Р. Р. Латыпова

БАЗЫ ДАННЫХ

Курс лекций

• ПРОСПЕКТ •

Рамиля Латыпова

Базы данных. Курс лекций. Учебное пособие

«Проспект»

Латыпова Р. Р.

Базы данных. Курс лекций. Учебное пособие / Р. Р. Латыпова — «Проспект»,

ISBN 978-5-39-219151-2

Рассматриваются принципы и механизмы обработки данных и знаний в информационных системах.Подробно описаны все этапы проектирования баз данных, требования к реляционным СУБД и перспективы их развития.

ISBN 978-5-39-219151-2

Содержание

Р. Р. Латыпова
Базы данных. Курс лекций
Учебное пособие

[битая ссылка] ebooks@prospekt.org

Лекция 1
Базы данных. Общие понятия

С самого начала развития вычислительной техники образовались два основных направления ее использования: выполнение численных расчетов и разработка информационных систем.

В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т. д.

Первые информационные системы появились в 1950-х гг. Они были предназначены для обработки счетов и расчета зарплаты и реализовывались на электромеханических бухгалтерских счетных машинах, что приводило к некоторому сокращению затрат и времени на подготовку бумажных документов.

В 1960-е гг. происходит изменение отношения к информационным системам. Информация стала применяться для периодической отчетности по многим параметрам. Для этого организациям требовалось компьютерное оборудование широкого назначения, способное обслуживать разнообразные функции.

В 1970-х – начале 1980-х гг. информационные системы начинают широко использоваться в качестве средства управленческого контроля, поддерживающего и ускоряющего процесс принятия решений. *Принятие решения* – акт целенаправленного воздействия на объект управления, основанный на анализе ситуации, определении цели, разработке программы достижения этой цели [5].

К концу 1980-х гг. концепция использования информационных систем вновь изменяется: они становятся стратегическим источником информации и используются на всех уровнях организации любого профиля. Информационные системы помогают организации достичь успеха в своей деятельности, создавать новые товары и услуги, находить новые рынки сбыта, партнеров, организовывать выпуск продукции по низкой цене и т. д. [4].

Структура управления любой организации традиционно делится на три уровня: операционный, функциональный и стратегический. Чем сложнее задача, тем более высокий уровень управления требуется для ее решения (рис. 1) [5].

Рис. 1. Пирамида уровней управления

Стратегическому уровню соответствует долгосрочное планирование, тактическому – среднесрочное, операционному – краткосрочное.

Простых задач, требующих немедленного (оперативного) решения, возникает значительно большее количество, чем задач тактического и тем более стратегического уровня [5].

На уровне *операционного управления* (нижнем уровне) большой объем занимают учетные задачи, такие как:

• учет количества произведенной продукции;

• учет затрат времени, сырья и материалов при выполнении отдельных производственных операций;

• учет произведенной продукции;

• бухгалтерский учет и т. д.

На операционном уровне находятся исполнители и менеджеры низшего звена (бригадиры, инженеры, ответственные исполнители, мастера, нормировщики, техники, лаборанты и т. п.).

Тактический (средний) уровень управления обеспечивает решение задач, требующих предварительного анализа информации, подготовленной на операционном уровне. На тактическом уровне большое значение приобретает такая функция управления, как *анализ*. Объем решаемых задач уменьшается, но возрастает их сложность. При этом не всегда удается выработать нужное решение оперативно, требуется дополнительное время на анализ, осмысление, сбор недостающих сведений и т. п. Управление связано с некоторой задержкой от момента поступления информации до принятия решений и их реализации, а также от момента реализации решений до получения реакции на них.

На тактическом уровне работают менеджеры среднего звена и специалисты (начальники служб, отделов, цехов, начальник смены, участка, научные сотрудники и т. п.).

Стратегический (верхний) уровень обеспечивает выработку управленческих решений, направленных на достижение долгосрочных стратегических целей организации. Поскольку результаты принимаемых решений проявляются спустя длительное время, особое значение на этом уровне имеет такая функция управления, как стратегическое планирование. Прочие функции управления на этом уровне в настоящее время разработаны недостаточно полно.

На стратегическом уровне управления находятся менеджеры высшего звена руководства организации (руководители фирм и их заместители).

При создании или классификации информационных систем возникают проблемы, связанные с формальным – математическим и алгоритмическим – описанием решаемых задач. Чем точнее математическое описание задачи, тем выше возможности компьютерной обработки данных и тем меньше степень участия человека в процессе ее решения.

Различают три типа задач, для решения которых создаются информационные системы (рис. 2):

1. Формализуемые (структурированные);
2. Неформализуемые (неструктурированные);
3. Частично формализуемые.

Рис. 2. Классификация задач в информационной системе

Формализуемая задача – это задача, где известны все ее элементы и взаимосвязи между ними. Содержание такой задачи можно выразить в форме математической модели, имеющей точный алгоритм решения. Подобные задачи обычно приходится решать многократно, и они носят рутинный характер. Целью использования информационной системы для решения структурированных задач является полная автоматизация их решения, т. е. сведение роли человека к нулю.

Например, задача расчета заработной платы. Это структурированная задача, где полностью известен алгоритм решения. Рутинный характер этой задачи определяется тем, что расчеты всех начислений и отчислений просты, но объем их велик, поскольку они должны многократно повторяться ежемесячно для всех категорий работающих.

Классические *базы данных* ориентированы на решение формализуемых задач.

Неформализуемая задача – та, в которой невозможно выделить элементы и установить между ними связи. При решении неформализуемых задач возникают трудности из-за невозможности создания точного математического описания. Здесь большое значение могут иметь эвристические соображения человека на основе опыта и косвенной информации из разных источников.

