

Лекция 1. Проектирование базы данных (БД)

Термин «реляционный» означает «основанный на отношениях». Реляционная база данных состоит из сущностей (таблиц), находящихся в некотором отношении друг с другом. Название произошло от английского слова **relation** (отношение). Проектирование базы данных состоит из двух основных фаз: логического и физического моделирования. Во время логического моделирования собираются требования и разрабатывается модель базы данных, не зависящую от конкретной СУБД (системы управления реляционными базами данных). Это похоже на то, как если бы создавался чертеж дома – можно было бы продумать и начертить все: где будет кухня, спальни, гостиная. Но это все на бумаге и в макетах. Во время физического моделирования создается модель, оптимизированную для конкретного приложения и СУБД. Именно эта модель реализуется на практике.

Процесс проектирования базы данных состоит из следующих этапов:

- 1) сбор информации;
- 2) определение сущностей;
- 3) определение атрибутов для каждой сущности;
- 4) описание структуры базы данных (в том числе определение связей между сущностями);
- 5) нормализация;
- 6) преобразование к физической модели;
- 7) создание базы данных.

Первые 5 этапов соответствуют фазе логического проектирования, а остальные два — фазе физического моделирования.

1.1 Логическая фаза

Логическая фаза состоит из нескольких этапов.

1. Сбор информации (требований)

На этом этапе необходимо точно определить, как будет использоваться база данных, и какая информация будет в ней храниться. Для этого нужно собрать как можно больше сведений о том, что разрабатываемая система (ресурс) должна делать и чего не должна.

Полезным будет составление списка всех видов данных, которые планируется хранить, а также сущностей (лиц, объектов, мест, событий и т.д.), которые описывают эти данные. В качестве примера рассмотрим условный процесс продажи товаров, где будут задействованы три категории данных – клиенты, менеджеры, товары и заказы.

Клиенты

- Имя
- Фамилия

- Дата рождения
- Адрес
- Город, регион
- Телефон
- Адрес электронной почты

Менеджер (консультант)

- Имя
- Фамилия
- Адрес
- Телефон
- Адрес электронной почты
- Общая стоимость выполненных заказов

Товары

- Название
- Описание
- Цена
- Количество на складе

Заказы

- Номер заказа
- Консультант
- Клиент
- Дата
- Товар(ы)
- Количество
- Цена
- Общая стоимость заказа

Отметим, что целесообразно разбить информацию на мельчайшие функциональные фрагменты. Например, город можно отделить от остальной части адреса, что позволит впоследствии отфильтровать людей по городу (городу) проживания. Также целесообразно не помещать одни и те же данные более чем в одну таблицу, так как это излишне усложняет базу и будет приводить к избыточности.

2. Определение сущностей и атрибутов

На этом этапе необходимо определить сущности, из которых будет состоять база данных.

Сущность — это объект в базе данных, в котором хранятся данные. Сущность может представлять собой нечто вещественное (дом, человек, предмет, место и т.д.) или абстрактное (банковская операция, отдел компании, маршрут автобуса и т.д.). В физической модели сущность называется таблицей.

Сущности состоят из атрибутов (столбцов таблицы) и записей (строк в таблице).

3. Проектирование и описание структуры базы данных

На данном этапе процесса необходимо представить базу данных в наглядной форме. Для этого необходимо разобраться в том, как именно устроена реляционная база данных.

Схожие данные в пределах базы группируются в таблицы, каждая из которых состоит из строк (или кортежей) и столбцов.

Чтобы преобразовать списки данных в таблицы, в первую очередь создайте таблицы по каждому типу сущности, (товар, продажа, клиенты и так далее).

Каждая строка таблицы называется записью. Записи содержат информацию о людях и объектах, например, о конкретном клиенте компании. В отличие от них, столбцы (которые также называют полями или атрибутами) содержат информацию одного типа, которая присутствует в каждой записи, например, адреса всех клиентов, перечисленных в таблице.

