**Лабораторная работа №**

**Тема: Основные понятия и определения. Этапы проектирования БД**

*Базу данных (БД)* можно определить как унифицированную совокупность данных, совместно используемую различными приложениями в рамках некоторой единой автоматизированной информационной системы.

Программное обеспечение, осуществляющее операции над БД, получило название СУБД – *система управления базами данных.*

*Запрос* – специальным образом описанное требование, определяющее состав производимых над БД операций по выборке или модификации хранимых данных.

Для подготовки запросов чаще всего используют *структурированный язык запросов – SQL (Structured Query Language)*. Этот язык стал фактическим стандартом языка работы с реляционными БД. Он является непроцедурным языком и не содержит операторов управления, организации подпрограмм, ввода-вывода и т. д. Поэтому SQL автономно не используется, а обычно погружен в среду встроенного языка программирования СУБД или процедурного языка (типа C++ ).

Современные СУБД позволяют создавать запросы, не применяя SQL. Однако его применение позволяет расширить возможности СУБД.

Категории SQL-запросов:

· *определение данных* (Data Definition Language, DDL) – SQL-запросы, позволяющие пользователям создавать и модифицировать структуру объектов БД (таблицы, представления и индексы); команды DDL влияют на контейнеры, содержащие данные, а не на данные;

· *запросы данных* (Data Query Language, DQL) – включает выражения SQL для получения данных из базы;

· *манипуляции с данными* (Data Manipulation Language, DML) – SQL-запросы, позволяющие пользова­телю добавлять и удалять данные (в форме строк), а также модифици­ровать имеющиеся в БД;

· *контроль данных* (Data Control Language, DCL) – SQL-запросы, позволяющие администраторам контролировать доступ к данным в базе и использовать различные системные привилегии СУБД;

· *контроль транзакций* – набор команд, которые пользователь применяет для того, чтобы вся транзакция либо была успешно выполнена, либо нет; команды контроля транзакций не вполне соответствуют син­таксису SQL-запросов, но положительно влияют на выполнение запросов, включенных в транзакцию.

Архитектура современных профессиональных СУБД базируется на принципах *клиент-серверного взаимодействия* программных компонентов. Сервер – процесс, обслуживающий информационную потребность клиента.

Клиент – приложение, посылающее запрос на обслуживание сервером. Клиент инициирует связь с сервером, определяет вид запроса, получает от сервера результат обслуживания, подтверждает окончание обслуживания. Поскольку стандартом интерфейса «клиент-сервер» в этом случае является язык SQL, СУБД называют *SQL-сервером*.

На клиентском компьютере может выполняться *SQL-клиент* – программа, предоставленная поставщиком СУБД и обеспечивающая пользователю возможность вводить SQL-запросы, посылать их в СУБД и просматривать результат.

По пользовательскому интерфейсу SQL-клиенты разделяются на три типа:

· *клиент с интерфейсом командной строки* – команды вводятся с клавиатуры как текст, клиент можно использовать в любой операционной системе;

· *клиент с графическим интерфейсом пользователя* (GUI, Graphical User Interface) – выполняется в оконной системе (Microsoft Windows) и отображает данные, используя графические элементы (значки, кнопки и диалоговые окна);

· *клиент с Web-интерфейсом* – выполняется на сервере БД, а для взаимодействия с пользователем используется Web-браузер на клиентском компьютере.

Одним из наиболее популярных SQL-серверов БД является *MySQL* – небольшая и надежная реляционная СУБД с возможностью отката и восстановления после сбоя, многопользовательская, многопоточная, с высокой производительностью. Сервер MySQL предназначен как для критических по задачам производственных систем с большой нагрузкой, так и для встраивания в программное обеспечение массового распространения.

MySQL – открытое программное обеспечение (распространяется с открытым исходным кодом). Благодаря высокой производительности и простоте настройки, богатому выбору API-интерфейсов, а также функциональным средствам работы с сетями, сервер MySQL стал одним из самых удачных вариантов для разработки Web-приложений, взаимодействующих с БД.

Система MySQL может быть реализована как:

· автономная настольная система;

· клиент-серверная система.

Если MySQL используется как автономная настольная система, то клиентское приложение исполняется на том же компьютере, на котором хранится программное обеспечение MySQL и БД. Сетевые соединения от клиента к серверу не устанавливаются. Настольные системы полезны в следующих случаях:

· при доступе к БД лишь одного пользователя;

· при небольшом числе пользователей, работающих с БД не одновременно.

