



Высшая Школа Цифровой Культуры

Университет ИТМО

[dc@itmo.ru](mailto:dc@itmo.ru)

**Оглавление**

[**1. Информационные системы 3**](#_heading=h.g73xgnrzci81)

[**2. Архитектура ИС 6**](#_heading=h.7qenfsm7avfh)

[**3. Основные функции систем управления данными 8**](#_heading=h.l81cie15dykb)

[**4. Архитектура СУБД 18**](#_heading=h.whwqo9or462x)

[**5. Введение в реляционные базы данных 25**](#_heading=h.g1ukatroqs7k)

[**6. Диаграммы сущность-связь (ER-диаграммы). Сущности. 29**](#_heading=h.hy5ckutsp89d)

[**7. Диаграммы сущность-связь (ER-диаграммы). Связи. 34**](#_heading=h.1i4uhtz6hjsi)

[**8. Преобразование ER-модели в реляционную базу данных 40**](#_heading=h.c44ooll7gtwa)

[**9. Нормализация в базах данных 46**](#_heading=h.f5mn9jqnll1t)

[**10. 3 нормальная форма 53**](#_heading=h.7dy58mafaidw)

[**12. Денормализация отношений 57**](#_heading=h.2dqju75wo2id)

# **Информационные системы**

В современном мире нас окружают различные информационные системы, благодаря которым базовые задачи, еще недавно требовавшие значительное время на выполнение, становятся простыми и незаметными. С помощью информационных систем мы совершаем покупки, регистрируемся на различные мероприятия, записываемся к врачу. При этом мы ожидаем, что системы должны работать быстро и без ошибок.

Разработка информационных систем является сложной задачей, требующей высокой квалификации. При этом, чтобы системами можно было пользоваться в реальном мире, к ним, в процессе разработки, предъявляют разные требования.

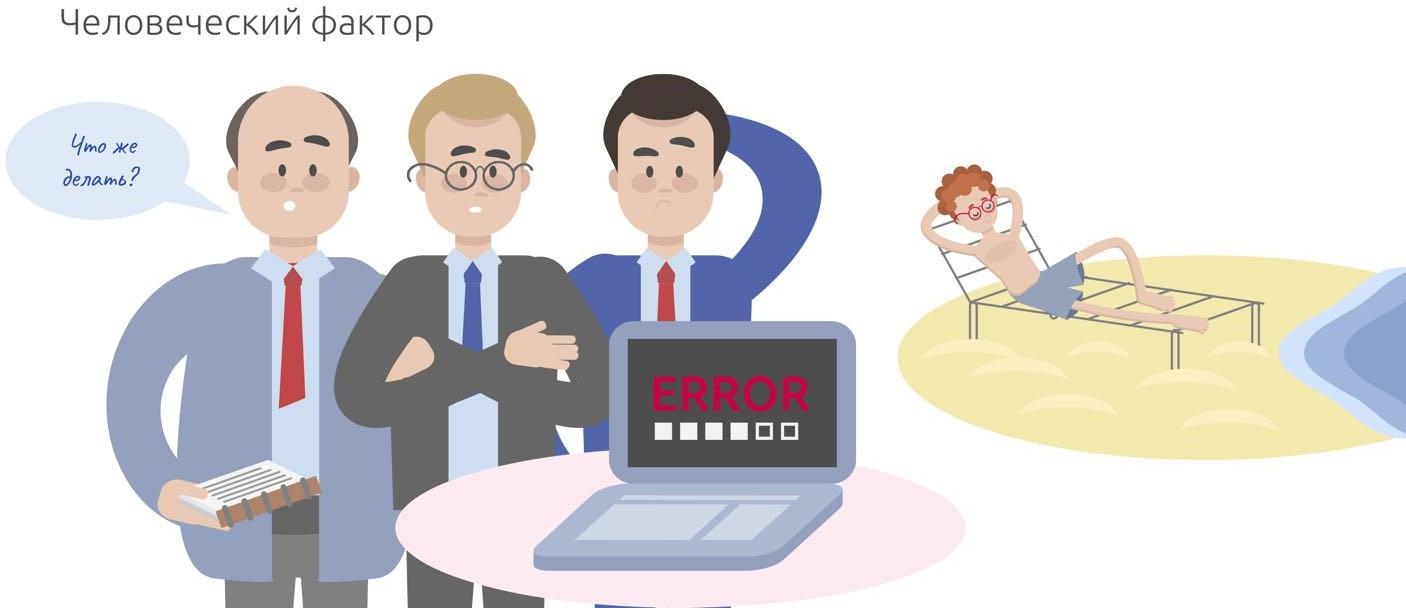
Существует множество различных требований к информационным системам. Среди основных можно выделить следующие:

* надежность;
* масштабируемость;
* удобство разработки и поддержки.

Давайте рассмотрим эти требования поподробнее. Надежность предполагает, что система должна продолжать выполнять свои функции корректно, как ожидает пользователь, даже при возникновении ошибок и нестандартных ситуаций. Для примера предположим, что при регистрации участника конференции студенту необходимо ввести номер телефона. Система, в которой осуществляется ввод этих данных, должна быть спроектирована и реализована таким образом, что заполнение поля произвольной последовательностью символов, не должно повлечь за собой негативных последствий. Программа должна распознать некорректные данные и оповестить об этом пользователя. Кроме того, могут возникнуть различные жизненные ситуации, такие как отключение интернета. В таком случае должен быть заранее определен протокол взаимодействия, например, система может информировать пользователя, что она находится в офлайн-режиме и все действия сохраняются на локальном компьютере (если это возможно), а после подключения к сети отложенные события выполняются в соответствии с обычным порядком.

Когда мы говорим про масштабируемость, то обычно подразумеваем способность системы справляться с увеличением нагрузки. Это может достигаться с помощью добавления требуемых ресурсов.

Если система плохо масштабируема или не масштабируема, то добавление новых ресурсов не улучшит систему, увеличение нагрузки на систему приведет к ухудшению ее работы или полному отказу. Стоит отметить, что про масштабирование можно говорить не только с точки зрения аппаратных ресурсов, на которых базируется система, но и с точки зрения её программной реализации, а именно: как будут адаптироваться к увеличению нагрузки, использованные в программе алгоритмы или подключенные библиотеки.

