МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
Национальный аэрокосмический университет им. Н.Е. Жуковского
«Харьковский авиационный институт»

И.В. Шевченко

ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ: ТЕОРИЯ И ПРАКТИКА

Учебное пособие

Харьков «ХАИ» 2018

УДК 004.65 (000.0)
ББК 00.00я00
Ш37

Наведено два підходи до проектування реляційних баз даних. Перший підхід – класичний і здійснюється в термінах реляційної моделі даних методом послідовних наближень до задовільного набору схем відносин, тобто являє собою процес нормалізації схем відносин. Другий підхід базується на побудові концептуальної схеми бази даних в термінах семантичного моделювання предметної області з наступним перетворенням її в реляційну схему (логічну та фізичну моделі даних) за спеціальною методикою, в якій достатньо чітко висвітлено всі етапи такого перетворення.

Подано велику кількість прикладів, які полегшують розуміння теоретичного матеріалу.

Для студентів всіх форм навчання напрямку підготовки 6.121 – «Інженерія програмного забезпечення», а також для всіх, хто бажає вивчити курс «Бази даних».

Рецензенты:

Шевченко, И.В.

Ш37 Основы проектирования баз данных [Текст]: учеб. пособие / И.В. Шевченко. – Х. : Нац. аэрокосм. ун-т им. Н.Е. Жуковского «Харьк. авиац. ин-т», 2018. – 87 с.

ISBN 000-000-000-000-0

Приведены два подхода к проектированию реляционных баз данных. Первый подход является классическим и осуществляется в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений, т.е. представляет собой процесс нормализации схем отношений. Второй подход базируется на построении концептуальной схемы базы данных в терминах семантического моделирования предметной области с последующим преобразованием ее в реляционную схему (логическую и физическую модели данных) по специальной методике, в которой достаточно четко оговорены все этапы такого преобразования.

Представлено большое количество примеров, облегчающих понимание теоретического материала.

Для студентов всех форм обучения направления подготовки 6.121 – «Инженерия программного обеспечения», а также для всех желающих изучить курс «Базы данных».

Ил. 28. Табл. 7. Библиогр.: 15 назв.

УДК 004.65 (000.0)
ББК 00.я00

«Харьковский авиационный институт», 2018

ISBN 000-000-000-000-0

ВВЕДЕНИЕ

В классической теории баз данных модель данных есть формальная теория представления и обработки данных в системе управления базами данных (СУБД), которая включает по крайней мере, три аспекта:

1) аспект структуры: методы описания типов и логических структур данных в базе данных;

2) аспект манипуляции: методы манипулирования данными;

3) аспект целостности: методы описания и поддержки целостности базы данных.

Аспект структуры определяет, что из себя логически представляет база данных, аспект целостности определяет средства описаний корректных состояний базы данных, аспект манипуляции определяет способы перехода между состояниями базы данных (то есть способы модификации данных) и способы извлечения данных из базы данных.

Модель данных − это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы − поведение данных.

Каждая БД и СУБД строится на основе некоторой явной или неявной модели данных. Все СУБД, построенные на одной и той же модели данных, относят к одному типу. Например, основой реляционных СУБД является реляционная модель данных, сетевых СУБД − сетевая модель данных, иерархических СУБД − иерархическая модель данных и т.д.

В литературе [1-3] иногда встречается использование термина «модель данных» в смысле «схема базы данных» («модель базы данных»). Такое использование является неверным, на что указывают многие авторитетные специалисты, в том числе К.Дж. Дейт, М.Р. Когаловский, С.Д. Кузнецов. Модель данных есть теория, или инструмент моделирования, в то время как модель базы данных (схема базы данных) есть результат моделирования. Соотношение между этими понятиями аналогично соотношению между языком программирования и конкретной программой на этом языке.

Проведем краткий обзор наиболее распространенных моделей БД: иерархической, сетевой, реляционной и объектно-ориентированной.

Иерархическая модель представляет собой древовидную структуру, в этом случае каждая запись связана только с одной записью, находящейся на более высоком уровне.

Такая система хорошо иллюстрируется системой классификации растений и животных. Примером может также служить структура хранения информации на дисках ПК. Главное понятие такой модели – уровень. Количество уровней и их состав зависят от принятой при создании БД классификации. Доступ к любой из этих записей осуществляется путем

прохода по строго определенной цепочке узлов. При такой структуре легко осуществлять поиск нужных данных, но если изначально описание неполное, или не предусмотрен какой-либо критерий поиска, то он становится невозможным. Для достаточно простых задач такая система эффективна, но она практически непригодна для использования в сложных системах с оперативной обработкой запросов.

На рис. 1 приведен пример иерархической базы данных для автомобильной компании. При выпуске автомобилей такой компании необходимо было знать, сколько деталей следует заказать у своих поставщиков. Чтобы ответить на этот вопрос, необходимо определить, из каких деталей состоят эти части и т.д. Например, машина состоит из двигателя, корпуса и ходовой части; двигатель состоит из клапанов, цилиндров, свеч и т.д. Работа со списками составных частей была как будто специально предназначена для компьютеров. Список составных частей изделия по своей природе является иерархической структурой.

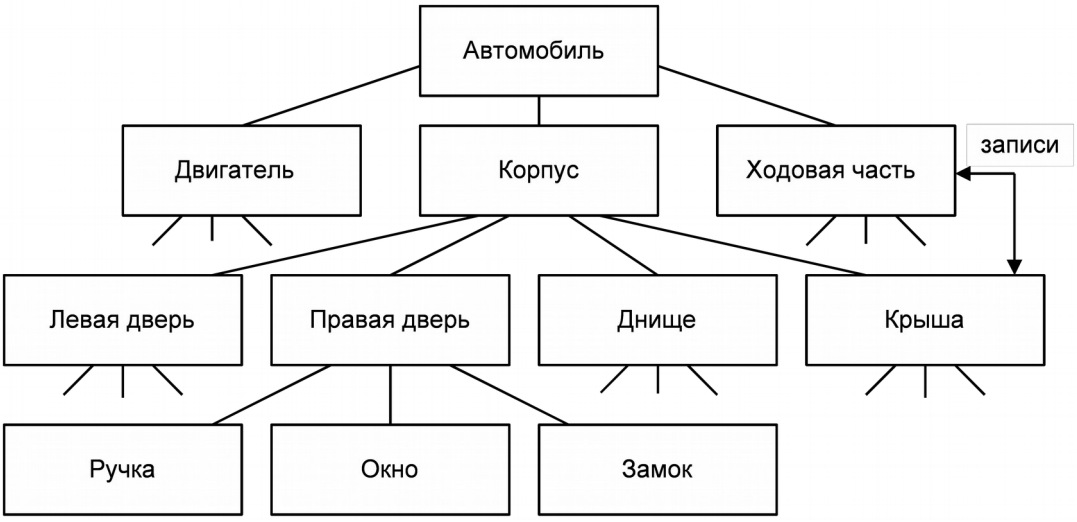


Рисунок 1 – Иерархическая база данных, содержащая информацию о составных частях

В этой модели каждая запись базы данных представляла конкретную деталь. Между записями существовали отношения предок/потомок, связывающие каждую часть с деталями, входящими в неё.

Чтобы получить доступ к данным, содержащимся в базе данных, программа могла:

- найти конкретную деталь (правую дверь) по её номеру;
- перейти "вниз" к первому потомку (ручка двери);
- перейти "вверх" к предку (корпус);
- перейти "в сторону" к другому потомку (правая дверь).

Таким образом, для чтения данных из иерархической базы данных требовалось перемещаться по записям, за один раз переходя на одну запись вверх, вниз или в сторону.

Одной из наиболее популярных иерархических СУБД была Information Management System (IMS) компании IBM, появившаяся в 1968 году.

Сетевая модель была призвана устранить некоторые из недостатков иерархических моделей. Если структура данных оказывалась сложнее, чем обычная иерархия, простота структуры иерархической базы данных становилась её недостатком. Например, в базе данных для хранения заказов один заказ мог участвовать в трёх различных отношениях предок/потомок, связывающих заказ с клиентом, разместившим его, со служащим, принявшим его, и с заказанным товаром. Такие структуры данных не соответствовали строгой иерархии.

В связи с этим была разработана новая сетевая модель данных. Она являлась улучшенной иерархической моделью, в которой одна запись могла участвовать в нескольких отношениях предок/потомок, как показано на рис. 2. В сетевой модели такие отношения назывались множествами.

В 1971 году на конференции по языкам систем данных был опубликован официальный стандарт сетевых баз данных, который известен как модель CODASYL. Компания IBM не стала разрабатывать собственную сетевую СУБД и вместо этого продолжала наращивать возможность IMS. Но в 70-х годах независимые производители программного обеспечения реализовали сетевую модель в таких продуктах, как IDMS компании Cullinet, Total компании Cincom и СУБД Adabas, которые приобрели большую популярность.

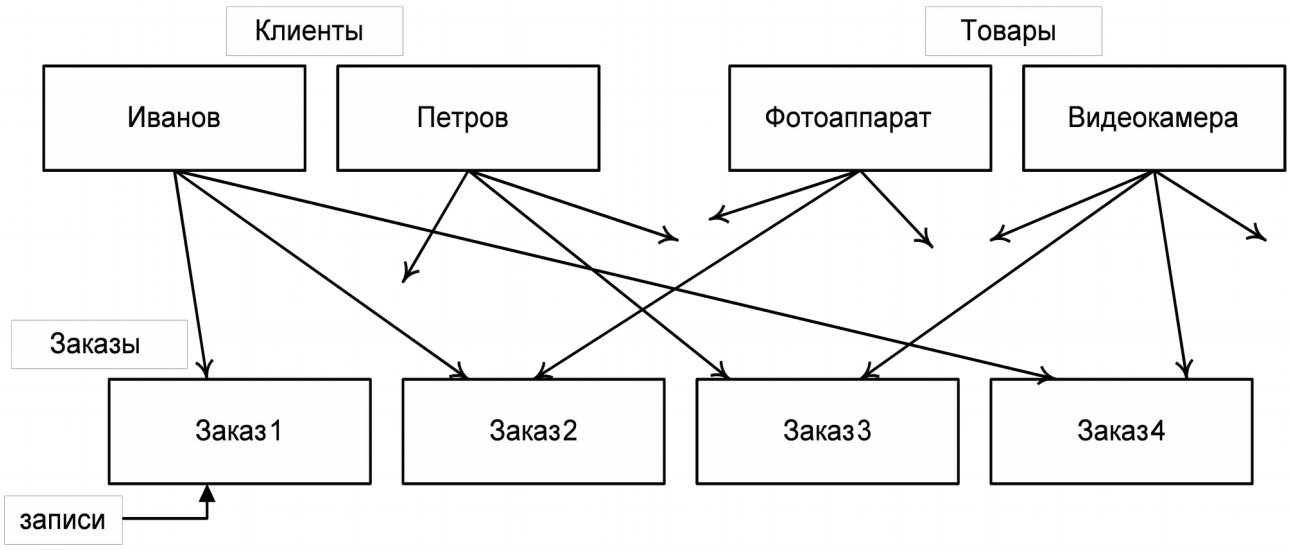


Рисунок 2 – Сетевая база данных, содержащая информацию о заказах

Однако, как и иерархические базы данных, сетевые базы данных

были очень жесткими. Наборы отношений и структуру записей приходилось задавать наперёд. Такая модель позволяла ускорить доступ к данным, но изменение структуры базы требовало значительных усилий и времени, а иногда и вовсе нужна была перестройка всей базы данных.

Как иерархическая, так и сетевая база данных были инструментами программистов. Чтобы получить ответ на вопрос типа "Какой товар наиболее часто заказывает компания А?", программисту приходилось писать программу для навигации по базе данных. Реализация пользовательских запросов часто затягивалась на недели и даже месяцы.

Реляционная модель была призвана устранить недостатки иерархической и сетевой моделей. Данная модель была создана Коддом в 1970 году и вызвала всеобщий интерес. Реляционная модель была попыткой упростить структуру базы данных. В ней отсутствовали явные указатели на предков и потомков, а все данные были представлены в виде простых таблиц, разбитых на строки и столбцы. На рис. 3 показана реляционная версия сетевой базы данных (рис. 2), содержащей информацию о заказах.

Кодд в 1985 году написал статью, где сформулировал 12 правил, которым должна удовлетворять любая база данных, претендующая на звание реляционной. С тех пор двенадцать правил Кодда считаются определением реляционной СУБД.

Таблица **PRODUCTS**

Description	Price	Qty_on_hand
Ratcher Link	$79.00	210
Widget Remower	$2750.00	25
Reducer	$355.00	38

Таблица **CUSTOMERS**

Company	Cust_rep	Credit_limit
JSP Inc.	101	$30000.00
First Corp.	102	$65000.00

Таблица **ORDERS**

Order_num	Order_date	Product	Qty
112233	1-Dec-2010	3AABL5	10
114488	2-Dec-2010	55FFDK	15
339922	3-Dec-2010	VB5JDG	27
883300	4-Dec-2010	FGT67V	55
225577	5-Dec-2010	POXC23	45

Рисунок 3 – Реляционная база данных, содержащая информацию о заказах

Однако можно сформулировать и более простое определение.

Реляционной называется база данных, в которой все данные, доступные пользователю, организованы в виде таблиц, а все операции над данными сводятся к операциям над этими таблицами.

Приведенное определение не оставляет места встроенным указателям, имеющимся в иерархических и сетевых СУБД. Несмотря на это, реляционная СУБД также способна реализовать отношения предок/потомок, однако эти отношения представлены исключительно значениями данных, содержащихся в таблицах.

Объектно-ориентированные модели применяют, если геометрия определенного объекта способна охватывать несколько слоев, атрибуты таких объектов могут наследоваться и для их обработки применяют специфические методы.

Создание и внедрение в практику современных информационных систем автоматизированных баз данных выдвигает новые задачи проектирования, которые невозможно решать традиционными приемами и методами. Большое внимание необходимо уделять вопросам проектирования баз данных. От того, насколько успешно будет спроектирована база данных, зависит эффективность функционирования системы в целом, ее жизнеспособность и возможность расширения и дальнейшего развития. Поэтому вопрос проектирования баз данных выделяют как отдельное, самостоятельное направление работ при разработке информационных систем.

Проектирование баз данных – это итерационный, многоэтапный процесс принятия обоснованных решений в процессе анализа информационной модели предметной области, требований к данным со стороны прикладных программистов и пользователей, синтеза логических и физических структур данных, анализа и обоснования выбора программных и аппаратных средств.

Ниже будут рассмотрены основы реляционных баз данных, а также подходы к процессу проектирования реляционных баз данных.

Основные задачи проектирования баз данных:

- обеспечение хранения в БД всей необходимой информации;
- обеспечение возможности получения данных по всем необходимым запросам;
- сокращение избыточности и дублирования данных;
- обеспечение целостности данных (правильности их содержания): исключение противоречий в содержании данных, исключение их потери и т.д.

1. ОСНОВНЫЕ ПОНЯТИЯ РЕЛЯЦИОННОЙ МОДЕЛИ БАЗ ДАННЫХ

Принято считать, что реляционный подход к организации баз данных был заложен в конце 60-х годов. Эдгаром Коддом. В последние десятилетия этот подход является наиболее распространенным. Достоинствами реляционного подхода принято считать следующие свойства:

- реляционный подход основывается на небольшом числе интуитивно понятных абстракций, на основе которых возможно простое моделирование наиболее распространенных предметных областей;
- эти абстракции могут быть точно и формально определены;
- теоретическим базисом реляционного подхода к организации баз данных служит простой и мощный математический аппарат теории множеств и математической логики;
- реляционный подход обеспечивает возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

Компьютерный мир далеко не сразу признал реляционные системы. В 70-е годы прошлого века, когда уже были получены почти все основные теоретические результаты и даже существовали первые прототипы реляционных СУБД, многие авторитетные специалисты отрицали возможность добиться эффективной реализации таких систем. Однако преимущества реляционного подхода и развитие методов и алгоритмов организации и управления реляционными базами данных привели к тому, что к концу 80-х годов реляционные системы заняли на мировом рынке СУБД доминирующее положение.

Рассмотрим реляционный подход к представлению данных подробнее.

Итак, в конце 60-х годов появились работы, в которых обсуждались возможности применения различных табличных даталогических моделей данных, т.е. возможности использования привычных и естественных способов представления данных. Наиболее значительной из них была статья сотрудника фирмы IBM д-ра Э. Кодда (Codd E.F., A Relational Model of Data for Large Shared Data Banks. CACM 13: 6, June 1970), где, вероятно, впервые был применен термин «реляционная модель данных».

Будучи математиком по образованию Э. Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как *отношение* (relation).

1.1. Таблицы

В реляционной базе данных информация организована в виде таблиц, разделённых на строки и столбцы, на пересечении которых содержатся значения данных. У каждой таблицы имеется уникальное имя, описывающее её содержимое. Более наглядно структуру таблицы иллюстрирует рис 1.1, на котором изображена таблица OFFICES.

Каждая горизонтальная строка этой таблицы представляет отдельную физическую сущность – один офис. Пять строк таблицы вместе представляют все пять офисов компании. Все данные, содержащиеся в конкретной строке таблицы, относятся к офису, который описывается этой строкой.

Рисунок 1.1 – Структура реляционной таблицы

Каждый вертикальный столбец таблицы OFFICES представляет один элемент данных для каждого из офисов. Например, в столбце CITY содержатся названия городов, в которых расположены офисы. В столбце SALES содержатся объёмы продаж, обеспечиваемые офисами.

На пересечении каждой строки с каждым столбцом таблицы содержится в точности одно значение данных. Например, в строке, представляющей нью-йоркский офис, в столбце CITY содержится значение «New York». В столбце SALES той же строки содержится значение $692.000.000, которое является объёмом продаж нью-йоркского офиса с начала года.

Все значения, содержащиеся в одном и том же столбце, являются данными одного типа. Например, в столбце CITY содержатся только слова, в столбце SALES – денежные суммы, а в столбце MGR приведены целые числа, представляющие идентификаторы служащих. Множество значений, которые могут содержаться в столбце, называется доменом этого столбца. Доменом столбца CITY является множество названий городов. Доменом столбца SALES является любая денежная

сумма. Домен столбца REGION состоит всего из двух значений: «Eastern» и «Western», поскольку у компании всего два торговых региона.

У каждого столбца в таблице есть своё имя, которое обычно служит заголовком столбца. Все столбцы в одной таблице должны иметь уникальные имена, однако разрешается присваивать одинаковые имена столбцам, расположенным в различных таблицах. На практике такие имена столбцов, как NAME, ADDRESS, QTY, PRICE и SALES, часто встречаются в различных таблицах одной базы данных.

Столбцы таблицы упорядочены слева направо, и их порядок определяется при создании таблицы. В любой таблице всегда есть как минимум один столбец. В стандарте ANSI/ISO не указывается максимально допустимое число столбцов в таблице, однако почти во всех коммерческих СУБД этот предел существует и обычно составляет примерно 255 столбцов.

В отличие от столбцов строки таблицы не имеют определённого порядка. Это значит, что если последовательно выполнить два одинаковых запроса для отображения содержимого таблицы, нет гарантии, что оба раза строки будут перечислены в одном и том же порядке.

В таблице может содержаться любое количество строк. Вполне допустимо существование таблицы с нулевым количеством строк. Такая таблица называется пустой. Пустая таблица сохраняет структуру, определённую её столбцами (просто в ней не содержится данные). Стандарт ANSI/ISO не накладывает ограничений на количество строк в таблице, и во многих СУБД размер таблиц ограничен лишь свободным дисковым пространством компьютера. В других СУБД имеется максимальный предел, однако он весьма высок – около двух миллиардов строк, а иногда и больше.

1.2. Первичные ключи

Поскольку строки в реляционной таблице не упорядочены, нельзя выбрать строку по ее номеру в таблице. В таблице нет «первой», «последней» или «тринадцатой» строки. Тогда каким же образом можно указать в таблице конкретную строку, например строку для офиса, расположенного в Денвере?

В правильно построенной реляционной базе данных в каждой таблице есть один или несколько столбцов, значения в которых во всех строках разные. Этот столбец (столбцы) называется первичным ключом таблицы. Если посмотреть на базу данных, показанную на рис. 1.1, то на первый взгляд первичным ключом таблицы OFFICES могут служить и столбец OFFICE, и столбец CITY. Однако в случае, если компания будет расширяться и откроет в каком-либо городе второй офис, столбец CITY больше не сможет выполнять роль первичного ключа. На практике в

качестве первичных ключей таблиц обычно следует выбирать идентификаторы, такие, как идентификатор офиса (OFFICE в таблице OFFICES), служащего (EMPL_NUM в таблице SALESREPS) и клиента (CUST_NUM в таблице CUSTOMES). В случае с таблицей ORDERS выбора нет — единственным столбцом, содержащим уникальные значения, является номер заказа (ORDER_NUM).

Таблица PRODUCTS, фрагмент которой показан на рис. 1.2, является примером таблицы, в которой первичный ключ представляет собой комбинацию столбцов. Такой первичный ключ называется составным. Столбец MRF_ID содержит идентификаторы производителей всех товаров, перечисленных в таблице, а столбец PRODUCT_ID содержит номера, присвоенные товарам производителями. Может показаться, что столбец PRODUCT_ID мог бы и один выполнять роль первичного ключа, однако ничто не мешает двум различным производителям присвоить своим изделиям одинаковые номера. Таким образом, в качестве первичного ключа таблицы PRODUCTS необходимо использовать комбинацию столбцов MRF_ID и PRODUCT_ID. Для каждого из товаров, содержащихся в таблице, комбинация значений в этих столбцах будет уникальной.

Таблица **PRODUCTS**

Mfr_id	Product_id	Description	Price	Qty_on_hand
REI	3AABL5	Ratcher Link	$79.00	210
ACI	55FFDK	Widget Remower	$2750.00	25
QSA	VB5JDG	Reducer	$355.00	38
BIC	FGT67V	Plate	$180.00	0

Первичный ключ

Рисунок 1.2 – Пример таблицы с составным первичным ключом

Первичный ключ для каждой строки таблицы является уникальным, поэтому в таблице с первичным ключом нет двух совершенно одинаковых строк. Таблица, в которой все строки отличаются друг от друга, в математических терминах называется отношением. Именно этому термину реляционные базы данных и обязаны своим названием, поскольку в их основе лежат отношения (таблицы с отличающимися друг от друга строками).

Хотя первичные ключи являются важной частью реляционной модели данных, в первых реляционных СУБД (System/R, DB2, Oracle и других) не была обеспечена явным образом их поддержка. Как правило, проектировщики базы данных сами следили за тем, чтобы у всех таблиц были первичные ключи, однако в самих СУБД не было возможности

определить для таблицы первичный ключ. И только в СУБД DB2 Version 2, появившейся в апреле 1988 года, компания IBM реализовала поддержку первичных ключей. После этого подобная поддержка была добавлена в стандарт ANSI/ISO.

1.3. Отношения предок/потомок

Одним из отличий реляционной модели от первых моделей представления данных было то, что в ней отсутствовали явные указатели, используемые для реализации отношений предок/потомок в иерархической модели данных. Однако вполне очевидно, что отношения предок/потомок существуют и в реляционных базах данных. Например, в базе данных каждый из служащих закреплен за конкретным офисом, поэтому ясно, что между строками таблицы OFFICES и таблицы SALESREPS существует отношение. Не приводит ли отсутствие явных указателей в реляционной модели к потере информации?

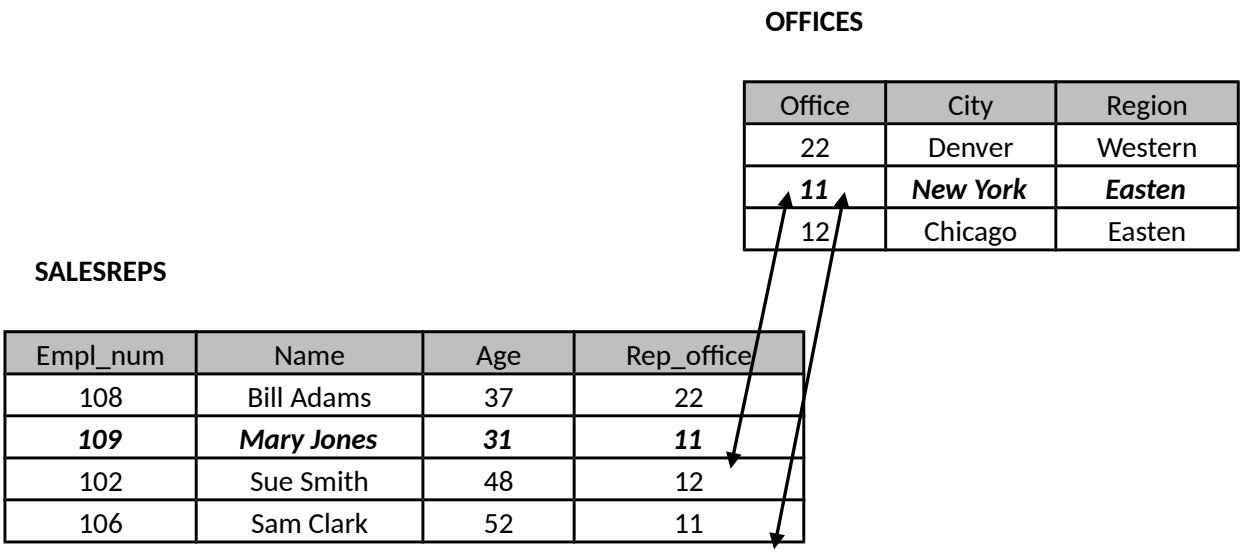


Рисунок 1.3 – Отношение предок/потомок в реляционной базе данных

Как следует из рис. 1.3, ответ на этот вопрос должен быть отрицательным. На рисунке изображено несколько строк из таблиц OFFICES и SALESREPS. Обратим внимание на то, что в столбце REP_OFFICE таблицы SALESREPS содержится идентификатор офиса, в котором работает сотрудник. Доменом этого столбца (множеством значений, которые могут в нем храниться) является множество идентификаторов офисов, содержащихся в столбце OFFICE таблицы OFFICES. То, в каком офисе работает Мэри Джонс, можно узнать, определив значение столбца REP_OFFICE в строке таблицы SALESREPS

для Мэри Джонс (число 11) и затем отыскав в таблице OFFICES строку с таким же значением в столбце OFFICE (под номером 11 находится офис в Нью-Йорке). Таким же образом, чтобы найти всех служащих нью-йоркского офиса, следует найти значение столбца OFFICE для Нью-Йорка (число 11), а потом просмотреть таблицу SALESREPS и найти все строки, в столбце REP_OFFICE которых содержится число 11 (это строки для Мэри Джонс и Сэма Кларка).

