию о связях сущностей. Примером ассоциативной сущности является *Сведения о прививках* на рис. 4.21.

Именующая - частный случай ассоциативной сущности, не имеющей собственных атрибутов (только атрибуты родительских сущностей, мигрировавших в качестве внешнего ключа). Примером именующей сущности является *Entity14* на рис. 4.20.

Категориальная - дочерняя сущность в иерархии наследования.

4.4.2.5. Ключи

Каждый экземпляр сущности должен быть уникален и отличаться от других атрибутов.

Первичный ключ (primary key) - это атрибут или группа атрибутов, однозначно идентифицирующая экземпляр сущности. Атрибуты первичного ключа на диаграмме не требуют специального обозначения - это те атрибуты, которые находятся в списке атрибутов выше горизонтальной линии. При внесении нового атрибута в диалог Attribute Editor для того, чтобы сделать его атрибутом первичного ключа, нужно включить флажок Key в верхней части закладки Attribute.

Каждая сущность должна иметь по крайней мере один потенциальный ключ. Многие сущности имеют только один потенциальный ключ. Такой ключ становится первичным. Некоторые сущности могут иметь более одного возможного ключа. Тогда один из них становится первичным, а остальные - альтернативными ключами. **Альтернативный ключ (Alternate Key)** - это потенциальный ключ, не ставший первичным.

Внешние ключи (Foreign Key) создаются автоматически, когда связь соединяет сущности: связь образует ссылку на атрибуты первичного ключа в дочерней сущности и эти атрибуты образуют внешний ключ в дочерней сущности (миграция ключа). Атрибуты внешнего ключа обозначаются символом (FK) после своего имени. Зависимая сущность может иметь один и тот же внешний ключ из нескольких родительских сущностей. Сущность может также получить один и тот же внешний ключ несколько раз от одного и того же родителя через несколько разных связей. Когда Toad Data Modeler Freeware обнаруживает одно из этих событий, он распознает, что два атрибута одинаковы, и

помещает атрибут внешнего ключа в зависимой сущности только один раз. Это комбинирование или объединение идентичных атрибутов называется **унификацией**.

Есть случаи, когда унификация нежелательна. Например, когда два атрибута имеют одинаковые имена, но на самом деле они отличаются по смыслу и необходимо, чтобы это отличие отражалось в диаграмме. В этом случае необходимо использовать имена ролей атрибутов внешнего ключа.

4.4.3 Создание физической модели данных

4.4.3.1. Уровни физической модели

Различают два уровня физической модели:

- трансформационная модель (Transformation Model);
- модель СУБД (DBMS Model).

Физическая модель содержит всю информацию, необходимую для реализации конкретной БД. Трансформационная модель содержит информацию для реализации отдельного проекта, который может быть частью общей ИС и описывать подмножество предметной области. Трансформационная модель позволяет проектировщикам и администраторам БД лучше представлять, какие объекты БД хранятся в словаре данных, и проверить, насколько физическая модель данных удовлетворяет требованиям к ИС.

Модель СУБД автоматически генерируется из трансформационной модели и является точным отображением системного каталога СУБД. Toad Data Modeler Freeware непосредственно поддерживает эту модель путем генерации системного каталога.

4.4.3.2. Выбор сервера

Физический уровень представления модели зависит от выбранного сервера. Для выбора СУБД служит редактор Target Database selection, который автоматически запускается при создании новой модели (рис. 4.23).

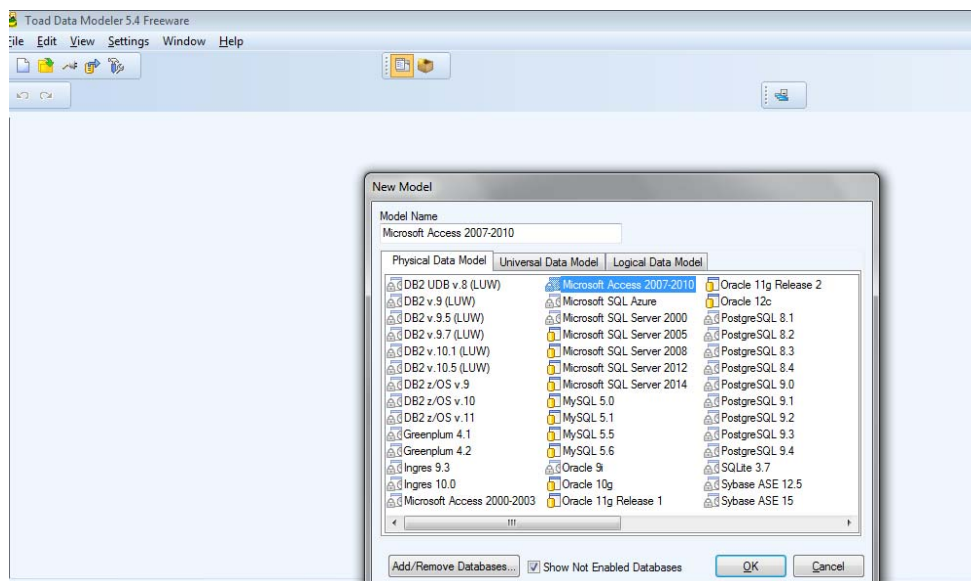


Рис. 4.23. Диалог Target Database selection

Toad Data Modeler Freeware поддерживает практически все распространенные СУБД, всего более 20 реляционных и нереляционных БД. Для выбора СУБД нужно щелкнуть по соответствующей кнопке рядом с именем СУБД.

Генерация базы данных выполняется по окончании построения модели с использованием кнопки на панели инструментов <SQL>. Диалог Script Generating позволяет задать правила ссылочной целостности, принимаемые по умолчанию и другие опции(рис. 4.24).

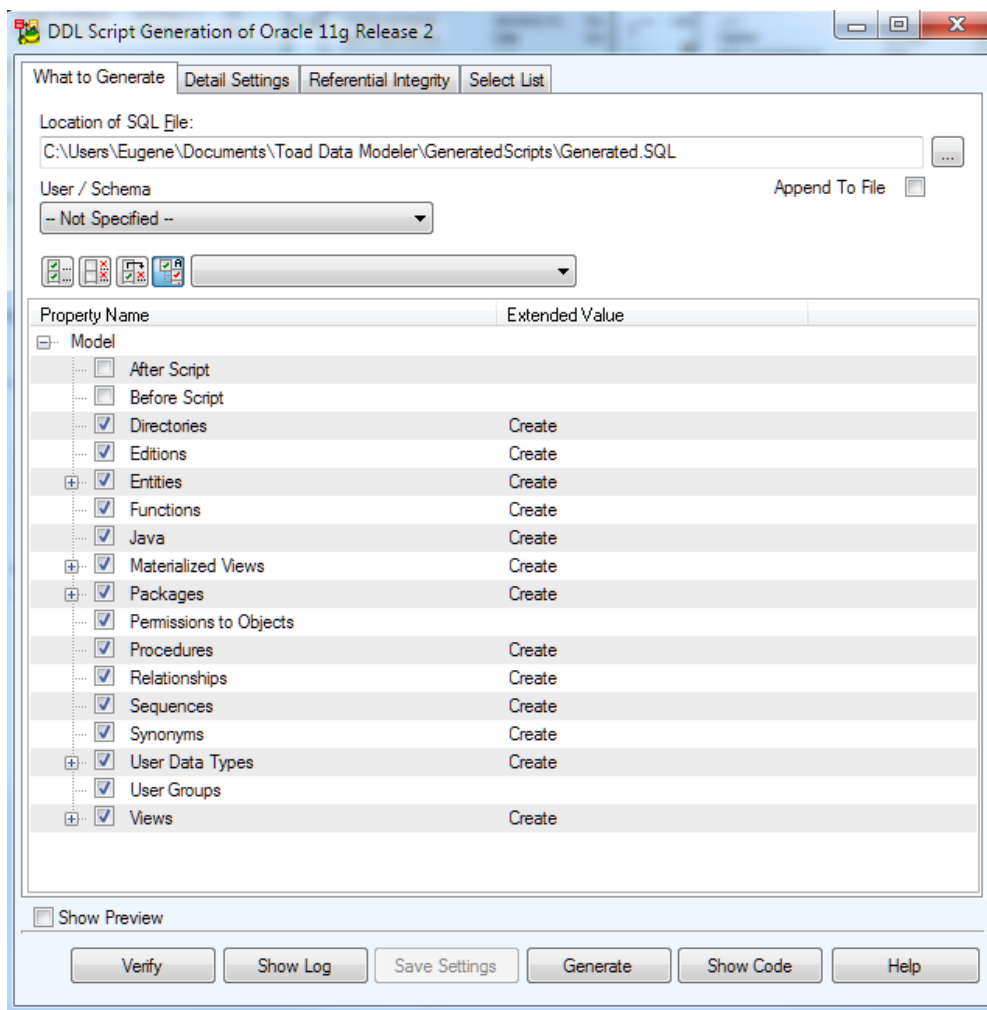



Рис. 4.24. Диалог Script Generating

Имена таблиц и колонок по умолчанию будут сгенерированы на основе имен сущностей и атрибутов логической модели.

Замечание: для корректной работы правил ссылочной целостности во вкладке How to generate диалога Script Generating необходимо выбрать опции Trigger.