Имя	Фамилия	Дата рождения	Адрес	Город, регион	Телефон	Адрес электронной почты
Иван	Иванов	19.07.1979	Московская, д. 17, кв. 15	Минск	+375296111111	ii@gmail.com
Петр	Петров	25.11.1990	Свердлова, д. 15, кв. 100	Минск	+375296222222	pp@gmail.com
Алексей	Васин	12.05.2001	Ульяновская, д. 23, кв. 51	Минск	+375296333333	va@gmail.com

Чтобы поддерживать постоянство формата всех записей, необходимо задать каждому столбцу свой тип данных. Вот примеры распространенных типов данных:

- CHAR — заданная длина текста,
- VARCHAR — текст разной длины,
- TEXT — крупный текстовый блок,
- INT — целое число (отрицательное или положительное),
- FLOAT, DOUBLE — может содержать число с плавающей запятой.

Сами по себе таблицы со строками данных (в том виде, как показано выше) не входят в наглядный обзор базы данных (который также называется схемой «сущность-связь», или ER-схемой), а представляются в упрощенном виде — будут обозначены прямоугольниками.

Также необходимо решить, какой атрибут будет служить первичным ключом каждой таблицы. Стоит отметить, что их может быть несколько — в таком случае ключ называется составным, а может и не быть вовсе. Также необходимо напомнить, что первичный ключ — это уникальный идентификатор определенной сущности. Это означает, например, что можно найти нужного клиента в базе, даже если вам известно только этот идентификатор.

И так, на ER-схеме таблица изображается прямоугольником со сплошными границами, и состоит из трех секций. В верхней секции указывается имя таблицы. Средняя – описание полей, входящих в первичный ключ, нижняя – всех остальных полей таблицы. В случае, если таблица имеет большое количество полей, допускается показывать только наиболее существенные. Средняя и нижняя секция делятся вертикальной линией на две части: в левой указываются индексные спецификаторы, в правой – наименование полей.

Client	
	ID
	Name
	Surname
	BirthDate
	Adress
	City
	Phone
	Email

Manager	
	ID
	Name
	Surname
	Adress
	Phone
	Email
	Sum

Product	
	ID
	ProductName
	Description
	Price
	Quantity

Order	
	ID_order
	ID_manager
	ID_client
	ID_tovar
	Date
	Quantity
	Price
	Sum

Еще раз отметим, что атрибуты, выбранные в качестве первичных ключей, должны обладать уникальностью, постоянством и определенным значением (они не могут оставаться пустыми или содержать значение NULL).

Так, например, в таблицах *Client* и *Manager* даже комбинация двух полей *Name* и *Surname* не может обеспечить уникальность, поэтому в данные таблицы добавлено уникальное поле *ID*. А вот для таблицы *Orders* номера заказов (*ID_order*) идеально подходят на роль первичных ключей.

Имена полей первичного ключа подчеркиваются. Спецификатор состоит из маркера типа индекса и номера индекса данного типа в пределах таблицы.

Используются следующие маркеры: – PK- первичный ключ; – FK – внешний ключ; – U – уникальный индекс; – I – индекс. Номер индекса для первичного ключа не указывается. Если поле используется в нескольких индексах, спецификаторы для него перечисляются через запятую.

Так, приведенная ранее таблица в соответствии с указанными правилами будет выглядеть следующим образом.

Client	
PK	ID
	Name
	Surname
	BirthDate
	Adress
	City
	Phone
	Email

Полезным также является оценка потенциального размера базы данных, чтобы спрогнозировать ее производительность и необходимый размер свободного места на диске.







Создание связей между сущностями

При схематичном описании БД отображают состав и связи таблиц базы данных. Логическая модель строится в терминах информационных единиц. Основным средством разработки логической модели данных являются различные варианты ER-диаграмм (Entity-Relationship, диаграммы сущность-связь).

Преобразовав сущности (вместе с данными) в обычные таблицы, можно проанализировать имеющиеся между ними логические связи. Количество элементов, взаимодействующих между двумя связанными таблицами, называется кардинальностью. Кардинальность помогает проконтролировать, насколько эффективно были разбиты данные на таблицы.

Теоретически, все сущности (таблицы) могут поддерживать между собой связи, однако на практике данный аспект не всегда реализуется.