Отношение предок/потомок, существующее между офисами и работающими в них людьми, в реляционной модели не потеряно; просто оно реализовано в виде одинаковых значений данных, хранящихся в двух таблицах, а не в виде явного указателя. Все отношения, существующие между таблицами реляционной базы данных, реализуются в таком виде.

1.4. Внешние ключи

Столбец одной таблицы, значения в котором совпадают со значениями столбца, являющегося первичным ключом другой таблицы, называется внешним ключом. На рис. 1.3 столбец REP_OFFICE представляет собой внешний ключ для таблицы OFFICES. Значения, содержащиеся в этом столбце, представляют собой идентификаторы офисов. Эти значения соответствуют значениям в столбце OFFICE, который является первичным ключом таблицы OFFICES. Совокупно первичный и внешний ключи создают между таблицами, в которых они содержатся, такое же отношение предок/потомок, как и в иерархической базе данных.

Внешний ключ, как и первичный, тоже может представлять собой комбинацию столбцов. На практике внешний ключ всегда будет составным (состоящим из нескольких столбцов), если он ссылается на составной первичный ключ в другой таблице. Очевидно, что количество столбцов и их типы данных в первичном и внешнем ключах совпадают.

Если таблица связана с несколькими другими таблицами, она может иметь несколько внешних ключей. На рис. 1.4 показаны три внешних ключа таблицы ORDERS из учебной базы данных:

- столбец REP является внешним ключом для таблицы SALESREPS и связывает каждый заказ со служащим, принявшим его;
- столбец CUST является внешним ключом для таблицы CUSTOMES и связывает каждый заказ с клиентом, разместившим его;
- столбцы MRF и PRODUCT совокупно представляют собой составной внешний ключ для таблицы PRODUCTS, который связывает каждый заказ с заказанным товаром.

Отношения предок/потомок, созданные с помощью трех внешних ключей в таблице ORDERS, могут показаться знакомыми. И действительно, это те же самые отношения, что и в сетевой базе данных,

представленной на рис. 1.4. Как показывает пример, реляционная модель данных обладает всеми возможностями сетевой модели по части выражения сложных отношений.

Внешние ключи являются неотъемлемой частью реляционной модели, поскольку реализуют отношения между таблицами базы данных.

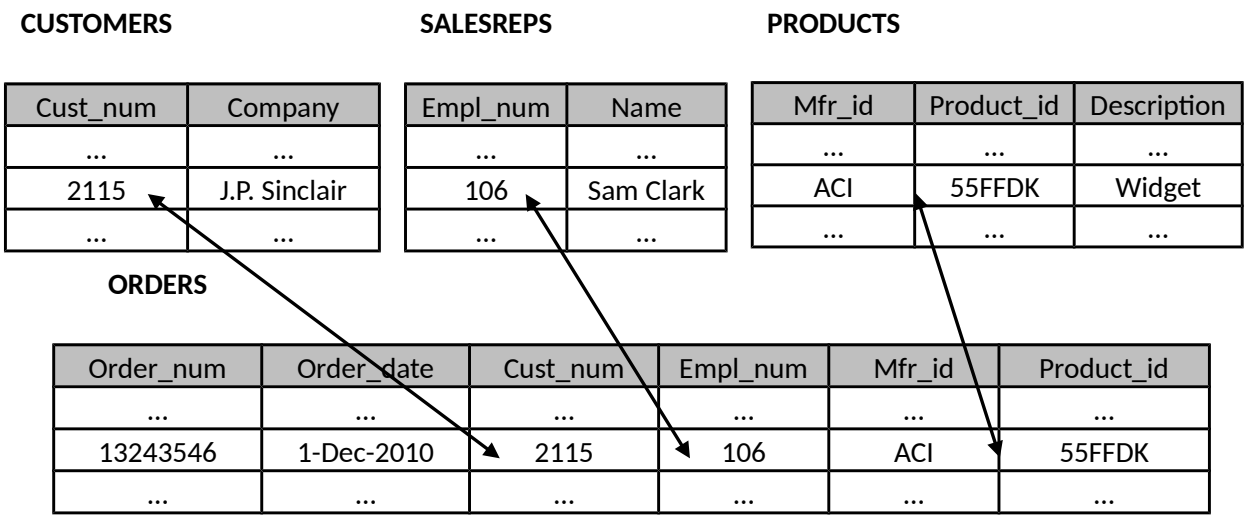


Рисунок 1.4 – Множественные отношения предок/потомок в реляционной базе данных

2. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ НА ОСНОВЕ НОРМАЛИЗАЦИИ ОТНОШЕНИЙ

В реляционной теории нет универсальных рецептов для проектирования надежной и эффективной в использовании базы данных. Разработчик волен выбирать различные инструменты и методы проектирования. Некоторые полагаются исключительно на интуицию и здравый смысл, другие используют различные вспомогательные средства.

Однако при всем разнообразии подходов все же есть некоторые каноны, нарушение которых весьма отрицательно скажется как при проектировании базы данных, так и при ее эксплуатации. Так, например, весьма актуальной является проблема нормализации баз данных [4,5]. Пренебрежение нормализацией делает структуру базы данных запутанной, а саму базу – ненадежной в работе.

2.1. Понятие отношения, атрибута и кортежа отношения

Итак, в конце 60-х годов Э.Кодд предложил использовать для

обработки данных аппарат теории множеств. Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как *отношение* (relation).

Отношение – это двумерная таблица, имеющая уникальное имя и состоящая из строк и столбцов, где строки соответствуют записям, а столбцы – атрибутам.

Заметим, что порядок следования атрибутов в отношении не влияет на само отношение, оно имеет один и тот же смысл при любом порядке их следования:

Студент (ФИО, группа, адрес)
Студент (ФИО, адрес, группа) *-- это два одинаковых (эквивалентных) отношения*

Отметим следующее свойство отношений. Все строки (записи, кортежи) отношения должны быть различны, т.е. в отношении (в таблице) не может быть двух одинаковых строк. Другими словами, каждое отношение должно иметь один атрибут (столбец), который будет однозначно идентифицировать строки в таблице (будет являться первичным ключом).

Студент

ФИО	*Группа*	*Адрес*
…	…	…
Иванов И. И.	611-п	ХАИ-7
…	…	..
Иванов И. И.	611-п	ХАИ-7
...	…	…

В приведенном выше отношении теоретически можно предположить, что могут быть два студента с одинаковой фамилией и инициалами, которые обучаются в группе 611-п и проживают в общежитии ХАИ-7. Поэтому для однозначной идентификации строк отношение Студент требует введения дополнительного атрибута, который и будет являться первичным ключом отношения. Таким атрибутом может выступить, например, номер зачетной книжки. В этом случае отношение Студент примет следующий вид:

Студент

Ном_зач	*ФИО*	*Группа*	*Адрес*
123/10	Иванов И. И.	611-п	ХАИ-7

…	…	…	..
127/10	Иванов И. И.	611-п	ХАИ-7
…	...	…	…

2.2. Избыточность данных

Избыточность данных – нежелательное явление в БД, которое ведет к увеличению объема памяти, необходимого для физического хранения отношений (таблиц).

Заметим, что избыточность данных вызывается прежде всего дублированием данных.

Однако не всегда дублирование данных нужно исключать. Если реализовать подобную ситуацию (исключить все дублирования), то можно получить разрозненные отношения. Поэтому необходимо только минимизировать избыточность за счет устранения некоторых ее видов и оставить ту долю избыточности, которая позволит удерживать БД как единое целое.

Рассмотрим пример отношения, содержащего нежелательную избыточность:

Студент

Ном_зач	*ФИО*	*Группа*	*Куратор*
1001	Иванов И.И.	611-п	проф. Туркин И.Б.
1002	Петров П.П.	612-п	ст. п. Соколова Е.В.
1003	Сидоров С.С.	611-п	проф. Туркин И.Б.
1004	Николаев Н.Н.	611-п	проф. Туркин И.Б.
1005	Александров А.А.	612-п	ст. п. Соколова Е.В.

В отношении Студент первичным ключом является атрибут Ном_зач. И что мы видим? В каждой строке с информацией о студентах дублируется информация о том, кто является куратором данной группы.

Кроме того, что такое дублирование потребует дополнительные ресурсы памяти, при дублировании информации очень легко допустить ошибку при вводе значений, что, в свою очередь, приведет к несогласованности БД.

Например, можно ввести не ст. п., а ст.п., или ошибиться в инициалах, и тогда при написании соответствующего запроса, часть данных будет упущена:

Select ном_зач
from Студент
where Куратор='ст. п. Соколова Е.В.';

Кроме того, избыточные данные могут привести к аномалиям обновлений.

2.3. Аномалии обновления, удаления, вставки данных

В процессе эксплуатации реляционной базы данных происходит модификация данных: в отношениях добавляются или удаляются кортежи, в результате определенных событий изменяются значения некоторых атрибутов кортежей и т.п. После манипулирования данными значения некоторых атрибутов могут дублироваться, порождая избыточность данных, значения некоторых атрибутов могут удаляться, исчезая из базы данных, хотя потребность в этих значениях остается, некоторые значения атрибутов не могут быть добавлены, поскольку их значения неизвестны, т.е. данные в базе данных с течением времени становятся несогласованными и противоречивыми. Потенциальная противоречивость (или аномалии) при выполнении операций манипулирования данными в базе данных зависит как от типа ФЗ между атрибутами, так и от группировки этих атрибутов по отношениям.

Рассмотрим типичные примеры аномалий обновлений, возникающих при выполнении операций включения, удаления и модификации данных.

Под аномалиями обновления понимаются трудности, с которыми приходится сталкиваться при выполнении операций добавления кортежей в отношение (INSERT), удаления кортежей (DELETE) и модификации кортежей (UPDATE).

Пример. Аномалии операций над базой данных

Поставки (Поставщик, Адрес, Товар, Количество, Стоимость)

Модификация. Если Поставщик поставляет несколько видов товара, то значение атрибута Адрес повторяется для каждого кортежа и, следовательно, при изменении адреса поставщика необходимо изменить значение этого атрибута во всех этих кортежах. В противном случае будет нарушена целостность данных базы данных.

Удаление. Если Поставщик прекращает поставку товаров на некоторое время, то кортежи со всеми его поставками удаляются. При этом происходит потеря реквизитов поставщика – Адреса и Наименования.

Вставка. Если договор на поставку уже заключен, а поставка еще не состоялась, то невозможно включить в рассматриваемое отношение значения атрибутов Поставщик и Адрес, поскольку нет сведений о поставках (проблема интерпретации null-значений).

Подобные аномалии в данных лишь отчасти дают ответ на вопрос, почему логическая структура реляционной БД может быть

неудовлетворительной. Некоторые проблемы избыточности данных можно разрешить, заменив, например, исходное отношение Поставки на два новых отношения – отношение Поставщик (Поставщик, Адрес) и отношение Поставка (Поставщик, Товар, Количество, Стоимость). Однако в целом остается ряд вопросов, на которые теория реляционных баз данных не дает удовлетворительных ответов: Как найти хорошую замену для плохой схемы отношений? Как определить, является ли выбранная замена выгодной? Как создать схему, обеспечивающую наилучшую производительность?

Рассмотрим еще пример аномалий обновления.

Пусть имеется отношение Сотрудники_проекты_задания, заданное следующими атрибутами: номер сотрудника, разряд сотрудника, зарплата сотрудника, номер проекта, задание сотрудника в проекте.

Сотрудники_проекты_задания (ном_сотр, разр_сотр, зарп_сотр, ном_проекта, задан_сотр).

Будем считать, что разряд служащего определяет размер его заработной платы и что каждый сотрудник может участвовать в нескольких проектах, но в каждом проекте он выполняет только одно задание.

Вставка. Нельзя дополнить отношение Служащие_проекты_задания данными о служащем, который в данное время еще не участвует ни в одном проекте. Между тем часто бывает так, что сначала служащего принимают на работу, устанавливают его разряд и размер зарплаты, а лишь потом назначают для него проект.

Удаление. Мы не можем сохранить в отношении Служащие_проекты_задания данные о служащем, завершившем участие в своем последнем проекте. Между тем характерна ситуация, когда между проектами возникают перерывы, не приводящие к увольнению служащих.

Модификация. Чтобы изменить разряд служащего, мы будем вынуждены модифицировать все кортежи с соответствующим значением атрибута ном_сотр.

2.4. Понятие функциональной зависимости

Классическая технология проектирования реляционных баз данных связана с теорией нормализации, основанной на анализе функциональных зависимостей между атрибутами отношений. Поэтому, прежде чем рассматривать вопросы нормализации отношений, необходимо ознакомиться с понятием функциональной зависимости (ФЗ).

Итак, хранение данных в реляционных базах данных преследует две цели: понизить избыточность данных и повысить их достоверность. Для этого необходимо иметь возможность накладывать некоторые

ограничения, как на сами хранимые данные, так и на взаимосвязи и взаимозависимости между ними, так как в противном случае вместо базы данных мы получим просто свалку, лишенную всякой структуры. Обычно эти ограничения вытекают из анализа предметной области, которая моделируется нашей информационной системой. Одним из средств формализации информации, полученной в результате такого анализа, являются зависимости между данными.

Существует несколько разновидностей зависимостей, которые рассматриваются реляционной теорией: F-зависимости, MV-зависимости и J-зависимости. В настоящий момент наибольший интерес среди них для нас представляют функциональные зависимости, или F-зависимости.

Если значения кортежа на некотором множестве атрибутов единственным образом определяют значения на другом множестве атрибутов, то говорят, что имеет место функциональная зависимость (F-зависимость).

Дадим более строгое определение на основе реляционной алгебры.

Определение 1. Пусть R (A1, A2, ..., An) – схема отношения R, а X и Y – подмножества R. Говорят, что X функционально определяет Y, если каждому значению атрибутов кортежа отношения из X соответствует не более одного значения атрибутов того же кортежа отношения из Y. Такая ФЗ обозначается как F: X→Y.

Если посмотреть на определение функциональной зависимости, то видно, что X фактически является ключом отношения r[XY].

Рассмотрим два отношения: Студент и Экзаменационная ведомость.

Студент (Номер_зачетки, Фамилия_студента, Факультет, Специальность)

Экзаменационная_ведомость (Ном_зачетки, Фамилия_студента, Код_экзамена, Предмет, Дата_сдачи, Оценка)

Студент

Номер_зачетки	Фамилия_студента	Факультет	Специальность
1000	Иванов	Экономический	Экономика предприятия
1001	Петров	Экономический	Экономика предприятия
1002	Сидоров	Инженерный	Программная инженерия
1003	Алешин	Инженерный	Программная инженерия

Экзаменационная_ведомость

Номер_ зачетки	Фамилия_ студента	Код_ экзамена	Предмет	Дата_сдачи	Оценка

1000	Иванов	1	Экономика	20.06.10	5
1001	Петров	1	Экономика	20.06.10	4
1000	Иванов	2	Ин. язык	24.06.10	5
1001	Петров	2	Ин. язык	24.06.10	5
1002	Сидоров	3	Физика	27.06.10	4
1003	Алешин	3	Физика	27.06.10	3
1002	Сидоров	4	Математика	30.06.10	5
1003	Алешин	4	Математика	30.06.10	4
1003	Алешин	5	История	3.07.10	5

Приведем пример функциональных зависимостей для отношения Студент:

Номер_зачетки → Фамилия_студента,
Номер_зачетки → Факультет,
Номер_зачетки → Специальность.

Приведем пример функциональных зависимостей для отношения Экзаменационная_ведомость:

Номер_зачетки → Фамилия_студента,
Код_экзамена → Предмет
(Номер_зачетки, Код_экзамена, Дата_сдачи) → Оценка
(Номер_зачетки, Предмет, Дата_сдачи) → Оценка

2.5. Понятие нормализации

Под нормализацией отношения подразумевается процесс приведения отношения к одной из так называемых нормальных форм (или в дальнейшем НФ). Однако перед рассмотрением НФ следует понять, зачем нужна нормализация.

Как уже неоднократно упоминалось в данном цикле, база данных – это не просто хранилище фактов (с этой задачей способны справиться и незатейливые плоские файлы). При проектировании баз данных упор в первую очередь делается на достоверность и непротиворечивость хранимых данных, причем эти свойства не должны утрачиваться в процессе работы с данными, т.е. после многочисленных изменений, удалений и дополнений данных по отношению к первоначальному состоянию БД.

Для поддержания БД в устойчивом состоянии используется ряд механизмов, которые получили обобщенное название средств поддержки целостности. Эти механизмы применяются как статически (на этапе проектирования БД), так и динамически (в процессе работы с БД).

Рассмотрим ограничения, которым должна удовлетворять БД в процессе создания, независимо от ее наполнения данными. Приведение структуры БД в соответствие этим ограничениям – это и есть

нормализация.

В целом суть этих ограничений весьма проста: каждый факт, хранимый в БД, должен храниться один-единственный раз, поскольку дублирование может привести (на практике непременно приводит как только проект приобретает реальную сложность) к несогласованности между копиями одной и той же информации. Следует избегать любых неоднозначностей, а также избыточности хранимой информации.

Следует заметить, что в процессе нормализации постоянно встречается ситуация, когда отношение приходится разложить на несколько других отношений. Поэтому более корректно было бы говорить о нормализации не отдельных отношений, а всей их совокупности в БД.

Итак, что же представляет собой процесс нормализации? Фактически это не что иное, как последовательное преобразование исходной БД к НФ, при этом каждая следующая НФ обязательно включает в себя предыдущую, что позволяет разбить процесс на этапы и производить его однократно, не возвращаясь к предыдущим этапам).

В теории реляционных БД 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 форма проекции-соединения (5NF или PJNF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле улучшает свойства предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

На практике, как правило, ограничиваются 3НФ, ее оказывается вполне достаточно для создания надежной схемы БД. НФ более высокого порядка представляют скорее академический интерес из-за чрезмерной сложности. Более того, при реализации абстрактной схемы БД в виде реальной базы иногда разработчики вынуждены сделать шаг назад – провести денормализацию в целях повышения эффективности, ибо идеальная с точки зрения теории структура может оказаться слишком накладной на практике.

2.6. Формы нормализации

При рассмотрении материала в данном разделе за основу был взят материал из [4].

2.6.1. Первая нормальная форма

На начальном этапе схема отношений должна находиться в первой нормальной форме (1НФ).

Определение 2.1. Отношение находится в первой нормальной форме (1NF), если все атрибуты отношения принимают простые значения (атомарные или неделимые), не являющиеся множеством или кортежем из более элементарных составляющих.

Рассмотрим отношение Сотрудники_проекты.

Сотрудники_проекты_задания

Ном_сотр	Разр_сотр	Зарп_сотр	Ном_проекта	Задан_сотр
355	1	3500	1	А
			2	Б
356	2	4000	1	Б
357	3	4500	1	В
			2	В
358	2	4000	2	А

Проанализируем отношение Сотрудники_проекты на соответствие 1NF. Видим, что атрибут ном_проекта принимает в первом и третьем кортежах неатомарные значения. Поэтому рассматриваемое отношение должно быть преобразовано к 1NF следующим образом:

Сотрудники_проекты_задания

Ном_сотр	Разр_сотр	Зарп_сотр	Ном_проекта	Задан_сотр
355	1	3500	1	А
355	**1**	**3500**	**2**	**Б**
356	2	4000	1	Б
357	3	4500	1	В
357	**3**	**4500**	**2**	**В**
358	2	4000	2	А

Здесь на пересечении любой строки и столбца находится одно значение и, следовательно, данная таблица находится в первой нормальной форме.

2.6.2. Вторая нормальная форма

Определение 2.3. Отношение находится во второй нормальной форме (2NF) тогда и только тогда, когда она находится в первой нормальной форме и каждый неключевой атрибут минимально

функционально зависит от первичного ключа.

Проанализируем отношение Сотрудники_проекты_задания с составным первичным ключом (ном_сотр, ном_проекта) на соответствие 2NF.

Сотрудники_проекты_задания (ном_сотр, разр_сотр, зарп_сотр, ном_проекта, задан_сотр).

Сотрудники_проекты_задания

Ном_сотр	Разр_сотр	Зарп_сотр	Ном_проекта	Задан_сотр
355	1	3500	1	А
355	1	3500	2	Б
356	2	4000	1	Б
357	3	4500	1	В
357	3	4500	2	В
358	2	4000	2	А

Отношение Сотрудники_проекты_задания не находится в 2NF. Например, функциональная зависимость (ном_сотр, ном_проекта) → разр_сотр не является минимальной, так как атрибут разр_сотр зависит только от значения атрибута ном_сотр, а не от комбинации значений атрибутов (ном_сотр, ном_проекта).

Любое отношение, находящееся в 1NF, но не находящееся в 2NF, может быть приведено к набору отношений, находящихся в 2NF, путем декомпозиции исходного отношения.

Таким образом, исходное отношение Сотрудники_проекты_задания может быть разбито на следующих два отношения:

Сотрудники (ном_сотр, разр_сотр, зарп_сотр)
Сотр_проект_задан (ном_сотр, ном_проекта, задан_сотр).

Следует отметить, что исходное отношение воспроизводится естественным соединением по общему атрибуту ном_сотр.

Заметим, в отношении Сотрудники может находиться кортеж с данными о сотруднике с номером 359, который еще не участвует ни в одном проекте. Наличие такого кортежа не влияет на результат естественного соединения, тело которого все равно будет совпадать с телом отношения Сотрудники_проекты_задания.

Сотрудники

Ном_сотр	Разр_сотр	Зарп_сотр
355	1	3500
356	2	4000

Сотр_проект_задан

Ном_сотр	Ном_проекта	Задан_сотр
355	1	А
355	2	Б

357	3	4500
358	2	4000
359	1	3500

356	1	Б
357	1	В
357	2	В
358	2	А

2.6.3. Третья нормальная форма

Определение 2.4. Отношение находится в третьей нормальной форме (3NF) в том и только в том случае, когда оно находится во второй нормальной форме, и каждый неключевой атрибут нетранзитивно функционально зависит от первичного ключа.

Проанализируем отношения Сотрудники и Сотр_проект_задан на соответствие 3NF.

Отношение Сотрудники не находится в 3NF (функциональная зависимость ном_сотр → зарп_сотр является транзитивной: ном_сотр → разр_сотр → зарп_сотр).

Любое отношение, находящееся в 2NF, но не находящееся в 3NF, может быть приведено к набору отношений, находящихся в 3NF путем декомпозиции отношения с транзитивной зависимостью на два.

Таким образом, исходное отношение Сотрудники может быть разбито на два отношения:

Сотруд (ном_сотр, разр_сотр) **Разряд** (разр_сотр, зарп_сотр)

Следует отметить, что исходное отношение воспроизводится естественным соединением по общему атрибуту разр_сотр.

Заметим, что в отношении Разряд может находиться кортеж с данными о разряде 4, который еще не присвоен ни одному служащему. Наличие такого кортежа не влияет на результат естественного соединения, тело которого все равно будет совпадать с телом отношения Сотрудники.

Сотруд

Ном_сотр	Разр_сотр
355	1
356	2
357	3
358	2
359	1

Разряд

Разр_сотр	Зарп_сотр
1	3500
2	4000
3	4500
4	5000

2.6.4. Нормальная форма Бойса-Кодда

До сих пор в определениях нормальных форм мы предполагали, что у декомпозируемого отношения имеется только один возможный ключ. На практике чаще всего бывает именно так. Но имеется один частный случай,

который (почти) удовлетворяет требованиям 2NF и 3NF, но тем не менее порождает аномалии обновления. Это тот случай, когда у отношения имеется несколько возможных ключей, и некоторые из этих возможных ключей «перекрываются», т. е. содержат общие атрибуты.

Рассмотрим несколько измененное отношение Сотр_проект_задан (ном_сотр, имя_сотр, ном_проекта, задан_сотр). Предполагается, что в отношении Сотр_проект_задан служащие уникально идентифицируются как по номерам удостоверений, так и по именам. Следовательно, существуют функциональные зависимости ном_сотр → имя_сотр, имя_сотр → ном_сотр. Поэтому для отношения Сотр_проект_задан возможны следующие первичные ключи:

Сотр_проект_задан (ном_сотр, имя_сотр, ном_проекта, задан_сотр)
Сотр_проект_задан (ном_сотр, имя_сотр, ном_проекта, задан_сотр)

В отношении Сотр_проект_задан все функциональные зависимости неключевых атрибутов от возможных ключей являются минимальными, кроме того, транзитивные функциональные зависимости отсутствуют. Однако для отношения Сотр_проект_задан свойственны аномалии обновления. Например, в случае изменения имени служащего требуется обновить атрибут имя_сотр во всех кортежах отношения Сотр_проект_задан, соответствующих данному служащему.

Причиной отмеченных аномалий является то, что в требованиях 2NF и 3NF не требовалась минимальная функциональная зависимость от первичного ключа атрибутов, являющихся компонентами других возможных ключей. Проблему решает нормальная форма, которую исторически принято называть нормальной формой Бойса-Кодда и которая является уточнением 3NF в случае наличия нескольких перекрывающихся возможных ключей.

Определение 2.5. Отношение находится в нормальной форме Бойса-Кодда (BCNF) в том и только в том случае, когда любая для этого отношения нетривиальная и минимальная функциональная зависимость имеет в качестве детерминанта некоторый возможный ключ данного отношения.

Отношение Сотр_проект_задан (ном_сотр, имя_сотр, ном_проекта, задан_сотр) может быть приведено к BCNF путем одной из двух декомпозиций:

Сотрудник_имя (ном_сотр, имя_сотр)
Сотр_пр_задан (ном_сотр, ном_проекта, задан_сотр)
или
Сотрудник_имя (ном_сотр, имя_сотр)
Сотр_пр_задан (имя_сотр, ном_проекта, задан_сотр)

Очевидно, что каждая из декомпозиций устраняет трудности, связанные с обновлением отношения Сотр_проект_задан.

Нормализация схемы базы данных способствует более эффективному выполнению системой управления базами данных операций обновления, поскольку сокращается число проверок и вспомогательных действий, поддерживающих целостность базы данных. При проектировании реляционной базы данных почти всегда добиваются второй нормальной формы всех входящих в базу данных отношений. В часто обновляемых базах данных обычно стараются обеспечить третью нормальную форму отношений. На нормальную форму Бойса-Кодда внимание обращают гораздо реже, поскольку на практике ситуации, в которых у отношения имеется несколько составных перекрывающихся возможных ключей, встречаются нечасто.

2.6.5. Четвертая нормальная форма

Иногда в отношениях требуется поддержка более сложных ограничений целостности, для выражения которых понятие функции оказывается недостаточным.

Класс зависимостей, опирающихся на понятие *функционала* – обобщение понятия функции, обнаружил в 1970-е гг. Рональд Фейджин. Он назвал такие зависимости многозначными, поскольку в них одному значению детерминанта соответствует множество значений зависимого атрибута.

Наличие в отношении многозначных зависимостей, не являющихся функциональными зависимостями от возможного ключа, приводит к аномалиям обновления таких отношений.

