

# ЗАРИСОВКА НА ТЕМУ «ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ»

Автор: Сергей «Elvin» Соболев

Реляционная модель данных была единовременно разработана в 1970 году Д.Коддом (работавшим в то время в рамках проекта SystemR в компании IBM) как ответ на возросшую потребность рынка информационных технологий в системах управления большими объемами данных. Слово "реляционная", введенное Коддом, обозначает простую вещь — все данные в системе представлены в виде таблиц, которые Кодд почему-то назвал отношениями (*relation* — *отношение*). Строку таблицы Кодд именовал *кортежем* отношения, а столбец — *аттрибутом* (в другой терминологии, которой преимущественно будем пользоваться мы — *поле, field*).

Итак, реляционная модель Кодда, как минимум предполагает представление данных реального мира в виде таблицы. Сразу возникает вопрос — сколько таблиц должно быть в модели данных? Изначально предполагается, что таблица одна.

Рассмотрим в качестве примера данные, которые потребуются для функционирования библиотеке в одном из высших учебных заведений и немедленно приступим к проектированию базы данных, попутно разбираясь в корне возникающих проблем и способах их решения.

## 1 ЭТАП. Изучение предметной области и составление исходной таблицы.

Потенциальный разработчик базы данных для библиотеки должен изучить особенности функционирования библиотеки (в таких случаях говорят, что разработчик должен получить представление о *предметной области*). Скорее всего, в процессе изучения он выяснит, что библиотека нуждается в информации о студентах, которые берут книги и, собственно, о библиотечном фонде. Мы рассмотрим максимально упрощенную модель данных, в которой нас будет интересовать следующая конкретная информация о библиотечном деле:

Информация о студенте (Фамилия, Имя, Отчество, учебная группа, "черная метка", причина занесения в черный список\*).

Информация о книге (Название, Авторство, год издания, издательство, общее количество экземпляров, наличие в данный момент).

\* — поля "черная метка" и причина занесения в черный список имеют следующее назначение. Если по какой-то причине студент занесен в черный список, то книги ему не отпускаются. Причины могут быть разными: например, студент шумит в читальном зале или не вовремя возвращает книги.

На данном этапе проектирования базы данных мы уже можем наметить *типы данных* будущих полей. Так, сведения о фамилии, имени и отчестве студента, очевидно, будут текстовыми; год издания книги — числовое поле; "черная метка" — признак, приобретающий всего два значения — либо студент внесен в черный список, либо нет — следовательно, для данного поля подойдет логический тип.

При выборе типа данных поля исходят из следующих соображений. Поскольку база данных в будущем может значительно разрастись, для снижения возможных проблем связанных со скоростью работы, тип данных должен быть выбран так, чтобы данные в поле

занимали в памяти компьютера минимально возможное место. Что это значит?

Рассмотрим в качестве примера поле данных "черная метка". Зададим для этого поля текстовый тип данных и будем предполагать, что если студент внесен в черный список, то значение данного поля должно быть установлено в "внесен в список", в противном случае — "не внесен в список". Такой способ хранения данных, конечно же, является наглядным. Однако представим теперь бедного пользователя базы данных, который всякий раз при внесении кого—либо в черный список оказывается вынужден писать длинную фразу "внесен в список".

Однако проблемы этим не исчерпываются.

Во—первых, системе управления базой данных, для хранения строки потребуется размер поля минимум 18 байт. Откуда взято это число? Посмотрите, именно столько букв в самой длинной строке данных, которые могут быть внутри поля — "не внесен в черный список". Поскольку при хранении строковых данных, обычно, одна буква занимает в памяти 1 байт, то поле потребует минимум 18 байт.

Во—вторых, при попытке выбрать данные из такой строки компьютеру придется анализировать ее целиком. Например, если мы захотим выбрать из таблицы все записи, которые имеют значение поля "черная метка" равное "внесен в список", то компьютеру придется анализировать все записи таблицы, выборку же он будет делать по значению данных в поле "черная метка". При этом ему придется анализировать каждую (!) букву строки и сравнивать ее со значением шаблона, заданным нашей просьбой. Не трудно заметить, что это может оказаться связано с большими вычислительными затратами (попытайтесь составить алгоритм побуквенного сравнения двух строк).

В—третьих, что делать если в поле окажется опечатка? Например, если пользователь занес в наше поле строку "внемен в список" (буквы "С" и "М" рядом на клавиатуре — велика вероятность опечатки). Скорее всего, это приведет к тому, что при определенных критериях отбора данных, данное значение вообще будет проигнорировано или наоборот использовано лишний раз.

Таким образом, логический тип данных для данного поля является оптимальным, поскольку он в памяти компьютера занимает всего один бит данных, приобретая значения 0 и 1. В нашей модели данных мы должны лишь обеспечить правильное интерпретирование этих данных. Предположим, что единице соответствует значение "внесен в список", а нулю — "не внесен в список". Преимущества этого подхода очевидны. Позже мы также установим, что благодаря связыванию таблиц в реляционной модели мы можем сохранить правила интерпретирования внутри базы данных, а не в своем уме.

Итак, исходная таблица данных может выглядеть следующим образом\*\* :

ФИО	учебная_группа	черная_метка	причина_занесения_в_черный_список	Название
авторство	год_издания	издательство	общее_количество_экземпляров	наличие_в_данный_момент

\*\* — В таблице получилось 10 полей. Мы для удобства изобразили их в два ряда. Однако на самом деле в реальной СУБД такого изображения Вы не увидите. Вам придется включить воображение и представить, что все поля расположены в один горизонтальный ряд.