Статические связи между таблицами показывают сплошными линиями, конечные точки которых обозначают следующую кратность.

	Один
	Много
	Только один
	Ноль или 1
	Один или много
	Ноль или много

Далее рассмотрим более подробно примеры некоторых типов связей.

Отношение «один к одному»

Если на каждый экземпляр сущности Б приходится только один экземпляр сущности А, то считается, что между ними существует связь «один к одному» (которая часто обозначается «1:1»). На ER-диаграммах такая связь обозначается линией с небольшой чертой на каждом конце.



Связь 1:1, как правило, указывает на то, что, если нет весомых причин держать их по отдельности, данные двух таблиц лучше всего объединить в одну.

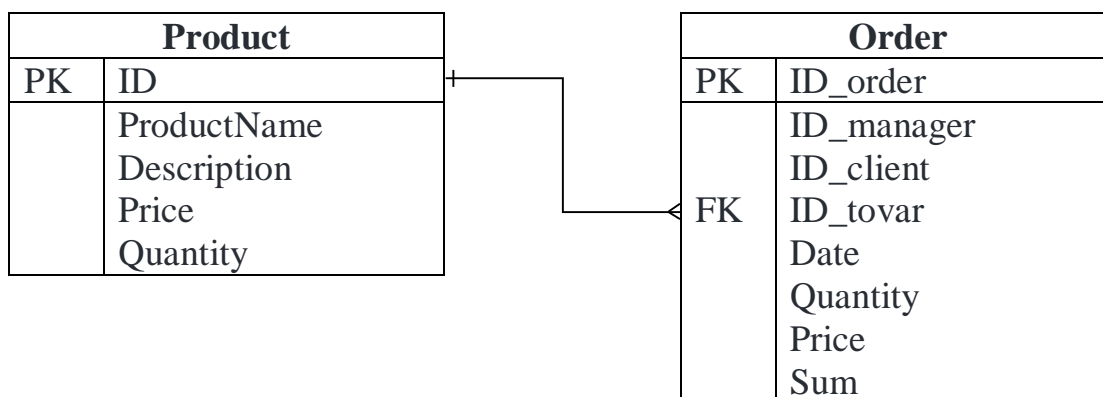
Тем не менее, в некоторых обстоятельствах использования таблиц со связями 1:1 вполне целесообразно. Если в таблицах имеются поля с необязательными данными, например, описаниями, и во многих случаях они пустуют, есть смысл перенести все описания в отдельную таблицу, что позволит избавиться от частых пробелов и повысить эффективность работы базы данных. Затем, чтобы правильно сопоставить данные, вам придется включить как минимум один идентичный столбец в каждую таблицу (для этого лучше всего выбрать первичный ключ).

Отношение «один ко многим»

Отношения такого рода возникают, когда запись одной таблицы связана с несколькими записями другой таблицы. К примеру, один и тот же клиент мог сделать несколько заказов, а менеджер — оформить несколько сделок и т.д. Связи «один ко многим» (или сокращенно «1:М») выражаются на схеме в виде нотации «вороньи лапки», как показано на примере ниже.



Чтобы применить связь 1:М при планировании базы данных, просто добавьте первичный ключ из таблицы «один» в качестве атрибута к таблице «многие». Если первичный ключ находится в другой таблице, он носит название «внешний ключ». Таблица «один» считается родительской, тогда как таблица «многие» — дочерней.



Отношение «многие ко многим»