На практике о большинстве задач можно сказать, что известна лишь часть их элементов и связей между ними. Такие задачи называются *частично формализуемыми*. В этих условиях можно создать информационную систему. Получаемая в ней информация анализируется человеком, который будет играть определяющую роль. Такие информационные системы являются автоматизированными, так как в их функционировании принимает участие человек [5].

Для решения неформализуемых и частично формализуемых задач разрабатываются *экспертные системы,* или *системы обработки знаний* [14].

Лекция 2
История развития баз данных

В настоящее время базы данных (БД) – наиболее массовая область информационных технологий.

Всякая программа для ЭВМ является моделью некоторой предметной области. База данных – это также модель взаимосвязей между объектами реального мира и описание этих объектов.

В настоящее время наиболее распространены *реляционные* БД, исторически им предшествовали *иерархические* и *сетевые* БД.

Первые БД, появившиеся в 1960-е гг., были предназначены для планирования выпуска продукции. Очень часто возникала потребность определить, сколько требуется комплектующих для выпуска того или иного вида продукции. Таким задачам соответствует древовидная (иерархическая) структура.

Например, при выпуске автомобиля получается структура, показанная на рис. 3.

Каждому прямоугольнику на рис. 3 соответствует запись в БД.

Рис. 3. Пример иерархической структуры

Между записями существуют отношения «предок – потомок». Для получения доступа к данным в иерархической БД можно указать номер записи, а также выполнить ряд действий:

1. Найти дерево по признаку;
2. Перейти «вниз» к первому потомку;
3. Перейти «в сторону» к следующему потомку;
4. Перейти «вверх» к предку;
5. Вставлять и удалять записи.

Таким образом, по записям можно перемещаться, переходя вниз, вверх или в сторону.

Иерархические БД имеют следующие достоинства:

1. Структура БД проста для понимания;

2. Отношения «предок – потомок» позволяют моделировать высказывания типа «*A* – часть *B*» или «*A* владеет *B*»;

3. Записи можно оптимально размещать на носителе информации, т. е. предки возле потомков, тем самым сокращается время доступа.

Самая известная из таких БД – это IMS (англ.: Information management system – система управления информацией) фирмы IBM (1968 г.). Эта система управления базами данных (СУБД) все еще активно эксплуатируется на больших ЭВМ.

Сетевые БД (здесь не имеются в виду сети ЭВМ) позволяют описать те случаи, когда одна запись может участвовать в нескольких отношениях «предок – потомок», т. е. иметь несколько предков (рис. 4). Такие отношения в сетевой модели называют множествами.

Рис. 4. Пример сетевой организации данных

Официальный стандарт сетевых БД был предложен в 1971 г., он получил название CODASYL (англ. COnference on DAta SYstems Language – Конференция по языкам систем обработки данных).

Доступ к данным в сетевой модели напоминает доступ к данным в иерархической модели. Программа может выполнять следующие действия:

1. Найти запись по ее номеру (признаку);
2. Перейти к первому потомку в конкретном множестве;
3. Перейти «в сторону» от потомка к потомку в конкретном множестве;
4. Перейти «вверх» от потомка к предку в другом множестве;
5. Вставлять и удалять записи.

Сетевые БД отличаются большей гибкостью, так как позволяют описать более сложные структуры данных.

Но иерархические и сетевые БД имеют общий недостаток: структура данных описывается жестко на этапе проектирования. При перестройке структуры нужно перестраивать всю БД.

Кроме того, иерархические и сетевые БД требовали участия специалиста – программиста для реализации запросов. Это вызывало задержки при эксплуатации БД.

Поэтому такие БД сейчас имеют меньшее распространение, чем реляционные БД. В переводе с англ. relation означает «отношение». Математический аппарат, который используется в таких БД, позволяет описывать таблицы и операции над ними.

Теоретический фундамент реляционных БД заложил Э. Кодд, разработавший в 70-е гг. математический аппарат теории отношений. Реляционная модель является теорией, но фактически ни одна из современных БД не придерживается на все 100 % положений этой теории. То есть пользователь должен учитывать теоретические рекомендации, но имеет возможности для их нарушения.

При математическом описании понятию *таблицы* соответствует понятие *отношение*, столбцу – *атрибут,* и строке – *кортеж*.

При практической разработке строки называют *записями*, а столбцы – *полями*. То есть запись – это набор полей, содержащих связанную информацию. Поле – это элемент данных в БД. Поле должно иметь имя и тип.

База данных – набор связанных таблиц, обычно идентифицируемых с помощью каталога, содержащего эти таблицы, или с помощью псевдонима, дающего имя БД.

По отношению к пользователю реляционные БД поддерживают два основных принципа:

1. Данные для пользователя представляются в виде таблиц;

2. Пользователь имеет в своем распоряжении операторы, позволяющие получить новые таблицы из старых.

При построении реляционных БД используется несколько простых правил:

1. Все значения данных состоят из простых типов данных. Отсутствуют сложные типы, такие как массивы, указатели, векторы и т. д.;

2. Все данные отображаются в виде двумерных таблиц (отношений). Каждая таблица содержит некоторое число строк (кортежей) и один или несколько столбцов (атрибутов);

3. После ввода данных можно сравнивать значения в различных столбцах и соотносить строки (в том числе и для разных таблиц);

4. Все операции определяются только логикой, а не положением строки в таблице;

5. Поскольку определить строку по ее положению в таблице нельзя, бывает необходимо иметь специальное поле в каждой строке – *первичный ключ*;

6. Каждое значение в столбце должно быть атомарной величиной, т. е. содержать только одно значение.

Таким образом, *таблица* – это основа реляционной БД. Это логическая структура, физическое представление может быть каким угодно.