*Клиент-серверная система* может иметь:

· двухзвенную установку;

· трехзвенную установку.

Независимо от варианта установки, программное обеспечение и базы данных MySQL размещаются на центральном компьютере (сервере баз данных)*.* Пользователи работают на компьютерах-клиентах. Доступ пользователей к серверу БД производится при помощи:

· приложений с компьютеров-клиентов (в двухзвенных системах);

· приложений, выполняющихся на специальном компьютере – сервере приложений(в трехзвенных системах).

В двухзвенных системах клиенты исполняют приложения, осуществляющие доступ к серверу БД непосредственно через сеть. Клиенты называются толстыми, поскольку выполняют два вида работы:

· исполняют программный код, соответствующий функциональным задачам;

· исполняют код, отображающий результаты доступа к БД.

Двухзвенная установка полезна при небольшом количестве пользователей, потому что для соединения с каждым из пользователей расходуются системные ресурсы (память и блокировки). Чем больше количество соединений с пользователями, тем хуже производительность системы из-за соперничества за ресурсы.

В трехзвенных системах в задачи компьютеров-клиентов входит лишь исполнение программного кода по вызову функций сервера приложений и отображение результатов. Такие клиенты называются тонкими. Сервер приложений исполняет многопотоковые приложения, с которыми могут работать много пользователей одновременно. Сервер приложений соединяется с сервером БД, осуществляет доступ к данным и возвращает результаты клиенту.

С распространением Интернета клиенты и серверы стали взаимодействовать в глобальной сети. Web-среда предоставила пользователям дружественный интерфейс, за формирование которого отвечает Web-сервер. Такой подход позволил использовать для работы с удаленными БД Web-браузер, не прибегая к услугам специфических клиентских программ. Например, клиенты торговой компании, желающие ознакомиться со списком товаров, используют браузер для посещения сайта компании. Web-страницу со списком товаров формирует специальный модуль (скрипт), выполняющийся на Web-сервере компании. Для получения информации этот скрипт посылает SQL-запросы СУБД, находящейся на сервере БД.

Таким образом, в трехуровневой архитектуре Интернета выделяются:

· клиент – Web-браузер (клиентское приложение), который взаимодействует с Web-сервером, посылая ему запросы на отображение той или иной Web-страницы;

· Web-сервер – на котором выполняется Web-приложение, формирующее SQL-запрос к СУБД (которая должна вернуть необходимые данные из БД);

· сервер баз данных – на котором размещены СУБД и база данных.

**Этапы проектирования реляционной базы данных**

Проектирование реляционной БД в общем случае включает три самостоятельных этапа: концептуальное, логическое и физическое проектирование.

На этапе *концептуального* проектирования изучается и описывается предметная область. Выявляется совокупность сведений и документов об объектах и процессах, характеризующих предметную область и подлежащих загрузке в БД. То есть, определяется информация, обеспечивающая реализацию возможных запросов к БД и решение задач пользователя.

На этом этапе разрабатывается *словарь данных*. Цель создания словаря данных – документирование данных. Он содержит информацию: об источниках данных, их форматах, взаимосвязях, характере использования. Два важнейших назначения у словаря данных:

1) централизованное ведение и управление данными как ресурсом на всех этапах проектирования, эксплуатации и развития БД;

2) обеспечение эффективного взаимодействия между всеми участниками проекта БД.

Пример. Словарь данных для документа «Отчет о ТС и их размещении по рабочим местам»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | Источник данных | Идентификатор данных | Назначение данных | Диапазон изменения | Тип данных | Вид данных | Примечание |
|  |  |  |  |  |  |  |  |
|  | ДО.7 | Name | Наименование | До 60 символов | Симвл. | У |  |
|  | Nomer\_kom | Номер комнаты | До 6 цифр | Целое | У |  |  |
|  | Nomer\_etaj | Номер этажа | До 100 цифр | Целое | У |  |  |
|  | Nomer\_rab\_mesta | Номер рабочего места | До 6 цифр | Целое | У |  |  |
|  | Inv\_Nomer | Инвентарный номер | До 6 цифр | Целое | У |  |  |
|  | Marka | Марка | До 20 символов | Симвл. | У |  |  |
|  | Date\_expl | Дата ввода в эксплуатацию | До 9 цифр | Целое |  |  |  |

Конечный результат этапа концептуального проектирования - информационно-логическая модель данных предметной области (концептуальная модель). Она определяет состав и структуры данных предметной области, функциональную связь между ними.