4.4.3.3. Таблицы, колонки

Для внесения новой таблицы в модель на физическом уровне служит кнопка  на палитре инструментов. Связи между таблицами создаются так же, как на логическом

уровне. Щелкнув правой клавишей мыши по таблице и выбрав во всплывающем меню пункты **Edit Entity** можно вызвать редакторы для задания свойств таблиц и колонок.

Toad Data Modeler Freeware автоматически создает имена таблиц и колонок на основе имен соответствующих сущностей и атрибутов, учитывая максимальную длину имени и другие синтаксические ограничения, накладываемые СУБД. При генерации имени таблицы или колонки по умолчанию все пробелы автоматически преобразуются в символы подчеркивания, а длина имени обрезается до максимальной длины, допустимой для выбранной СУБД. Редактор **Edit Entity** позволяет задать свойства любой таблицы модели, отличные от значения по умолчанию, в том числе имя таблицы, синонимы, правила валидации, процедуры и т.д. Для задания свойств колонок, отличных от значения по умолчанию, служит редактор **Attribute** (рис. 4.25). Чтобы вызвать его, нужно щелкнуть левой клавишей мыши по колонке.

По умолчанию Toad Data Modeler Freeware присваивает режимы нулевых значений всем неключевым колонкам, исходя из значений по умолчанию, устанавливаемых в редакторе **Target Database selection**. Для колонок первичного ключа и альтернативных ключей устанавливается режим **NOT NULL**.

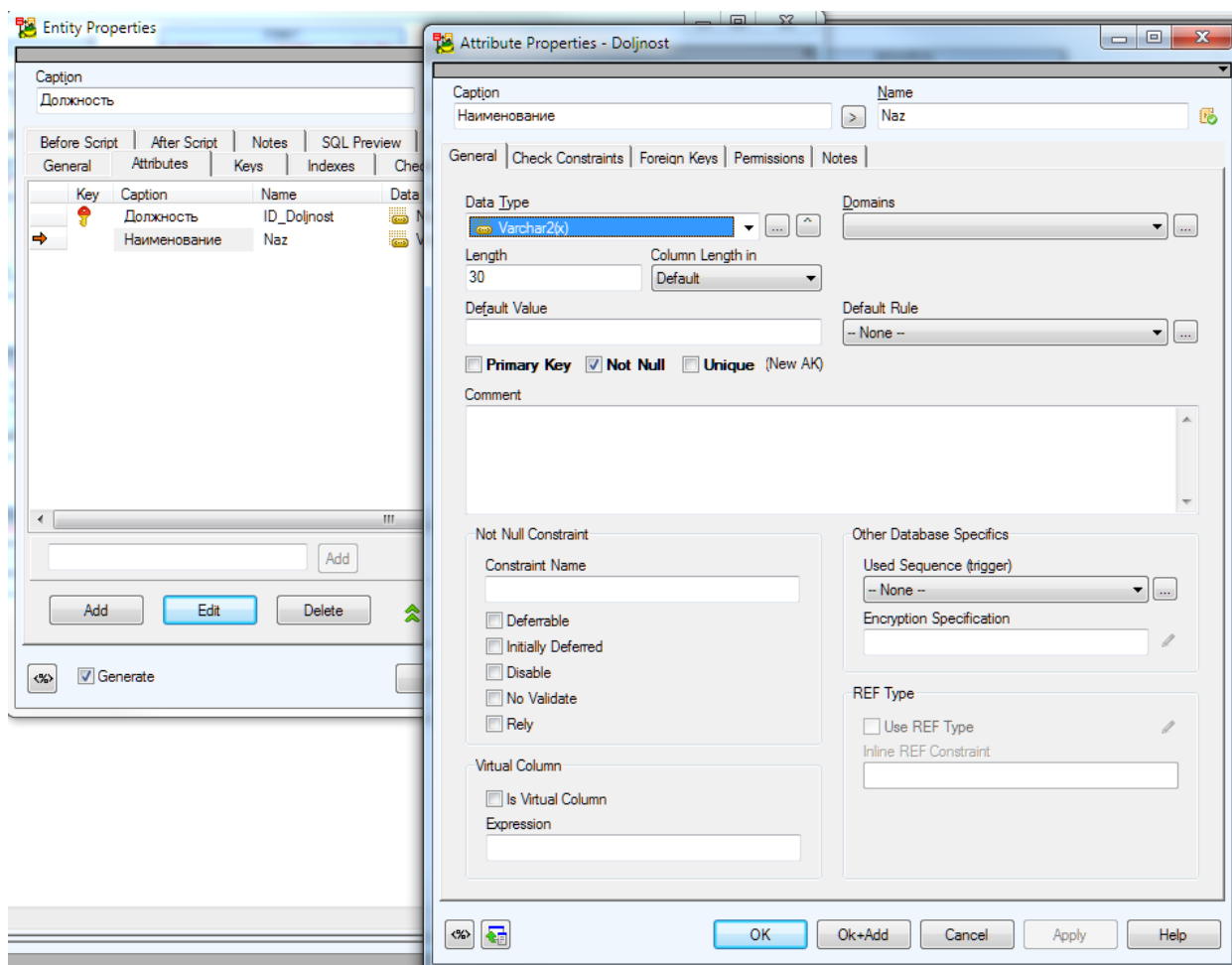


Рис. 4.25. Диалог *Attribute*

Comment. Служит для внесения комментария к каждой колонке.

4.4.3.5. Индексы

В таблице БД данные обычно хранятся в том же порядке, в котором их ввели в таблицу. Многие реляционные СУБД имеют страничную организацию, при которой физически таблица может храниться фрагментарно в разных областях диска, причем строки таблицы располагаются на страницах неупорядоченно. Хотя такой способ хранения и позволяет быстро вводить новые данные, но для того, чтобы найти нужную строку, придется просмотреть всю таблицу. В промышленных системах каждая таблица может содержать миллионы строк, поэтому простой перебор ведет к катастрофическому падению производительности ИС.

Чтобы решить проблему поиска данных, СУБД использует особый объект, называемый индексом. Он подобен содержанию книги, которое указывает на все номера страниц, посвященных конкретной теме. Индекс содержит отсортированную по колонке или нескольким колонкам информацию и указывает на строки, в которых хранится конкретное значение колонки.

При генерации схемы физической БД Toad Data Modeler Freeware автоматически создает отдельный индекс на основе первичного ключа каждой таблицы, а также на основе всех альтернативных ключей, внешних ключей и инверсионных входов, поскольку эти колонки наиболее часто используются для поиска данных. Можно отказаться от генерации индексов по умолчанию и для повышения производительности создать собственные индексы. Администратор СУБД должен анализировать наиболее часто выполняемые запросы и создавать индексы с различными колонками и порядком сортировки для увеличения эффективности поиска при работе конкретных приложений.

При создании индекса на основе ключа Toad Data Modeler Freeware вводит в его состав все колонки ключа. Toad Data Modeler Freeware автоматически генерирует имя индекса, созданного на основе ключа по принципу "X" + имя ключа + имя таблицы (физическое имя таблицы, а не логическое имя сущности!), где имя ключа "PK" для первичного ключа, "IFn" - для внешнего, "AKn" - для альтернативного, "IEн" - для инверсионного входа. Изменить характеристики существующего индекса или создать новый можно в редакторе Indexes. Для его вызова следует вызвать соответствующую вкладку в диалоге Entity. Во вкладке Indexes можно изменить имя индекса, изменить его определение так, чтобы он принимал уникальные или дублирующиеся значения, или изменить порядок сортировки данных.

Toad Data Modeler Freeware создает индексы, которые могут содержать либо повторяющиеся, либо только уникальные значения. При создании нового уникального индекса следует включить опцию Unique в диалоге Attribute, для создания индекса с неповторяющимися значениями опцию следует выключить. Если на основе колонки

создается уникальный индекс, то при попытке вставить запись с неunikальным (повторяющимся) значением сервер выдаст ошибку и значение не будет вставлено. Иногда необходимо разрешить повторяющиеся значения, если ожидается, что индексированная колонка будет с большой вероятностью содержать повторяющуюся информацию. Неуникальный индекс генерируется также на основе внешнего ключа. На основе первичного и альтернативных ключей генерируются уникальные индексы. Имя сгенерированного индекса в дальнейшем при необходимости можно изменить вручную. По умолчанию ERwin автоматически сохраняет значения в порядке возрастания (значения сортируются по алфавиту от A до Z, а числа от 0 до 9).

4.5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как называются объекты в IDEF1 диаграммах?
2. Какие виды сущностей вы знаете?
3. В чем заключается отличие идентифицирующей связи от неидентифицирующей? Как они обозначаются на диаграммах?
4. Для чего используются CASE-средства проектирования?
5. Что представляет собой масштабирование в Toad Data Modeler Freeware?
6. Назовите уровни отображения модели в Toad Data Modeler Freeware.
7. Перечислите основные компоненты диаграммы Toad Data Modeler Freeware.
8. Как указать мощность и имя связи в Toad Data Modeler Freeware?
9. Для чего используются триггеры?
10. Где задаются правила ссылочной целостности в Toad Data Modeler Freeware?
11. Как выбирают установки для реализации ссылочной целостности?
12. Как реализована связь *: * в моделях Toad Data Modeler Freeware?
13. Какие ключи существуют в моделях Toad Data Modeler Freeware? В чем их отличительные особенности?
14. Как создать физическую модель в Toad Data Modeler Freeware?