Кроме того, реляционные БД используют еще ряд объектов. К ним относятся:

1. Формы – позволяют ограничить объем информации, отображаемой на экране, и представить ее в оптимальном виде. Формы используются для просмотра данных и ввода их в таблицы. С помощью *мастера* форм можно легко создать форму, поместив в нее поля исходной таблицы в соответствии с одним из шаблонов. С помощью *конструктора* можно создать форму любой степени сложности;

2. Отчеты – используются для отображения информации из БД. Они также могут строиться с использованием мастера или конструктора. В отчете можно сгруппировать поля исходной таблицы, добавить вычисляемые поля, сделать нужное оформление;

3. Формы и отчеты иногда называют конструкторскими объектами. Они могут включать в себя элементы управления, такие как надписи, прямоугольники, линии, рисунки, выключатели, флажки и т. п.;

4. Запросы – это средства извлечения информации из БД. Данные могут извлекаться из нескольких таблиц одновременно, т. е. связи между таблицами могут устанавливаться в момент исполнения запроса. Это упрощает разработку БД;

5. Макросы – предназначены для выполнения часто исполняемых операций. Каждый макрос содержит одну или несколько макрокоманд. Каждая макрокоманда выполняет определенное действие (открытие формы, печать отчета и т. п.);

6. CASE-средства (Computer-Aided Software Engineering) – это программы для разработки структуры БД в виде диаграмм и автоматической генерации БД на их основе.

Для построения запросов к реляционным БД был разработан язык SQL (англ.: Structured Query Language – язык структурированных запросов). SQL получил характер промышленного стандарта. Его поддерживают все современные БД. При переходе с одной БД на другую разработчик имеет дело с одним и тем же языком SQL. Это позволяет не вникать в детали низкоуровневого доступа к данным, а учитывать только логическое описание БД. SQL является языком более высокого уровня, чем обычные языки программирования.

Операторы SQL выполняются на уровне множеств. Этот язык является декларативным, т. е. пользователь описывает, что ему нужно получить, но описывает алгоритм, при помощи

которого это можно сделать. Процедура получения решения строится без участия пользователя [7].

Лекция 3
Локальные и серверные базы данных

В задачах обработки информации, основанных на системах баз данных, существуют два варианта расположения данных: *локальный* и *удаленный*. Соответственно, существуют «локальные», или «персональные», БД, а также промышленные, или серверные, БД.

К локальным БД относятся Paradox, dBase, Access, FoxPro и др. БД Access занимает особое положение, потому что входит в состав распространенного пакета Microsoft Office. Локальные данные, как правило, располагаются на жестком диске компьютера, на котором работает пользователь, и находятся в монопольном ведении этого пользователя. Пользователь при этом работает автономно, не завися от других пользователей и никоим образом не влияя на их работу.

К серверным БД относятся Oracle, Sybase, SQL Server и др. Удаленные данные располагаются вне компьютера пользователя – на файловом сервере сети или на специально выделенном для этих целей компьютере.

Всего можно выделить три архитектуры серверных БД:

1. Архитектура «файл-сервер»;
2. Архитектура «клиент-сервер»;
3. Многозвенная архитектура.

При работе с локальными БД режим однопользовательский.

В стандартной *файл-серверной* архитектуре данные, располагаясь на файл-сервере, являются, по сути, пассивным источником. На компьютере пользователя запускается копия приложения. При этом, поскольку обработка данных осуществляется на компьютере пользователя, по сети передается вся необходимая для этой обработки информация, хотя интересующий пользователя объем данных может быть намного меньше пересылаемого. Например, если пользователя интересуют все работники какого-либо предприятия, участвующие в конкретном проекте, его приложение получит сначала информацию обо всех работниках и обо всех проектах из базы данных и только после этого произведет требуемую выборку.

Кроме того, вся ответственность за получение, обработку, а также за поддержание целостности БД лежит на пользователе. Данные, с которыми работает пользователь, время от времени обновляются из реальной БД, расположенной на файл-сервере. При этом изменения, которые вносит один пользователь, могут быть на протяжении какого-то времени неизвестны другим пользователям. Поэтому возникает проблема блокировки одновременного доступа к данным разных пользователей [12].

Исторически на персональных компьютерах использовался именно этот подход – как более простой в освоении. Однако большой объем передаваемых по сети данных быстро «забивает» сеть уже при небольшом числе пользователей, существенно ограничивая возможности роста. Этот основной и самый существенный недостаток заставил искать способы уменьшения нагрузки на сеть.

В архитектуре *клиент-сервер* для обработки данных выделяется специальное ядро – так называемый *сервер баз данных*, который принимает на себя функции обработки запросов пользователей, именуемых теперь *клиентами*. Сервер баз данных представляет собой программу, выполняющуюся, как правило, на мощном компьютере. Приложения-клиенты посылают с рабочих станций запросы на выборку (вставку, обновление, удаление) данных. При этом сервер выполняет всю «грязную» работу по отбору данных, отправляя клиенту только требуемую «выжимку». Если приведенный выше пример перестроить с учетом клиент-серверной

архитектуры, то приложение-клиент получит с сервера в качестве результата список только тех работников, которые участвуют в заданном проекте.

Такой подход обеспечивает решение трех важных задач:

1. Уменьшение нагрузки на сеть;
2. Уменьшение требований к компьютерам-клиентам;
3. Повышение надежности и сохранение логической целостности базы данных.

Действительно, теперь сервер БД (в случае реляционных баз данных называемый SQL-сервером) возвращает клиентскому приложению только «выжимку» того, что он просмотрел в базе, а она («выжимка») в общем случае действительно составляет малую часть от общего объема. Поэтому в сети не наблюдается резкого увеличения нагрузки при увеличении числа клиентов. Клиентские же приложения могут выполняться на менее мощных (по сравнению с сервером) компьютерах благодаря тому, что им практически не требуется выполнять никакой дополнительной обработки полученных от сервера результатов запроса (хотя, конечно, это не запрещено). Побочным эффектом уменьшения нагрузки на сеть является повышение скорости выполнения приложений клиентов. Кроме того, система легче масштабируется – легче и дешевле заменить один сервер на более мощный, чем десятки рабочих станций.