Рассмотрим отношение Сотр_пр_задан (ном_сотр, ном_проекта, задан_сотр). В данном отношении единственным возможным ключом является (ном_сотр, ном_проекта, задан_сотр), т.е.

Сотр_пр_задан (ном_сотр, ном_проекта, задан_сотр).

Предположим, что каждый сотрудник может участвовать в нескольких проектах, но в каждом проекте, в котором он участвует, им должны выполняться одни и те же задания:

Сотр_пр_задан

Ном_сотр	Ном_проекта	Задан_сотр
355	1	А
355	1	Б

356	1	В
356	1	Г
355	2	А
355	2	Б
356	2	В
356	2	Г

Наличие такого ограничения (как мы скоро увидим, оно порождается наличием многозначной зависимости) приводит к тому, что при работе с отношением Сотр_пр_задан проявляются аномалии обновления.

Добавление. Если уже участвующий в проектах сотрудник присоединяется к новому проекту, то в отношение Сотр_пр_задан требуется добавить столько кортежей, сколько заданий выполняет этот сотрудник.

Удаление. Если сотрудник прекращает участие в проектах, то отсутствует возможность сохранить данные о заданиях, которые он может выполнять.

Модификация. При изменении одного из заданий сотрудника необходимо изменить значение атрибута Задан_сотр в стольких кортежах, в скольких проектах он участвует.

Определение 2.6. В отношении R с атрибутами A, B, C (в общем случае составными) имеется многозначная зависимость B от A (A—>>B) в том и только в том случае, когда множество значений атрибута B, соответствующее паре значений атрибутов A и C, зависит от значения A и не зависит от значения C.

В отношении Сотр_пр_задан выполняются две многозначные зависимости: ном_сотр —>> ном_проекта и ном_сотр —>> задан_сотр. Первая из них означает, что каждому значению атрибута ном_сотр соответствует множество значений атрибута ном_проекта. Вторая многозначная зависимость означает, что каждому значению атрибута ном_сотр соответствует множество значений атрибута задан_сотр.

Определение 2.7. Отношение R находится в четвертой нормальной форме (4NF) в том и только в том случае, когда оно находится в BCNF, и все многозначные зависимости отношения R являются функциональными зависимостями с детерминантами – возможными ключами отношения R.

Отношение **Сотр_пр_задан** (ном_сотр, ном_проекта, задан_сотр) может быть приведено к 4NF путем следующей декомпозиции без потерь:

Сотрудник_проект (ном_сотр, ном_проекта)
Сотрудник_задание (ном_сотр, задан_сотр)

Сотрудник_проект

Ном_сотр	Ном_проекта
355	1
356	1

Сотрудник_задание

Ном_сотр	Задан_сотр
355	А
355	Б

355	2
356	2

356	В
356	Г

Очевидно, что такая декомпозиция устраняет трудности, связанные с обновлением отношения Сотр_пр_задан.

Добавление. Если некоторый уже участвующий в проектах сотрудник присоединяется к новому проекту, то в отношение Сотрудник_проект требуется добавить один кортеж, соответствующий новому проекту.

Удаление. Если сотрудник прекращает участие в проектах, то данные о заданиях, которые он может выполнять, остаются в отношении Сотрудник_задание.

Модификация. При изменении одного из заданий сотрудника необходимо изменить значение атрибута задан_сотр в одном кортеже отношения Сотрудник_задание.

В сущности 4NF является BCNF, в которой многозначные зависимости вырождаются в функциональные. В рассмотренном примере отношение Сотр_пр_задан не находится в 4NF, поскольку содержит многозначные зависимости, с другой стороны, отношения Сотрудник_проект и Сорудник_задание находятся в BCNF и не содержат многозначных зависимостей, поэтому они находятся в 4NF.

2.6.6. Пятая нормальная форма

Приведение отношения к 4NF предполагает его декомпозицию без потерь на два отношения (как и в случае 2NF, 3NF и BCNF). Однако бывают (хотя и нечасто) случаи, когда декомпозиция без потерь на два отношения невозможна, но можно произвести декомпозицию без потерь на большее число отношений.

Определение 2.8. n-декомпозируемым отношением называется отношение, которое может быть декомпозировано без потерь на n отношений (говорят также, декомпозировано без потерь на n проекций).

До сих пор мы имели дело с 2-декомпозируемыми отношениями.

Рассмотрим отношение Сотр_пр_задан (ном_сотр, ном_проекта, задан_сотр). В данном отношении единственным возможным ключом является (ном_сотр, ном_проекта, задан_сотр), т.е.

Сотр_пр_задан (ном_сотр, ном_проекта, задан_сотр).

Предполагаем, что в отношении Сотр_пр_задан нет многозначных зависимостей.

Сотр_пр_задан

Ном_сотр	Ном_проекта	Задан_сотр
355	1	А
355	1	Б
355	2	А
356	1	А

Рассмотрим следующее ограничение реального мира, которое для отношения Сотр_пр_задан может быть сформулировано на естественном языке следующим образом: «Если сотрудник с номером N участвует в проекте P, и в этом проекте выполняется задание Z, и сотрудник с номером N выполняет задание Z, то сотрудник с номером N выполняет задание Z в проекте P».

В общем виде такое ограничение называется зависимостью проекции-соединения.

Определение 2.9. Пусть задано отношение R, и A, B, …, Z являются произвольными подмножествами атрибутов R (составными, перекрывающимися атрибутами). Отношение R удовлетворяет зависимости проекции-соединения тогда и только тогда, когда R восстанавливается без потерь путем соединения своих проекций на атрибуты A, B, …, Z.

В отношении Сотр_пр_задан выполняется зависимость проекции-соединения (ном_сотр, ном_проекта), (ном_проекта, задан-сотр), (ном_сотр, задан_сотр). Наличие такой зависимости проекции-соединения обеспечивает возможность декомпозиции отношения на три отношения-проекции, но возникает вопрос, зачем это нужно? Чем плохо исходное отношение Сотр_пр_задан? Ответ следующий: этому отношению свойственны аномалии обновления.

Для примера рассмотрим отношение Сотр_пр_задан с такими значениями кортежей:

Сотр_пр_задан

Ном_сотр	Ном_проекта	Задан_сотр
355	1	Б
355	2	А

Продемонстрируем аномалию обновления, которая возникает при добавлении данных в отношение Сотр_пр_задан.

Добавление. Пусть в отношение Сотр_пр_задан добавляется кортеж <357, 1, А>. Что это означает? Что в проекте с номером 1 имеется работа с названием А. Так как сотрудник с номером 355 знает работу А (это следует из кортежа <355, 2, А>) и работа А входит в проект с номером 1 (это следует из добавляемого кортежа <357, 1, А>), а также сотрудник с

номером 355 принимает участие в проекте с номером 1 (это следует из кортежа <355, 1, Б>), поэтому в отношение Сотр_пр_задан вместе с кортежем <357, 1, А> должен быть добавлен и кортеж <355, 1, А> (это обуславливается наложенным на предметную область ограничением).

Отношение Сотр_пр_задан в этом случае примет вид:

Сотр_пр_задан

Ном_сотр	Ном_проекта	Задан_сотр
355	1	Б
355	2	А
357	1	А
355	1	А

Продемонстрированная аномалия обновления автоматически снимается после выполнения декомпозиции. Для этого декомпозируем отношение Сотр_пр_задан на три отношения: Сотруд_проект (ном_сотр, ном_проекта), Сотруд_задан (ном_сотр, задан_сотр) и Проект_задан (ном_проекта, задан_сотр). Результат декомпозиции отношения Сотр_пр_задан:

Сотруд_проект

Ном_сотр	Ном_проекта
355	1
355	2

Сотруд_задан

Ном_сотр	Задан_сотр
355	Б
355	А

Проект_задан

Ном_проекта	Задан_сотр
1	Б
2	А

Если теперь необходимо добавить данные о сотруднике с номером 357, выполняющем задание А в проекте 1, то необходимо в отношение Сотруд_проект добавить кортеж <357, 1>, в отношение Сотруд_задан – <357, А>, в отношение Проект_задан – <1, А>. В результате имеем:

Сотруд_проект

Ном_сотр	Ном_проекта
355	1
355	2

Сотруд_задан

Ном_сотр	Задан_сотр
355	Б
355	А

357	1

357	A

Проект_задан

Ном_проекта	Задан_сотр
1	Б
2	А
1	А

Но если теперь выполнить естественное соединение отношений Сотруд_проект, Сотруд_задан, Проект_задан, то получим отношение с атрибутами (ном_сотр, ном_проекта, задан_сотр) и следующими кортежами:

Ном_сотр	Ном_проекта	Задан_сотр
355	1	Б
355	2	А
357	1	А
355	1	А

Заметим, что последнее отношение полностью соответствует отношению Сотр_пр_задан:

Сотр_пр_задан

Ном_сотр	Ном_проекта	Задан_сотр
355	1	Б
355	2	А
357	1	А
355	1	А

Аналогично можно проиллюстрировать простоту и корректность операций удаления кортежей в декомпозированных отношениях Сотруд_проект, Сотруд_задан, Проект_задан.

Определение 2.10. Отношение R находится в пятой нормальной форме (нормальной форме проекции-соединения — PJ/NF) в том и только в том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.

Отношения Сотруд_проект, Сотруд_задан, Проект_задан находятся в пятой нормальной форме.

Таким образом, чтобы распознать, что данное отношение R находится в 5NF, необходимо знать все возможные ключи R и все зависимости проекции-соединения этого отношения. Обнаружение всех зависимостей соединения является нетривиальной задачей, и для ее

решения нет общих методов. Поэтому на практике проектирование реляционных баз методом нормализации обычно завершается после достижения 4NF, и отношения, находящиеся в 4NF, как правило, находятся и в 5NF. Зачем же тогда была введена эта туманная и труднодостижимая пятая нормальная форма?

Ответ на этот вопрос состоит в том, что 5NF является "окончательной" нормальной формой, которая достигается в процессе нормализации на основе проекций. "Окончательность" понимается в том смысле, что у отношения, находящегося в 5NF, отсутствуют аномалии обновлений, которые можно было бы устранить путем его декомпозиции. Другими словами, такие отношения далее нормализовать бессмысленно.

2.7. Проектирование базы данных на принципах нормализации универсального отношения

Классический подход к проектированию реляционной базы данных начинается с построения универсального отношения проектируемой БД. В одно универсальное отношение включаются все представляющие интерес атрибуты, и оно может содержать все данные, которые предполагается размещать в БД в будущем. Для малых БД (включающих не более 15 атрибутов) универсальное отношение может использоваться в качестве отправной точки при проектировании БД.

Рассмотрим пример универсального отношения, которое моделирует сдачу студентами сессий. Структура отношения Студент_сессия определяется следующим набором атрибутов (ном_зк, ФИО, ном_гр, фак_т, спец_ть, сем_р, код_предм, предмет, оценка).

Вариант таблицы Студент_сессия (см. табл. 2.1), показывает пример возможных значений, которые будут храниться.

Отношение, которое соответствует табл. 2.1, не находится в 1NF, так как содержит неатомарные значения. А вот преобразованное к 1NF отношение представлено в табл. 2.2. Однако такое преобразование привело к возникновению большого объема избыточных данных.

Заметим, что отношение, представленное в табл. 2.2, является примером универсального отношения.

Для устранения избыточности, противоречивости и аномалий обновления выполним нормализацию рассмотренного универсального отношения.

Проанализируем универсальное отношение на соответствие 2NF.

Таблица 2.1 – Отношение **Студент_сессия** ненормализованное

Ном_зк	*ФИО*	*Ном_гр*	*Фак_т*	*Спец_ть*	*Сем_р*	*Код_предм*	*Предмет*	*Оценка*
10001	Иванов И.И.	611	6	Программная инженерия	1	1	Программирование	5
						2	История Украины	5
						3	Высшая математика	4
					2	1	Программирование	5
						4	Украинский язык	4
10002	Петров П.П.	515	5	АСУ	1	1	Программирование	4
						4	Украинский язык	5
					2	5	Английский язык	4
						3	Высшая математика	4
10003	Сидоров С.С.	613	6	Экономика предприятия	1	2	История Украины	4
						5	Английский язык	4
						6	Экономика	5
					2	7	Бухгалтерский учет	4
						5	Английский язык	5
10004	Николаев Н.Н.	614	6	Финансы	1	4	Украинский язык	3
						2	История Украины	4
					2	3	Высшая математика	3
						7	Бухгалтерский учет	4
10005	Алешин А.А.	513	5	АСУ	1	1	Программирование	5
						4	Украинский язык	4
					2	5	Английский язык	5
						3	Высшая математика	5

Таблица 2.2 – Отношение **Студент_сессия** в 1NF

Ном_зк	*ФИО*	*Ном_гр*	*Фак_т*	*Спец_ть*	*Сем_р*	*Код_предм*	*Предмет*	*Оценка*
10001	*Иванов И.И.*	*611*	*6*	*Программная иженерия*	*1*	*1*	*Программирование*	*5*
10001	Иванов И.И.	611	6	Программная иженерия	1	2	История Украины	5
10001	Иванов И.И.	611	6	Программная иженерия	1	3	Высшая математика	4
10001	Иванов И.И.	611	6	Программная иженерия	2	1	Программирование	5
10001	Иванов И.И.	611	6	Программная иженерия	2	4	Украинский язык	4
10002	Петров П.П.	515	5	АСУ	1	1	Программирование	4
10002	Петров П.П.	515	5	АСУ	1	4	Украинский язык	5
10002	Петров П.П.	515	5	АСУ	2	5	Английский язык	4
10002	Петров П.П.	515	5	АСУ	2	3	Высшая математика	4
10003	Сидоров С.С.	613	6	Экономика предприятия	1	2	История Украины	4
10003	Сидоров С.С.	613	6	Экономика предприятия	1	5	Английский язык	4
10003	Сидоров С.С.	613	6	Экономика предприятия	1	6	Экономика	5
10003	Сидоров С.С.	613	6	Экономика предприятия	2	7	Бухгалтерский учет	4
10003	Сидоров С.С.	613	6	Экономика предприятия	2	5	Английский язык	5
10004	Николаев Н.Н.	614	6	Финансы	1	4	Украинский язык	3
10004	Николаев Н.Н.	614	6	Финансы	1	2	История Украины	4
10004	Николаев Н.Н.	614	6	Финансы	2	3	Высшая математика	3
10004	Николаев Н.Н.	614	6	Финансы	2	7	Бухгалтерский учет	4
10005	Алешин А.А.	513	5	АСУ	1	1	Программирование	5
10005	Алешин А.А.	513	5	АСУ	1	4	Украинский язык	4
10005	Алешин А.А.	513	5	АСУ	2	5	Английский язык	5
10005	Алешин А.А.	513	5	АСУ	2	3	Высшая математика	5

Так как каждый студент сдает целый набор дисциплин в процессе сессии, то первичным ключом отношения Студент_сессия может быть такой набор атрибутов (ном_зк, сем_р, код_предм), который однозначно определяет каждую строку рассматриваемого отношения:

Студент_сессия (ном_зк, ФИО, ном_гр, фак_т, спец_ть, сем_р, код_предм, предмет, оценка).

Заметим, что атрибуты ФИО, ном_гр,фак_т,спец_ть зависят только от части первичного ключа — от значения атрибута ном_зк, а атрибут предмет – от значения атрибута код_предм. Поэтому мы должны констатировать наличие неполных функциональных зависимостей в данном отношении. А значит отношение Студент_сессия (табл. 2.2) не находится в 2NF.

Для приведения данного отношения к 2NF следует разбить его на проекции, при этом должно быть соблюдено условие восстановления исходного отношения без потерь. Такими проекциями могут быть следующие отношения:

Студент (ном_зк, ФИО, ном_гр, фак_т, спец_ть)
Предметы (код_предм, предмет)
Сессия (ном_зк, сем_р, код_предм, оценка)

Этот набор отношений не содержит неполных функциональных зависимостей, т.е. в каждом из полученных отношений все неключевые атрибуты функционально полно зависят от первичного ключа. Можно сделать вывод о том, что эти отношения находятся в 2NF.

Проанализируем отношения Студент, Предметы и Студент_сессия на соответствие 3NF. Для этого необходимо проверить данные отношения на наличие транзитивных зависимостей.

Группа, в которой учится студент, однозначно определяет факультет, на котором он учится, а также специальность. Таким образом, в отношении Студент имеются следующие функциональные зависимости:

ном_зк → ном_гр
ном_гр → фак_т
ном_гр → спец_ть.

А значит, в отношении Студент имеются следующие транзитивные зависимости:

ном_зк → ном_гр → фак_т
ном_зк → ном_гр → спец_ть.

Следовательно, отношение Студент не находится в 3NF и требуется его декомпозиция:

Студент (ном_зк, ФИО, ном_гр)
Группа_факультет (ном_гр, фак_т)
Группа_специальность (ном_гр, спец_ть)
Предметы (код_предм, предмет)
Сессия (ном_зк, сем_р, код_предм, оценка)

Так как в полученных отношениях Студент, Группа_факультет, Группа_специальность, Предметы и Сессия отсутствуют транзитивные зависимости, то можно утверждать, что данные отношения находятся в 3NF.

Проанализируем отношения Студент, Группа_факультет, Группа_специальность, Предметы и Сессия на соответствие BCNF. Так как данные отношения не имеют альтернативных ключей, то соответственно они находятся в BCNF.

В большинстве случаев достижение третьей нормальной формы или даже формы Бойса-Кодда считается достаточным для реальных проектов баз данных.

Однако, если проверить отношения Студент, Группа_факультет, Группа_специальность, Предметы и Сессия на соответствие 4NF и 5NF, то можно сделать следующий вывод: данные отношения не имеют многозначных зависимостей, поэтому они находятся в 4NF; отсутствие зависимостей проекции-соединения в рассматриваемых отношениях свидетельствует о соответствии их 5NF.

3. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ НА ОСНОВЕ КОНЦЕПТУАЛЬНОГО МОДЕЛИРОВАНИЯ

3.1. Процесс проектирования: от концепции к физической модели данных

Процесс проектирования БД [6-12] представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели. В общем случае можно выделить следующие этапы проектирования (рис. 3.1) [6]:

- системный анализ и словесное описание информационных объектов предметной области (подготовительный уровень);

- проектирование инфологической модели предметной области – частично формализованное описание объектов предметной области в терминах некоторой семантической модели;
- даталогическое (или логическое) проектирование БД, то есть описание БД в терминах принятой даталогической модели данных;
- физическое проектирование БД, то есть выбор эффективного размещения БД на внешних носителях для обеспечения наиболее эффективной работы приложения.

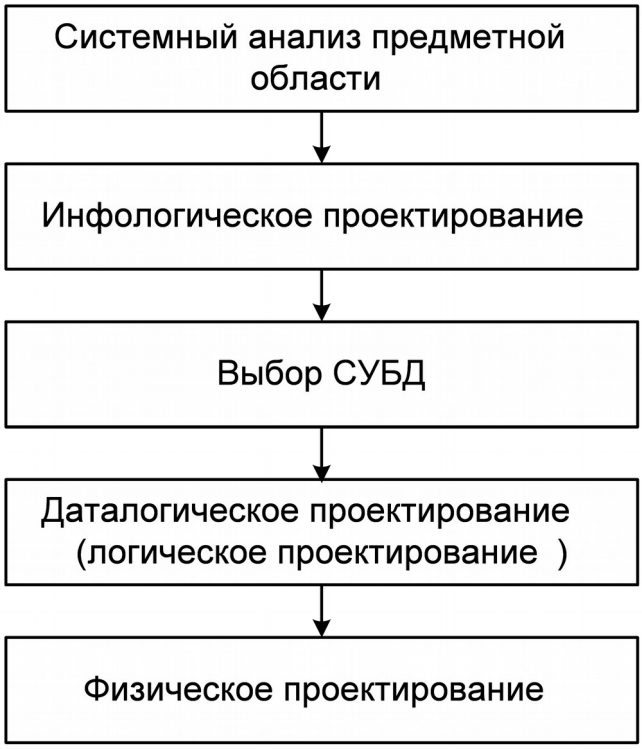


Рисунок 3.1 – Этапы проектирования БД

От того, насколько квалифицированно спроектирована БД, зависят производительность информационной системы и полнота обеспечения функциональных потребностей пользователей и прикладных программ. Неудачно спроектированная БД может усложнить процесс разработки прикладного программного обеспечения, обусловить необходимость использования более сложной логики, которая, в свою очередь, увеличит время реакции системы, а в дальнейшем может привести к необходимости перепроектирования логической модели БД. Реструктуризация или внесение изменений в логическую модель БД это очень нежелательный процесс, поскольку он является причиной необходимости модификации или даже перепрограммирования отдельных задач.

Рассмотрим понятие модели данных и классификацию моделей данных применительно к каждому из трех уровней проектирования БД.

Модель данных – это некоторая абстракция, которая, будучи приложима к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними.

В соответствии с рассмотренной ранее трехуровневой архитектурой мы сталкиваемся с понятием модели данных по отношению к каждому уровню. Классификация моделей данных, применяемых к тому или иному уровню проектирования БД, приведена на рис. 3.2.

Рисунок 3.2 – Классификация моделей данных

Из приведенной классификации моделей данных видно, что, с одной стороны, на каждом этапе проектирования БД необходимо обоснованное принятие решения относительного того, какую именно модель данных необходимо выбрать, а с другой – одна и та же инфологическая модель может быть переложена на разные даталогические модели; в свою очередь, конкретная даталогическая модель может быть переложена на разные физические модели данных. Это позволяет сравнивать применения различных моделей данных к одной предметной области.

3.2. Системный анализ предметной области

С точки зрения проектирования БД в рамках системного анализа необходимо осуществить первый этап, то есть провести подробное словесное описание объектов предметной области и реальных связей, которые присутствуют между описываемыми объектами. Желательно, чтобы данное описание позволяло корректно определить все взаимосвязи между

объектами предметной области.

Чаще всего на практике при выполнении анализа предметной области рекомендуется: с одной стороны, ориентироваться на конкретные задачи или функциональные потребности пользователей, а с другой, – учитывать возможность наращивания новых задач или даже новых приложений.

Системный анализ должен заканчиваться: а) подробным описанием информации об объектах предметной области, которая требуется для решения конкретных задач и должна будет храниться в БД; б) формулировкой конкретных задач, которые будут решаться с использованием данной БД с кратким описанием алгоритмов их решения; в) описанием выходных документов, которые должны генерироваться в системе; г) описанием входных документов, которые служат основанием для заполнения данными БД.

Рассмотрим пример системного анализа предметной области, приведенный в [7].

Пусть требуется разработать информационную систему для автоматизации учета получения и выдачи книг в библиотеке. Система должна предусматривать режимы ведения системного каталога, отражающего перечень областей знаний, по которым имеются книги в библиотеке. Внутри библиотеки области знаний в систематическом каталоге могут иметь уникальный внутренний номер и полное наименование. Каждая книга может содержать сведения из нескольких областей знаний. Каждая книга в библиотеке может присутствовать в нескольких экземплярах. Каждая книга, хранящаяся в библиотеке, характеризуется следующими параметрами:

- уникальный шифр;
- название;
- фамилии авторов (могут отсутствовать);
- место издания (город);
- издательство;
- год издания;
- количество страниц;
- стоимость книги;
- количество экземпляров книги в библиотеке.

Книги могут иметь одинаковые названия, но они различаются по своему уникальному шифру (ISBN).

В библиотеке ведется картотека читателей.

На каждого читателя в картотеку заносятся следующие сведения:

- фамилия, имя, отчество;
- домашний адрес;
- телефон (будем считать, что у нас два телефона — рабочий и домашний);
- дата рождения.

Каждому читателю присваивается уникальный номер читательского билета.

Каждый читатель может одновременно держать на руках не более пяти книг. Читатель не должен одновременно держать более одного экземпляра книги одного названия.

Каждая книга в библиотеке может присутствовать в нескольких экземплярах. Каждый экземпляр имеет следующие характеристики:

- уникальный инвентарный номер;
- шифр книги, который совпадает с уникальным шифром из описания книг;
- место размещения в библиотеке.

При выдаче книги на руки читателю в библиотеке остается специальный вкладыш, в котором указаны:

- номер билета читателя, взявшего книгу;
- дата выдачи книги;
- дата возврата книги.

Предусмотреть такие ограничения на информацию в системе:

1. Книга может быть без единого автора.
2. В библиотеку могут быть записаны читатели не моложе 17 лет.
3. В библиотеке находятся книги, изданные с 1960 по текущий год.
4. Каждому читателю выдается на руки не более пяти книг.
5. Каждый читатель при регистрации в библиотеке должен оставить для связи номер своего телефона: он может быть рабочий или домашний.
6. Каждая область знаний может содержать ссылки на множество книг, но каждая книга может относиться к различным областям знаний.

С данной информационной системой должны работать такие группы пользователей:

- библиотекари;
- читатели;
- администрация библиотеки.

При работе с системой библиотекарь должен:

1. Принимать новые книги и регистрировать их в библиотеке.
2. Относить книги к одной или к нескольким областям знаний.
3. Проводить каталогизацию книг, т.е. назначать новые инвентарные номера вновь принятым книгам, и размещать их на полках библиотеки, запоминая место размещения каждого экземпляра.
4. Проводить дополнительную каталогизацию, если поступило несколько экземпляров книги, которая уже есть в библиотеке, при этом информация о книге в предметный каталог не вносится, а каждому новому экземпляру присваивается новый инвентарный номер и для него определяется место на полке библиотеки.
5. Проводить списание старых и не пользующихся спросом книг. Списывать книги можно при условии, что ни один экземпляр не находится у читателей. Списание проводится по специальному акту, который

утверждается администрацией библиотеки.

6. Вести учет выданных читателям книг, при этом предполагается два режима работы: выдача книг читателю и прием возвращаемых им книг. При этом фиксируется дата выдачи и какой экземпляр книги был выдан данному читателю, а также к какому сроку читатель должен вернуть этот экземпляр книги. Наличие свободного экземпляра книги и его конкретный номер могут определяться по заданному уникальному шифру этой книги или же инвентарный номер может быть известен заранее. Не требуется вести "историю" чтения книг, т.е. следует отражать только текущее состояние библиотеки. Инвентарный номер возвращаемой книги должен соответствовать выданному номеру. Возвращенная книга занимает свое прежнее место на полке библиотеки.

7. Списывать утерянные читателем книги по специальным актам списания или замены, подписанным администрацией библиотеки.

8. Закрывать абонемент читателя, т.е. удалять данных о нем, если читатель хочет выписаться из библиотеки и не является ее должником (за ним не числится ни одной библиотечной книги).

Читатель должен иметь возможность:

1. Просматривать системный каталог, т.е. перечень всех областей знаний книг, которые есть в библиотеке.

2. По выбранной области знаний получить полный перечень книг, которые числятся в библиотеке.