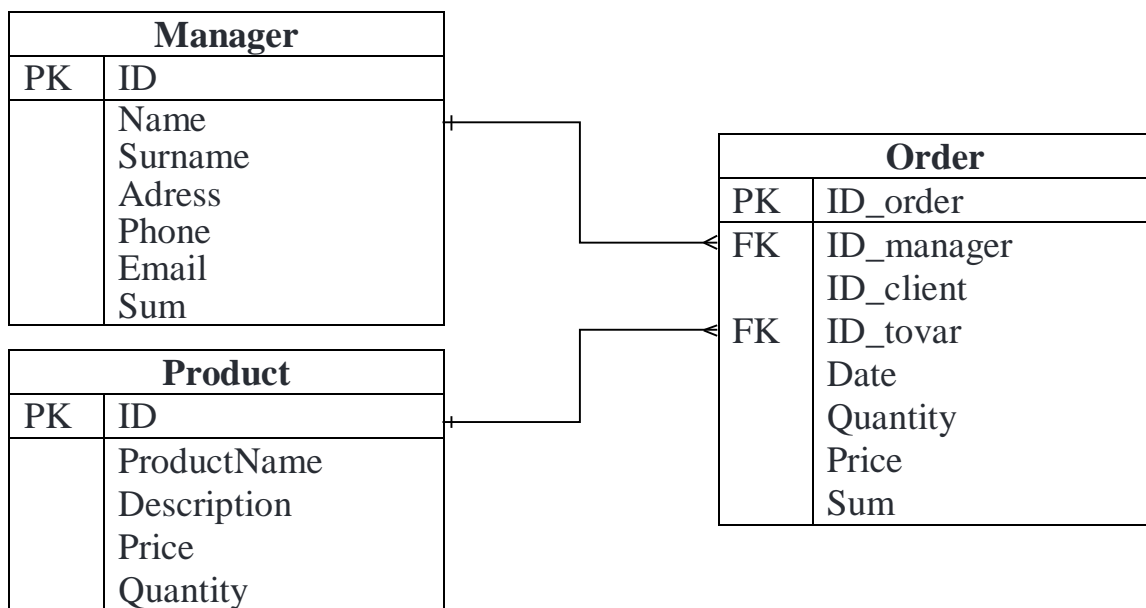
Когда несколько сущностей одной таблицы могут быть соединены с несколькими сущностями другой, считается, что между ними существует связь типа «многие ко многим» (или «М:М»). Например, такая связь существует между клиентами и менеджерами, поскольку каждый клиент может осуществить покупки через разных менеджеров, а каждый менеджер может оформлять покупки (заказы) с многими клиентами.

На ER-диаграмме этот тип связи может отображаться следующим образом:



К сожалению, напрямую реализовать такую связь в базе данных невозможно. Поэтому ее придется разбить на две связи типа «один ко многим».

Для этого понадобится создать новую сущность между двумя таблицами. Если связь М:М установлена между «Manager» и «Product», новую сущность можно назвать «Order», так как в ней будет представлено содержимое каждой продажи. С «Order» и у таблицы «Manager», и у таблицы «Product» будет установлена связь по типу 1:M.



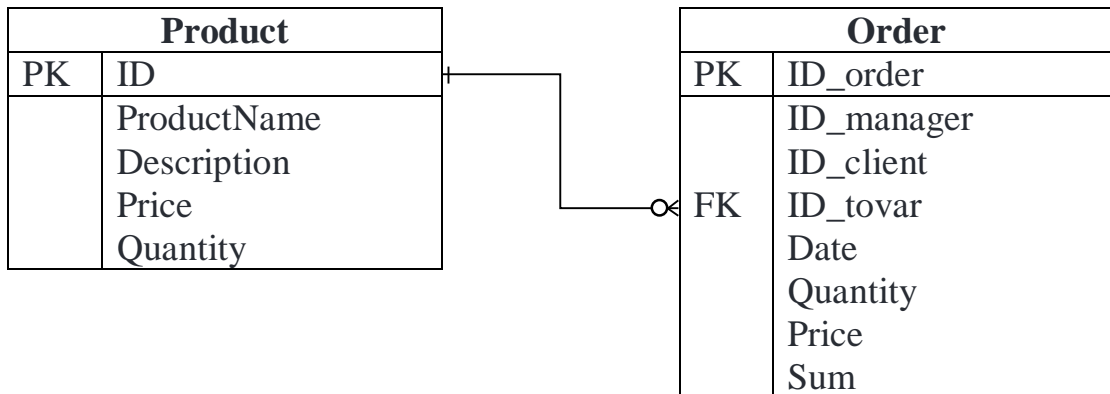
В разных моделях такие промежуточные сущности называются по-разному — «связующие таблицы», «ассоциативные сущности» или «узловые таблицы».