Но наиболее важным результатом перехода в архитектуру «клиент-сервер» является гарантированное сохранение логической целостности базы данных, т. е. система становится более устойчивой и более защищенной. Достигается это благодаря возможности переложения заботы о сохранении целостности базы на сервер. Для этого «хорошие» серверы обладают большим набором встроенных механизмов, защищающих систему от неверных действий клиентов. Среди этих механизмов можно назвать такие, как ограничения целостности, декларативная ссылочная целостность, триггеры, виртуальные таблицы (представления), авторизация пользователей и др.

Лекция 4
Основные понятия реляционных баз данных

При работе с таблицами часто используют два представления: собственно таблицу и структуру таблицы. Пример приведен на рис. 5.

Таблица «Студенты»

Номер
Фамилия
Имя
Рост
Вес

Структура таблицы «Студенты»

Поле
Тип поля

Номер
Счетчик

Фамилия
Текстовый

Имя
Текстовый

Рост
Числовой

Вес
Числовой

Рис. 5. Пример описания таблицы и ее структуры

Таблица может иметь *первичный ключ*, под которым понимается поле или набор полей, однозначно идентифицирующих запись.

В таблице не должно быть записей с одним и тем же значением первичного ключа.

Например, если рассматривается таблица «Студенты», то в качестве первичного ключа нельзя использовать фамилию, имя или дату рождения, поскольку эта информация не уникальна.

В общем случае в качестве первичного ключа выгоднее использовать семантически незначащее (не несущее смысловой нагрузки) поле (счетчик), с помощью которого каждая запись получает уникальный номер.

Первичный ключ является разновидностью более общего понятия *потенциального ключа*, т. е. ключа, который может быть выбран в качестве первичного.

Между двумя и более таблицами БД могут существовать *отношения подчиненности*. Это означает, что для каждой записи главной таблицы (родительской, или мастер-таблицы (англ.:

master)) может существовать одна или несколько записей в подчиненной (или детальной (англ.: detail)) таблицы.

Связывание таблиц выполняется для устранения избыточности информации.

Существуют три разновидности связей между таблицами:

1. Связь «один-ко-многим» (или «многие-к-одному»);
2. Связь «один-к-одному»;
3. Связь «многие-ко-многим».

Связываемые поля не обязательно должны иметь одинаковые имена, но они должны иметь одинаковые типы данных.

Отношение «один-ко-многим» является самым распространенным, оно моделирует иерархию данных.

Рассмотрим пример, когда одной записи в родительской таблице соответствует несколько записей в дочерней таблице (рис. 6). В этом примере одной записи в родительской таблице «Товары» соответствует несколько записей в дочерней таблице «Отпуск товаров».

Отношение «один-к-одному» применяется тогда, когда стремятся сократить объем информации в одной таблице или защитить часть информации от доступа. Но здесь приходится выполнять больше операций чтения при извлечении связанных данных. В этом случае одной записи в главной таблице соответствует одна запись в подчиненной таблице (рис. 7).

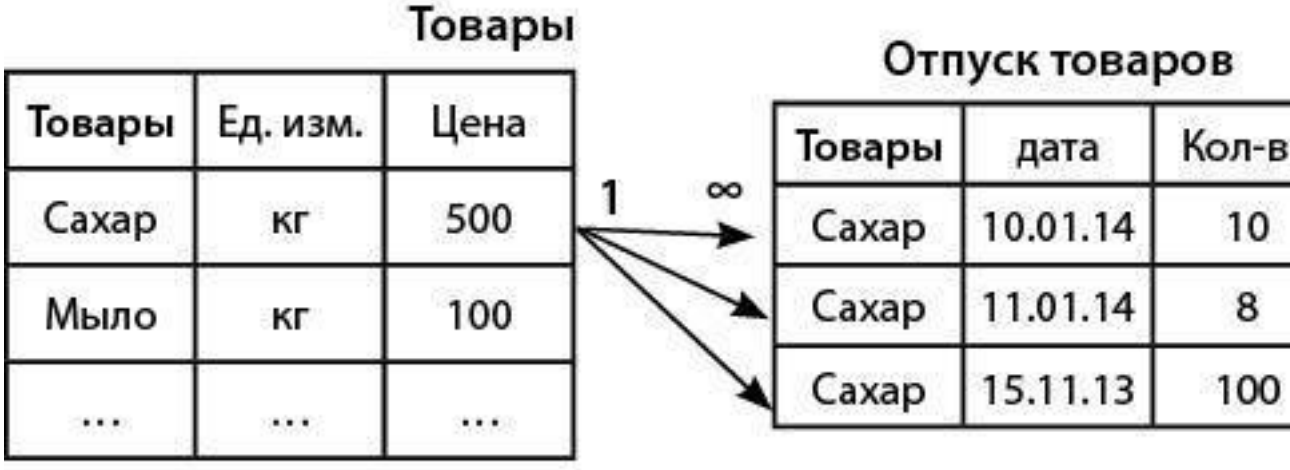

Рис. 6. Связь «один-ко-многим»

