Д.Е.СКАЧКОВ

ФОРМИРОВАНИЕ БАЗ ДАННЫХ

АДМИНИСТРАЦИЯ СМОЛЕНСКОЙ ОБЛАСТИ
ДЕПАРТАМЕНТ СМОЛЕНСКОЙ ОБЛАСТИ ПО КУЛЬТУРЕ
ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СМОЛЕНСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ ИСКУССТВ»

УТВЕРЖДАЮ:

Проректор по учебной и воспитательной работе _____ Е.В.Горбылева

«_____» _____ 20____г.

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ

Сертификат: 052a556b0039ad5f8c4f749b3ade26ca30
Владелец: Подгузова Елена Евгеньевна
Действителен: с 05.08.2022 по 29.10.2023

Обсуждена на заседании кафедры:

Зав. кафедрой _____ / _____

Протокол № _____

от «_____» _____ 20____г.

Д.Е.СКАЧКОВ

ФОРМИРОВАНИЕ БАЗ ДАННЫХ

рабочая программа дисциплины для обучающихся по направлению 51.03.06 Библиотечно-информационная деятельность, **направленности –** Библиотечно-информационное обеспечение потребителей информации;
форме обучения: заочной

Смоленск
2022

1. ЦЕЛИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целями освоения дисциплины являются:

Формирование профессиональных компетенций выпускника в области анализа, создания, внедрения и сопровождения современных технологий баз данных;

Овладение понятийным аппаратом теории БД;

Приобретение опыта анализа и описания предметной области и учета ее специфики при принятии проектных решений по созданию и модернизации БД;

Изучение истории, современного состояния и перспектив развития БД, а также систем управления базами данных (СУБД)

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОПОП ВО

Дисциплина «Формирование баз данных» находится факультативных дисциплинах учебного плана.

3. ФОРМИРУЕМЫЕ КОМПЕТЕНЦИИ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ ДИСЦИПЛИНЫ.

ОПК-3. Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности

ПК-10. Готов к участию в проектировании, создании и эффективной эксплуатации электронных информационных ресурсов

В результате освоения дисциплины обучающийся должен:

Знать: основные программные и технические средства решения библиотечно-библиографических и информационных задач.

Уметь: ставить и решать прикладные (библиотечно-библиографические и информационные) задачи с использованием БД и СУБД;

Владеть: навыками обоснования и выбора программных средств и операционной среды при проектировании БД.

4. СТРУКТУРА И СОДЕРЖАНИЕ РАБОЧЕЙ ПРОГРАММЫ ДИСЦИПЛИНЫ.

Общая трудоемкость дисциплины составляет 2 зачетные единицы, 72 часа.

Очная форма обучения

№ п/п	Раздел дисциплины	Семестр по очной форме обучения	Недели семестра по очной форме обучения	Виды учебной работы, включая самостоятельную работу и трудоемкость в часах по очной форме обучения				Форма текущего контроля успеваемости (по неделям семестра) форма промежуточной аттестации (по семестрам) по очной форме обучения
				Лекции	Семинары	Практические	Самостоятельная работа	
1	Раздел 1. Ведение в базы данных	7		4		2	30	
2	Раздел 2. Современные БД	8		2		4	30	
				6		**6**	**60**	

5. ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ

Курс «Формирование БД» для обучающихся по направлению подготовки «Библиотечно-информационная деятельность».

При проведении занятий и организации самостоятельной работы обучающихся используются:

ТРАДИЦИОННЫЕ ТЕХНОЛОГИИ ОБУЧЕНИЯ, предполагающиепередачу информации в готовом виде, формирование учебных умений пообразцу: лекция-изложение, лекция-объяснение, практические работы,контрольная работа и др.

Использование традиционных технологий обеспечивает ориентирование обучающегося в потоке информации, связанной с различными подходами к определению сущности, содержания, методов, форм развития информационных технологий.

Практическиезанятия обеспечивают развитие и закрепление умений и навыков работы с информационными технологиями; анализ современных подходов к развитию информационных технологий библиотечной сферы.

ИНТЕРАКТИВНЫЕ ТЕХНОЛОГИИ ОБУЧЕНИЯ, предполагающиеорганизацию обучения как продуктивной деятельности в режимевзаимодействия обучающихсядруг с другом и с преподавателем.

Согласно учебному плану количество часов по дисциплине –108, из нихпроводимых в интерактивной форме –13 часов по очной форме обучения.

5.1.СОДЕРЖАНИЕ РАЗДЕЛОВ (ТЕМ) ДИСЦИПЛИНЫ

Очная форма обучения

Раздел / тема	Семестр по очной форме обучения	Недели семестра по очной форме обучения	Количество часов по очной форме обучения (по заочной форме)				
			Всего	Лекций	Семинаров	Практических	Самостоятельная работа
РАЗДЕЛ 1. ВЕДЕНИЕ В БАЗЫ ДАННЫХ							
Тема 1. Основные понятия теории баз данных	7		**6**	2			4
Тема 2. 3-х уровневая архитектура Ansi-Sparc	7		**4**				4
Тема 3. Реляционная модель данных	7		**4**				4
Тема 4. Проектирование БД методом нормализации	7		**10**	2		2	6
Тема 5. Проектирование БД с использованием семантических моделей	7		**8**			2	6
Тема 6.Язык SQL	7		**6**				6
РАЗДЕЛ 2. СОВРЕМЕННЫЕ БД							
Тема 7. Объектно-ориентированные базы данных	8		**8**	2			6
Тема 8. Распределенные базы данных и системы клиент-сервер	8		**6**				6
Тема 9. Перспективные модели баз данных	8		**8**			2	6
Тема 10. Публикация баз данных в Интернет	8		**6**				6
Тема11. Современные СУБД и их применение	8		**8**			2	6
ИТОГО			**72**	**6**		**6**	**60**

РАЗДЕЛ 1.ВЕДЕНИЕ В БАЗЫ ДАННЫХ

Тема 1. Основные понятия теории баз данных

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

Теория управления базами данных как самостоятельная дисциплина начала развиваться приблизительно с начала 50-х годов двадцатого столетия. За это время в ней сложилась определенная система фундаментальных понятий. Приведем некоторые из них.

Предметной областью принято называть часть реального мира, подлежащую изучению с целью организации управления в этой сфере и последующей автоматизации процесса управления. В рамках данной

книги для нас в первую очередь представляют интерес предметные области, так или иначе связанные со сферой экономики и финансов.

Объектом называется элемент информационной системы, сведения о котором хранятся в базе данных. Иногда объект также называют сущностью (от англ, entity). Классом объектов называют их совокупность, обладающую одинаковым набором свойств.

Атрибут – это информационное отображение свойств объекта. Каждый объект характеризуется некоторым набором атрибутов.

Ключевым элементом данных называются такой атрибут (или группа атрибутов), который позволяет определить Значения других элементов-данных. Запись данных (англ, эквивалент record) - это совокупность значений связанных элементов данных.

Первичный ключ – это атрибут (или группа атрибутов), который уникальным образом идентифицируют каждый экземпляр объекта (запись). Вторичным ключом называется атрибут (или группа атрибутов), значение которого может повторяться для нескольких записей (экземпляров объекта). Прежде всего вторичные ключи используются в операциях поиска записей.

Процедуры хранения данных в базе должны подчиняться некоторым общим принципам, среди которых в первую очередь следует выделить:

- целостность и непротиворечивость данных, под которыми понимается как физическая сохранность данных, так и предотвращение неверного использования данных, поддержка допустимых сочетаний их значений, защита от структурных искажений и несанкционированного доступа;
- минимальная избыточность данных обозначает, что любой элемент данных должен храниться в базе в единственном виде, что позволяет избежать необходимости дублирования операций, производимых с ним.

Программное обеспечение, осуществляющее операции над базами данных, получило название СУБД – система управления базами данных. Очевидно, что его работа должна быть организована таким образом, чтобы выполнялись перечисленные принципы.

Модели организации данных

Набор принципов, определяющих организацию логической структуры хранения данных в базе, получил название модели данных. Модели баз данных определяются тремя компонентами:

- допустимой организацией данных;
- ограничениями целостности;
- множеством допустимых операций.

В теории систем управления базами данных выделяют модели четырех основных типов: иерархическую, сетевую, реляционную и объектно-реляционную.

Терминологической основой дляиерархической и сетевой моделей являются понятия: атрибут, агрегат и запись. Под атрибутом (элементом данных) понимается наименьшая поименованная структурная единица данных. Поименованное множество атрибутов может образовывать агрегат данных. В некоторых случаях отдельно взятый агрегат может состоять из множества экземпляров однотипных данных, или, как еще говорят, являться множественным элементом. Наконец, записью называют составной агрегат, который не входит в состав других агрегатов. В иерархической модели все записи, агрегаты и атрибуты базы данных образуют иерархически организованный набор, то есть такую структуру, в которой все элементы связаны отношениями подчиненности, и при этом любой элемент может подчиняться только одному какому-нибудь другому элементу. Такую форму зависимости удобно изображать с помощью древовидного графа (схемы, состоящей из точек и стрелок, которая связна и не имеет циклов). Пример иерархической структуры базы данных приведен на рис. 1.

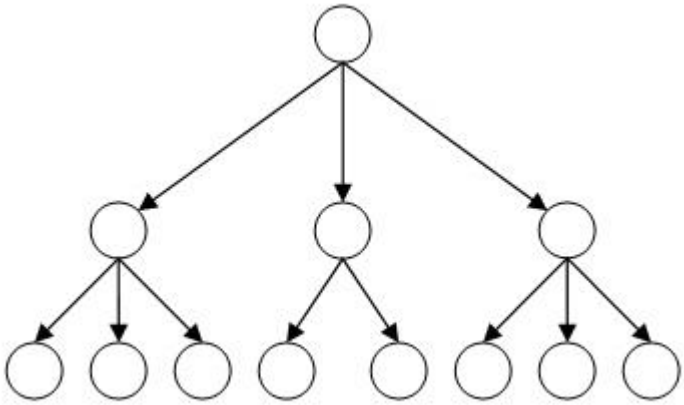


Рис. 1. Схема иерархической модели данных

Типичным представителем семейства баз данных, основанных на иерархической модели, является InformationManagementSystem (IMS) фирмы IBM, первая версия которой появилась в 1968 г. Концепция сетевой модели данных связана с именем Ч. Бахмана. Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков (рис. 7.2).

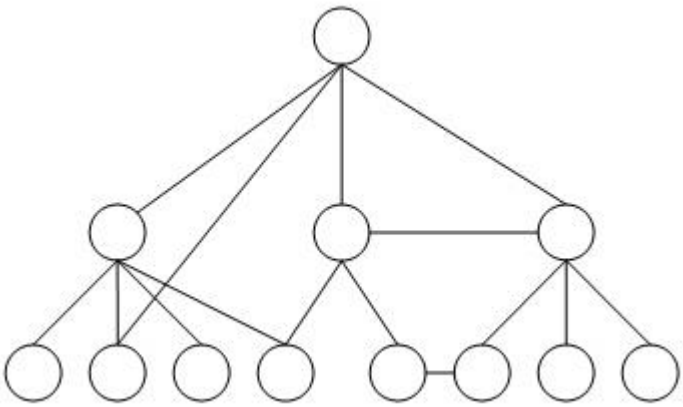


Рис. 2. Схема сетевой модели данных

Сетевая БД состоит из набора записей и набора связей между этими записями, точнее, из набора экземпляров записей заданных типов (из допустимого набора типов) и набора экземпляров из заданного набора типов связи. Примером системы управления данными с сетевой организацией является IntegratedDatabaseManagementSystem (IDMS) компании CullinetSoftwareInc., разработанная в середине 70-х годов. Она предназначена для использования на "больших" вычислительных машинах. Архитектура системы основана на предложениях DataBaseTaskGroup (DBTG), ConferenceonDataSystemsLanguages (CODASYL), организации, ответственной за определение стандартов языка программирования Кобол.

Среди достоинств систем управления данными, основанных на иерархической или сетевой моделях, могут быть названы их компактность и, как правило, высокое быстродействие, а среди недостатков - неуниверсальность, высокая степень зависимости от конкретных данных.

Тема 2. 3-х уровневая архитектура Ansi-Sparc

Основная цель системы управления БД заключается в том, чтобы предоставить пользователю абстрактное представление данных, скрыв конкретные особенности хранения и управления данными. Следовательно, отправной точкой при проектировании БД должно быть общее описание информационных потребностей организации, которые должны найти отражение в создаваемой БД.

Кроме того, поскольку БД является общим ресурсом, то каждому пользователю может потребоваться свое, отличное от других представление о характеристиках информации, представленной в БД. Для удовлетворения этих потребностей структура большинства современных коммерческих СУБД в той или иной степени строится на базе так называемой архитектуры ANSI-SPARC.

Архитектура ANSI-SPARC представляет собой 3 уровня описания элементов данных – внешний, концептуальный и внутренний. Цель трехуровневой архитектуры заключается в отделении пользовательского представления БД от ее физического представления.

Внешний уровень – представление данных с точки зрения пользователей. Он состоит из нескольких различных внешних представлений БД.

Каждый пользователь имеет дело с представлением “реального мира” в наиболее удобной для него форме. Внешнее представление данного пользователя содержит только те сущности, атрибуты и связи, которые ему интересны. Он может ничего не знать о других сущностях, содержащихся в базе. Кроме того, различные представления могут по-разному отображать одни и те же данные (например, форматы дат). Некоторые представления могут содержать производные или вычисляемые данные, которые не хранятся в БД, а создаются по мере надобности.

Концептуальный уровень – обобщающее представление БД. Описывает то, какие данные содержатся в БД, а также связи между ними. Этот уровень содержит логическую структуру БД с точки зрения администратора БД. На концептуальном уровне представлены следующие компоненты:

· все сущности, их атрибуты и связи;

· ограничения, накладываемые на данные;

· семантическая информация о данных;

· информация о мерах обеспечения безопасности и поддержки целостности данных.

Концептуальный уровень поддерживает каждое внешнее представление. Однако этот уровень не содержит никаких сведений о методах хранения данных. Например, описание сущности должно содержать сведения о типах данных, их длине или максимальном количестве символов, но не должно содержать сведений об организации хранения, например об объеме пространства, в байтах.

Внутренний уровень. Физическое представление БД в памяти компьютера. Этот уровень описывает, как информация хранится в базе данных. 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 ОС. На внутреннем уровне хранится следующая информация:

· распределение дискового пространства для хранения данных и индексов.

· сведения о размещении записей.

· сведения о сжатии данных и выбранных методах их шифрования.

Эта архитектура позволяет обеспечить логическую и физическую независимость при работе с данными. Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же БД. Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с этими данными.

Тема 3. Реляционная модель данных

1. Основные понятия реляционной модели данных

В основе реляционных систем лежит реляционная модель данных. Принципы реляционной модели были заложены в 1969–1970 гг. американским ученым Е. Ф. Коддом (E. F. Codd), в то время работавшим в корпорации IBM. Будучи математиком по образованию, он привнес в область управления базами данных строгие математические принципы и точность, которых не хватало ранним системам. Хотя реляционный подход утвердился не сразу, можно отметить, что почти все созданные с конца 70-х гг. продукты баз данных

основаны именно на реляционном подходе. Подавляющее большинство научных исследований в области баз данных в течение последних 35 лет также проводилось именно в этом направлении.

Рассматривая и постепенно уточняя основные понятия реляционной модели, будем иметь в виду три компоненты модели данных:

- структуры данных,
- операции, которые можно выполнять над данными, и
- ограничения, связанные с обеспечением целостности данных.

Основной структурой данных в реляционной модели являются таблицы, называемые в реляционной теории отношениями. Собственно от термина отношение (по-английски relation) и произошло само название модели – реляционная. На рис. 6.1 приведен пример такой таблицы-отношения и пояснение основных терминов реляционной модели – кортеж, кардинальное число, атрибут, степень, домен, первичный ключ.

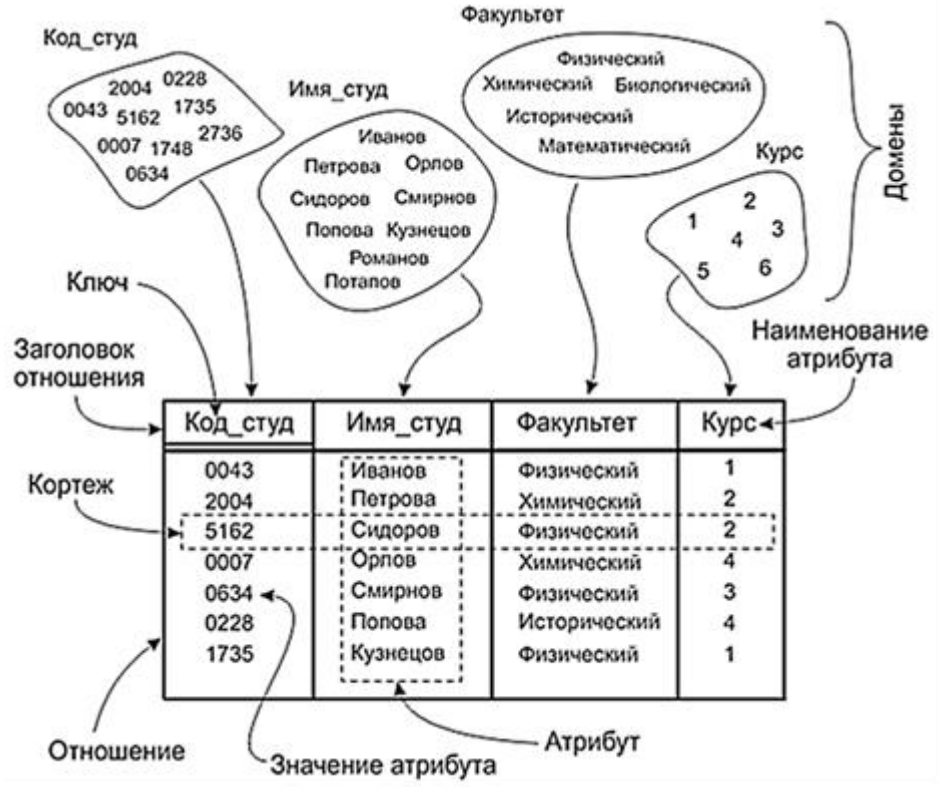


Рис. 3.1. Отношение и его компоненты

Коротко, пока не очень строго, основные понятия реляционной модели можно определить следующим образом.

Отношение это таблица, подобная приведенной на рис. 6.1, состоящая из строк и столбцов и имеющая вверху строку, называемую заголовок отношения.

Строки таблицы-отношения называются кортежами (tuple), а столбцы атрибутами (attribute).

Количество кортежей в отношении называется кардинальным числом отношения, а количество атрибутов называется степенью отношения.

Каждый атрибут в отношении имеет наименование, которое указывается в заголовочной части отношения.

Ключ отношения – это атрибут или набор атрибутов отношения такие, что в любой момент времени в отношении не существует строк, для которых значение или комбинация значений ключевых атрибутов являются одинаковыми. Ключ, таким образом, является уникальным идентификатором кортежей отношения (на рис. 6.1 ключевой атрибут выделен жирным шрифтом).

Домен отношения – это множество значений, из которого могут браться значения конкретного атрибута. То есть конкретный набор значений атрибута в любой момент времени должен быть подмножеством множества значений домена, на котором определен этот атрибут. Значения атрибута, которые отсутствуют в множестве, задаваемом доменом, являются недопустимыми.

Понятие домена является важным для реляционной модели. Домен фактически задает ограничения, которым должны удовлетворять значения соответствующего атрибута.

Как уже отмечалось, приведенные выше определения не являются строгими. Такие термины как таблица, строка, столбец, строго говоря, не являются полностью эквивалентными используемым в реляционной модели математическим понятиям отношение, кортеж, атрибут соответственно. Однако на

практике их часто используют именно как синонимы, что, в общем, допустимо, если при этом понимать, какой действительный смысл вкладывается в эти термины.

2. Структуры данных реляционной модели. Реляционные отношения

Отношение является основной структурой, с помощью которой представляются данные в реляционной модели.

Прежде чем дать более строгое определение понятию отношение, необходимо отметить, что следует различать два понятия: отношение как переменная и значение отношения. Отношение-переменная имеет тот же смысл, что и переменные в языках программирования.

Отношение-переменная – это именованный объект, значение которого может изменяться со временем. Значение этой переменной в любой момент времени и будет значением отношения.

Вначале рассмотрим определение отношения в смысле значение отношения. Отношение-значение представляет собой математическое множество элементов следующего вида.

$<A_1:D_1>$	$<A_2:D_2>$	…	$<A_n:D_n>$
$<A_1:v_{11}>$	$<A_2:v_{12}>$	…	$<A_n:v_{1n}>$
$<A_1:v_{21}>$	$<A_2:v_{22}>$	…	$<A_n:v_{2n}>$
…	…	…	…
$<A_1:v_{j1}>$	$<A_2:v_{j2}>$	…	$<A_n:v_{jn}>$
…	…	…	…
$<A_1:v_{m1}>$	$<A_2:v_{m2}>$	…	$<A_n:v_{mn}>$

Рис. 3.2. Множество элементов, представляющих отношение

Как видно из рисунка 2, множество элементов, из которых образуется отношение R содержит две части: заголовок отношения и тело отношения.

Множество A1, A2, …, An – это множество имен атрибутов отношения.

Совокупность D1, D2, …, Dn – это набор доменов, на котором определено это отношение.

Заголовок отношения – это фиксированное множество атрибутов или, точнее, пар <имя-атрибута: имя-домена>:

{<A1:D1 >, <A2:D2 >, …, <An:Dn >},

причем каждый атрибут Ak соответствует одному и только одному домену Dk (k = 1, 2, …, n), на которых определено отношение.

В отношении все имена атрибутов A1, A2, …, An, должны быть уникальными. Элементы же набора доменов D1, D2, …, Dn не обязательно различны. Отсюда следует, что разные атрибуты отношения могут быть определены на одинаковых доменах.

Иногда структуру, задаваемую заголовком отношения, называют схемой отношения.

Тело отношения – это структура, представляющая собой множество кортежей.

Каждый кортеж –это множество пар <имя-атрибута: значение-атрибута> следущего вида:

{ <A1:Vi 1 >, <A2:Vi 2 >, …, <An:Vi n > }

(i = 1, 2, …, m, где m – количество кортежей в этом множестве).

Каждая такая пара <имя-атрибута :значение-атрибута>, то есть <Ak:Vi k>, соответствует паре <Ak:Dk> для каждого атрибута Ak в заголовке отношения. Для любой такой пары <Ak:Vi k> элемент Vi k является значением из домена Dk , который связан с атрибутом Ak.

Значение *m*, то есть число кортежей в отношении, называется кардинальным числом отношения.

Значение *n*, то есть число атрибутов отношения, называется степенью отношения **R**.

Исходя из приведенных определений уже должно быть видно, что определенное таким образом отношение и обычная двумерная таблица – это не одно и то же. Можно согласиться с тем, что таблица может использоваться в качестве конкретного графического представления на бумаге или экране значения отношения. Однако должно быть понятно, что не всякая таблица является изображением отношения. При использовании таблицы в качестве представления отношения следует придерживаться определенных правил интерпретации ее элементов в соответствие с приведенным определением отношения и его свойствами, которые более подробно рассматриваются ниже.

Определив таким образом понятие отношения в смысле значение отношения, теперь можно сформулировать определение для отношения, понимаемого в смысле переменной, то есть структуры, которая в разные моменты времени принимает различные конкретные значения (вид которых определен выше). Это определение имеет следующий вид.

Пусть **R** – переменная отношения. Переменная R может принимать в разные моменты времени различные значения, однако при этом все возможные значения конкретной переменной отношения R будут иметь одинаковые заголовки (схему).

3 Свойства отношений

Реляционное отношение обладает следующими свойствами.

В отношении нет одинаковых кортежей.

Кортежи отношения не упорядочены (сверху вниз).

Атрибуты отношения не упорядочены (слева направо).

Все значения атрибутов атомарны (скалярны, неделимы).

Поясним эти свойства.

Отсутствие в отношении одинаковых кортежей

Это свойство следует из того факта, что тело отношения – это математическое множество (кортежей). В классической теории множеств по определению множество не содержит одинаковых элементов.

Это свойство является одним из примеров отмечаемой выше неэквивалентности понятий отношение и таблица. Таблица (в общем случае) может содержать одинаковые строки и, следовательно, таблица, содержащая одинаковые строки, не может быть отношением по определению.

Важным следствием того, что в отношении нет одинаковых строк-кортежей, является то, что в отношении всегда существует по крайней мере один потенциальный ключ. Действительно, так как кортежи уникальны, то обязательно комбинация всех или части атрибутов будет обладать свойством уникальности, и, следовательно, может служить ключом отношения, однозначно идентифицирующим кортежи.

Кортежи отношения не упорядочены (сверху вниз)

Это свойство также следует из того, что тело отношения – это математическое множество, а простые множества в математике не упорядочены. Так, в отношении, представленном на рис. 6.1, кортежи могли быть расположены в любом другом порядке, и, тем не менее, это все равно было бы то же самое отношение. Исходя из сказанного, к отношению не применимы такие понятия, как первый кортеж, последний кортеж, десятый кортеж, следующий или предыдущий кортеж и т. д. Другими словами, в отношении отсутствует позиционной адресация кортежей.

Обращение к конкретному кортежу, его идентификация могут быть осуществлены только по ключу отношения.

Это свойство также служит иллюстрацией не эквивалентности понятий отношение и таблица. В отличие от отношения, строки в таблице всегда упорядочены – есть первая строка, вторая, последняя и т. д.

Атрибуты отношения не упорядочены (слева направо)

Это свойство следует из того факта, что заголовок отношения также определен как простое математическое множество, а именно: множество пар <имя-атрибута: имя-домена>.

Исходя из этого, если в представленной на рис. 6.1 таблице те же атрибуты были бы расставлены в каком-либо другом порядке, то это все равно было бы то же самое отношение. Поэтому не существует таких понятий, как первый атрибут, последний атрибут, следующий или предыдущий атрибут и т. д. Атрибут всегда адресуется или определятся по его имени, а не по расположению в отношении.

По этому пункту понятия отношение и таблица также не совпадают. Столбцы обычной таблицы, в отличие от атрибутов отношения, всегда упорядочены слева направо.

Значения всех атрибутов являются атомарными

В реляционной модели домены, на которых определены атрибуты отношения, и из которых «черпаются» фактические значения атрибутов, могут содержать только атомарные (неделимые, скалярные) значения. Другими словами на пересечении столбца и строки таблицы, представляющей отношение, должно быть в точности одно значение, а не набор значений или какая либо сложная (составная) структура значений.

Исходя из изложенного выше материала, термин реляционная база данных можно определить следующим образом.

Реляционная база данных – это база данных, воспринимаемая пользователем как набор нормализованных отношений различной степени.

Выражение воспринимаемая пользователем говорит о том, что идея реляционной модели относится к внешнему и концептуальному уровням системы, а не к внутреннему уровню.

Концептуальная привлекательность реляционной модели данных состоит в том, что в ней все данные предметной области представляются одним и только одним способом с помощью достаточно простой структуры данных, которой является отношение, и четко описанным набором небольшого числа операций над ними. С помощью отношений представляются как сущности предметной области, так и связи между сущностями. При этом от пользователя скрыты явные указатели на файлы или записи данных, их физический или логический порядок и место расположения.

Проводя параллель между описанием предметной области в терминах сущностей, их свойств и связей между ними в модели сущность-связь и рассматриваемой реляционной моделью, можно отметить, что реляционное отношение является средством для представления множества экземпляров однотипных (обладающих одинаковым набором свойств) сущностей или связей моделируемой предметной области. Атрибуты отношения, вместе с доменами, на которых они определены, служат средством описания свойств сущностей или связей. Кортежи отношения представляют в этом случае множество конкретных экземпляров сущностей и связей одного типа. Значения ключевых атрибутов являются средством идентификации этих представляемых кортежами экземпляров.