На этапе *логического* проектирования осуществляется выбор конкретной системы управления базами данных (СУБД) и отображение концептуальной модели в логическую модель, основанную уже на структурах, характерных для выбранной СУБД. Для реляционной БД - это разработка структуры таблиц, определение их ключей, связей между ними, оптимизация создаваемой модели БД (минимизация избыточности данных, устранение их дублирования).

На этапе *физического* проектирования логическая модель расширяется характеристиками, необходимыми для определения способов физического хранения и использования БД, типа устройств для хранения, объема памяти, правил сопровождения БД и т.п.

Обычно современные СУБД содержат средства, позволяющие создавать таблицы, ключи, связи. Существуют и утилиты (поставляемые отдельно от СУБД) с такими возможностями. Все более популярным становится использование специальных средств, называемых CASE-средствами. Существует несколько их типов, но для проектирования БД чаще всего используются инструменты создания диаграмм «сущность-связь» (ER-diagrams). С их помощью создается *логическая* модель данных, в которой прототипы таблиц называются сущностями, а поля – их атрибутами. После определения атрибутов, проведения нормализации, установления связей между сущностями создается *физическая* модель данных для конкретной СУБД, в которой определяются все таблицы, поля и другие объекты БД. После этого можно сгенерировать саму БД.

**Концептуальное проектирование**

Основными задачами инфологического проектирования являются определение предметной области системы и формирование взгляда на ПО с позиций сообщества будущих пользователей БД, т.е. концептуальной модели ПО.

Концептуальная модель ПО представляет собой описание структуры и динамики ПО, характера информационных потребностей пользователей в терминах, понятных пользователю и не зависимых от реализации БД. Это описание выражается в терминах не отдельных объектов ПО и связей между ними, а их типов, связанных с ними ограничений целостности и тех процессов, которые приводят к переходу предметной области из одного состояния в другое.

Рассмотрим основные подходы к созданию концептуальной модели предметной области.

*Функциональный подход к проектированию БД*

Этот метод реализует принцип "от задач" и применяется тогда, когда известны функции некоторой группы лиц и/или комплекса задач, для обслуживания информационных потребностей которых создаётся рассматриваемая БД.

*Предметный подход к проектированию БД*

Предметный подход к проектированию БД применяется в тех случаях, когда у разработчиков есть чёткое представление о самой ПО и о том, какую именно информацию они хотели бы хранить в БД, а структура запросов не определена или определена не полностью. Тогда основное внимание уделяется исследованию ПО и наиболее адекватному её отображению в БД с учётом самого широкого спектра информационных запросов к ней.

*Проектирование с использованием метода "сущность-связь"*

Метод "сущность-связь" (entity-relation, ER-method) является комбинацией двух предыдущих и обладает достоинствами обоих. Этап концептуального проектирования начинается с моделирования ПО. Проектировщик разбивает её на ряд локальных областей, каждая из которых (в идеале) включает в себя информацию, достаточную для обеспечения запросов отдельной группы будущих пользователей или решения отдельной задачи (подзадачи). Каждое локальное представление моделируется отдельно, затем они объединяются.

Выбор локального представления зависит от масштабов ПО. Обычно она разбивается на локальные области таким образом, чтобы каждая из них соответствовала отдельному внешнему приложению и содержала 6-7 сущностей.

**Сущность** - это объект, о котором в системе будет накапливаться информация. Сущности бывают как физически существующие (например, *СОТРУДНИК* или *АВТОМОБИЛЬ*), так и абстрактные (например, *ЭКЗАМЕН* или *ДИАГНОЗ*).

Для сущностей различают тип сущности и экземпляр. Тип характеризуется именем и списком свойств, а экземпляр - конкретными значениями свойств.

Типы сущностей можно классифицировать как сильные и слабые. Сильные сущности существуют сами по себе, а существование слабых сущностей зависит от существования сильных. Например, читатель библиотеки - сильная сущность, а абонемент этого читателя - слабая, которая зависит от наличия соответствующего читателя. Слабые сущности называют подчинёнными (дочерними), а сильные - базовыми (основными, родительскими).