Каждая запись связующей таблицы соединяет между собой две разных сущности соседних таблиц (а также может содержать дополнительную информацию).

Анализ связей

Еще один подход к анализу связей заключается том, чтобы установить, какая из соединенных сущностей является обязательным условием наличия другой сущности. Необязательная сторона связи отмечается кругом на соединительной линии. Рассмотрим следующий пример: чтобы с товар был продан (оформлен факт продажи – запись в таблице «Order»), он должен быть в системе, т.е. в таблице «Товар»), но обратное не является обязательным, т.е. могут существовать товары, которые не были проданы.

В таком случае правильнее связь было бы показать следующим образом.



Возможен вариант и когда две сущности могут быть взаимозависимыми (то есть одна не может существовать без другой).

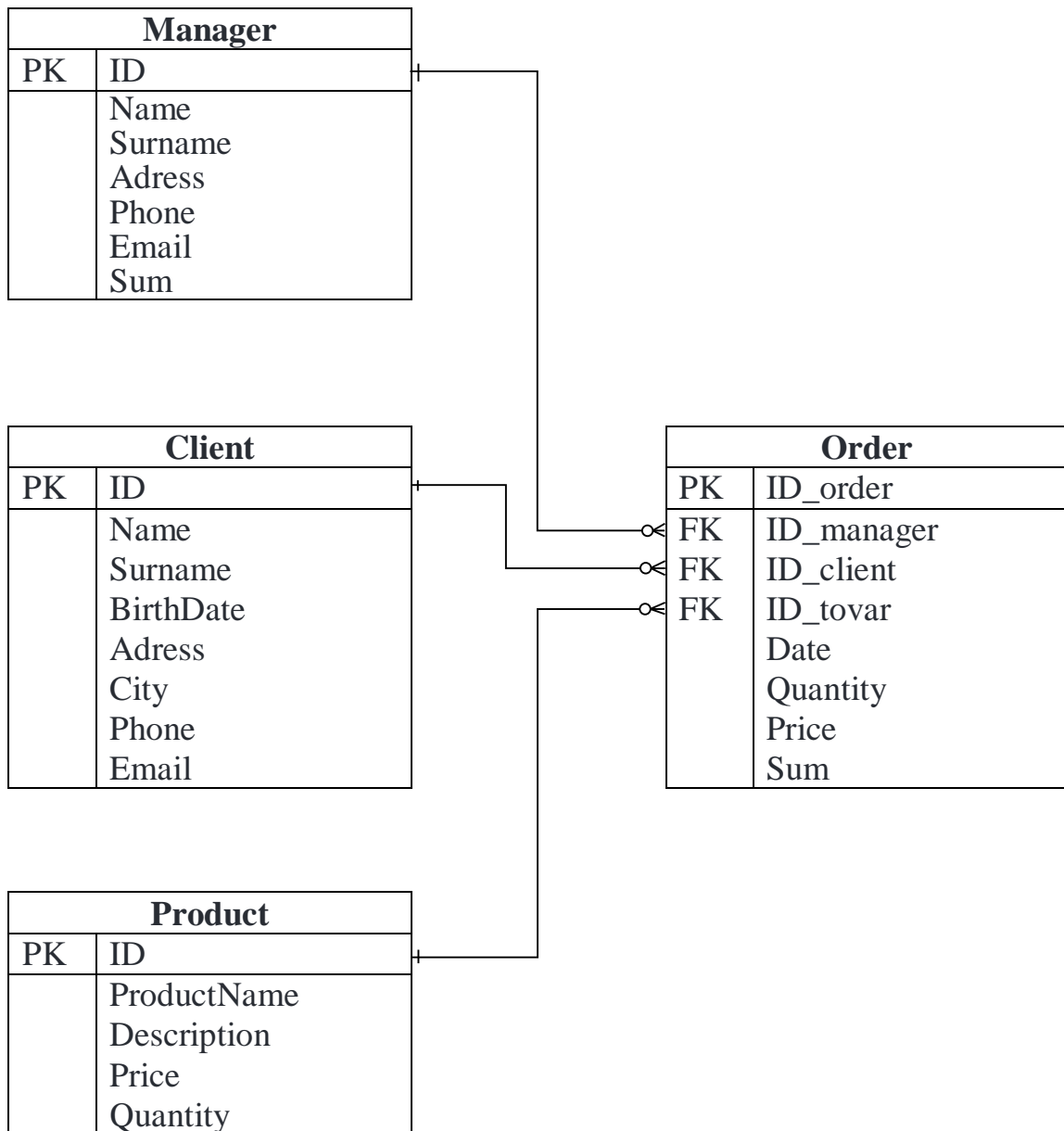