3. Для выбранной книги получить инвентарный номер свободного экземпляра книги или сообщение о том, что свободных экземпляров книги нет. В случае отсутствия свободных экземпляров книги читатель должен иметь возможность узнать дату ближайшего предполагаемого возврата экземпляра данной книги. Читатель не может узнать данные о том, у кого в настоящий момент находятся на руках экземпляры данной книги.

4. Для указанного автора получить список его книг, которые числятся в библиотеке.

Администрация библиотеки должна иметь возможность получать:

- сведения о читателях-должниках библиотеки, которые не вернули в срок взятые книги;
- сведения о книгах, которые не являются популярными, т.е. ни один экземпляр которых не находится на руках у читателей;
- сведения о стоимости конкретной книги, для того чтобы возместить стоимость утерянной книги или заменить ее другой книгой;
- сведения о наиболее популярных книгах, т.е. таких, все экземпляры которых находятся на руках у читателей.

Итак, совсем небольшой пример показывает, что перед началом разработки информационной системы с базой данных необходимо иметь точное представление о том, что же должно выполняться в нашей системе, какие пользователи в ней будут работать, какие задачи будет решать каждый пользователь. И это правильно, ведь прежде чем

построить здание, заранее предполагают, для каких целей оно предназначено, в каком климате оно будет стоять, на какой почве, и в зависимости от этого проектировщики могут предложить тот или иной проект. Но, к сожалению, очень часто по отношению к базам данных считается, что все можно определить потом, когда проект системы уже создан. Отсутствие четких целей создания БД может свести на нет все усилия разработчиков, и проект БД получится «плохим», неудобным, не соответствующим ни реально моделируемому объекту, ни задачам, которые должны решаться с использованием данной БД.

3.3. Построение концептуальной (инфологической) модели предметной области

3.3.1. Семантические модели данных

Зачем нужна инфологическая модель и какую пользу она дает проектировщикам?

Инфологическое проектирование, прежде всего, связано с попыткой представления семантики предметной области в модели БД. Реляционная модель данных в силу своей простоты и лаконичности не позволяет отобразить семантику, то есть смысл предметной области. Ранние теоретико-графовые модели в большей степени отображали семантику предметной области. Они в явном виде определяли иерархические связи между объектами предметной области.

Проблема представления семантики давно интересовала разработчиков, и в семидесятых годах было предложено несколько моделей данных, названных семантическими моделями. К ним можно отнести семантическую модель данных, предложенную Хаммером (Hammer) и Мак-Леоном (McLeon) в 1981 году, функциональную модель данных Шипмана (Shipman), также созданную в 1981 году, модель «сущность – связь», предложенную Ченом (Chen) в 1976 году, и ряд других моделей. У всех моделей были свои положительные и отрицательные стороны, но испытание временем выдержала только последняя.

В настоящее время именно модель Чена «сущность – связь», или «Entity Relationship», стала фактическим стандартом при инфологическом моделировании баз данных. Общепринятым стало сокращенное название ER-модель, большинство современных CASE-средств содержат инструментальные средства для описания данных в формализме этой модели. Кроме того, разработаны методы автоматического преобразования проекта БД из ER-модели в реляционную, при этом преобразование выполняется в даталогическую модель, соответствующую конкретной СУБД. Все CASE-системы имеют развитые

средства документирования процесса разработки БД, автоматические генераторы отчетов позволяют подготовить отчет о текущем состоянии проекта БД с подробным описанием объектов БД и их отношений как в графическом виде, так и в виде готовых стандартных печатных отчетов, что существенно облегчает ведение проекта.

В настоящее время не существует единой общепринятой системы обозначений для ER-модели, и разные CASE-системы [13-15] используют разные графические нотации, но разобравшись в одной, можно легко понять и другие нотации.

3.3.2. Базовые понятия ER-модели

Модель «сущность – связь» или ER-модель является наиболее известным представителем класса семантических (концептуальных, инфологических) моделей предметной области.

ER-модель является одной из самых простых визуальных моделей данных. Она позволяет обозначить структуру «крупными мазками» в общих чертах. Это общее описание структуры называется ER-диаграммой или онтологией выбранной предметной области (area of interest).

Основные преимущества ER-диаграмм:

- наглядность;
- модель позволяет проектировать базы данных с большим количеством объектов и атрибутов;
- ER-модели реализованы во многих системах автоматизированного проектирования баз данных (например, ERWin).

Как любая модель, модель «сущность – связь» имеет несколько базовых понятий, из которых строятся уже более сложные объекты по заранее определенным правилам.

Эта модель в наибольшей степени согласуется с концепцией объектно-ориентированного проектирования, которая в настоящий момент несомненно является базовой для разработки сложных программных систем.

В основе ER-модели лежат следующие базовые понятия:

- сущность;
- связь;
- тип связи (или множественность, мощность связи);
- обязательность связи (или класс принадлежности связи);
- категоризация сущностей.

Сущность – класс моделируемых однотипных объектов. Сущность имеет имя, уникальное в пределах моделируемой системы. Так как сущность соответствует некоторому классу однотипных объектов, то предполагается, что в системе существует множество экземпляров данной сущности.

Атрибут сущности. Объект, которому соответствует понятие

сущности, имеет свой набор атрибутов-характеристик, определяющих свойства данного представителя класса. При этом набор атрибутов должен быть таким, чтобы можно было различать конкретные экземпляры сущности.

Опишем сущность *Сотрудник*. Определим набор атрибутов для сущности *Сотрудник*, которые характеризуют экземпляры данной сущности. Экземпляром этой сущности будет являться конкретный сотрудник фирмы.

Чтобы сформировать набор атрибутов сущности *Сотрудник*, необходимо ответить на вопрос: Что характеризует конкретного сотрудника фирмы? – Табельный номер, фамилия, имя, отчество, дата рождения, домашний адрес, домашний телефон.

Отметим, что набор атрибутов, однозначно идентифицирующий конкретный экземпляр сущности, называют ключевым. Для сущности *Сотрудник* ключевым будет атрибут «табельный номер», поскольку для всех сотрудников данной фирмы табельные номера будут различны.

Связи – бинарные ассоциации между сущностями, показывающие, каким образом сущности соотносятся или взаимодействуют между собой. Связь может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). Она показывает, как связаны экземпляры сущностей между собой. Если связь устанавливается между двумя сущностями, то она определяет взаимосвязь между экземплярами одной и другой сущности.

Связь между сущностями характеризуется своим типом. По типу различают связи:

- один-к-одному (1:1);
- один-ко-многим (1:N);
- многие-ко-многим (N:M).

Связь 1:1 означает, что экземпляр одной сущности связан только с одним экземпляром другой сущности. Например, между сущностями *Сотрудник* и *Паспорт* может быть установлена связь 1:1, так как сотрудник имеет один паспорт (наличие загранпаспорта не берется во внимание), и в то же время конкретный паспорт может принадлежать только одному сотруднику фирмы.

Связь 1:N означает, что один экземпляр сущности, расположенный слева по связи, может быть связан с несколькими экземплярами сущности, расположенными справа по связи. Например, между сущностями *Сотрудник* и *Заказы* может быть установлена связь 1:N, так как один сотрудник может разместить много заказов клиентов фирмы, но конкретный заказ может быть оформлен только одним сотрудником фирмы.

Связь N:M означает, что один экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и наоборот, один экземпляр второй сущности может быть связан с несколькими

экземплярами первой сущности. Например, между сущностями *Сотрудник* и *Отдел* может быть установлена связь N:M, если из предметной области следует, что сотрудник может работать в нескольких отделах и в каждом отделе может числиться много сотрудников.

Между двумя сущностями может быть задано сколько угодно связей с разными смысловыми нагрузками, при этом установленные связи могут иметь разные типы множественности. Что определяется смысловой нагрузкой связи.

Обязательность связи. Связь любого из этих типов может быть обязательной, если в данной связи должен участвовать каждый экземпляр сущности, и необязательной, – если не каждый экземпляр сущности должен участвовать в данной связи. При этом связь может быть обязательной, с одной стороны, и необязательной – с другой.

Рассмотрим связь между сущностями *Сотрудник* и *Заказы.* Из предметной области известно, что заказ обязательно оформляется сотрудником фирмы, в то же время сотрудник может не иметь размещенных им заказов (например, он только устроился на работу). Таким образом, между сущностями *Сотрудник* и *Заказы* будет установлена связь, обязательная со стороны сущности *Сотрудник* и необязательная – со стороны сущности *Заказы*.

Категоризация сущностей. Кроме того, в ER-модели допускается принцип категоризации сущностей. Это значит, что, как и в объектно-ориентированных языках программирования, вводится понятие подтипа сущности, то есть сущность может быть представлена в виде двух или более своих подтипов – сущностей, каждая из которых может иметь общие атрибуты и отношения и/или атрибуты и отношения, которые определяются однажды на верхнем уровне и наследуются на нижнем уровне.

В заключение отметим, что на этапе перехода к реализации данной ER-диаграммы в виде реальной информационной системы или программы, происходит отображение ER-модели в более детальную модель данных реляционной (объектной, сетевой, логической, или др.) базы данных, которая называется даталогической моделью данных по отношению к исходной ER-диаграмме.

3.3.3. Нотации для построения ER-диаграмм

В настоящий момент не существует единой общепринятой системы обозначений для ER-модели, и разные CASE-системы используют разные графические нотации. Первый вариант модели сущность-связь был предложен Питером Ченом. В дальнейшем многими авторами были разработаны свои варианты подобных моделей (нотация Мартина, нотация IDEF1X, нотация Баркера и др.). Кроме того, различные программные средства, реализующие одну и ту же нотацию, могут

отличаться своими возможностями. По сути все нотации диаграмм «сущность-связь» исходят из одной идеи – рисунок всегда нагляднее текстового описания. Все такие диаграммы используют графическое изображение сущностей предметной области, их свойств (атрибутов) и взаимосвязей между сущностями.

Нотация Питера Чена

Рассмотрим изображение основных элементов ER-диаграммы в нотации Питера Чена (табл. 3.1).

Для изображения сущности используются прямоугольники, внутри которых указывается имя сущности, выражаемое существительным. Атрибут сущности изображается в виде овала, который связывается линией с соответствующей сущностью. Связи отображаются в виде ромбов. Ассоциируемые сущности связываются линиями. Если связь не является обязательной, то линия – пунктирная. Возле каждой сущности на линии, соединяющей ее со связью, цифрами указывается класс принадлежности. На рис. 3.3 приведен пример ER-диаграммы в нотации Питера Чена.

Таблица 3.1 – Элементы ER-диаграммы в нотации Чена

Элемент диаграммы	**Обозначение**
	независимая сущность
	зависимая сущность
	родительская сущность в иерархической связи
	связь
	идентифицирующая связь
	Атрибут

Окончание табл. 3.1

Элемент диаграммы	Обозначение
	первичный ключ
	внешний ключ
	многозначный атрибут
	получаемый (наследуемый) атрибут в иерархических связях

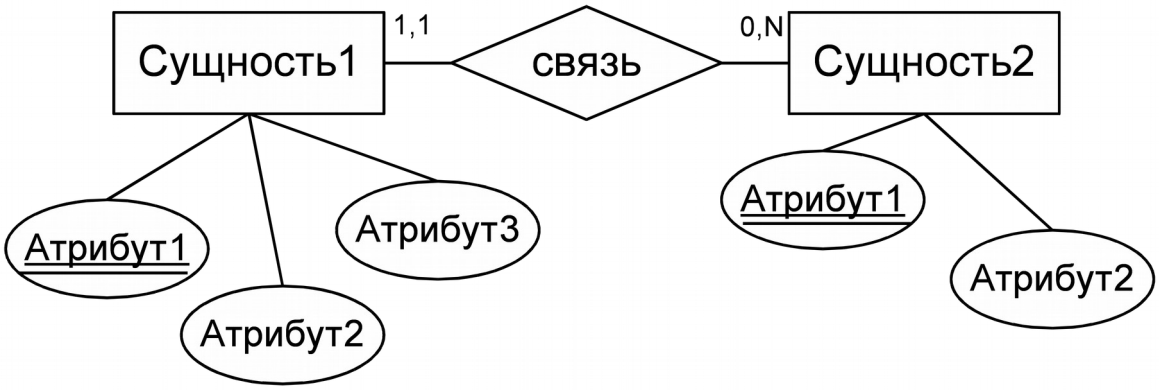


Рисунок 3.3 – ER-диаграмма в нотации Чена

Нотация Мартина

Согласно данной нотации (табл. 3.2) *сущность* изображается в виде прямоугольника, содержащего её имя, выражаемое существительным. Имя сущности должно быть уникальным в рамках одной модели.

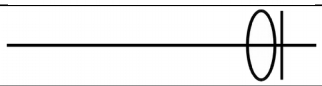
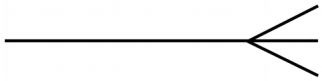
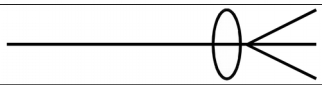
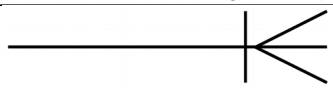
Связь изображается линией, которая связывает две сущности, участвующие в связи. Степень конца связи указывается графически, множественность связи изображается в виде «вилки» на конце связи. Модальность связи также изображается графически – необязательность связи помечается кружком на конце связи. Наименование обычно выражается одним глаголом в изъявительном наклонении настоящего времени: «Имеет», «Принадлежит» и т.д.; или глаголом с поясняющими словами: «Включает_в_себя», и т.п. Наименование может быть одно для

всей связи или два – для каждого из концов связи. Во втором случае название левого конца связи указывается над линией связи, а правого – под линией. Каждое из названий располагается рядом с сущностью, к которой оно относится.

Атрибуты сущности записываются внутри прямоугольника, изображающего сущность и выражаются существительным в единственном числе (возможно с уточняющими словами). Среди атрибутов выделяется ключ сущности. Ключевые атрибуты подчеркиваются.

На рис. 3.4 приведен фрагмент ER-диаграммы в нотации Мартина.

Таблица 3.2 – Элементы ER-диаграммы в нотации Мартина

Элемент диаграммы	**Обозначение**
	независимая сущность
	зависимая сущность
	родительская сущность в иерархической связи
	кардинальность связи – не установлена
	кардинальность связи – 1,1
	кардинальность связи – 0,1
	кардинальность связи – M,N
	кардинальность связи – 0,N
	кардинальность связи – 1,N

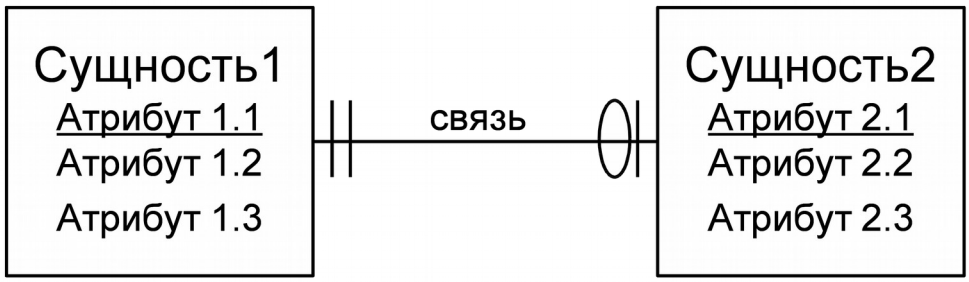


Рисунок 3.4 – Фрагмент ER-диаграммы в нотации Мартина

Нотация Баркера

Согласно данной нотации (см. табл. 3.3) сущности обозначаются прямоугольниками, внутри которых приводится список атрибутов. Ключевые атрибуты отмечаются символом # (решетка). Связи обозначаются линиями с именами, место соединения связи и сущности определяет кардинальность связи.

На рис. 3.5 приведен фрагмент ER-диаграммы в нотации Баркера.

Таблица 3.3 – Элементы ER-диаграммы в нотации Баркера

Элемент диаграммы	**Обозначение**
имя	сущность
	связь 1:1
	связь 1:N
	необязательная связь
	обязательная связь

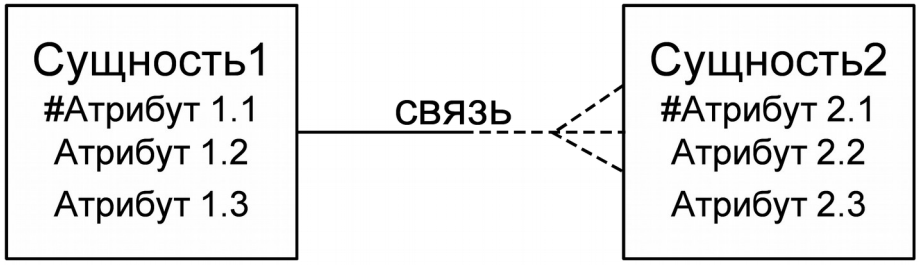


Рисунок 3.5 – Фрагмент ER-диаграммы в нотации Баркера

Для обозначения отношения категоризации вводится элемент «дуга» (рис. 3.6).

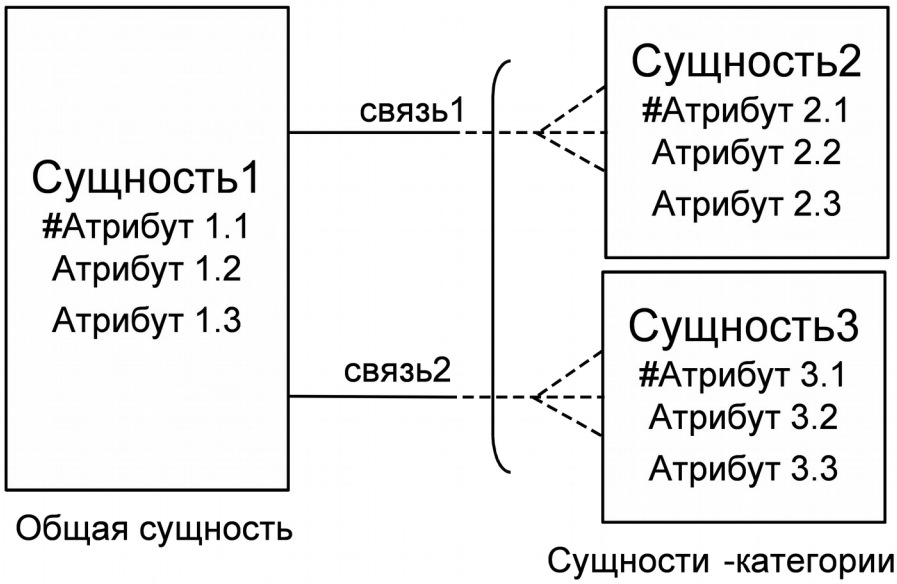


Рисунок. 3.6 – Обозначение категоризации в нотации Баркера

Нотация IDEF1X

Обозначения конструктивных элементов концептуальной схемы базы данных нотации IDEF1X приведены в табл. 3.4.

Список атрибутов приводится внутри прямоугольника, обозначающего сущность. Атрибуты, составляющие ключ сущности, группируются в верхней части прямоугольника и отделяются горизонтальной чертой (рис. 3.7).

Кроме того, в IDEF1X вводится понятие отношение категоризации, по смыслу эквивалентное иерархической связи. Обозначение отношения полной категоризации (сущности-категории составляют полное множество потомков родительской сущности) приведено на рис. 3.8, а. Обозначение отношения неполной категоризации (сущности-категории составляют неполное множество потомков общей сущности) приведено на рис. 3.8, б.

Таблица 3.4 – Элементы ER-диаграммы в нотации IDEF1X

Элемент диаграммы	Обозначение
имя	независимая сущность
имя	зависимая сущность
	идентифицирующая связь
	неидентифицирующая связь
Сущность 1 — Сущность 2	сущности 1 соответствует 0, 1 или много (M) сущностей 2
Сущность 1 — P — Сущность 2	сущности 1 соответствует 1 или M сущностей 2
Сущность 1 — Z — Сущность 2	сущности 1 соответствует 0 или 1 сущностей 2
Сущность 1 — 7 — Сущность 2	сущности 1 соответствуют ровно N (N=7) сущностей 2

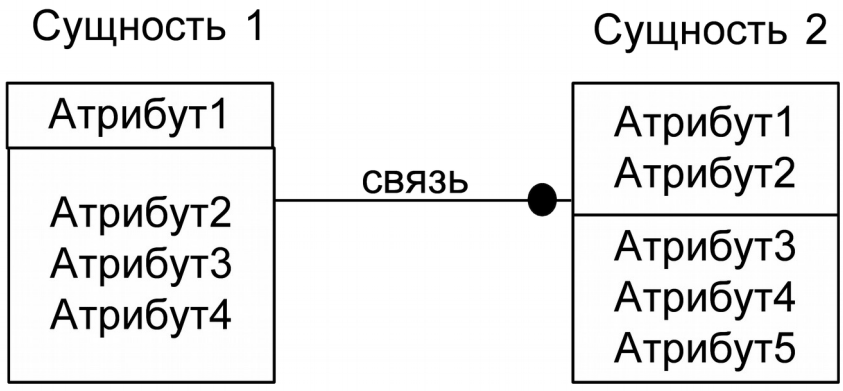


Рисунок 3.7 – Фрагмент ER-диаграммы в нотации IDEF1X

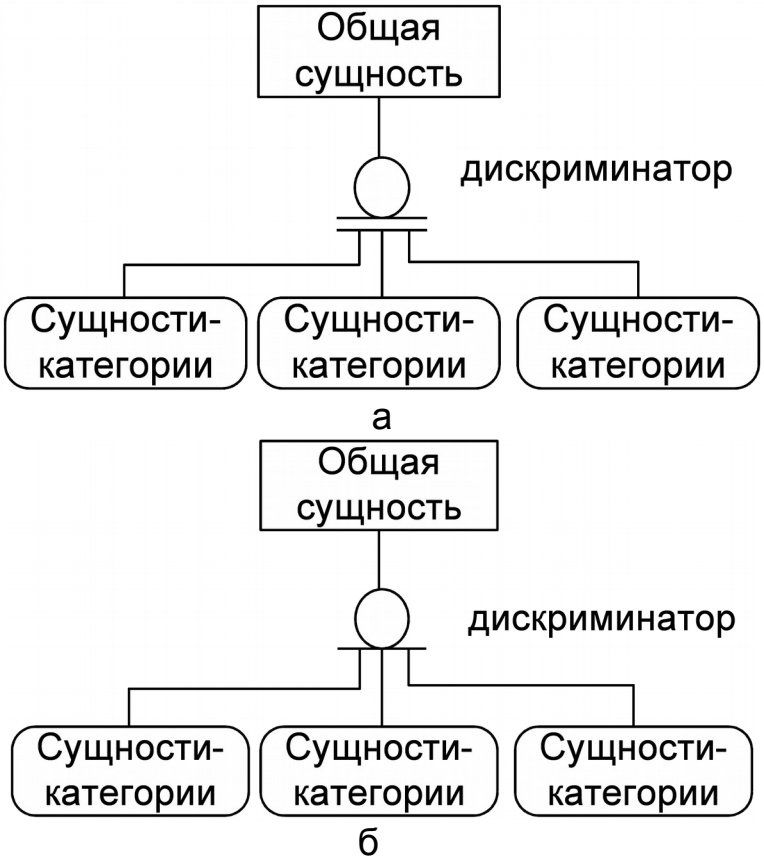


Рисунок 3.8 – Отношение полной (а) и неполной (б) категоризации в нотации IDEF1X

3.3.4. Пример построения ER-модели

В качестве примера спроектируем инфологическую модель системы в виде ER-модели, предназначенную для хранения информации о книгах и областях знаний, представленных в библиотеке (описание предметной области было приведено в разд. 3.2). Разработку модели начнем с выделения основных сущностей и связей между ними.

Итак, можно выделить сущность *Книга*. Что характеризует каждую книгу? – Название книги, автор, год издания, количество страниц, номер ISBN. Так как номер ISBN является уникальным для каждой книги, то именно данный атрибут будет являться ключом сущности *Книга*. Ну а как же издательство? Ведь каждая книга имеет издательство и место издания.

Введем сущность *Издательство* со следующими атрибутами: код издательства, название издательства и место издательства. Ключевым атрибутом в данной сущности будет выступать код издательства.

Между сущностями *Издательство* и *Книга* имеется связь 1:N, потому, что в одном конкретном издательстве может быть издано много книг, но книга с некоторым ISBN печатается в одном конкретном

издательстве.

Заметим, что каждый экземпляр сущности *Книги* соответствует не конкретной книге, стоящей на полке, а описанию некоторой книги, которое дается обычно в предметном каталоге библиотеки. Каждая книга может присутствовать в нескольких экземплярах, и это как раз те конкретные книги, которые стоят на полках библиотеки. Для того чтобы отразить это, мы должны ввести сущность *Экземпляры книги*, которая будет содержать описания всех экземпляров книг, которые хранятся в библиотеке. Каждый экземпляр сущности *Экземпляры книги* соответствует конкретной книге на полке. Что характеризует каждую конкретную книгу в библиотеке, стоящую на полке? – Инвентарный номер, наличие в библиотеке (книга может находиться или в библиотеке, или на руках у читателя), и в случае, если книга выдана, то необходимо хранить дату выдачи книги читателю и дату предполагаемого ее возврата. Так как инвентарный номер книги уникален, то именно данный атрибут и будет являться ключом сущности *Экземпляры книги*.

Между сущностями *Книги* и *Экземпляры книги* существует связь «один-ко-многим» (1:N), обязательная с двух сторон.

Опишем сущность *Читатель*. Определим набор атрибутов для сущности *Читатель*, которые характеризуют экземпляры данной сущности. Экземпляром сущности *Читатель* будет являться конкретный читатель библиотеки.

Чтобы сформировать набор атрибутов сущности *Читатель*, необходимо определить, что характеризует конкретного читателя библиотеки: фамилия, имя, отчество, дата рождения, домашний адрес, домашний телефон, место работы, должность, рабочий телефон.

Опишем сущность *Читательский билет*. Определим набор атрибутов для сущности *Читательский билет*, которые характеризуют экземпляры данной сущности. Экземпляром данной сущности будет являться выписанный читательский билет.

Итак, что характеризует конкретный читательский билет? – Номер читательского билета, дата, когда был выписан данный билет. Так как номер читательского билета уникален, то именно данный атрибут и будет являться ключом сущности *Читательский билет.*

Так как один читатель имеет только один читательский билет в некоторую библиотеку и конкретный билет может принадлежать только одному читателю, то между сущностями *Читатель* и *Читательский билет* имеется связь 1:1.