Для каждой сущности выбираются свойства (атрибуты). Различают:

1. *Идентифицирующие и описательные атрибуты*. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются *потенциальными ключами.* Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один первичный ключ (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам записи. Кроме того, ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными и заключают в себе интересующие свойства сущности.

2. *Составные и простые атрибуты*. Простой атрибут состоит из одного компонента, его значение неделимо. Составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, ФИО или адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от характера его обработки и формата пользовательского представления этого атрибута.

3. *Однозначные и многозначные атрибуты* (могут иметь соответственно одно или много значений для каждого экземпляра сущности).

4. *Основные и производные атрибуты*. Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов (например, возраст студента вычисляется на основе даты его рождения и текущей даты).

Спецификация атрибута состоит из его названия, указания типа данных и описания ограничений целостности - множества значений (или домена), которые может принимать данный атрибут.

Далее осуществляется спецификация связей внутри локального представления. Связи могут иметь различный содержательный смысл (семантику). Различают связи типа "сущность-сущность", "сущность-атрибут" и "атрибут-атрибут" для отношений между атрибутами, которые характеризуют одну и ту же сущность или одну и ту же связь типа "сущность-сущность".

Каждая связь характеризуется именем, обязательностью, типом и степенью. Различают **факультативные** и **обязательные** связи. Если вновь порождённый объект одного типа оказывается по необходимости связанным с объектом другого типа, то между этими типами объектов существует обязательная связь (обозначается двойной линией). Иначе связь является факультативной.

По типу различают множественные связи "один к одному" (1:1), "один ко многим" (1:n) и "многие ко многим" (m:n). ER-диаграмма, содержащая различные типы связей, приведена на рисунке 2. Обратите внимание, что обязательные связи на рисунке 2 выделены двойной линией.

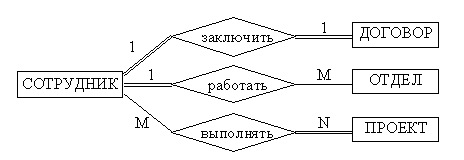


Рисунок 2. ER-диаграмма с примерами типов множественных связей

Степень связи определяется количеством сущностей, которые охвачены данной связью. Пример бинарной связи - связь между отделом и сотрудниками, которые в нём работают. Примером тернарной связи является связь типа *экзамен* между сущностями *ДИСЦИПЛИНА,СТУДЕНТ,* *ПРЕПОДАВАТЕЛЬ*. Из последнего примера видно, что связь также может иметь атрибуты (в данном случае это *Дата проведения* и *Оценка*). Пример ER-диаграммы с указанием сущностей, их атрибутов и связей приведен на рисунке 3.

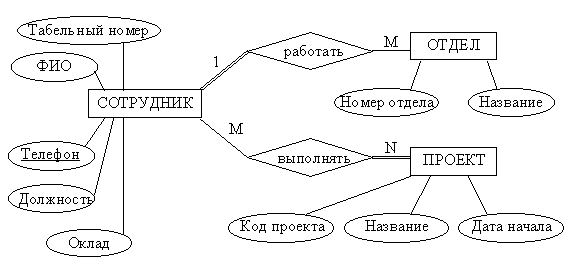


Рисунок 3. Пример ER-диаграммы с однозначными и многозначными атрибутами

После того, как созданы локальные представления, выполняется их объединение. При небольшом количестве локальных областей (не более пяти) они объединяются за один шаг. В противном случае обычно выполняют бинарное объединение в несколько этапов.

При объединении проектировщик может формировать конструкции, производные по отношению к тем, которые были использованы в локальных представлениях. Такой подход может преследовать следующие цели:

· объединение в единое целое фрагментарных представлений о различных свойствах одного и того же объекта;

· введение абстрактных понятий, удобных для решения задач системы, установление их связи с конкретными понятиями, использованными в модели;

· образование классов и подклассов подобных объектов (например, класс "изделие" и подклассы типов изделий, производимых на предприятии).

На этапе объединения необходимо выявить и устранить все противоречия. Например, одинаковые названия семантически различных объектов или связей или несогласованные ограничения целостности на одни и те же атрибуты в разных приложениях. Устранение противоречий вызывает необходимость возврата к этапу моделирования локальных представлений с целью внесения в них соответствующих изменений.