4.6 УПРАЖНЕНИЯ

Задание: На основании концептуальной и реляционной моделей, описывающих зоопарк, построить в программе Toad Data Modeler Freeware логическую и физическую модели. Логическая модель представлена на рисунке 4.26, а физическая на рисунке 4.27.

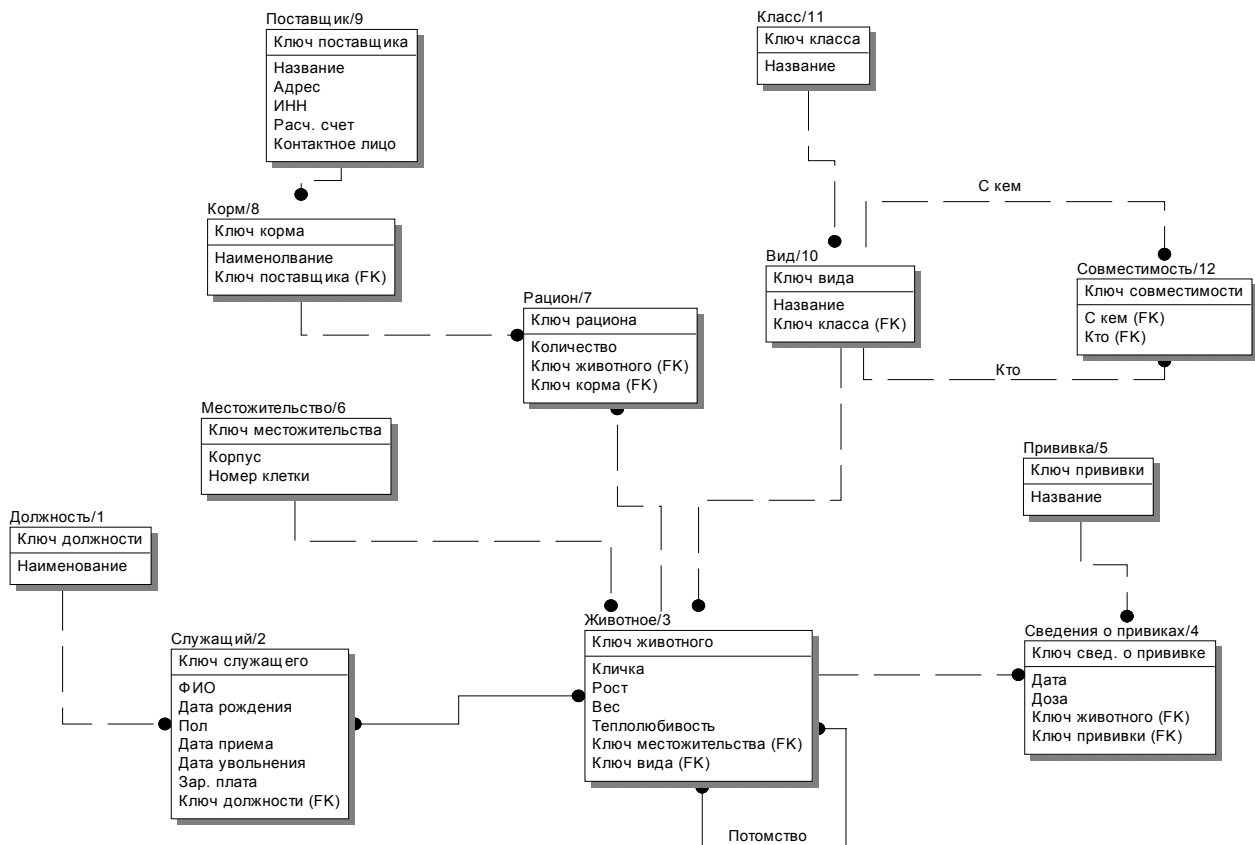


Рис. 4.26. Логическая модель, описывающая зоопарк

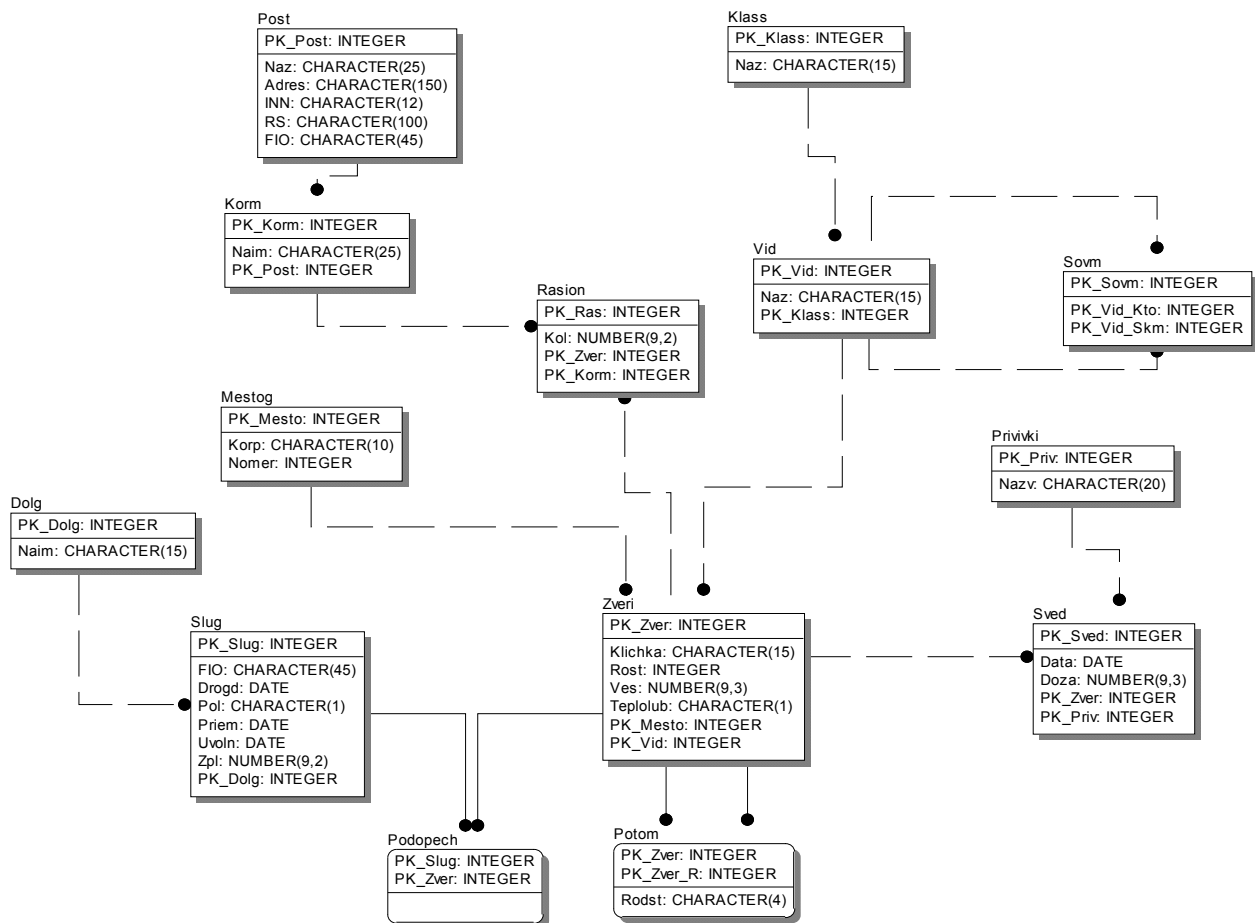


Рис. 4.27. Физическая модель, описывающая зоопарк

Задание: На основании концептуальной и реляционной моделей, построенных во втором и третьем модулях, построить в программе Toad Data Modeler Freeware логическую и физическую модели для СУБД Oracle.

4.7 ТЕСТЫ

1. Объекты модели IDEF1X называются:

- 1) Экземплярами
- 2) Атрибутами
- 3) Связями
- 4) Сущностями
- 5) Нет правильного ответа

2. Сущности в модели IDEF1X бывают:

- 1) Идентифицированными, неидентифицированными
- 2) Идентифицированными, зависимыми
- 3) Неидентифицированным, зависимыми
- 4) Зависимыми , независимыми
- 5) Нет правильного ответа

3. Связи в модели IDEF1X бывают:

- 1) Идентифицированными, неидентифицированными
- 2) Идентифицированными, зависимыми
- 3) Неидентифицированным, зависимыми
- 4) Зависимыми , независимыми
- 5) Нет правильного ответа

4. Неидентифицирующие связи в модели IDEF1X изображаются:

- 1) ромбом
- 2) прямоугольником
- 3) сплошной линией
- 4) пунктирной линией
- 5) нет правильного ответа

5. Идентифицирующие связи в модели IDEF1X изображаются:

- 1) ромбом
- 2) прямоугольником
- 3) сплошной линией
- 4) пунктирной линией
- 5) нет правильного ответа

6. Какие уровни модели бывают в программе Toad Data Modeler Freeware:

- 1) Концептуальный, Логический

- 2) Логический, Реляционный
- 3) Реляционный, Физический
- 4) Концептуальный, Физический
- 5) Логический, Физический

Ответы на тест:

1-4; 2-4; 3-1; 4-4; 5-3; 6-5.