При именовании полей мы пользовались следующими правилами (совокупность которых обычно называется *мнемоникой*):

- имена всех полей содержат только строчные символы;
- смысловые части имени разделяются знаком подчеркивания;
- имена полей, являющихся ссылками на данные других таблиц формируются по

правилу: "имя внешней таблицы"\_"имя поля во внешней таблице" \*\*\*;  
— первичный ключ во всякой таблице должен называться "id" \*\*\*.

\*\*\* — данным правилом мы воспользуемся позднее.

Обычно мнемоническая система также включает в себя систему *префиксов* (приставок) и *суффиксов*, которые добавляются к имени объекта и позволяют сходу определить какой—либо его признак, например тип данных или принадлежность к какой—либо группе. По сути дела, имя внешней ссылки, которое мы сформируем дальше сформировано с помощью суффикса "\_id" и одного взгляда нам будет достаточно, что данное поле таблицы является внешним ключом определенной таблицы.

При проектировании базы данных пользоваться мнемоникой, конечно, не обязательно. Однако, как только в вашей базе данных окажется более 5—10 таблиц Вы осознаете насколько удобно использовать при назначении имен полей и переменных в текстах программ какую—либо единообразную систему. Кроме того, Вы, разумеется, можете пользоваться своей собственно системой именования.

Каждая строка данных в исходной таблице, по нашему разумению, должна содержать сведения о студенте и книге, которая у него на руках. Пример:

фио	учебная группа	черная метка	причина занесения в черный список	название
авторство	год издания	издательство	общее количество экземпляров	наличие в данный момент
Петров Иван Сергеевич	АТП—95	0		Теоретическая механика
Тарг	1935	Энергия	10	8
Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале	Снежная королева
Андерсен П.	1999	ЭкспоБук	1	0

Возможно Вы уже заметили, что основная (но далеко не единственная) проблема представленной таблицы в том, что каждый студент в нашей модели *не может* иметь на руках более одной книги, а это противоречит естественной сущности данных — библиотека не скажет нам «спасибо» за подобное ограничение базы данных. Как решить эту проблему? В теории БД для таких случаев предусмотрено два тривиальных решения. Первое — заключается в дублировании сущности в пределах кортежа (напомним, что кортежем называется строка таблицы). Если всю информацию о книге (а именно поля "название", "авторство", "год\_издания", "издательство", "общее\_количество\_экземпляров", "наличие\_в\_данный\_момент") обозначить как "книга", то дублирование будет проявлено в том, что появятся несколько записей, навроде: "книга1", "книга2" и т.п. Данный путь рано или поздно принуждает нас остановиться, определившись с количеством дублей, например — 3.

Посмотрим куда мы пришли \*\*\*\*:

\*\*\*\* — В данном варианте таблицы мы для удобства отображения переместили положение поля "название". Мы имели полное право сделать это, поскольку в теории реляционных баз данных место положения поля в таблице не играет никакой роли (наблюдается симметрия горизонтального положения полей, но не значений!)

	фио	учебная_группа	черная_метка	причина_занесения_В_черный_список		
книга1	авторство1	год_издания1	издательство1	общее_количество_экземпляров1	наличие_в_данный_момент1	название1
книга2	авторство2	год_издания2	издательство2	общее_количество_экземпляров2	наличие_в_данный_момент2	название2
книга3	авторство3	год_издания3	издательство3	общее_количество_экземпляров3	наличие_в_данный_момент3	название3
<hr/>						
	Петров Иван Сергеевич	АТП—95	0			
	Тарг	1935	Энергия	10	8	Теоретическая механика
	Кузнецов	1980	МЭИ	50	45	Введение в мат. анализ
	—	—	—	—	—	—
<hr/>						
	Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале		
	Андерсен П.	1999	ЭкспоБук	1	0	Снежная королева
	Маркес Г.	2003	БукБерриХаус	1	0	Полковнику никто не пишет
	Узницецкая П.Л., Гарольд К.	1997	Изд. дом "МадженПресс"	1	0	Шитьё крючком: математика и творчество

Вам кажется, все в порядке? Заведующая нашей воображаемой библиотекой так не думает. Она только что сообщила мне, что Маша Иванова на самом деле еще набрала целый ворох журналов «ELLE» и «GQ»! «Одну минуту» — отвечаем ей мы – «Сейчас мы добавим в исходную таблицу еще несколько дублированных полей!». Чуете к чему это ведет? Оказалось, что предложенное решение совсем неудобно:

— во—первых не ясно, когда нужно остановится при создании дублей и в любой момент может оказаться, что предложенное нами решение представляет собой ограничение естественной сущности данных;

— во—вторых наше решение привело к возникновению *избыточности второго рода*.

*Избыточность* — это серьезная проблема в теории реляционных баз данных. Она проявляется в том, что некоторые данные представлены в таблицах несколько раз — то есть с повторами – в то время, как они бы могли быть представлены единожды. Как минимум, чрезмерная избыточность угрожает нашей базе данных непомерным ростом объема памяти, требуемого для работы с ней, что так или иначе скажется позднее на скорости обработки данных компьютером. Мы упомянули об избыточности второго рода. Так, мы назвали ситуацию возникновения пустых полей в сущности данных, хранящей сведения о книгах Ивана Петрова (мы закрасили эти поля желтым цветом и дополнительно проставили в них прочерки). Обратите внимание, в то время как Маша Иванова, в связи с увлеченностью всем, кроме учебы, набрала множество наименований литературы, поля созданные под это множество остаются незаполненными у Ивана Петрова. Поля не заполнены, но они есть, а это означает, что для их хранения компьютер все равно отводит место в своей памяти.