Тема 4. Проектирование БД методом нормализации

При проектировании базы данных решаются две основных проблемы:

- Каким образом отобразить объекты предметной области в абстрактные объекты модели данных? Это логическое проектирование баз данных.
- Как обеспечить эффективность выполнения запросов к базе данных? Это проблема физического проектирования баз данных.

В случае реляционных баз данных трудно представить какие-либо общие рецепты по части физического проектирования. Здесь слишком много зависит от используемой СУБД. Поэтому мы ограничимся вопросами логического проектирования реляционных баз данных, которые существенны при использовании любой реляционной СУБД.

Будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том,

- из каких отношений должна состоять БД и
- какие атрибуты должны быть у этих отношений
- определение первичных, внешних ключей и ограничений целостности.

Проектирование реляционных баз данных с использованием нормализации.

Сначала рассмотрим классический подход, при котором весь процесс проектирования производится в терминах реляционной модели данных, путем последовательных приближений к удовлетворительному набору схем отношений.

Предметная область - это часть реального мира, данные о которой мы хотим отразить в базе данных. Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т.д. Предметная область бесконечна и содержит как существенно важные понятия и данные, так и малозначащие или вообще не значащие данные. Так, если в качестве предметной области выбрать учет товаров на складе, то понятия "накладная" и "счет-фактура" являются существенно важными понятиями, а то, что сотрудница, принимающая накладные, имеет двоих детей - это для учета товаров неважно. Однако, с точки зрения отдела кадров данные о наличии детей являются существенно важными. Таким образом, важность данных зависит от выбора предметной области.

Исходной точкой проектирования является представление предметной области в виде одного или нескольких отношений. На каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами. Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

Рассмотрим в качестве предметной области некоторую организацию, выполняющую некоторые проекты. Модель предметной области опишем следующим неформальным текстом:

- Сотрудники организации выполняют проекты.
- Проекты состоят из нескольких заданий.
- Каждый сотрудник может участвовать в одном или нескольких проектах, или временно не участвовать ни в каких проектах.
- Над каждым проектом может работать несколько сотрудников, или временно проект может быть приостановлен, тогда над ним не работает ни один сотрудник.
- Над каждым заданием в проекте работает ровно один сотрудник.
- Каждый сотрудник числится в одном отделе.
- Каждый сотрудник имеет телефон, находящийся в отделе сотрудника.

В ходе дополнительного уточнения того, какие данные необходимо учитывать, выяснилось следующее:

- О каждом сотруднике необходимо хранить табельный номер и фамилию. Табельный номер является уникальным для каждого сотрудника.

- Каждый отдел имеет уникальный номер.
- Каждый проект имеет номер и наименование. Номер проекта является уникальным.
- Каждая работа из проекта имеет номер, уникальный в пределах проекта. Работы в разных проектах могут иметь одинаковые номера.

Первая нормальная форма (1НФ) - это обычное отношение. Любое отношение автоматически уже находится в 1НФ. Напомним кратко свойства отношений (это и будут свойства 1НФ):

- В отношении нет одинаковых кортежей.
- Кортежи не упорядочены.
- Атрибуты не упорядочены и различаются по наименованию.
- Все значения атрибутов атомарны.

В ходе логического моделирования на первом шаге предложено хранить данные в одном отношении, имеющем следующие атрибуты:

СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ (Н_СОТР, ФАМ, Н_ОТД, ТЕЛ, Н_ПРО, ПРОЕКТ, Н_ЗАДАН)
где
Н_СОТР *- табельный номер сотрудника*
ФАМ *- фамилия сотрудника*
Н_ОТД *- номер отдела, в котором числится сотрудник*
ТЕЛ *- телефон сотрудника*
Н_ПРО *- номер проекта, над которым работает сотрудник*
ПРОЕКТ *- наименование проекта, над которым работает сотрудник*
Н_ЗАДАН *- номер задания, над которым работает сотрудник*

*Т.к. каждый сотрудник в каждом проекте выполняет ровно одно задание, то в качестве потенциального первичного ключа отношения необходимо взять пару атрибутов {**Н_СОТР**, **Н_ПРО**}.*
В текущий момент состояние предметной области отражается следующими фактами(выделены ключевые атрибуты):

Н_СОТР	***ФАМ***	***Н_ОТД***	***ТЕЛ***	***Н_ПРО***	***ПРОЕКТ***	***Н_ЗАДАН***
1	*Иванов*	*1*	*11-22-33*	***1***	*Космос*	*1*
1	*Иванов*	*1*	*11-22-33*	***2***	*Климат*	*1*
2	*Петров*	*1*	*11-22-33*	***1***	*Космос*	*2*
3	*Сидоров*	*2*	*33-22-11*	***1***	*Космос*	*3*
3	*Сидоров*	*2*	*33-22-11*	***2***	*Климат*	*2*

Таблица 1 Отношение СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ

Недостатки этого отношения:

- Невозможность вставить сотрудника, который не принимает участие ни в одном проекте, аналогично невозможно вставить проект, если им пока никто не занимается – аномалия вставки. Причина этой аномалии - хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту).
- Изменение информации о фамилии, названии проекта, телефоне приводит к необходимости изменять информацию в нескольких местах – аномалия обновления. Причина этой аномалии –избыточность хранения информации (повтор фамилий, названий проектов, телефонов).
- При удалении информации, например о проекте Космос, будет утеряна информация о Петрове, т.к. он работает только в том проекте – аномалия удаления. Причина этой аномалии - хранение в одном отношении разнородной информации.

<u>Вывод</u> - *Отношение **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** находится в 1НФ, но при этом, как было показано выше, логическая модель данных не соответствует модели предметной области. База данных, основанная на такой модели, будет работать неправильно. Таким образом, первой нормальной формы недостаточно для правильного моделирования данных.*

Для устранения указанных аномалий, для правильного проектирования модели данных применяется метод нормализации отношений.

Перевод отношения в следующую нормальную форму осуществляется методом "декомпозиции без потерь". Такая декомпозиция должна обеспечить то, что запросы (выборка данных по условию) к исходному отношению и к отношениям, получаемым в результате декомпозиции, дадут одинаковый результат.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма «лучше» предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих форм сохраняются.

В основе процесса проектирования лежит метод нормализации или декомпозиции отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Метод нормализации (а значит процесс проектирования) основан на понятии функциональной зависимости атрибутов отношения – фундаментальное понятие в теории реляционных баз данных.

Определение. Функциональная зависимость

В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: R.X (r) R.Y $X \rightarrow Y$.

***<u>Пример 1</u>**. В отношении **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** можно привести следующие примеры функциональных зависимостей:*

Зависимость атрибутов от ключа отношения:

{Н_СОТР, Н_ПРО} $\rightarrow$ ***ФАМ***
{Н_СОТР, Н_ПРО} $\rightarrow$ ***Н_ОТД***
{Н_СОТР, Н_ПРО} $\rightarrow$ ***ТЕЛ***
{Н_СОТР, Н_ПРО} $\rightarrow$ ***ПРОЕКТ***
{Н_СОТР, Н_ПРО} $\rightarrow$ ***Н_ЗАДАН***

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

Н_СОТР $\rightarrow$ ***ФАМ***
Н_СОТР $\rightarrow$ ***Н_ОТД***
Н_СОТР $\rightarrow$ ***ТЕЛ***

Зависимость наименования проекта от номера проекта:

Н_ПРО $\rightarrow$ ***ПРОЕКТ***

Зависимость номера телефона от номера отдела:

Н_ОТД $\rightarrow$ ***ТЕЛ***

<u>Замечание</u>. Если атрибуты X составляют потенциальный ключ отношения R, то любой атрибут отношения функционально зависит от X.

*Необходимо отличать понятия функциональной зависимости в математике и РБД. В математическом понятии функции для фиксированного значения X соответствующее значение функции $y = f(x)$ всегда одно и то же. В противоположность этому в отношениях значение зависимого атрибута может принимать различные значения в различных состояниях базы данных. Например, атрибут **ФАМ** функционально зависит от атрибута **Н_СОТР**. Предположим, что сейчас сотрудник с табельным номером 1 имеет фамилию «Иванов», но сотрудник может сменить фамилию, например на «Сидоров».*

Определение. Неключевой атрибут

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав любого ключа (в частности первичного ключа).

Пример ФАМ, Н_ОТД, ТЕЛ, ПРОЕКТ, Н_ЗАДАН

Вторая нормальная форма

Определение. Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда находится в 1NF, и каждый неключевой атрибут полностью зависит от первичного ключа (т.е.

от всех атрибутов в него входящих), если первичный ключ отношения является простым (т.е. ключ состоит из одного атрибута), то отношение автоматически находится в 2НФ.

Отношение ***СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ*** *не находится в 2НФ, т.к. есть атрибуты, зависящие от части составного первичного ключа:*
Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника является зависимостью от части сложного ключа:
Н_СОТР → ФАМ ***Н_СОТР → Н_ОТД*** ***Н_СОТР → ТЕЛ***

Зависимость наименования проекта от номера проекта является зависимостью от части первичног ключа: ***Н_ПРО → ПРОЕКТ***
Для того, чтобы устранить зависимость атрибутов от части первичного ключа, нужно произвести ***декомпозицию*** *отношения на несколько отношений. При этом те атрибуты, которые зависят от части первичного ключа, выносятся в отдельное отношение.*
Отношение ***СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ*** *декомпозируем на три отношения –*

СОТРУДНИКИ_ОТДЕЛЫ, ***ПРОЕКТЫ***, ***ЗАДАНИЯ***.

Отношение ***СОТРУДНИКИ_ОТДЕЛЫ (Н_СОТР, ФАМ, Н_ОТД, ТЕЛ)****:*
Функциональные зависимости:
Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:
Н_СОТР → ФАМ
Н_СОТР → Н_ОТД
Н_СОТР → ТЕЛ
Зависимость номера телефона от номера отдела:
Н_ОТД → ТЕЛ

Н_СОТР	ФАМ	Н_ОТД	ТЕЛ
1	Иванов	1	11-22-33
2	Петров	1	11-22-33
3	Сидоров	2	33-22-11

Отношение СОТРУДНИКИ_ОТДЕЛЫ

Отношение ***ПРОЕКТЫ (Н_ПРО, ПРОЕКТ)****:*
Функциональные зависимости:
Н_ПРО → ПРОЕКТ

Н_ПРО	ПРОЕКТ
1	Космос
2	Климат

Отношение ПРОЕКТЫ

Отношение ***ЗАДАНИЯ (Н_СОТР, Н_ПРО, Н_ЗАДАН)****:*
Функциональные зависимости:
{Н_СОТР, Н_ПРО} → Н_ЗАДАН

Н_СОТР	Н_ПРО	Н_ЗАДАН
1	1	1
1	2	1
2	1	2
3	1	3
3	2	2

Отношения ЗАДАНИЯ

Анализ декомпозированных отношений:

Отношения, полученные в результате декомпозиции, находятся в 2НФ. Действительно, отношения ***СОТРУДНИКИ_ОТДЕЛЫ*** *и* ***ПРОЕКТЫ*** *имеют простые ключи, следовательно автоматически находятся в 2НФ, отношение* ***ЗАДАНИЯ*** *имеет сложный ключ, но единственный неключевой атрибут* ***Н_ЗАДАН*** *функционально зависит от всего ключа* ***{Н_СОТР, Н_ПРО}****.*

Часть аномалий устранена:

- *Так, данные о сотрудниках и проектах теперь хранятся в различных отношениях, поэтому при появлении сотрудников, не участвующих ни в одном проекте просто добавляются кортежи в отношение* ***СОТРУДНИКИ_ОТДЕЛЫ****. Точно также, при появлении проекта, над которым не работает ни один сотрудник, просто вставляется кортеж в отношение* ***ПРОЕКТЫ****.*
- *Фамилии сотрудников и наименования проектов теперь хранятся без избыточности. Если сотрудник сменит фамилию или проект сменит наименование, то такое обновление будет произведено в одном месте*
- *Если по проекту временно прекращены работы, но требуется, чтобы сам проект сохранился, то для этого проекта удаляются соответствующие кортежи в отношении* ***ЗАДАНИЯ****, а данные о самом проекте и данные о сотрудниках, участвовавших в проекте, остаются в отношениях* ***ПРОЕКТЫ*** *и* ***СОТРУДНИКИ_ОТДЕЛЫ****.*

Тем не менее, часть аномалий разрешить не удалось:

- *В отношение* ***СОТРУДНИКИ_ОТДЕЛЫ*** *нельзя вставить кортеж (4, Пушников, 1, 33-22-11), т.к. при этом получится, что два сотрудника из 1-го отдела (Иванов и Пушников) имеют разные номера телефонов, а это противоречит модели предметной области. В этой ситуации можно предложить два решения, в зависимости от того, что реально произошло в предметной области. Другой номер телефона может быть введен по двум причинам - по ошибке человека, вводящего данные о новом сотруднике, или потому что номер в отделе действительно изменился. Тогда можно написать триггер, который при вставке записи о сотруднике проверяет, совпадает ли телефон с уже имеющимся телефоном у другого сотрудника этого же отдела. Если номера отличаются, то система должна задать вопрос, оставить ли старый номер в отделе или заменить его новым. Если нужно оставить старый номер (новый номер введен ошибочно), то кортеж с данными о новом сотруднике будет вставлен, но номер телефона будет у него будет тот, который уже есть в отделе (в данном случае, 11-22-33). Если же номер в отделе действительно изменился, то кортеж будет вставлен с новым номером, и одновременно будут изменены номера телефонов у всех сотрудников этого же отдела. И в том и в другом случае не обойтись без разработки громоздкого триггера.).*
- *если в отделе меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где этот номер телефона встречаются, иначе отношение станет некорректным –* ***аномалия обновления****. Причина аномалии - избыточность данных, также порожденная тем, что в одном отношении хранится разнородная информация.*
- *При удалении некоторых данных по-прежнему может произойти потеря другой информации. Например, если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11 –* ***аномалия удаления****. Причина аномалии - хранение в одном отношении разнородной информации (и о сотрудниках, и об отделах).*

Вывод - логическая модель данных не соответствует предметной области. База данных, основанная на такой модели, будет работать неправильно. Хотя основные трудности теперь связаны только с одним отношением ***СОТРУДНИКИ_ОТДЕЛЫ****.*

Третья нормальная форма

Определение. Атрибуты называются взаимно независимыми, если ни один из них не является функционально зависимым от другого.

Определение. Отношение R находится в третьей нормальной форме (3НФ) тогда и только тогда, когда отношение находится в 2НФ и все неключевые атрибуты взаимно независимы.

Отношение ***СОТРУДНИКИ_ОТДЕЛЫ*** *не находится в 3НФ, т.к. имеется функциональная зависимость неключевых атрибутов (зависимость номера телефона от номера отдела):*

Н_ОТД $\rightarrow$ ***ТЕЛ***

Для того, чтобы устранить зависимость неключевых атрибутов, нужно произвести декомпозицию отношения на несколько отношений. При этом те неключевые атрибуты, которые являются зависимыми, выносятся в отдельное отношение.

Отношение ***СОТРУДНИКИ_ОТДЕЛЫ*** *декомпозируем на два отношения -* ***СОТРУДНИКИ****,* ***ОТДЕЛЫ****.*

Отношение ***СОТРУДНИКИ*** *(****Н_СОТР****,* ***ФАМ****,* ***Н_ОТД****):*

Функциональные зависимости:

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

Н_СОТР $\rightarrow$ ***ФАМ***

Н_СОТР $\rightarrow$ ***Н_ОТД***

Н_СОТР	***ФАМ***	***Н_ОТД***
1	*Иванов*	*1*
2	*Петров*	*1*
3	*Сидоров*	*2*

Отношение СОТРУДНИКИ

Отношение ***ОТДЕЛЫ (Н_ОТД, ТЕЛ):***
Функциональные зависимости:
Зависимость номера телефона от номера отдела:
Н_ОТД $\rightarrow$ ***ТЕЛ***

Н_ОТД	***ТЕЛ***
1	*11-22-33*
2	*33-22-11*

Отношение ОТДЕЛЫ

Обратим внимание на то, что атрибут ***Н_ОТД****, не являвшийся ключевым в отношении* ***СОТРУДНИКИ_ОТДЕЛЫ****, становится потенциальным ключом в отношении* ***ОТДЕЛЫ****. Именно за счет этого устраняется избыточность, связанная с многократным хранением одних и тех же номеров телефонов.*
Вывод. Таким образом, все обнаруженные аномалии обновления устранены. Реляционная модель, состоящая из четырех отношений ***СОТРУДНИКИ****,* ***ОТДЕЛЫ****,* ***ПРОЕКТЫ****,* ***ЗАДАНИЯ****, находящихся в третьей нормальной форме соответствует описанной предметной области.*

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

Замечание. На практике, при создании логической модели данных, как правило, не следуют прямо приведенному алгоритму нормализации. Опытные разработчики обычно сразу строят отношения в 3НФ. Тем не менее, приведенный алгоритм важен:

- этот алгоритм показывает, какие проблемы возникают при разработке слабо нормализованных отношений.
- 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.

Что получилось – обобщение примера:

Н_СОТР	***ФАМ***	***Н_ОТД***
1	*Иванов*	*1*
2	*Петров*	*1*
3	*Сидоров*	*2*

Отношение СОТРУДНИКИ

Н_ОТД	***ТЕЛ***
1	*11-22-33*
2	*33-22-11*

Отношение ОТДЕЛЫ

Н_ПРО	***ПРОЕКТ***
1	*Космос*
2	*Климат*

Отношение ПРОЕКТЫ

Н_СОТР	*Н_ПРО*	*Н_ЗАДАН*
1	*1*	*1*
1	*2*	*1*
2	*1*	*2*
3	*1*	*3*
3	*2*	*2*

Отношения ЗАДАНИЯ

НФБК (Нормальная Форма Бойса-Кодда)

При приведении отношений при помощи алгоритма нормализации к отношениям в 3НФ неявно предполагалось, что все отношения содержат один потенциальный ключ. Это не всегда верно. Рассмотрим следующий пример отношения, содержащего два ключа.

Пример 1. Пусть требуется хранить данные о поставках деталей некоторыми поставщиками. Предположим, что наименования поставщиков являются уникальными. Кроме того, каждый поставщик имеет свой уникальный номер. Данные о поставках можно хранить в следующем отношении:

Номер поставщика PNUM	**Наименование поставщика PNAME**	**Номер детали DNUM**	**Поставляемое количество VOLUME**
1	Фирма 1	1	100
1	Фирма 1	2	200
1	Фирма 1	3	300
2	Фирма 2	1	150
2	Фирма 2	2	250
3	Фирма 3	1	1000

Отношение "Поставки"

Данное отношение содержит два потенциальных ключа - **{*PNUM, DNUM*}** и **{*PNAME, DNUM*}**. Видно, что данные хранятся в отношении с избыточностью - при изменении наименования поставщика, это наименование нужно изменить во всех кортежах, где оно встречается.

Выявляем имеющиеся функциональные зависимости (как обычно, курсивом выделены ключевые атрибуты):

PNUM $\rightarrow$ ***PNAME***.
PNAME $\rightarrow$ ***PNUM***

{*PNUM, DNUM*} $\rightarrow$ **VOLUME**
{*PNUM, DNUM*} $\rightarrow$ ***PNAME***

{*PNAME, DNUM*} $\rightarrow$ **VOLUME**
{*PNAME, DNUM*} $\rightarrow$ ***PNUM***

Данное отношение не содержит *неключевых* атрибутов, зависящих от части сложного ключа (см. определение 2НФ). Действительно, от части сложного ключа зависят атрибуты ***PNAME*** и ***PNUM***, но они сами являются ключевыми. Таким образом, отношение находится в 2НФ.

Кроме того, отношение не содержит зависимых друг от друга *неключевых* атрибутов, т.к. неключевой атрибут всего один - **VOLUME** (см. определение 3НФ). Таким образом, показано, что отношение "Поставки" находится в 3НФ.

Таким образом, описанный ранее алгоритм нормализации неприменим к данному отношению. Очевидно, однако, что аномалия данного отношения устраняется путем декомпозиции его на два следующих отношения:

Номер поставщика PNUM	**Наименование поставщика PNAME**
1	**Фирма 1**
2	**Фирма 2**
3	**Фирма 3**

Отношение "Поставщики"

Номер поставщика PNUM	Номер детали DNUM	Поставляемое количество VOLUME
1	1	100
1	2	200
1	3	300
2	1	150
2	2	250
3	1	1000

Отношение "Поставки-2"

Определение . Отношение R находится в нормальной форме Бойса-Кодда (НФБК) тогда и только тогда, когда множество атрибутов X (детерминанты) всех функциональных зависимостей являются потенциальными ключами.

$X \rightarrow Y$ X –множество атрибутов X – детерминанта; Y – зависимая часть

Отношение "Поставки" не находится в НФБК, т.к. имеются зависимости (***PNUM*** $\rightarrow$ ***PNAME*** и ***PNAME*** $\rightarrow$ ***PNUM***), детерминанты которых не являются потенциальными ключами.

Для того чтобы устранить зависимость от детерминантов, не являющихся потенциальными ключами, необходимо провести декомпозицию, вынося эти детерминанты и зависимые от них части в отдельное отношение. Отношения "Поставщики" и "Поставки-2", полученные в результате декомпозиции находятся в НФБК.

Замечание. Отношение "Поставки-2", полученное в результате декомпозиции имеет всего *один потенциальный ключ*. Поэтому, для анализа отношения "Поставки-2" не требуется привлекать определение НФБК, достаточно определения 3НФ.

Отношение "Поставщики" имеет два потенциальных ключа, но, т.к. других атрибутов в нем нет, оно уже так просто устроено, что упростить его дальше нельзя.

4НФ (Четвертая Нормальная Форма)

Рассмотрим следующий пример. Пусть требуется учитывать данные об абитуриентах, поступающих в ВУЗ. При анализе предметной области были выделены следующие требования:

- Каждый абитуриент имеет право сдавать экзамены на несколько факультетов одновременно.
- Каждый факультет имеет свой список сдаваемых предметов.
- Один и тот же предмет может сдаваться на нескольких факультетах.
- Абитуриент обязан сдавать все предметы, указанные для факультета, на который он поступает, несмотря на то, что он, может быть, уже сдавал такие же предметы на другом факультете.

Предположим, что нам требуется хранить данные о том, какие предметы должен сдавать каждый абитуриент. Попытаемся хранить данные в одном отношении "Абитуриенты-Факультеты-Предметы":

Абитуриент	Факультет	Предмет
Иванов	Математический	Математика
Иванов	Математический	Информатика
Иванов	Физический	Математика
Иванов	Физический	Физика
Петров	Математический	Математика
Петров	Математический	Информатика

Отношение "Абитуриенты-Факультеты-Предметы"

В данный момент в отношении хранится информация о том, что абитуриент Иванов поступает на два факультета (математически и физический), а абитуриент Петров - только на математический. Кроме того, можно сделать вывод, что на математическом факультете нужно сдавать математику и информатику, а на физическом - математику и физику.

Кажется, что в отношении имеется аномалия обновления, связанная с тем, что дублируются фамилии абитуриентов, наименования факультетов и наименования предметов. Однако эта аномалия легко устраняется стандартным способом - вынесением всех наименований в отдельные отношения, оставляя в исходном отношении только соответствующие номера:

Номер	Номер	Номер

Абитуриента	Факультета	Предмета
1	1	1
1	1	2
1	2	1
1	2	3
2	1	1
2	1	2

Модифицированное отношение "Абитуриенты-Факультеты-Предметы"

Номер Абитуриента	Абитуриент
1	Иванов
2	Петров

Отношение "Абитуриенты"

Номер Факультета	Факультет
1	Математический
2	Физический

Отношение "Факультеты"

Номер Предмета	Предмет
1	Математика
2	Информатика
3	Физика

Отношение "Предметы"

Теперь каждое наименование встречается только в одном месте.

И все-таки как в исходном, так и в модифицированном отношении имеются аномалии обновления, возникающие при попытке вставить или удалить кортежи.

Аномалия вставки. При попытке добавить в отношение "Абитуриенты-Факультеты-Предметы" новый кортеж, например (Сидоров, Математический, Математика), мы обязаны добавить также и кортеж (Сидоров, Математический, Информатика), т.к. все абитуриенты математического факультета обязаны иметь один и тот же список сдаваемых предметов. Соответственно, при попытке вставить в модифицированное отношении кортеж (3, 1, 1), мы обязаны вставить в него также и кортеж (3, 1, 2).

Аномалия удаления. При попытке удалить кортеж (Иванов, Математический, Математика), мы обязаны удалить также и кортеж (Иванов, Математический, Информатика) по той же самой причине.

Таким образом, вставка и удаление кортежей не может быть выполнена независимо от других кортежей отношения.

Кроме того, если мы удалим кортеж (Иванов, Физический, Математика), а вместе с ним и кортеж (Иванов, Физический, Физика), то будет потеряна информация о предметах, которые должны сдаваться на физическом факультете.

Декомпозиция отношения "Абитуриенты-Факультеты-Предметы" для устранения указанных аномалий не может быть выполнена на основе функциональных зависимостей, т.к. это отношение не содержит никаких функциональных зависимостей. Это отношение является полностью ключевым, т.е. ключом отношения является все множество атрибутов. Но ясно, что какая-то взаимосвязь между атрибутами имеется. Эта взаимосвязь описывается понятием многозначной зависимости.

В отношении "Абитуриенты-Факультеты-Предметы" имеется многозначная зависимость Факультет $\rightarrow\rightarrow$ Абитуриент|Предмет.

Словами это можно выразить так - для каждого факультета (для каждого значения из X) каждый поступающий на него абитуриент (значение из Y) сдает *один и тот же список* предметов (набор значений из Z), и для каждого факультета (для каждого значения из X) каждый сдаваемый на факультете экзамен (значение из Z) сдается *одним и тем же списком* абитуриентов (набор значений из Y). Именно наличие

этой зависимости не позволяет независимо вставлять и удалять кортежи. Кортежи обязаны вставляться и удаляться одновременно *целыми наборами*.

В отношении "Абитуриенты-Факультеты-Предметы" имеется именно нетривиальная многозначная зависимость Факультет $\rightarrow\rightarrow$ Абитуриент|Предмет.

Определение . Отношение R находится в ***четвертой нормальной форме*** (***4НФ***) тогда и только тогда, когда отношение находится в НФБК и *не содержит нетривиальных многозначных зависимостей*.

Отношение "Абитуриенты-Факультеты-Предметы" находится в НФБК, но не в 4НФ. Это отношение можно без потерь декомпозировать на отношения:

Факультет	Абитуриент
Математический	Иванов
Физический	Иванов
Математический	Петров

Отношение "Факультеты-Абитуриенты"

Факультет	Предмет
Математический	Математика
Математический	Информатика
Физический	Математика
Физический	Физика

Отношение "Факультеты-Предметы"

В полученных отношениях устранены аномалии вставки и удаления, характерные для отношения "Абитуриенты-Факультеты-Предметы".

Тема 5. Проектирование БД с использованием семантических моделей

Широкое распространение реляционных СУБД и их использование в самых разнообразных приложениях показывает, что реляционная модель данных достаточна для моделирования разнообразных предметных областей. Однако проектирование реляционной базы данных в терминах отношений на основе кратко рассмотренного нами в двух предыдущих лекциях механизма нормализации часто представляет собой очень сложный и неудобный для проектировщика процесс.

Ограниченность реляционной модели при проектировании баз данных

При использовании в проектировании ограниченность реляционной модели проявляется в следующих аспектах.

- Модель не обеспечивает достаточных средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика. В частности, это относится к отмечавшейся нами ранее проблеме представления ограничений целостности, выходящих за пределы ограничений первичного и внешнего ключа.
- Во многих прикладных областях трудно моделировать предметную область на основе плоских таблиц40). В ряде случаев на самой начальной стадии проектирования дизайнеру приходится нелегко, поскольку от него требуется описать предметную область в виде одной (возможно, даже ненормализованной) таблицы.
- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет какие-либо формализованные средства для представления этих зависимостей.
- Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области («сущностей») и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо механизма для разделения сущностей и связей.