5 СТРУКТУРИРОВАННЫЙ ЯЗЫК ЗАПРОСОВ

5.1 ОСНОВНЫЕ КОНСТРУКЦИИ SQL

SQL (Structured Query Language, язык структурированных запросов) - это специальный язык, используемый для определения данных, доступа к данным и их обработки. SQL относится к *непроцедурным (nonprocedural)* языкам - он лишь описывает нужные компоненты (например, таблицы) и желаемые результаты, не указывая, как именно эти результаты должны быть получены. Каждая реализация SQL является надстройкой над *процессором базы данных (database engine)*, который интерпретирует операторы SQL и определяет порядок обращения к структурам БД для корректного и эффективного формирования желаемого результата.

SQL является «подязыком данных», который предназначен только для использования в качестве языка взаимодействия с базой данных. Сам по себе SQL не содержит тех средств, которые необходимы для разработки законченных программ, и может использоваться в виде одной из трех прикладных реализаций:

1. **Интерактивный или автономный SQL** дает возможность пользователям непосредственно извлекать информацию из базы данных или записывать ее в базу.
2. **Статический SQL** – фиксированный (исполняемый), записанный заранее, а не генерируемый во время выполнения программы код SQL, который обычно используется в приложениях. Существуют две версии статического SQL. Встроенный SQL – это код SQL, включенный в код исходного текста программы, написанной на другом языке. Другое использование статического SQL – модульный язык. В этом случае модули SQL скомпонованы с модулями кода других языков.
3. **Динамический SQL** – код SQL, сгенерированный приложением во время исполнения. Он заменяет статический SQL в тех случаях, когда необходимый код SQL еще не может быть определен во время написания приложения, так как сам код зависит от того, какой выбор сделает пользователь.

SQL отличается от языков программирования высокого уровня несколькими признаками. Во-первых, он относится к непроцедурным языкам. На языке типа С можно записать для компьютера шаг за шагом все инструкции, необходимые для исполнения задания. SQL просто декларирует, что нужно делать, а исполнение возлагает на СУБД. Такой подход лежит в русле философии реляционных баз данных. СУБД в данном случае рассматривается как «черный ящик»: что делается внутри него – пользователя не касается. Его интересует только получение правильного ответа из базы данных и внесение в нее необходимых изменений. Другим отличием SQL является трехзначная логика..

Данные содержатся в таблицах, таблицы сгруппированы в схемы, а схемы – в каталоги. Каталоги могут быть в дальнейшем сгруппированы в кластеры. В некоторых приложениях баз данных эти термины несколько отличаются от определений стандарта.

С точки зрения конкретного сеанса SQL кластер содержит все таблицы, к которым имеется доступ в данном сеансе.

Схемой называется именованный набор объектов базы данных, управляемых одним пользователем и в определенных случаях рассматриваемых как единое целое.

Рекомендуется использовать домены, позволяющие построить более точную классификацию данных по типам, чем та, которая достигается с помощью стандартного набора типов данных. Например, телефонные номера относятся не к тому типу данных, к которому принадлежат номера социальных страховок, даже если и те и другие выражаются числами, в то же время, информацию одного и того же типа данных иногда не имеет смысла сравнивать, так как она может принадлежать разным доменам.

Определение домена содержит тип данных, но может, кроме того, включать предложения, которые определяют значения по умолчанию, ограничения и последовательность сортировки для упорядочения наборов символов для домена (под ограничениями понимаются правила, ограничивающие значения данных, которые разрешено размещать в определенном столбце).

Язык SQL состоит из двух специальных наборов команд. DDL (Data Definition Language, язык определения данных) - это подмножество SQL, используемое для определения и модификации различных структур данных, а DML (Data Manipulation Language, язык манипулирования данными) - это подмножество SQL, применяемое для получения и обработки данных, хранящихся в структурах, определенных ранее с помощью DDL. DDL состоит из большого количества команд, необходимых для создания таблиц, индексов, представлений и ограничений, а в DML входит всего четыре оператора:

INSERT

Добавляет данные в базу данных.

UPDATE

Изменяет данные в базе данных.

DELETE

Удаляет данные из базы данных.

SELECT

Извлекает данные из базы данных.

Некоторым кажется, что применение DDL является прерогативой администраторов базы данных, а операторы DML должны писать разработчики, но эти два языка не так-то просто разделить. Сложно организовать эффективный доступ к данным и их обработку, не понимая, какие структуры доступны и как они связаны. Также сложно проектировать соответствующие структуры, не зная, как они будут обрабатываться. Сказав это, сосредоточимся на DML. DDL в книге будет встречаться лишь тогда, когда это необходимо для иллюстрации применения DML. Причины особого внимания к DML таковы:

- DDL хорошо описан во многих книгах по проектированию и администрированию баз данных, а также в справочниках по SQL.
- Проблемы производительности обычно бывают вызваны неэффективными операторами DML.
- Хотя операторов всего четыре, DML - большая тема.

Эффективное хранение и извлечение информации сейчас важно как никогда ранее:

- **Все** больше компаний предлагают свои услуги через Интернет. В часы пик они вынуждены обслуживать тысячи параллельных запросов, и задержки означают прямую потерю прибыли. Для таких систем каждый оператор SQL должен быть тщательно продуман, чтобы обеспечивать требуемую производительность при увеличении объема данных.
- Сегодня есть возможность хранить гораздо больше данных, чем пять лет назад. Один дисковый массив вмещает десятки терабайт данных, и уже не за горами хранение сотен терабайт. Программное обеспечение, применяемое для загрузки и анализа данных в этих средах, должно использовать весь потенциал SQL, чтобы обрабатывать неизменно увеличивающийся объем данных за постоянные (или сокращающиеся) промежутки времени.

Оператор SELECT

Оператор SELECT используется для извлечения данных из базы. Множество данных, извлекаемое оператором SELECT, называется *результатирующим множеством (result set)*. Как и таблица, результирующее множество состоит из строк и столбцов, что позволяет заполнить таблицу данными результирующего множества. Общий вид оператора SELECT таков:

SELECT <один или несколько объектов>

FROM <одно или несколько мест>

WHERE <ни одного, одно или несколько условий>

Инструкции SELECT и FROM необходимы, а вот инструкция WHERE необязательна (хотя и она почти всегда используется). Начнем с простого примера, извлекающего три столбца из каждой строки таблицы CUSTOMER (заказчики):

SELECT cust_nbr, name, region.id FROM customer;

Инструкция WHERE не была использована, и никаких ограничений на данные мы не наложили, поэтому запрос возвращает все строки таблицы заказчиков. Если необходимо ограничить возвращаемый набор данных, можно добавить в оператор инструкцию WHERE с одним условием:

**SELECT cust_nbr, name, region_id FROM customer
WHERE region_id = 8;**

Теперь результирующее множество содержит только заказчиков, проживающих в области с идентификатором region_id, равным 8. Но что если нужно сослаться на область по имени, а не по номеру? Можно выбрать нужное имя из таблицы REGION, а затем, зная region_id, обратиться к таблице CUSTOMER. Чтобы не писать два разных запроса, можно получить тот же результат с помощью одного запроса, использующего *объединение (join)*:

**SELECT customer.cust_nbr, customer.name, region.name FROM
customer, region WHERE region.name = 'New England' AND
region.region_id = customer.region_id;**

Теперь в инструкции FROM не одна таблица, а две, и инструкция WHERE содержит *условие объединения (join condition)*, которое указывает, что таблицы заказчиков и областей должны быть объединены по столбцу region_id, имеющемуся в каждой таблице.

Так как обе таблицы содержат столбец с названием *name*, необходимо как-то определить, какой именно столбец вас интересует. В предыдущем примере это делается с помощью точечной нотации - добавления через точку имени таблицы перед именем столбца. Если же на написание полных имен таблиц у вас уходит слишком много времени, назначьте для каждого названия таблицы в инструкции FROM *псевдоним (alias)* и используйте его вместо имени в инструкциях SELECT и WHERE:

**SELECT c.cust_nbr, c.name, r.name FROM
customer c, region r WHERE r.name = 'New
England' AND r.region_id = c.region_id;**

В этом примере псевдоним «с» был присвоен таблице заказчиков, а псевдоним «г» - таблице областей. Теперь можно в инструкциях SELECT и WHERE писать «с» вместо «customer» и «г» вместо «region».

Элементы инструкции SELECT

Рассмотренные ранее результирующие множества, порожденные нашими запросами, содержали столбцы одной или нескольких таблиц. Как правило, элементами инструкции SELECT действительно являются ссылки на столбцы; среди них также могут встречаться:

- Константы, такие как числа (1) или строки ('abc')
- Выражения, например shape, diameter * 3.1415927
- Функции, такие как TO_DATE('01-JAN-2002', 'DD-MON-YYYY')
- Псевдостолбцы, например ROWID, ROWNUM или LEVEL

Если первые три пункта в списке довольно просты, то последний нуждается в пояснении. В Oracle есть несколько столбцов-призраков, называемых *псевдостолбцами (pseudocolumns)*, которые не присутствуют ни в одной таблице. Их значения появляются во время выполнения запроса, и в некоторых ситуациях они бывают полезны.