По завершении объединения результаты проектирования являют собой концептуальную инфологическую модель предметной области. Модели локальных представлений - это внешние инфологические модели

**Практический пример.**

**Пример концептуальной модели**

В этом примере рассматривается схема данных для приложения, автоматизирующего работу факультетов университета. Фрагмент соответствующей концептуальной модели представлен на [рис. 4](https://intuit.ru/studies/professional_skill_improvements/1467/courses/218/lecture/27271?page=2#image.8.3).

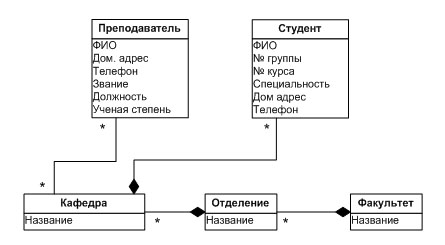


Рис. 4 Пример концептуальной модели

Анализируя эту предметную область, можно выделить следующие сущности - "Студент", "Преподаватель", "Кафедра", "Отделение" и "Факультет", а также их отношения и атрибуты; для отношений показывается множественность. Важно, что в концептуальной модели нет типов атрибутов, а также ключей и индексов, сущности не нормализуются (то есть допускается наличие сложных атрибутов, например "Адрес" и "ФИО"). Все это нужно для того, чтобы такую модель можно было легко обсуждать со специалистами в той предметной области, для которой создается данное приложение, - секретарем декана, заместителем декана по учебной части и пр. Если в концептуальную модель будет добавлена лишняя программистская информация, то, как показывает опыт, она сразу перестанет быть понятной этим людям. В каждом случае этот "порог" может быть своим; он зависит от IT-компетентности специалистов предметной области, соответственно, диапазон используемых модельных средств может варьироваться.

**Реалиазация отношения "многие-ко-многим" для реляционных СУБД**

Этот вид отношения задает связь одного множества объектов с объектами другого множества. На UML-диаграммах такими являются связи, у которых с обоих концов множественность больше единицы - например, и там и там по звездочке, как у связи, соединяющей сущности "Преподаватель" и "Кафедра" на рис. 4 То есть на кафедре может работать много преподавателей, и один преподаватель может работать на многих кафедрах.

Отношение "многие-ко-многим", будучи удобным средством моделирования, не представимо напрямую в реляционной модели данных. Поэтому, рано или поздно, имея в виду, что наши модели схемы данных должны превратиться в структуру реляционных таблиц, это отношение нужно "раскрыть". Часто это целесообразно сделать при переходе от концептуальной модели к логической. И вот почему.

Рассмотрим пример. Слева на рис. 5 слева можно видеть пару сущностей из концептуальной модели - "Преподаватель" и "Кафедра", - которые связаны отношением "многие-ко-многим". Справа на этом же рисунке представлена диаграмма, где отношение "многие-ко-многим" раскрыто с помощью новой сущности и пары отношений "один-ко-многим".

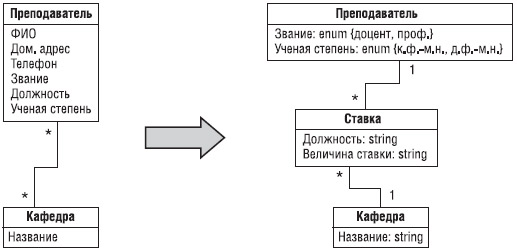


Рис. 5 Пример реализации отношения "многие-ко-многим"

В данном случае новой сущностью является "Ставка". При этом на кафедре может быть много ставок, но каждая ставка принадлежит ровно одной кафедре. И у одного преподавателя может быть много ставок, но одна ставка принадлежит только одному преподавателю. Очевидно, что диаграмма слева эквивалентна диаграмме справа. С одним исключением.

Можно заметить, что в этом примере новая сущность оказалась не фиктивной, а содержательной. В данной предметной области действительно есть такое понятие, как "ставка", и у этой ставки есть свои атрибуты - должность (профессор, доцент и т. д.) и величина ставки (полная, половина, одна треть и т. д.). Каждый преподаватель числится на определенной кафедре с определенными значениями этих атрибутов. Один и тот же преподаватель может работать на разных кафедрах, на разных должностях и на разных ставках.