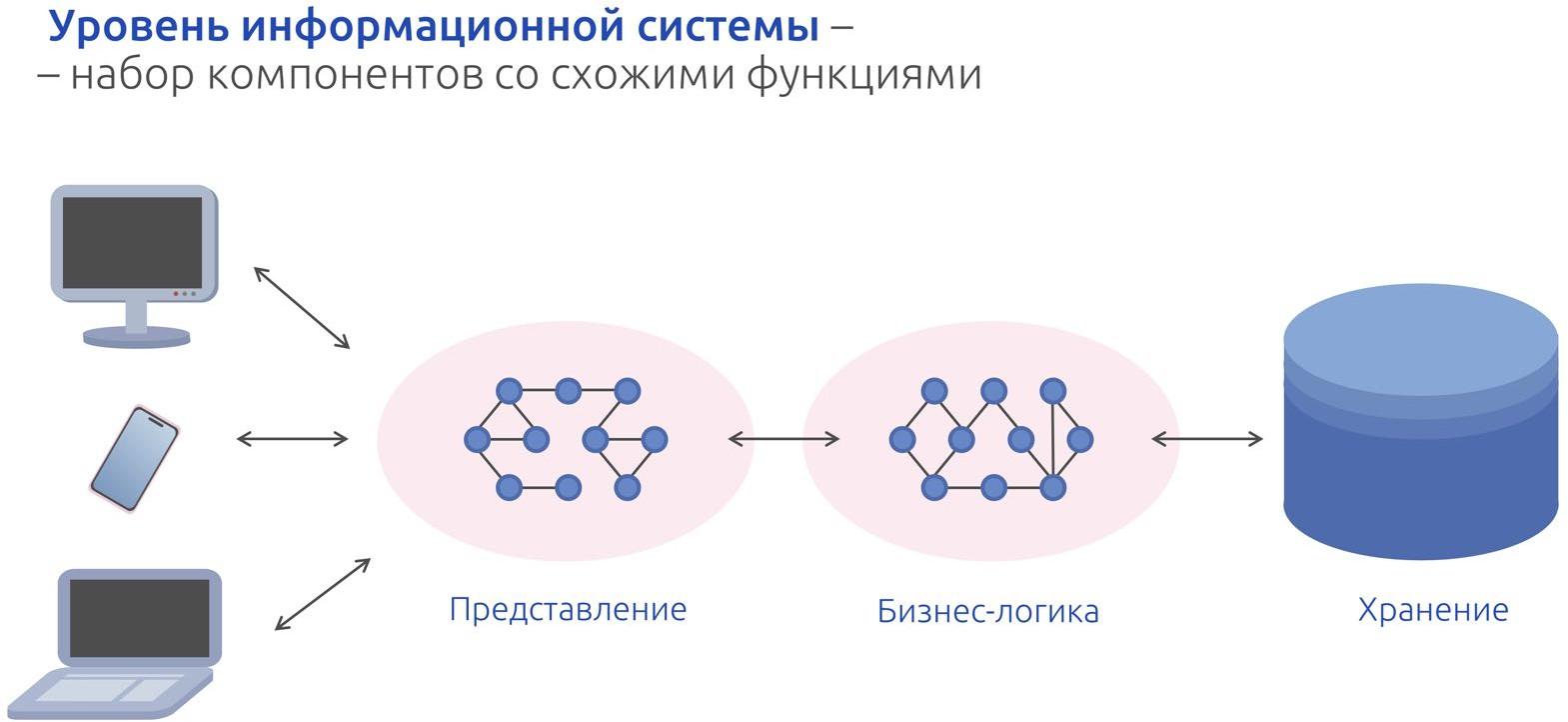
Важным требованием к системам является возможность добавления новых функциональных возможностей в информационную систему после ввода ее в эксплуатацию. Ситуация может сложиться так, что система разработана и успешно выполняет свои первоначальные задачи. Однако в один прекрасный день изменился формат документов, с которым работала первоначальная система и теперь необходимо добавить поддержку нового формата. Для удачно спроектированной системы потребуется незначительно изменить код или даже просто поменять некоторые настройки системы. Система, при создании которой не учитывалась возможность последующих изменений, потребует значительных дорогостоящих изменений в лучшем случае, а в худшем случае систему или ее часть потребуется полностью переписать.

Также важно учитывать человеческий фактор. Обычно системы создаются командами разработчиков. При этом состав команды часто меняется. При разработке системы важно учитывать, что люди, которые создают систему сейчас, могут отсутствовать, когда будет необходима доработка системы. В связи с этим важно, чтобы при разработке и сопровождении все разработчики использовали одни и те же согласованные и стандартизированные технологии и средства, которые могут быть изучены новыми членами команды в относительно короткое время.

Что из этого следует? Для того, чтобы можно было реализовать эти требования на практике, систему принято строить из отдельных небольших частей, которые называются модулями или компонентами. Каждый компонент отвечает за определенную часть системы. Например, может быть компонент, отвечающий за отправку сообщений пользователя, отображение истории сообщений, формирование различной статистики. Эти компоненты, как различные части конструктора, соединяются друг с другом, чтобы получилась итоговая система с требуемыми функциональными возможностями.

# **Архитектура ИС**

Среди различных компонентов можно выделить те, которые имеют сходства с точки зрения выполняемых ими функций и используемых технологий. Множество таких компонентов называется слоем или уровнем приложения информационной системы.



Обычно в информационной системе выделяют следующие уровни:

* **Уровень представления**, отвечающий за формирование пользовательского интерфейса. Компоненты, входящие в уровень представления, отвечают за отрисовку различных интерфейсов системы и взаимодействие системы с пользователем. Например, к этому уровню относится визуализация банковской истории пользователя в онлайн банке или форма регистрации на студенческую конференцию.
* **Уровень бизнес-логики**. В рамках этого уровня реализуются функции, специфичные для внутренних процессов той или иной информационной системы. Например, это может быть реализация алгоритма поиска курсов в системе, чтобы студент определенного направления мог записаться на необходимые курсы согласно его учебному плану, или формирование рейтинга студентов группы на основе баллов, полученных учащимися при прохождении курсов.
* **Уровень хранения** используется для постоянного хранения данных приложения. В состав данного слоя могут входить различные компоненты, отвечающие за организацию взаимосвязи информационной системы с хранилищем, осуществление эффективного извлечения данных из долговременного хранилища или реализацию хранения данных.

Для упрощения разработки и последующего сопровождения системы компоненты, находящиеся на одном уровне могут взаимодействовать только с компонентами, находящимися на соседних уровнях. То есть, например, очень не рекомендуется, чтобы уровень представления мог напрямую взаимодействовать с уровнем хранения информационной системы. Система, реализованная таким образом, может работать, но осуществлять поддержку и модификацию такой системы оказывается гораздо сложнее.

В рамках лекции мы будем рассматривать, каким образом можно организовать уровень хранения информационной системы. С одной стороны, мы знаем, что данные хранятся в файлах и нам ничто не мешает использовать обычные файлы в качестве основы для уровня хранения. Однако в большинстве случаев для долговременного хранения данных используют специальные средства – базы данных и системы, которые ими управляют. Давайте разберемся, почему для при организации уровня хранения редко используются обычные файлы.

Когда мы говорили о файловых системах, мы описали файл как абстракцию, определяющую область данных и информацию на физическом носителе, которая может храниться потенциально неограниченное количество времени. Файлы представляют собой линейный массив байтов, внутренняя структура которого зависит от приложения, которое использует данный файл. Например, для текстовых файлов содержимое представляет последовательность символов, представленную байтами. Получается, что если программа определенным образом или в определенном формате сохраняет данные в файл, то другая программа, чтобы получить доступ к данным, должна знать в точности структуру данного файла. При этом, если программист решит изменить формат файла, то все остальные программы, использующие файл, должны также быть изменены, или они перестанут корректно работать.

Также, одни и те же данные в разных программах могут быть по-разному представлены и организованы. Структура данных, если нет изначальной схемы, определяется программистом. Собрать воедино данные, используемые разными программами в своих форматах, и избавиться от избыточности очень непросто. Осложняется этот процесс тем, что данные программ могут быть не статичными, они изменяются и дополняются. В связи с этим процесс централизованного сбора и переработки данных со всех приложений будет носить постоянный характер.

Кроме всего вышеперечисленного возникает следующий вопрос, как осуществить конкурентный доступ к одним и тем же данным из нескольких различных приложений. При этом основная проблема – не просто прочитать и записать данные, а учитывать изменения, которые производят приложения, работающие с теми же данными в одно и то же время.

Многократная реализация такой функциональности, требуемая для работы с файлами, значительно удорожает и усложняет конечный программный продукт. Поэтому возникла идея создавать централизованные системы, обеспечивающие доступ к данным, хранящимся в определенном формате.

Таким образом, появились специализированные средства для хранения данных – базы данных. По сути, базы данных – это файлы, снабженные описанием хранимых в них данных и находящиеся под управлением специальных программных комплексов, называемых Системами управления базами данных (СУБД). Благодаря использованию таких комплексов можно достичь снижения стоимости разработки приложений, так как системы управления базами обеспечивают надежное хранение и унифицированный доступ к требуемым данным, о чем мы с вами поговорим в следующем разделе.