Упорядочение результатов

В общем случае нет гарантии, что результирующее множество будет сформировано в каком-либо определенном порядке. Если нужно отсортировать результаты по одному или нескольким столбцам, следует добавить инструкцию ORDER BY сразу же после WHERE. Следующий пример сортирует заказчиков из Новой Англии по фамилиям:

```
SELECT c.cust_nbr, c.name, r.name FROM customer c,  
region r WHERE r.name = 'New England'  
AND r.region_id = c.region_id ORDER BY c.name;
```

Сортируемые столбцы можно определять по их положению в инструкции SELECT. Отсортируем предыдущий запрос по столбцу CUST_NBR (номер заказчика), который в инструкции SELECT указан первым:

```
SELECT c.cust_nbr, c.name, r.name FROM customer c,  
region r WHERE r.name = 'New England' AND  
r.region_id = c.region_id ORDER BY 1;
```

Указание ключей сортировки по позиции экономит вам немного времени, но может привести к ошибкам, если в дальнейшем порядок следования столбцов в инструкции SELECT будет изменен.

Удаление дубликатов

В некоторых случаях результирующее множество может содержать одинаковые данные. Например, при формировании списка проданных за последний месяц деталей те детали, которые присутствовали в нескольких заказах, встретятся в результирующем множестве несколько раз. Чтобы исключить повторы, вставьте в инструкцию SELECT ключевое слово DISTINCT:

```
SELECT DISTINCT li.part_nbr  
FROM cust_order co, line_item li  
WHERE co.order_dt >= TO_DATE('01-JUL-2001','DD-MON-YYYY')  
AND co.order_dt < TO_DATE('01-AUG-2001','DD-MON-YYYY')  
AND co.order_nbr = li.order_nbr;
```

Запрос возвращает множество отличающихся друг от друга записей о деталях, заказанных в течение июля 2001 года. Без ключевого слова DISTINCT вывод содержал бы по одной строке для каждой строки каждого заказа, и одна деталь могла бы встречаться несколько раз, если бы она содержалась в нескольких заказах. Принимая решение о включении DISTINCT в инструкцию SELECT, следует иметь в виду, что поиск и удаление дубликатов требует сортировки, которая будет служить дополнительной нагрузкой при выполнении запроса.

Оператор INSERT

Оператор INSERT - это механизм загрузки данных в базу данных. За один раз данные можно вставить только в одну таблицу, зато брать вставляемые данные можно из нескольких дополнительных таблиц. Вставляя данные в таблицу, не нужно указывать значения для каждого столбца, однако следует обратить внимание на то, допускают ли столбцы использование значений NULL¹ или же нет.

В операторе INSERT необходимо указать значения как минимум для тех столбцов, которые содержат пометку NOT NULL. Например, как показано ниже:

```
INSERT INTO employee (emp_id, lname, dept_id)  
VALUES (101, 'Smith', 2);
```

Количество элементов в инструкции VALUES должно совпадать с количеством элементов в списке столбцов, а их типы данных должны соответствовать определениям

столбцов. В нашем примере emp_id и dept_id хранят численные значения, а lname - строковое, поэтому оператор INSERT выполнится без ошибок. Oracle всегда автоматически пытается преобразовать один тип данных в другой, поэтому следующий оператор тоже выполнится без ошибок:

```
INSERT INTO employee (emp_id, lname, dept_id)
VALUES ('101', 'Smith', '2');
```

Иногда вставляемые данные нужно предварительно извлечь из одной или нескольких таблиц. Так как оператор SELECT формирует результирующее множество, состоящее из строк и столбцов, то можно непосредственно передать его оператору INSERT:

```
INSERT INTO employee (emp_id, fname, lname, dept_id, hire_date)
SELECT 101, 'Dave', 'Smith', d.dept_id, SYSDATE
FROM department d WHERE d.name = 'Accounting';
```

В данном примере оператор SELECT извлекает идентификатор отдела для бухгалтерии (Accounting). Остальные четыре столбца в операторе SELECT представлены константами.

Оператор DELETE

Оператор DELETE обеспечивает удаление данных из базы. Как и SELECT, оператор DELETE содержит инструкцию WHERE с условиями для идентификации удаляемых строк. Забыв указать инструкцию WHERE в операторе DELETE, вы удалите все строки из указанной таблицы. Следующий оператор удаляет всех сотрудников с фамилией Hooper из таблицы EMPLOYEE:

```
DELETE FROM employee WHERE lname = 'Hooper';
```

Иногда значения, необходимые для построения условия в инструкции WHERE, располагаются в других таблицах. Например, решение компании вынести вовне функции бухучета потребует удаления всего бухгалтерского персонала из таблицы EMPLOYEE:

```
DELETE FROM employee WHERE dept_id = (SELECT dept_id FROM department WHERE
name = 'Accounting');
```

Подобное использование оператора SELECT носит название *подзапроса (subquery)*.

Оператор UPDATE

С помощью оператора UPDATE вносятся изменения в существующие данные. Как и DELETE, оператор UPDATE включает в себя инструкцию WHERE для указания тех строк, которые будут изменены. Посмотрим, как можно предоставить 10-процентное повышение зарплаты тем, у кого годовой доход меньше 40 000 долларов:

```
UPDATE employee
SET salary = salary * 1.1
WHERE salary < 40000;
```

Если необходимо изменить несколько столбцов, вы можете выбрать один из двух вариантов: задать набор пар столбец-значение, разделенных запятыми, или указать набор столбцов и подзапрос. Два следующих оператора UPDATE изменяют столбцы inactive_dt и inactive_ind в таблице CUSTOMER для клиентов, не сделавших ни одного заказа за последний год:

```
UPDATE customer
SET inactive_dt = SYSDATE, inactive_ind = 'Y'
```

```
WHERE last_order_dt < SYSDATE - 365;
```

```
UPDATE customer SET (inactive_dt, inactive_ind) =  
(SELECT SYSDATE, Y FROM dual) WHERE last_order_dt < SYSDATE - 365;
```

Подзапрос во втором примере выглядит немного неестественно, так как он обращается к таблице dual для построения результирующего множества, состоящего из двух констант; он приведен для иллюстрации использования подзапросов в операторе UPDATE.

CREATE TABLE

Создает таблицу.

Пример простейшей команды по созданию таблицы.

```
CREATE TABLE dept  
(deptno NUMBER (2) PRIMARY KEY,  
  dname VARCHAR2(10),  
  loc VARCHAR2(9) )
```

CREATE SEQUENCE

Создает sequence. Sequence - это объект базы данных необходимый для того, чтобы несколько пользователей могли генерировать уникальное целое значение. Обычно sequences используется для автоматической генерации значения первичного ключа.

Когда sequence генерирует число, его значение увеличивается. Если два пользователя пытаются одновременно получить значение одного и того же sequence, то сначала генерируется значение для первого одного пользователя, а затем для другого. Пользователь не может получить значение сгенерированное для другого пользователя

Когда sequence создан, вы можете получить доступ к его значениям в SQL - выражениях с помощью псевдоколонок CURRVAL (возвращает текущее значение sequence) или NEXTVAL (увеличивает значение sequence и возвращает это новое значение).

```
CREATE SEQUENCE eseq INCREMENT BY 10;
```

При первом обращении к ESEQ.NEXTVAL возвратит 1. При втором возвратит 11. И т.д.

```
CREATE SEQUENCE ADM.GURSEQ INCREMENT BY 1 START WITH 10 CYCLE;
```

Объединения

Часто бывает необходима информация из нескольких таблиц. Конструкция языка SQL, комбинирующая данные двух и более таблиц, называется *объединением (join)*. В данной главе будут рассмотрены объединения, их типы и способы использования.

Объединение - это SQL-запрос, который извлекает информацию из двух или более таблиц или представлений. При указании в инструкции FROM нескольких таблиц или представлений Oracle выполняет объединение, связывая вместе строки различных таблиц. Существует несколько типов объединений:

Внутренние объединения (inner joins)

Внутренние объединения - это стандартный вариант объединения, который возвращает строки, удовлетворяющие условию объединения. Каждая строка, возвращенная внутренним объединением, содержит данные всех таблиц, включенных в объединение.

Внешние объединения (outer join)

Внешние объединения - это расширение внутренних. Внешнее объединение возвращает строки, удовлетворяющие условию объединения, а также те строки одной

таблицы, для которых не найдено строк другой таблицы, отвечающих условию объединения.

Внутренние объединения

Внутреннее объединение возвращает строки, удовлетворяющие условию объединения. Давайте рассмотрим понятие «объединение» на примере. Пусть необходимо вывести фамилию и название подразделения для каждого сотрудника. Используем следующий оператор SQL:

```
SELECT E.LNAME, D.NAME  
FROM EMPLOYEE E, DEPARTMENT D  
WHERE E.DEPT_ID = D.DEPT_ID;
```

В этом примере запрос обращается к двум таблицам, так как фамилия служащего хранится в таблице EMPLOYEE, а название подразделения - в таблице DEPARTMENT. Обратите внимание на то, что в инструкции FROM названия двух таблиц, EMPLOYEE и DEPARTMENT, перечислены через запятую. Если нужно объединить три и более таблиц, укажите все таблицы в инструкции FROM, перечислив их через запятую. В списке оператора SELECT могут упоминаться столбцы из любой таблицы, указанной в инструкции FROM.