Семантические модели данных

Потребность проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области вызвала к жизни направление семантических моделей данных. Хотя любая развитая семантическая модель данных, как и реляционная модель, включает структурную, манипуляционную и целостную части, главным назначением семантических моделей является обеспечение возможности выражения семантики данных.

Прежде чем мы коротко рассмотрим особенности двух распространенных семантических моделей, остановимся на возможных областях их применения. Чаще всего на практике семантическое моделирование используется на первой стадии проектирования базы данных. При этом в терминах семантической модели производится концептуальная схема базы данных, которая затем вручную преобразуется к реляционной (или

какой-либо другой) схеме. Этот процесс выполняется под управлением методик, в которых достаточно четко оговорены все этапы такого преобразования. Основным достоинством данного подхода является отсутствие потребности в дополнительных программных средствах, поддерживающих семантическое моделирование. Требуется только знание основ выбранной семантической модели и правил преобразования концептуальной схемы в реляционную схему.

Следует заметить, что многие начинающие проектировщики баз данных недооценивают важность семантического моделирования вручную. Зачастую это воспринимается как дополнительная и излишняя работа. Эта точка зрения абсолютно неверна. Во-первых, построение мощной и наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на стадии проектирования схемы реляционной БД. Во-вторых, на этапе семантического моделирования производится важная документация (хотя бы в виде вручную нарисованных диаграмм и комментариев к ним), которая может оказаться очень полезной не только при проектировании схемы реляционной БД, но и при эксплуатации, сопровождении и развитии уже заполненной БД.

Неоднократно приходилось и приходится наблюдать ситуации, в которых отсутствие такого рода документации существенно затрудняет внесение даже небольших изменений в схему существующей реляционной БД. Конечно, это относится к случаям, когда проектируемая БД содержит не слишком малое число таблиц. Скорее всего, без семантического моделирования можно обойтись, если число таблиц не превышает десяти, но оно совершенно необходимо, если БД включает более сотни таблиц. Для справедливости заметим, что процедура создания концептуальной схемы вручную с ее последующим преобразованием в реляционную схему БД затруднительна в случае больших БД (содержащих несколько сотен таблиц). Причины, по всей видимости, не требуют пояснений.

История систем автоматизации проектирования баз данных (CASE-средств) началась с автоматизации процесса рисования диаграмм, проверки их формальной корректности, обеспечения средств долговременного хранения диаграмм и другой проектной документации. Конечно, компьютерная поддержка работы с диаграммами для проектировщика БД очень полезна. Наличие электронного архива проектной документации помогает при эксплуатации, администрировании и сопровождении базы данных. Но система, которая ограничивается поддержкой рисования диаграмм, проверкой их корректности и хранением, напоминает текстовый редактор, поддерживающий ввод, редактирование и проверку синтаксической корректности конструкций некоторого языка программирования, но существующий отдельно от компилятора. Кажется естественным желание расширить такой редактор функциями компилятора, и это действительно возможно, поскольку известна техника компиляции конструкций языка программирования в коды целевого компьютера. Но коль скоро имеется четкая методика преобразования концептуальной схемы БД в реляционную схему, то почему бы не выполнить программную реализацию соответствующего «компилятора» и не включить ее в состав системы проектирования баз данных?

Эта идея, естественно, показалась разумной производителям CASE-средств проектирования БД. Подавляющее большинство подобных систем, представленных на рынке, обеспечивает автоматизированное преобразование диаграммных концептуальных схем баз данных, представленных в той или иной семантической модели данных, в реляционные схемы, специфицированные чаще всего на языке SQL. У читателя может возникнуть вопрос, почему в предыдущем предложении говорится про «автоматизированное», а не про «автоматическое» преобразование? Все дело в том, что в типичной схеме SQL-ориентированной БД могут содержаться определения многих объектов (ограничений целостности общего вида, триггеров и хранимых процедур и т. д.), которые невозможно сгенерировать автоматически на основе концептуальной схемы. Поэтому на завершающем этапе проектирования реляционной схемы снова требуется ручная работа проектировщика.

Еще раз обратите внимание на то, какой ход рассуждений привел нас к выводу о возможности автоматизации процесса преобразования концептуальной схемы БД в реляционную схему. Если создатели семантической модели данных предоставляют методику преобразования концептуальных схем в реляционные, то почему бы не реализовать программу, которая производит те же преобразования, следуя той же методике? Зададимся теперь другим, но, по существу, схожим вопросом. Если создатели семантической модели данных предоставляют язык (например, диаграммный), используя который проектировщики БД на основе исходной информации о предметной области могут сформировать концептуальную схему БД, то почему бы не реализовать программу, которая сама генерирует концептуальную схему БД в соответствующей семантической модели, используя исходную информацию о предметной области? Хотя нам не известны коммерческие CASE-средства проектирования БД, поддерживающие такой подход, экспериментальные системы успешно существовали. Они представляли собой интегрированные системы проектирования с автоматизированным созданием концептуальной схемы на основе интервью с экспертами предметной области и последующим преобразованием концептуальной схемы в реляционную.

Как правило, CASE-средства, автоматизирующие преобразование концептуальной схемы БД в реляционную, производят реляционную схему базы данных в третьей нормальной форме. Нормализация более высокого уровня усложняет программную реализацию и редко требуется на практике.

Наконец, третья возможность, которую следует упомянуть, хотя она еще не вышла (или только выходит, а может быть, так никогда и не выйдет) за пределы исследовательских и экспериментальных проектов, – это работа с базой данных в семантической модели, т.е. СУБД, основанные на семантических моделях данных. При этом снова рассматриваются два варианта: обеспечение пользовательского интерфейса на основе семантической модели данных с автоматическим отображением конструкций этого интерфейса в реляционную модель данных (это задача примерно того же уровня сложности, что и автоматическая компиляция концептуальной схемы базы данных в реляционную схему) и прямая реализация СУБД, основанная на какой-либо семантической модели данных. Многие авторитетные специалисты полагают, что ближе всего ко второму подходу объектно-ориентированные СУБД, чьи модели данных по многим параметрам близки к семантическим моделям (хотя в некоторых аспектах они более мощны, а в некоторых – более слабы).

Далее переходим к использованию терминов таблица, строка и столбец вместо строгих реляционных терминов отношение, атрибут и таблица, поскольку здесь под «реляционными» базами данных понимаются, главным образом, SQL-ориентированные базы данных, для которых эта упрощенная терминология более естественна.

Многие сторонники реляционного подхода считают отсутствие раздельного представления сущностей и связей преимуществом реляционной модели данных, мотивируя это тем, что зачастую то, что вчера считалось сущностью, сегодня разумнее принять за связь, и наоборот. Это, безусловно, верно с точки зрения поддержки и модификации существующих реляционных баз данных, но отнюдь не так с точки зрения проектирования базы данных.

Издавна существует отдельный класс программных систем, предназначенных для автоматизации проектирования новых продуктов в разных областях промышленности – автомобилестроении, аэрокосмической промышленности, электронной промышленности и т.д. Очевидно, что процесс проектирования автомобиля принципиально отличается от процесса проектирования микропроцессора, но, тем не менее, для обозначения любой Системы Автоматизации Проектирования используется собирательный термин САПР (CAD – Computer Aided Design). Это оправдывается тем, что разные подклассы САПР имеют гораздо больше общих черт, чем различий. Так вот, система автоматизации проектирования БД по своему назначению и строению в большей степени является системой класса САПР, чем системой класса CASE (Computer Aided Software Engineering). По всей видимости, средства автоматизированной поддержки проектирования баз данных стали в свое время называть CASE-средствами, поскольку они обычно включали не только инструменты для поддержки проектирования, но и инструменты, поддерживающие проектирование и разработку приложений баз данных. В последние годы такие инструменты все реже производятся в виде одного пакета, и сам термин «CASE-средство» почти вышел из употребления. Тем не менее, поскольку не появилось какое-либо другое собирательное название средств поддержки проектирвания баз данных, мы будем продолжать использовать именно этот термин.

Семантическая модель Entity-Relationship (Сущность-Связь)

В этой лекции мы кратко рассмотрим некоторые черты одной из наиболее популярных семантических моделей данных – модели «Сущность-Связь» (часто ее называют кратко ER-моделью от Entity-Relationship).

Здесь следует сделать два замечания, касающиеся, главным образом, терминологии. Оба термина relation и relationship могут быть переведены на русский язык как отношение. Поэтому в русскоязычной литературе ER-модель иногда называют моделью сущность-отношение, а иногда и реляционной семантической моделью. Наверное, в этом нет ничего страшного, если говорить о ER-модели в отрыве от тематики проектирования реляционных баз данных.

Но если требуется одновременно использовать термины ER-модели и реляционной модели данных, то, безусловно, требуется применять для терминов relation и relationship разные русские эквиваленты. За этими терминами стоят весьма различные понятия. В реляционной модели отношение (relation) – это единственная родовая структура данных. С помощью этого же механизма представляются «связанные» сущности (вспомните, например, про внешние ключи). Как мы увидим немного позже, в ER-модели для представления схемы базы данных используются два равноправных понятия – сущность и связь. Связи в ER-модели играют роль, отличную от той, какую играют отношения в реляционной модели данных.

Кроме того, в русскоязычную терминологию вошла и чистая транслитерация термина relation именно в смысле отношение. Мы говорим, например, про реляционную модель данных, реляционную алгебру и т. д., понимая модель данных, основанную на отношениях, алгебру отношений и т. п. По этому поводу, по крайней мере, в контексте баз данных, разумно окончательно зарезервировать термины relation и отношение для обозначения понятий реляционной модели данных, а для термина relationship использовать другой допустимый русскоязычный эквивалент – связь.

На использовании разных вариантов ER-модели основано большинство современных подходов к проектированию баз данных (главным образом, реляционных). Модель была предложена Питером Ченом (Peter Chen) в 1976г. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. Простота и наглядность представления концептуальных схем баз данных в ER-модели привели к ее широкому распространению в CASE-системах, поддерживающих автоматизированное проектирование реляционных баз данных. Среди множества разновидностей ER-моделей одна из наиболее популярных и развитых применялась в системе CASE компании Oracle. Мы обсудим некоторый упрощенный вариант этой модели. Если говорить более точно, сосредоточимся на ее структурной и целостной частях.

Основные понятия ER-модели

Основными понятиями ER-модели являются сущность, связь и атрибут. Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступной.43) В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности – это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных экземпляров этого типа.

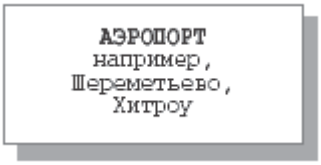


Рис. 5.1. Пример типа сущности

На рис. 5.1 изображена сущность АЭРОПОРТ с примерными экземплярами «Шереметьево» и «Хитроу». Эта примитивная диаграмма тем не менее несет важную информацию. Во-первых, она показывает, что в базе данных будут содержаться однотипные структуры данных (экземпляры сущности), описывающие аэропорты. Во-вторых, поскольку в жизни существует несколько точек зрения на аэропорты (например, точка зрения пилота, точка зрения пассажира, точка зрения администратора) и этим точкам зрения соответствуют разные структуры данных, то приведенные примеры аэропортов позволяют несколько сузить допустимый набор точек зрения. В нашем случае приведены примеры международных аэропортов, так что, скорее всего, имеется точка зрения пассажира или пилота международных авиарейсов.

При определении типа сущности необходимо гарантировать, что каждый экземпляр сущности может быть отличим от любого другого экземпляра той же сущности. Это требование в некотором роде аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах.

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя типами сущностей. Как и сущность, связь – это типовое понятие, все экземпляры обоих связываемых типов сущностей подчиняются устанавливаемым правилам связывания. Поэтому правильнее говорить о типе связи, устанавливаемой между типами сущности, и об экземплярах типа связи, устанавливаемых между экземплярами типа сущности.45) В обсуждаемом здесь варианте ER-модели эта ассоциация всегда является бинарной и может существовать между двумя разными типами сущностей или между типом сущности и им же самим (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указываются имя конца связи, степень конца связи (сколько экземпляров данного типа сущности должно присутствовать в каждом экземпляре данного типа связи), обязательность связи (т. е. любой ли экземпляр данного типа сущности должен участвовать в некотором экземпляре данного типа связи).46)

Связь представляется в виде ненаправленной линии, соединяющей две сущности или ведущей от сущности к ней же самой. При этом в месте «стыковки» связи с сущностью используются:

- трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут (или должны) использоваться много (many) экземпляров сущности;

одноточечный вход, если в связи может (или должен) участвовать только один экземпляр сущности.

Обязательный конец связи изображается сплошной линией, а необязательный – прерывистой линией.

Связь между сущностями БИЛЕТ и ПАССАЖИР, показанная на рис. 5.2, связывает билеты и пассажиров. Конец связи с именем «для» позволяет связывать с одним пассажиром более одного билета, причем каждый билет должен быть связан с каким-либо пассажиром. Конец связи с именем «имеет» показывает, что каждый билет может принадлежать только одному пассажиру, причем пассажир не обязан иметь хотя бы один билет.

Рис. 5.2. Пример типа связи

Лаконичная устная трактовка изображенной диаграммы состоит в следующем:

- каждый БИЛЕТ предназначен для одного и только одного ПАССАЖИРА;
- каждый ПАССАЖИР может иметь один или более БИЛЕТОВ.

На следующем примере (рис. 5.3) изображена рекурсивная связь, связывающая сущность МУЖЧИНА с ней же самой. Конец связи с именем «сын» определяет тот факт, что несколько людей могут быть сыновьями одного отца. Конец связи с именем «отец» означает, что не у каждого мужчины должны быть сыновья.

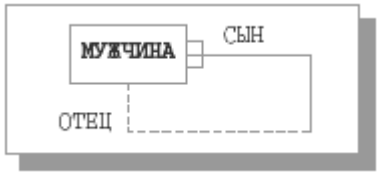


Рис. 5.3. Пример рекурсивного типа связи

Лаконичная устная трактовка изображенной диаграммы состоит в следующем:

- каждый МУЖЧИНА является сыном одного и только одного МУЖЧИНЫ;
- каждый МУЖЧИНА может являться отцом одного или более МУЖЧИН.

Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами.

Пример типа сущности ЧЕЛОВЕК с указанными атрибутами показан на рис. 5.4. С технической точки зрения атрибуты типа сущности в ER-модели похожи на атрибуты отношения в реляционной модели данных. И в том, и в другом случаях введение именованных атрибутов вводит некоторую типовую структуру данных, имя которой совпадает с именем типа сущности в случае ER-модели или с именем переменной отношения в случае реляционной модели. Этой типовой структуре должны следовать все экземпляры типа сущности или все кортежи отношения. Но имеется и важное отличие. Напомним, что в реляционной модели данных атрибут определяется как упорядоченная пара <имя_атрибута, имя_домена> (или <имя_атрибута, имя_базового_типа_данных>, если понятие домена не поддерживается). Заголовок отношения, определяемый как множество таких пар, представляет собой полный аналог структурного типа данных в языках программирования.

Рис. 5.4. Пример типа сущности с атрибутами

При определении атрибутов типа сущности в ER-модели указание домена атрибута не является обязательным, хотя это и возможно (см. ниже). Обсудим, чем вызвана эта возможность «ослабленного» определения атрибутов. Прежде всего, как уже отмечалось, семантические модели данных используются для построения концептуальных схем БД, и эти схемы преобразуются в реляционные схемы БД, которые поддерживаются той или иной СУБД. Несмотря на то, что в настоящее время типовые возможности РСУБД в основном стандартизованы (на основе стандарта языка SQL), детали базового набора типов данных и средств определения доменов в разных системах могут различаться. Поскольку производители CASE-средств проектирования реляционных БД стремятся не связывать обеспечиваемые ими возможности семантического моделирования с конкретной реализацией СУБД, они стимулируют откладывание строгого определения типов атрибутов до стадии полного определения реляционной схемы.

Кроме того, напомним, что при определении атрибута отношения допускается использование имен атрибутов, совпадающих с именами своих доменов (это два разных пространства имен, и наличие одинаковых имен у атрибутов и доменов не вызывает коллизий). Поэтому при определении атрибутов типов сущности можно так подбирать их имена, что они в дальнейшем будут подсказывать, какие домены у этих атрибутов имеются в виду. Пониманию предполагаемой сути доменов способствует и возможность указания

примеров значений атрибутов. Например, на рис. 5.4 имеется атрибут год рождения, в качестве примерного значения которого указано «1976». Это подсказывает, что в реляционной схеме при определении соответствующего атрибута наиболее естественным базовым типом данных будет темпоральный тип «ДАТА», значения которого задают дату с точностью до года.

Уникальные идентификаторы типов сущности

Как отмечалось выше, при определении типа сущности необходимо гарантировать, что каждый экземпляр сущности является отличимым от любого другого экземпляра той же сущности. Поскольку сущность является абстракцией реального или представляемого объекта внешнего мира, это требование нужно иметь в виду уже при выборе кандидата в типы сущности. Например, предположим, что проектируется база данных для поддержки работы книжного склада. На складе могут храниться произвольные части тиража любого издания любой книги. Может ли в этом случае индивидуальная книга являться прообразом типа сущности? Утверждается, что нет, поскольку отсутствует возможность различения книг одного издания. Для книжного склада прообразом типа сущности будет набор одноименных книг одного автора, вышедших в одном издании. Одним из атрибутов этого типа сущности будет число книг в наборе. Но когда книга поступает в библиотеку и ей присваивается уникальный библиотечный номер, она становится разумным прообразом типа сущности. Плохо устроены библиотеки, в которых не различаются индивидуальные книги (даже одноименные книги одного автора, вышедшие в одном издании).

Но при проектировании базы данных мало того, чтобы проектировщик убедился в правильном выборе типов сущности, гарантирующем различие экземпляров каждого типа сущности. Необходимо сообщить системе автоматизации проектирования БД, каким образом будут различаться эти экземпляры, т. е. сообщить, как конструируются уникальные идентификаторы экземпляров каждого типа сущности. В ER-модели у экземпляра типа сущности не может быть назначаемого пользователем имени или назначаемого системой внешнего уникального идентификатора. Экземпляр типа сущности может идентифицироваться только своими индивидуальными характеристиками, а они представляются значениями атрибутов и экземплярами типов связи, связывающими данный экземпляр типа сущности с экземплярами других типов сущности или этого же типа сущности. Поэтому уникальным идентификатором сущности может быть атрибут, комбинация атрибутов, связь, комбинация связей или комбинация связей и атрибутов, уникально отличающая любой экземпляр сущности от других экземпляров сущности того же типа.

Приведем несколько примеров. На рис. 5.5 показан тип сущности КНИГА, пригодный для использования в базе данных книжного склада. При издании любой книги в любом издательстве (кроме пиратских, которыми мы для простоты пренебрежем) ей присваивается уникальный номер – ISBN. Понятно, что значение атрибута isbn будет уникально идентифицировать партию книг на складе. Кроме того, конечно, в качестве уникального идентификатора годится и комбинация атрибутов <автор, название, номер издания, издательство, год издания>.

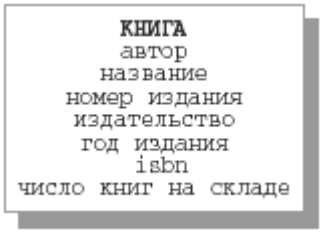


Рис. 5.5. Тип сущности, экземпляры которого идентифицируются атрибутами

На рис. 10.6 диаграмма включает два связанных типа сущности. У каждого взрослого человека имеется один и только один паспорт (мы снова не берем в расчет особый случай, когда у одного человека имеется несколько паспортов), и каждый паспорт может принадлежать только одному взрослому человеку (некоторые уже готовые паспорта могут быть еще никому не выданы). Тогда связь человека с его паспортом (конец связи ИМЕЕТ) уникально идентифицирует взрослого человека, т. е., грубо говоря, паспорт определяет взрослого человека. Поскольку могут существовать паспорта, еще не выданные какому-либо человеку, эта связь не является уникальным идентификатором сущности ПАСПОРТ.

Рис. 5.6. Тип сущности, экземпляры которого идентифицируются связью

На рис. 5.7 диаграмма включает три связанных типа сущности. Профессора обладают знаниями в нескольких учебных дисциплинах. Преподавание каждой дисциплины доступно нескольким профессорам. Другими словами, между сущностями ПРОФЕССОР и ДИСЦИПЛИНА определена связь «многие ко

многим». Каждый профессор может готовить курсы по любой доступной ему дисциплине. Каждой дисциплине может быть посвящено несколько учебных курсов. Но каждый профессор может готовить только один курс по любой доступной ему дисциплине, и каждый курс может быть посвящен только одной дисциплине. Тем самым, каждый экземпляр типа сущности КУРС уникально идентифицируется экземпляром сущности ПРОФЕССОР и экземпляром сущности ДИСЦИПЛИНА, т. е. парой связей с именами концов ГОТОВИТСЯ и ПОСВЯЩЕН на стороне сущности КУРС. Заметим, что сущности ПРОФЕССОР и ДИСЦИПЛИНА связями не идентифицируются.

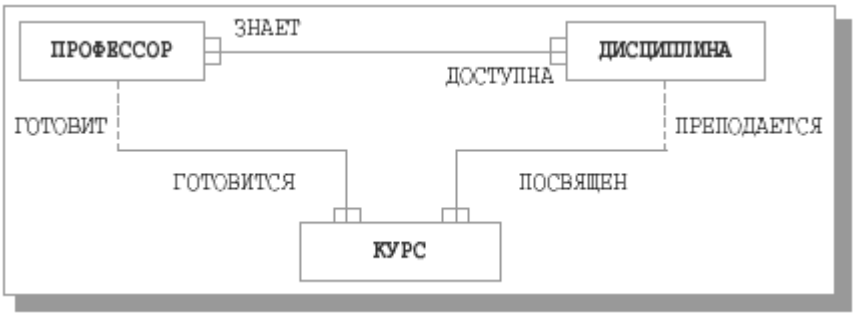


Рис. 5.7. Тип сущности, экземпляры которого идентифицируются комбинацией связей

Наконец, на рис. 5.8 приведен пример типа сущности, уникальный идентификатор которого является комбинацией атрибутов и связей. Это несколько уточненный вариант сущности с рекурсивной связью с рис. 5.3. У каждого человека могут быть дети, и у каждого человека имеется отец. Тогда, если предположить, что близнецам, появившимся на свет одновременно, не дают одинаковых имен, то уникальным идентификатором типа сущности ЧЕЛОВЕК может быть комбинация атрибутов <дата рождения, ФИО> и связь с именем конца РЕБЕНОК.

Рис.5.8. Тип сущности, экземпляры которого идентифицируются комбинацией атрибутов и связей

Понятно, что это «определение» на самом деле является тавтологией, поскольку, во-первых, мы пытаемся определить термин сущность через не определенный термин объект, а во-вторых, попытки определения термина объект настолько же безнадежны. Обычно авторы пытаются оправдываться тем, что в подобном контексте они имеют в виду «житейское», а не сколько-нибудь формализованное понятие объекта. Конечно, от этого не становится легче, поскольку понятие сущности должно пониматься в достаточно точном смысле. Но эта тавтология не изобретена автором этого курса; она традиционна для области семантического моделирования. В этой области стремятся максимально избегать формальностей.

Хотя было бы правильнее всегда использовать термины тип сущности и экземпляр типа сущности, для избежания многословности (и следуя традиции) в тех случаях, где это не приводит к двусмысленности, мы будем использовать термин сущность в значении типа сущности.

Тем не менее, как и в случае типа сущности, мы будем часто использовать термин связь в значении типа связи.

В некоторых вариантах ER-модели конец связи называют ролью связи в данной сущности. Тогда можно говорить об имени роли, степени роли и обязательности роли связи в данной сущности.

Нормальные формы ER-диаграмм

Как и в случае схем реляционных баз данных, для ER-диаграмм вводится понятие нормальных форм, причем их смысл очень близко соответствует смыслу нормальных форм отношений. Заметим, что определения нормальных форм ER-диаграмм делают более понятным смысл нормализации схем отношений. Мы приведем только очень краткие и неформальные определения трех первых нормальных форм. Конечно, можно было бы ввести дальнейшие нормальные формы ER-диаграмм, аналогичные нормальной форме Бойса-Кодда, 4NF и 5NF, но на практике к такой нормализации обычно не прибегают, а общие идеи после ознакомления с лекцией 9 должны быть понятны и так.

Первая нормальная форма ER-диаграммы

В первой нормальной форме ER-диаграммы устраняются атрибуты, содержащие множественные значения, т. е. производится выявление неявных сущностей, «замаскированных» под атрибуты.

На рис. 5.9 (а) показана диаграмма, в которой тип сущности АЭРОДРОМ не удовлетворяет требованию первой нормальной формы. Здесь для нас несущественны атрибуты сущности АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ, но сущность АЭРОДРОМ помимо атрибутов, отражающих собственные характеристики аэродромов (длина взлетно-посадочной полосы, число ангаров и т.д.) содержит атрибут, множественное значение которого характеризует самолеты, приписанные к этому аэродрому. Очевидно, что самолеты нуждаются в ремонте, т. е. должны обслуживаться некоторым авиаремонтным предприятием. Но поскольку самолеты являются частью сущности АЭРОДРОМ, единственным способом фиксации этого факта на диаграмме является проведение связи «многие ко многим» между типами сущности АЭРОДРОМ и АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ. Таким образом выражается то соображение, что для ремонта разных самолетов, приписанных к одному аэродрому, могут использоваться разные транспортные предприятия, и каждое транспортное предприятие может обслуживать несколько аэродромов.

Рис. 5.9. Пример приведения ER-диаграммы к первой нормальной форме

Чем плоха эта ситуация? Прежде всего, тем, что скрывается тот факт, что авиаремонтное предприятие ремонтирует самолеты, а не аэродромы. Наша же связь на самом деле означает, что любой аэродром из группы аэродромов обслуживается любым авиаремонтным предприятием из группы таких предприятий. Проблема состоит именно в том, что значением атрибута «самолеты» является множество экземпляров типа сущности САМОЛЕТ, и этот тип сущности сам обладает атрибутами и связями.

Ситуацию исправляет ER-диаграмма, показанная на рис. 5.9 (b). Здесь мы выделили тип сущности САМОЛЕТ. Связь между сущностями АЭРОПОРТ и САМОЛЕТ показывает, что к одному аэродрому приписывается несколько самолетов. Связь между сущностями САМОЛЕТ и АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ означает, что каждый самолет из группы самолетов (группу самолетов могут составлять, например, все самолеты одного типа) обслуживается любым транспортным предприятием из некоторой группы таких предприятий. ER-диаграмма на рис. 10.9 (b) находится в первой нормальной форме и, как мы видим, лучше отображает реальную ситуацию.

Вторая нормальная форма ER-диаграммы

Во второй нормальной форме устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность.

На рис. 5.10 (а) показана диаграмма, на которой тип сущности ЭЛЕМЕНТ РАСПИСАНИЯ не удовлетворяет требованиям второй нормальной формы. На этой диаграмме у сущности ЭЛЕМЕНТ РАСПИСАНИЯ имеются следующие свойства. Элементы расписания предназначены для сохранения данных о рейсах самолетов, вылетающих в течение дня. Некоторыми важными характеристиками рейса являются номер рейса, аэропорт вылета, аэропорт назначения, дата и время вылета, бортовой номер самолета, тип самолета. Если говорить про российские авиационные компании, то (1) у каждого рейса имеется заранее приписанный ему номер (уникальный среди всех других имеющихся номеров рейсов), (2) не все рейсы совершаются каждый день, поэтому характеристикой конкретного рейса является дата и время его совершения, (3) бортовой номер самолета определяется парой <номер рейса, дата-время вылета>. Имеется связь «многие к одному» между сущностями ЭЛЕМЕНТ РАСПИСАНИЯ и ГОРОД. Экземпляры типа сущности ГОРОД характеризуют город, в который прибывает данный рейс.