Если Вы еще не забыли, мы рассматриваем тривиальные решения проблемы устранения некоторых ограничений в исходной таблице. Обратим внимание на второе решение, которое заключается в дублировании кортежей (и еще раз напомним, что кортежем в теории Кодда называется строка таблицы). Ранее мы виртуально сгруппировали поля данных таблицы, относящиеся к сведениями о книгах, в группы с именем "книга№". Мы расширили исходную таблицу до трех таких групп, но выяснилось, что естественная схема

данных требует неопределенного числа книг для произвольной записи таблицы. Так, Маша Иванова (из—за своего пристрастия к модным журналам) имеет на руках значительно больше 3 наименований литературы. Дублирование кортежей производится по следующей схеме: для внесения дополнений данная строка данных полностью дублируется в другой строке, а данные требующие расширения в дублированной строке заменяются новыми данными. Для нашей таблицы это может выглядеть так:

книга	фio	учебная_группа	черная_метка	причина_занесения_В_черный_список		
	авторство1	год_издания1	издательство1	общее_количество_экземпляров1	наличие_в_данный_момент1	название1
	Петров Иван Сергеевич	АТП—95	0			
	Тарг	1935	Энергия	10	8	Теоретическая механика
	Петров Иван Сергеевич	АТП—95	0			
	Кузнецов	1980	МЭИ	50	45	Введение в мат. анализ
	Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале		
	Андерсен П.	1999	ЭкспоБук	1	0	Снежная королева
	Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале		
	Маркес Г.	2003	БукБерриХаус	1	0	Полковнику никто не пишет
	Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале		
	Узницева П.Л., Гарольд К.	1997	Изд. дом "МадженПресс"	1	0	Шитьё крючком: математика и творчество
	Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале		
		1997	Изд. дом "GQ Publishing"	1	0	GQ # 27

и т.д. ...

Таким образом, мы сохранили в исходной таблице сведения о 8—ми книгах Маши. Вам кажется очевидной необходимость полного дублирования всей строки целиком? Нам так не кажется и тому есть свои причины. Самая главная причина связана с тем, что при таком способе представления данных в таблице возникает избыточность первого рода. "Опять эта избыточность!" — скажете Вы и будете правы: совсем бесперспективно дублировать таким образом строки, всякий раз осуществляя перенабор значений в полях. Вы можете решить не заполнять какие—либо поля при дублировании, но в таком случае вы рискуете нарушить целостность данных, потому что в общем случае не понятно какие именно поля можно оставлять незаполненными, а какие требуется заполнять в любом случае. Рассмотрим это на нашем примере. Из—за лени мы оставили незаполненными поля связанные с черным списком, а также поле "учебная\_группа" во всех дублях (мы выделили эти поля желтым цветом и поставили в них прочерки):

книга	фио	учебная_группа	черная_метка	причина_занесения_В_черный_список		
	авторство1	год_издания1	издательство1	общее_количество_экземпляров1	наличие_в_данный_момент1	название1
	Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале		
1	Андерсен П.	1999	ЭкспоБук	1	0	Снежная королева
	Иванова Мария Владимировна	—	—	—		
2	Маркес Г.	2003	БукБерриХаус	1	0	Полковнику никто не пишет
	Иванова Мария Владимировна	—	—	—		
3	Узницецкая П.Л., Гарольд К.	1997	Изд. дом "МадженПресс"	1	0	Шитьё крючком: математика и творчество
	Иванова Мария Владимировна	—	—	—		
4		1997	Изд. дом "GQ Publishing"	1	0	GQ # 27
и т.д...						

Уже лучше с точки зрения практического набора данных, но декан факультета не согласен с нами. Он сообщает, что в группе "ИВТ — 97" учится прирожденный математик Иванова Мария Владимировна и это совсем другой человек! Возникает праведный вопрос: как определить какая из Маш владеет книгами, обозначенными номерами 2—8? Кроме того, напомним, что незаполненность полей вовсе не гарантирует уменьшение объемов места, занимаемых этими полями.

Чтобы избежать подобной проблемы при дублировании необходимо ВСЕГДА дублировать все записи целиком. Однако и в этом случае целостность легко нарушить. В конце семестра Маша Иванова из группы ПТЭ—96 завалила сессию. Благодаря помощи знакомого врача она оформила академический отпуск и приступит к учебе повторно в следующем году с группой ПТЭ—97 (как Вы уже догадались, номер в имени группы в нашем вузе означает учебный год, с которого началось обучение студента). Случилось так, что пользователь базы данных, внес исправленный вариант записи в поле "учебная\_группа" только в одной из строк (ячейки некорректных строк мы выделили желтым цветом):

Иванова Мария Владимировна	ПТЭ — 97	1	шумит в читальном зале	0	Снежная королева
Андерсен П.	1999	ЭкспоБук	1		
Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале	0	Полковнику никто не пишет
Маркес Г.	2003	БукБерриХаус	1		
Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале	0	Шитьё крючком: математика и творчество
Узницецкая П.Л., Гарольд К.	1997	Изд. дом "МадженПресс"	1		
Иванова Мария Владимировна	ПТЭ — 96	1	шумит в читальном зале	0	GQ # 27
	1997	Изд. дом "GQ Publishing"	1		
и т.д...					