Условие объединения

Обычно при выполнении объединения в инструкцию WHERE включается условие, которое устанавливает соответствие таблиц, указанных в инструкции FROM. Такое условие называется условием объединения. Условие объединения определяет, как следует объединять строки одной таблицы со строками другой. Как правило, условие объединения применяется к столбцам, которые являются внешними ключами таблиц.

В первом примере предыдущего раздела в инструкции WHERE было задано условие объединения, в котором указывалось равенство столбцов DEPT_ID таблицы EMPLOYEE и таблицы DEPARTMENT:

```
WHERE E.DEPT_ID = D.DEPT_ID
```

Для выполнения объединения Oracle берет одну комбинацию строк из двух таблиц и проверяет истинность условия объединения. Если условие объединения истинно, Oracle включает данную комбинацию строк в результирующее множество. Процесс повторяется для всех сочетаний строк двух таблиц. Приведем несколько важных фактов, касающихся условия объединения.

- Нет необходимости включать столбцы, входящие в условие объединения, в список SELECT. В следующем примере условие объединения содержит столбец DEPT_ID таблицы EMPLOYEE и таблицы DEPARTMENT, но при этом столбец DEPT_ID не участвует в выборке:

```
SELECT E.LNAME, D.NAME  
FROM EMPLOYEE E, DEPARTMENT D  
WHERE E.DEPT_ID = D.DEPT_ID;
```

- Обычно условие объединения указывается для столбцов, являющихся внешним ключом одной таблицы и первичным или уникальным ключом другой таблицы. Однако можно использовать и другие столбцы. Каждое условие объединения затрагивает столбцы, которые устанавливают связь между двумя таблицами.

- Условие объединения может включать в себя несколько столбцов. Так обычно бывает, если внешний ключ состоит из нескольких столбцов.

- Общее количество условий объединения всегда на единицу меньше общего количества таблиц.

- Условия объединения должны содержать столбцы с совместимыми типами данных. Обратите внимание на то, что типы данных должны быть *совместимыми*, но не обязаны *совпадать*. При необходимости Oracle выполняет автоматическое преобразование типа.

- Оператор равенства (=) не обязательно должен входить в условие объединения. Возможно использование других операторов. В объединениях могут участвовать операторы, о которых будет рассказано далее в этом разделе.

Внешние объединения

При объединении двух таблиц может возникнуть необходимость вывести все строки одной из таблиц, даже если для них не существует соответствующих строк во второй таблице. Рассмотрим две таблицы: поставщиков (SUPPLIER) и деталей (PART):

SELECT * FROM SUPPLIER;

SUPPLIER ID NAME

101 Pacific Disks, Inc.
102 Silicon Valley Microchips
103 Blue River Electronics

SELECT * FROM PART;

PART_NBR	NAME	SUPPLIER_ID	STATUS	INVENTORY_QTY	UNIT_COST	RESUPPLY_DATE
HD211	20 GB Hard Disk	101	ACTIVE	5	2000	12-DEC-00
P3000	3000 MHz Processor	102	ACTIVE	12	600	03-NOV-00

Если нужно вывести всех поставщиков и поставляемые ими детали, естественно использовать следующий запрос:

SELECT S.SUPPLIER_ID, S. NAME SUPPLIER.NAME, P.PARTNBR, P.NAME PART_NAME

FROM SUPPLIER S, PART P

WHERE S.SUPPLIER_ID = P.SUPPLIER_ID;

SUPPLIER_ID	SUPPLIER_NAME	PART_NBR	PART_NAME
101	Pacific Disks, Inc.	HD211	20 GB Hard Disk
102	Silicon Valley Microchips	P3000	3000 MHz Processor

Обратите внимание на то, что, хотя поставщиков трое, запрос выводит только двоих, потому что третий поставщик (Blue River Electronics) в данный момент ничего не поставляет. Когда Oracle выполняет объединение между таблицами SUPPLIER и PART, сопоставляются столбцы SUPPLIER_ID этих двух таблиц (как указано в условии объединения). Так как для SUPPLIER_ID = 103 не существует соответствующих записей в таблице PART, этот поставщик не включается в результирующее множество. Такой тип объединения является наиболее естественным и называется *внутренним объединением*.

Понятие внутреннего объединения легче пояснить в терминах декартова произведения. При выполнении объединения таблиц SUPPLIER и PART сначала формируется декартово произведение (физически оно не материализуется), а затем условия инструкции WHERE ограничивают результат только теми строками, в которых совпадают значения SUPPLIER_ID.

Но хотелось бы получить полный список поставщиков, включающий и тех, кто в данный момент ничего не поставляет. Oracle предоставляет специальный тип объединения, который позволяет включать в результирующее множество строки одной таблицы, для которых не найдены соответствующие строки в другой таблице. Такое объединение называется *внешним (outer)*. Внешнее объединение позволит вывести строки для всех поставщиков, а если поставщик в настоящий момент поставляет какие-то детали, то и соответствующие строки деталей. Если в настоящий момент поставщик не поставляет детали, в результирующем множестве в столбцах таблицы PART будут возвращены значения NULL.

Синтаксис внешнего объединения несколько отличается от синтаксиса внутреннего объединения. Применяется специальный оператор, называемый *оператором внешнего объединения*, который выглядит как знак «плюс», заключенный в круглые скобки, то есть (+). Этот оператор используется в условии объединения инструкции WHERE вслед за именем поля той таблицы, которую вы хотите рассматривать как необязательную. В рассматриваемом примере про детали и поставщиков таблица PART не содержит информацию об одном поставщике. Просто добавляем оператор (+) к условию объединения со стороны таблицы PART. Запрос и результирующее множество будут выглядеть следующим образом:

```
SELECT S.SUPPLIER_ID, S.NAME SUPPLIER_NAME, P.PART_NBR, P.NAME
PART_NAME
FROM SUPPLIER S, PART P
WHERE S.SUPPLIER_ID = P.SUPPLIER_ID (+);
```

SUPPLIER_ID	SUPPLIER_NAME	PART_NBR	PART_NAME
101	Pacific Disks, Inc.	HD211	20 GB Hard Disk
102	Silicon Valley Microchips	P3000	3000 MHz Processor
103	Blue River Electronics		

Заметьте, что оператор (+) следует за P.SUPPLIER_ID, что делает таблицу PART необязательной (в данном объединении). Если поставщик ничего не поставляет в настоящий момент, Oracle создаст для данного поставщика запись в таблице PART со значениями NULL во всех ячейках. Результирующее множество теперь содержит всех поставщиков независимо от состояния их текущих поставок. Как видите, столбцы PART для поставщика с идентификатором 103 содержат NULL.

Оператор внешнего объединения (+) может появляться как в левой, так и в правой части условия объединения. Вы только должны быть уверены в том, что применяете оператор к соответствующей таблице (в контексте данного запроса). Например, если поменять местами части оператора равенства из предыдущего примера, это никак не повлияет на результат:

```
SELECT S.SUPPLIER_ID, S.NAME SUPPLIER.NAME, P.PART_NBR, P.NAME
PART.NAME
FROM SUPPLIER S, PART P
WHERE P.SUPPLIER_ID (+) = S.SUPPLIER_ID;
```

SUPPLIER_ID	SUPPLIERNAME	PART_NBR	PART_NAME
101	Pacific Disks, Inc.	HD211	20 GB Hard Disk
102	Silicon Valley Microchips	P3000	3000 MHz Processor
103	Blue River Electronics		

Ограничения, налагаемые на внешние объединения

Существует ряд правил и ограничений, относящихся к использованию внешних объединений в запросах. Если в запросе выполняется внешнее объединение, Oracle не разрешает использовать в этом же запросе некоторые другие операции. Далее мы поговорим о таких ограничениях и о некоторых способах их обхода.

- Оператор внешнего объединения может присутствовать только в одной части условия объединения. При попытке использовать его в обеих частях возникает ошибка ORA-1468. Например:

```
SELECT S.SUPPLIER.ID, S.NAME SUPPLIER.NAME, P.PART.NBR, P.NAME PART.
NAME
FROM SUPPLIER S, PART P
WHERE S.SUPPLIER.ID (+) = P.SUPPLIER.ID (+);
WHERE S.SUPPLIER.ID (+) = P.SUPPLIER_ID (+)
```

*

ERROR at line 3:

ORA-01468: a predicate may reference only one outer-joined table

• Если в объединении участвует более двух таблиц, то каждая из таблиц в запросе не может участвовать во внешнем объединении с более чем одной другой таблицей.