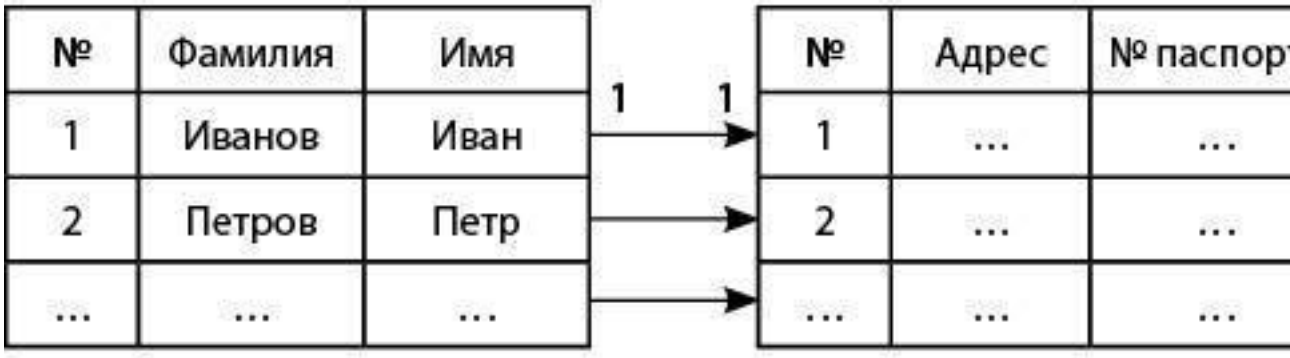

Рис. 7. Связь «один-к-одному»

Такие связи также могут быть жесткими и нежесткими.

Рассмотрим отношение «многие-ко-многим». В этом случае возможны два варианта:
записи в родительской таблице соответствует более одной записи в дочерней таблице;
записи в дочерней таблице соответствует более одной записи в родительской таблице.
Пример приведен на рис. 8.

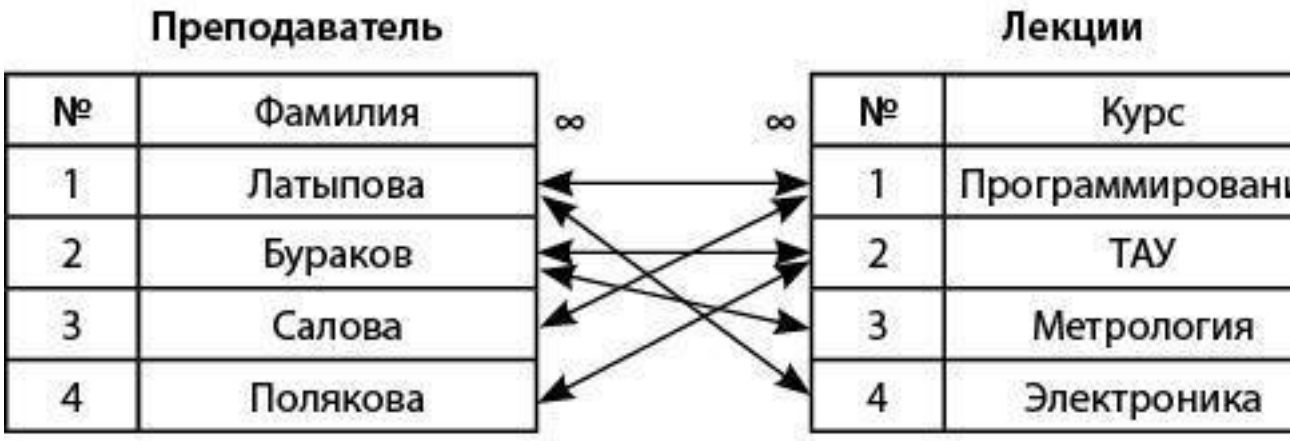

Рис. 8. Связь «многие-ко-многим»

Здесь имеется в виду, что один преподаватель читает разные курсы, а один и тот же курс могут читать разные преподаватели.

Любая связь «многие-ко-многим» может быть заменена на одну или более связей «один-ко-многим». Для этого нужно ввести промежуточную таблицу (рис. 9).

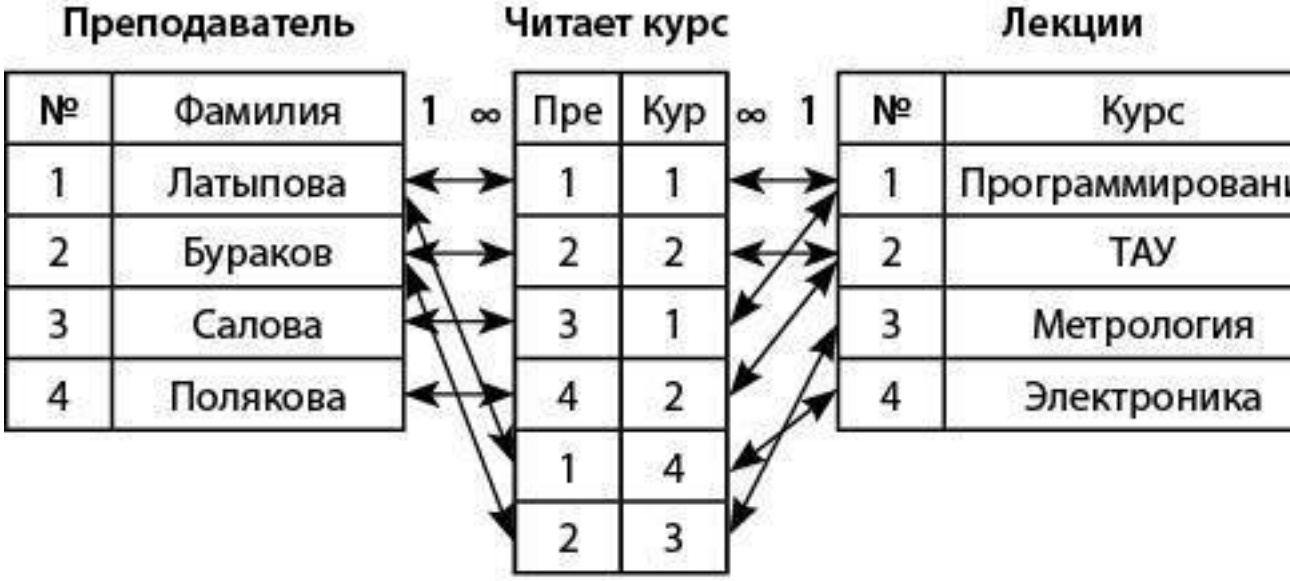

Рис. 9. Преобразование связи «многие-ко-многим»

При связывании таблиц необходимо обеспечивать целостность данных, которая может быть нарушена при изменении полей связи.