Рекурсивные связи

Иногда таблица может ссылаться на себя же. Например, в таблице сотрудников может присутствовать атрибут «менеджер», который будет отсылать нас к другому сотруднику в той же таблице. Это и есть рекурсивная связь.

Лишние связи

Связи считаются лишними, если они выражаются более одного раза. Как правило, одну из них можно удалить без потери важной информации. К примеру, если сущность «Client» связана с сущностью «Manager» не только напрямую, но и косвенно через «Order», есть смысл удалить связь между сущностями «Client» и «Manager». Обосновано это решение тем, что, например, определить с какими клиентами работает определенный менеджер можно посредством информации о заказах.

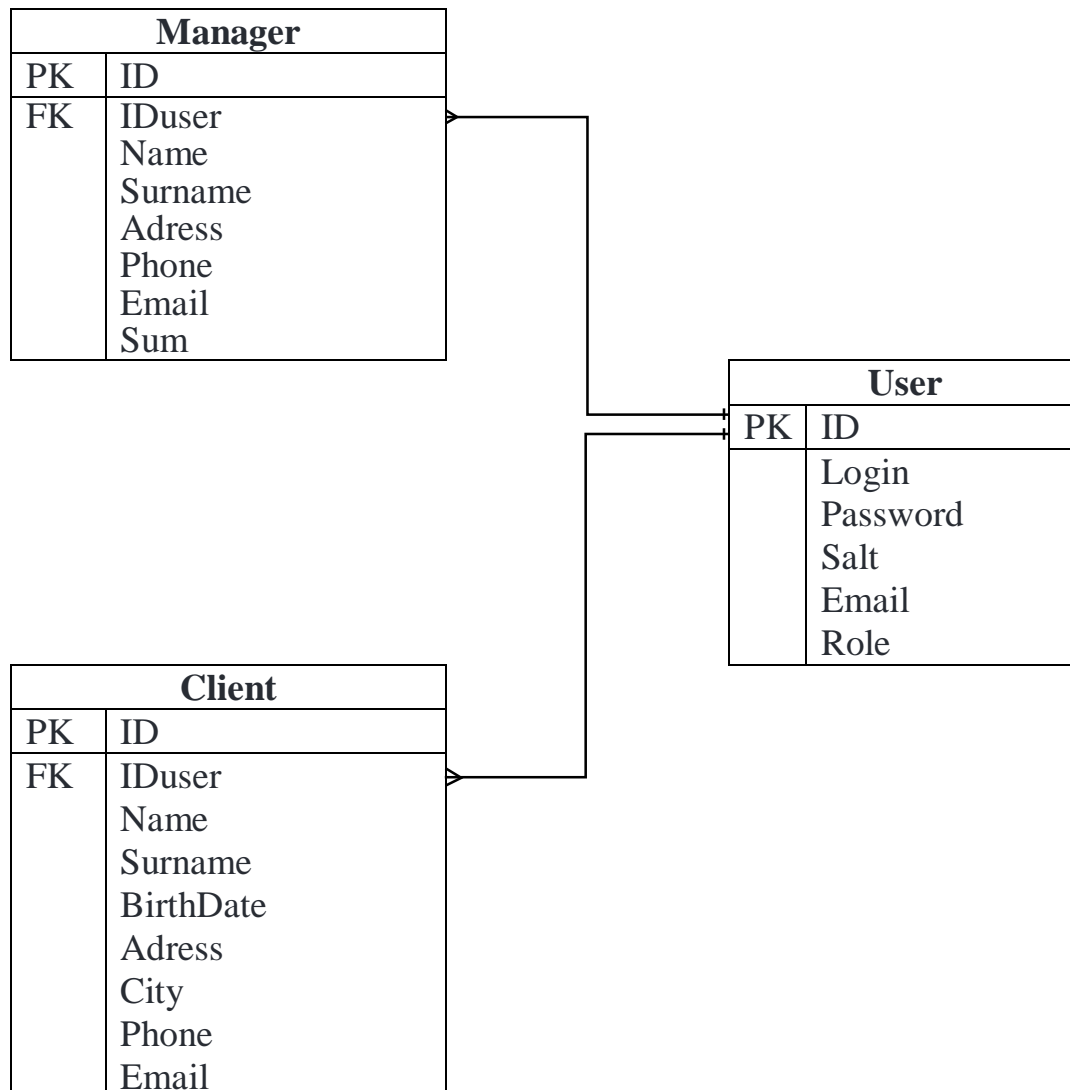
В целом схема БД для представленного примера с 4-мя таблицами – Client, Manager, Tovar, Order – может выглядеть следующим образом.