Рис. 5.10. Пример приведения ER-диаграммы ко второй нормальной форме

Уникальным идентификатором типа сущности ЭЛЕМЕНТ РАСПИСАНИЯ является пара атрибутов <номер рейса, дата-время вылета>. Если вернуться к терминам функциональных зависимостей, то между атрибутами этой сущности имеются следующие FD:

- {номер рейса, дата-время вылета} ⟶ бортовой номер самолета;
- номер рейса ⟶ аэропорт вылета;
- номер рейса ⟶ аэропорт назначения;
- бортовой номер самолета ⟶ тип самолета.

Кроме того, очевидно, что каждый экземпляр связи с сущностью ГОРОД также определяется значением атрибута номер рейса. Налицо нарушение требования второй нормальной формы. Мы получаем не только избыточное хранение значений атрибутов аэропорт вылета и аэропорт назначения в каждом экземпляре типа сущности ЭЛЕМЕНТ РАСПИСАНИЯ с одним и тем же значением номера рейса. Искажается и затемняется смысл связи с сущностью ГОРОД. Можно подумать, что в разные дни один и тот же рейс прибывает в разные города.

На рис. 10.10 (b) показан нормализованный вариант диаграммы, в котором все сущности находятся во второй нормальной форме. Теперь имеются три типа сущности: РЕЙС с атрибутами номер рейса, аэропорт вылета, аэропорт назначения, ЭЛЕМЕНТ РАСПИСАНИЯ с атрибутами дата-время вылета, бортовой номер самолета, тип самолета и ГОРОД. Уникальным идентификатором сущности РЕЙС является атрибут номер рейса, уникальный идентификатор ЭЛЕМЕНТ РАСПИСАНИЯ состоит из атрибута дата вылета и конца связи КОГДА, НА ЧЕМ. Мы видим, что ни в одном типе сущности больше нет атрибутов, определяемых частью уникального идентификатора. Свойства второй нормальной формы удовлетворяются, и мы имеем более качественную диаграмму.

Третья нормальная форма ER-диаграммы

- В третьей нормальной форме устраняются атрибуты, которые зависят от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.

Взглянем еще раз на тип сущности ЭЛЕМЕНТ РАСПИСАНИЯ на рис. 5.10 (b). Конечно, каждый день каждый рейс выполняется только одним самолетом, поэтому бортовой номер самолета полностью зависит от уникального идентификатора. Но бортовой номер является уникальной характеристикой каждого самолета, и от этой характеристики зависят все остальные характеристики, в частности, тип самолета. Другими словами, между уникальным идентификатором и другими атрибутами типа сущности ЭЛЕМЕНТ РАСПИСАНИЯ имеются следующие функциональные зависимости:

{КОГДА, НА ЧЕМ, дата-время вылета} ⟶ бортовой номер самолета
{КОГДА, НА ЧЕМ, дата-время вылета} ⟶ тип самолета
бортовой номер самолета ⟶ тип самолета

Как видно, имеется транзитивная FD {КОГДА, НА ЧЕМ, дата вылета} ⟶ тип самолета, и наличие этой FD вызывает нарушение требования третьей нормальной формы. На самом деле, тип сущности ЭЛЕМЕНТ РАСПИСАНИЯ на рис. 5.10 (b) включает в себя (по крайней мере, частично) тип сущности САМОЛЕТ. Это вызывает избыточность хранения и затуманивает смысл диаграммы. На рис. 5.11 показан нормализованный вариант диаграммы, в котором все сущности находятся в третьей нормальной форме.

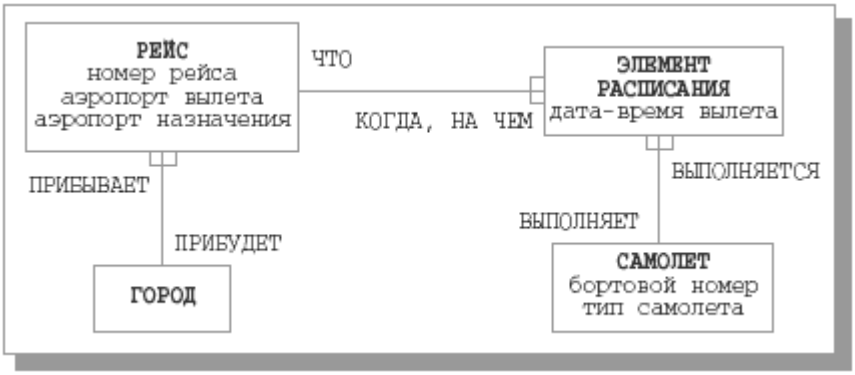


Рис. 5.11. Пример приведения ER-диаграммы к третьей нормальной форме

Более сложные элементы ER-модели

До сих пор мы рассматривали только самые основные и наиболее очевидные понятия ER-модели данных. К числу некоторых более сложных элементов модели относятся следующие.

- Подтипы и супертипы сущностей. Подобно тому как это делается в языках программирования с развитыми типовыми системами (например, в языках объектно-ориентированного программирования), в ER-модели поддерживается возможность определения нового типа сущности путем наследования некоторого супертипа сущности. Механизм наследования в ER-модели обладает несколькими особенностями: в частности, интересные нюансы связаны с необходимостью графического изображения этого механизма (см. ниже).
- Уточняемые степени связи. Иногда бывает полезно определить возможное количество экземпляров сущности, участвующих в данной связи (например, ввести ограничение, связанное с тем, что служащему разрешается участвовать не более чем в трех проектах одновременно). Для выражения этого семантического ограничения разрешается указывать на конце связи ее максимально допустимую или обязательную степень.
- Взаимно исключающие связи. Для заданного типа сущности можно определить такой набор типов связи с другими типами сущности, что для каждого экземпляра заданного типа сущности может (если набор связей является необязательным) или должен (если набор связей обязателен) существовать экземпляр только одной связи из этого набора.
- Каскадные удаления экземпляров сущностей. Некоторые связи бывают настолько сильными (конечно, в случае связи «один ко многим»), что при удалении опорного экземпляра сущности (соответствующего концу связи «один») нужно удалить и все экземпляры сущности, соответствующие концу связи «многие». Соответствующее требование каскадного удаления можно специфицировать при определении связи.
- Домены. Как и в случае реляционной модели данных, в некоторых случаях полезна возможность определения потенциально допустимого множества значений атрибута сущности (домена).

Эти и другие усложненные элементы модели данных «Сущность-Связь» делают ее более мощной, но одновременно несколько затрудняют ее использование. Конечно, при реальном применении ER-диаграмм для проектирования баз данных необходимо ознакомиться со всеми возможностями. Ниже мы подробнее обсудим два элемента из числа упомянутых выше – супертипы и подтипы сущности, а также приведем пример сущности с взаимно исключающими связями.

Наследование типов сущности и типов связи

Сущность может быть расщеплена на два или большее число взаимно исключающих подтипов, каждый из которых включает общие атрибуты и/или связи. Эти общие атрибуты и/или связи явно определяются один раз на более высоком уровне. В подтипах могут определяться собственные атрибуты и/или связи. В принципе, подтипизация может продолжаться на более низких уровнях, но опыт использования ER-модели при проектировании баз данных показывает, что в большинстве случаев оказывается достаточно двух-трех уровней.

Если у типа сущности A имеются подтипы B1, B2,..., Bn, то:

- любой экземпляр типа сущности B1, B2,..., Bn является экземпляром типа сущности A (включение);
- если a является экземпляром типа сущности A, то a является экземпляром некоторого подтипа сущности Bi (i = 1, 2, ..., n) (отсутствие собственных экземпляров у супертипа сущности);

- ни для каких подтипов Bi и Bj (i, j = 1, 2, ..., n) не существует экземпляра, типом которого одновременно являются типы сущности Bi и Bj (разъединенность подтипов).

Тип сущности, на основе которого определяются подтипы, называется супертипом. Как мы видели выше, подтипы должны образовывать полное множество, т. е. любой экземпляр супертипа должен относиться к некоторому подтипу. Иногда для обеспечения такой полноты приходится определять дополнительный подтип ПРОЧИЕ.

Пример супертипа ЛЕТАТЕЛЬНЫЙ АППАРАТ и его подтипов АЭРОПЛАН, ВЕРТОЛЕТ, ПТИЦЕЛЕТ и ПРОЧИЕ показан на рис. 5.12. У подтипа АЭРОПЛАН имеются два собственных подтипа – ПЛАНЕР и МОТОРНЫЙ САМОЛЕТ. Для супертипа сущности ЛЕТАТЕЛЬНЫЙ АППАРАТ определен атрибут максимальная дальность полета и необязательная связь «многие ко многим» с типом сущности ПИЛОТ. Эти атрибут и связь наследуется всеми подтипами этого супертипа сущности. У непосредственного подтипа сущности АЭРОПЛАН определяется один дополнительный атрибут, так что в совокупности у данного типа сущности имеются два атрибута максимальная дальность полета и размах крыльев и одна унаследованная связь с типом сущности ПИЛОТ. У подтипа второго уровня МОТОРНЫЙ САМОЛЕТ супертипа АЭРОПЛАН определяется один дополнительный атрибут мощность мотора и одна дополнительная (обязательная) связь с типом сущности АЭРОДРОМ. Тем самым, у типа сущности МОТОРНЫЙ САМОЛЕТ имеются три атрибута: два унаследованных – максимальная дальность полета и размах крыльев и один собственный – мощность мотора, а также две связи: одна унаследованная – с типом сущности ПИЛОТ и одна собственная – с типом сущности АЭРОДРОМ. И так далее. Понятно, что для типа сущности ПРОЧИЕ, скорее всего, бессмысленно определять собственные атрибуты и связи, так что свойства этого типа будут совпадать со свойствами его супертипа.

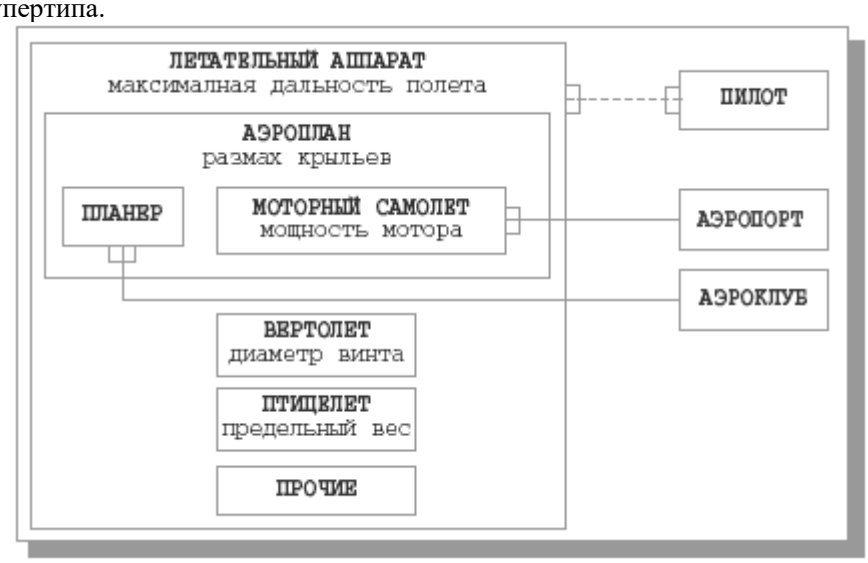


Рис. 5.12. Супертипы и подтипы сущности

Как же следует понимать диаграмму, представленную на рис. 10.12? Если начинать от супертипа, то диаграмма изображает ЛЕТАТЕЛЬНЫЙ АППАРАТ, который должен быть АЭРОПЛАНОМ, ВЕРТОЛЕТОМ, ПТИЦЕЛЕТОМ или ДРУГИМ ЛЕТАТЕЛЬНЫМ АППАРАТОМ. Если начинать от подтипа (например, сущности ВЕРТОЛЕТ), то это ВЕРТОЛЕТ, который относится к типу ЛЕТАТЕЛЬНОГО АППАРАТА. Если начинать от подтипа, который является одновременно супертипом, то это АЭРОПЛАН, который относится к типу ЛЕТАТЕЛЬНОГО АППАРАТА и должен быть ПЛАНЕРОМ или МОТОРНЫМ САМОЛЕТОМ.

В механизме наследования ER-модели допускается наличие двух или более разбиений сущности на подтипы. Например, тип сущности ЧЕЛОВЕК может быть расщеплен на подтипы по профессиональному признаку (ПРОГРАММИСТ, ДОЯРКА и т. д.), а может быть расщеплен и по половому признаку (МУЖЧИНА, ЖЕНЩИНА).

Взаимно исключающие связи

Пример диаграммы из двух сущностей с взаимно исключающими связями показан на рис. 10.13(а). Самолет может находиться в рабочем состоянии, и тогда у него имеется один и только один пилот. Или же самолет может находиться на ремонте на одном из нескольких возможных авиаремонтных предприятий (каждое предприятие может производить ремонт нескольких самолетов).

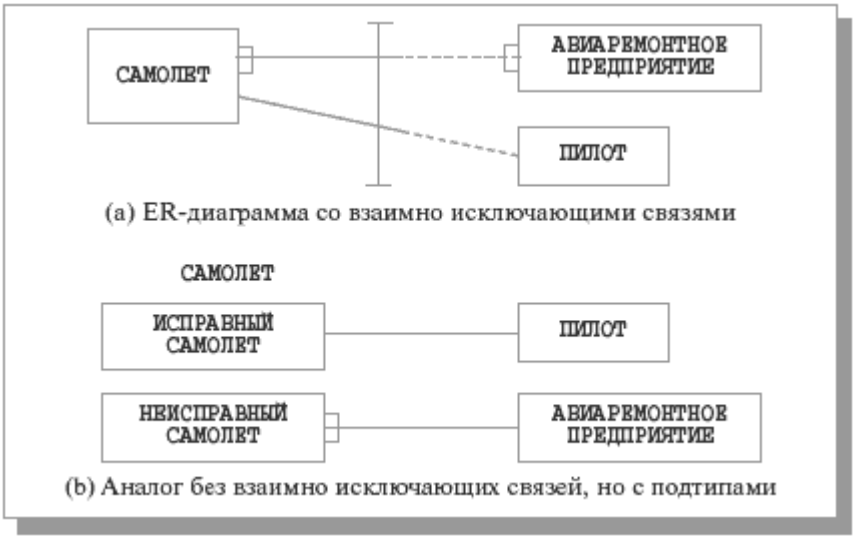


Рис. 5.13. Пример ER-диаграммы со взаимно исключающими связями

В данном случае для каждого экземпляра типа сущности САМОЛЕТ должен существовать экземпляр одной из указанных связей. Для экземпляров типа сущности САМОЛЕТ, соответствующих исправным самолетам, должен существовать экземпляр связи «один к одному» с экземпляром типа сущности ПИЛОТ, а экземпляры, соответствующие неисправным самолетам, должны участвовать в экземпляре типа связи «многие к одному» с экземпляром типа сущности АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ.

Как показано на рис. 5.13(b), диаграмма со взаимно исключающими связями из рис. 5.13(a) может быть преобразована к диаграмме без взаимно исключающих связей путем введения подтипов. Поскольку любой самолет может быть либо исправным, либо неисправным, можно корректным образом ввести два подтипа супертипа САМОЛЕТ – ИСПРАВНЫЙ САМОЛЕТ и НЕИСПРАВНЫЙ САМОЛЕТ. На уровне супертипа сущности связи не определяются. Для подтипа ИСПРАВНЫЙ САМОЛЕТ определяется обязательная связь «один к одному» с типом сущности ПИЛОТ, а для подтипа НЕИСПРАВНЫЙ САМОЛЕТ определяется обязательная связь «многие к одному» с типом сущности АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ.

Заметим, что для того чтобы описанная схема реализации механизма взаимно исключающих связей на основе механизма наследования действительно могла работать, в средствах манипулирования данными ER-модели должна быть предусмотрена возможность динамического изменения типа сущности у экземпляра. Конкретно для нашего случая требуется возможность изменения типа экземпляра сущности ИСПРАВНЫЙ САМОЛЕТ на тип сущности НЕИСПРАВНЫЙ САМОЛЕТ, и наоборот (исправный самолет может ломаться, неисправный самолет – приводиться в рабочее состояние). Конечно, при такой смене типа должен изменяться и экземпляр связи. Заметим, что в рассматриваемом случае мы имеем дело с ограниченным динамическим изменением типа экземпляра, поскольку и исправные, и неисправные самолеты являются экземплярами супертипа САМОЛЕТ.

Получение реляционной схемы из ER-диаграммы

Опишем типовую многошаговую процедуру преобразования ER-диаграммы в реляционную (более точно, в SQL-ориентированную) схему базы данных.

Базовые приемы

Каждый простой тип сущности превращается в таблицу. (Простым типом сущности называется тип сущности, не являющийся подтипом и не имеющий подтипов.) Имя сущности становится именем таблицы. Экземплярам типа сущности соответствуют строки соответствующей таблицы.

Каждый атрибут становится столбцом таблицы с тем же именем; может выбираться более точный формат представления данных. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, – не могут.

Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификаторов, для первичного ключа выбирается наиболее характерный. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно, и в общем случае может привести к зацикливанию). Для именования этих столбцов используются имена концов связей и/или имена парных типов сущностей.

Связи «многие к одному» (и «один к одному») становятся внешними ключами, т. е. образуется копия уникального идентификатора сущности на конце связи «один», и соответствующие столбцы составляют внешний ключ таблицы, соответствующей типу сущности на конце связи «многие». Необязательные связи соответствуют столбцам внешнего ключа, допускающим наличие неопределенных значений; обязательные связи – столбцам, не допускающим неопределенных значений. Если между двумя типами сущности A и B имеется связь «один к одному», то соответствующий внешний ключ по желанию проектировщика может быть объявлен как в таблице A, так и в таблице B. Чтобы отразить в определении таблицы ограничение, которое заключается в том, что степень конца связи должна равняться единице, соответствующий (возможно, составной) столбец должен быть дополнительно специфицирован как возможный ключ таблицы (в случае использования языка SQL для этого служит спецификация UNIQUE – см. лекцию 16).

Для поддержки связи «многие ко многим» между типами сущности A и B создается дополнительная таблица AB с двумя столбцами, один из которых содержит уникальные идентификаторы экземпляров сущности A, а другой – уникальные идентификаторы экземпляров сущности B. Обозначим через УИД(c) уникальный идентификатор экземпляра c некоторого типа сущности C. Тогда, если в экземпляре связи «многие ко многим» участвуют экземпляры a1, a2, ..., an типа сущности A и экземпляры b1, b2, ..., bm типа сущности B, то в таблице AB должны присутствовать все строки вида {УИД(ai), УИД(bj)} для i = 1, 2, ..., nn, j = 1, 2, ..., m. Понятно, что, используя таблицы A, B и AB, с помощью стандартных реляционных операций можно найти все пары экземпляров типов сущности, участвующих в данной связи.

Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Представление в реляционной схеме супертипов и подтипов сущности

В этом подразделе мы предполагаем, что реляционная схема базы данных проектируется в расчете на использование обычной SQL-ориентированной СУБД, не поддерживающей объектно-реляционные расширения. Кстати, заметим, что поддержка таких расширений не слишком помогает при переходе от концептуальной схемы базы данных в модели «Сущность-Связь» к объектно-реляционной схеме, соответствующей последним стандартам языка SQL.

Если в концептуальной схеме (ER-диаграмме) присутствуют подтипы, то возможны два способа их представления в реляционной схеме:

(a) собрать все подтипы в одной таблице;

(b) для каждого подтипа образовать отдельную таблицу.

При применении способа (a) таблица создается для максимального супертипа (типа сущности, не являющегося подтипом), а для подтипов могут создаваться представления. Таблица содержит столбцы, соответствующие каждому атрибуту (и связям) каждого подтипа. В таблицу добавляется, по крайней мере, один столбец, содержащий код ТИПА; он становится частью первичного ключа. Для каждой строки таблицы значение этого столбца определяет тип сущности, экземпляру которого соответствует строка. Столбцы этой строки, которые соответствуют атрибутам и связям, отсутствующим в данном типе сущности, должны содержать неопределенные значения.

При использовании метода (b) для каждого подтипа первого уровня (для более глубоких уровней применяется метод (a)) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы – столбцы супертипа).

- У каждого способа есть свои достоинства и недостатки. К достоинствам первого способа (одна таблица для супертипа и всех его подтипов) можно отнести следующее:
- соответствие логике супертипов и подтипов; поскольку любой экземпляр любого подтипа является экземпляром супертипа, логично хранить вместе все строки, соответствующие экземплярам супертипа;
- обеспечение простого доступа к экземплярам супертипа и не слишком сложный доступ к экземплярам подтипов;
- возможность обойтись небольшим числом таблиц.

Недостатки метода (a):

- прикладная программа, имеющая дело с одной таблицей супертипа, должна включать дополнительную логику работы с разными наборами столбцов (в зависимости от значения столбца ТИП) и разными ограничениями целостности (в зависимости от особенностей связей, определенных для подтипа);
- общая для всех подтипов таблица потенциально может стать узким местом при многопользовательском доступе по причине возможности блокировки таблицы целиком;
- для индивидуальных столбцов подтипов должна допускаться возможность содержать неопределенные значения; таким образом, потенциально в общей таблице будет содержаться много неопределенных значений, что при использовании некоторых РСУБД может потребовать значительного объема внешней памяти.

Достоинства метода (b) состоят в следующем:

- действуют более понятные правила работы с подтипами (каждому подтипу соответствует одноименная таблица);
- упрощается логика приложений; каждая программа работает только с нужной таблицей.

Недостатки метода (b):

- в общем случае требуется слишком много отдельных таблиц;
- работа с экземплярами супертипа на основе представления, объединяющего таблицы супертипов, может оказаться недостаточно эффективной;
- поскольку множество экземпляров супертипа является объединением множеств экземпляров подтипов, не все РСУБД могут обеспечить выполнение операций модификации экземпляров супертипа.

Представление в реляционной схеме взаимно исключающих связей

Существуют два способа формирования схемы реляционной БД при наличии взаимно исключающих связей (имеются в виду связи «один ко многим», причем конец связи «многие» находится на стороне сущности, для которой связи являются взаимно исключающими):

(a) общее хранение внешних ключей;

(b) раздельное хранение внешних ключей.

Понятно, что если имеются взаимно исключающие связи упомянутой категории, то в таблице, соответствующей сущности, для которой связи являются взаимно исключающими, необходимо хранить внешние ключи. Если внешние ключи всех потенциально связанных таблиц имеют общий формат, то можно применить способ (a), т. е. создать два столбца: идентификатор связи и идентификатор сущности (возможно, составной). Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. В столбце (столбцах) идентификатора сущности хранятся значения уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то приходится прибегать к использованию способа (b), т. е. создавать для каждой связи, покрываемой дугой исключения, явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

Преимущество подхода (a) состоит в том, что в таблице, соответствующей сущности, появляется всего два дополнительных столбца. Очевидным недостатком является усложнение выполнения операции соединения: чтобы воспользоваться для соединения одной из альтернативных связей, нужно сначала произвести ограничение таблицы в соответствии с нужным значением столбца, содержащего идентификаторы связей.

При использовании подхода (b) соединения являются явными (и естественными). Недостаток состоит в том, что требуется иметь столько столбцов, сколько имеется альтернативных связей. Кроме того, в каждом из таких столбцов будет содержаться много неопределенных значений, хранение которых потенциально может привести к серьезным накладным расходам внешней памяти.

Рис. 5.14. Возможные модификации ER-диаграмм, позволяющие избежать взаимно исключающих связей

Модификация, показанная на рис. 5.14 (b), основана на том наблюдении, что коль скоро связи являются альтернативными, то они разделяют множество экземпляров сущности А на два или более непересекающихся подмножества, которые могут лежать в основе определения подтипов А1 и А2. Это хороший вариант, если такие подтипы могут пригодиться еще для чего-нибудь. Например, в случае взаимно исключающей связи, представленной на рис. 5.12, у исправных и неисправных самолетов могут имется несовпадающие множества атрибутов (скажем, у типа сущности ИСПРАВНЫЕ САМОЛЕТЫ может иметься атрибут дата завершения гарантийного срока, а у типа сущности НЕИСПРАВНЫЕ САМОЛЕТЫ – атрибут тип неисправности). С другой стороны, как отмечалось в предыдущем разделе, для использования этого подхода требуется возможность динамического изменения типа существующего экземпляра.

Модификация, показанная на рис. 5.14 (c), основана на том наблюдении, что коль скоро типы сущности В и С участвуют в альтернативной связи, то, по всей видимости, у этих сущностей имеется что-то общее. Возможно, их было бы правильнее определять как подтипы некоторого общего типа сущности. Заметим, что пример с рис. 5.12 явно демонстрирует, что далеко не всегда типы сущности, участвующие в альтернативной связи, обладают общими чертами. Создание общего супертипа для типов сущности ПИЛОТ и АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ представляется весьма странной идеей.

На этом мы заканчиваем краткую экскурсию в семантическое моделирование с использованием ER-диаграмм.

Заключение

Основной целью данной лекции было ознакомление с семантическими моделями данных на примере упрощенного варианта ER-модели. Представленный вариант ER-модели, с одной стороны, является достаточно развитым, чтобы можно было почувствовать общую специфику семантических моделей данных, а с другой стороны, не перегружен деталями и излишними понятиями, затрудняющими общее понимание подхода.

С практической точки зрения наибольшую пользу могут принести рассмотренные приемы перехода от ER-диаграмм к схеме реляционной базы данных. Особенно могут пригодиться рекомендации по представлению в реляционной схеме связей «многие ко многим», подтипов и супертипов сущности и взаимно исключающих связей.

Тема 6. Язык SQL

SQL: история языка

Одним из наиболее важных шагов на пути к признанию SQL на рынке стало появление стандартов на этот язык. Обычно при упоминании стандарта SQL имеют в виду официальный стандарт, утвержденный Американским институтом национальных стандартов (American National Standards Institute – ANSI) и Международной организацией по стандартам (International Standards Organization – ISO). Однако существуют и другие важные стандарты SQL, включая SQL, реализованный в системе DB2 компании IBM, и стандарт X/OPEN для SQL в среде UNIX.

Стандарт SQL ANSI/ISO

Работа над официальным стандартом SQL началась в 1982 году, когда ANSI поставил перед своим комитетом ХЗН2 задачу по созданию стандарта языка реляционных баз данных. Вначале в комитете обсуждались достоинства различных предложенных языков. Однако поскольку к тому времени SQL уже стал фактическим стандартом, комитет ХЗН2 остановил свой выбор на нем и занялся стандартизацией SQL.

Разработанный в результате стандарт в большой степени был основан на диалекте SQL системы DB/2, хотя и содержал в себе ряд существенных отличий от этого диалекта. После нескольких доработок, в 1986 году стандарт был официально утвержден как стандарт ANSI номер Х3.135, а в 1987 году — в качестве стандарта ISO. Затем стандарт ANSI/ISO был принят правительством США как федеральный стандарт США по обработке информации (FIPS — Federal Information Processing Standard). Этот стандарт, незначительно пересмотренный в 1989 году, обычно называют стандартом "SQL-89", ил"SQLI". Когда в данной книге упоминается "стандарт ANSI/ISO", то подразумевается SQLI, который в настоящее время лежит в основе большинства коммерческих продуктов.

Многие из членов комитетов по стандартизации ANSI и ISO представляли фирмы- поставщики различных СУБД, в каждой из которых был реализован собственный диалект SQL. Как и диалекты человеческого языка, диалекты SQL были в основном похожи друг на друга, однако несовместимы в деталях. Во многих случаях комитет просто обошел существующие различия и не стандартизировал некоторые части языка, определив, что они реализуются по усмотрению разработчика. Этот подход позволил объявить большое число реализаций SQL совместимыми со стандартом, однако сделал сам стандарт относительно слабым.