# **Основные функции систем управления данными**

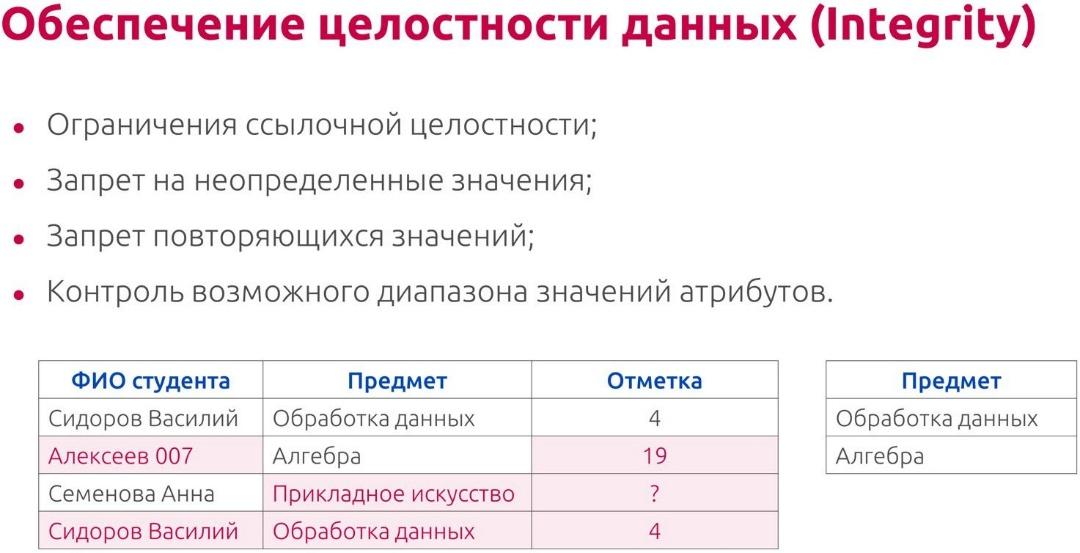
Рассмотрим основные функции, которыми должны обладать системы управления базами данных (СУБД).

Обеспечение независимости данных и приложений изначально рассматривалось как важнейший элемент систем управления базами данных. Те принципы, которые понимаются под независимостью данных и приложений, разбиваются на три возможные группы:

* одни и те же данные могут использоваться для различных приложений;
* появление новых требований к данным (например, добавление новых полей, таблиц, бизнес-логики поведения данных и т.п.) не должно оказывать влияния на работу существующих приложений;
* допустимо асинхронное внедрение новых версий приложений.

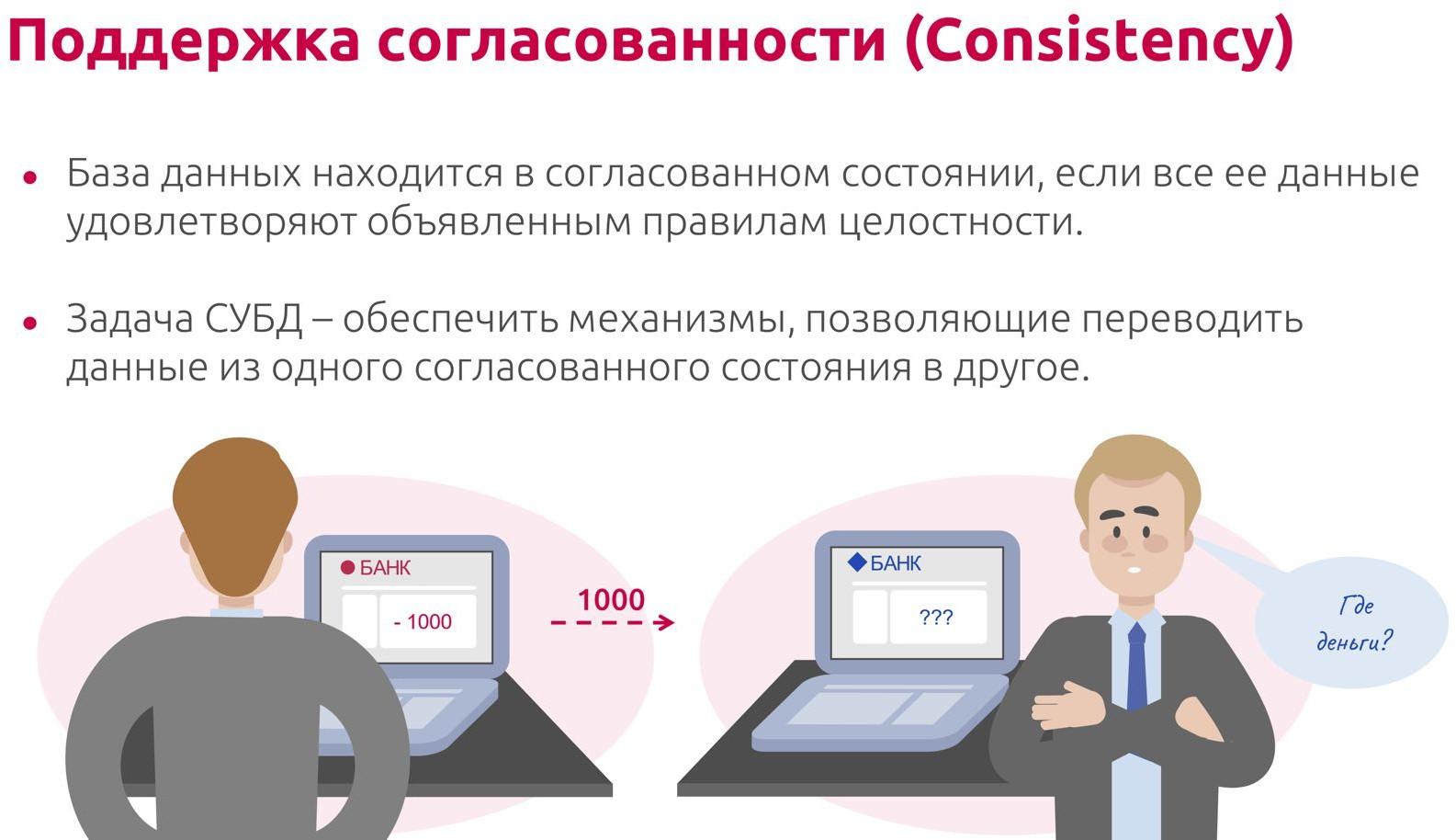
Для поддержки этих требований были введены языки, позволяющие описывать структуры и бизнес-логику поведения данных. Описание структур данных хранится в специальных словарях, предусмотренных в системах управления данными. Словари данных и языки описания структуры и бизнес-логики данных в том или ином виде включены почти во все современные СУБД. Большинство систем строго придерживаются принципов независимости данных, однако следует отметить, что последнее время стали появляться системы хранения данных, которые для повышения производительности и простоты управления отказываются от этих важных принципов. Появляются хранилища данных, узко ориентированные на специфику данных определенного вида (например, тексты, фотографии и пр.) и функциональность одного конкретного приложения. Изменение структуры данных в таких приложениях, как правило, затрагивает работу приложений и требует его модернизации.

Система управления базой данных контролирует соответствие хранимых данных описанной структуре. Кроме структуры данных, можно описать некоторые правила, которым должны удовлетворять хранимые данные. Например, отметка за экзамен студента должна быть в интервале от 1 до 5; предмет, по которому ставится отметка, должен присутствовать в списке читаемых курсов и т.п. Такого рода правила называют ограничениями целостности, и они также описываются с помощью специальных языков описания данных. За проверку выполнения заданных ограничений отвечает СУБД. Добавление или изменение данных, нарушающих эти условия, не могут быть выполнены.

Типовыми ограничениями целостности, которые на сегодняшний день присутствуют в большинстве СУБД, являются:

* ограничения ссылочной целостности;
* запрет на неопределенные значения;
* запрет повторяющихся значений;
* контроль возможного диапазона значений атрибутов.