Рассмотрим наиболее часто встречающуюся связь «один-ко-многим». Пример приведен на рис. 10.

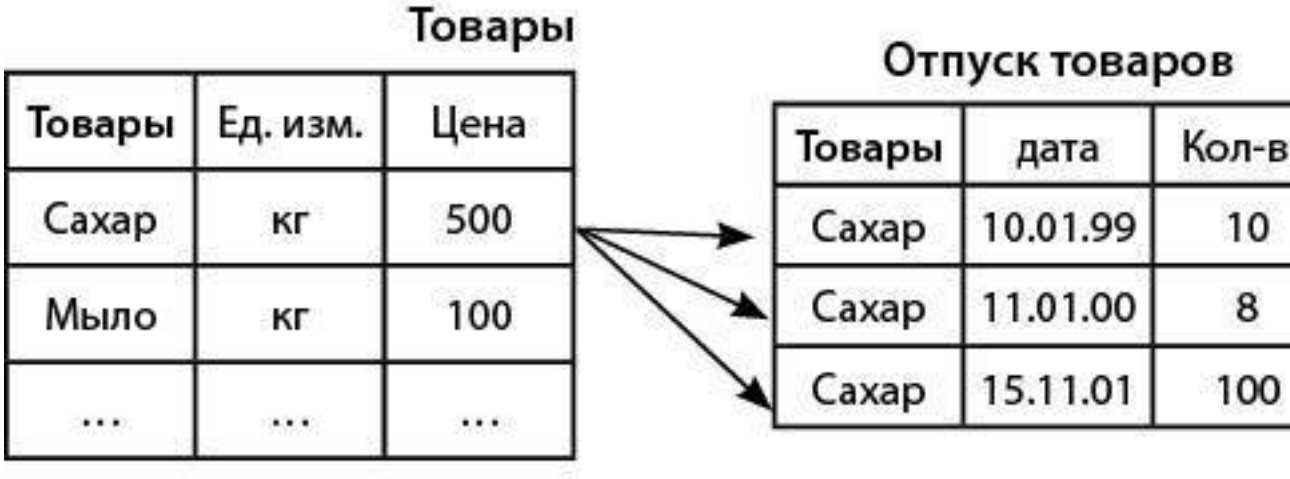

Рис. 10. Пример связывания таблиц

Эти две таблицы связаны по общему полю «Товар». Таблица «Товары» является главной, таблица «Отпуск товаров» – подчиненной. Потеря связей возможна в двух случаях:

1. Изменяется значение в поле связей главной таблицы без изменения значений полей связи в соответствующих записях дочерней таблицы. Например, если вместо товара «Сахар» в таблице «Товары» написать «Песок», то все записи в дочерней таблице для «Сахар» потеряют связь и не будут иметь единицы измерения и цены;

2. Изменяется значение поля связи одной из записей дочерней таблицы без изменения значения поля связи в соответствующих записях главной таблицы. Например, если вместо товара «Сахар» в одной из записей таблицы «Отпуск товаров» написать «Песок», то будут недостоверными записи об отпуске товара «Сахар». Запись в дочерней таблице для «Песок» не будет иметь связи и, следовательно, единицы измерения и цены.

Таким образом, в обоих случаях возникает нарушение целостности БД.

Действия, нарушающие ссылочную целостность БД, должны блокироваться.

Для сохранения ссылочной целостности может использоваться также механизм каскадных изменений:

1. Синхронные изменения поля связи в дочерней таблице при внесении изменений в поле связи главной таблицы;

2. Синхронные удаления поля связи в дочерней таблице при удалении поля связи главной таблицы.

Разрешение или запрещение каскадных изменений реализуется при описании связей между таблицами БД.

Обычно в СУБД для реализации ссылочной целостности в дочерней таблице создают *внешний ключ*, ссылающийся на родительскую таблицу, и указывают вид каскадных воздействий.

Внешний ключ создается в дочерней таблице. В него входят поля связи дочерней таблицы. Для связей типа «один-ко-многим» внешний ключ должен совпадать по составу полей с первичным ключом главной таблицы или с частью первичного ключа (в этом случае нормализация таблиц БД выполнена не полностью).

При определении первичного и внешнего ключей СУБД автоматически строит *индексы*. Индекс, соответствующий внешнему ключу, строится для обеспечения связей родительской и дочерней таблиц [2].

Индексы обеспечивают механизм быстрого доступа к данным в таблицах. Индексы хранят значения индексных полей (по которым построен индекс) и указатель на запись в таблице.

Использование индексов позволяет использовать не просто последовательный, а индексно-последовательный доступ.

При последовательном доступе просматриваются все записи таблицы – от первой до последней, что неэффективно.

При индексно-последовательном доступе указатель в индексе устанавливается на первую строку, соответствующую условиям запроса (или его части), и считывается запись из таблицы по хранящемуся на нее в индексе указателю. Далее последовательно считываются остальные записи, удовлетворяющие условиям запроса.

Таким образом, во втором случае поиск ведется по индексу, а не по самой таблице. Таблица может быть неупорядоченна, а небольшой по объему индексный файл может быть легко отсортирован.