Из описания предметной области известно, что каждый читатель может держать на руках несколько экземпляров книг. Таким образом, между сущностями *Читатели* и *Экземпляры книги* существует связь 1:N. Но почему связь устанавливается не между сущностями *Читатели* и *Книги*? Потому что читатель берет из библиотеки конкретный экземпляр книги, а не просто книгу. В тоже время один и тот же экземпляр книги

может читаться многими читателями, поэтому между сущностями *Читатели* и *Экземпляры книги* существует связь 1:N и в другом направлении. Таким образом, между сущностями *Читатели* и *Экземпляры книги* существует связь N:M. Данная связь является необязательной с двух сторон, потому что читатель в данный момент может не иметь ни одной книги на руках, а с другой стороны, данный экземпляр книги может не находиться ни у кого, а просто стоять на полке в библиотеке.

Рассмотрим последнюю сущность в данном примере, которая будет соответствовать системному каталогу в библиотеке, с которого обычно начинается поиск книг, если не известны автор или название книги. Введем сущность *Системный каталог*. Определим набор атрибутов для сущности *Системный каталог*, которые характеризуют экземпляры данной сущности. Экземпляром сущности *Системный каталог* будет являться системный каталог рассматриваемой библиотеки.

Что характеризует системный каталог некоторой библиотеки? – Код области знаний, название области знаний. Атрибут «код области знаний» будет ключевым атрибутом сущности.

Из описания предметной области известно, что каждая книга может содержать сведения из нескольких областей знаний, а с другой стороны, в библиотеке может присутствовать множество книг, относящихся к одной и той же области знаний, поэтому между сущностями *Системный каталог* и *Книги* должна быть установлена связь N:M, обязательная с двух сторон. Действительно, в системном каталоге не должно присутствовать такой области знаний, сведения по которой не представлены ни в одной книге нашей библиотеки, противное было бы бессмысленно. И наоборот, каждая книга должна быть отнесена к одной или нескольким областям знаний для того, чтобы читатель мог ее быстрее найти.

Инфологическая модель предметной области «Библиотека» представлена на рис. 3.9.

3.4. Построение логической модели реляционной базы данных

Инфологическая модель используется на ранних стадиях разработки проекта. Если понимать язык условных обозначений, которые соответствуют категориям ER-модели, то ее можно легко «читать», следовательно, она доступна для анализа программистам-разработчикам, которые будут разрабатывать отдельные приложения. Она имеет однозначную интерпретацию, в отличие от некоторых предложений естественного языка, и поэтому здесь не может быть никакого недопонимания со стороны разработчиков.

Язык ER-модели стал условным общепринятым языком описания базы данных. Для ER-модели существует алгоритм однозначного

преобразования ее в реляционную модель данных, что позволило в дальнейшем разработать множество инструментальных систем, поддерживающих процесс разработки информационных систем, базирующихся на технологии баз данных.

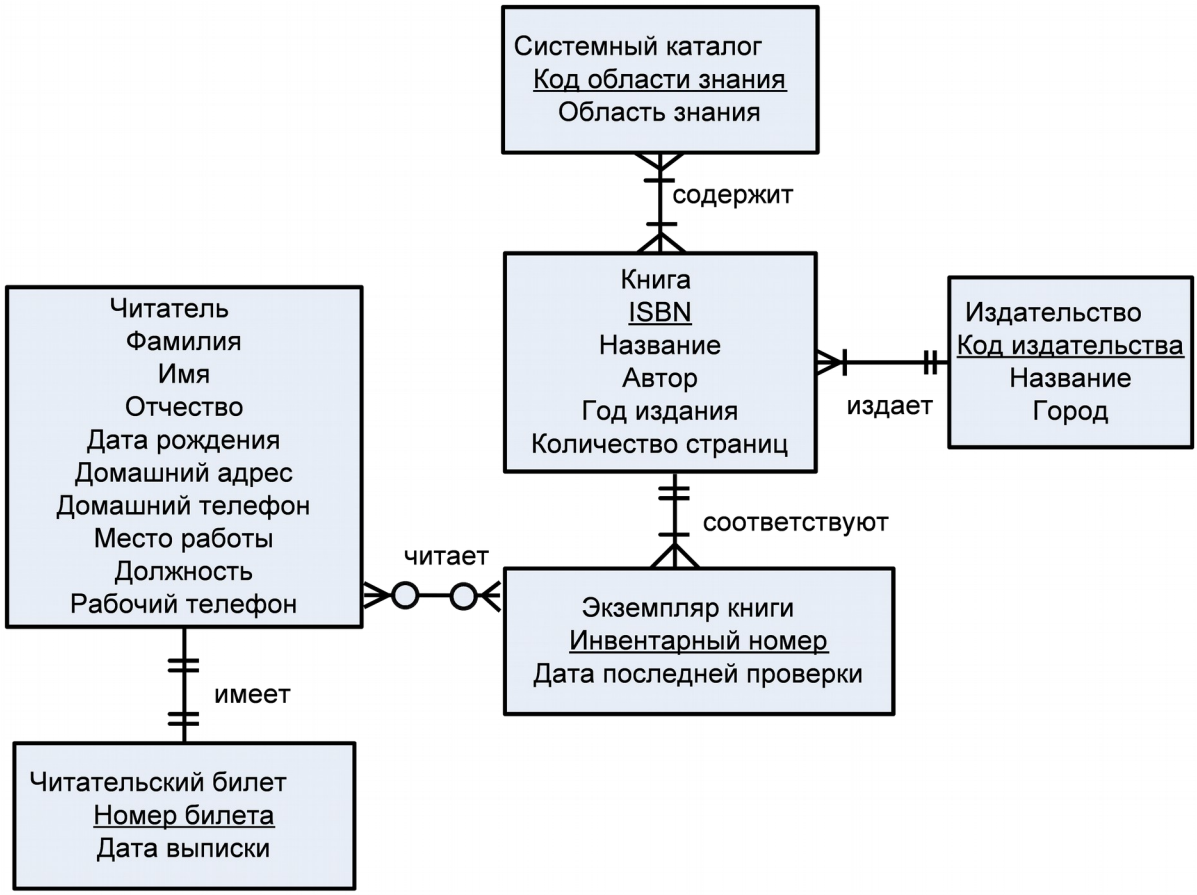


Рисунок 3.9 – ER-модель предметной области «Библиотека» в нотации Мартина

Логическое (даталогическое) проектирование базы данных – отображение инфологической модели на модель данных, используемую в конкретной СУБД, например на реляционную модель данных. Для реляционных СУБД даталогическая модель – набор таблиц, обычно с указанием ключевых полей, связей между таблицами. Если инфологическая модель построена в виде ER-диаграмм (или других формализованных средств), то даталогическое проектирование представляет собой построение таблиц по определённым формализованным правилам, а также нормализацию этих таблиц. Этот этап может быть в значительной степени автоматизирован.

3.4.1. Правила преобразования сущностей и их атрибутов

Каждой сущности ставится в соответствие отношение (таблица) реляционной модели данных. При этом имена сущности и отношения могут быть различными, потому что на имена сущностей могут не накладываться дополнительные синтаксические ограничения, кроме уникальности имени в рамках модели. Имена отношений могут быть ограничены требованиями конкретной СУБД, чаще всего эти имена являются идентификаторами в некотором базовом языке, они ограничены по длине и не должны содержать пробелов и некоторых специальных символов. Например, сущность может быть названа "Книги", а соответствующее ей отношение желательно назвать, например, BOOKS (без пробелов и латинскими буквами).

Каждый атрибут сущности становится атрибутом соответствующего отношения. Переименование атрибутов должно происходить в соответствии с теми же правилами, что и переименование отношений. Для каждого атрибута задается конкретный допустимый в СУБД тип данных и обязательность или необязательность данного атрибута (допустимость или недопустимость NULL значений для него).

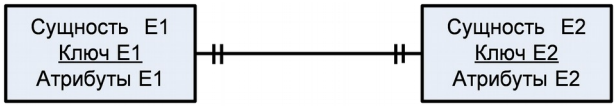
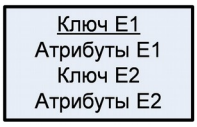
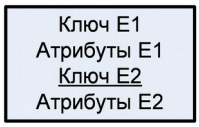
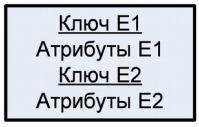
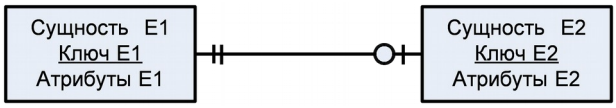
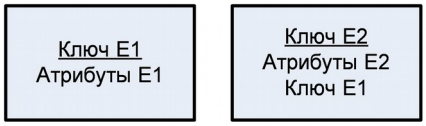
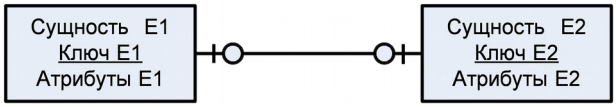
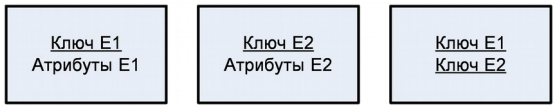
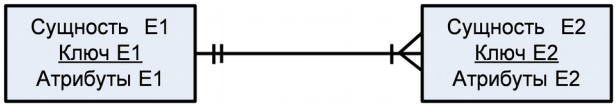
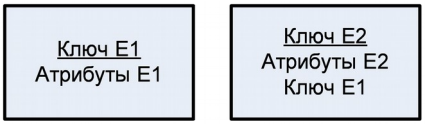
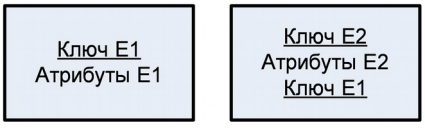
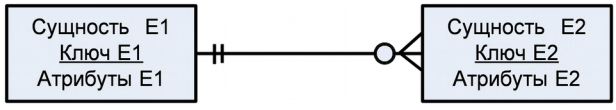
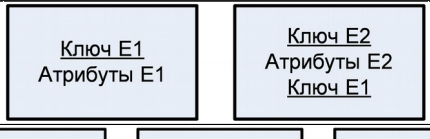
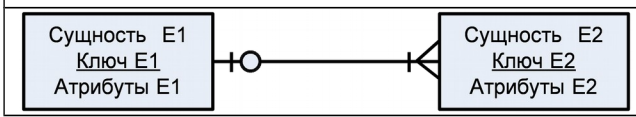
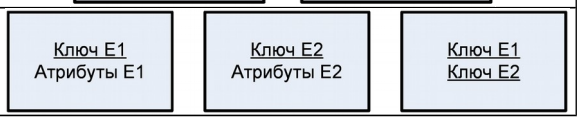
Первичный ключ сущности становится PRIMARY KEY соответствующего отношения. Атрибуты, входящие в первичный ключ отношения, автоматически получают свойство обязательности (NOT NULL).

3.4.2. Правила преобразования связей между сущностями

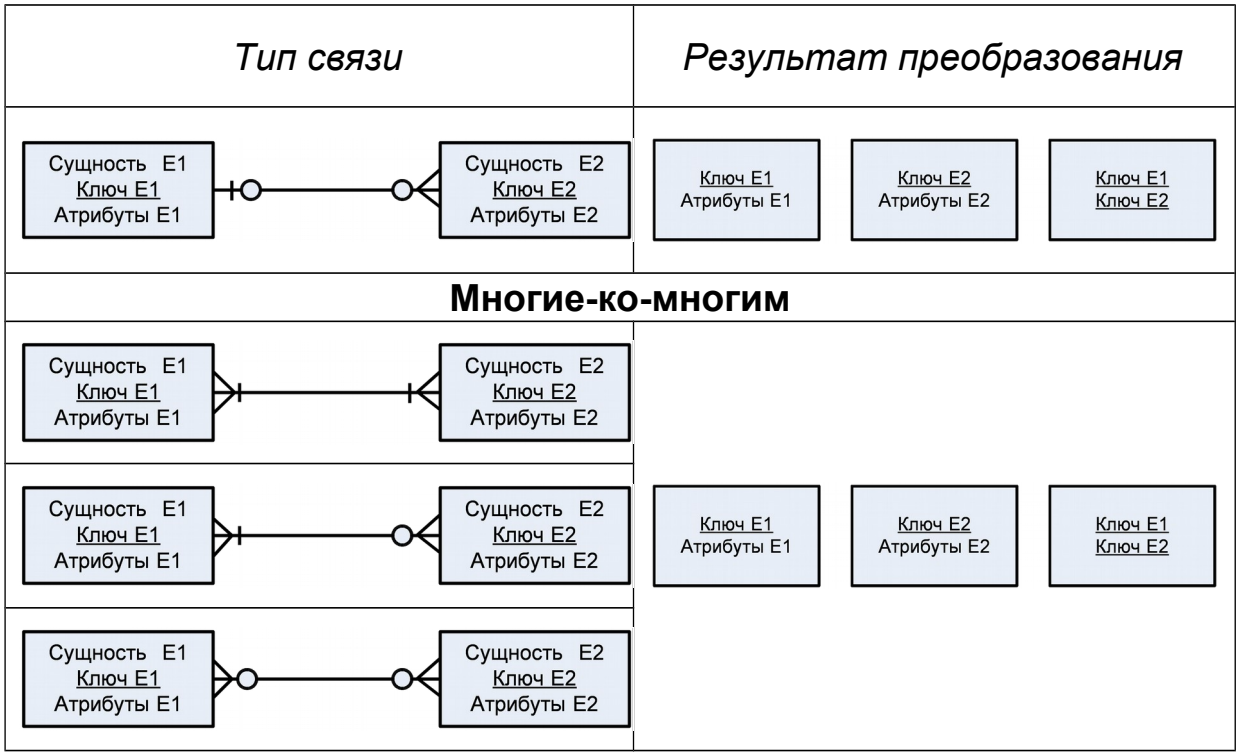
В данном подразделе будут рассмотрены правила преобразования связей типа «один-к-одному», «один-ко-многим» и «многие-ко-многим» между сущностями (см. табл. 3.5).

В соответствии с описанными правилами преобразуем ER-модель предметной области «Библиотека» в реляционную схему базы данных (рис. 3.10). На рис. 3.10, а направление стрелок показывает зависимость одной сущность от другой (например, сущность *Книга* зависит от сущности *Издательство*), тогда как на рис. 3.10, б направление стрелок показывает путь перехода (миграции) ключевого атрибута. Проектировщик может выбирать между этими способами отображения связей между таблицами логической схемы базы данных. Также направление стрелок будет определяться и выбранным CASE-средством для проектирования баз данных.

Таблица 3.5 – Преобразование связей

Тип связи	*Результат преобразования*
Один-к-одному	
	  
	
	
Один-ко-многим	
	
	
	 
	

Окончание табл. 3.5



3.5. Построение физической модели реляционной базы данных

Физическое проектирование базы данных – реализация даталогической модели средствами конкретной СУБД, а также выбор решений, связанных с физической средой хранения данных: выбор методов управления дисковой памятью, методов доступа к данным, методов сжатия данных и т.д. – эти задачи решаются в основном средствами СУБД и скрыты от разработчика БД.

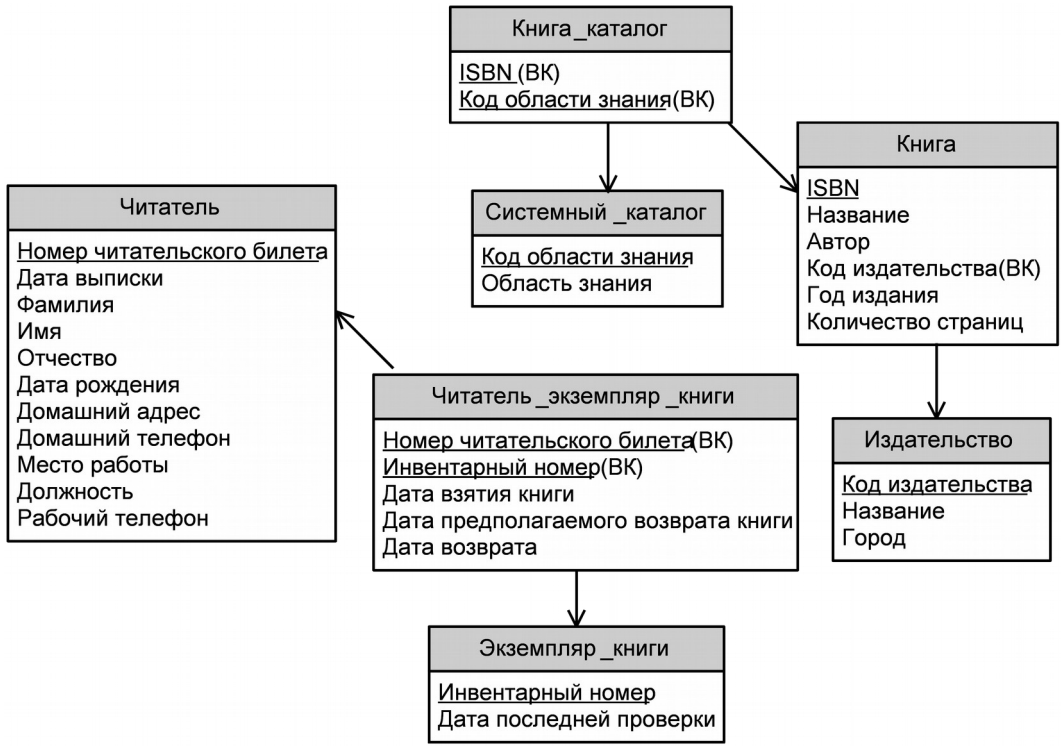
Для построения физической модели реляционной базы данных необходимо решить две задачи [12]:

- создать объекты для хранения данных;
- учесть влияние транзакций.

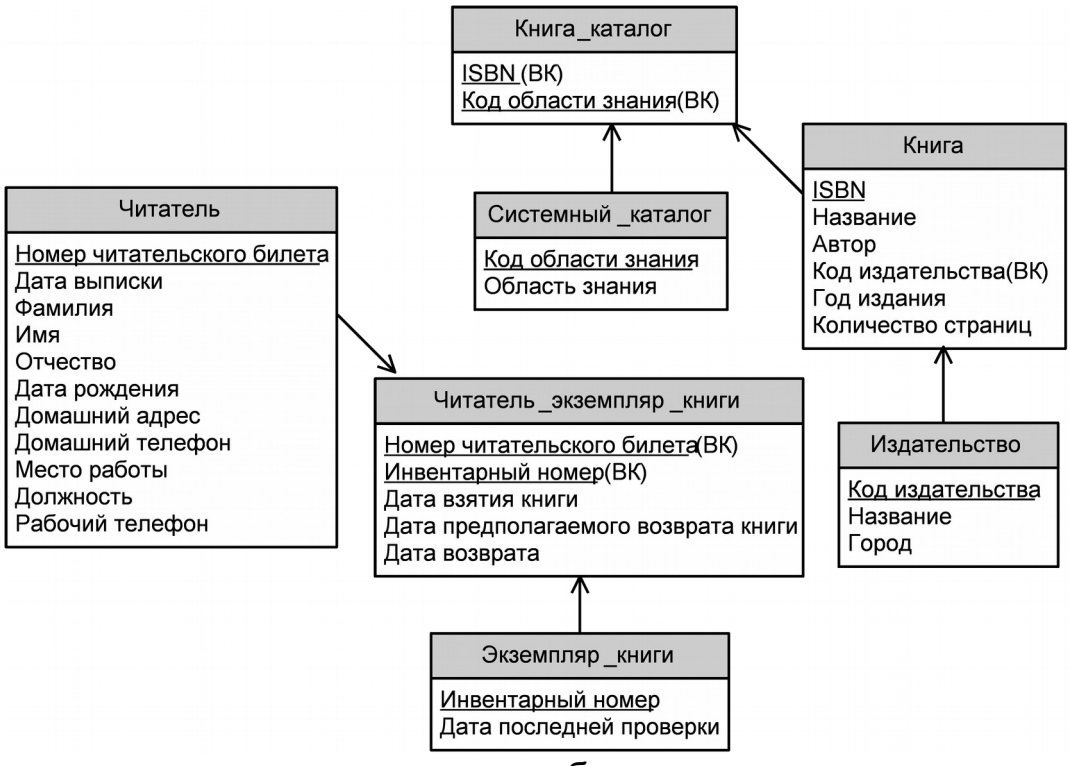
3.5.1. Обеспечение хранения данных в БД

Решение первой задачи создания физической модели базы данных требует от проектировщиков достижения таких целей:

- удовлетворить потребности в хранении данных предметной области в рамках реляционной модели данных, т.е. были созданы базовые таблицы для хранения информации обо всех сущностях предметной области;



а



б

Рисунок 3.10 – Логическая схема реляционной базы данных «Библиотека»

- удовлетворить требования целостности данных, т.е. были

определены типы колонок и наложены ограничения на значения колонок базовых таблиц, которые следовали бизнес-правилам предметной области;

- удовлетворить требования ссылочной целостности, т.е. в случае принятия решения о поддержке ссылочной целостности встроенными средствами СУБД были наложены ограничения ссылочной целостности на таблицы, исходя из бизнес-правил ссылочной целостности предметной области;
- частично удовлетворить требования независимости представления данных для конечного пользователя от характера физического хранения данных, т.е. была построена первая итерация внешней схемы.

Таким образом, первая задача построения физической модели данных сводится к созданию таблиц, индексов, представлений в базе данных, в которой будет храниться информация о сущностях предметной области. Решая эту задачу, проектировщик базы данных отображает отношения логической модели реляционной базы данных в таблицы и индексы реляционной базы данных. Для выполнения этой задачи используется подмножество команд SQL – язык определения данных DDL (Data Definition Language). Например, для СУБД Oracle эти действия могут быть выполнены в программе SQL*PLUS.

В рамках концепции трехуровневой архитектуры баз данных ANSI/SPARC эту задачу проектировщика базы данных называют еще созданием внутренней схемы. Результатом решения этой задачи является скрипт для создания таблиц и индексов, что составляет первоначальный прототип физической модели базы данных.

Остановимся более подробно на такой задаче хранения данных, как обеспечение ссылочной целостности.

Понятие ссылочной целостности

Ссылочная целостность (англ. referential integrity) – необходимое качество реляционной базы данных, заключающееся в отсутствии в любом её отношении внешних ключей, ссылающихся на несуществующие кортежи.

Связи между данными, хранимыми в разных отношениях, в реляционной БД устанавливаются с помощью использования внешних ключей. Благодаря наличию связей в реляционной БД можно хранить факты без избыточного дублирования, т.е. в нормализованном виде.

Ссылочная целостность может быть определена следующим образом. Дана пара отношений A и B, связанных внешним ключом. Первичный ключ отношения B — атрибут B.key. Внешний ключ отношения A, ссылающийся на B — атрибут A.b. Ссылочная целостность для пары отношений A и B имеет место тогда, когда выполняется такое условие:

для каждого кортежа отношения A существует соответствующий кортеж отношения B, т.е. кортеж, у которого (B.key = A.b).

База данных обладает свойством ссылочной целостности, когда для любой пары связанных внешним ключом отношений в ней условие ссылочной целостности выполняется.

Если вышеприведённое условие не выполняется, говорят, что в базе данных нарушена ссылочная целостность. Такая БД не может нормально эксплуатироваться, так как в ней разорваны логические связи между зависимыми друг от друга фактами. Непосредственным результатом нарушения ссылочной целостности становится то, что корректным запросом не всегда удаётся получить корректный результат.

Приведем пример нарушения ссылочной целостности в базе данных (рис. 3.11).

Таблица **«Сотрудник»**

Табельный_ номер	Фамилия	Имя	Отчество	Год_ рождения	Номер_ отдела
10011	Иванов	Иван	Иванович	1975	1
10022	Петров	Петр	Петрович	1977	2
10033	Сидоров	Сидор	Сидорович	1980	3
10044	Николаев	Николай	Николаевич	1978	4
10055	Михайлов	Михаил	Михайлович	1973	5
10066	Сергеев	Сергей	Сергеевич	1983	Null

?

Таблица «Отдел»

Номер_отдела	Название_отдела
1	Цех
3	Бухгалтерия
4	Плановый отдел
5	Охрана

Рисунок 3.11 – Пример БД с нарушением ссылочной целостности

БД состоит из таблиц «Сотрудник» и «Отдел», которые обеспечивают хранение личной информации о сотруднике, и информации о том, кто из сотрудников в каком отделе работает. Очевидно, что полноценная информация о сотруднике должна быть представлена двумя связанными записями в обеих названных таблицах, что технически выражается в таком условии: для любой записи таблицы «Сотрудник» в таблице «Отдел» должна существовать соответствующая запись с Сотрудник.Номер_отдела=Отдел.Номер_отдела.

Чтобы получить информацию о всех сотрудниках с названием

отдела, в котором он работает, из таблиц, в которых соблюдается ссылочная целостность, достаточно применить к данным таблицам SQL-запрос:

```
Select *
From Сотрудник,Отдел
Where Сотрудник.Номер_отдела=Отдел.Номер_отдела;
```

В данном примере, однако, ссылочная целостность нарушена. Запись таблицы «Сотрудник» (Табельный_номер = 10022) имеет в поле «Номер_отдела» так называемую «висящую» ссылку – значение, которое не соответствует записи в таблице «Отдел». Из-за этого результат вышеприведённого запроса не будет содержать записи о сотруднике с табельным номером 10022.

Заметим, что в записи, которая соответствует сотруднику с табельным номером 10066, используется вариант намеренного (в некоторых случаях легального) нарушения ссылочной целостности – в поле внешнего ключа записан NULL. Чтобы получить информацию о сотрудниках, даже тех, у которых не указан отдел, в котором он работает, необходимо использовать внешнее соединение, которое в синтаксисе Oracle записывается так:

```
Select *
From Сотрудник,Отдел
Where Сотрудник.Номер_отдела=Отдел.Номер_отдела (+);
```

Причины нарушений ссылочной целостности БД

Правильно спроектированная и поддерживаемая база данных не допускает возможности нарушения ссылочной целостности. Тем не менее такие нарушения могут появиться в ходе эксплуатации базы по целому ряду причин. Некоторые из них:

Некорректная работа прикладного программного обеспечения. Понятно, что при ошибке в программе, выполняющей модификацию базы данных, база может быть модифицирована недопустимым образом, в результате чего образуются «висящие» ссылки. Программа может совершать ошибки следующих видов:

Неполная запись объектов. Данные объекта размещаются в записях нескольких таблиц, а программа не записывает какую-то из них.

Некорректная правка ссылки. Значение внешнего ключа изменяется на такое, которому не соответствует ни одна запись в связанной таблице.

Правка первичного ключа без каскадного обновления. В таблице, на которую есть ссылки, правится первичный ключ, но при этом внешние

ключи в связанных с ней таблицах остаются без изменения.

Удаление записи без каскадного обновления. Из таблицы удаляется запись, на которую имеются ссылки по внешним ключам других таблиц, при этом в связанных записях внешние ключи не меняются. В результате все ссылающиеся на неё записи других таблиц становятся некорректными.