Таким образом, наличие в предметной области этой важной информации требует, чтобы это отношение "многие-ко-многим" было раскрыто раньше, чем в физической модели - например, при переходе от концептуальной модели к логической.

Тут следует отметить, что иногда такие отношения действительно лучше раскрывать при переходе к физической модели, если существенных атрибутов в новую сущность не добавляется. Ведь диаграммы, созданные с использованием отношения "многие-ко-многим", значительно компактнее тех, которые получаются после "раскрытия" этого отношения. Но опыт показывает, что очень часто такие атрибуты находятся. Более того, в данном примере они настолько важны, что если заказчик и пользователь, с которыми обсуждается концептуальная модель, хоть немного IT-подкованы и способны читать чуть более сложные диаграммы, чем та, которая представлена на рис.4 , то отношение "многие-ко-многим" следует раскрыть уже в концептуальной модели.

**Пример логической модели**

На рис. 6 показан тот же фрагмент предметной области, что и на рис.4, но "расписанный" в терминах логической модели.

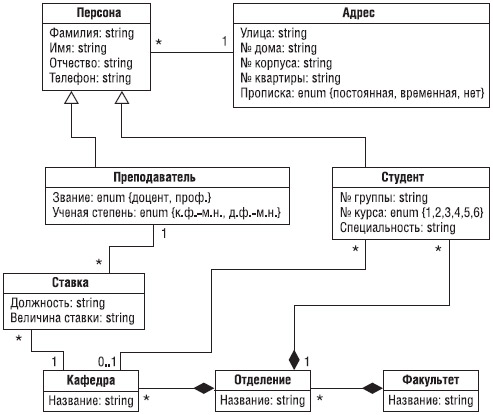


Рис. 6 Пример логической модели

Каковы отличия моделей, представленных на рис. 4 и рис. 6? На первый взгляд видно, что появилось больше сущностей, а у атрибутов уже есть типы. Но это далеко не все.

1. При анализе типов атрибутов некоторые из них - например "Адрес" - были вынесены в отдельные сущности. Необходимо заметить, что типы атрибутов в логической модели могут не совпадать с типами целевой платформы, а нужны для того, чтобы уточнить схему данных: ведь, задумываясь о типах атрибутов, можно, например, создавать новые сущности для сложных типов. Типы также могут быть перечислимыми, т. е. состоять из списка предопределенных значений. Например, важно, что званий бывает два - доцент и преподаватель, курсов - всего шесть, с первого по шестой, ученых степеней всего две - к.ф.-м.н. и д.ф.-м.н. и т. д.
2. Использование наследования. В данном случае это оказалось следствием анализа атрибутов сущностей "Преподаватель" и "Студент". Часть их общих атрибутов была "вынесена" в общего предка - сущность "Персона". Но наследование может появляться и "сверху", когда несколько сущностей являются различными частными случаями одной исходной. В этом случае наследование может использоваться уже в концептуальной модели, но здесь нужно следить, чтобы оно было понятно тем, с кем программисты обсуждают эту модель.
3. Уточнение связей - значений множественности (не все они были точно обозначены в концептуальной модели), а также связанные с этим нюансы предметной области. Например, аналитик понял, что студенты только после второго курса распределяются по кафедрам, а до этого времени учатся все вместе. Но на определенное отделение факультета они поступают изначально. Поэтому сущность "Студент" будет агрегироваться не кафедрой, а отделением. А с кафедрой у него остается связь, причем ее множественность со стороны кафедры - 0..1 (этой связи может не быть, если студент учится на первом или втором курсе).
4. Раскрытие отношения "многие-ко-многим". Об этом уже было рассказано.

Фрагмент логической модели, изображенный на рис. 4, получился сильно упрощенным. Например, часть схемы данных информационной системы для автоматизации государственного университета, отвечающая только за адрес, состоит из девяти разных сущностей - учитывается возможность задания сельского и городского адреса, в состав городского адреса включается возможность задать район и т. д. Преподаватель и студент также описываются с помощью внушительного набора сущностей.

Задание для выполнения:

1. Изучить теоретический материал
2. Выбрать предметную область (согласовать с преподавателем)
3. Выполнить этап концептуального проектирования
4. Выполнить этап логического проектирования
5. Оформить отчет