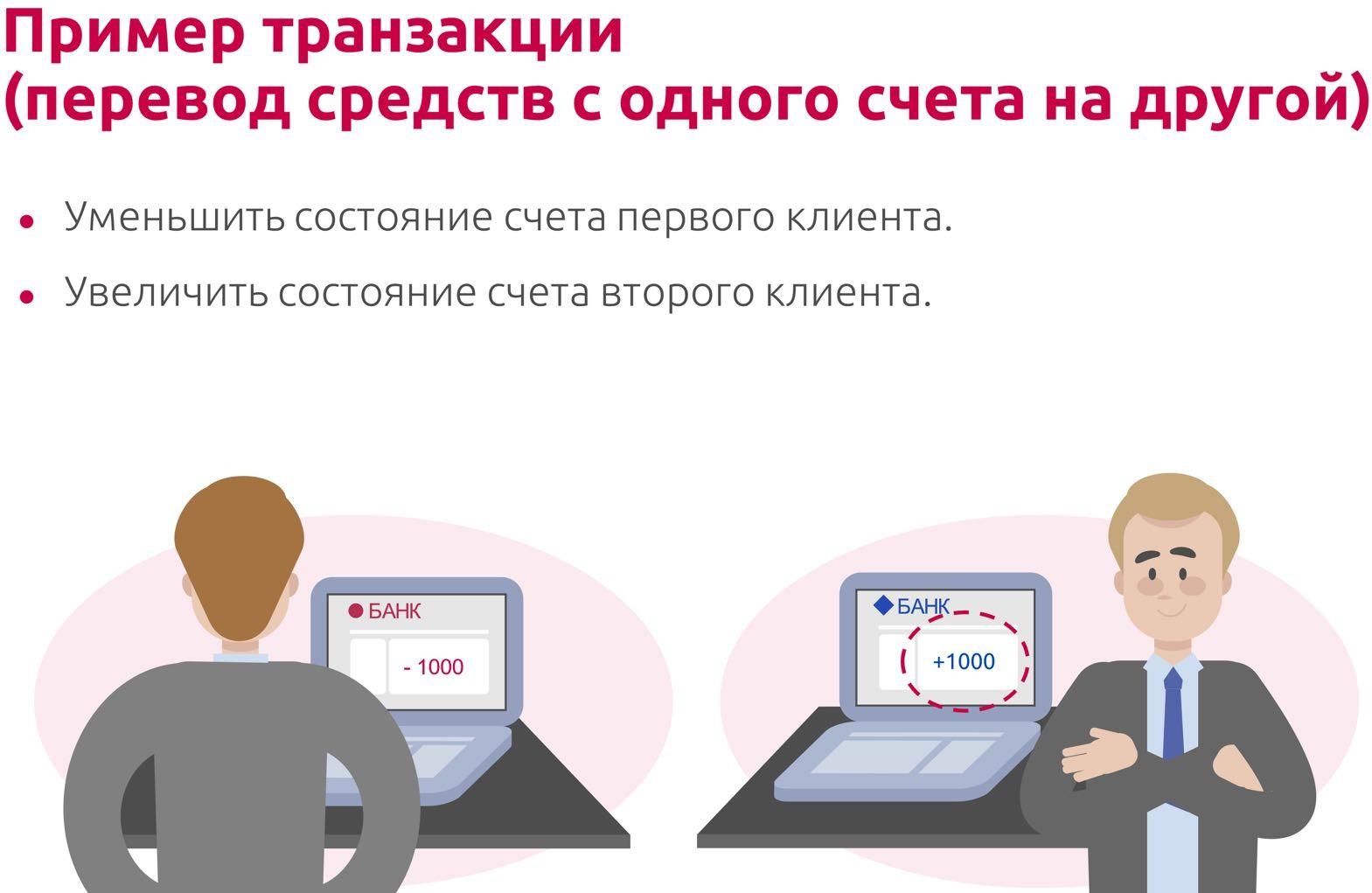
Однако существуют СУБД, в которых правила целостности либо вообще отсутствуют, либо присутствуют лишь частично. Если на уровне хранения данных правила целостности не проверяются, а для предметной области эти правила, тем не менее, важны, то приходится их контролировать их на уровне создаваемых приложений.



Поддержка согласованности связана с состояниями данных. Часто единая операция с точки зрения бизнес-логики состоит из нескольких мелких операций с данными. Например, перевести деньги с одного счета на другой, увеличить всем студентам стипендию на 10%. В процессе выполнения этих операций может возникнуть ситуация, когда данные какой-то период времени не будут соответствовать всем правилам целостности. База данных находится в согласованном состоянии, если все ее данные удовлетворяют объявленным правилам целостности. Задача СУБД – обеспечить механизмы, позволяющие переводить данные из одного согласованного состояния в другое.

Поддержка согласованности данных тесно связано с понятием транзакции. Дадим формальное определение этому понятию.

Набор операций, переводящих базу из одного согласованного состояния в другое, принято называть транзакцией.

Еще одно возможное определение транзакции звучит так: логически неделимая последовательность операций, результат которых должен сохраняться целиком, либо не сохраняться вообще.

Для пояснения понятия транзакции приводят обычно следующий пример. В базе данных, содержащей сведения о клиентах банка, необходимо зафиксировать перевод денег со счета одного клиента другому. С точки зрения бизнеса (и базы) это одна неделимая операция, которую принято называть транзакцией. Но на самом деле за этой операций стоят три действия, которые необходимо выполнить в базе:

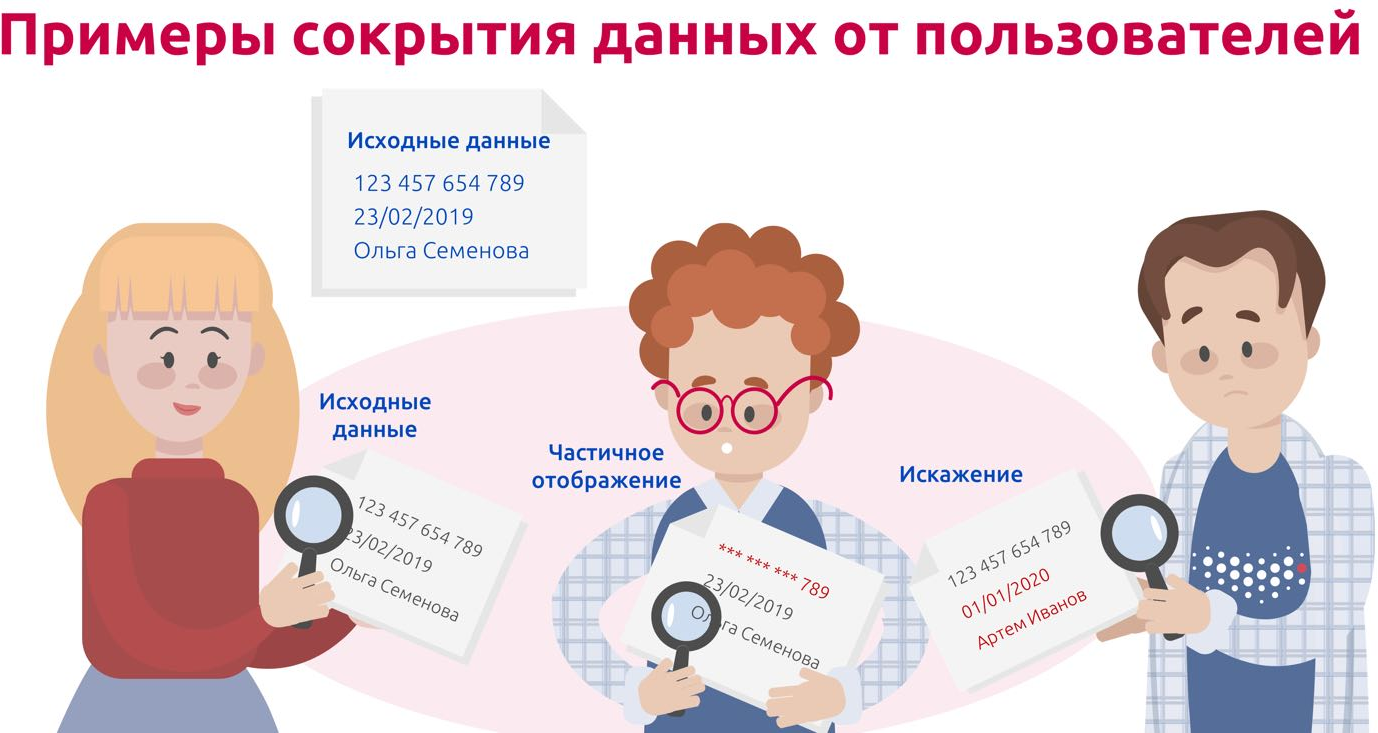
* уменьшить состояние счета первого клиента;
* увеличить состояние счета второго клиента;
* зафиксировать соответствующую запись в журнале проводок.