Сбои в работе системного программного обеспечения и оборудования. Даже когда прикладное программное обеспечение работает совершенно правильно, возможно нарушение ссылочной целостности. Например, если при добавлении объекта в базу нужно добавить несколько связанных записей в несколько таблиц, очевидно, что ссылочная целостность будет нарушена в процессе добавления данных (когда часть связанных записей уже добавлена, а часть — ещё нет) и восстановится только после завершения операции. Если во время выполнения операции она будет прервана (из-за переполнения диска, сбоя питания, или по каким-то другим причинам), часть записей будет добавлена в БД, часть — нет. Часть добавленных записей останется с некорректными ссылками.

Обеспечение ссылочной целостности

Пустые внешние ключи

Возможна ситуация, когда внешний ключ вместо ссылки на существующую запись в таблице БД содержит «отсутствующее значение» NULL. Такое положение можно трактовать, как отсутствие какой-то части объекта. Хотя с точки зрения чистой теории это недопустимо, на практике иногда бывает удобно разрешить использование пустых внешних ключей. Чтобы корректно работать с группами связанных таблиц, допускающих пустые внешние ключи, используется специфическая операция языка SQL – открытое соединение (другое название — «внешнее соединение», англ. outer join).

Транзакции

Обязательным (хотя и не достаточным) условием сохранения ссылочной целостности базы данных является поддержка транзакций. Если программное обеспечение выполняет группу связанных между собой операций, которые по отдельности могут приводить к нарушению целостности ссылок, СУБД должна предоставлять возможность выполнения всей этой группы в одной транзакции, то есть так, чтобы при любом сбое производилась автоматическая отмена всех операций группы, в том числе уже полностью завершённых.

Ссылочная целостность на триггерах

Возможно поддержание ссылочной целостности БД с использованием механизма триггеров. В этом случае для любой потенциально опасной операции над таблицей создаётся триггер, который производит необходимые проверки или даже изменяет данные в связанных таблицах, чтобы исключить потерю ссылок.

Так, для обеспечения каскадных изменений триггер может быть установлен на операцию изменения записи в таблице. Если окажется, что при редактировании изменилось значение ключевого поля, триггер должен произвести согласованные изменения во всех таблицах, связанных с данной, поменяв старое значение внешних ключей на новое.

Для исключения потери ссылок от некорректного редактирования внешнего ключа триггер должен при каждом изменении соответствующего поля проверять, имеется ли в связанной таблице запись с таким первичным ключом.

Для защиты от удаления записи, на которую имеются ссылки, триггер на связанной таблице должен при удалении проверять наличие ссылок и в зависимости от необходимости либо запрещать удаление, либо обнулять внешние ключи тем или иным образом.

Ссылочная целостность на внешних ключах

СУБД может иметь механизм автоматического поддержания ссылочной целостности, основанный на явном описании ссылок при создании БД. При описании таблиц БД программист явно описывает, какие поля таблиц являются внешними ключами и на какие таблицы они ссылаются. Эта информация сохраняется в служебных областях памяти БД. Любая операция, изменяющая данные в таблице, вызывает автоматическую проверку ссылочной целостности.

При операции добавления или редактирования записи автоматически проверяется, ссылаются ли внешние ключи в этой записи на существующие записи в заявленных при описании связанных таблицах. Если выясняется, что операция приведёт к появлению некорректных ссылок, она не выполняется — система возвращает ошибку.

При операции редактирования записи проверяется, не изменяется ли её первичный ключ и нет ли на неё ссылок. Если первичный ключ изменяется и при этом на данную запись имеются ссылки, то операция редактирования завершается с ошибкой.

При операции удаления записи проверяется, нет ли на неё ссылок. Если ссылки имеются, то возможно три варианта дальнейших действий (фактически выполняемый вариант зависит от СУБД и от выбора программиста, который он должен сделать при описании связи):

- *запрет* – удаление блокируется и возвращается ошибка.
- *каскадное удаление* – в одной транзакции производится удаление

данной записи и всех записей, ссылающихся на данную. Если на удаляемые записи также есть ссылки и настройки также требуют удаления, то каскадное удаление продолжается дальше. Таким образом, после удаления данной записи в базе не остаётся ни одной записи, прямо или косвенно ссылающейся на неё. Если хотя бы одну из ссылающихся записей удалить не получается (либо для неё настроен запрет, либо происходит какая-либо ещё ошибка), то все удаления запрещаются.

- *обнуление внешних ключей* – во все внешние ключи записей, ссылающихся на данную, записывается псевдозначение NULL (SQL). Если хотя бы для одной из ссылающихся записей это невозможно (например, если поле внешнего ключа описано так, что его нельзя обнулять), то удаление запрещается.

3.5.2. Обеспечение производительности БД

Кроме того, требуется достичь и главной цели физического проектирования реляционной базы данных: дать гарантию того, что база данных обеспечивает требуемый уровень производительности. Обычно производительность базы данных измеряется в терминах производительности транзакций (transaction performance).

Решающим фактором для проектировщика базы данных в борьбе за производительность является решение задачи выбора между физическими конструкциями СУБД, которые могут быть использованы для повышения производительности транзакций.

Чтобы успешно решить данную задачу, проектировщик базы данных должен иметь хорошо определенные транзакции, которые могут существовать в базе данных. Однако в большинстве практических случаев получить изначально полностью прописанные транзакции к базе данных является весьма сложной организационной задачей. В условиях недостаточных сведений о транзакциях проектировщику приходится зачастую полагаться на свой собственный опыт проектирования, выполнять выборочную настройку полученной первой итерации физической модели базы данных и переадресовывать окончательное решение данной задачи администратору базы данных. К тому же часть проблем, связанная с производительностью базы данных, может быть выявлена лишь на стадии тестирования и опытной эксплуатации базы данных.

Отметим, что при решении задач этого этапа нельзя опираться только на знание стандарта SQL. В действие вступают конструкции конкретной СУБД, выбранной для реализации базы данных. Основными механизмами промышленных СУБД для решения настоящей задачи повышения производительности являются денормализация, индексы, кластеризация и разделение.

3.6. Подробный план проектирования базы данных

На основе изложенного материала сформулируем подробный план проектирования базы данных на основе концептуального проектирования.

Предварительным этапом проектирования БД является анализ требований к ПО.

Этап 1. Анализ требований к БД

На основе анализа диаграммы вариантов использования и требований к ПО необходимо:

- выделить те функции разрабатываемого ПО, для реализации которых будет необходима БД;
- проанализировать данные, которые будут храниться в БД;
- проанализировать выходные данные (документы), которые будут выдаваться (генерироваться) на основе данных БД.

Этап 2. Концептуальное (инфологическое) проектирование базы данных

На основе проведенного анализа требований к ПО необходимо:

- в выбранной нотации (IDEF1x, нотация Бахмана, Мартина, UML или др.) построить ER-модель предметной области (желательно использование CASE-средства, поддерживающего разработку ER-модели с возможностью последующей генерации SQL-скрипта);
- проверить ER-модель на избыточность;
- проверить ER-модель на возможность реализации функций ПО, требующих использования БД;
- сформулировать основные запросы к БД (например, «Вывести список всех сотрудников для заданного отдела» и т.д.), действия (например, «Выполнить расчет премий сотрудников: 15% от продаж в расчетном месяце»), триггеры (например, «Недопустимо принимать на работу сотрудников старше 60 лет»).

Для построения ER-модели необходимо выполнить следующие действия:

- выделить сущности предметной области;
- определить атрибуты каждой сущности и их домены (типы);
- для каждой сущности определить атрибуты, которые могут являться первичным ключом данной сущности;
- установить связи между сущностями и задать их имена;
- установить тип связей между сущностями: один-к-одному (1:1), один-ко-многим (1:N), многие-ко-многим (N:M);
- установить обязательность связей между сущностями.

Этап 3. Логическое проектирование БД

На основе построенной концептуальной модели предметной области в виде ER-модели необходимо построить схему реляционной базы данных.

Для этого необходимо:

- с помощью правил преобразования ER-модели в реляционную схему базы данных определить набор отношений (таблиц) БД;
- реализовать связи между сущностями путем миграции первичных ключей в дочерние сущности;
- определить необходимость миграции атрибутов в состав ключевых или неключевых атрибутов дочерней сущности;
- проверить полученные отношения с помощью правил нормализации (обосновать число нормальных форм, которым соответствует разработанная БД).

Этап 4. Физическое проектирование БД

На основе простроенной логической модели БД необходимо разработать ее физическую модель.

Для этого необходимо:

- обоснованно выбрать СУБД;
- реализовать отношения (таблицы), т.е. получить SQL-скрипт для выбранной СУБД;
- указать обязательность (NOT NULL) значений атрибутов таблиц;
- при необходимости указать значения по умолчанию (DEFAULT);
- реализовать ограничения (constraints) предметной области на значения атрибутов таблиц (BETWEEN, CHEC);
- указать первичные ключи таблиц (PRIMARY KEY);
- реализовать ограничения внешней ссылочности, т.е. указать внешние ключи (FOREING KEY);
- определить индексы;
- определить требования к дисковой памяти для обеспечения функционирования БД;
- реализовать запросы к БД, которые были выделены на этапе концептуального проектирования.
- в случае необходимости разработать триггеры базы данных;
- в случае необходимости разработать хранимые процедуры.

4. ПРИМЕР ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ В СРЕДЕ ERWIN

В данном разделе рассматривается пример поэтапного построения базы данных для системы автоматизации расчетов за услугу «Холодное водоснабжение» для общества совладельцев многоквартирного дома.

4.1. Анализ требований к БД

На рис. 4.1 представлена диаграмма вариантов использования для системы автоматизации расчетов за услугу «Холодное водоснабжение» для общества совладельцев многоквартирного дома.

На предварительном этапе проектирования БД требуется проанализировать варианты использования разрабатываемого ПО, в целях выявления таких вариантов использования, для реализации которых необходимо будет взаимодействие с БД.

В результате проведенного анализа диаграммы вариантов использования были выделены такие варианты использования, которые требуют взаимодействия с БД, а именно:

- авторизация;
- регистрация индивидуального счетчика холодной воды;
- изменение данных об индивидуальном счетчике холодной воды;
- внесение показаний индивидуального счетчика холодной воды за месяц;
- регистрация общего счетчика холодной воды;
- изменение данных об общем счетчике холодной воды;
- внесение показаний общего счетчика холодной воды за месяц;
- генерация квитанций за месяц;
- начисления за месяц;
- печать квитанций;
- внесение оплаты;
- получение списка должников;
- расчет суммы долга;
- добавление адреса;
- удаление адреса;
- изменение данных о квартиросъемщике;
- добавление улицы;
- редактирование названия улицы;
- удаление улицы;
- добавление дома;
- редактирование номера дома;
- удаление дома.

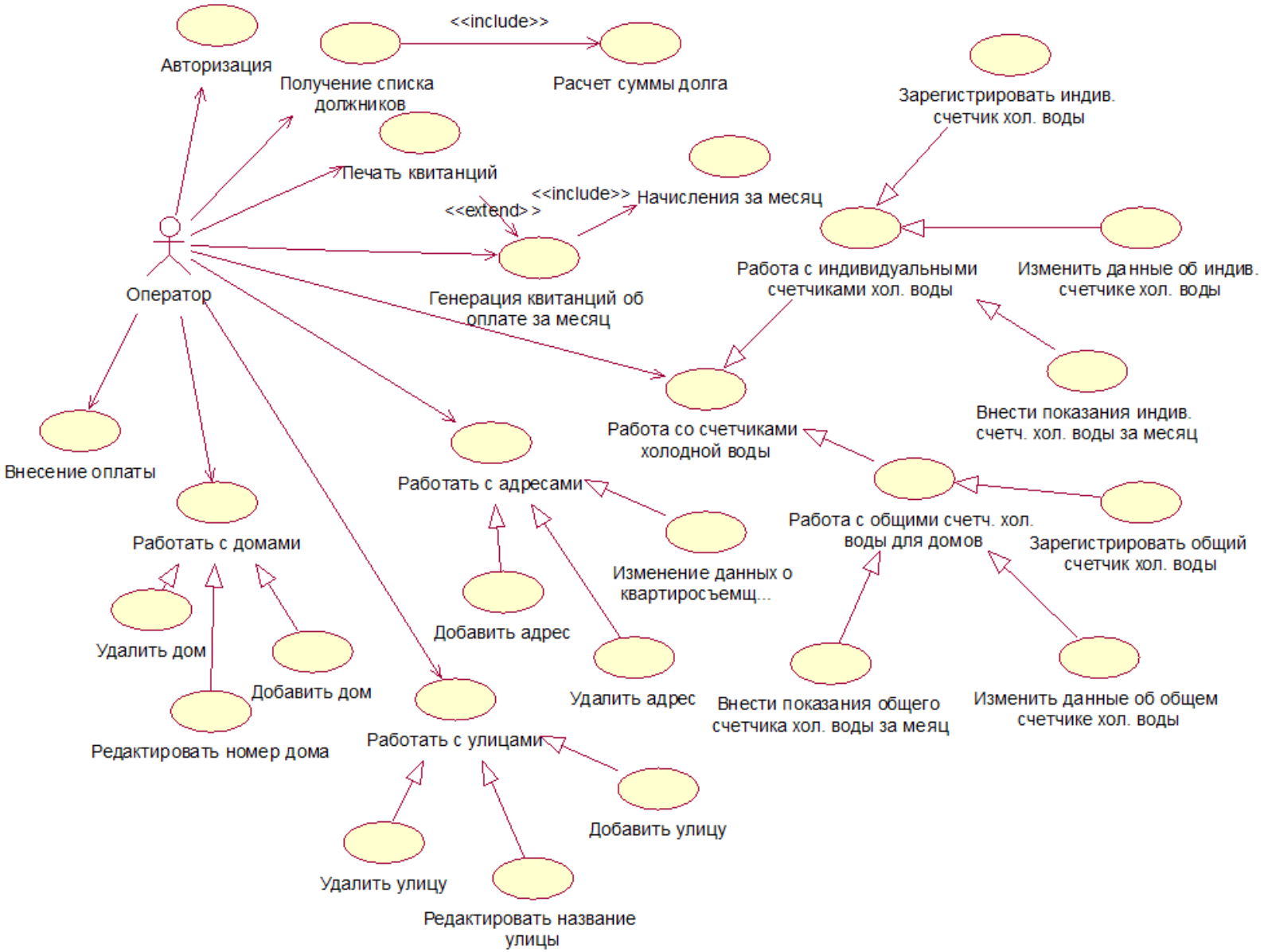


Рисунок 4.1 – Диаграмма вариантов использования системы «Холодное водоснабжение»

В результате анализа функций разрабатываемого ПО были выделены данные, которые будут храниться в БД: данные об улицах, домах, адресах, квартиросъемщиках, установленных индивидуальных и общих счетчиках холодной воды, показаниях индивидуальных и общих счетчиков холодной воды за месяц.

На основе данных, которые будут храниться в БД в системе расчетов за услугу «Холодное водоснабжение», будут происходить начисления за месяц за потребленную услугу, а также генерироваться квитанции для оплаты с последующей их печатью.

4.2. Концептуальное (инфологическое) проектирование базы данных

На этапе концептуального (инфологического) проектирования БД необходимо на основе проведенного анализа требований к ПО построить модель «сущность-связь» (ER-модель) предметной области.

Для графического представления ER-модели необходимо определиться с нотацией. Для представления ER-модели выполнения расчетов за услугу «Холодное водоснабжение» была выбрана нотация IDEF1x, которая поддерживается CASE-средством ERWin Data Modeler 7.0 [14,15].

Для построения ER-модели необходимо выделить сущности предметной области. В результате анализа предметной области были выделены сущности, представленные на рис. 4.2.

Рисунок 4.2 – Сущности предметной области

После выделения сущностей предметной области необходимо определить атрибуты сущности. На рис. 4.3 приведены выделенные сущности с определенными для них атрибутами. Для обозначения доменов атрибутов сущности используется графическая форма их отображения.

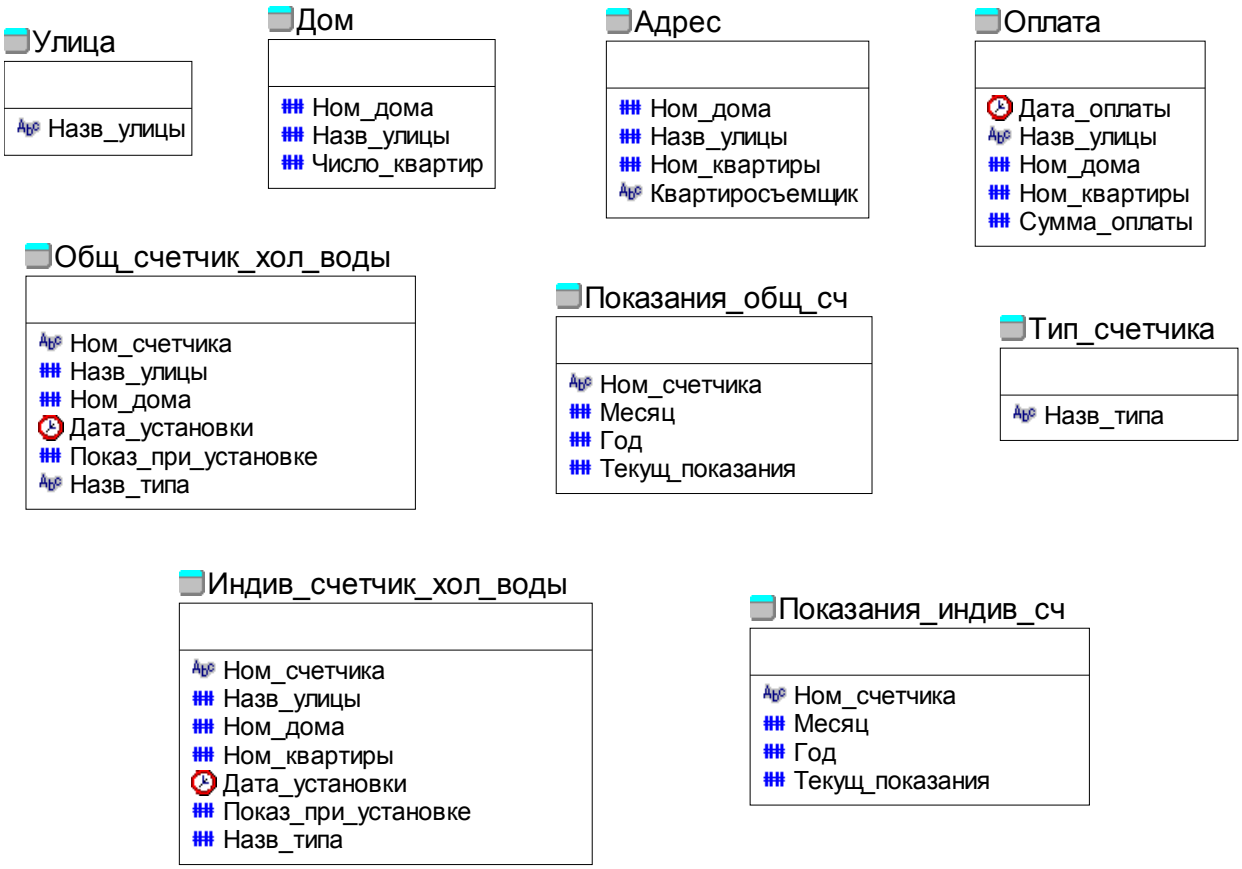


Рисунок 4.3 – Атрибуты сущностей предметной области

Определим первичные ключи для выделенных сущностей. На рис. 4.4 приведены выделенные сущности с атрибутами, которые определены в качестве первичного ключа.

Следующим этапом построения ER-модели является определение связей между сущностями. На рис. 4.5 приведены связи между выделенными сущностями.

Сущность, которая находится на конце связи с жирной точкой, называется дочерней сущностью, другая сущность связи называется родительской. Например, для связи «на *Улице* расположены *Дома*», сущность «Улица» является родительской, тогда как сущность «Дом» – дочерней.

В нотации IDEF1x выделяют два типа дочерних сущностей: зависимые и независимые.

Для указания того, что дочерняя сущность будет зависимой от родительской, в нотации IDEF1x используется идентифицирующая связь (обозначается сплошной линией). Для указания того, что дочерняя сущность будет независимой от родительской используется неидентифицирующая связь (обозначается пунктирной линией).

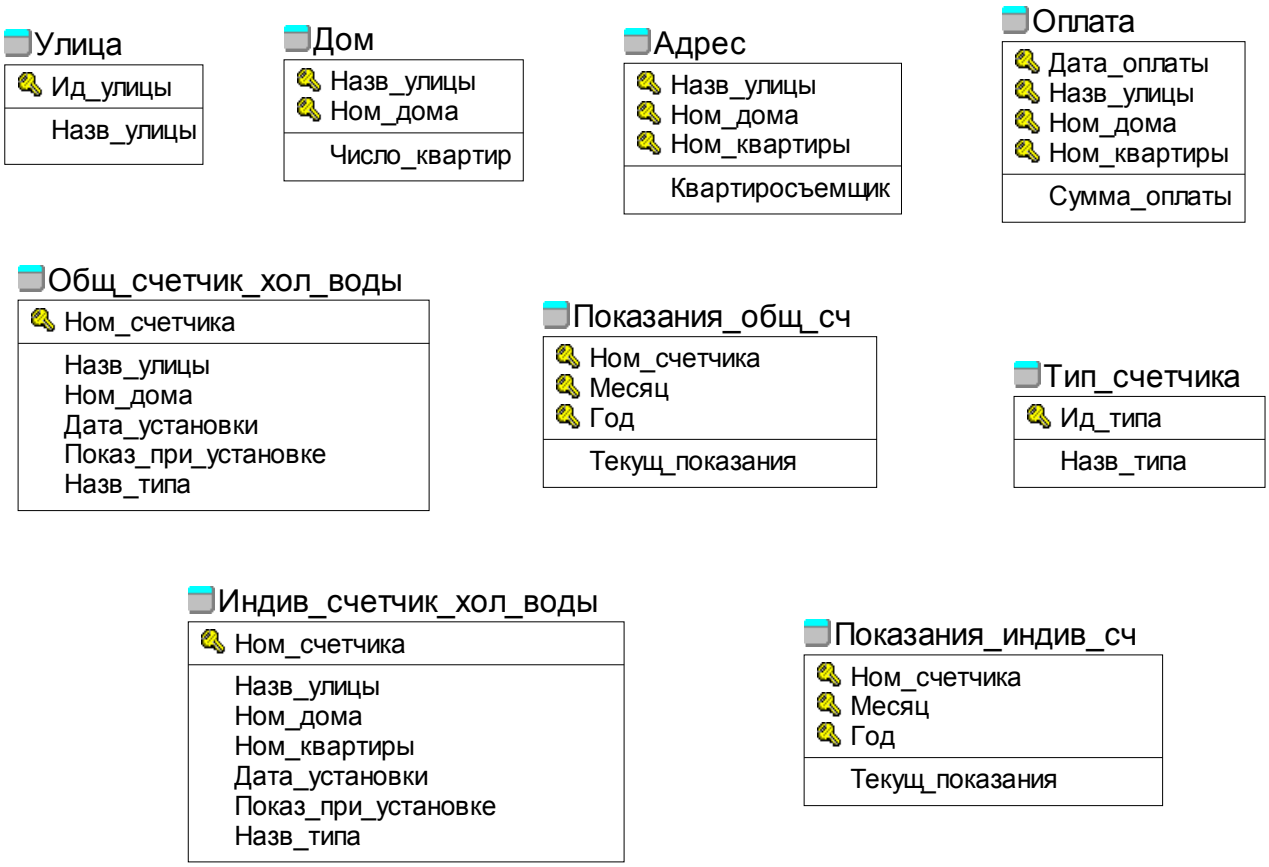


Рисунок 4.4 – Первичные ключи сущностей предметной области

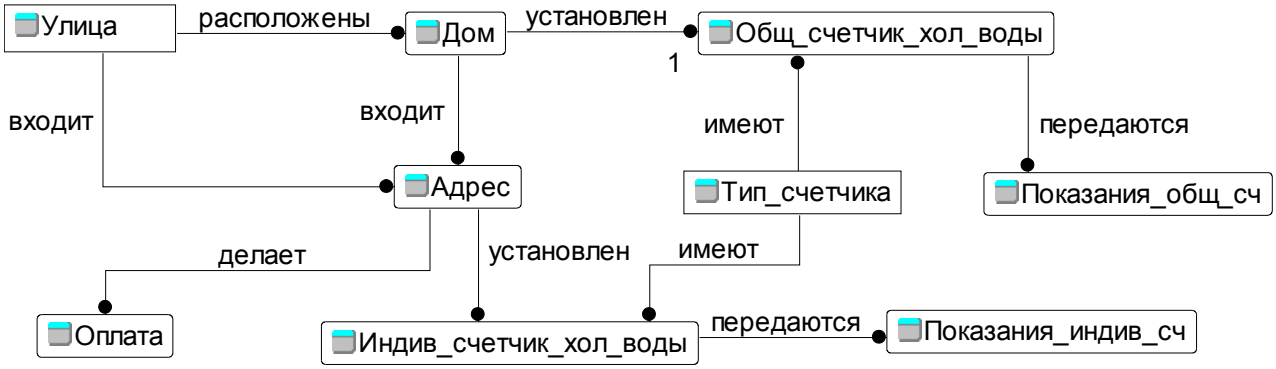


Рисунок 4.5 – Связи между сущностями предметной области

Проанализируем построенную ER-модель на предмет зависимых и независимых дочерних сущностей, т.е. изменим там, где это необходимо, идентифицирующую связи на неидентифицирующую.

В результате проведенного анализа дочерних сущностей была получена измененная ER-модель, представленная на рис. 4.6.

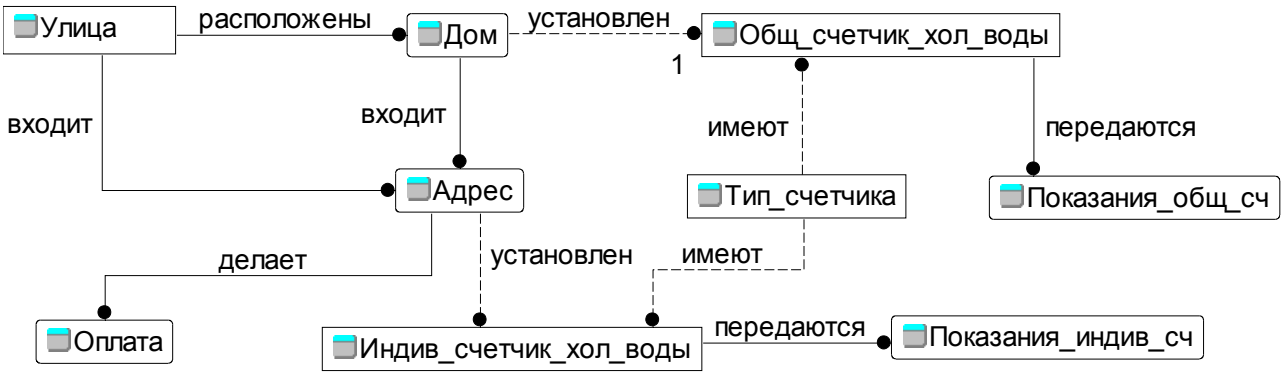


Рисунок 4.6 – ER-модель предметной области в нотации IDEF1x

Все выделенные связи между сущностями являются обязательными.