Безусловно, данная схема БД полноценной работы системы не совсем подходит. Так, например, целесообразно добавить еще таблицу с информацией о зарегистрированных пользователях (**user**) хотя бы со следующей структурой:

User	
PK	ID
	Login
	Password
	Salt
	Email
	Role

Для организации связи данной таблицы **User** с таблицами **Manager** и **Client** в последние необходимо добавить в качестве внешнего ключа (FK) поля *IDuser*, по средством которых будут установлены связи с родительской таблицей **User** типа «1:M».



Могут понадобиться и другие таблицы для хранения информации (категории товаров и т.д.), а также установление связей между ними.

Нормализация базы данных

Разработав предварительный вариант своей базы данных, можно применить к ней правила нормализации — они помогут убедиться, что таблицы составлены правильно. Данные правила необходимо воспринимать как своего рода стандарты.

Тем не менее, не все базы данных подходят для нормализации. В целом, нормализация необходима в базах данных систем оперативной обработки транзакций (OLTP), где пользователи регулярно создают, читают, обновляют и удаляют записи.

Зато базам данных систем оперативной аналитической обработки (OLAP), где приветствуются анализ и отчетность, не мешает некоторая степень денормализации, так как в их случае упор делается на скорость подсчетов. Сюда же относятся и приложения поддержки принятия решений, в которых данные требуют быстрого анализа без внесения изменений.

Каждая форма (или уровень) нормализации включает в себя правила, закрепленные за формами более низкого уровня.

Первая нормальная форма

Первая нормальная форма (или сокращенно 1НФ) говорит, что в каждой ячейке таблицы может содержаться лишь одно значение и ни в коем случае не список. Таким образом, таблица ниже не отвечает этому требованию:

ID	Description	Price
1	коричневый, желтый	15
2	красный, зеленый	13
3	синий, оранжевый	11

Отметим, что обойти эту проблему, распределив данные по дополнительным столбцам, нельзя — это тоже противоречит правилам: таблица, где содержится группа повторяющихся или тесно связанных между собой атрибутов, не отвечает требованиям первой нормальной формы.

Правильный подход — разбить таблицу на несколько таблиц или записей, пока в каждой ячейке не останется по одному значению, а в самих таблицах не будет лишних столбцов.

Если следовать первому варианту решения данной проблемы, то надо для проектируемой БД выделить таблицу **ProductDescription**, например, со следующей структурой.

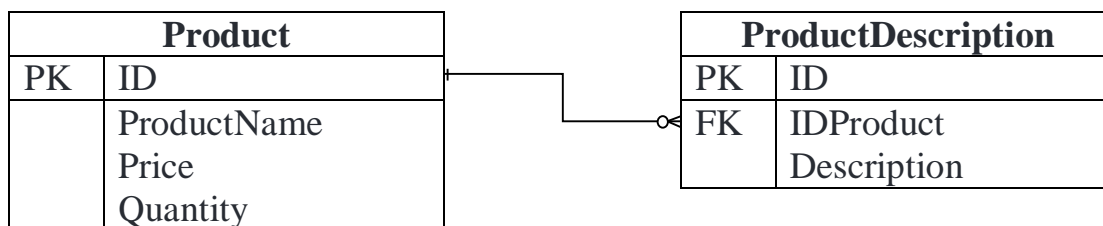
ProductDescription	
PK	ID
FK	IDProduct Description

С учетом вышеприведенной таблицы с описанием товаров, получим следующие записи в таблице **ProductDescription**.