Поскольку запись для "Иванова Мария Владимировна" существует в отношении сразу двух учебных групп логично предположить, что речь идет о двух разных людях. Как это не прискорбно, но наша Маша раздвоилась — целостность базы данных нарушена, ибо данные в таблице противоречат действительности.

## 2 ЭТАП. Упрощение полей.

До основополагающего труда Кодда 1970 году мы не имели инструментов расширения таблиц за исключением двух видов дублирования. Дублирование, как было показано выше, ведет к возникновению избыточности. Избыточность часто вызывает нарушения целостности данных.

На примере исходной таблицы хорошо видно, что избыточность появляется не только в связи с дублированием. Так, избыточностью первого рода страдают поля "учебная\_группа" и "причина\_занесения\_в\_черный\_список" — значения обоих полей склонны повторяться от строки к строке. Таким образом, для процесса проектирования базы данных становятся важны средства устранения избыточности, которые волей словообразовательного таланта Кодда были названы *нормализацией*. Заслуга Кодда в том, что он разработал весьма четкую стратегию устранения избыточности в базе данных за пять шагов. После выполнения каждого шага таблица оказывается на одну ступень ближе к идеалу, в котором вообще нет избыточности. Каждый шаг устранения избыточности был назван Коддом *нормальной формой*.

Первый шаг — приведение к первой нормальной форме — заключается в устранении *семантически сложных* полей. Звучит загадочно, не так ли? В науках о языках, *семантика* — это раздел, изучающий взаимосвязь между формой и смыслом. Формой в языке является слово, ну а смысл закреплен за интерпретатором слова. (Яснее пока не стало, правда?). Семантически сложное поле — это поле, за которым закреплено сразу несколько смыслов из—за того, что внутри него содержится несколько форм. Семантически сложные поля трудны для машинного анализа — для их разбора в компьютере должны быть предусмотрены специальные средства (например программы лексического анализа текста). Так зачем же усложнять себе и компьютеру жизнь, когда такие поля можно изначально разделить на несколько полей? Об этом и говорит Кодд, вводя понятие первой нормальной формы: таблица представлена в первой нормальной форме, если все ее поля являются простыми и (синтаксически) неделимыми.

В исходной таблице мы имеем, по крайней мере, одно синтаксически сложное поле — "фио". Представьте ситуацию, что в некоторой ведомости нам требуются только фамилии студентов. Если мы оставим поле "фио" как есть, то для извлечения из него фамилии нам потребуются какие—либо вспомогательные средства, например, текст программы написанной на языке программирования, умеющей извлекать фамилию из поля "фио". Такая программа будет скорее всего предполагать, что фамилия является первым словом внутри поля "фио" и признаком завершения подстроки, содержащей фамилию будет наличие пробела. К сожалению, пользователь базы данных однажды позабыл о такой особенности нашей баз и набрал строку вида:

1	Алексей	Сергеевич	ПТЭ	—	0		
	Сидоров		96				
	Тарг		1935		Энергия	10	8 Теоретическая механика

В итоге среди фамилий в ведомости оказалось имя — целостность базы данных снова нарушена.

Мы сможем избежать этой и многих других проблем упростив поле "фио", разбив его на три простых поля: "фамилия", "имя" и "отчество".

### 3 ЭТАП. Связанные таблицы.

Приведения к первой нормальной форме недостаточно, чтобы в разумных пределах устранить избыточность в базе данных. Мы устраним избыточность только на данном этапе, приведя все таблицы к третьей нормальной форме<sup>\*\*\*\*</sup>.

\*\*\*\* — Вы возможно заметили, что мы упустили вторую нормальную форму. Поясним с чем это связано. Формулировка нормализации Кодда предполагает, что таблица приводится к третьей нормальной форме будучи уже приведенной ко второй и первой. Поэтому мы ничего не теряем, разбирая третью нормальную форму. Более того, проектирую БД поэтапно (так, как это делаем мы), Вы гарантированно приведете ее к третьей нормальной форме, избежав *транзитивных зависимостей*. Формализм Кодда связан с емкостью его методики — из—за того, что в базе данных возможны составные первичные ключи возникают транзитивные зависимости. В нашей методике мы пользуемся только одинарными первичными ключами и даже более — мы не пользуемся естественными полями исходной таблицы при создании ключевых полей, а это полностью избавляет нас от возникновения транзитивных зависимостей. Если Вас пугает термин "транзитивная зависимость", мы поясним, что он связан с тем, что в таблице присутствуют поля зависящие от первичного ключа через какое—либо постороннее (транзитное) поле. Так, в исходной ненормализованной таблице мы имеем поле "год\_издания" книги находящейся на руках у какого—либо студента. Очевидно, что значение этого поля связано с тем, какая именно книга на руках у студента. В то же время то, какая это книга, целиком определяется ее принадлежностью студенту. Первичной здесь является информация о студенте А. Информация о книге Б является зависящей от студента. Информация о годе издания (В) является зависимой от книги. Имеем транзитивную зависимость:  $V(B(A)) = V(A)$ . Разумеется, такого рода зависимость имеется только в отношении построенной нами модели.

В качестве примера мы устраним избыточность поля "учебная\_группа". Для этого создадим вспомогательную таблицу "группы", в которой перечислим все возможные значения поля "учебная\_группа", а в поле "учебная\_группа" каким—либо образом укажем, с какой именно записью таблицы "группы" оно связано. И вот как мы это сделаем.