• В условии внешнего объединения, содержащем оператор (+), запрещено использование оператора IN. Например:

```
SELECT E.LNAME, J.FUNCTION
FROM EMPLOYEE E, JOB J
WHERE E.JOB.ID (+) IN (66B, 670, 667);
WHERE E.JOB.ID (+) IN (668, 670, 667)
```

ERROR at line 3:

ORA-01719: outer join operator (+) not allowed in operand of OR or IN

• Условие внешнего объединения, содержащее оператор (+), нельзя комбинировать с другими условиями при помощи оператора OR. Например:

```
SELECT E.LNAME, D.NAME
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.DEPT.ID = D.DEPT.ID (+) OR D.DEPT.ID =10;
WHERE E.DEPT_ID = D.DEPT_ID (+)
```

ERROR at line 3:

ORA-01719: outer join operator (+) not allowed in operand of OR or IN

• Условие внешнего объединения, содержащее оператор (+), не может содержать подзапрос. Например:

```
SELECT E.LNAME
FROM EMPLOYEE E
WHERE E.DEPT_ID (+) =
(SELECT DEPT_ID FROM DEPARTMENT WHERE NAME =
'ACCOUNTING');
(SELECT DEPT_ID FROM DEPARTMENT WHERE NAME = 'ACCOUNTING')
```

ERROR at line 4:

ORA-01799: a column may not be outer-joined to a subquery

Чтобы достичь желаемого эффекта и избежать ошибки, можно использовать встроеное представление:

```
SELECT E.LNAME
FROM EMPLOYEE E,
(SELECT DEPT_ID FROM DEPARTMENT WHERE NAME = 'ACCOUNTING')
V
WHERE E.DEPTJD (+) = V.DEPT_ID;
```

Групповые операции

В повседневной работе SQL-программист часто имеет дело с групповыми операциями. Используя SQL для доступа к базе данных, часто задаются вопросы, подобные перечисленным ниже:

- Какова максимальная заработная плата в данном подразделении?
- Сколько в каждом подразделении менеджеров?
- Сколько заказчиков существует для каждого продукта?
- Можно ли вывести среднемесячное значение продаж для каждого региона?

Для ответа на такие вопросы необходимы групповые операции. Oracle предоставляет широкий спектр возможностей по обработке групповых операций, в том числе обобщающие функции, инструкции GROUP BY и HAVING, функцию GROUPING и расширения инструкции GROUP BY: ROLLUP и CUBE.

Обобщающие функции

Если говорить по существу, *обобщающая функция (aggregate function)* суммирует результаты выражения для некоторого количества строк, возвращая одно значение. Синтаксис большинства обобщающих функций таков:

обобщающая_функция([DISTINCT | ALL] *выражение*)

Приведем перечень элементов конструкции:

обобщающая функция

Указывает имя функции, например SUM, COUNT, AVG, MAX, MIN и др.

DISTINCT

Указывает, что обобщающая функция должна учитывать только неповторяющиеся значения *выражения*.

ALL

Указывает, что обобщающая функция должна учитывать все значения *выражения*, в том числе и все дублирующиеся. По умолчанию считается, что использовано ALL.

выражение

Указывает столбец или любое другое выражение, по которому необходимо выполнить обобщение.

Давайте рассмотрим простой пример. Для нахождения максимальной зарплаты сотрудников SQL-оператор использует функцию MAX:

```
SELECT MAX(SALARY) FROM EMPLOYEE;  
MAX(SALARY)  
5000
```

Инструкция GROUP BY

Инструкция GROUP BY, используемая совместно с обобщающими функциями, разбивает результирующее множество на несколько групп, а затем для каждой группы выдается одна строка сводной информации. Например, если нужно вычислить общее количество заказов каждого клиента, выполним следующий запрос:

```
SELECT CUST.NBR, COUNT(ORDER_NBR)  
FROM CUST.ORDER  
GROUP BY CUST.NBR;
```

Запрос выдает одну сводную строку для каждого клиента. В этом заключается суть запроса GROUP BY. Мы просим Oracle сгруппировать (GROUP) результаты по номеру клиента (BY CUST_NBR), поэтому для каждого уникального значения CUST_NBR порождается одна строка вывода. Каждое значение для определенного клиента представляет собой сводную информацию по всем строкам данного клиента.

Необобщенное выражение CUST_NBR из списка SELECT присутствует и в инструкции GROUP BY. Если в списке SELECT присутствует смесь обобщенных и необобщенных значений, SQL считает, что вы собираетесь выполнить операцию GROUP BY, поэтому все необобщенные выражения должны быть указаны и в инструкции GROUP BY. Если этого не сделать, SQL выдаст сообщение об ошибке.

Аналогично, если не включить все необобщенные выражения списка SELECT в инструкцию GROUP BY, то SQL выдаст такую ошибку:

```
SELECT CUST_NBR, SALES_EMP_ID, COUNT(ORDER_NBR)  
FROM CUST_ORDER  
GROUP BY CUST_NBR;  
SELECT CUST_NBR, SALES_EMP_ID, COUNT(ORDER_NBR)  
ERROR at line 1:  
ORA-00979: not a GROUP BY expression
```

Наконец, не разрешено использование групповой (обобщающей) функции в инструкции GROUP BY. При попытке такого использования, как в приведенном ниже примере, вы получите следующее сообщение об ошибке:

```
SELECT CUST_NBR, COUNT(ORDER_NBR)
```

```

FROM CUST_ORDER
GROUP BY CUST_NBR, COUNT(ORDER_NBR);
GROUP BY CUST_NBR, COUNT(ORDER_NBR)
*
```

ERROR at line 3:

ORA-00934: group function is not allowed here

Инструкция HAVING

Инструкция HAVING тесно связана с инструкцией GROUP BY. Инструкция HAVING используется для наложения фильтра на группы, созданные инструкцией GROUP BY. Если запрос содержит инструкцию HAVING и инструкцию GROUP BY, результирующее множество будет содержать только те группы, которые удовлетворяют условию, указанному в инструкции HAVING. Давайте рассмотрим несколько примеров, иллюстрирующих вышесказанное. Приведенный ниже запрос возвращает количество заказов каждого клиента:

```

SELECT CUST_NBR, COUNT(ORDER_NBR)
FROM CUST_ORDER
GROUP BY CUST_NBR
HAVING CUST_NBR < 260;
```

CUST_NBR	COUNT(ORDER_NBR)
201	2
231	6
244	2
255	6

Заметьте, что в выводе присутствуют только клиенты с номерами, меньшими, чем 260. Это объясняется тем, что в инструкции HAVING указано условие CUST_NBR < 260. Количество заказов подсчитывается для всех клиентов, но выводятся только те группы, для которых выполнено условие инструкции HAVING.

Этот пример является не очень удачной иллюстрацией возможностей инструкции HAVING; в данном случае она просто указывает данные, которые не должны включаться в результирующее множество. Было бы эффективнее использовать WHERE CUST.NBR < 260, а не HAVING CUST.NBR < 260, так как **инструкция WHERE исключает строки из рассмотрения до проведения группировки, а HAVING устраняет уже созданные группы**. Правильнее было бы записать предыдущий запрос так:

```

SELECT CUST_NBR, COUNT(ORDER_NBR)
FROM CUST_ORDER
WHERE CUST_NBR < 260;
```

Следующий пример демонстрирует более удачное применение инструкции HAVING:

```

SELECT CUST_NBR, COUNT(ORDER_NBR)
FROM CUST_ORDER
GROUP BY CUST_NBR
HAVING COUNT(ORDER_NBR) > 2;
```

Обратите внимание на использование в инструкции HAVING обобщающей функции. Здесь инструкция HAVING применена надлежащим образом, так как результат выполнения обобщающей функции доступен только после проведения группировки.

Синтаксис инструкции HAVING подобен синтаксису инструкции WHERE. Но для условия инструкции HAVING существует одно ограничение. Это условие может относиться только к выражениям списка SELECT или инструкции GROUP BY. Если

указать в инструкции HAVING выражение, не содержащееся ни в списке SELECT, ни в инструкции GROUP BY, то в ответ будет выдано сообщение об ошибке. Например:

Обработка дат и времени

Разработчики баз данных постоянно имеют дело с данными, относящимися к датам и времени. Потребность в эффективной обработке значений дат и времени становится критичной на рубеже веков, когда приходится изобретать способы корректного управления двузначными значениями годов, когда они переходят от 99 к 00, а затем к 01. В эпоху глобальной электронной коммерции понятие времени актуально как никогда ранее: торговля происходит двадцать четыре часа в сутки во всех временных зонах.

База данных должна эффективно и рационально организовывать хранение, извлечение и манипулирование следующих типов данных:

- Дата
- Время
- Интервалы дат и времени
- Часовые пояса

Обработка дат и времени в Oracle продуманна и эффективна. Oracle9i обеспечивает удобную работу с датами и временем. В Oracle9i вводится новый ряд возможностей, включая поддержку долей секунды, интервалов дат и времени и часовых поясов.

Вставка дат в БД и извлечение дат из БД

В реальном мире даты не всегда представляются в формате типа данных DATE Oracle. Постоянно будет возникать необходимость преобразования значений типа DATE в другие типы данных и наоборот. Это особенно важно при сопряжении базы данных Oracle с внешней системой, например, если данные получаются из внешней системы, в которой даты представлены символьными строками (или даже числами), или при отправке данных из базы Oracle в другие приложения, которые не поддерживают тип DATE. Также нужно преобразовывать значения DATE при отображении дат на экране или создании отчета.

Oracle предоставляет две чрезвычайно полезные функции преобразования дат:

- TO_DATE
- TO_CHAR

Как следует из их названий, функция TO_DATE используется для преобразования символьных или числовых данных в значение типа DATE, а функция TO_CHAR выполняет преобразование значения DATE в строку символов. Обсуждаемые далее в этом разделе форматы даты хорошо приспособлены для таких преобразований.

TO_DATE

TO_DATE - это встроенная функция SQL, конвертирующая символьную строку в дату. На вход функции TO_DATE может подаваться символьная строка, переменная PL/SQL или столбец базы данных типа CHAR или VARCHAR2.