Поддержка согласованности данных со стороны СУБД означает, что после удачного или неудачного завершения работы приложения база данных в любом случае окажется в согласованном состоянии. Для транзакций это означает, что все транзакции приложения либо будут выполнены полностью, либо не оставят никаких следов в данных (то есть СУБД обеспечит откат не полностью завершенных транзакций).

Это требование называют атомарностью транзакций (Atomicity). После завершения транзакции изменения в данных становятся постоянными. Такое свойство называют долговечностью транзакций (Durabibity).



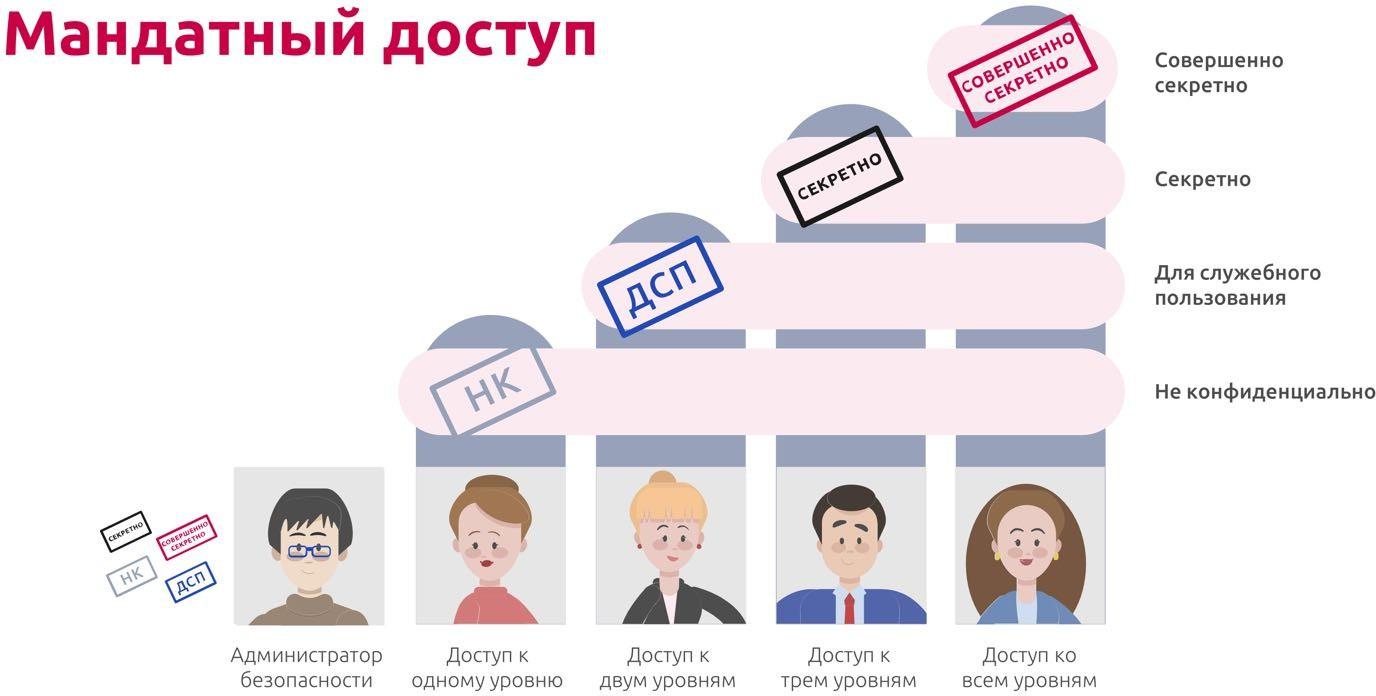
Если с данными параллельно работают несколько пользователей, то их действия могут привести данные в некорректное состояние. Например, два пользователя могут одновременно изменять одни и те же данные. Поддержка согласованности в таком случае существенно усложняется. Для обеспечения параллельной работы нескольких пользователей в СУБД были разработаны разнообразные алгоритмы исполнения транзакций, создающие для пользователей иллюзию изолированности (Isolation) работы с данными, но не приводящие базу к несогласованному состоянию. Про СУБД, поддерживающие все эти свойства, говорят, что они обладают ACID-свойствами.

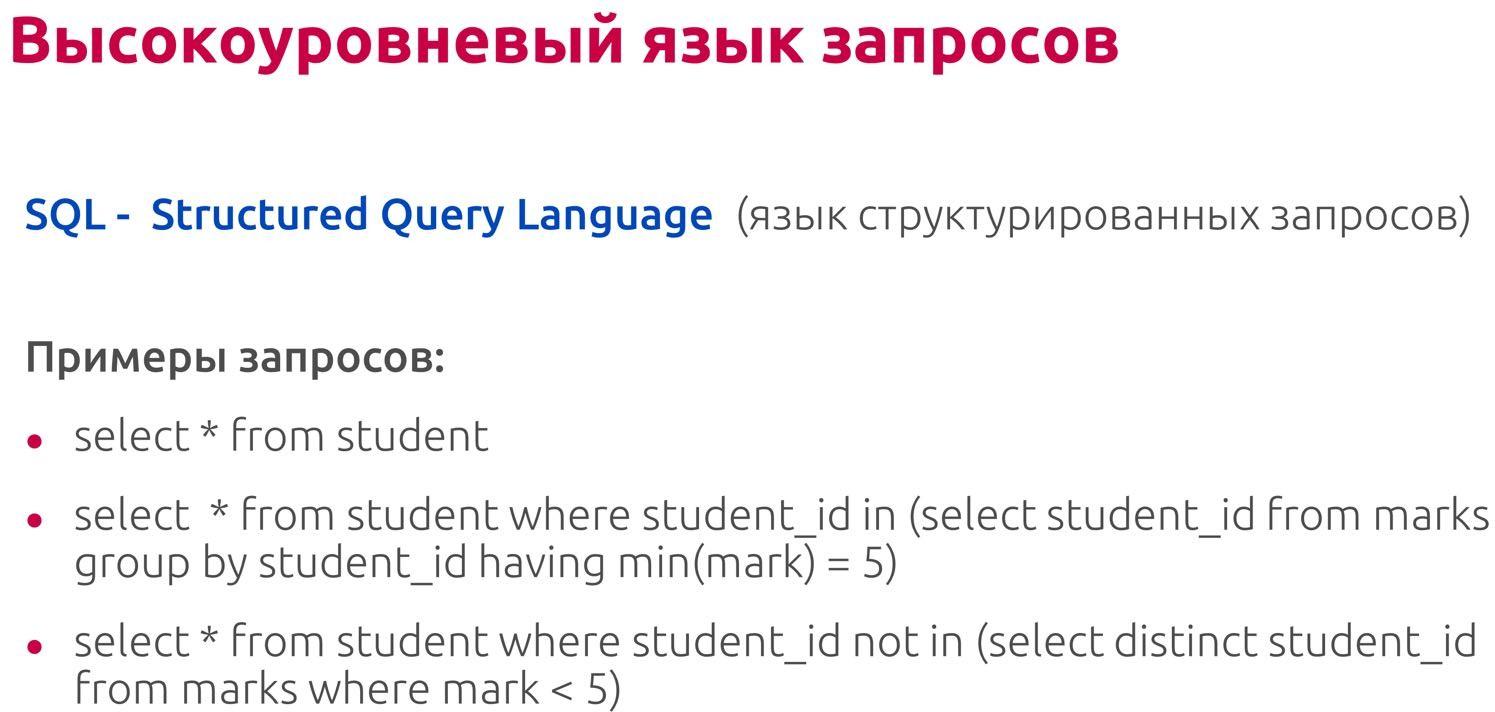
Функции защиты данных от несанкционированного доступа и разграничение доступа является перманентной задачей практически любой СУБД. Сразу после создания пользователей в базе возникает необходимость управления их доступом к различным фрагментам базы данных. Например, сотрудник отдела кадров организации может иметь право просматривать данные о зарплате сотрудников, но не должен обладать полномочиями для изменения ставок. Менеджеры должны видеть данные, которые относятся к сотрудникам только своих подразделений и т.п. С этой целью в СУБД были разработаны разнообразные приемы, позволяющие скрывать реальные данные от пользователей. Например, с помощью, так называемых, представлений можно скрывать от пользователя реальную структуру данных и отображать ему только те данные, на которые ему разрешено смотреть.