Помимо перечисления значений в таблицу "группы" мы включим специальное поле, значение которого будет однозначно идентифицировать запись в таблице "группы". По терминологии реляционных баз данных такое поле (или поля) называется *ключевым полем*, *первичным ключом* (*primary key*) или, просто, *ключом* (*key*). Введя правила мнемоники на первом этапе мы условились, что первичный ключ будет всегда именоваться "id" — общепринятое сокращение от слова идентификатор. Таким образом, таблица "группы" будет иметь следующий вид:

Таблица "группы"

id	название группы
1	ПТЭ — 96
2	АТП — 95
3	ПТЭ — 97
4	ИВТ — 97

В исходной таблице мы введем вспомогательное числовое поле, являющееся ссылкой на значение в таблице "группы". Назовем это поле в соответствии с принятыми правилами

именования "группы\_id". Поле "учебная\_группа" можно совсем удалить. Тогда исходная таблица приобретает вид (опустим в этой модифицированной таблице все поля, кроме приведенных и назовем ее "студенты"):

Таблица "студенты"

фамилия	имя	отчество	группы_id
Петров	Иван	Сергеевич	2
Иванова	Мария	Владимировна	1
Сидоров	Алексей	Сергеевич	1
Иванова	Мария	Владимировна	4

Таким образом, зная, что поле "группы\_id" является ссылкой на соответствующую строку таблицы "группы" мы всегда можем восстановить исходную таблицу, выполнив операцию склеивания таблиц, называемую естественным объединением. Поле "группы\_id" содержит в себе значения ключа, представленного в поле "id" таблицы "группы"; по отношению к исходной таблице "группы" является внешней, поэтому ключи, подобные "группы\_id" называют *внешними*.

Надеюсь, Вы обратили внимание, что ключи подобны закладкам в книге, они позволяют легко восстановить данные по своему содержанию, являясь в то же время намного проще самих данных. Так, вместо приведения целой главы некой книги в монографии, мы можем просто указать ключ — страницу, на которой эта монография представлена. Именно так мы понимаем понятие ключа в базе данных. Важное свойство ключа — его *уникальность*: в одной таблице не может быть двух строк с одним и тем же значением ключа (как в книге не может быть двух страниц с одним и тем же номером — это бессмысленно). Именно уникальность первичного ключа позволяет зная его, обращаться к строго определенной строке таблицы.

Самое замечательное в произведенном нами разбиении — то, что мы можем дополнять внешнюю таблицу данными и это никак не скажется на исходной таблице. Так случилось что в нашем виртуальном вузе каждая группа имеет страницу в интернете. Если мы разместим эту информацию в исходной таблице, то мы столкнемся сразу с двумя типами избыточности — первого (потому что адрес страницы будет продублирован сразу у всех студентов группы) и третьего (поскольку такая информация является транзитивной и зависит от студента через поле "группы\_id") рода. Имея внешнюю таблицу "группы" нам кажется такое расширение нецелесообразным. Действительно, отнесем эти сведения к группе и они автоматически по внешнему ключу станут доступны для каждого из студентов:

Таблица "группы"

id	название группы	интернет страница
1	ПТЭ — 96	www.virtualvuz.ru/pte96
2	АТП — 95	www.virtualvuz.ru/atp95
3	ПТЭ — 97	www.virtualvuz.ru/pte97
4	ИВТ — 97	www.virtualvuz.ru/ivt97

Сама же исходная таблица никак не изменится.

Устраняя избыточность всех прочих полей исходной таблицы мы приходим к структуре базы данных, в которой присутствуют основные и вспомогательные таблицы:

## ОСНОВНЫЕ ТАБЛИЦЫ

**студенты** (id / фамилия / имя / отчество / группы\_id)

**книжный\_фонд** (id / название / авторство / год\_издания / издательство / всего\_экземпляров)

## ВСПОМОГАТЕЛЬНЫЕ ТАБЛИЦЫ

Справочники (справочные таблицы)

**группы** (id / название\_группы / интернет\_страница)

**причина\_попадения\_в\_черный\_список** (id / причина)

Таблицы-связи

**черный\_список** (студенты\_id / причина\_попадения\_в\_черный\_список\_id)

**книги\_на\_руках** (студенты\_id / книжный\_фонд\_id)

Обратите внимание, что все таблицы (за исключением, быть может, таблиц-связей) в этой системе снабжены первичным ключом "id".

Разделение на основные и справочные таблицы весьма условно; назначение справочных таблиц — расшифровывать значения внешних ключей. Наша система может показаться Вам слишком простой или неудобной, мы, разумеется, используем ее для иллюстрации. У Вас может возникнуть закономерный вопрос: как мы пришли к этой системе? Мы не можем дать ответа на этот вопрос, поскольку во многом это творческий процесс. Как и в любом процессе наложения системы на предметную область, в процессе разбиения исходной таблицы могут быть варианты. Конечный вариант определяется из требования к базе данных, из условий эффективности исполнения запросов и т.п. Мы исходили из того, что если данные некоторой предметной подобласти требуют для хранения несколько полей, то они могут полностью ограничиваться какой-либо таблицей и их разумно располагать в этой таблице. В итоге мы получили модель данных, в которой основные таблицы полностью самодостаточны (если, конечно же, внешние ключи расшифрованы в результате операции естественного объединения основных таблиц со справочниками или другими основными таблицами).