Чтобы заполнить эти пробелы, комитет ANSI продолжил свою работу и создал проект нового, более жесткого стандарта SQL2. В отличие от стандарта 1989 года, проект SQL2 предусматривал возможности, выходящие за рамки таковых, уже существующих в реальных коммерческих продуктах. А для следующего

за ним стандарта SQL3 были предложены еще более глубокие изменения. В результате предложенные стандарты SQL2 и SQL3 оказались более противоречивыми, чем исходный стандарт. Стандарт SQL2 прошел процесс утверждения в ANSI и был окончательно принят в октябре 1992 года. В то время, как первый стандарт 1986 года занимает не более ста страниц, официальный стандарт SQL2 содержит около шестисот.

Вопреки стандарту SQL2, во всех существующих на сегодняшний день коммерческих продуктах поддерживаются собственные диалекты SQL. Более того, поставщики СУБД включают в свои продукты новые возможности и расширяют собственные диалекты SQL, чем еще больше отдаляют их от стандарта. Однако ядро SQL стандартизировано довольно жестко. Там, где это можно было сделать, не ущемляя интересы клиентов, поставщики СУБД привели свои продукты в соответствие со стандартом SQL-89, то же самое постепенно произойдет и с SQL2.

SQL Data Definition Language – язык определения реляционных данных

Реляционная модель данных, как и любая логическая модель данных, рассматривается с трех сторон: структура объектов, обеспечение целостности и манипулирование объектами.

Структура реляционных данных

Начнем с неформального определения терминов

Пример 1.

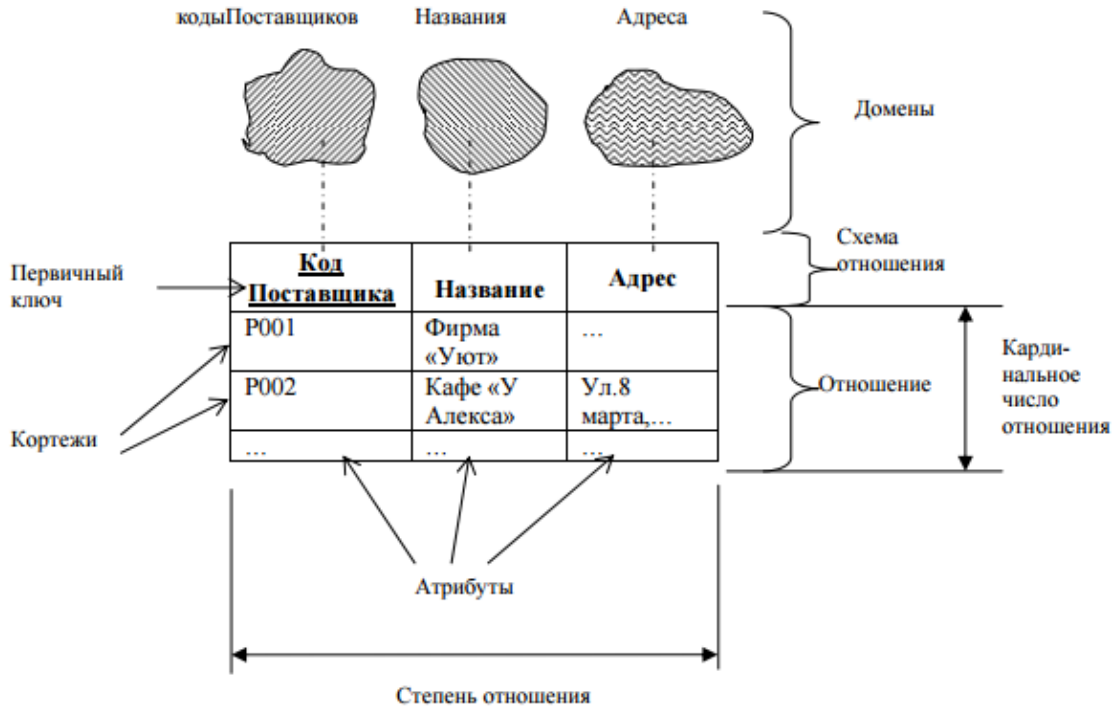


Рисунок 6.1-Реляционное отношение «ПОСТАВЩИКИ» очень подробно.

Отношение соответствует таблице, кортеж – строке таблицы, атрибут – столбцу таблицы. Первичный ключ – уникальный идентификатор для таблицы, т.е. столбец или комбинация столбцов, для которых в любой момент времени не существует двух строк с одинаковыми значениями в этих столбцах.

Домен – множество допустимых значений атрибута.

Минимальной единицей данных в реляционной модели данных является отдельное значение данных. В реляционной модели данных все отдельные значения данных являются атомарными, т.е. у них нет внутренней структуры, и они неразложимы на составляющие.

Определение реляционных данных

Домен D – конечное или счетное множество атомарных значений одного типа.

Пример 2. Домен категорий товаров, таких как «бытовая техника», «продукты питания» и т.п. может быть таким: строки русских символов длиной 20 символов.

Инструкция для создания домена в SQL (все предложения SQL даны по стандарту SQL92) выглядит так:

CREATE DOMAIN имя_домена тип_данных
DEFAULT значение;

Пример 3. Определим домен для примера 2 как

CREATE DOMAIN Category varchar(20);

Инструкция для удаления домена такова:

DROP DOMAIN имя_домена;

Инструкция SQL на создание таблицы выглядит так:

CREATE TABLE имя_таблицы (
определение_поля[, определение_поля,…n] ограничение_на_таблицу[,…n]
);

где:

определение_поля имеет вид
имя_поля тип [(размер)] [ограничение_на_поле]

ограничение_на_поле имеет вид:
[CONSTRAINT имя_ограничения]
{PRIMARY KEY | UNIQUE | NOT NULL}

ограничение_на_таблицу имеет вид:
[CONSTRAINT имя_ограничения]

{PRIMARY KEY (ключевое_поле1[, ключевое_поле2 [, ...]]) | UNIQUE (уникальное_поле1[, уникальное_поле2 [, ...]]) | NOT NULL (непустое_поле1[, непустое_поле2 [, ...]]) |
FOREIGN KEY (поле_ссылки1[, поле_ссылки2 [, ...]]) REFERENCES
внешняяТаблица [(внешнее_поле1 [, внешнее_поле2 [, ...]])]}

имя_поля, имя_таблицы – допустимые имена в Access; названия таблиц и полей, включающие пробелы, а также символы национальных кодировок, заключаются в прямоугольные скобки

«[ ]», например [название кинофильма]. Допустимыми считаются имена длиной до 64 символов, не содержащие точки «.», восклицательного знака «!», символа обратной кавычки «`» и символов квадратных скобок «[ ]», и начинающиеся не с пробела.

Инструкция SQL на удаление таблицы такова:

DROP TABLE имя_таблицы;

В стандарте SQL предложение CONSTRAINT используется для задания ограничений на значения полей записи (в частности, уникальность – UNIQUE, обязательность – NOT NULL), для установления отношений между таблицами (REFERENCES), а также для задания ограничений на таблицу целиком.

Отметим, что имена ограничений в выражениях CONSTRAINT в пределах одной базы данных должны быть различными.

Пример 4.

```
CREATE TABLE E1 (
a1 autoincrement not null CONSTRAINT pk_E1 PRIMARY KEY, a2 datetime not null,
a3 text(40)
);
```

Создается таблица из трех полей, причем поле a1 становится первичным ключом, и для этого поля создается индекс. Для определения первичного ключа здесь используется ограничение на поле с именем pk_E1.

Пример 5.

```
CREATE TABLE E2 (
b1 float,
b2 datetime, b3 long,
b4 text(100),
CONSTRAINT pk_E2 PRIMARY KEY (b2,b3), CONSTRAINT fk_E2_E1 FOREIGN KEY (b3) REFERENCES E1 (a1));
```

Создается таблица, в которой первичный ключ состоит из двух полей, b2 и b3, и при этом значения поля b3 берутся из значений поля a1 таблицы E1.

Целостность реляционных данных

В любой момент времени любая база данных содержит некоторый набор значений данных. Предполагается, что база данных является моделью реального мира. Однако, набор значений не имеет смысла, если значения не представляют определенного состояния реального мира.

Пример 6.

«Поставщик А сделал заказ на (-100) кондиционеров» - бессмысленное значение.

Следовательно, определение базы данных нуждается в расширении, включающем правила целостности. Суть правил состоит в том, чтобы сообщать СУБД об ограничениях на сочетания значений в реальном мире. Такие правила целостности называются специализированными, т.к. они применяются только в рамках конкретной базы данных.

Поэтому в реляционной модели данных есть и общие для всех реляционных баз данных правила целостности, и специализированные, относящиеся к конкретной базе данных.

Общих правил целостности всего два:

Первое правило: В первичном ключе не должно быть атрибутов, которые могут принимать неизвестные (неопределенные) значения.

В SQL для описания первичного ключа используется определение PRIMARY KEY

Второе правило: База данных не должна содержать несогласованных значений внешних ключей .

Замечание: Несогласованное значение – это значение внешнего ключа, для которого не существует отвечающего ему значения первичного ключа.

Определить внешний ключ можно оператором SQL:

FOREIGN KEY (список атрибутов внешнего ключа)
REFERENCES имя_таблицы с первичным ключом

(список атрибутов первичного ключа)

Пример 7.

Рассмотрим ситуацию, когда в базе данных предприятия хранятся сведения о сотрудниках и о детях сотрудников.

Ответим на такие вопросы:

Что произойдет, если удалить запись, на которую ссылается внешний ключ в таблице

«ДЕТИ_СОТРУДНИКОВ» (например, удалить запись о сотруднике с Таб.№ 5052, если этот сотрудник имеет двух детей)?

Что произойдет при попытке обновить значение первичного ключа записи о сотруднике, на который ссылается внешний ключ?

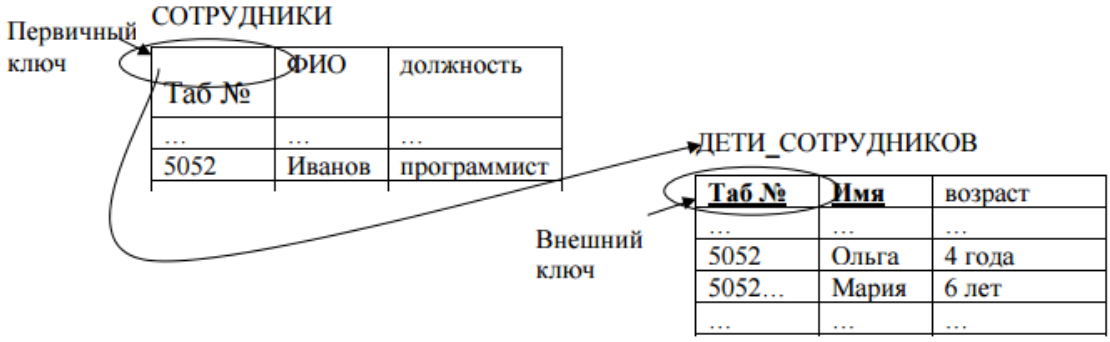


Рисунок 6.2 – Первичный и внешний ключи в таблицах

В общем, существует по крайней мере две возможности:

Ограничить (в SQL - оператор NO ACTION) –

ограничить операцию «удаление»/«обновление» до того момента, когда уже не будет существовать соответствующее данному значению Таб № первичного ключа записей в отношении с внешним ключом (в противном случае операция запрещается)

Каскадировать (в SQL - оператор CASCADE) –

каскадировать операцию «удаления»/«обновления», удаляя/обновляя также и внешние ключи.

Поэтому, в SQL92 задается ссылочная целостность в явном виде:

FOREIGN KEY (список атрибутов внешнего ключа) REFERENCES имя_таблицы

ON DELETE опция

ON UPDATE опция

где задание поведения СУБД при удалении/обновлении ключа в родительской таблице задается опцией:

CASCADE NO ACTION

вызов определенной процедуры (хранимой процедуры или процедуры триггера

Пример 8.

Вместо каскадирования или ограничения на изменение значений может вызываться некоторая процедура, например, при удалении записи информация заносится в архивную базу и т.п.

Целостность и неопределенные значения данных (null-значения)

Проблема отсутствующей информации достаточно часто встречается на практике. Для решения проблемы создаются разные подходы. Часто используются «значения по умолчанию». Например, если цвет детали не известен, или не имеет значения в данной базе, то «значение по умолчанию» - «нет» иначе – название цвета.

Другой подход, предложенный Коддом, базируется на обозначении отсутствующего значения специальным маркером, который называется null-значением. Это значение используется всегда, когда правильное значение для любого атрибута базы неизвестно или отсутствует.

Теория, изучающая null-значения, базируется на трехзначной логике (когда разрешаются три значения: «да», «нет», «UNK” – unknown - неизвестно)

Таким образом, становится очевидным правило целостности по сущностям, которые мы сформулировали в этой лекции выше:

«Ни один элемент первичного ключа не может иметь неопределенное, «null-значение»

Аналогично и для правила целостности по ссылкам:

«Всегда значения элементов внешнего ключа или являются null-значениями, или совпадают с некоторыми не null- значениями элементов первичного ключа»

Если СУБД разрешает использование null-значений, то к двум возможностям обработки ссылок по внешним ключам, добавляется третья.

Правило null-значений: Для каждого внешнего ключа разработчик должен решить, может ли внешний ключ содержать null-значения.

Как правило, можно задать разрешение на использование Null-значений на всю базу данных с помощью оператора SQL -

NULLS ALLOWED (разрешается использование Null-значений), либо

NULLS NOT ALLOWED (запрещается использование Null-значений)

Тогда, если Null-значения разрешаются, то добавляется новая опция: SET NULL (аннулировать) наряду с CASCADE и NO ACTION.

В соответствии с опцией SET NULL, при изменении записи о сотруднике внешний ключ устанавливается равным Null для всех записей о детях, а запись о сотруднике потом удаляется/обновляется.

В SQL92 к опциям CASCADE и NO ACTION добавляется SET NULL и SET DEFAULT: FOREIGN KEY (список атрибутов внешнего ключа)

REFERENCES имя_таблицы ON DELETE опция ON UPDATE опция

где опция –

либо CASCADE либо NO ACTION либо SET NULL

либо SET DEFAULT (как установлено по умолчанию для всей базы данных)

Пример 10.

Допустим, мы разрешили существование Null-значений в таблице «ДЕТИ_СОТРУДНИКОВ». Примером может быть ситуация, когда сотрудника увольняют, но о детях продолжают заботиться, и информация о них остается в базе данных.

Опишем эту ситуацию на SQL:

```
CREATE TABLE Employees
(tab_number integer NOT NULL UNIQUE, name varchar (50) NOT NULL,
position varchar (20) NOT NULL) PRIMARY KEY tab_number;
CREATE TABLE Childrens (tab_number integer NOT NULL,
child_name varchar (10) NOT NULL, age integer NOT NULL)
PRIMARY KEY (tab_number, child_name)
FOREIGN KEY (tab_number) REFERENCES Employees ON DELETE SET NULL
ON UPDATE CASCADE;
```

Таблица 1. Типы данных MS SQL Server 2000.

Тип данных	**Описание**
Bit	Один бит. В столбце с этим типом может храниться либо 0, либо 1.
Binary(n)	Двоичный данные фиксированной длины до 8000 байт.
Varbinary(n)	Двоичные данные переменной длины до 8000 байт.
Image	Двоичные данные длиной до 2Гб (2^31–1). Пространство для хранения данных этого типа отводится страницами, размер которых составляет 8 КБ.
Decimal	Используется для хранения дробных чисел с фиксированным количеством знаков до и после десятичной точки. От -10^38 +1 до 10^38 –1.
Numeric	Является аналог типа данных Decimal.
Float(n)	Используется для хранения дробных чисел с приблизительной точностью. От -1.79E+308 до 1.79E+308.
Real	Является аналогом float(24). От -3.40E+38 до 3.40E+38.
Int	Занимает 4 байта и может хранить целые числа в диапазоне от -2^31 до 2^31-1
Smallint	Занимает 2 байта и может хранить целые числа в диапазоне от -2^15 до 2^15-1
Tinyint	Занимает 1 байт и может хранить целые числа в диапазоне от 0 до 255

Bigint	Занимает 8 байт и может хранить целые числа в диапазоне от -2^63 до 2^63-1
Char(n)	Строковый тип данных фиксированной длины без поддержки Unicode длиной до 8000 символов.
Nchar(n)	Строковый тип данных фиксированной длины с поддержкой Unicode длиной до 4000 символов.
Varchar(n)	Строковый тип данных переменной длины без поддержки Unicode длиной до 8000 символов.
Nvarchar(n)	Строковый тип данных переменной длины с поддержкой Unicode длиной до 4000 символов.
Text	Строковый тип данных без поддержки Unicode длиной до 2 миллиардов символов (2^31–1). Пространство для хранения данных этого типа отводится страницами по 8 Кб.
Ntext	Строковый тип данных с поддержкой Unicode длиной до 1 миллиардов символов. Пространство для хранения данных этого типа отводится страницами по 8 Кб.
Datetime	Тип данных для хранения даты и времени с высокой точностью в диапазоне от 1 января 1753 года до 31 декабря 9999 года. Занимает 8 байт. Точность - 3,33 миллисекунды.
Smalldatetime	Тип данных для хранения даты и времени с нормальной точностью в диапазоне от 1 января 1900 года до 06 июня 2079 года. Занимает 4 байта. Точность - 1 минута.
Money	Используется для хранения денежных данных в большом диапазоне. Точность хранения до 4 знаков после десятичной точки. Занимает 8 байт. От -2^63 до 2^63 – 1.
Smallmoney	Используется для хранения денежных данных в нормальном диапазоне. Точность хранения до 4 знаков после десятичной точки. Занимает 4 байта. От -214,748.3648 до 214,748.3647.
Sql_variant	Тип данных, позволяющий хранить данные нескольких типов в одном и том же столбце. Например, при использовании этого типа данных в столбце могут находиться одновременно данные целочисленные, дробные, символьные и т.д.
Sysname	Используется для указания имен объектов СУБД. Аналог
	nvarchar(128).
Timestamp	Временной штамп. Значения в столбце этого типа изменяются SQL Server автоматически при каждом изменении строки. Можно использовать псевдоним rowversion.
Table	Этот тип данных предназначен для временного хранения сложных типов данных – для переменных Transact-SQL, параметров хранимых процедур и для значений, возвращаемых функциями пользователя.
Uniqueidentifier	Используется для хранения глобальных уникальных идентификаторов.
cursor	Предназначен для хранения ссылок на курсоры.

РАЗДЕЛ 2. СОВРЕМЕННЫЕ БД

Тема 7. Объектно-ориентированные базы данных

Направление объектно-ориентированных баз данных (ООБД) возникло сравнительно давно. Публикации появлялись уже в середине 1980-х. Однако наиболее активно это направление развивается в последние годы. С каждым годом увеличивается число публикаций и реализованных коммерческих и экспериментальных систем.

Большое число работ не означает, что все проблемы ООБД полностью решены. Как отмечается в Манифесте группы ведущих ученых, занимающихся ООБД, современная ситуация с ООБД

напоминает ситуацию с реляционными системами середины 1970-х. При наличии большого количества экспериментальных проектов (и даже коммерческих систем) отсутствует общепринятая объектно-ориентированная модель данных, и не потому, что нет ни одной разработанной полной модели, а по причине отсутствия общего согласия о принятии какой-либо модели. На самом деле, имеются и более конкретные проблемы , связанные с разработкой декларативных языков запросов, выполнением и оптимизацией запросов, формулированием и поддержанием ограничений целостности, синхронизацией доступа и управлением транзакциями и т.д.

Конечно, ООБД возникли не на пустом месте. Соответствующий базис обеспечивают как предыдущие работы в области БД, так и давно развивающиеся направления языков программирования с абстрактными типами данных и объектно-ориентированных языков программирования.

Связь объектно-ориентированных СУБД с общими понятиями объектно-ориентированного подхода.

В наиболее общей и классической постановке объектно-ориентированный подход базируется на следующих концепциях:

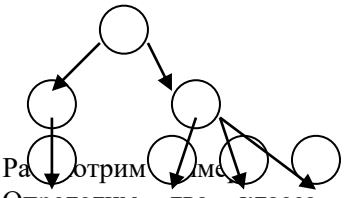
- объекта и идентификатора объекта;
- атрибутов и методов;
- классов;
- иерархии и наследования классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом все время его существования и не меняется при изменении состояния объекта.

Каждый объект имеет состояние и поведение. Состояние объекта - набор значений его атрибутов. Поведение объекта - набор методов (программный код), оперирующих над состоянием объекта. Значение атрибута объекта - это тоже некоторый объект или множество объектов. Состояние и поведение объекта инкапсулированы в объекте; взаимодействие объектов производится на основе передачи сообщений и выполнении соответствующих методов.

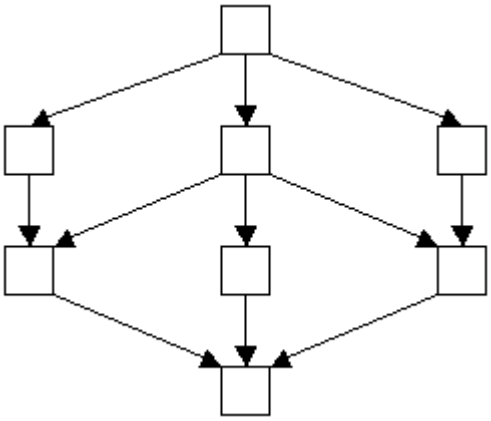
Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Объект должен принадлежать только одному классу (если не учитывать возможности наследования). Допускается наличие примитивных предопределенных классов, объекты-экземпляры которых не имеют атрибутов: целые, строки и т.д. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется доменом этого атрибута.

Допускается порождение нового класса на основе уже существующего класса - наследование. В этом случае новый класс, называемый подклассом существующего класса (суперкласса), наследует все атрибуты и методы суперкласса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различаются случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного суперкласса, во втором случае суперклассов может быть несколько. Если в языке или системе поддерживается единичное наследование классов, набор классов образует древовидную иерархию. При поддержании множественного наследования классы связаны в ориентированный граф с корнем, называемый решеткой классов. Объект подкласса считается принадлежащим любому суперклассу этого класса.



Рассмотрим схему.

Определим два класса верхнего уровня: "Отдел" и "Служащий". Мы также определим класс Руководитель, который является подклассом класса Служащий.



Служащий	
Свойство	**Тип**
Имя Служащего Номер Служащего Местонахождение Список Отделов	*Текстовый* *Цифровой* *Текстовый* *Список объект. ссылок: Отдел*
Метод	***Возвращает***
Повысить (Отдел)	*Ссылка на (Руководитель)*

Руководитель	
(подкласс класса Служащий)	
Свойство	**Тип**
Имя Служащего Номер Служащего Местонахождение Список Отделов Ск. времени руководит Звание	*Текстовый* *Цифровой* *Текстовый* *Список ссылок на: Отдел* *Целое* *Текстовый*
Метод	***Возвращает***
<Повысить>	*Явно не унаследованный*

Объекты (экземпляры класса) могут быть созданы путем применения метода "Создать экземпляр", который является общим для всех классов. Объект класса "Служащий", однажды созданный, связывается с отделом при помощи метода "Нанять" класса "Отдел" (со ссылкой на объект "Служащий" в качестве аргумента).

Отдел	
Свойство	**Тип**
Название Отдела Бюджет Список Служащих Руководитель Отдела	*Текстовый* *Цифровой* *Список ссылок на: Служащий* *Ссылка на: Руководитель*
Метод	***Возвращает***
Нанять(Служащий)	<ничего>

Для того, чтобы сделать служащего руководителем, надо применить метод "Повысить" к служащему (аргумент - ссылка на "Отдел"). Этот метод создает новый объект класса "Руководитель" с дубликатами наследуемых свойств, и затем уничтожает существующий экземпляр "Служащего" (используя метод "Уничтожить", также общий для всех классов). Возвращается ссылка на новый объект (дескриптор объекта). Отметим, что метод также должен скорректировать свойство "Список служащих" каждого отдела, входящего в "Список отделов", в которых работает служащий. Это необходимо для сохранения целостности БД. Поскольку "Руководитель" является подклассом "Служащего", этот "Список Служащих" не будет иметь никаких конфликтов (ссылка на "Служащего", на самом деле, может быть ссылкой на Руководителя, так как он является подклассом служащего - обратное неверно).

Отметим также, что наследование метода "Повысить" явно заблокировано в подклассе Руководитель (это обычно достигается определением на уровне класса Руководитель метода "Повысить", который ничего не делает). Это годится лишь когда служащий может быть руководителем только одного отдела. Если же отдельный служащий может руководить несколькими отделами, метод "Повысить" должен быть применим и к руководителям. В этом случае метод "Повысить" обязан вносить коррективы в свойство "Руководитель отдела" для соответствующих отделов из "Список отделов" объекта "Руководитель".

Объектная технология расширяет традиционную методику разработки приложений новым моделированием данных и методами программирования. Для повторного использования кода и улучшения сохранности целостности данных в объектном программировании данные и код для их обработки организованы в объекты. Таким образом, практически полностью снимаются ограничения на типы данных.

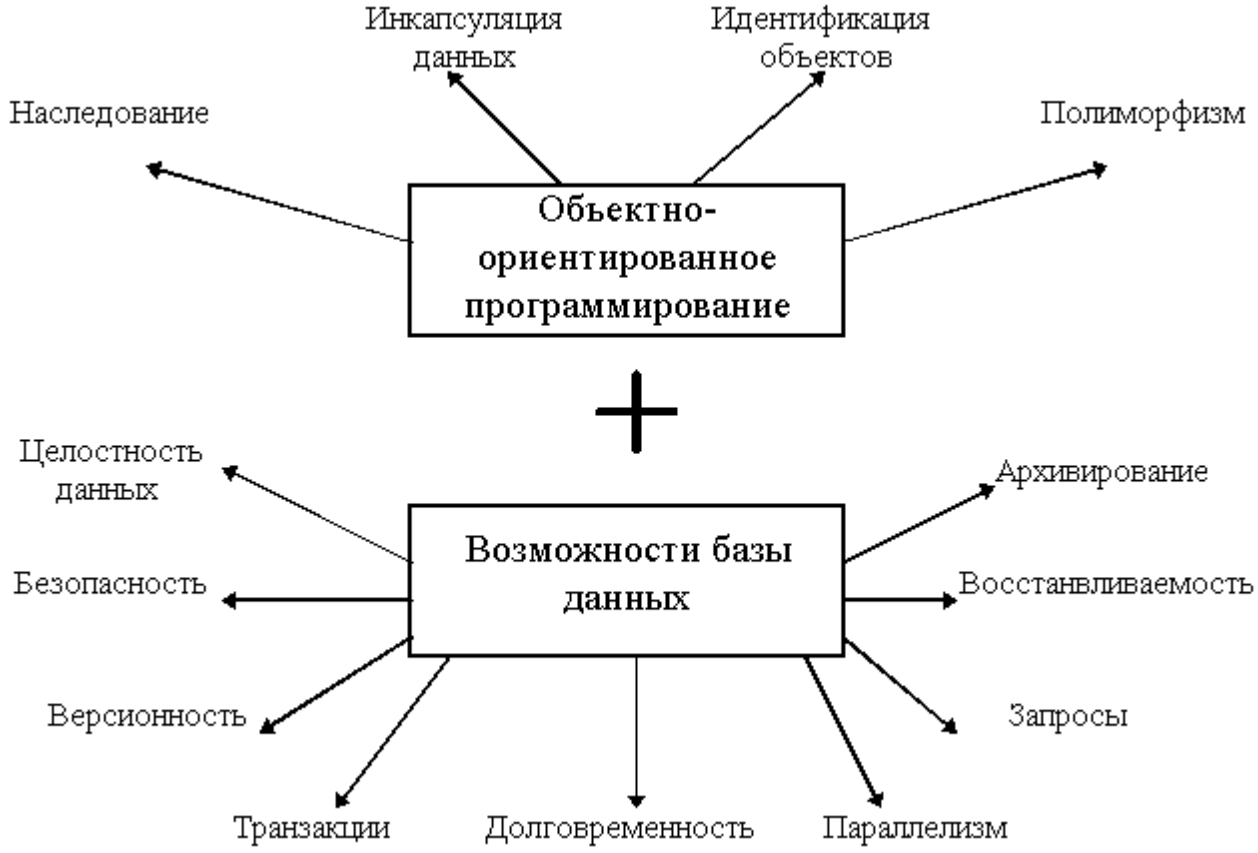
Если данные состоят из коротких, простых полей фиксированной длины (имя, адрес, баланс банковского счета), то лучшим решением будет применение реляционной базы данных. Если, однако,

данные содержат вложенную структуру, динамически изменяемый размер, определяемые пользователем произвольные структуры (мультимедиа, например), представление их в табличной форме будет, как минимум, непростым. В то же время в ООСУБД каждая определенная пользователем структура – это объект, непосредственно управляемый базой данных.