Таким образом, на рис. 4.6 приведена ER-модель выполнения расчетов за услугу «Холодное водоснабжение», построенная с помощью нотации IDEF1x и CASE-средства ERWin Data Modeler 7.0.

4.3. Логическое проектирование БД

На этапе логического (даталогического) проектирования БД необходимо на основе построенной ER-модели предметной области получить схему реляционной базы данных.

Для построения реляционной схемы БД необходимо прежде всего определить набор отношений (таблицы) БД.

Преобразование ER-модели в схему реляционной БД CASE-средство ERWin реализует автоматически по следующим правилам:

- каждой сущности ER-модели ставится в соответствие таблица БД;
- связи один-к-одному, один-ко-многим реализуется путем миграции первичного ключа родительской сущности в дочернюю;
- связь многие-ко-многим реализуется путем создания дополнительной таблицы, в которую мигрируют первичные ключи обоих сущностей, находящихся в связи многие-ко-многим.

На рис. 4.7 представлены таблицы БД для выполнения расчетов за услугу «Холодное водоснабжение», полученные автоматически с помощью CASE-средства ERWin Data Modeler 7.0.

Видим, что полученная в автоматическом режиме схема реляционной БД имеет недостатки.

Например, рассмотрим таблицу «Дом», которая имеет атрибуты Ид_улицы и Назв_улицы. Атрибут Назв_улицы был выделен на этапе построения ER-модели, тогда как атрибут Ид_улицы является атрибутом, который мигрировал в таблицу «Дом» в результате реализации связи «на

Улице расположены *Дома*». Так как по значению атрибута Ид_улицы можно однозначно определить название улицы, для этого необходимо обратиться к таблице «Улица», поэтому удалим из таблицы «Дом» атрибут Назв_улицы.

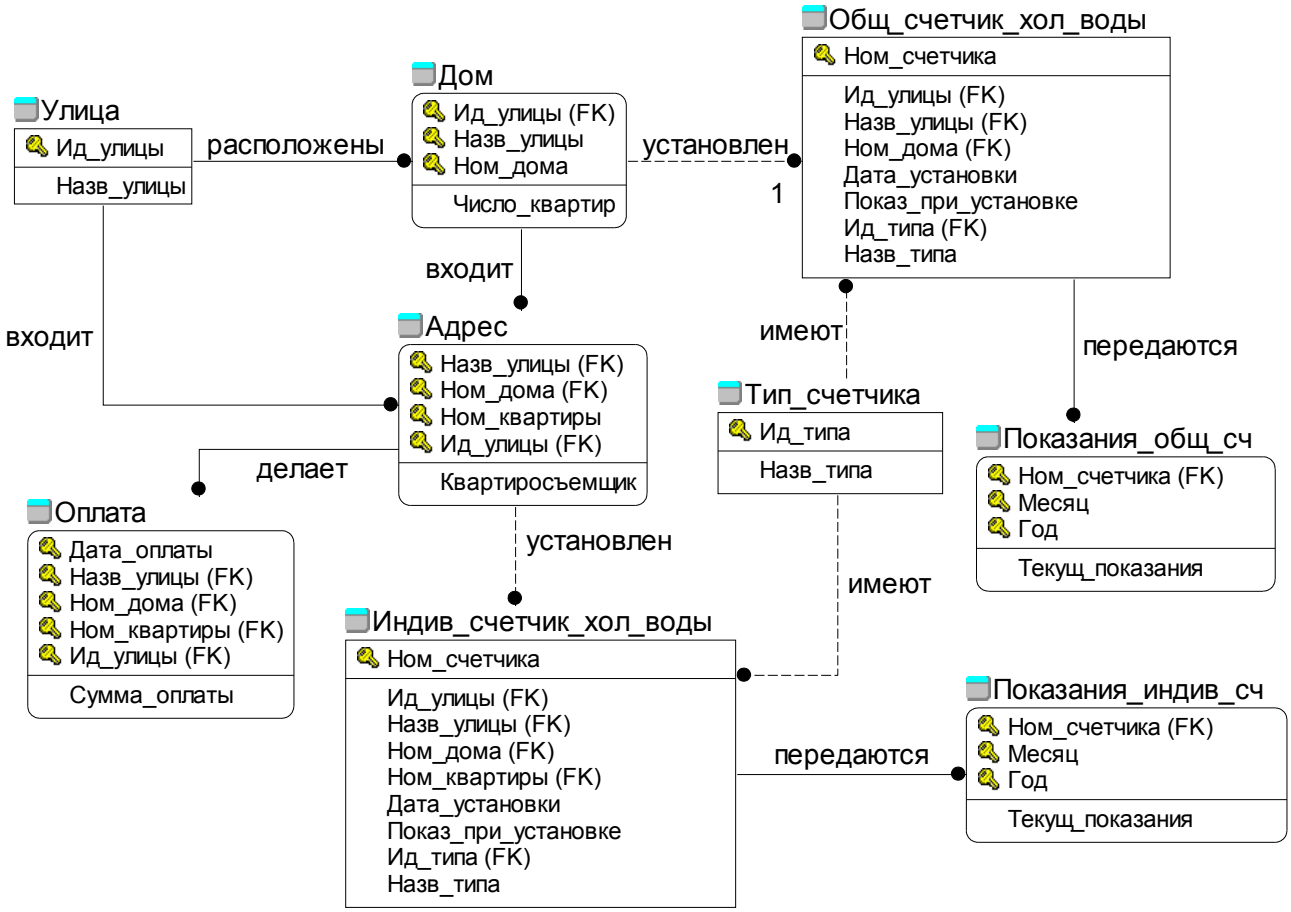


Рисунок 4.7 – Схема базы данных

Заметим, что атрибут Назв_улицы таблицы Дом мигрирует в дочерние таблицы данной сущности, поэтому его удаление из таблицы Дом приведет к изменениям в структуре других таблиц.

Аналогично требует своего удаления и атрибут Назв_типа в таблице Общ_счетчик_хол_воды, а также атрибут Назв_типа в таблице Индив_счетчик_хол_воды.

В результате проведенного анализа всех таблиц была получена схема БД, представленная на рис. 4.8.

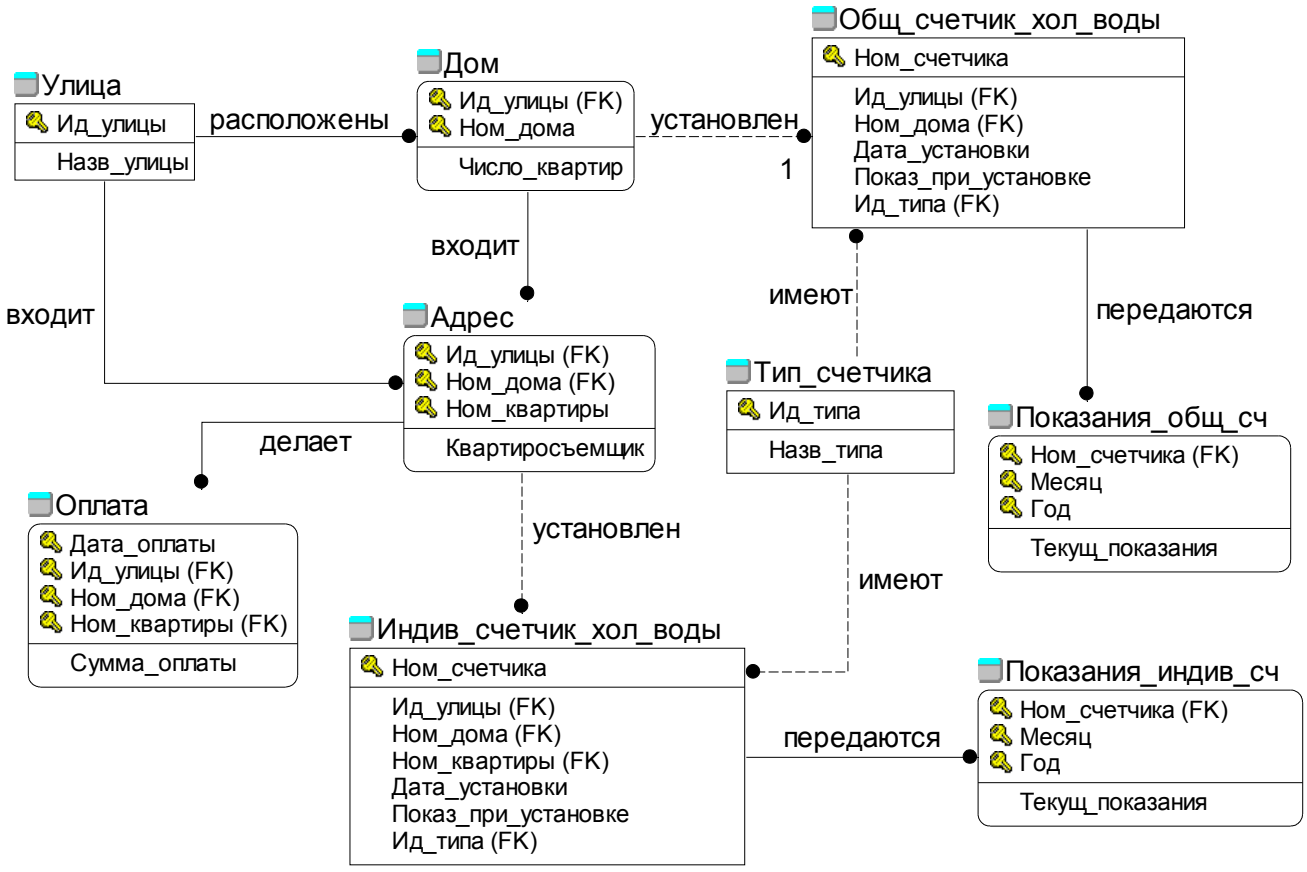


Рисунок 4.8 – Схема БД после удаления дубликатов

Проиллюстрируем синтезированную схему БД с указанием доменов атрибутов таблиц (рис. 4.9).

Синтезированные отношения требует своей проверки с помощью правил нормализации. Нормализация схемы базы данных способствует более эффективному выполнению системой управления базами данных операций обновления базы данных, поскольку сокращается число проверок и вспомогательных действий, поддерживающих целостность базы данных. При проектировании реляционной базы данных почти всегда добиваются второй нормальной формы всех входящих в базу данных отношений. В часто обновляемых базах данных обычно стараются обеспечить третью нормальную форму отношений.

Проверим построенные отношения на соответствие третьей нормальной форме.

Для этого запишем отношения в таком виде:

Улица (Ид_улицы, Назв_улицы)

Дом (Ид_улицы, Ном_дома, Число_квартир)

Адрес (Ид_улицы, Ном_дома, Ном_квартиры, Квартиросъемщик)

Оплата (Дата_оплаты, Ид_улицы, Ном_дома, Ном_квартиры, Сумма_оплаты)

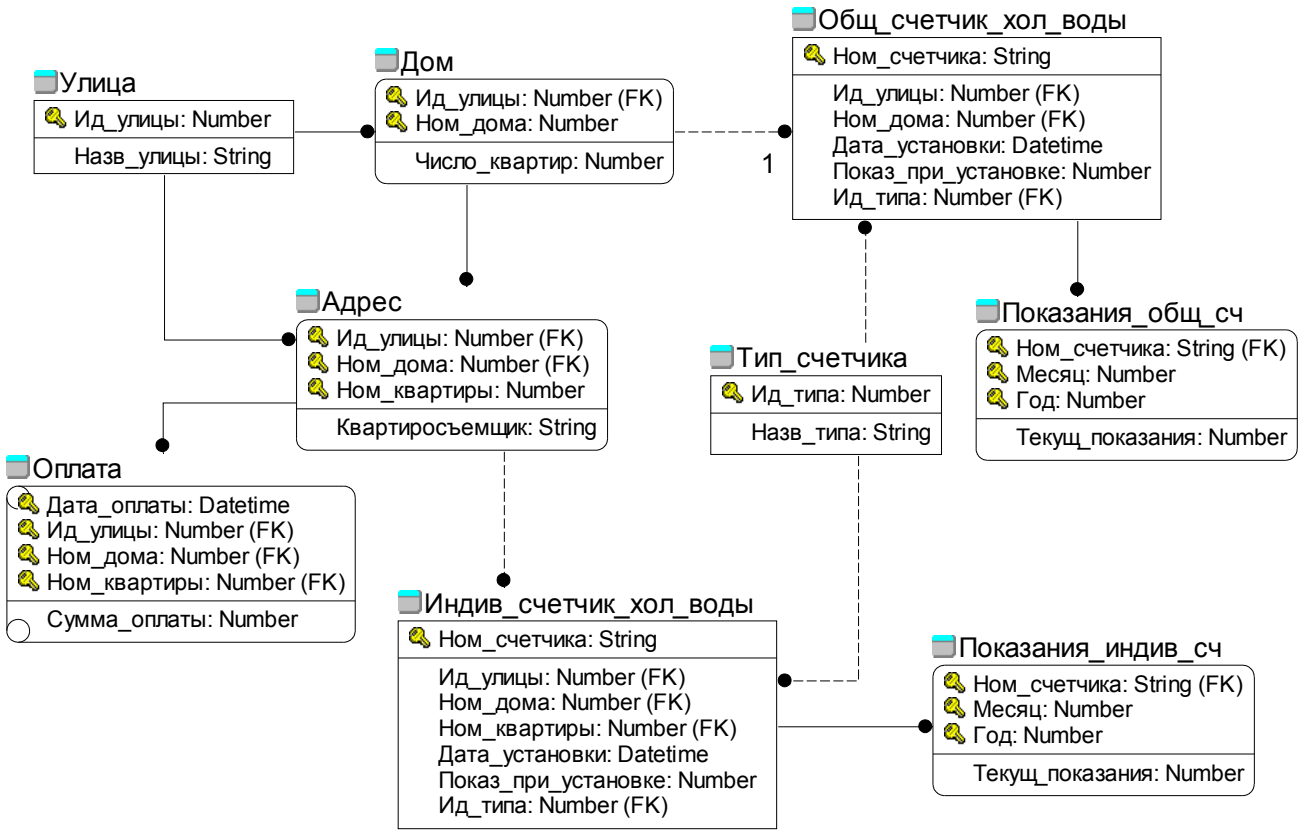


Рисунок 4.9 – Схема БД с указанием доменов атрибутов

Общ_счетчик_хол_воды (Ном_счетчика, Ид_улицы, Ном_дома, Дата_установки, Показ_при установке, Ид_типа)

Индив_счетчик_хол_воды (Ном_счетчика, Ид_улицы, Ном_дома, Ном_квартиры, Дата_установки, Показ_при_установке, Ид_типа)

Тип_счетчика (Ид_типа, Назв_типа)

Показ_общ_сч (Ном_счетчика, Месяц, Год, Текущ_показания)

Показ_индив_сч (Ном_счетчика, Месяц, Год, Текущ_показания)

Приведенные отношения находятся в первой нормальной форме (1NF), так как все атрибуты отношений принимают простые значения (атомарные или неделимые), не являющиеся множеством или кортежем из более элементарных составляющих.

Приведенные отношения находятся во второй нормальной форме (2NF), так как они находятся в первой нормальной форме, и каждый неключевой атрибут конкретного отношения минимально функционально зависит от своего первичного ключа.

Приведенные отношения находятся в третьей нормальной форме (3NF), так как они находятся во второй нормальной форме, и каждый неключевой атрибут конкретного отношения нетранзитивно функционально зависит от своего первичного ключа.

Опишем все таблицы синтезированной схемы реляционной БД для выполнения расчетов за услугу «Холодное водоснабжение» для общества совладельцев многоквартирного дома. Все атрибуты выделенных таблиц схемы БД являются обязательными.

Таблица **«Улица»**

Атрибуты	*Тип*	*Описание*
Ид_улицы	Целое число	Первичный ключ (2 цифры)
Назв_улицы	Строка	Максим. длина строки = 50 символов

Таблица **«Дом»**

Атрибуты	*Тип*	*Описание*	
Ид_улицы	Целое число	Первичный ключ	2 цифры
Ном_дома	Целое число		3 цифры
Число_квартир	Целое число	3 цифры	

Таблица **«Общ_счетчик_хол_воды»**

Атрибуты	*Тип*	*Описание*
Ном_счетчика	Строка	Первичный ключ (формат строки XXXXXX)
Ид_улицы	Целое число	Внешний ключ (2 цифры)
Ном_дома	Целое число	Внешний ключ (3 цифры)
Дата_установки	Дата	Формат даты ДД:ММ:ГГ
Показ_при_установке	Вещ_число	Формат числа XXXXXX.X (число >0)
Ид_типа	Целое число	2 цифры

Аналогично описываются таблицы «Показ_общ_сч», «Адрес», «Индив_счетчик_хол_воды», «Показ_индив_сч», «Тип_счетчика», «Оплата».

Опишем действия с таблицами БД (в терминах ДОБАВИТЬ, УДАЛИТЬ, ИЗМЕНИТЬ), которые должны быть реализованы в соответствии с функциями разрабатываемого ПО:

- добавить улицу;
- удалить улицу;
- изменить название улицы;
- добавить дом;
- удалить дом;
- изменить данные о доме;
- добавить адрес;

- изменить данные о квартиросъемщике;
- добавить общий счетчик;
- изменить данные об общем счетчике;
- удалить общий счетчик;
- добавить индивидуальный счетчик;
- изменить данные об индивидуальном счетчике;
- удалить индивидуальный счетчик;
- добавить показания общего счетчика;
- добавить показания индивидуального счетчика;
- добавить тип счетчиков;
- удалить тип счетчиков;
- изменить название типа счетчика.

Опишем **запросы** к БД (в терминах ВЫВЕСТИ …), которые необходимо будет реализовать в соответствии с функциями разрабатываемого ПО.

Запрос. Вывести список адресов с фамилиями квартиросъемщиков и суммами оплат, произведенных в указанный период.

Опишем **действия**, которые необходимо реализовать на основе БД в соответствии с функциями разрабатываемого ПО.

Действие. Выполнить начисления для каждого квартиросъемщика за текущий месяц (по переданным текущим показаниям индивидуальных счетчиков, а в случае отсутствия таковых – по текущим показаниям общих счетчиков, разделенных пропорционально количеству жильцов, проживающих в квартире).

Действие. Получить список должников для заданного адреса с указанием текущей суммы долга.

Действие. Получить отсортированный по убыванию список должников для заданного адреса с указанием текущей суммой долга.

Действие. Вычислить общую сумму начисления за указанный период по конкретному дому.

Действие. Вычислить общую сумму долга за указанный период по конкретному дому.

Опишем **триггеры**, которые необходимо будет реализовать:

Триггер. Если квартиросъемщик не внес оплату за услугу «Холодное водоснабжение» в течение 20 дней со дня получения квитанции, то ему начисляется пеня в размере 0,5% от потребленной услуги за каждый просроченный день.

4.4. Физическое проектирование БД

На этапе физического проектирования БД необходимо на основе простроенной логической модели БД разработать ее физическую модель.

Для построения физической модели прежде всего необходимо выбрать СУБД. Для разработки системы автоматизации расчетов за услугу «Холодное водоснабжение» заказчиков была выбрана СУБД Access 2000.

CASE-средство ERWin Data Modeler 7.0 позволяет задать для каждого атрибута таблицы соответствующий домену конкретный тип, поддерживаемый СУБД Access 2000.

В результате задания конкретных типов атрибутов таблиц была получена следующая схема БД, представленная на рис. 4.10.

CASE-средство ERWin Data Modeler 7.0 позволяет также задать правила-ограничения на значения атрибутов таблиц. Например, можно задать правило, которое будет указывать, что значение атрибута Показ_при_утановке таблицы Общ_счетчик_хол_воды должно быть положительным.

Такие правила-ограничения на значения атрибутов таблиц были созданы в соответствии с описанием таблиц, приведенным выше.

Кроме того, CASE-средство ERWin Data Modeler 7.0 позволяет в автоматическом режиме сгенерировать SQL-скрипт. Фрагмент сгенерированного скрипта:

```
CREATE TABLE Адрес (
    Ид_улицы            NUMBER(2) NOT NULL,
    Ном_дома            NUMBER(3) NOT NULL,
    Ном_квартиры        NUMBER(3) NOT NULL,
    Квартиросъемщик     VARCHAR(50) NOT NULL);


ALTER TABLE Адрес
    ADD  ( PRIMARY KEY (Ид_улицы, Ном_дома, Ном_квартиры) ) ;


CREATE TABLE Дом (
    Ид_улицы            NUMBER(2) NOT NULL,
    Ном_дома            NUMBER(3) NOT NULL,
    Число_квартир       NUMBER(3) NOT NULL);


ALTER TABLE Дом
    ADD  ( PRIMARY KEY (Ид_улицы, Ном_дома) ) ;
```

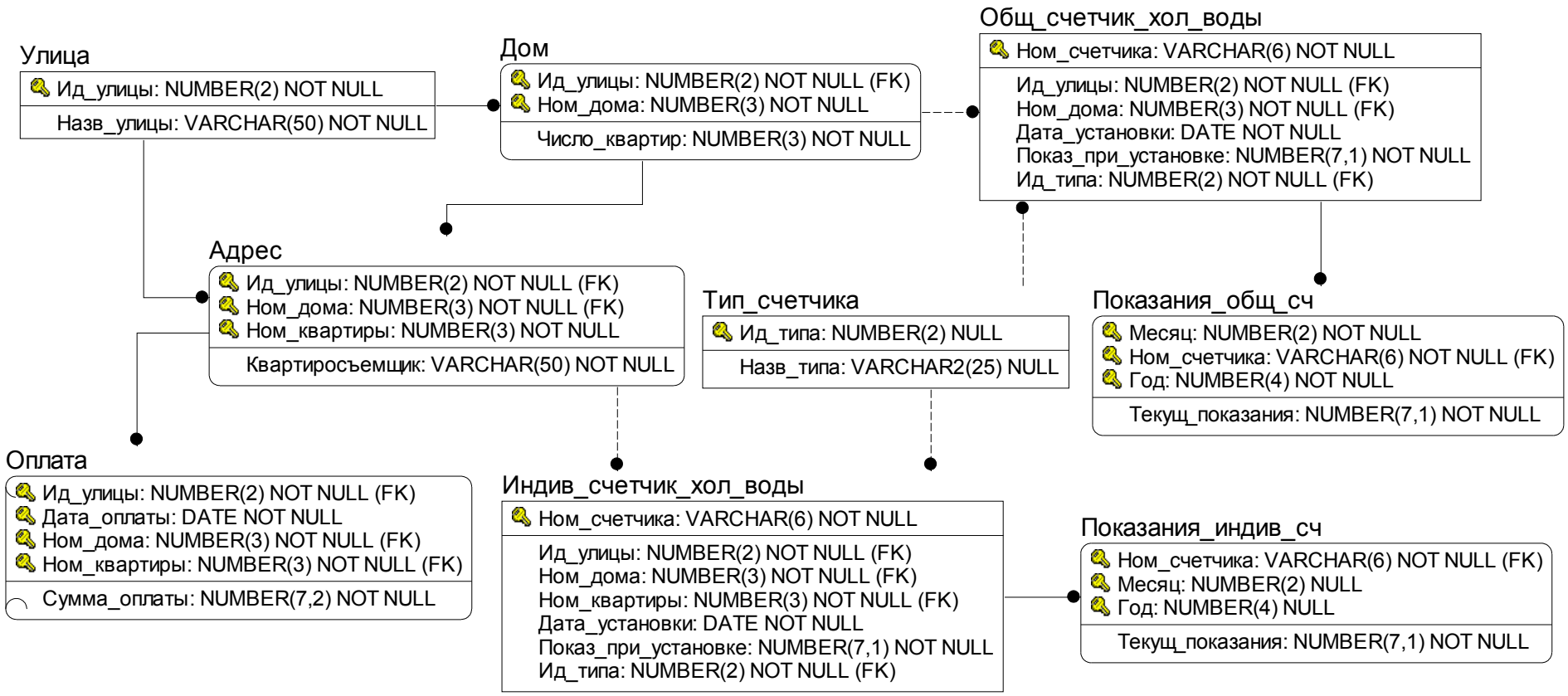


Рисунок 4.10 – Схема БД с конкретными типами атрибутов

```
CREATE TABLE Индив_счетчик_хол_воды (
    Ид_улицы            NUMBER(2) NOT NULL,
    Ном_счетчика        VARCHAR(6) NOT NULL,
    Ном_дома            NUMBER(3) NOT NULL,
    Ном_квартиры        NUMBER(3) NOT NULL,
    Дата_установки      DATE NOT NULL,
    Показ_при_установке NUMBER(7,1) NOT NULL
                        CHECK (Показ_при_установке >= 0),
    Ид_типа             NUMBER(2) NOT NULL);


ALTER TABLE Индив_счетчик_хол_воды
    ADD  ( PRIMARY KEY (Ном_счетчика) ) ;
```

Следующим этапом физического проектирования БД является разработка запросов, хранимых процедур и триггеров БД, покрывающих логику функционирования разрабатываемого ПО.

Общие формулировки запросов и действий с данными БД, которые необходимо чтобы выполняла система, были описаны на этапе логического проектирования БД. Представим SQL-скрипт запросов, а также хранимых процедур и функций.

Вывести список адресов с фамилиями квартиросъемщиков и суммами оплат, произведенных в указанный период:

Select … from ….. where …. ;

Выполнить начисления для каждого квартиросъмщика за текущий месяц:

```
CREATE OR REPLACE PROCEDURE начисления_за_месяц
    AS
    BEGIN
    …
    END начисления_за_месяц;
```

Получить список должников для заданного адреса с указанием текущей суммы долга:

```
CREATE OR REPLACE PROCEDURE список_должников
    AS
    BEGIN
    …
    END список_должников;
```

Получить отсортированный по убыванию список должников для заданного адреса с указанием текущей суммой долга.

```
CREATE OR REPLACE PROCEDURE список_должников_упоряд
    AS
    BEGIN
    …
    END список_должников_упоряд;
```

Вычислить общую сумму начисления за указанный период по конкретному дому.

```
CREATE OR REPLACE FUNCTION общая_сумма_начислений
(нач_периода, кон_периода, назв_улицы, номер_дома)
     RETURN  вещ_число
    AS ...
    BEGIN
        …
        RETURN сумма_начислений
    END;
```

Вычислить общую сумму долга за указанный период по конкретному дому.

```
CREATE OR REPLACE FUNCTION общая_сумма_долга (нач_периода,
кон_периода, назв_улицы, номер_дома)
     RETURN  вещ_число
    AS ...
    BEGIN
        …
        RETURN сумма_долга
    END;
```

Если квартиросъемщик не внес оплату за услугу «Холодное водоснабжение» в течение 20 дней со дня получения квитанции, то ему начисляется пеня в размере 0,5% от потребленной услуги за каждый просроченный день.

```
CREATE TRIGGER Пеня
    BEFORE
    INSERT ON Оплата
    FOR EACH ROW
    WHEN …
    DECLARE  …
    BEGIN   …
    END;
```

В результате выполненных шагов была спроектирована база данных для системы автоматизации расчетов за услугу «Холодное водоснабжение» для общества совладельцев многоквартирного дома.

5. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

5.1. Лабораторная работа 1 «Построение ER-модели предметной области»

Цель работы: научиться строить ER-модель (модель «сущность-связь») для заданной предметной области.

Задание:

1. Построить диаграмму вариантов использования для программного обеспечения (см. вариант в п. _____) и выполнить ее анализ для выявления тех функций ПО, которые требуют сохранения, извлечения и обработки данных.

   Замечание: перечень основных функций программного обеспечения, указанных в варианте, не является исчерпывающим и может быть дополнен студентом.
2. Выделить сущности предметной области и их атрибуты.
3. Установить связи между выделенными сущностями.
4. Проименовать выделенные связи и установить их кратность.
5. Отобразить выделенные сущности и связи с помощью выбранной нотации (например, нотация Чена, Мартина и т.д.).
6. Подготовить отчет.

5.2. Лабораторная работа 2
«Построение логической модели базы данных»

Цель работы: научиться преобразовывать ER-модель предметной области в логическую модель базы данных.

Задание:

1. Определить набор таблиц базы данных.
2. Реализовать связи между таблицами (кроме связей «многие-ко-многим»).
3. Проверить полученные таблицы на соответствие первой, второй, третьей нормальным формам и нормальной форме Бойса-Кодда.
4. В ***словесной*** *форме* сформулировать запросы к таблицам для покрытия функциональности ПО.
5. В ***словесной*** *форме* сформулировать назначение хранимых процедур и функций, а также триггеров для обеспечения функциональности разрабатываемого программного обеспечения

5.3. Лабораторная работа 3
«Построение физической модели данных»

Цель работы: научиться преобразовывать логическую модель базы данных в физическую.

Задание:

1. Преобразовать связи «многие-ко-многим» между таблицами (создание ассоциативных таблиц; при необходимости, дополнение ассоциативных таблиц дополнительными атрибутами).
2. Обоснованно выбрать систему управления базой данных (СУБД).
3. Назначить столбцам таблиц типы данных, поддерживаемые выбранной СУБД, установить ограничения на значения столбцов таблиц базы данных.
4. Получить SQL-скрипты для создания таблиц базы данных.
5. Создать таблицы базы данных (выполнить SQL-скрипты, полученные в предыдущем пункте).
6. Наполнить созданные таблицы данными.

5.4. Лабораторная работа 4 «Разработка SQL-скриптов запросов, хранимых процедур и функций, триггеров»

Цель работы: разработать SQL-скрипты запросов, хранимых процедур и функций, а также триггеров для обеспечения функциональности разрабатываемого программного обеспечения

Задание:

1. Составить, реализовать и протестировать SQL-скрипты запросов для обеспечения функциональности разрабатываемого программного обеспечения.
2. Составить, реализовать и протестировать SQL-скрипты хранимых процедур и функций для обеспечения функциональности разрабатываемого программного обеспечения.
3. Составить, реализовать и протестировать SQL-скрипты триггеров для обеспечения функциональности разрабатываемого программного обеспечения.

6. РАСЧЕТНОЕ ЗАДАНИЕ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ БАЗЫ ДАННЫХ

6.1. План выполнения работы

Цель работы: спроектировать и реализовать программное обеспечение с использованием базы данных.

Основными этапами выполнения расчетного задания являются:

1. Проектирование базы данных.
2. Реализация базы данных.
3. Проектирование программного обеспечения.
4. Реализация программного обеспечения.
5. Тестирование программного обеспечения.
6. Составление пояснительной записки.

Первые два пункта расчетного задания полностью соответствуют четырем лабораторным работам, приведенным выше.

В пункте «Проектирование ПО» необходимо привести разработанную диаграмму классов, описать методы классов.

В пункте «Реализация ПО» необходимо привести обоснованный выбор языка программирования и инструментальной среды программирования, привести экранные формы разработанного ПО.

В пункте «Тестирование ПО» необходимо привести спецификации тестов и результаты тестирования.

Каждый этап работы должен быть подробно описан в пояснительной записке к домашнему заданию. В конце пояснительной записки приводятся выводы по работе, список использованной литературы и листинг кода разработанного ПО.

Объем пояснительной записки – 35-40 страниц текста.

6.2. Варианты заданий

Вариант 1. Деканат факультета

Программное обеспечение должно обеспечить учет сессионной успеваемости студентов факультета университета.

В программе должны быть предусмотрены следующие функции:

- учет специальностей факультета;
- учет учебных планов для каждой специальности конкретного года набора студентов;
- сессионная успеваемость студентов (электронная зачетная книжка студента: кто, что, как, когда, кому сдал зачет или экзамен), оценки проставляются в национальной, 100-балльной шкалах и ECTS-шкале.
- построение рейтингового списка студентов по специальностям, курсу, факультету в трех шкалах (национальной, 100-балльной, ECTS-шкале).

Вариант 2. Депозитные вклады

Программное обеспечение должно обеспечить учет депозитных вкладов клиентов банка, а также начисление и выплаты процентов по ним. Депозитные программы предлагают различные условия:

- валюта вклада (гривны, доллары США, евро);
- срок вклада;
- процентная ставка (в зависимости от валюты, срока и суммы вклада);
- минимальная сумма вклада (для каждой валюты);
- возможность пополнения депозита (не для всех вкладов);
- выплата процентов (ежемесячная или капитализации процентов, т.е. зачисление их на депозитный счет);

- пролонгация вклада (выполняется атоматически или не осуществляется).

В программе должны быть предусмотрены следующие функции:

- условия различных депозитных программ;
- учет депозитных вкладов клиентов банка;
- начисления процентов по вкладам (капиталиция процентов при необходимости);
- пролонгация вклада (при необходимости);
- каждый владелец депозитного вклада имеет свой «кабинет» для промотра начислений ежемесячных процентов и текущего состояния депозитного вклада.

Вариант 3. Гостиница

Программное обеспечение должно обеспечить учет заселения номерного фонда гостиницы и его оплату.

В программе должны быть предусмотрены следующие функции:

- учет номерного фонда гостиницы;
- бронирование мест в гостинице;
- заселение номеров в гостинице;
- предоставление дополнительных услуг (например, питание, стирка и глажка одежды, интернет и т.п.);
- формирование квитанции за проживание в гостинице (с учетов стоимости бронирования и предоставленных дополнительных услуг).

Вариант 4. Частная клиника

Программное обеспечение должно обеспечить учет предоставленных платных услуг в частной клинике.

В программе должны быть предусмотрены следующие функции:

- учет клиентов клиники;
- учет приемов врачами различной специализации (у каждого врача своя стоимость приема;
- учет платных лабораторных исследований;
- формирование квитанции для оплаты предоставленных клиенту услуг;
- гибкая система скидок для клиентов;
- учет доходов и расходов частной клиники.

Вариант 5. Компания экспресс-доставки грузов

Программное обеспечение должно обеспечить учет заказов на перевозку и доставку грузов.

В программе должны быть предусмотрены следующие функции:

- учет клиентов компании;
- учет заказов на перевозку грузов;
- учет тарифов на доставку грузов;
- учет тарифных зон;
- расчет стоимости доставки (стоимость доставки = цена за 1 кг * массу груза + стомость оформления заказа + сумма комиссии, где сумма комиссии = 0,5% * заявленная стоимость груза);
- отслеживание перевозки груза.

Вариант 6. Отдел продаж фирмы

Программное обеспечение должно обеспечить учет заказов на поставку изготавливаемых фирмой изделий.

В программе должны быть предусмотрены следующие функции:

- учет изготавливаемых изделий и норм материалов для их изготовления;
- учет материалов;
- учет клиентов фирмы;
- учет заказов на изготовление изделий;
- формирование квитанций на оплату заказа;
- анализ возможности реализации заказа с учетов имеющихся ресурсов;
- закупка необходимых материалов (формирование закупочной ведомости);
- система скидок для постоянных клиентов.

Вариант 7. Туристическое агентство

Программное обеспечение должно обеспечить учет имеющихся и проданных туров клиентам фирмы.

В программе должны быть предусмотрены следующие функции:

- предоставление информации об имеющихся турах по странам, городам и типам отелей;
- учет сотрудников агентства;
- учет клиентов туристического агентства;
- оформление туров;
- система скидок;
- формирование квитанций на оплату;
- формирование рейтинга отелей по числу оформленных туров;
- начисление премии сотрудникам за оформление тура (премия = стоимость тура * премиальный процент);

- начисление заработной платы сотрудникам (заработная плата = оклад + премия).

Вариант 8. Банковские операции

Программное обеспечение должно обеспечить открытие, ведение и закрытие банковских счетов юридических и физических лиц, а также оформление операций по купле-продаже иностранной валюты.

В программе должны быть предусмотрены следующие функции:

- открытие счетов юридических и физических лиц с указанием банковского процента за зачисление на данный счет денежных средств;
- ведение счетов юридических и физических лиц (зачисление и снятие денежных средств);
- закрытие счетов юридических и физических лиц;
- учет операций по купле-продаже иностранной валюты (с учетом текущего курса валюты);
- формирование квитанций: о зачислении или снятии денежных средств, купле-продаже иностранной валюты;
- формирование отчета о поступивших и снятых денежных средств за указанный период;
- формирование отчета об объемах купленной и проданной иностранной валюты за указанный период.

Вариант 9. Сеть аптек

Программное обеспечение должно обеспечить учет движения медицинских препаратов и средств.

В программе должны быть предусмотрены следующие функции:

- учет медицинских товаров по категориям и фирмам-производителям;
- поиск медицинских товаров;
- поступление товаров в аптеку с указанием поставочных цен;
- формирование наценки на медицинские товары (по категориям и фирмам-производителям);
- расход товаров (формирование чека);
- учет остатков товара в аптеке;
- дисконтная программа (процент скидки изменяется в зависимости от общей суммы всех купленных медицинских товаров);
- перемещение товара между аптеками;
- резервирование товара;
- формирование прайс-листа;
- формирование полного акта инвентаризации (список медицинских товаров, которые на данный момент должны быть в наличии в аптеке).

Вариант 10. Оператор мобильной связи

Программное обеспечение должно обеспечить учет телефонных разговоров абонентов оператора мобильной связи.

В программе должны быть предусмотрены следующие функции:

- хранение информации о тарифных планах;
- учет абонентов и используемого им тарифного плана;
- учет разговоров абонента и их тарификация;
- формирование отчета по звонкам абонента за указанный период;
- формирование отчета о пополнениях счета абонента за указанный период.

Вариант 11. Пиццерия

Программное обеспечение должно обеспечить учет заказов и доставки пиццы клиентам.

В программе должны быть предусмотрены следующие функции:

- хранение информации о видах пиццы, ее составе и весе, а также о ее стоимости;
- учет клиентов пиццерии;
- учет заказов клиентов и их доставки;
- хранение информации об акционных предложениях и их учете при формировании заказа;
- формирование отчетов о сделанных заказах по видам пиццы за указанный период с целью выявления наиболее популярных предложений;
- накопительная система скидок (процент скидки изменяется в зависимости от суммы всех сделанных заказов).

Вариант 12. Детский центр развития

Программное обеспечение должно обеспечить учет посещений детьми развивающий кружков.

В программе должны быть предусмотрены следующие функции:

- хранение информации о занятиях с указанием их стоимости;
- распределение занятий с учетом возрастных категорий: от нуля до двух лет, от двух до четырех лет, от четырех до шести лет;
- хранение информации о преподавателях, ведущих занятия;
- хранение информации о расписании занятий;
- хранение информации о детях, посещающих центр развития;
- хранение информации о том, какие занятия посещают дети;
- хранение новостей центра развития;
- хранение отзывов родителей;

- формирование отчетов о поступивших финансах за указанный период времени;
- система скидок для детей, которые посещают несколько кружков.

Вариант 13. Оптовый магазин

Программное обеспечение должно обеспечить учет поставок и покупок в оптовом магазине.

В программе должны быть предусмотрены следующие функции:

- хранение информации о товарах и их остатках;
- хранение информации о поставках товаров в оптовый магазин (учет приходных накладных);
- хранение информации о продажах товара (учет расходных накладных) и их доставки клиентам;
- хранение информации о клиентах;
- накопительная система скидок (процент скидки изменяется в зависимости от суммы всех сделанных покупок);
- хранение информации об акциях и их учет при продажах товаров;
- оплата товара в рассрочку;
- учет платежей клиента за покупку товара в рассрочку;
- формирование финансовых отчетов (затраты/поступления) за указанный период.

Вариант 14. Библиотека

Программное обеспечение должно обеспечить учет движения книжного фонда библиотеки.

В программе должны быть предусмотрены следующие функции:

- хранение информации о книжном фонде библиотеки;
- хранение информации о новинках книжного фонда;
- хранение информации о читателях библиотеки;
- хранение электронных изданий и обеспечение возможности доступа к ним читателям библиотеки;
- хранение информации о книгах, находящихся на руках у читателя, с указанием срока возврата книг;

формирование списка читателей, которые не вернули в указанный срок книги.

Вариант 15. Домашняя бухгалтерия

Программное обеспечение должно обеспечить учет движения финансов всех членов семьи. В программе должны быть предусмотрены следующие возможности:

- учет расходов по каждому члену семьи;
- учет доходов по каждому члену семьи;
- учет кредитов и их погашение;
- планирование расходов (общее и по каждому члену семьи);
- планирование доходов (общее и по каждому члену семьи);
- построение отчетов и диаграмм по доходам и расходам за месяц, квартал, год;
- обеспечение конфидициальности расходов и доходов каждого члена семьи (при необходимости).

Вариант 16. Супермаркет

Программное обеспечение должно обеспечить учет поступивших и проданных товаров. В программе должны быть предусмотрены следующие функции:
- учет товаров по категориям и подкатегориям;
- продажа товаров (формирование чека);
- учет дисконтных карт клиентов;
- накопительная система скидок;
- различные акции на отдельные товары на указанный период времени (например, снижения стоимости, «два по цене одного», «три по цене двух» и т.п.).

Вариант 17. Риелторская фирма (агентство недвижимости)

Программное обеспечение должно обеспечить учет заключенных договоров купли-продажи и аренды коммерческой и жилой недвижимости.

В программе должны быть предусмотрены следующие функции:
- учет риелторов фирмы;
- учет коммерческой и жилой недвижимости, выставленной на продажу;
- учет коммерческой и жилой недвижимости, предлагаемой для аренды;
- учет заключенных договоров (сделок) купли-продажи и аренды;
- расчет комиссии риелторской фирмы за совершение сделки (комиссия фирмы = сумма сделки * процент комиссии);
- расчет премии риелтора за сделку (премия = комиссия фирмы * премиальный процент);
- начисление заработной платы риелторам (заработная плата = оклад + премиальный процент всех совершенных сделок за месяц).

Вариант 18. Сеть магазинов бытовой техники

Программное обеспечение должно обеспечить учет сотрудников сети магазинов бытовой техники и проданных ими товаров. В программе должны быть предусмотрены следующие функции:

- учет сотрудников по отделам магазина и должностным категориям с окладами, которые соответствуют категории сотрудника;
- учет товаров по категориям и фирмам-производителям;
- продажа товаров (формирование расходной накладной на конкретного сотрудника);
- начисление премии сотруднику за месяц (премия = продажи за месяц * процент премии, который зависит от категории и фирмы производителя товара);
- начисление заработной платы сотрудникам за месяц (заработная плата = оклад + премия);
- формирование рейтинга сотрудников по их продажам за указанный период;
- формирование рейтинга сотрудников по их продажам по всей сети магазинов;
- формирование рейтинга магазинов по объему продаж.

Вариант 19. Страховая компания

Программное обеспечение должно обеспечить учет страховых договоров и возникших страховых случаев.

В программе должны быть предусмотрены следующие функции:

- учет страховых услуг (название и их условия);
- учет заключенных страховых договоров и их оплата;
- учет возникших страховых случаев и выплаты по ним;
- пролонгация страхового договора и установление скидки при этом;
- формирование отчетов (ежемесячных, ежеквартальных, ежегодных).

Вариант 20. Бухгалтерия фирмы

Программное обеспечение должно обеспечить учет сотрудников фирмы, начисление и выплаты им заработной платы.

В программе должны быть предусмотрены следующие функции:

- учет сотрудников по занимаемым должностям и отделам, за которыми они закреплены;
- переводы сотрудников на другую должность или другое место работы;
- начисление ежемесячных премиальных выплат каждому сотруднику в зависимости от выполненных работ (процент от оклада);
- начисления заработной платы (заработная плата = оклад согласно должности + премия за текущий месяц – отчисления);
- выплаты сотрудникам (аванс и заработная плата);

- формирование ежемесячных расчетных листов для каждого сотрудника с указанием начислений, удержаний и выплат в текущем месяце.

Вариант 21. Автозаправочная станция

Программное обеспечение должно обеспечить учет поставленного и проданного топлива.

В программе должны быть предусмотрены следующие функции:

- учет поставок топлива различных видов;
- учет продаж топлива различных видов;
- учет дисконтных карт клиентов;
- накопительная система бонусов (1 литр = 1 бонус);
- покупка сопутствующих товаров за бонусы;
- проведение различных акций с указанием их периода (например, двойное начисление бонусов);
- списание бонусов (1 бонус = 0,1 грн), но списать можно не более 30% от стоимости топлива.

Вариант 22. Объединение совладельцев многоквартирного дома

Программное обеспечение должно обеспечить учет начислений и оплат за предоставленные услуги холодного и горячего водоснабжения, теплоснабжения, электроснабжения, газоснабжения и эксплуатационных расходов для жильцов объединения совладельцев многоквартирного дома. Предполагается, что все квартиры ОСМД обеспечены счетчиками для учета расходов холодной и горячей воды, электричества, газа и теплоснабжения.

В программе должны быть предусмотрены следующие функции:

- учет квартирного фонда ОСМД с указанием: общей и жилой площади каждой квартиры, фамилии, имени и отчества главного квартиросъемщика, числа прописанных человек и т.п.
- учет потребленных за месяц (согласно счетчику) объемов холодной и горячей воды (в куб.м), электричества (в кВт), газа (в куб. м), тепловой энергии (в Гкал) для каждой квартиры;
- эксплуатационные расходы на содержание дома рассчитываются от общей площади квартиры с учетом установленного тарифа за 1 кв.м;
- начисления за месяц для каждой квартиры за: холодное водоснабжение, горячее водоснабжение, электричество, газ, теплоснабжение, эксплуатацию дома (для каждого вида услуг существуют свои тарифы);

- формирование ежемесячной квитанции со всеми видами услуг для каждой квартиры (начислено к оплате = долг на начало текущего месяца + начисления за месяц);
- учет оплат жильцов на единый расчетный счет ОСМД;
- формирование списка должников на указанную дату.

Вариант 23. Интернет-магазин

Программное обеспечение должно обеспечить учет поступивших и проданных товаров в интернет-магазине. В программе должны быть предусмотрены следующие функции:
- регистрация пользователей;
- учет товаров по категориям и подкатегориям;
- фильтрация товаров по указанным параметрам;
- продажа товаров (наполнение корзины, обработка и выполнение заказа, доставка товаров);
- накопительная система скидок (процент скидки изменяется в зависимости от суммы всех сделанных заказов);
- акции на отдельные товары (снижение стоимости товаров с указанием акционного периода и количества акционных товаров).

Вариант 24. WEB-форум

WEB-приложение должно обеспечить создание пользователями (посетителями форума) своих тем с их последующим обсуждением, т.е. размещением сообщений внутри этих тем.

В WEB-приложении должны быть предусмотрены следующие возможности:
- работа с различными типами пользователей (модератор, администратор, зарегистрированный и незарегистрированный посетитель форума);
- модераторы и администраторы форума могут создавать разделы для объединения тем одной тематики;
- зарегистрированный посетитель форума может создавать темы в указанном разделе, оставлять сообщения;
- модераторы могут редактировать сообщения посетителей или удалять их, а также перемещать сообщения в другие темы;
- зарегистрированные пользователи могут обмениваться между собой личными сообщениями;
- зарегистрированный пользователь имеет свой «кабинет», где он может увидеть все свои темы, оставленные сообщения-комментарии, входящие и исходящие личные сообщения.

Вариант 25. Сайт электронных общественных петиций

Программное обеспечение должно обеспечить учет электронных общественных петиций в органы городского управления. В программе должны быть предусмотрены следующие функции:

- регистрация пользователей, желающих подать электронную петицию;
- подача зарегистрированным пользователем петиции;
- возможность зарегистрированным пользователям проголосовать за выбранную им петицию;
- возможность администратору изменить единый срок сбора голосов;
- возможность администратору изменить единое количество голосов, необходимое для отправки петиции на рассмотрение в органы городского управления;
- в случае, если петиция набрала необходимое число голосов, она отправляется на рассмотрение в органы городского управления, после чего публикуется официальный ответ на петицию.

Вариант 26. Аренда автомобилей

Программное обеспечение должно обеспечить учет имеющегося парка автомобилей, а также аренду автомобилей. В программе должны быть предусмотрены следующие функции:

- регистрация нового автомобиля;
- регистрация клиентов;
- бронирование автомобиля на определенный период;
- учет аренды автомобилей;
- учет повреждений автомобилей во время их аренды;
- генерация счетов за аренду автомобиля с учетом предстоящего ремонта автомобиля.
- контроль оплаты счетов клиента.

Вариант 27. Сайт акционных предложений

Программное обеспечение должно обеспечить учет акционных предложений (купонов) для совершения коллективных покупок. В программе должны быть предусмотрены следующие функции:

- регистрация администратором нового акционного предложения (купона) в выбранной категории с установлением срока акции и количества предлагаемых купонов и стоимости купона;

- регистрация пользователя;
- возможность покупки купона, в случае, если сроки акции действительны;
- возможность возврата купленного купона;
- личный кабинет пользователя с возможность просмотра всех купленных купонов.

Вариант 28. Анкетирование

Программное обеспечение должно обеспечить возможность создания анкеты и учет результатов анкетирования. В программе должны быть предусмотрены следующие функции:

- возможность составления анкеты из различных типов вопросов, различного типа заполнения (персонализированное/анонимное);
- возможность заполнение анкеты зарегистрированным или анонимным пользователем (в зависимости от ее типа), в случае если анкетирование открыто;
- сбор статистических данных по ответам на различные вопросы конкретной анкеты.

Вариант 29. Тестирование знаний

Программное обеспечение должно обеспечить возможность создания теста и учет результатов тестирования. В программе должны быть предусмотрены следующие функции:

- возможность составления теста из различных видов заданий закрытого типа (с альтернативным и множественным выбором);
- возможность персонализированного и анонимного прохождения теста;
- личный кабинет тестируемого с возможностью просмотра истории прохождения выбранного теста;
- генерация отчетов о результатах прохождения выбранного теста.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Дейт, К.Дж. Введение в системы баз данных [Текст] / К.Дж. Дейт. − М.: Вильямс, 2006. − 1328 с.
2. Гарсиа-Молина, Г. Системы баз данных. Полный курс [Текст] / Г. Гарсиа-Молина, Дж. Ульман, Дж. Уидом. − М.: Вильямс, 2003. − 1088 с.
3. Кириллов, В.В. Основы проектирования реляционных баз данных [Электронный ресурс] / В.В. Кириллов. – URL: http://www.citforum.ru/database/dbguide/1-1.shtml
4. Кузнецов, С.Д. Введение в реляционные базы данных [Электронный ресурс] / С.Д. Кузнецов. – URL: http://www.intuit.ru/department/database/rdbintro/
5. Нормализация реляционных баз данных [Электронный ресурс]. – URL: http://www.cyberguru.ru/database/database-theory/relational-database-normalization.html
6. Курс лекций по проектированию баз и хранилищ данных [Электронный ресурс]. – URL: http://www.radioland.net.ua/contentid-122-page1.html
7. Карпова, Т.С. Базы данных: модели, разработка, реализация [Текст] / Т.С. Карпова. – СПб.: Питер, 2001. – 304 с.
8. Грошев, А.С. Основы работы с базами данных информация [Электронный ресурс] / А.С. Грошев. – URL: http://www.intuit.ru/department/database/basedbw/
9. Зеленков, Ю.А. Проектирование реляционных баз данных [Электронный ресурс] / Ю.А. Зеленков. – URL: http://www.interface.ru/misc/proekt_1.htm
10. Баженова, И.Ю. Основы проектирования приложений баз данных [Электронный ресурс] / И.Ю. Баженова. – URL: http://www.intuit.ru/department/database/cdba2/
11. Швецов, В.И. Базы данных [Электронный ресурс] / В.И. Швецов. – URL: http://www.intuit.ru/department/ database/databases/
12. Туманов, В.Е. Основы проектирования реляционных баз данных [Электронный ресурс] / В.Е. Туманов. – URL: http://www.intuit.ru/department/database/rdbdev/
13. Рыбанов, А.А. Инструментальные средства автоматизированного проектирования баз данных [Текст]: учеб. пособие и варианты заданий к лаб. работам по дисциплине «Базы данных» / А.А. Рыбанов. – Волгоград: ВолгГТУ, 2007. – 96 с.
14. Маклаков, С.В. BPwin и ERwin: CASE-средства для разработки информационных систем [Текст] / С.В. Маклаков. – М.: Диалог-МиФи, 2000. – 256 с.
15. Шевченко, И.В. Моделирование информационных систем. BPwin, Erwin [Текст]: учеб. пособие / И.В. Шевченко. – Х.: Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2008. – 75 с.

ОГЛАВЛЕНИЕ

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

98

Навчальне видання

Шевченко Ілона Володимирівна

ОСНОВИ ПРОЕКТУВАННЯ БАЗ ДАНИХ: ТЕОРІЯ ТА ПРАКТИКА

(Російською мовою)

Редактор

Зв. план, 2018
Підписано до видання 00.00.2018
Формат 60x84 1/16. Папір офс. № 2. Офс. друк
Ум. друк. арк. 0,0. Обл.-вид. арк. 0,0. Наклад 100 пр. Замовлення 000. Ціна вільна

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001