Выше мы разобрались с устройством двух таблиц из этой схемы — таблицы "студенты" и таблицы "группы". Мы также установили, что естественное объединение этих таблиц производится посредством хранения внешнего ключа в таблице "студенты", который позволяет восстановить значение в таблице "группы" исходя из равенства первичного и внешнего ключей. Дополним таблицу студенты первичным ключом и она будет полностью готова:

Таблица "студенты"

id	фамилия	имя	отчество	группы id
1	Петров	Иван	Сергеевич	2
2	Иванова	Мария	Владимировна	1
3	Сидоров	Алексей	Сергеевич	1
4	Иванова	Мария	Владимировна	4

Рассмотрим как в нашей базе данных осуществляется прочее деление и база

приобретает законченный вид (как это показано на схеме).

Сначала разберемся с черным списком студентов. Во-первых, устраним избыточность поля "причина\_попадения\_в\_черный\_список" исходной таблицы тем же способом, как это было сделано для поля "учебная\_группа", а именно, создадим вспомогательную справочную таблицу:

Таблица "причина\_попадения\_в\_черный\_список"

id	причина
1	шумит в читальном зале
2	возвращает книги не в срок

Далее мы должны были бы включить ссылку на поле "id" этой таблицы в исходную таблицу, как мы это делали при устранении избыточности поля "учебная\_группа". Мы поступим по-другому. Поскольку скорее всего лишь малая часть студентов, по отношению к их общему числу, входит в черный список (хочется надеяться, что это не противоречит естественной сущности данных ;), то при работе с полем "черная\_метка" мы сталкиваемся с избыточностью второго рода — зачем вообще для всех прочих студентов хранить сведения о том, внесены ли они в черный список и какова причина несуществующего внесения? С аналогичным проявлением избыточности второго рода мы сталкиваемся при необходимости хранения в базе данных любых "редких" качеств: например, сведений о том, какие студенты составляют футбольную команду вуза (если наш вуз не спортивный, то очевидно таковых будет не много) или кто из студентов в прошлом году получал стипендию Фонда Сороса (таких явно немного :). Поэтому указанные сведения мы сохраним в отдельной вспомогательной таблице "черный\_список". Эта таблица будет содержать только информацию о том, какие студенты внесены в список и по какой причине:

Таблица "причина\_попадения\_в\_черный\_список"

студенты id	причина_попадения_в_черный_список id
2	1
3	2

Как видим, "читабельность" информации в этой таблице существенно ниже той, что мы имели в самом начале. Но нам об этом можно не волноваться — все современные БД имеют средства отображения данных в виде отчетов и форм и конечный пользователь может даже и не подозревать, что на самом деле работает с нормализованными разбитыми таблицами.

Зная, что из себя представляют фактические данные таблицы мы можем произвести *естественное объединение*. Так, первая строка этой таблицы содержит два числа: 2 и 1.

Первое число представляет собой внешний ключ "студенты\_id", значит нам нужно просмотреть все записи таблицы "студенты" и отыскать там ту запись, значение "id" которой равно 2 — эта запись соответствует студенту "Иванова Мария Владимировна". Таким образом Иванова Мария Владимировна включена в черный список (чтобы узнать из какой группы эта студентка мы должны будем также расшифровать внешний ключ "группы\_id" в таблице "студенты").

Второе число позволяет нам расшифровать причину, поскольку оно является внешним ключом, ссылающимся на поле "id" в таблице "причина\_попадения\_в\_черный\_список", следовательно нам нужно просмотреть все записи в таблице "причина\_попадения\_в\_черный\_список" и найти единственную с id = 1 — найдя эту запись мы можем установить причину обратившись к полю "причина". Записи с "id"=1

соответствует "шумит в читальном зале". В итоге имеем естественное объединение: "Иванова Мария Владимировна шумит в читальном зале".

Проведя подобные манипуляции для второй строки мы можем установить, что "Сидоров Алексей Сергеевич возвращает книги не в срок".

Обратим внимание на тот факт, что полное объединение часто бывает не нужно. Так, если нам потребуется перечень студентов, не занесенных в "черный список" нашей библиотеки, мы выполним естественное объединение таблиц "студенты" и "черный\_список" и осуществим выбор тех значений таблицы "студенты" поле "id" которых не представлено среди значений внешнего ключа "студенты\_id" таблицы "черный\_список". При этом нам совсем не нужна оказывается таблица "причина\_попадения\_в\_черный\_список". Действительно, в процессе отбора подобного перечня нам интересен факт занесения в "черный\_список" и совсем не интересная причина. Здесь же еще раз отметим, что компьютер гораздо быстрее осуществляет сравнение числовых значений, чем строк, поэтому внедрение числовых ключей вполне оправдано.

Связи, возникшие между таблицами, благодаря введению понятия внешнего и первичного ключей, с которыми мы столкнулись до сих пор называются связями типа "*один-ко-многим*". Действительно, одна и та же причина попадания в черный список может встречаться у многих студентов, таким образом одному значению первичного ключа "причина\_попадения\_в\_черный\_список.id" может соответствовать много внешних ключей "черный\_список.причина\_попадения\_в\_черный\_список\_id" \*\*\*\*\*.

\*\*\*\*\* — здесь и далее для удобства записи мы указываем имя таблицы и после символа точки "." имя поля этой таблицы вместо записи типа "поле такое-то таблицы такой-то".

Аналогичная ситуация с полем "учебная\_группа", которое в самом начале было разбито на первичный ключ "группы.id" и внешний ключ "студенты.группы\_id": одной записи первичного ключа "группы.id" соответствует много записей внешнего ключа "студенты.группы\_id" поскольку в одной группе может быть много студентов (и обычно это именно так).