Применение объектной модели предпочтительнее для баз данных с большим количеством сложных связей: перекрестных ссылок, ссылок, связывающих несколько объектов с несколькими (many-to-many relationships) двунаправленными ссылками.

В объектно-ориентированных базах данных, в отличие от реляционных, хранятся не записи, а объекты. ОО-подход представляет более совершенные средства для отображения реального мира, чем реляционная модель: естественное представление данных. В реляционной модели все отношения принадлежат одному уровню, именно это осложняет преобразование иерархических связей модели "сущность-связь" в реляционную модель. ОО-модель можно рассматривать послойно, на разных уровнях абстракции.

Объектные СУБД реализуют весь набор функций, присущих системам управления базами данных плюс возможности объектного программирования.



В то же время, ОО-модели присущ и ряд недостатков:

- осутствуют мощные непроцедурные средства извлечения объектов из базы. Все запросы приходится писать на процедурных языках, проблема их оптимизации возлагается на программиста.
- вместо чисто декларативных ограничений целостности (типа явного объявления первичных и внешних ключей реляционных таблиц с помощью ключевых слов PRIMARY KEY и REFERENCES) или полудекларативных триггеров для обеспечения внутренней целостности приходится писать процедурный код.

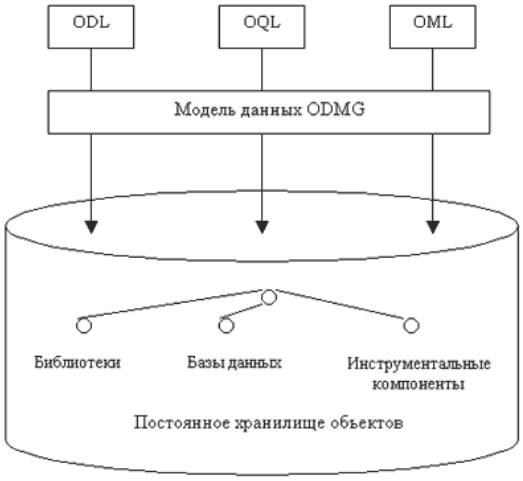
Очевидно, что оба эти недостатка связаны с отсутствием развитых средств манипулирования данными. Эта задача решается двумя способами - расширение ОО-языков в сторону управления данными (стандарт ODMG), либо добавление объектных свойств в реляционные СУБД.

Стандарт ODMG.

ODMG (Object Data Management Group) - консорциум поставщиков ООБД и других заинтересованных организаций, созданный в 1991 г. Его задачей является разработка стандарта на хранение объектов в базах даннных.

Архитектура ODMG.

Предлагаемая ODMG архитектура показана на рисунке. В этой архитектуре определяются способ хранения данных и разные виды пользовательского доступа к этому “хранилищу данных”. Единое хранилище данных доступно из языка определения данных, языка запросов и ряда языков манипулирования данными. ODL означает Object Definition Language (язык определения объектов), OQL – Object Query Language (язык объектных запросов) и OML – Object Manipulation Language (язык манипулирования объектами).



Архитектура ODMG

Объектная модель данных. Все данные, сохраняемые ООСУБД, структуризуются в терминах конструкций модели данных. В модели данных определяется точная семантика всех понятий. Модель данных – основа стандарта ODMG, представляет организационную структуру, в которой сохраняются все данные, управляемые ООСУБД. Язык определения объектов, язык запросов и языки манипулирования разработаны таким образом, что все их возможности опираются на модель данных.

Язык определения данных (ODL) – средство определения схемы базы данных (по аналогии с DDL в реляционных СУБД). ODL создает слой абстрактных описаний так, что схема базы данных становится независима как от языка программирования, так и от СУБД.

Язык объектных запросов (ODL). Язык имеет синтаксис, похожий на синтаксис языка SQL, но опирается на семантику объектной модели ODMG . В стандарте допускается прямое использование OQL и его встраивание в один из языков категории OML .

Языки манипулирования объектами (OML). Для программирования реализаций операций и приложений требуется наличия объектно-ориентированного языка программирования. OML представляется собой интегрирование языка программирования с моделью ODMG ; это интегрирование производится за счет определенных в стандарте правил языкового связывания (language binding и обеспечивает языковое связывание.

В одной ООСУБД могут поддерживаться несколько OML . В стандарте ODMG-2 были специфицированы правила связывания для языков C++ и Smalltalk.

Постоянное хранилище объектов. Логическая организация хранилища данных любой ООСУБД, совместимой со стандартном ODMG, должна основываться на модели данных ODMG. Физическая организация у разных ООСУБД может различаться, но в любом случае она должна обеспечивать эффективные структуры данных для хранения иерархии типов и объектов, являющихся экземплярами этих типов. Иерархия типов связана не только с данными, но и с различными библиотеками и компонентами инструментальных средств, поддерживающими разработку приложений. Так что в ООСУБД, совместимой со стандартом ODMG, хранилище представляет собой интегрированную систему, где согласованным образом сохраняются данные и программный код.

Инструментальные средства и библиотеки. Инструментальные средства, поддерживающие, например, разработку пользовательских приложений и их графических интерфейсов, программируются на одном из OML и сохраняются как часть иерархии классов. Библиотеки функций доступа, арифметических функций и т.д. также сохраняются в иерархии типов и являются единообразно доступными из программного кода разработчика приложения. Ассортимент инструментальных средств и библиотек в стандарте не определяется.

Список объектно-ориентированных систем управления базами данных включает в себя, например, следующие продукты:

Objectivity/DB, компании Objectivity, Inc.
СУБД GemStone, корпорации GemStone Systems, Inc.
СУБД ObjectStore, Object Design Inc.
СУБД Versant, Versant Object Technology, Inc.
Itasca, Itasca Systems Inc.
O2, O2 Technology

В общем смысле все системы соответствуют базовой модели ODMG , но следует обратить внимание, что крайне редко в качестве языка запросов поддерживается OQL, и ни в одной системе не поддерживается ODL .

Ни одна из компаний, производящих ООСУБД, так и не смогла набрать критическую массу, достаточную для того, чтобы стать лидером, диктующим моду в данной области (как это произошло с IBM и Oracle в области SQL -ориентированных СУБД). Крупные компьютерные компании так и не решились приобрести какой-нибудь продукт ООСУБД, чтобы развивать его и продвигать на рынке.

Объектные расширения реляционных СУБД. Объектные расширения Oracle.

Объектные расширения реализованы в трех ведущих РСУБД – Oracle , IBM Informix и DB 2. Заметим, что компания Microsoft решила не внедрять объектные расширения в свой основной продукт управления базами данных – Microsoft SQL Server .

После выпуска в 1997г. системы Oracle 8, которая действительно претендовала на статус объектно-реляционной системы, впоследствии корпорация Oracle не стала уделять большого внимания развитию именно этих свойств своей системы. Лишь в системе Oracle 9i (2002 г.) появились существенные дополнения.

Корпорация Oracle расширила SQL (как DDL, так и DML), чтобы позволить пользователям:

- определять свои собственные типы (которые представляют их бизнес-объекты) и связи (например, наследование и агрегирование) между этими определяемыми пользователями типами;
- хранить их экземпляры (то есть, объекты) в базе данных (либо в столбцах таблиц, либо как сами таблицы);
- запрашивать, вставлять и изменять эти экземпляры.

Бизнес-объект:

- может содержаться внутри другого бизнес-объекта;
- на него может ссылаться другой объект (используя конструкцию, называемую REF);
- к нему можно получить доступ;
- с ним можно манипулировать как с коллекциями (collections) или наборами (sets), используя структуры, называемые массивами переменной длины (VARRAYS) и вложенными таблицами (Nested Tables).

Примеры:

Корневой тип иерархии создается с помощью оператора CREATE TYPE, в котором должен быть указано ключевое слово NOT FINAL (нетерминальный).

```
CREATE TYPE Person_t AS OBJECT(
name VARCHAR2(100),
dob DATE,
MEMBER FUNCTION age() RETURN number,
MEMBER FUNCTION print() RETURN varchar2) NOT FINAL;
```

Под нетерминальным типом может быть создан подтип. Он наследует все атрибуты и методы от своего супертипа. В нем можно добавить новые атрибуты и методы и/или переопределить унаследованные методы.

```
CREATE TYPE Employee_t UNDER Person_t(
salary NUMBER,
bonus NUMBER,
MEMBER FUNCTION wages() RETURN number,
OVERRIDING MEMBER FUNCTION print() RETURN varchar2);
```

Объектное представление может быть создано как подпредставление другого объектного представления, создавая тем самым иерархию представлений. Преимущества иерархии представлений вытекают из того факта, что строки иерархии объектных представлений включают все строки их

подпредставлений. Благодаря этому свойству, можно запрашивать объекты типов, принадлежащих к некоторой иерархии, осмысленным образом.

CREATE VIEW Persons OF Person_t WITH OBJECT ID (name) AS SELECT name, dob FROM r_persons;

CREATE VIEW Employees OF Employee_t UNDER Persons

AS SELECT name, dob, salary, bonus from r_employees;

Запрос к представлению Persons выберет всех лиц, включая служащих.

SELECT VALUE(p) FROM Persons p;

Тема 8. Распределенные базы данных и системы клиент-сервер

Понятие распределенной системы

По утверждению известного специалиста в области информатики Э. Таненбаума, не существует общепринятого и в то же время строгого определения распределенной системы. Некоторые остряки утверждают, что распределенной является такая вычислительная система, в которой неисправность компьютера, о существовании которого пользователи ранее даже не подозревали, приводит к остановке всей их работы. Значительная часть распределенных вычислительных систем, к сожалению, удовлетворяют такому определению, однако формально оно относится только к системам с уникальной точкой уязвимости (single point of failure).

Часто при определении распределенной системы во главу угла ставят разделение ее функций между несколькими компьютерами. При таком подходе распределенной является любая вычислительная система, где обработка данных разделена между двумя и более компьютерами. Основываясь на определении Э. Таненбаума, несколько более узко распределенную систему можно определить как набор соединенных каналами связи независимых компьютеров, которые с точки зрения пользователя некоторого программного обеспечения выглядят единым целым.

Такой подход к определению распределенной системы имеет свои недостатки. Например, все используемое в такой распределенной системе программное обеспечение могло бы работать и на одном единственном компьютере, однако с точки зрения приведенного выше определения такая система уже перестанет быть распределенной. Поэтому понятие распределенной системы, вероятно, должно основываться на анализе образующего такую систему программного обеспечения.

Как основу описания взаимодействия двух сущностей рассмотрим общую модель взаимодействия клиент-сервер, в которой одна из сторон (клиент) инициирует обмен данными, посылая запрос другой стороне (серверу). Сервер обрабатывает запрос и при необходимости посылает ответ клиенту (рис. 1.1).

Рис. 1.1. Модель взаимодействия клиент сервер

Взаимодействие в рамках модели клиент сервер может быть как синхронным, когда клиент ожидает завершения обработки своего запроса сервером, так и асинхронным, при котором клиент посылает серверу запрос и продолжает свое выполнение без ожидания ответа сервера. Модель клиента и сервера может использоваться как основа описания различных взаимодействий. Для данного курса важно взаимодействие составных частей программного обеспечения, образующего распределенную систему.

Рис. 1.2. Логические уровни приложения

Рассмотрим некое типичное приложение, которое в соответствии с современными представлениями может быть разделено на следующие логические уровни (рис. 1.2): пользовательский интерфейс (ИП), логика приложения (ЛП) и доступ к данным (ДД), работающий с базой данных (БД). Пользователь системы взаимодействует с ней через интерфейс пользователя, база данных хранит данные, описывающие предметную область приложения, а уровень логики приложения реализует все алгоритмы, относящиеся к предметной области.

Поскольку на практике разных пользователей системы обычно интересует доступ к одним и тем же данным, наиболее простым разнесением функций такой системы между несколькими компьютерами будет разделение логических уровней приложения между одной серверной частью приложения, отвечающим за доступ к данным, и находящимися на нескольких компьютерах клиентскими частями, реализующими интерфейс пользователя. Логика приложения может быть отнесена к серверу, клиентам, или разделена между ними (рис. 1.3).

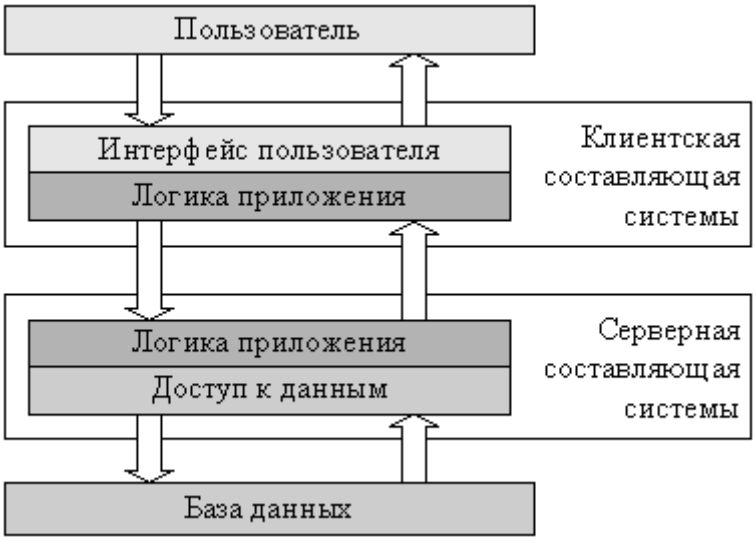


Рис. 1.3. Двухзвенная архитектура

Архитектуру построенных по такому принципу приложений называют клиент серверной или двухзвенной. На практике подобные системы часто не относят к классу распределенных, но формально они могут считаться простейшими представителями распределенных систем.

Развитием архитектуры клиент-сервер является трехзвенная архитектура, в которой интерфейс пользователя, логика приложения и доступ к данным выделены в самостоятельные составляющие системы, которые могут работать на независимых компьютерах (рис. 1.4).

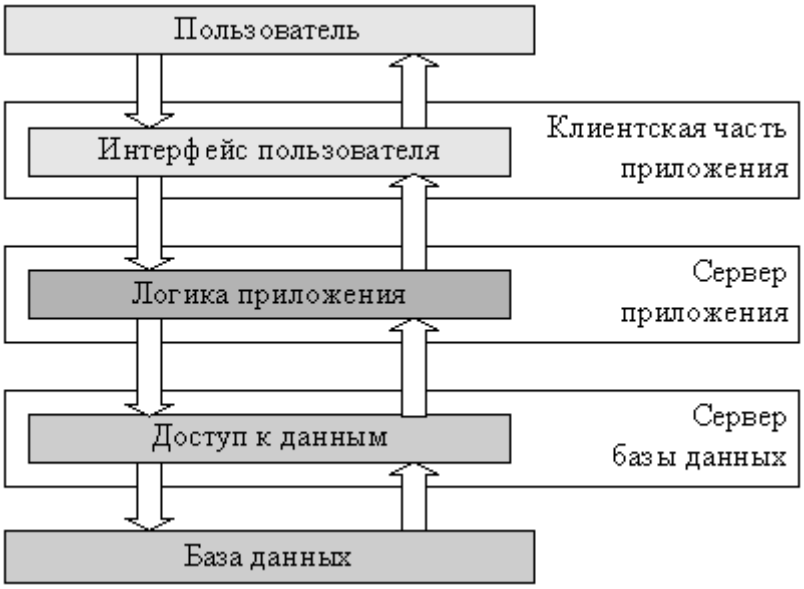


Рис. 1.4. Трехзвенная архитектура

Запрос пользователя в подобных системах последовательно обрабатывается клиентской частью системы, сервером логики приложения и сервером баз данных. Однако обычно под распределенной системой понимают системы с более сложной архитектурой, чем трехзвенная.

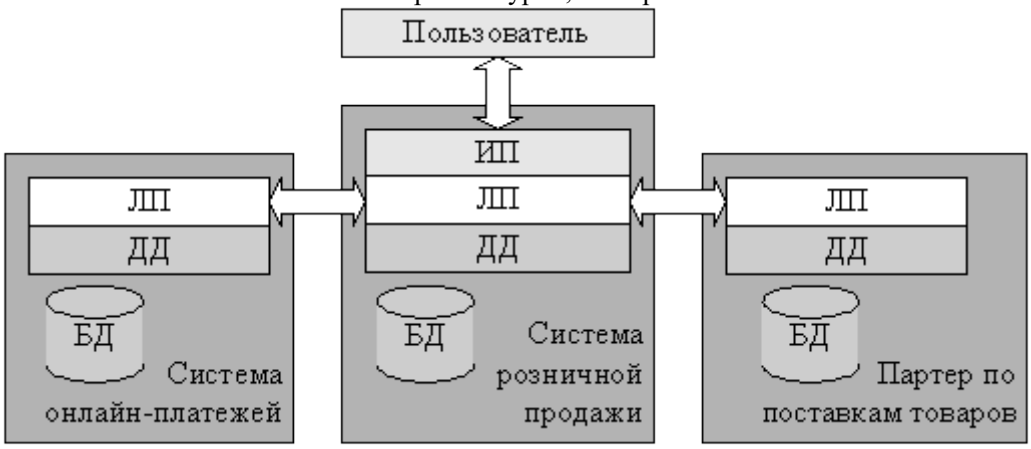


Рис. 1.5. Распределенная система розничных продаж

Применительно к приложениям автоматизации деятельности предприятия, распределенными обычно называют системы с логикой приложения, распределенной между несколькими компонентами системы, каждая из которых может выполняться на отдельном компьютере. Например, реализация логики приложения системы розничных продаж должна использовать запросы к логике приложения третьих фирм, таких как поставщики товаров, системы электронных платежей или банки, предоставляющие потребительские кредиты (рис. 1.5).

Таким образом, в обиходе под распределенной системой часто подразумевают рост многозвенной архитектуры "в ширину", когда запросы пользователя не проходят последовательно от интерфейса пользователя до единственного сервера баз данных.

В качестве другого примера распределенной системы можно привести сети прямого обмена данными между клиентами (peer-to-peer networks). Если предыдущий пример имел "древовидную" архитектуру, то сети прямого обмена организованы более сложным образом, рис. 1.6. Подобные системы являются в настоящий момент, вероятно, одними из крупнейших существующих распределенных систем, объединяющие миллионы компьютеров.

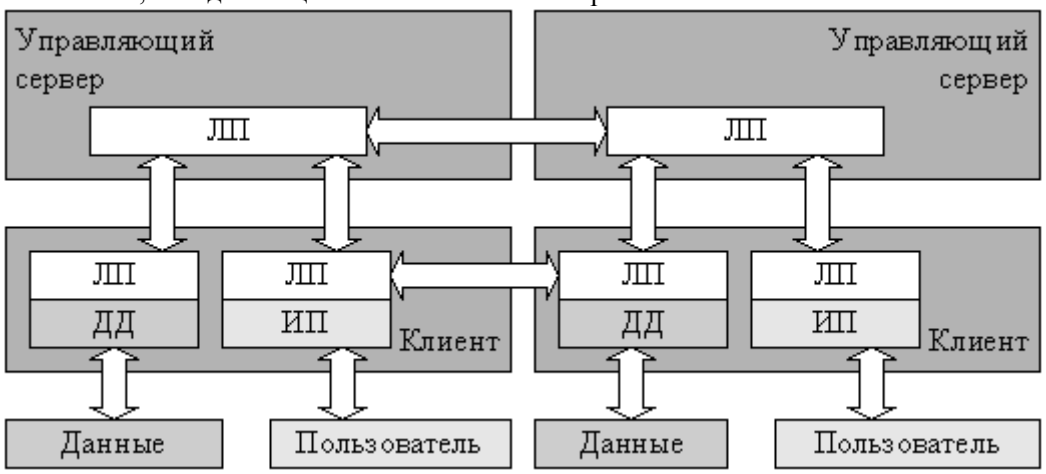


Рис. 1.6. Система прямого обмена данными между клиентами

Определение распределенной системы. Программные компоненты

В распределенных системах функции одного уровня приложения могут быть разнесены между несколькими компьютерами. С другой стороны, программное обеспечение, установленное на одном компьютере, может отвечать за выполнение функций, относящихся к разным уровням. Поэтому подход к определению распределенной системы, считающей ее совокупностью компьютеров, условен. Для описания и реализации распределенных систем было введено понятие программной компоненты.

Программная компонента – это единица программного обеспечения, исполняемая на одном компьютере в пределах одного процесса, и предоставляющая некоторый набор сервисов, которые используются через ее внешний интерфейс другими компонентами, как выполняющимися на этом же

компьютере, так и на удаленных компьютерах. Ряд компонент пользовательского интерфейса предоставляют свой сервис конечному пользователю.

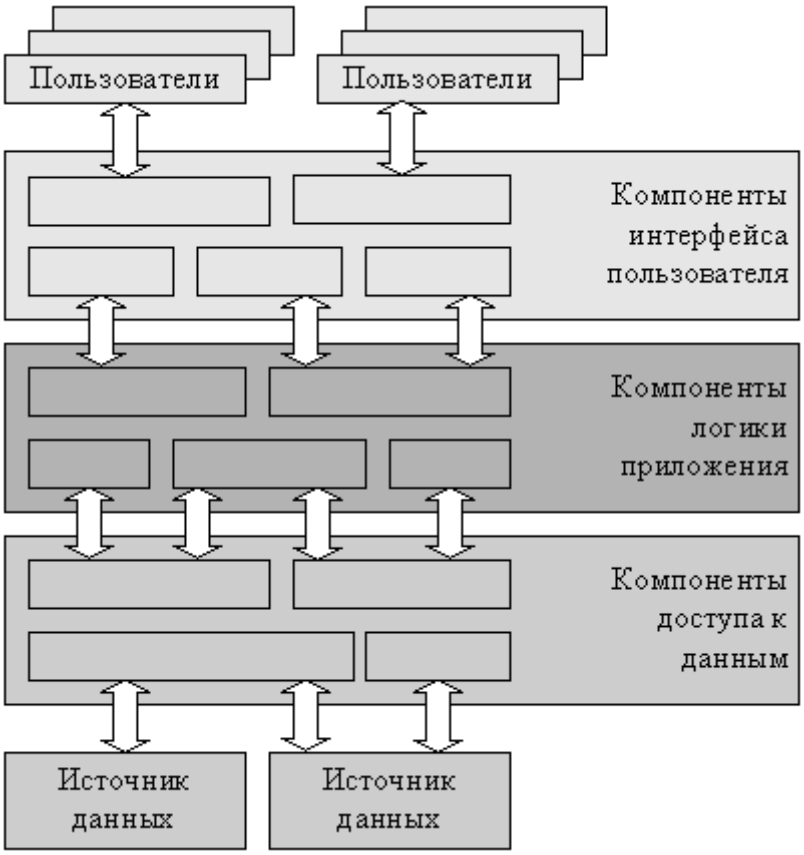


Рис. 1.7. Компоненты распределенной системы

Применительно к программам с использованием платформы CLI (Command Line Interface), под процессом в приведенном определении компоненты следует понимать домен приложения (application domain), который можно рассматривать как аналог процесса в управляемом коде.

Основываясь на определении программной компоненты, можно дать более точное определение распределенной системы. Согласно нему, распределенная система есть набор взаимодействующих программных компонент, выполняющихся на одном или нескольких связанных компьютерах и выглядящих с точки зрения пользователя системы как единое целое (рис. 1.7). Прозрачность является атрибутом распределенной системы. При исправном функционировании системы от конечного пользователя должно быть скрыто, где и как выполняются его запросы.

Программная компонента является минимальной единицей развертывания распределенной системы. В ходе модернизации системы одни компоненты могут быть обновлены независимо от прочих компонент.

В хорошо спроектированной системе функции каждой компоненты относятся только к одному уровню приложения. Однако разделение только на три уровня представляется недостаточным для классификации компонент. Например, часть компонент пользовательского интерфейса могут взаимодействовать с пользователем, а часть – предоставлять свои сервисы другим компонентам, но с пользователем не взаимодействовать. Классификации подобного рода существуют, однако они не являются общепринятыми и часто в значительной степени привязаны к приложениям автоматизации деятельности предприятия, что все таки не является синонимом распределенной системы

Требования к распределенным системам

Чтобы достигнуть цели своего существования – улучшения выполнения запросов пользователя – распределенная система должна удовлетворять некоторым необходимым требованиям. Можно сформулировать следующий набор требований, которым в наилучшем случае должна удовлетворять распределенная вычислительная система.

Открытость. Все протоколы взаимодействия компонент внутри распределенной системы в идеальном случае должны быть основаны на общедоступных стандартах. Это позволяет использовать для создания компонент различные средства разработки и операционные системы. Каждая компонента должна иметь точную и полную спецификацию своих сервисов. В этом случае компоненты распределенной системы могут быть созданы независимыми разработчиками. При нарушении этого

требования может исчезнуть возможность создания распределенной системы, охватывающей несколько независимых организаций.

Масштабируемость. Масштабируемость вычислительных систем имеет несколько аспектов. Наиболее важный из них для данного курса – возможность добавления в распределенную систему новых компьютеров для увеличения производительности системы, что связано с понятием балансировки нагрузки (load balancing) на серверы системы. К масштабированию относятся так же вопросы эффективного распределения ресурсов сервера, обслуживающего запросы клиентов.

Поддержание логической целостности данных. Запрос пользователя в распределенной системе должен либо корректно выполняться целиком, либо не выполняться вообще. Ситуация, когда часть компонент системы корректно обработали поступивший запрос, а часть – нет, является наихудшей.

Устойчивость. Под устойчивостью понимается возможность дублирования несколькими компьютерами одних и тех же функций или же возможность автоматического распределения функций внутри системы в случае выхода из строя одного из компьютеров. В идеальном случае это означает полное отсутствие уникальной точки сбоя, то есть выход из строя одного любого компьютера не приводит к невозможности обслужить запрос пользователя.

Безопасность. Каждый компонент, образующий распределенную систему, должен быть уверен, что его функции используются авторизированными на это компонентами или пользователями. Данные, передаваемые между компонентами, должны быть защищены как от искажения, так и от просмотра третьими сторонами.

Эффективность. В узком смысле применительно к распределенным системам под эффективностью будет пониматься минимизациянакладных расходов, связанных с распределенным характером системы. Поскольку эффективность в данном узком смысле может противоречить безопасности, открытости и надежности системы, следует отметить, что требование эффективности в данном контексте является наименее приоритетным. Например, на поддержку логической целостности данных в распределенной системе могут тратиться значительные ресурсы времени и памяти, однако система с недостоверными данными вряд ли нужна пользователям. Желательным свойством промежуточной среды является возможность организации эффективного обмена данными, если взаимодействующие программные компоненты находятся на одном компьютере. Эффективная промежуточная среда должна иметь возможность организации их взаимодействия без затрагивания стека TCP/IP. Для этого могут использоваться системные сокеты (unix sockets) в POSIX системах или именованные каналы (named pipes).