В последних версиях некоторых промышленных СУБД появляются разнообразные приемы для частичного отображения данных, а иногда и для преднамеренного, достаточно правдоподобного искаженного отображения данных. Некоторые приемы для отображения и искажения данных вы можете видеть на рисунке.

Разграничение прав доступа с помощью привилегий и ролей было придумано для того, чтобы явно декларировать каждому пользователю базы права на выполнение тех или иных операций в базе (например, просмотр или редактирование отдельных фрагментов базы данных, создание новых структур данных и т.п.).

Мандатный доступ основан на назначении меток конфиденциальности для информации, содержащейся в базе данных, и выдаче официальных разрешений (допусков) пользователям на обращение к информации того или иного уровня конфиденциальности. Например, допуск к материалам с меткой «Секретно».

Еще одно перспективное направление развития защиты данных – изоляция администраторов серверов баз от, собственно, данных. В большинстве сегодняшних СУБД администраторы серверов баз данных могут видеть и изменять все данные в БД. Средства защиты в последних версиях некоторых СУБД позволяют администраторам выполнять все операции по администрированию БД, но не позволяют видеть и менять данные.

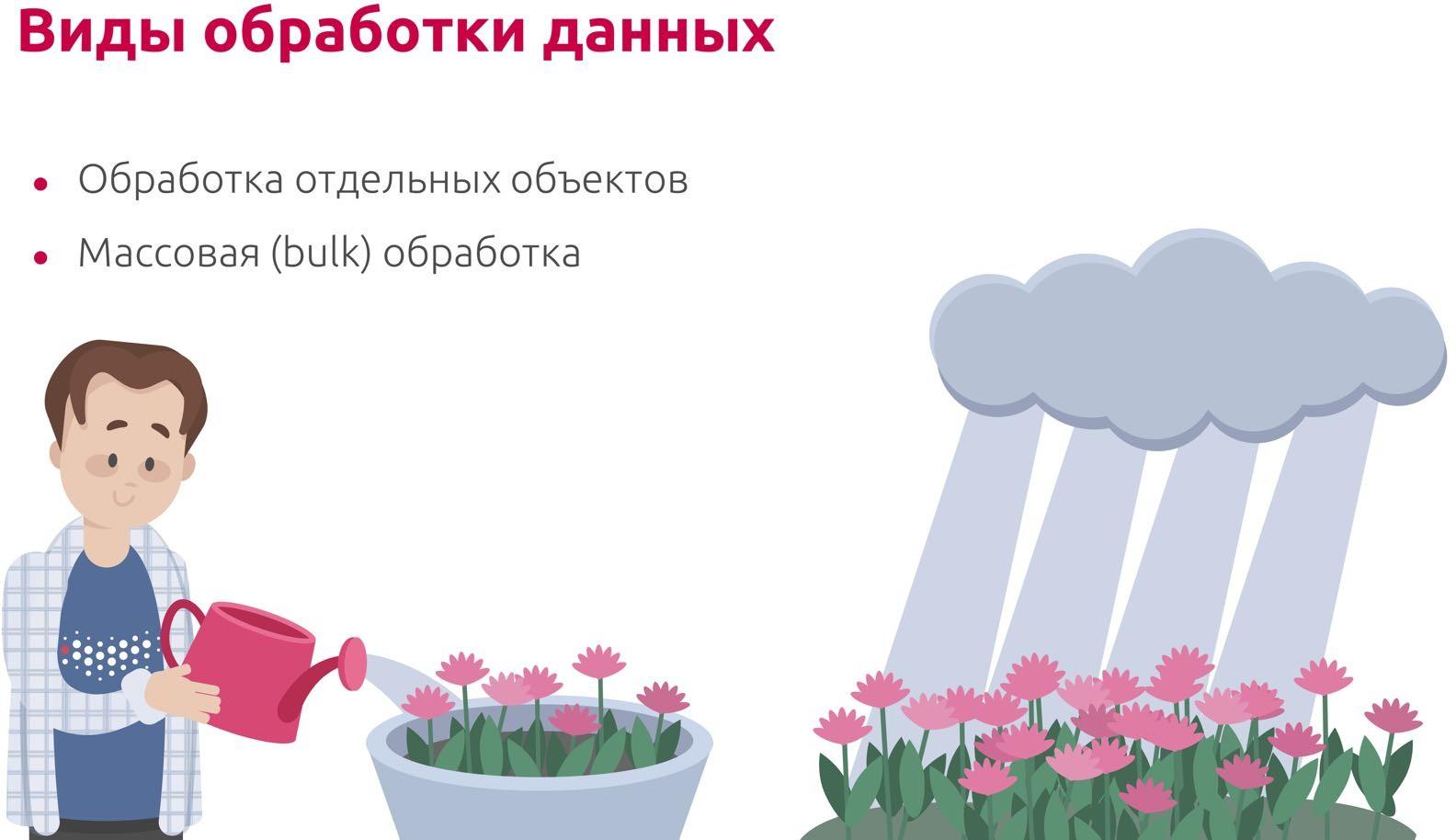
Наличие высокоуровневого и эффективного языка запросов является, пожалуй, наиболее важной отличительной чертой СУБД.

С этой целью в рамках разнообразных систем управления были реализованы высокоуровневые декларативные языки для манипуляций над данным, из которых особого внимания заслуживает SQL (Structured Query Language – язык структурированных запросов). Элегантность, декларативность и независимость языка от специфики СУБД, а также его поддержка ведущими производителями баз данных, сделали язык SQL основным стандартом для обработки данных. Когда пишут запрос на языке высокого уровня (и в частности, на SQL), то говорят, какие данные надо извлечь из таблиц, но не говорят, как. Почти любой, даже самый простой запрос может быть исполнен разными способами – за счет перестановки элементарных операций, из которых он состоит, за счет использования дополнительных структур (например, индексов), созданных для ускорения выполнения запросов и т.п. Способ выполнения запроса называется планом запроса.

Как выбрать оптимальный план запроса? Какой компонент СУБД отвечает за эту задачу? Практически во всех СУБД, использующих высокоуровневые языки запросов, появились, так называемые, оптимизаторы запросов. Основная функция оптимизатора запросов состоит в построении возможных планов выполнения запросов и выборе оптимального из них. Как правило, наилучшим планом является тот, на выполнение которого необходимо меньше времени, что чаще всего оценивается количеством операций чтения, однако в некоторых случаях критерии могут зависеть от требований приложения. Например, требуется минимизировать время получения первых строк запроса.