Если мы предположим, что один студент попадает в "черный\_список" по единственной причине, то связь между первичным ключом "студенты.id" и внешним ключом "черный\_список.студенты\_id" будет выражать тип связей "*один-к-одному*": единственному значению первичного ключа соответствует единственное значение внешнего ключа и сам внешний ключ может рассматриваться как первичный ключ внешней таблицы.

Обратим внимание на тот факт, что таблица "черный\_список" не имеет первичного ключа. Это связано с тем, что ни в одной другой таблице модели данных нашей БД не потребуется ссылаться на записи таблицы "черный\_список".

Теперь рассмотрим более интересный случай, когда в таблице "черный\_список" один и тот же студент может встречаться несколько раз. То есть связь между первичным ключом "студенты.id" и внешним ключом "черный\_список.студенты\_id" является связью типа "*один-ко-многим*". То есть мы можем предположить возможность дублирования строк этой таблицы, при котором расширением дублирования является поле "черный\_список.причина\_попадения\_в\_черный\_список\_id", а поле "черный\_список.студенты\_id" дублировано. Это значит, что, например, Мария Владимировна Иванова не только шумит в читальном зале, но еще и сдает книги не в срок. Дублирование

будет выглядеть следующим образом:

студенты id	причина попадания в черный список id
2	1
2	2

Выполнив естественное объединение мы сможем получить две строки "Иванова Мария Владимировна шумит в читальном зале" и "Иванова Мария Владимировна сдает книги не в срок", что может вполне соответствовать естественной сущности данных.

Теперь посмотрим на взаимоотношения, сформировавшиеся между ключами такой таблицы. Внешний ключ "черный\_список.студенты\_id" и первичный ключ "студенты.id" связаны, как было показано выше, типом отношения "один-ко-многим"; с другой стороны внешний ключ "черный\_список.причина\_попадения\_в\_черный\_список\_id" и первичный ключ "причина\_попадения\_в\_черный\_список.id" также связаны типом отношения "один-ко-многим".

Какое отношение возникает между внешними ключами "черный\_список.студенты\_id" и "черный\_список.причина\_попадения\_в\_черный\_список\_id"? Оказывается, что одна причина может быть сразу у многих студентов, но один студент может иметь сразу много причин. Такое отношение в теории реляционных баз данных называется связью "многие-ко-многим" и оно возникло в нашей базе данных само собой — мы ничего не делали специально для его развития, мы просто проанализировали связи уже сформированных отношений.

Как видим отношение "многие-ко-многим" возникло на пересечении внешних ключей двух отношений "один-ко-многим". Мы воспользуемся этим фактом для разрешения очень сложной ситуации, возникшей при проектировании базы данных для нашей библиотеки — связь типа "многие-ко-многим", несмотря на свое неудобство, естественным образом возникает при анализе взаимных отношений между сущностью, представляющей собой студента и сущностью, представляющей собой "книгу". Каждый студент может иметь на руках множество книг (неопределенное число — мы столкнулись с этим, когда выяснили, что Маша Иванова читает модные журналы). В то же время книга одно и того же наименование книжного фонда библиотеки может быть у большого числа студентов (благодаря тому факту, что каждая книга представлена не одним экземпляром).

Далее мы увидим, что любое отношение "многие-ко-многим" может быть разрешено парой отношений "один-ко-многим" внутри таблицы связи.

Сначала реализуем таблицу, в которой описаны книги:

Таблица "книжный фонд"

id	название	авторство	год издания	издательство	всего экземпляров
1	Теоретическая механика	Тарг	1935	Энергия	10
2	Введение в мат. анализ	Кузнецов	1980	МЭИ	50
3	Снежная королева	Андерсен П.	1999	ЭкспоБук	1
4	Полковнику никто не пишет	Маркес Г.	2003	БукБерриХаус	1
5	Шитьё крючком: математика и творчество	Узницецкая П.Л., Гарольд К.	1997	Изд. дом "МадженПресс"	1
6	Журнал GQ. №27	Изд. дом "GQ Publishing"	2003	Изд. дом "GQ Publishing"	1
7	Журнал GQ. №30	Изд. дом "GQ Publishing"	2003	Изд. дом "GQ Publishing"	1
И т.д. . .					

Далее нам нужна информация о том какой студент взял какую книгу. Где нам расположить эту информацию? В самом начале нашего экскурса мы показали, что размещение внутри таблицы "студенты" нелогично, потому что приводит к дублированию. Та же ситуация возникает при попытке размещения внутри таблицы "книжный фонд". Мы создадим отдельную таблицу и назовем ее "книги\_на\_руках". В этой таблице мы представим всю информацию о студенте путем указания внешнего ключа "книги\_на\_руках.студенты\_id" и всю информацию о книге путем указания внешнего ключа "книги\_на\_руках.книжный фонд\_id". Заполним эту таблицу внимательно стыкуя внешние ключи с первичными, не допуская нарушения целостности:

Таблица "книги\_на\_руках"

студенты id	книжный фонд id
-------------	-----------------

У Маши Ивановой (id=2) 8 книг

1	2	3
2	2	4
3	2	5
4	2	6
5	2	7
6	2	8
7	2	9
8	2	10

У Ивана Петрова (id=1) 2 книги

1	1	1
2	1	2

Кроме того Теоретическую механику (id=1) взял Сидоров Алексей (id = 3)

1	3	1
---	---	---

Порядок следования записей в таблице "книги\_на\_руках" не имеет никакого значения. Это справедливо для любых таблиц в реляционной модели: говорят, имеется симметрия строк таблицы в вертикальном направлении. Поэтому таблица "книги\_на\_руках" могла выглядеть и так тоже:

Таблица "книги\_на\_руках"

студенты id	книжный фонд id
1	1
2	3
2	10
1	2
3	1
2	9
2	5
2	6
2	7
2	8
2	6

Обратите внимание, что в нашей модели потерялось одно данное, присутствующее в исходной таблице: это поле "наличие\_в\_данный\_момент", которое предполагает возможность установления факта присутствия книги на книжной полке. В итоговой схеме

данных мы вообще отказались от этого поля, поскольку его значение может быть вычислено.