Вызов TO_DATE выглядит следующим образом:

TO_DATE(строка [, формат])

Перечислим элементы конструкции:

строка

Символьная строка, переменная PL/SQL или столбец базы данных, содержащий символьные (или числовые) данные, преобразуемые в дату.

формат

Задаёт формат преобразуемой строки. Формат должен представлять собой допустимую комбинацию элементов формата.

Указание формата даты является необязательным. Если не задавать формат, то будет считаться, что строка имеет формат по умолчанию (определяемый параметром NLS_DATE_FORMAT).

С помощью функции TO_DATE можно преобразовать число в формат DATE. Когда вы подаете на вход функции TO_DATE число, Oracle неявно преобразует введенное число в строку, затем эта строка передается функции TO_DATE.

Использование для даты формата по умолчанию

Каждая база данных Oracle имеет формат даты по умолчанию. Если администратор базы данных не определил ничего иного, то этот формат таков:

DD-MON-YY

В Oracle Database 10g Express Edition по умолчанию установлен следующий формат даты: **DD.MM.RR.**

При вызове функции TO_DATE без явного указания формата даты Oracle считает, что строка ввода имеет формат даты по умолчанию. Следующий оператор INSERT преобразует строку в формате по умолчанию в значение типа DATE и вставляет его в таблицу EMPLOYEE:

```
INSERT INTO EMPLOYEE
(EMP_ID, FNAME, LNAME, DEPT_ID, MANAGER_EMP_ID, SALARY,
HIRE.DATE)
VALUES
(2304, -John', -Smith-, 20, 1258, 20000, TO_DATE('22-OCT-99'));
1 row created.
SELECT * FROM EMPLOYEE;
```

EMP_ID	FNAME	LNAME	DEPT_ID	MANAGER_EMP_ID	SALARY	HIRE_DATE
2304	John	Smith	20	1258	20000	22-OCT-99

Обратите внимание, что столбец HIRE_DATE имеет тип DATE, и символьная строка '22-OCT-99' была преобразована в дату функцией TO_DATE. В данном случае указание формата не требуется, так как вставляемая строка имеет формат даты по умолчанию. В действительности, если предлагаемая строка имеет формат даты по умолчанию, не нужна и сама функция TO_DATE. Oracle автоматически выполняет неявное преобразование типов, как в приведенном ниже примере:

```
INSERT INTO EMPLOYEE
(EMP.ID, FNAME, LNAME, DEPT_ID, MANAGER_EMP_ID, SALARY,
HIRE.DATE)
VALUES
(2304, 'John', •Smith', 20, 1258, 20000, '22-OCT-99');
1 row created.
```

Но несмотря на то что Oracle производит неявное преобразование типов, рекомендуем всегда использовать явное преобразование, так как неявные преобразования не очевидны и могут привести к путанице. К тому же, если администратор базы данных изменит формат даты по умолчанию, неявные преобразования могут и не привести к желаемому результату.

TO_CHAR

Функция TO_CHAR является обратной по отношению к TO_DATE и преобразует дату в символьную строку. Вызов TO_CHAR выглядит следующим образом:

TO_CHAR(дата [,формат])

Рассмотрим элементы конструкции:

дата

Переменная PL/SQL или столбец базы данных типа DATE.

формат

Указывает формат выводимой строки. Формат должен представлять собой допустимую комбинацию форматов.

Указание формата даты является необязательным. Если не задавать формат, то дата выводится в формате по умолчанию (определяемом параметром NLS_DATE_FORMAT).

В следующем примере функция TO_CHAR применяется для преобразования вводимой даты в строку с использованием формата даты по умолчанию:

```
SELECT FNAME, TO_CHAR(HIRE_DATE) FROM EMPLOYEE;  
FNAME      TO_CHAR(H
```

```
-----  
John      22-OCT-99
```

А вот как функция TO_CHAR применяется для преобразования вводимой даты в строку с явным указанием формата даты:

```
SELECT FNAME, TO_CHAR(HIRE_DATE,'MM/DD/YY') FROM EMPLOYEE;  
FNAME      TO_CHAR(
```

```
-----  
John      10/22/99
```

Бывают случаи, когда необходимо использовать функции TO_CHAR и TO_DATE вместе. Например, если вы хотите узнать, каким днем недели было 1 января 2000 года, можно выполнить такой запрос:

```
SELECT TO_CHAR(TO_DATE('01-JAN-2000','DD-MON-YYYY'),'Day') FROM  
DUAL;
```

```
TO_CHAR(T
```

```
-----  
Saturday
```

В данном примере сначала строка '01 -JAN-2000' преобразуется в значение типа DATE, которое затем функция TO_CHAR преобразует в строку, представляющую день недели.

5.2 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение SQL.
2. Реализации SQL.
3. Наборы команд SQL.
4. Команда SELECT.
5. Команда INSERT.
6. Команда UPDATE.
7. Команда DELETE.
8. Типы данных Oracle.
9. Последовательности и их использование.
10. Инструкция WHERE.
11. Объединения.
12. Групповые операции и их применение.
13. Инструкция HAVING.

5.3 ТЕСТ

1. В результате выполнения запроса

```
SELECT ename, job, dept.deptno, dname  
FROM emp, dept  
WHERE emp.deptno=dept.deptno;  
будет выдано:
```

Возможные ответы:

- а) пересечение таблиц emp и dept, по совпадению полей deptno
- б) таблица emp и поля таблицы dept, если совпадают поля deptno
- в) таблица dept и поля таблицы emp, если совпадают поля deptno

Ответы на тест:

1-а.

Список литературы:

1. Архипенков С и др. Хранилища данных. От концепции до внедрения. – М: Диалог-МИФИ, 2002, 424 с.
2. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд./Пер. с англ. - М.: “ Вильямс ”, 2008г. - 721 с., ил.
3. Гайдамакин Н.А. Автоматизация информационных систем, базы и банки данных. – М.: Гелиос АРВ, 2002. – 368с.
4. Гарсиа-Молина, Гектор, Ульман, Джеффри Д., Уидом, Дженнифер. Системы баз данных. Полный курс, Вильямс, 2003.
5. Дейт К. Введение в системы баз данных. М.:Вильямс,2008.-1328с. с ил.
6. Кайт, Том. Oracle для профессионалов: архитектура, методики программирования и основные особенности версий 9i и 10g – М.: Вильямс, 2007.- 848с.
7. Карпова И. П. Базы данных: курс лекций и материалы для практических занятий /И. П. Карпова.-Санкт-Петербург: Питер, 2013.-240 с.: ил.-(Учебное пособие) ISBN 978-5-496-00546-3р.388.70.-1000
8. Коннолли, Томас, Бегг, Каролин. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 1440 с.: ил. – Парал. тит. англ.
9. Маклаков С.В.. BPwin и Erwin. CASE-средства разработки информационных систем. - М.: ДИАЛОГ-МИФИ, 1999 - 256 с.
10. Малыхина М.П. Базы данных: основы, проектирование, использование, 2-е изд. перераб. и доп. – СПб.: БХВ-Петербург, 2007.- 512с.
11. Мишра С., Бьюли А. Секреты Oracle SQL. – Пер. с англ. – СПб: Символ-Плюс, 2005. – 368 с., ил.
12. Ролланд, Фред, Основные концепции баз данных, Вильямс, 2002.

ОГЛАВЛЕНИЕ

1 Основные понятия	3
1.1 Информация как ресурс	3
1.2 Недостатки традиционных файловых систем	6
1.3 Информационные системы. Использующие базы данных	8
1.4 База данных	9
1.5 Преимущества и недостатки СУБД	12
1.6 Программное обеспечение	18
1.7 Компоненты СУБД	21
1.8 Модели данных на основе записей	25
1.9 История развития систем управления базами данных	28
1.10 Стратегическое планирование базы данных	29
1.11 Жизненный цикл базы данных (ЖЦБД)	32
1.12 Трехуровневая архитектура ANSI-SPARC	33
1.13 Архитектура многопользовательских СУБД	38
1.14 Этапы проектирования базы данных	42
1.15 Контрольные вопросы	47
1.16 Упражнения	47
1.17. Тесты	57
2. Построение концептуальной модели	59
2.1 Реальность и модели	59
2.2 Критерии оценки модели данных	60
2.3 Концептуальные модели	60
2.4 Классы и объекты	62
2.5 Классификация	64
2.6 Основы концептуальной модели	72
2.7 Моделирование концептуальных и физических объектов	83
2.8 Примеры	84
2.9 Контрольные вопросы	90
2.10 Упражнения	90
2.11 Тесты	91
3.Реляционная модель	93
3.1 использование реляционной модели	93

3.2 Нормализация	98
3.3 Перевод объектно-ориентированной модели в реляционную	111
3.4 Реляционная алгебра	114
3.5 Контрольные вопросы	127
3.6 Упражнения	128
3.7 Тесты	130
4 Средства автоматизации проектирования БД	132
4.1 Модель «сущность-связь»	132
4.2 Методология IDEF1X	132
4.3 Проблемы ER-моделирования	135
4.4 Создание модели данных с помощью Toad Data Modeler Freeware	137
4.5 Контрольные вопросы	156
4.6 Упражнения	156
4.7 Тесты	158
5 Структурированный язык запросов	160
5.1 Основные конструкции SQL	160
5.2 Контрольные вопросы	175
5.3 Тесты	175
Список литературы	176
Оглавление	177