Оптимизаторы выбирают планы на основе явно или неявно определенной функции стоимости запроса. В настоящее время алгоритмы оптимизации достаточно хорошо проработаны и реализованы в промышленных СУБД, и только в достаточно редких случаях требуется ручная настройка (с помощью, так называемых, подсказок оптимизатору) или полное переписывание запросов с целью улучшения их производительности. Еще одним аспектом высокоуровневого языка запросов выступает вид обработки данных, который обеспечивает этот язык запросов, т.е. какие единицы данных могут быть обработаны с помощью одной команды языка. Различают два вида обработки:

* обработка отдельных объектов;
* массовая (bulk) обработка.



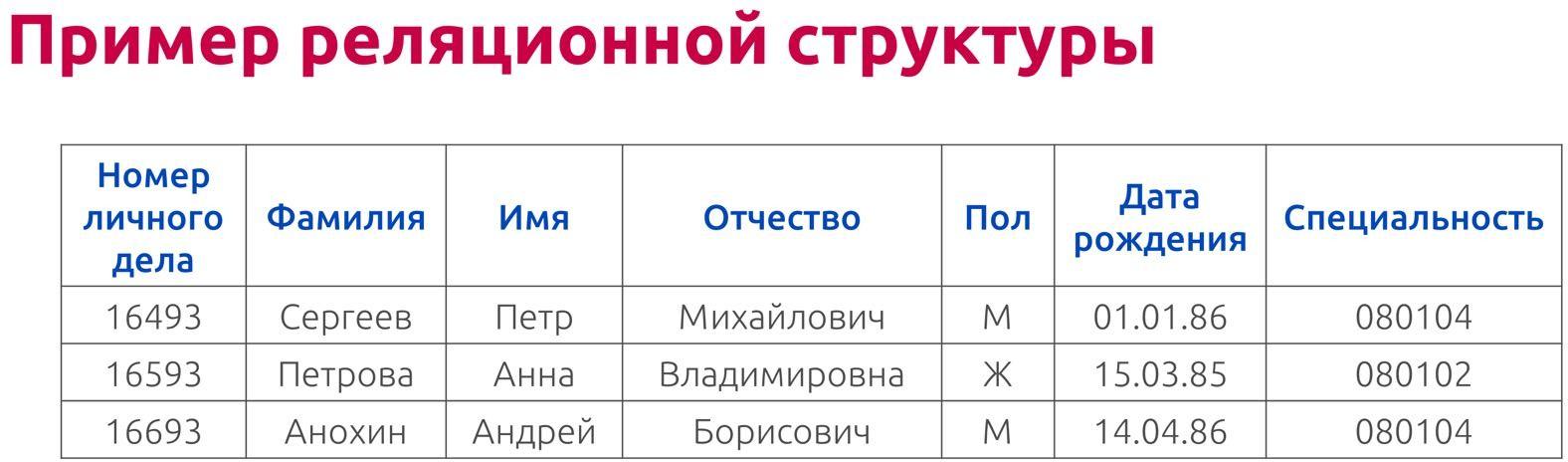
К примеру, язык SQL изначально ориентирован на массовую обработку данных. С помощью одной команды можно обработать большое количество строк. Более того, особенностью всех популярных реализаций SQL в рамках традиционных СУБД является то обстоятельство, что такие команды обработки выполняются значительно эффективнее обработки единичных объектов, то есть эффективнее обработать с помощью одной команды 10000 объектов базы, чем 10000 раз выполнять обработку отдельных объектов. Но следует отметить, что системы, в которых языки запросов ориентированы на обработку отдельных элементов, также существуют и, более того, активно развиваются и обсуждаются.

Какое-то время назад еще можно было классифицировать СУБД по типам поддерживаемых моделей данных: иерархических, сетевых, реляционных и т.д. Модели баз данных зарождались именно в таком порядке, как перечислены на рисунке.

В иерархических базах данных каждая запись имеет одного родителя. Это создаёт древовидную структуру, в которой записи идентифицируются по их отношениям с цепочкой родительских записей. Пример данных, организованных таким образом вы можете видеть на рисунке.

Сетевые модели баз данных расширили функциональность иерархических: записи могли иметь более одного родителя. А следовательно, появилась возможность моделировать достаточно сложные отношения. Пример данных, организованных таким образом вы можете видеть на рисунке.

Особо широкое распространение получили реляционные модели данных, в основе которых лежит идея представления любых данных в виде структурированных таблиц. Каждый столбец в таблице имеет имя и тип. Каждая строка представляет отдельную запись о каком-то хранимом в базе объекте. СУБД, управляющие данными такой структуры, принято называть реляционными или традиционными.



Ввиду разнообразия существующих моделей данных сегодня принято делить модели (по крайней мере, на самом верхнем уровне) на:

* структурированные;
* слабоструктурированные;
* неструктурированные.

Поясним, что это означает. Если хранимые данные обладают четким набором однотипных признаков, то такие данные принято называть структурированными. Примерами таких данных могут быть данные клиентов банка, студентов университета, товаров в магазине. Для работы со структурированными данными отлично подходит реляционная модель данных. Но в настоящее время появилось огромное количество данных, не имеющих четкой структуры – это фотографии, видео, текстовые документы – так называемые, неструктурированные данные. Если удается извлечь хотя бы часть общих признаков – дату создания, автора документа, размер и формат фотографии – то такие данные становятся слабоструктурированными.



В настоящее время большинство крупных производителей, которые первоначально позиционировали свои СУБД как реляционные (например, ORACLE и DB2) позволяют создавать разнообразные модели данных, включая неструктурированные. Такие СУБД принято называть мультимодельными. В нашем курсе мы ознакомимся с разными моделями данных, но начнем, с наиболее распространенной – реляционной модели данных.

# **Архитектура СУБД**

Одной из важнейшей характеристик современных СУБД является наличие архитектуры, допускающей многопользовательскую работу с данными. Развитие вычислительной техники и программного обеспечения приводили (и продолжают приводить) к развитию архитектурных решений, используемых для реализации доступа к данным в контексте использования СУБД. Рассмотрим эти решения в хронологическом порядке.

**Централизованная архитектура**

При использовании централизованной архитектуры база данных, СУБД и прикладная программа (т.е. приложение) располагаются на одном компьютере. Работа организована следующим образом: Работа организована следующим образом:

* база данных находится на жестком диске компьютера;
* на том же компьютере установлены СУБД и приложение для работы с БД;
* пользователь запускает приложение и инициирует обращение к БД на выборку/обновление информации;
* все обращения к БД идут через СУБД, которая содержит сведения о физической структуре БД;
* СУБД обеспечивает выполнение запросов пользователя;
* СУБД возвращает в приложение результаты запросов;
* приложение отображает результаты выполнения запросов.

**Архитектура файл–сервер**

Появление персональных компьютеров и локальных вычислительных сетей привели к появлению новой архитектуры с названием файл–сервер. Эта архитектура предполагает назначение одного из компьютеров сети выделенным сервером, предназначенным для хранения файлов базы данных. В соответствии с запросами пользователей файлы с файл–сервера передаются на клиентские компьютеры пользователей, где и осуществляется основная часть обработки данных. Выделенный сервер выполняет роль хранилища файлов, но не участвует в обработке самих данных. Итак, работа организована следующим образом:

* база данных находится на жестком диске специально выделенного компьютера (файлового сервера);
* локальная сеть объединяет файловый сервер и клиентские компьютеры;
* на каждом клиентском компьютере установлена СУБД и приложение для работы с БД;
* приложения инициируют обращение к БД на выборку/обновление информации;
* все обращения к БД идут через СУБД, которая содержит сведения о физической структуре БД, расположенной на файловом сервере;
* СУБД обращается к данным, находящимся на файловом сервере;
* в результате обращений СУБД часть БД копируется на клиентский компьютер и там же и обрабатывается;
* в случае изменения, данные отправляются назад на файловый сервер с целью обновления БД;
* СУБД возвращает результаты запросов в приложение;
* приложение отображает результаты выполнения запросов.