Например, рассмотрим табл. 1.

Таблица 1

Отпуск товара

Номер	Дата	Товар	Количество
1	06.01.14	Спички	2
2	02.01.14	Мыло	100
3	03.01.14	Мука	5000
4	08.01.14	Спички	10

Структура индексов по каждому из четырех полей показана в табл. 2.

Таблица 2

Индексированные поля таблицы

По дате прихода	По наименованию	По количеству
Дата	Товар	Количество
Номер записи	Номер записи	Номер записи

Номер записи
Количество
Номер записи

02.01.14
2
Мука
3
5000
3

03.01.14
3
Мыло
2
100
2

06.01.14
1
Спички
1
10
4

08.01.14
4
Спички
4
2
1

Если несколько товаров имеет одно и то же наименование, то достаточно найти в индексе, построенном по столбцу «Наименование», первую запись, а затем повторить чтение подряд для всех товаров с этим наименованием. То же самое касается даты и т. д.

Индексы наиболее выгодны для статичных таблиц, по которым часто выполняются запросы.

Лекция 5
Нормализация таблиц БД

При создании БД необходимо выполнить анализ предметной области, для которой разрабатывается БД. Процесс разработки БД является циклическим, т. е. на разных этапах происходят возвраты на более ранние этапы с целью коррекции. Субъективные взгляды разработчика всегда могут найти отражение в БД, но есть ряд объективных требований, соблюдение которых всегда может принести пользу. К таким требованиям относится нормализация БД. Процесс нормализации позволяет устранить избыточность данных и ускорить доступ к ним.

В основе нормализации лежит одна основная идея: поля таблицы должны зависеть только от ключа таблицы и ни от чего другого. Если это не так, то следует разбить таблицу на отдельные таблицы [1].

Общие требования нормализации формулируются в виде пяти нормальных форм (НФ), к которым последовательно приводятся таблицы БД. На практике наиболее часто применяются только первые три НФ [10].

Рассмотрим первую нормальную форму (1НФ).

Таблица в 1НФ должна удовлетворять следующим требованиям:

1. В таблице не должно быть повторяющихся записей;

2. Каждое поле таблицы должно быть неделимым (атомарным), т. е. на пересечении строки и столбца должен быть атомарный объект;

3. В таблице должны отсутствовать повторяющиеся группы полей.

Рассмотрим пример нормализации таблицы «Продажи», в которой содержится 21 поле (табл. 3).

Таблица 3

Продажи

Номер	**Поле**	**Тип поля**
1	Фамилия	Текст
2	Имя	Текст
3	Отчество	Текст
4	Телефон	Текст
5		

Факс
Текст

6
Индекс
Текст

7
Страна
Текст

8
Город
Текст

9
Адрес
Текст

10
Название предприятия
Текст

11
Руководитель предприятия
Текст

12
Web-сайт предприятия
Текст

13
E-mail предприятия
Текст

14
Код товара
Числовой

15
Дата заказа
Дата/время

16
Заказано
Числовой

17
Дата продажи

Дата/время

18
Продано
Числовой

19
Цена
Денежный

20
Категория товара
Числовой

21
Наименование товара
Текстовый

В табл. 3 каждое поле неделимое, и никакое из полей не является уникальным.

Таблица с такой структурой может иметь повторяющиеся группы полей, в которых будут записаны данные об одном и том же покупателе (поля с 1-го по 13-е). Чтобы привести таблицу к 1НФ, она разбивается на две таблицы: «Клиенты» и «Заказы», находящиеся в отношении «один-ко-многим».

Поскольку ни одно из полей исходной таблицы не было уникальным, здесь в качестве первичного ключа таблицы «Клиенты» лучше ввести новое поле – «Код клиента». Это поле будет внешним ключом в таблице «Заказы» (рис. 11).

Рис. 11. Разбиение со связью «один-ко-многим»

В таблице «Заказы» ни одно из полей не является уникальным, поэтому в качестве первичного ключа можно добавить поле «Код заказа» или использовать комбинацию трех полей – «Код клиента», «Код заказа» и «Дата заказа» – в качестве составного ключа (если один клиент делает один заказ в день). Рассмотрим второй вариант – таблицу с составным первичным ключом. Такая таблица должна удовлетворять требованиям 2-й нормальной формы (2НФ).

Таблица находится во 2НФ, если удовлетворены два условия:

1. Таблица удовлетворяет условиям 1НФ;

2. Любое неключевое поле однозначно идентифицируется полным набором ключевых полей.

Очевидно, что в таблице «Заказы» второе условие не выполняется, поскольку поля «Категория» и «Наименование товара» зависят только от поля «Код товара». Чтобы привести таблицу к 2НФ, нужно выделить эти поля в отдельную таблицу с наименованием «Товар» (рис. 12).