Так, определим наличие в данный момент книги "Теоретическая механика". Из таблицы "книжный\_фонд" определяем "id"=1 и "всего\_экземпляров"=10. Теперь обратимся к таблице "книги\_на\_руках".

Таблица "книги\_на\_руках"

студенты id	книжный фонд id
1	1
2	3
2	10
1	2
3	1
2	9
2	5
2	6
2	7
2	8
2	6

Выделив в ней, только записи, внешний ключ "книжный\_фонд\_id" которых совпадает с id книги "Теоретическая механика" и подсчитав общее количество таких записей  $X=2$ , мы можем легко установить:

$$\text{наличие\_в\_данный\_момент} = \text{"книжный\_фонд.всего\_экземпляров"} - X = 8.$$

Полученное значение совпадает с числом, находящимся для этого поля в исходной таблице.

Возможность установления производных величин, а также критериальный отбор данных — существенно определяют качество выбранной модели данных в базе данных.

В спроектированной нами базе всего шесть таблиц, но она уже является достаточно сложной. Наша база может быть существенным образом расширена, путем введения дополнительных полей и справочных таблиц. Увеличение числа таблиц сильно снижает возможности "охвата" базы данных целиком, поэтому для представления модели данных часто пользуются вспомогательными механизмами. Так, Microsoft Access позволяет рисовать схему данных, показывая на ней как связаны друг с другом таблицы:



Полностью ли нормализована наша база данных? Очевидно, нет. Так, в таблице "книжный\_фонд" у нас остались семантически сложные поля, например, поле "авторство", допускающее хранение группы авторов. Такое устройство поля "авторство" на первых порах существенно облегчает нашу жизнь, позволяя сосредоточить максимум внимания на реализации других возможностей БД. Когда-нибудь у нас дойдут руки и до этого поля и мы применим к нему все трансформации, которые мы применили ранее в полях типа "книга". При этом у нас неизбежно возникнет дилемма: где хранить информацию об одном конкретном авторе? Видимо, это будет таблица "персоны", в которой помимо первичного ключа будут поля "фамилия", "имя" и "отчество". Эти поля аналогичны таким же полям в таблице "студенты". Вот почему о таких полях говорят, что они семантически подобны. Хранение семантически подобных полей в разных таблицах демонстрирует нам пример последнего — четвертого — рода избыточности. В идеале мы также сможем устранить ее, сведя все записи о людях в одну таблицу "персоны" и сделав внешние ссылки на нее из таблиц "студенты" и "авторство" (как это показано на рисунке ниже). Читательность таблицы "студенты" при этом еще более снизится и наша основная таблица, по сути дела, превратится в аналог таблицы-связи. От таблицы-связи, навряде "книги\_на\_руках", ее будет отличать лишь то, что в ней могут храниться не только внешние ключи, но и какие-нибудь собственные данные, например, номер зачетной книжки студента и т.п. На этом примере хорошо видно, что деление схемы данных на основные и вспомогательные таблицы весьма условно.



### Синописис

Проектирование БД:

- 1 этап. Проектирование исходной ненормализованной таблицы.
- 2 этап. Упрощение полей.
- 3 этап. Разбиение исходной таблицы на совокупность связанных таблиц.

## Словарик

*Атрибут* – то же, что и *поле*.

*База данных* – набор нормализованных таблиц.

*Естественное объединение* – процесс склеивания таблиц, в результате которого восстанавливается исходная таблица. Склеивание возможно благодаря выполнению условия равенства значений между первичными и внешними *ключами* таблиц базы данных.

*Запись* – то же, что и *кортеж*.

*Избыточность первого рода* — наличие в таблице повторяющихся значений (не являющихся числами).

*Избыточность второго рода* — наличие в таблице полей несущественных (неразличимых) для основного числа кортежей (строк) таблицы.

*Избыточность третьего рода* (транзитивные зависимости в ненормализованной таблице) — наличие в таблице полей, зависящих от какого—либо поля (полей), не являющегося первичным ключом.

*Избыточность четвертого рода* — хранение семантически подобных данных в разных таблицах.

*Кортеж* – строка таблицы.

*Ключ, ключевое поле* – поле таблицы, однозначно идентифицирующее данную строку.

*Ключ внешний* – ключ, расположенный во внешней по отношению к данной таблице и представляющий собой ссылку на первичный ключ данной таблицы.

*Ключ первичный* – ключ, расположенный в данной таблице и идентифицирующий данную строку.

*Мнемоника* – набор правил записи текста, необязательный к выполнению.

*Нормализация* — устранение избыточности в модели данных.

*Отношение* – таблица с данными.

*Поле* – столбец таблицы.

*Префикс* – приставка, добавляемая к началу слова.

*Реляционная (модель)* – построенная на таблицах.

*Семантика* – смысловое начало.

*Суффикс* – приставка, добавляемая к концу слова.

*Тип данного* – информация, определяющая исключительный набор операций, а также семантику некоторого данного.