Устойчивость распределенной системы связана с понятием масштабируемости, но не эквивалентна ему. Допустим, система использует набор обрабатывающих запросы серверов и один диспетчер запросов, который распределяет запросы пользователей между серверами. Такая система может считаться достаточно хорошо масштабируемой, однако диспетчер является уязвимой точкой такой системы. С другой стороны, система с единственным сервером может быть устойчива, если существует механизм его автоматической замены в случае выхода его из строя, однако она вряд ли относится к классу хорошо масштабируемых систем. На практике достаточно часто встречаются распределенные системы, не удовлетворяющие данным требованиям: например, любая система с уникальным сервером БД, реализованным в виде единственного компьютера, имеет уникальную точку сбоя. Выполнение требований устойчивости и масштабируемости обычно связано с некоторыми дополнительным расходами, что на практике может быть не всегда целесообразно. Однако используемые при построении распределенных систем технологии должны допускать принципиальную возможность создания устойчивых и высоко масштабируемых систем.

Классическим примером системы, в значительной мере отвечающей всем представленным выше требованиям, является система преобразования символьных имен в сетевые IP-адреса (DNS). Система имен – организованная иерархически распределенная система, с дублированием всех функций между двумя и более серверами (рис. 1.8).

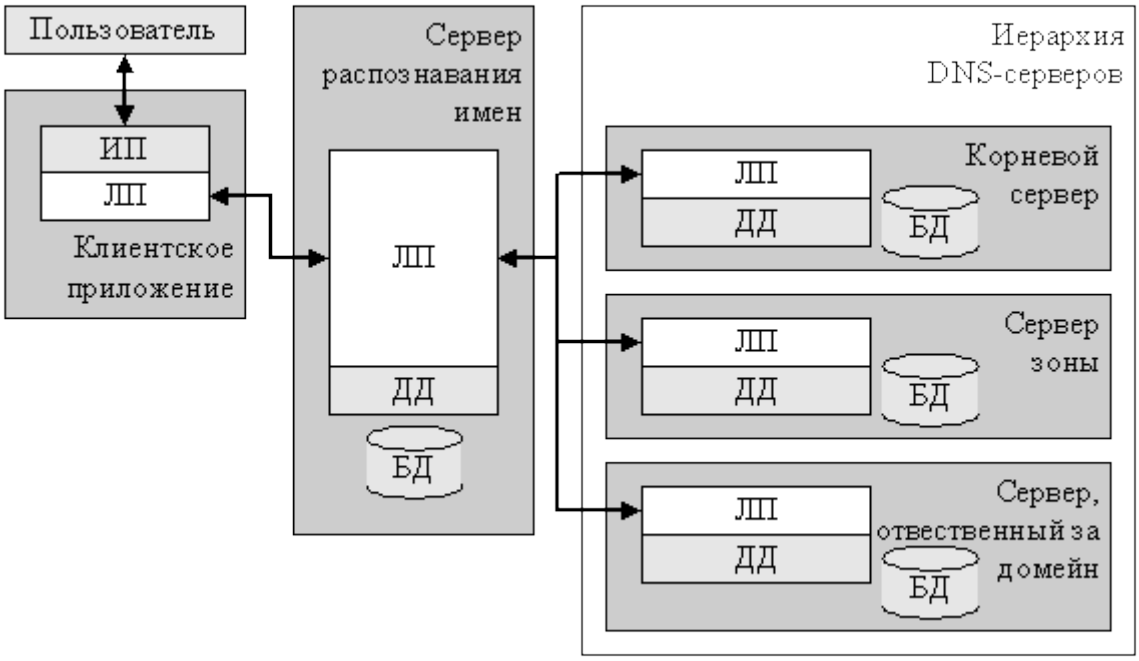


Рис. 1.8. Система DNS

Запрос пользователя на преобразование имени (например, w3c.org) в сетевой адрес передается серверу распознавания имен поставщика услуг интернета. Сервер распознавания имен по очереди опрашивает серверы из иерархии службы имен. Опрос начинается с корневых серверов, который возвращает адреса серверов, ответственных за зону домена. Затем опрашивается сервер, ответственный за зону (в данном случае – .org), возвращающий адреса серверов, ответственных за домен второго уровня, и так далее. Серверы имен кешируют информацию о соответствии имен и адресов для уменьшения нагрузки на систему. Программное обеспечение на компьютере пользователя обычно имеет возможность соединиться с как минимум двумя различными серверами распознавания имен.

Тем не менее, и в системе распознавания имен не все требования к распределенным системам выполнены. В частности, она не содержит каких-либо явных механизмов обеспечения безопасности. Это приводит к регулярным атакам на серверы имен в надежде вывести их из строя, например, большим количеством запросов.

Тема 9. Перспективные модели баз данных

Хранилища данных

Как уже неоднократно отмечалось, технологии баз данных предназначены, как правило, для решения текущих задач обработки данных организации. В базу данных постоянно вносятся изменения, то есть база данных отражает моментальный снимок определенной области деятельности предприятия. Для эффективного принятия решений руководством при управлении организацией важно не только знать текущее положение дел, но и иметь возможность анализировать динамику (изменение во времени) основных показателей, причем, зачастую из разных баз данных. Такую возможность дает технология так называемых хранилищ данных.

Приведем определение хранилища данных (Bill Inmon).

Хранилище данных – предметно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений.

Под предметной ориентированностью здесь понимается ориентированность на предметы (определенные группы данных), а не на конкретные приложения. Например, ориентация на данные о сотрудниках, а не только о расчете их заработной платы.

Под интегрированностью здесь понимается возможное объединение данных из разных источников (баз данных), имеющих разный формат и несогласованных.

Привязка ко времени предполагает, что для всех данных указан момент или промежуток времени, в который они корректны.

Данные в хранилище не изменяются, они лишь регулярно пополняются из оперативных баз данных.

Общая схема взаимодействия информационного хранилища и баз данных приводится на рис.

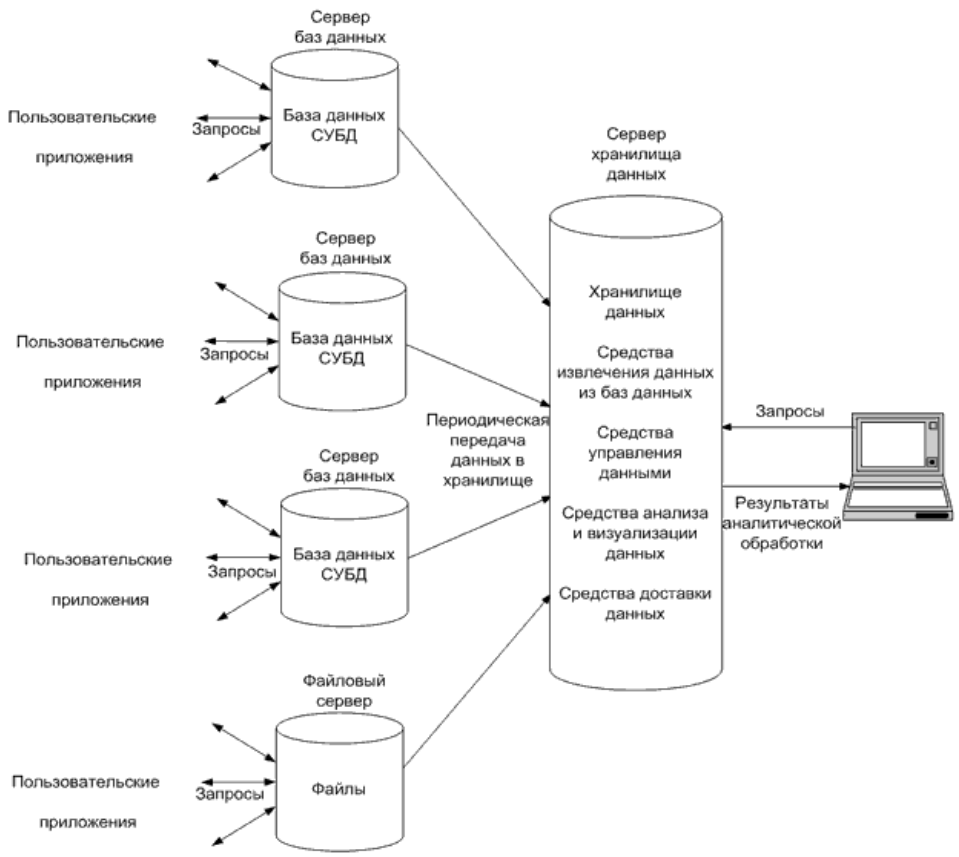


Рис. Схема организации работы хранилища данных

Еще раз подчеркнем, что основной целью хранилищ данных является бизнес-анализ или информационная поддержка принятия управленческих решений.

Для реализации всей необходимой обработки информации в соответствии с этой схемой необходимы следующие программные средства:

средства извлечения данных из баз данных;

средства управления данными хранилища (система управления базой данных хранилища);

средства анализа данных хранилища (используется OLAP-технология):

средства доставки данных;

средства визуализации результатов обработки для конечных пользователей.

Для работы соответствующих программных средств необходимо описание структуры содержимого информационного хранилища (метаописание).

Для самого общего случая, если данные берутся из баз данных, управляемых разными СУБД, из файлов разных типов, а данные разнородны, средства управления данными хранилища пока не созданы. Однако, если данные в информационное хранилище выбираются только из реляционных баз данных, то в качестве средств управления данными хранилища может быть взята мощная реляционная СУБД. Поэтому разработчики современных СУБД включают в состав программного обеспечения СУБД средства организации работы с хранилищами данных.

Рассмотрим в качестве примера возможности СУБД Microsoft SQL Server 2008 для организации хранилищ данных.

Microsoft SQL Server 2008 содержит в своем составе средства извлечения, преобразования и загрузки данных (SQL Server 2008 Integration Services), способные интегрировать данные из различных источников, проверять данные на допустимость и преобразовывать перед загрузкой в хранилище. Эти средства также способствуют перемещению данных, поддерживают текстовый анализ и нечеткий поиск. Нужно отметить также среду визуальной разработки (Business Intelligence Development Studio) для создания многомерных кубов, отчетов, пакетов извлечения, преобразования и загрузки данных.

Существенной особенностью хранилищ данных является их очень большой объем. Microsoft SQL Server 2008 как средство управления данными хранилища позволяет работать с большими объемами данных, причем для сокращения времени обработки предусмотрена поддержка параллельных вычислений (путем разделения таблиц и индексов на секции и обеспечение параллельной обработки секций). В системе предусмотрена возможность сжатия данных

(таблиц), что позволяет уменьшить физический размер таблиц и существенно сокращает время обмена между оперативной и внешней памятью.

В качестве средств анализа данных хранилища используется SQL Server 2008 Analysis Services, применяемый для построения многомерных кубов (многомерных моделей данных). Это средство содержит семь эффективных алгоритмов анализа данных с целью поддержки принятия управленческих решений, в том числе анализ тенденций и статистический анализ данных.

В качестве средств представления аналитических данных пользователям предлагается использовать средство генерации отчетов SQL Server 2008 Reporting Services.

Таким образом, Microsoft SQL Server 2008 является эффективным средством реализации хранилищ данных на основе реляционных баз данных.

Тема 10. Публикация баз данных в Интернет

Публикация баз данных в интернете — это размещение информации их баз данных на WEB-страницах в сети. Отметим, что такая публикация связана с решением следующих типичных задач, возникающих перед разработчиками современного программного обеспечения:
• Организация взаимосвязи СУБД, работающих на различных платформах.

• Построение информационных систем в сети Internet на основе многоуровненевой архитектуры БД (архитектура таких систем включает дополнительный уровень — WEB-сервер с модулями расширения серверной части, который и реализует возможность информационного обмена и публикации БД в глобальной сети)

• Построение локальных интранет-сетей на основе технологии публикации БД в Интернете.

• Использование в Internet информации из существующих локальных сетевых баз данных.

• Применение БД для упорядочивания информации.

• Поддержка языка SQL.

• Использование средств СУБД для обеспечения безопасности данных.

• Стандартизация пользовательского интерфейса на основе применения WEB-браузера.

Размещение информации из БД в Internet представляет собой новую информационную технологию, получившую широкое распространение в последнее время в связи с ростом популярности глобальной паутины.

В Internet вся информация размещается на WEB-страницах, для написания которых используется язык HTML (язык разметки текста) или его расширения, такие как DHTML и XML. В содержимое WEB-страницы может входить как и текстовая информация, так и графические изображения, ссылки на другие страницы и даже аудио и видео информация.

Для расширения возможностей WEB-сервера создаются соответственные программы расширения. При организации работы БД в сети часто используют следующие средства:

• Сценарии (JavaScript, JScript, VBScript).

• Апплеты и сервлеты, написанные на Java.

• Элементы управления ActiveX.

• Консольные программы, реализованные с поддержкой интерфейсов CGI.

• Программы, поддерживающие интерфейс WinCGI.

• Динамические библиотеки, поддерживающие ISAPI.

• Динамические страницы IDC/HTX.

• Активные серверные страницы ASP.

Дадим краткую характеристику этих средств.

Технологии публикации БД

• **Сценарии JavaScript, JScript, VBScript**

Сценарии, написанные на скриптовых языках, используют для динамического управления интерфейсными объектами WEB-документа. Все эти языки в подавлябщем своём большинстве интерпретируемые. Интерпритация осуществляется или браузером, или WEB-сервером. Скрипты рассматриваются как расширение языка HTML и могут включаться в тело документа. Часть скрипта выполняется в момент загрузки, а какие-то части — в ответ на действие пользователя. Выбор конкретного скриптового языка зависит в первую очередь от выбранного браузера.

• **Элементы управления ActiveX**

Элементы управления ActiveX представляют собой модули расширения, реализованные в виде DLL, которые могут быть использованы и на стороне клиента, и на стороне сервера. Механизм работы элементов управления позволят получать с их помощью неограниченный доступ к локальным ресурсам компьютера. Так как код ActiveX может содержать вирус, использование этих элементов чаще всего неоправданно.

• **Апплеты и сервлеты Java**

Апплеты Java применяются для создания динамически формируемого интерфейса пользователя. Язык Java является объектно-ориентированным языком. Однако возможности Java по доступу к локальным ресурсам сильно урезаны, что делает его безопасным для работы в сети. По мимо этого Java интерпретируема, такой механизм помимо безопасности гарантирует и целостность данных пользователя.

Сервлеты, в отличие от апплетов, выполняются на стороне сервера. В первую очередь сервлеты служат для обработки запросов, передаваемых от браузера.

• Интерфейсы CGI и WinCGI

Для создания модулей расширеня WEB-сервера часто используются интерфейсы: общий шлюзовой (CGI) или интерфейсы прикладного программирования (API).

CGI – стандартный протокол взаимодействия между WEB-сервером и модулем расширения, который используется для выполнения нестандартных функций. Такие модули могут использоваться для обработки получаемой пользователем информации.

Обмен информацией между CGI-приложением и сервером осуществляется через стандартный потоковый ввод-вывод, а передача параметров организуется через переменные окружения или URL-адреса.

Для запуска модуля, необходимо обратиться к нему через его URL. Для каждого такого запроса сервер запускает новую копию CGI-приложения.

Для среды Windows существует адаптированный вариант интерфейса — WinCGI. В нём управляющие параметры передаются через ini-файл. В остальном WinCGI повторяет принципы работы родительского интерфейса.

• Интерфейсы ISAPI/NSAPI

Более перспективными являются интерфейсы ISAPI/NSAPI, разработанные фирмами Microsoft и Netscape соответственно. Осуществление взаимодействия с сервером осуществляется при помощи специальных объектов request-response. Основное отличие от CGI в том, что при многопользовательском режиме работы не происходит излишней нагрузки сервера, потому что загрузка DLL выполняется только один раз при первом обращении.

• ASP, PHP и IDC/HTX-страницы

Это специальные типы страниц, используемые для динамического формирования страниц, содержащих информацию из БД.
IDC-страница содержит псевдоним (alias), запрос, идентификатор пользователя и пароль.

HTX-страница содержит шаблон, определяющий какую информацию и в каком формате передавать браузеру. ASP-страница содержит одновременно HTML-шаблон и SQL-запрос к БД. В ASP-страница используются средства JScript и объектная модель доступа к данным.

Протоколы передачи данных

В сети Internet используются серверы с различным программным обеспечением. Запросы от браузера поступают на WEB-сервер в соответствии с установленным протоколом обмена, например HTTP. Вообще-то в сети Internet поддерживается более десятка протоколов. Протокол HTTP предназначен для передачи между браузером и сервером. Соединение чаще всего обеспечивается на уровне протокола TCP/IP, но при работе с БД лучше использовать протокол IIOP, который умеет сохранять состояние.

Виды публикации

• Статическая публикация

Генерацию таких страниц выполняет обычное приложение Windows, имеющее доступ к БД. Этот способ используется только если публикуемая информация обновляется очень редко. Зато при поступлении запроса не требуется активное состояние сервера.

• Динамическая публикация

Используется в случаях, когда требуется получить доступ к БД, информация в которой часто обновляется. Таким способом работают Internet-магазины, авиакассы, да и вообще все системы реального времени.

Вывод

Информационные системы, построенные на WEB-приложениях используют многоуровневую архитектуру и позволяют воспользоваться всеми преимущества всемирной паутины. Однако разработчикам надо учитывать, что такая архитектура имеет недостатки:

— Совместимость браузеров.

— Разграничение прав доступа.

— Безопасность.

— Надёжность линий связи.

Требования к WEB-СУБД

— Защищённость доступа к ценным данным.

— Независимый способ подключения.
— Независимость работы от типа сервера и браузера.
— Открытость архитектуры.
— Поддержка COM+ и CORBA.
— Масштабируемость.
— Поддержка транзакций.
— Поддержка сеансов.
— Производительность.
— Минимальная требование к администрации.
Плюсы
— Простота.
— Независимость от ПО.
— Графический интерфейс.
— Стандартизация.
— Межплатформенная поддержка.
— Прозрачность.
— Масштабируемость.
Минусы
— Малая надёжность.
— Низкая скорость.
— Низкий уровень безопасности.
— Высокая стоимость.
— Трудность в определение масштаба.
— Ограничение html.
— Требования к пропускной способности.
— Несовершенная разработка.

Тема 11. Современные СУБД и их применение

Конечно, несмотря на всю их привлекательность, классические реляционные системы управления базами данных являются ограниченными. Они идеально походят для таких традиционных приложений, как системы резервирования билетов или мест в гостиницах, а также банковских систем, но их применение в системах автоматизации проектирования, интеллектуальных системах обучения и других системах, основанных на знаниях, часто является затруднительным. Это прежде всего связано с примитивностью структур данных, лежащих в основе реляционной модели данных. Плоские нормализованные отношения универсальны и теоретически достаточны для представления данных любой предметной области. Однако в нетрадиционных приложениях в базе данных появляются сотни, если не тысячи таблиц, над которыми постоянно выполняются дорогостоящие операции соединения, необходимые для воссоздания сложных структур данных, присущих предметной области.

Другим серьезным ограничением реляционных систем являются их относительно слабые возможности по части представления семантики приложения. Самое большее, что обеспечивают реляционные СУБД,- это возможность формулирования и поддержки ограничений целостности данных. Как мы отмечали в лекции 6, после проектирования реляционной базы данных многие знания проектировщика остаются зафиксированными в лучшем случае на бумаге по причине отсутствия в системе соответствующих выразительных средств.

Осознавая эти ограничения и недостатки реляционных систем, исследователи в области баз данных выполняют многочисленные проекты, основанные на идеях, выходящих за пределы реляционной модели данных. По всей видимости, какая-либо из этих работ станет основой систем баз данных будущего. Следует заметить, что тематика современных исследований, относящихся к базам данных, исключительно широка. В завершающей части курса мы приведем только короткий обзор наиболее важных направлений.

В этом разделе очень кратко рассматриваются основные направления исследований и разработок в области так называемых постреляционных систем, т.е. систем, относящихся к следующему поколению (хотя термин "next-generation DBMS" зарезервирован для некоторого подкласса современных систем).

Хотя отнесение СУБД к тому или иному классу в настоящее время может быть выполнено только условно (например, иногда объектно-ориентированную СУБД O2 относят к системам следующего поколения), можно отметить три направления в области СУБД следующего поколения. Чтобы не изобретать названий, будем обозначать их именами наиболее характерных СУБД.

1. *Направление Postgres*. Основная характеристика: максимальное следование (насколько это возможно с учетом новых требований) известным принципам организации СУБД (если не считать коренной переделки системы управления внешней памятью).

2. *Направление Exodus/Genesis*. Основная характеристика: создание собственно не системы, а генератора систем, наиболее полно соответствующих потребностям приложений. Решение достигается путем создания наборов модулей со стандартизованными интерфейсами, причем идея распространяется вплоть до самых базисовых слоев системы.

3. *Направление Starburst*. Основная характеристика: достижение расширяемости системы и ее приспосабливаемости к нуждам конкретных приложений путем использования стандартного механизма управления правилами. По сути дела, система представляет собой некоторый интерпретатор системы правил и набор модулей-действий, вызываемых в соответствии с этими правилами. Можно изменять наборы правил (существует специальный язык задания правил) или изменять действия, подставляя другие модули с тем же интерфейсом.

В целом можно сказать, что СУБД следующего поколения - это прямые наследники реляционных систем. Тем не менее, различные направления систем третьего поколения стоит рассмотреть отдельно, поскольку они обладают некоторыми разными характеристиками.

Одним из основных положений реляционной модели данных является требование нормализации отношений: поля кортежей могут содержать лишь атомарные значения. Для традиционных приложений реляционных СУБД - банковских систем, систем резервирования и т.д. - это вовсе не ограничение, а даже преимущество, позволяющее проектировать экономные по памяти БД с предельно понятной структурой. Запросы с соединениями в таких системах сравнительно редки, для динамической поддержки целостности используются соответствующие средства SQL.

Однако с появлением эффективных реляционных СУБД их стали пытаться использовать и в менее традиционных прикладных системах - САПР, системах искусственного интеллекта и т.д. Такие системы обычно оперируют сложно структурированными объектами, для реконструкции которых из плоских таблиц реляционной БД приходится выполнять запросы, почти всегда требующие соединения отношений. В соответствии с требованиями разработчиков нетрадиционных приложений появилось направление исследований баз сложных объектов. Основной смысл этого направления состоит в том, что в руки проектировщиков даются настолько же мощные и гибкие средства структуризации данных, как те, которые были присущи иерархическим и сетевым системам базам данных.

Однако важным отличием является то, что в системах баз данных, поддерживающих сложные объекты, сохраняется четкая граница между логическим и физическим представлениями таких объектов. В частности, для любого сложного объекта (произвольной сложности) должна обеспечиваться возможность перемещения или копирования его как единого целого из одной части базы данных в другую ее часть или даже в другую базу данных. Это очень обширная область исследований, в которой затрагиваются вопросы моделей данных, структур данных, языков запросов, управления транзакциями, журнализации и т.д. Во многом эта область соприкасается с областью объектно-ориентированных БД. (И в этой области настолько же плохо обстоят дела с теоретическим обоснованием.)

Близкое, но, вообще говоря, основанное на других принципах направление представлено системами баз данных, основанных на реляционной модели, в которой не обязательно поддерживается первая нормальная форма отношений. Напомним, что требование атомарности значений, которые могут храниться в элементах кортежей отношений, является базовым требованием классической реляционной модели. Приведение исходного табличного представления предметной области к "плоскому" виду является обязательным первым шагом в процессе проектирования реляционной базы данных на основе принципов нормализации. С другой стороны, абсолютно очевидно, что такое "уплощение" таблиц хотя и является необходимым условием получения неизбыточной и "правильной" схемы реляционной базы данных, в дальнейшем потенциально вызывает выполнение многочисленных соединений, наличие которых может свести на нет все преимущества "хорошей" схемы базы данных.

Так вот, в "ненормализованных" реляционных моделях данных допускается хранение в качестве элемента кортежа кортежей (записей), массивов (регулярных индексированных множеств данных), регулярных множеств элементарных данных, а также отношений. При этом такая вложенность может быть, по существу, неограниченной. Если внимательно продумать эти идеи, то станет понятно, что они приводят (только) к логически обособленным (от физического представления) возможностям иерархической модели данных. Но это уже не так уж и мало, если учесть, что к настоящему времени фактически полностью сформировано теоретическое основание реляционных баз данных с отказом от нормализации. Скорее всего, в этой теории все еще имеются темные места (они наличествуют даже в классической реляционной теории), но тем не менее большинство известных теоретических результатов реляционной теории уже распространено на ненормализованную модель, и

даже такой пурист реляционной модели, как Дейт, полагает возможным использование ограниченной и контролируемой реляционной модели в SQL-3.

Одной из наиболее известных СУБД третьего поколения является система Postgres, а создатель этой системы М.Стоунбрекер, по всей видимости, является вдохновителем всего направления. В Postgres реализованы многие интересные средства: поддерживается темпоральная модель хранения и доступа к данным (см. ниже) и в связи с этим абсолютно пересмотрен механизм журнализации изменений, откатов транзакций и восстановления БД после сбоев; обеспечивается мощный механизм ограничений целостности; поддерживаются ненормализованные отношения (работа в этом направлении началась еще в среде Ingres), хотя и довольно странным способом: в поле отношения может храниться динамически выполняемый запрос к БД.

Одно свойство системы Postgres сближает ее со свойствами объектно-ориентированных СУБД. В Postgres допускается хранение в полях отношений данных абстрактных, определяемых пользователями типов. Это обеспечивает возможность внедрения поведенческого аспекта в БД, т.е. решает ту же задачу, что и ООБД, хотя, конечно, семантические возможности модели данных Postgres существенно слабее, чем у объектно-ориентированных моделей данных. Основная разница состоит в том, что системы класса Postgres не предполагают наличия языка программирования, одинаково понимаемого как внешней системой программирования, так и системой управления базами данных. Если с использованием такой системы программирования определяются типы данных, хранимых в базе данных, то СУБД оказывается не в состоянии контролировать безопасность этих определений, т.е. то отсутствует гарантия, что при выполнении процедур абстрактных типов данных не будет разрушена сама база данных.

Заметим, что в середине 1995 г. компания Sun Microsystems объявила о выпуске нового продукта - языка и семейства интерпретаторов под названием Java. Язык Java является расширенным подмножеством языка Си++. Основные изменения касаются того, что язык является пооператорно интерпретируемым (в стиле языка Бейсик), а программы, написанные на языке Java, гарантированно безопасны (в частности, при выполнении любой программы не может быть поврежден интерпретатор). Для этого, в частности, из языка удалена арифметика над указателями. В то же время Java остается мощным объектно-ориентированным языком, включающим развитые средства определения абстрактных типов данных. Компания Sun продвигает язык Java с целью расширения возможностей службы Всемирной Паутины (World Wide Web) Internet (основная идея состоит в том, что из сервера WWW в клиенты передаются не данные, а объекты, методы которых запрограммированы на языке Java и интерпретируются на стороне клиента; этот подход, в частности, решает проблему нестандартизованного представления мультимедийной информации). Однако, как кажется, интерпретируемый и безопасный язык типа Java может быть успешно применен и в системах баз данных, допускающих хранение данных с типами, определенными пользователями.

5.2. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Практическое занятие №1
Создание БД.

Цели работы:

- познакомиться с основными понятиями базы данных;
- научиться создавать таблицу базы данных в режиме конструктора;
- освоить переход из режима конструктора в режим таблицы;
- освоить основные приемы заполнения и редактирования таблиц базы данных;
- познакомиться с простой сортировкой значений таблицы;
- познакомиться с поиском записей по образцу;
- научиться сохранять и загружать базу данных;
- научиться распечатывать таблицы

Задание:

- Продумать предметную область для реализации БД в библиотечно-информационной сфере.
- Обсудить и уточнить структуру будущей реляционной БД с преподавателем.
- Создать соответствующие таблицы в СУБД Access в различных режимах.
- Наполнить БД записями по согласованию с преподавателем.
- Познакомиться с поиском записей по образцу; научиться сохранять и загружать базу данных; научиться распечатывать таблицы.