ID	IDProduct	Description
1	1	коричневый
2	1	желтый
3	2	красный
4	2	зеленый
5	3	синий
6	3	оранжевый

При этом поле Description из таблицы Product целесообразно удалить.

Связь в таком случае будет организована между полем IDProduct таблицы **ProductDescription** и полем ID таблицы **Product**.



Приводя таблицы к первой нормальной форме, получаются «атомарные» данные, то есть данные, разбитые на мельчайшие осмысленные фрагменты.

Вторая нормальная форма

Вторая нормальная форма (или сокращенно 2НФ) требует, чтобы каждый атрибут полностью зависел от всего первичного ключа. Это означает, что каждый атрибут должен зависеть от первичного ключа напрямую, а не опосредованно через другой атрибут.

К примеру, считается, что атрибут «возраст», зависящий от «даты рождения», которая, в свою очередь, зависит от «ID» клиента, характеризуется частичной функциональной зависимостью, а сама таблица, где содержатся все эти атрибуты, не отвечает требованиям второй нормальной формы.

Более того, таблица, где содержится первичный ключ, составленный из нескольких полей, нарушает условия второй нормальной формы, если одно или несколько остальных полей не зависят от всех частей ключа.

Так, например, таблица «Order» не отвечает условиям второй нормальной формы, так как атрибут «Price» зависит от кода товара (ID_tovar), но не зависит от номера заказа (ID_order):

Order	
PK	ID_order
FK	ID_manager
FK	ID_client
FK	ID_tovar
	Date
	Quantity
	Price
	Sum

Третья нормальная форма

Третья нормальная форма (3НФ) добавляет к указанным выше требованиям еще одно: ни один не ключевой столбец не должен зависеть от

других столбцов. Если переменна значения в одном не ключевом столбце ведет к изменению другого значения, таблица не отвечает условиям третьей нормальной формы.

Это правило предотвращает хранение в таблице производных данных, например, если бы имелось в таблице **Order** поле *Discont*, содержимое которого может напрямую зависит от значения в другом поле – *Quantity*.

ID_order	Quantity	Price	Discont	Sum
1	5	2,00	1 %	9,9
2	10	0,70	5 %	6,65
3	2	1,20	0 %	2,40

Были предложены и другие формы нормализации, включая форму Бойса-Кодда, формы с четвертой по шестую и нормальную форму «домен-ключ», однако первые три формы имеют наиболее широкое распространение.

Хотя в этих формах изложены основные общие правила, степень нормализации все же зависит от контекста базы данных.

Денормализация — это умышленное изменение структуры базы, нарушающее правила нормальных форм. Обычно это делается с целью улучшения производительности базы данных. Теоретически, надо всегда стремиться к полностью нормализованной базе, однако на практике полная нормализация базы почти всегда означает падение производительности. Чрезмерная нормализация базы данных может привести к тому, что при каждом извлечении данных придется обращаться к нескольким таблицам. Обычно в запросе должны участвовать четыре таблицы или менее. Стандартными приемами денормализации являются: объединение нескольких таблиц в одну, сохранение одинаковых атрибутов в нескольких таблицах, а также хранение в таблице сводных или вычисляемых данных.

1.2 Физическая модель

Следующим шагом, после создания логической модели, является построение физической модели. Физическая модель — это практическая реализация базы данных. Физическая модель определяет все объекты, которые предстоит реализовать. При переходе от логической модели к физической сущности преобразуются в таблицы, а атрибуты в столбцы. Отношения между сущностями можно преобразовать в таблицы или оставить как внешние ключи. Первичные ключи преобразуются в ограничения первичных ключей. Возможные ключи — в ограничения уникальности.