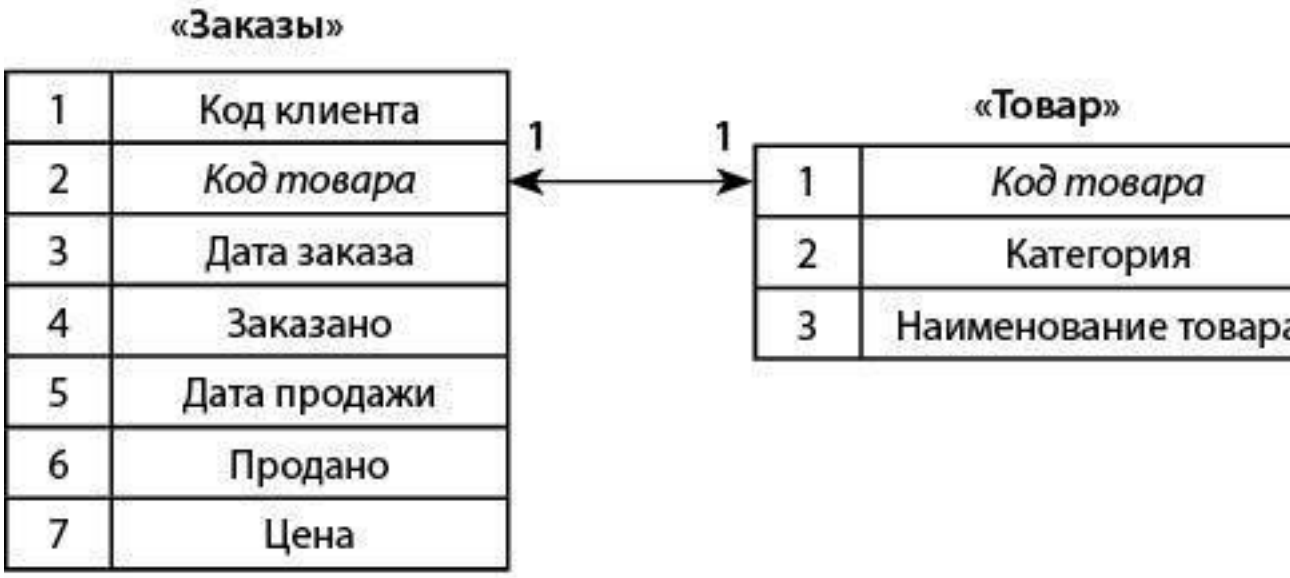

Рис. 12. Разбиение со связью «один-к-одному»

Рассмотрим далее таблицу «Клиенты». Здесь нет составного ключа, поэтому требования 2НФ выполнены автоматически и требуется рассматривать третью нормальную форму (3НФ).

Таблица находится в 3НФ, если она удовлетворяет условиям 2НФ и ни одно из неключевых полей таблицы не идентифицируется с помощью другого неключевого поля.

Иначе говоря, все неключевые поля должны быть независимы. Если какие-то поля зависят не от ключа, а от другого неключевого поля, то такие поля должны быть выделены в отдельную таблицу.

В таблице «Клиенты» неключевые поля «Руководитель фирмы», «Web-сайт фирмы» и «E-mail фирмы» определяются неключевым полем «Название фирмы», поэтому необходимо создать новую таблицу с названием «Фирма» (рис. 13).

Таким образом, в рассмотренном примере из одной исходной таблицы в соответствии с требованиями нормализации было получено четыре таблицы.

В целом нормализация не только предоставляет преимущества, но и несет некоторые недостатки.

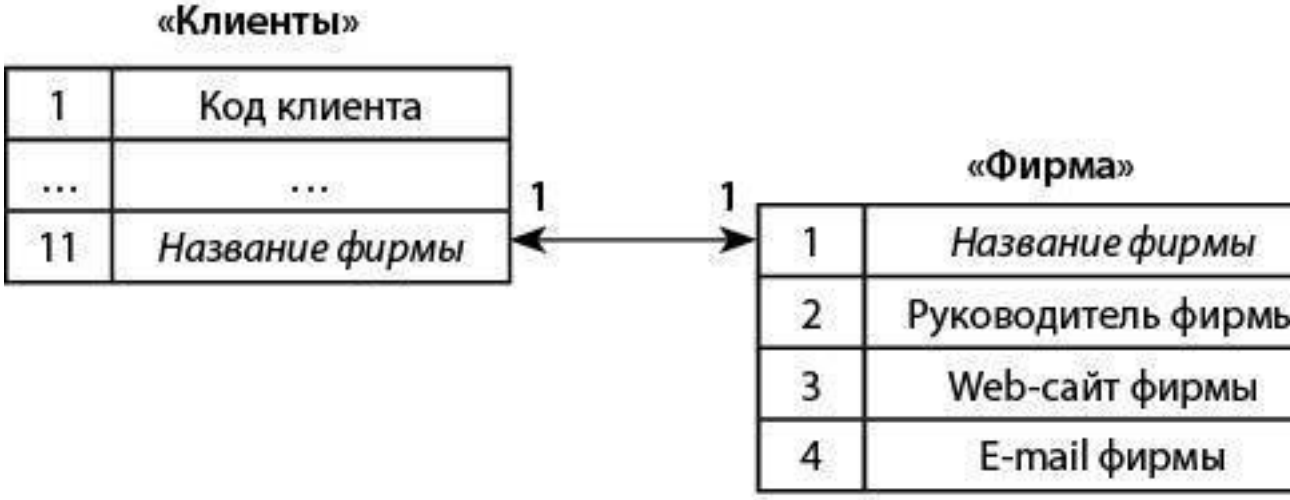

Рис. 13. Устранение зависимости неключевых полей

Главное достоинство состоит в том, что после нормализации в таблицах нет избыточных данных, поэтому экономится память ЭВМ (хотя появляются поля связи, присутствующие одновременно у главной и подчиненной таблиц).

В качестве недостатков могут быть названы следующие:

1. Чем шире предметная область, тем больше получается набор таблиц после нормализации. Для крупного предприятия БД может содержать сотни взаимосвязанных таблиц, что превышает возможности человеческого восприятия. Поэтому функционирование БД может стать труднообъяснимым;

2. При считывании связанных данных необходимо объединять записи в связанных таблицах, что приводит к необходимости выполнения поисковых операций. В сложных БД это может вызывать временные издержки.

Таким образом, нормализация является желательной, но в сложных БД могут появляться другие критерии, мешающие полной нормализации таблиц БД [9].

Вопросы для самопроверки

1. Каковы два основных направления использования вычислительной техники?
2. Что такое информационная система?
3. Когда появились первые информационные системы?

Конец ознакомительного фрагмента.