Практическое занятие №2
Создание и использование запросов

Цели работы:

- закрепить навыки по созданию таблиц;
- продолжить знакомство с основными видами запросов;
- научиться создавать запросы: на обновление; на добавление; на удаление; на создание таблицы;
- научиться создавать перекрестные запросы.

Практическое занятие №3
Создание отчетов

Цели работы: научиться создавать отчеты.

Справочная информация.

Отчет - это способ вывода данных из базы данных на печать в том виде, в котором требуется пользователю, например в виде справок об обучении, экзаменационных ведомостей, таблиц, объединенных каким-либо признаком, и др.

С помощью отчета вы располагаете информацию на листе в удобном для вас виде с подходящим оформлением (цвет, рамки, выравнивание, шрифт и т. д.).

Отчет можно разработать: самостоятельно с помощью Конструктора; используя готовые варианты оформления (автоотчеты); с помощью Мастера отчетов.

5.3. СЕМИНАРСКИЕ ЗАНЯТИЯ

Тема 10. Публикация баз данных в Интернет

Тема11. Современные СУБД и их применение

5.4. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ

График СРС

Планирование СРС

по дисциплине:«Формирование баз данных»

Преподаватель:Скачков Д.Е., старший преподаватель

Очная форма обучения

№ п/п	Название раздела (темы) дисциплины	Виды СРС	Периодичность (сроки) контроля СРС	№ семестра	Время на изучение, выполнение задания
РАЗДЕЛ 1. ВЕДЕНИЕ В БАЗЫ ДАННЫХ					
1	Тема 1. Основные понятия теории баз данных	Составление конспекта		7	4
2	Тема 2. 3-х уровневая архитектура Ansi-Sparc	Составление конспекта		7	4
3	Тема 3. Реляционная модель данных	Составление конспекта		7	4
4	Тема 4. Проектирование БД методом нормализации	Составление конспекта; подготовка к практическому занятию		7	6
5	Тема 5. Проектирование БД с использованием семантических моделей	Составление конспекта; подготовка к практическому занятию		7	6
6	Тема 6.Язык SQL	Составление конспекта; подготовка к практическому занятию		7	6

РАЗДЕЛ 2. СОВРЕМЕННЫЕ БД					
7	Тема 7. Объектно-ориентированные базы данных	Составление конспекта; подготовка к практическому занятию		8	6
8	Тема 8. Распределенные базы данных и системы клиент-сервер	Составление конспекта; подготовка к практическому занятию		8	6
9	Тема 9. Перспективные модели баз данных	Составление конспекта		8	6
10	Тема 10. Публикация баз данных в Интернет	подготовка к практическому занятию		8	6
11	Тема11. Современные СУБД и их применение	подготовка к практическому занятию		8	6
				Итого по дисциплине	**60**

Курс предусматривает также самостоятельное его изучение. Формы самостоятельной работы:
- выполнение заданий, связанных с закреплением пройденного материала;
- подготовка к выполнению практических заданий;
- подготовка к зачетам.

Формы и методы контроля самостоятельной работы:
- проверка практических работ;
- реферат;

6. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ, ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ИТОГАМ ОСВОЕНИЯ ДИСЦИПЛИНЫ И УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Вопросы экзамена;
Тематика рефератов;
Задания для самостоятельной работы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

- Базы данных. СУБД. Классификация.
- Типология БД. Документальные БД. Фактографические БД.
- Типология БД. Гипертекстовые и мультимедийные БД. Объектно-ориентированные БД.
- Типология БД. Распределенные БД. Коммерческие БД.
- Иерархическая и сетевая модели данных.
- Элементы реляционной модели данных.
- Реляционное исчисление. Организация процессов обработки данных в БД. Ограничения целостности.
- Организация процессов обработки данных в БД. Индексирование таблиц. Связывание таблиц.
- Постреляционная и многомерная модель данных.
- Реляционная алгебра (объединение, пересечение, вычитание, произведение, выборка).
- Реляционная алгебра (проекция, деление, соединение).
- Язык SQL.
- Проблемы проектирования реляционных БД.
- Принципы построения БД. Нормальные формы: 1НФ, 2НФ, 3НФ.
- Нормальные формы: НФБК, 4НФ, 5НФ.
- Принципы построения БД. Метод «Сущность-связь».
- Пример разработки ER-модели.
- Хранение отношений. Организация индексов.
- Транзакции. Сериализация транзакций.
- Системы управления БД следующего поколения.

- XML. XML-серверы.
- Стилевые таблицы XSL.
- Жизненный цикл БД. Модели жизненного цикла ПО.
- Модели структурного проектирования. Метод структурного анализа и проектирования.
- Проблема создания и сжатия больших информационных массивов, информационных хранилищ и складов данных. Сжатие без потерь в реляционных СУБД.

ВОПРОСЫ ЗАЧЕТА

1. Базы данных. СУБД. Классификация.
2. Типология БД. Документальные БД. Фактографические БД.
3. Типология БД. Гипертекстовые и мультимедийные БД. Объектно-ориентированные БД.
4. Типология БД. Распределенные БД. Коммерческие БД.
5. Иерархическая и сетевая модели данных.
6. Элементы реляционной модели данных.
7. Реляционное исчисление. Организация процессов обработки данных в БД. Ограничения целостности.
8. Организация процессов обработки данных в БД. Индексирование таблиц. Связывание таблиц.
9. Постреляционная и многомерная модель данных.
10. Реляционная алгебра (объединение, пересечение, вычитание, произведение, выборка).
11. Реляционная алгебра (проекция, деление, соединение).
12. Язык SQL.
13. Проблемы проектирования реляционных БД.
14. Принципы построения БД. Нормальные формы: 1НФ, 2НФ, 3НФ.
15. Нормальные формы: НФБК, 4НФ, 5НФ.
16. Принципы построения БД. Метод «Сущность-связь».
17. Пример разработки ER-модели.
18. Хранение отношений. Организация индексов.
19. Транзакции. Сериализация транзакций.
20. Системы управления БД следующего поколения.
21. XML. XML-серверы.
22. Стилевые таблицы XSL.
23. Жизненный цикл БД. Модели жизненного цикла ПО.
24. Модели структурного проектирования. Метод структурного анализа и проектирования.
25. Проблема создания и сжатия больших информационных массивов, информационных хранилищ и складов данных. Сжатие без потерь в реляционных СУБД.
26. Объектно-ориентированные базы данных
27. Распределенные базы данных и системы клиент-сервер
28. Перспективные модели баз данных

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

7.1. РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

7.1.1. ОСНОВНАЯ ЛИТЕРАТУРА

1. Грошев, А.С. Информатика : учебник для вузов / А.С. Грошев. - Москва ; Берлин : Директ-Медиа, 2015. - 484 с. : ил. - Библиогр.: с. 466. - ISBN 978-5-4475-5064-6 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428591 (02.10.2020)
2. Гураков, А.В. Информатика: Введение в Microsoft Office : учебное пособие / А.В. Гураков, А.А. Лазичев ; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). - Томск : Эль Контент, 2012. - 120 с. : ил. - ISBN 978-5-4332-0033-3 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=208646 (02.10.2020).
3. Ефимова, И.Ю. Компьютерное моделирование: сборник практических работ / И.Ю. Ефимова, Т.Ю. Варфоломеева. - 2-е изд., стер. - Москва : Издательство «Флинта», 2014. - 68 с. : табл., граф., ил. - Библиогр. в кн. - ISBN 978-5-9765-2039-4 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=482123 (02.05.2021).
4. Исакова, А.И. Информационные технологии : учебное пособие / А.И. Исакова, М.Н. Исаков ; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). - Томск : Эль Контент, 2012. - 174 с. : ил.,табл.,

схем. - ISBN 978-5-4332-0036-4 ; То же [Электронный ресурс]. -
URL: http://biblioclub.ru/index.php?page=book&id=208647 (02.06.2020).
5. Соболева, М.Л. Информационные технологии : лабораторный практикум / М.Л. Соболева, А.С. Алфимова. - М. : Прометей, 2012. - 48 с. : схем., ил., табл. - Библиогр. в кн. - ISBN 978-5-7042-2338-2 ; То же [Электронный ресурс]. -
URL: http://biblioclub.ru/index.php?page=book&id=437357 (02.06.2017).
6. Информационные технологии : учебник / Ю.Ю. Громов, И.В. Дидрих, О.Г. Иванова, и др. ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2015. - 260 с. : ил., табл., схем. - Библиогр. в кн.. - ISBN 978-5-8265-1428-3 ; То же [Электронный ресурс]. -
URL: http://biblioclub.ru/index.php?page=book&id=444641 (02.06.2020).
7. Лыткина, Е.А. Применение информационных технологий : учебное пособие / Е.А. Лыткина ; Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северный (Арктический) федеральный университет имени М. В. Ломоносова», Министерство образования и науки Российской Федерации. - Архангельск : САФУ, 2015. - 91 с. - ISBN 978-5-261-01049-4 ; То же [Электронный ресурс]. -
URL: http://biblioclub.ru/index.php?page=book&id=436329 (02.06.2020).
8. Красильникова, В.А. Информационные и коммуникационные технологии в образовании : учебное пособие / В.А. Красильникова. - М. : Директ-Медиа, 2013. - 231 с. : ил.,табл., схем. - ISBN 978-5-4458-3000-9 ; То же [Электронный ресурс]. -
URL: http://biblioclub.ru/index.php?page=book&id=209292 (02.06.2020).
9. Информационные Web-технологии / Ю. Громов, О.Г. Иванова, Н.Г. Шахов, В.Г. Однолько ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2014. - 96 с. : ил. - Библиогр. в кн. - ISBN 978-5-8265-1365-1 ; То же [Электронный ресурс]. -
URL: http://biblioclub.ru/index.php?page=book&id=277935 (02.06.2020).
10. Майстренко, Н.В. Мультимедийные технологии в информационных системах : учебное пособие / Н.В. Майстренко, А.В. Майстренко ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2015. - 82 с. : ил., схем. - Библ. в кн. - ISBN 978-5-8265-1478-8 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=444959 (02.06.2017).
11. Красильникова, В. Использование информационных и коммуникационных технологий в образовании : учебное пособие / В. Красильникова ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Оренбургский государственный университет». - 2-е изд. перераб. и дополн. - Оренбург : ОГУ, 2012. - 292 с. ; То же [Электронный ресурс]. -
URL: http://biblioclub.ru/index.php?page=book&id=259225 (02.06.2020).

7.1.2. ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

1. Агальцов, В.П. Базы данных. В 2-х т. Т. 2. Распределенные и удаленные базы данных: Учебник / В.П. Агальцов. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. - 272 с.
2. Агальцов, В.П. Базы данных. В 2-х т.Т. 1. Локальные базы данных: Учебник / В.П. Агальцов. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. - 352 с.
3. Голицына, О.Л. Базы данных / О.Л. Голицына, Н.В. Максимов, И.И. Попов. - М.: Форум, 2004. - 352 с.
4. Голицына, О.Л. Базы данных: Учебное пособие / О.Л. Голицына, Н.В. Максимов, И.И. Попов. - М.: Форум, 2012. - 400 с.
5. Карпова, И.П. Базы данных: Учебное пособие / И.П. Карпова. - СПб.: Питер, 2013. - 240 с.
6. Кириллов, В.В. Введение в реляционные базы данных.Введение в реляционные базы данных / В.В. Кириллов, Г.Ю. Громов. - СПб.: БХВ-Петербург, 2012. - 464 с.
7. Кошелев, В.Е. Базы данных в ACCESS 2007: Эффективное использование / В.Е. Кошелев. - М.: Бином-Пресс, 2009. - 592 с.
8. Кузин, А.В. Базы данных: Учебное пособие для студ. высш. учеб. заведений / А.В. Кузин, С.В. Левонисова. - М.: ИЦ Академия, 2012. - 320 с.
9. Ливена, С.В. Практика увольнений за прогул. По материалам базы данных "Пакет кадровика" / С.В. Ливена. - М.: ИНФРА-М, 2008. - 51 с.
10. Пирогов, В.Ю. Информационные системы и базы данных: организация и проектирование: Учебное пособие / В.Ю. Пирогов. - СПб.: БХВ-Петербург, 2009. - 528 с.

11. Советов, Б.Я. Базы данных: теория и практика: Учебник для бакалавров / Б.Я. Советов, В.В. Цехановский, В.Д. Чертовской. - М.: Юрайт, 2013. - 463 с.
12. Фуфаев, Э.В. Базы данных: Учебное пособие для студентов учреждений среднего профессионального образования / Э.В. Фуфаев, Д.Э. Фуфаев. - М.: ИЦ Академия, 2012. - 320 с.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ КУРСА ДЛЯ ПРЕПОДАВАТЕЛЯ

При подготовке к занятию преподаватель, ведущий семинарские занятия, должен внимательно ознакомиться с учебно-методическим комплексом по дисциплине и уточнить план его проведения. Более того необходимо ознакомиться с новыми публикациями по теме семинара и дополнительной литературы по вопросам плана занятия. Оказывать методическую помощь студентам в подготовке докладов и рефератов. В ходе семинара во вступительном слове преподаватель должен раскрыть теоретическую и практическую значимость темы семинарского занятия, определить порядок его проведения, время на обсуждение каждого учебного вопроса. Дать возможность выступить всем желающим, а также предложить выступить тем студентам, которые по тем или иным причинам пропустили лекционное занятие или проявляют пассивность. Целесообразно в ходе обсуждения учебных вопросов задавать выступающим и аудитории дополнительные и уточняющие вопросы с целью выяснения их позиций по существу обсуждаемых проблем. Поощрять выступления с места в виде кратких дополнений и постановки вопросов выступающим и преподавателю. Для наглядности и закрепления изучаемого материала преподаватель может использовать таблицы, схемы и т.д. В заключительной части семинарского занятия следует подвести его итоги: дать объективную оценку выступлений каждого студента и учебной группы в целом. Раскрыть положительные стороны и недостатки проведенного семинарского занятия. Ответить на вопросы студентов. Назвать тему очередного занятия. К тому же преподаватель обязан проводить групповые и индивидуальные консультации студентов по вопросам, возникающим у студентов в ходе их подготовки к текущей и промежуточной аттестации, рекомендовать в помощь учебные и другие материалы, а также справочную литературу. Освоение дисциплины предполагает использование как традиционных (лекции, практические занятия с использованием методических материалов), так и инновационных образовательных технологий с использованием в учебном процессе интерактивных форм проведения занятий. Семинарские занятия в традиционной форме проводятся в соответствии с утвержденной учебно-методической разработкой для проведения семинарских занятий и самостоятельной работы студентов по соответствующей дисциплине. Каждая тема разработки содержит перечень основных вопросов для обсуждения, контрольные вопросы темы, задания для самостоятельной работы и список литературы, рекомендуемой к изучению. Если занятие проводится в интерактивной форме, преподаватель должен обсудить в группе на предыдущем занятии план проведения последующего семинара в интерактивной форме. При этом необходимо распределить задания, темы презентаций, дать рекомендации студентам по подготовке и проведению занятия в интерактивной форме. В процессе обучения необходимо обращать внимание в первую очередь на те методы, при которых слушатели идентифицируют себя с учебным материалом, включаются в изучаемую ситуацию, побуждаются к активным действиям, переживают состояние успеха и соответственно мотивируют свое поведение. Всем этим требованиям в наибольшей степени отвечают интерактивные методы обучения. Учебный процесс, опирающийся на использование интерактивных методов обучения, организуется с учетом включенности в процесс познания всех студентов группы без исключения. Совместная деятельность означает, что каждый вносит свой особый индивидуальный вклад, в ходе работы идет обмен знаниями, идеями, способами деятельности. Организуются индивидуальная, парная и групповая работа, используется проектная работа, осуществляется работа с документами и различными источниками информации.

Интерактивные методы основаны на принципах взаимодействия, активности обучаемых, опоре на групповой опыт, обязательной обратной связи. Создается среда образовательного общения, которая характеризуется открытостью, взаимодействием участников, равенством их аргументов, накоплением совместного знания, возможностью взаимной оценки и контроля. Ведущий преподаватель вместе с новыми знаниями ведет участников обучения к самостоятельному поиску. Активность преподавателя уступает место активности студентов, его задачей становится создание условий для их инициативы. Преподаватель отказывается от роли своеобразного фильтра, пропускающего через себя учебную информацию, и выполняет функцию помощника в работе, одного из источников информации.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ КУРСА ДЛЯ ОБУЧАЮЩЕГОСЯ

Успешное освоение материала курса возможно лишь при систематической работе в соответствии с РПД. Поможет в этом и серьезное изучение ряда базовых дисциплин Курс дает материал обобщающего характера, он конкретизируется в других дисциплинах. Для овладения понятийным аппаратом важно не только знание терминологических стандартов, но их оценка специалистами. Важно отслеживать изменения в них по печатным и электронным источникам. Освоение сложного курса в последнее время облегчено изданием учебной литературы разных жанров. Она указана в списках основной и дополнительной литературы. Вместе с тем, разнообразие научных концепций и подходов к содержанию учебной дисциплины усиливает ориентирующую роль лекций и занятий, проводимых преподавателем. При подготовке к любым видам занятий, читая и конспектируя источники, необходимо выделять спорные моменты, противоположные точки зрения и др. Самостоятельная работа, как аудиторная, так и внеаудиторная, осуществляется в виде подготовки к семинарам и практическим занятиям, выполнения домашних заданий. Обязательно использование новых информационных технологий: поиск определений в сети, мониторинг отраслевого документального потока. При изучении курса необходимы:

1. подготовка к семинарским занятиям;
2. подготовка к практическим занятиям;
3. создание собственного «банка данных», включающего:

а) дайджесты к семинарам;

б) «тезаурус» (словарь основных терминов предметной области);

в) структурно-логические схемы и таблицы;

г) выполнение аудиторных практических занятий и домашних заданий.

Подготовка к лекции не нужна. Подготовка к семинарским занятиям заключается в освоении теоретического материала по рекомендуемой литературе и конспектам лекций. После выполнения задания обсуждаются результаты. Подготовка к зачету должна быть регулярной. Она начинается с первого занятия (общее знакомство с ресурсной базой дисциплины, в том числе ее методическим обеспечением; информирование о формах контроля) и завершается подготовкой к тестированию - повторением материала дисциплины

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ ОБУЧАЮЩИХСЯ

Обучающийся, в ходе выполнения самостоятельной работы должен проявить способность к самостоятельному поиску в русле выбранной проблематики; умение находить и использовать нужную информацию; показать умение строить научное развернутое и аргументированное высказывание.

При изучении материала необходимо наличие требуемых текстов для рассмотрения. Для достижения четкости и структурированности работы студент должен фиксировать выполнение самостоятельных заданий и оформлять записи в рабочих тетрадях.

Для успешного освоения материала профессиональных статей и составления грамотного конспекта необходимо сначала внимательно прочитать статью или все статьи, выделить основные положения и только после этого приступить к конспектированию. Конспект не должен превращаться в механическое «переписывание», в конспекте нужно кратко и сжато отразить основные концепции статьи. Самый лучший конспект – тезисы, которые являются результатом глубокой проработки материала.

Одной из важных форм повышения качества подготовки студентов является чётко организованная самостоятельная работа, которая протекает во всех видах учебной работы и в самых различных формах. Задача преподавателя состоит в том, чтобы активизировать эту деятельность у студентов и управлять ею.

Одной из форм самостоятельной работы студентов является подготовка к семинарским занятиям. Она включает в себя: отработку лекционного материала, подготовку вопросов, вынесенных на самостоятельное изучение, проработку рекомендованной литературы, конспектирование материалов учебников и статей.

Обсуждение вопросов на семинарском занятии позволит студентам научиться чётко и логично излагать свои мысли, вести дискуссию, аргументировано защищать свою точку зрения.

В рамках изучения курса «Формирование БД» особым видом самостоятельной работы является работа творческого характера. Она может быть представлена в форме исследований, докладов, рефератов, контрольных работ. Эта работа носит особый характер, она помогает овладеть методами поисковой деятельности, обработки, обобщения, анализа информации. Она способствует проявлению индивидуальности, формированию творческого мышления, активизации интеллектуального потенциала, углублению и закреплению полученных при обучении знаний, расширению общего кругозора.

Каждая тема курса сопровождается списком обязательной и дополнительной литературы, а также вопросами и заданиями для самостоятельной работы. Студент должен овладеть умениями и навыками самостоятельной работы с предлагаемыми источниками, продемонстрировать способность самостоятельно сформулировать проблему и правильно интерпретируя материал делать собственные выводы.

При изучении курса «Формирование БД» процесс развития самостоятельной работы студентов должен происходить систематически, что будет способствовать формированию способности к самообразования, без которого выпускник не сможет стать конкурентоспособным специалистом

7.2. СРЕДСТВА ОБЕСПЕЧЕНИЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ

7.2.1. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ И МАТЕРИАЛЫ ПО ВИДАМ ЗАНЯТИЙ

1. Скачков Д.Е. Информационные технологии: методические указания к выполнению практических работ по дисциплине "Информационные технологии" для специальностей 070503.65 "Музейное дело и охрана памятников", 071201.65 "Библиотечно-информационная деятельность" / Д.Е. Скачков. – Смоленск: СГИИ, 2012. – 53 с

2. Васильева, Н.Н., Скачков, Д.Е. Web-дизайн: методические указания к выполнению практических работ по дисциплине «Дизайн культурной среды: web-дизайн» для направления 071800.62 «Социально-культурная деятельность», профиль «Менеджмент социально-культурной деятельности» /Н.Н.Васильева, Д.Е.Скачков. – Смоленск: СГИИ, 2014. – 68 с.

7.2.2. ИНФОРМАЦИОННО-ПРОГРАММНЫЕ СРЕДСТВА

1. Персональные компьютеры с доступом к сети Интернет.
2. Программное обеспечение: пакет программ MicrosoftOffice (MSWord, MSExcel, MSACCESS), браузеры (Chrome, IE, Яндекс браузер).

Интернет-ресурсы:

- CITforum– http://www.citforum.ru/
- Электронная библиотека материалов в области информационно-коммуникационных технологий – http://www.ict.edu.ru/lib
- Институт систем информатики им. А.П.Ершова (ИСИ СО РАН) – http://www.iis.nsk.su/
- Современные технологии автоматизации – http://www.cta.ru/

- Афинотека – http://athena.vvsu.ru/
- Библиотека братьев Фроловых – http://www.frolov-lib.ru/
- Библиотека электронных книг по информатике и компьютерным технологиям – http://eruditus.name/kopilka.html
- Бесплатная электронная библиотека Чувашского государственного университета им. И. Н. Ульянова– http://by-chgu.ru/category/informatika
- Перспективные информационные технологии и интеллектуальные системы – http://pitis.tsure.ru/
- WwwMASTER.ru– http://www.wwwmaster.ru/
- Библиотека учебников компьютерной тематики – http://biblioteka.net.ru/
- Библиотека компьютерной литературы – http://www.netbook.perm.ru/komputer.html

8. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

8.1. СПЕЦИАЛИЗИРОВАННЫЕ АУДИТОРИИ

Учебная аудитория для проведения практических (лекционных, семинарских) занятий – компьютерный класс; помещения для самостоятельной работы студентов; библиотека с читальным залом.

8.2. УЧЕБНО-ЛАБОРАТОРНОЕ ОБОРУДОВАНИЕ

Аудитория, укомплектованная учебной мебелью, техническими средствами, наглядными пособиями, литературой; компьютерная и множительная техника с подключением к сети Интернет.

ГЛОССАРИЙ

Администратор базы данных – лицо или группа лиц, реализующие управление базой данных.
Атрибут– поименованная характеристика сущности, принимающая значение из некоторого множества.
База данных – даталогическое представление информационной модели предметной области.
Данные – информация, фиксированная в определенной форме, пригодная для последующей обработки и хранения.
Домен – множество значений элементов данных одного типа.
Защита данных – предотвращение доступа к ним со стороны несанкционированных пользователей.
“Золотое правило” – ни одна их операций изменения не имеет права переводить переменную-отношение в состояние, нарушающее ее собственный предикат.
Ключ записи – один или несколько атрибутов, которые однозначно определяют объект.
Первичный ключ– идентификатор объекта.
ER-модель– модель “сущность-связь”. Операторы ER-модели являются операторами реляционной алгебры.
Метаданные – “данные о данных”, называемые иногда дескрипторами.
Реляционная алгебра состоит из набора операторов, использующих отношения в качестве операндов и возвращающих отношения в качестве результата. Она состоит из восьми операторов, составляющих две группы. Первая группа – традиционные операции над множествами: объединение, пересечение, разность, декартово произведение. Вторая группа – специальные реляционные операции: выборка, проекция, соединение, деление.
Реляционная база данных – база данных, в которой данные представлены в виде таблиц.
Реляционная модель – абстрактная теория данных, основанная на некоторых положениях теории множеств и логики предикатов. Принципы реляционной модели были изначально предложены в 1969 и 1970 годах Е.Ф.Коддом.
Связь– один из конструктивных элементов модели “сущность-связь” (ER-модели). Связь выступает в модели в качестве средства, с помощью которого представляются отношения между сущностями.
Сущность– абстракция данных или собирательное понятие об объекте, процессе или явлении, о которых необходимо хранить информацию в системе.
Транзакция- логическая единица работы, обычно включающая несколько операций базы данных, в частности, несколько операций изменения. Каждая транзакция содержит несколько агентов, где под агентом подразумевается процесс, который выполняется для данной транзакции на отдельном узле.
Управление транзакциями подразделяется на управление восстановлением и управление параллельностью обработки.
Фрагментация – разбиение таблиц на разделы.
Хранилище данных – база с данными для поддержки принятия решений.

Целостность данных – термин, используемый для описания точности и корректности хранящейся в базе информации. Поддержание целостности данных подразумевает предотвращение их разрушения при доступе со стороны санкционированных пользователей.

Язык описания данных – язык определения данных (data definition language – DDL), который поддерживает определения или объявления объектов базы данных.

Язык манипулирования данными – язык обработки данных (data manipulation language – DML), который поддерживает операции с объектами базы данных и их обработку. Операторами манипулирования данными языка SQL являются: SELECT, INSERT, UPDATE, DELETE.

Язык SQL– Международный стандарт языка баз данных, принятый в 1992г. и используемый для работы с реляционными базами данных. Он поддерживает следующие встроенные типы данных: CHARACTER INTEGER DATE, BIT SMALLINT TIME, NUMERIC FLOAT TIMESTAMP, DECIMAL INTERVAL

ЛИСТ СОГЛАСОВАНИЯ РАБОЧЕЙ ПРОГРАММЫ ДИСЦИПЛИНЫ

Рабочая программа дисциплины ***Формирование баз данных*** разработана в соответствии с федеральным государственным образовательным стандартом высшего образования по направлению ***51.03.06 Библиотечно-информационная деятельность,*** направленности ***Библиотечно-информационное обеспечение потребителей информации*** (утвержден ***Приказом Минобрнауки России № 1182 от 06.12.2017 г. с изменениями и дополнениями от 26.11.2020 г., 08.02.2021г.***), учебным планом института по этому же направлению, утвержденным Ученым советом ***30.06.2021г., Протокол № 7***, с учетом основной профессиональной образовательной программы (утверждена ***30.06.2021 г.***).

Автор программы – ***Скачков Д.Е., ст. преподаватель***

Рабочая программа рассмотрена и одобрена на заседании кафедры ***библиотечно-информационной деятельности и музеологии*** (рецензент – **_____,** протокол №___ от «____» _____ 20___г.)

Заведующий кафедрой _____ Е.С. Мертенс _____
(подпись) (И.О.Фамилия) (дата)

Согласовано:
Начальник УМУ _____ В.В.Азарова _____