**Архитектура клиент–сервер**

Следующая архитектура клиент–сервер также предполагает наличие компьютеров, объединенных в сеть, один из которых выполняет особые управляющие функции (его называют сервером базы данных). Архитектура клиент–сервер разделяет функции приложения пользователя (его принято называть клиентом ) и сервера. Приложение-клиент формирует запрос к серверу, на котором расположена БД, на специальном языке запросов (как правило – SQL). При этом ресурсы клиентского компьютера не участвуют в выполнении запроса, клиентский компьютер лишь отсылает запрос к серверу базы данных и получает результат, после чего интерпретирует его и отображает пользователю. Так как клиентскому приложению посылается только результат выполнения запроса, по сети пересылаются только те данные, которые необходимы клиенту. В итоге снижается нагрузка на сеть.

Итак, работа организована следующим образом:

* база данных находится на жестком диске специально выделенного компьютера (сервера базы данных);
* СУБД также располагается на сервере;
* сеть обЪединяет сервер базы данных и клиентские компьютеры;
* на каждом клиентском компьютере установлено приложение для работы с БД;
* приложения инициируют обращение к БД на выборку/обновление информации. Для общения с сервером используется язык запросов (как правило, SQL), т.е. по сети от клиента к серверу передается лишь текст запроса или вызовы хранимых в базе процедур и функций;
* все обращения к БД идут через СУБД, которая хранит внутри себя все сведения о физической структуре БД;
* на сервере осуществляется все выполнение запросов;
* на клиентский компьютер (в приложение) отправляются только результаты выполнения запросов;
* приложения отображают результаты выполнения запросов;
* механизм транзакций, поддерживаемый СУБД, не допускает одновременное изменение одних и тех же данных различными пользователями и возвращает базу к исходному состоянию при ошибках в работе программного обеспечения и вычислительной техники.

**Многоуровневая архитектура «клиент–сервер»**

В трехуровневой (или многоуровневой) архитектуре клиент–сервер бизнес логика, ранее входившая в клиентские приложения, выделяется в отдельное звено, называемое сервером приложений. При этом клиентским приложениям остается лишь пользовательский интерфейс. Как правило, в качестве клиентского приложения выступает Web-браузер. Что улучшается при использовании трехзвенной архитектуры? Теперь при изменении бизнес-логики более нет необходимости изменять клиентские приложения и обновлять их у всех пользователей. Кроме того, максимально снижаются требования к устройствам пользователей.

В результате работа строится следующим образом:

* база данных находится на жестком диске сервера базы данных;
* СУБД также располагается на сервере;
* существует специально выделенный сервер приложений, на котором располагается программное обеспечение (ПО) для реализации бизнеслогики;
* существует множество клиентских компьютеров, на каждом из которых установлен так называемый «тонкий клиент» – клиентское приложение, реализующее интерфейс пользователя (как правило – Web-браузер);
* сеть обЪединяет сервер базы данных, сервер приложений и клиентские компьютеры;
* на каждом из клиентских компьютеров пользователи имеют возможность запустить приложение – тонкого клиента;
* тонкий клиент на клиентском компьютере инициирует обращение к ПО сервера приложений;
* сервер приложений анализирует требования пользователя и формирует запросы к БД. Для общения с БД используется специальный язык запросов (как правило – SQL), т.е. по сети от сервера приложений к серверу БД передается лишь текст запроса;
* все обращения к БД идут через СУБД;
* результаты выполнения запросов возвращаются серверу приложений;
* сервер приложений анализирует результаты запросов и возвращает результат в клиентское приложение;
* клиентское приложение отображает результат.

**Облачная архитектура**

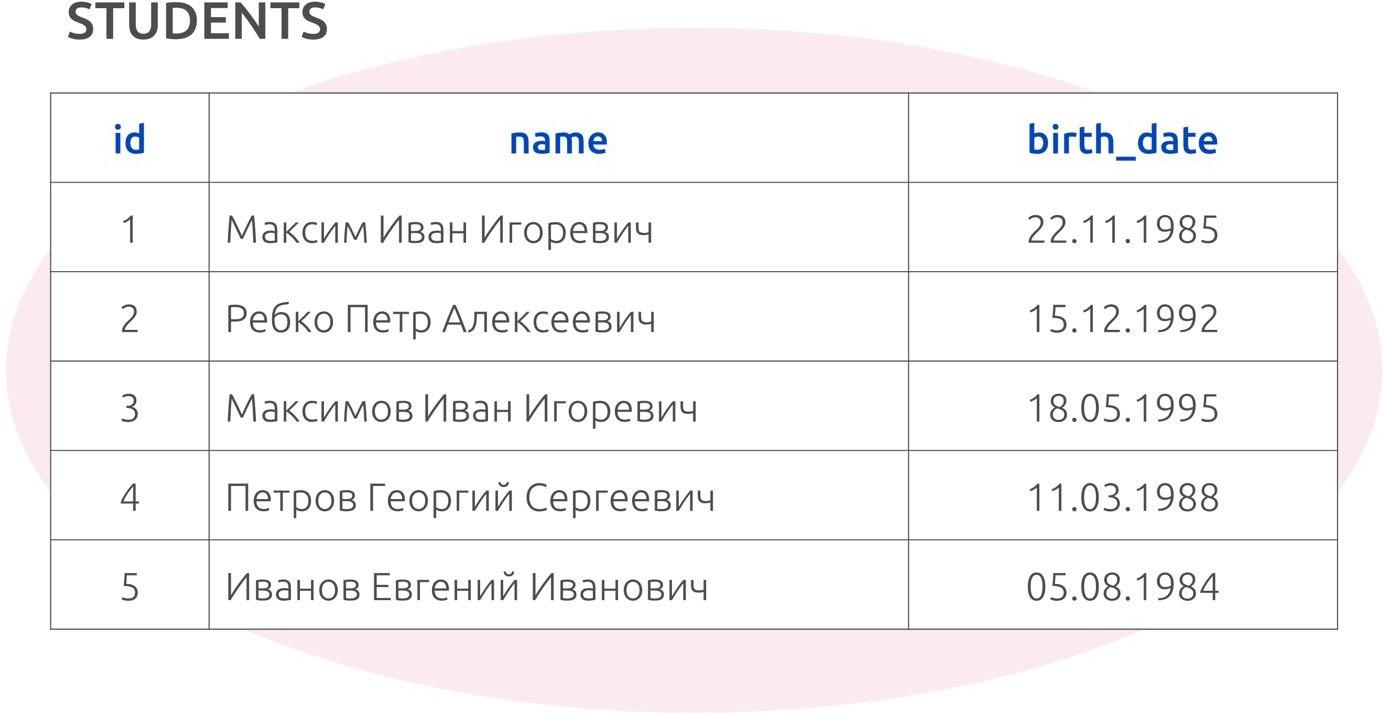
Сегодня среди множества облачных сервисов на рынке программного обеспечения имеется, конечно, и такая полезная услуга, как облачный доступ к СУБД. Особенно востребована она предприятиями, не желающими тратиться на инфраструктуру локальной сети, администрирование, масштабирование и другие типовые накладные расходы, характерные для СУБД. Что же такое – облачные СУБД?

Облачные СУБД – это полностью автоматизированный многопользовательский и неограниченно масштабируемый сервис, который через сеть Internet предоставляет функциональность СУБД, но управляется и администрируется провайдером сервиса. При этом не следует путать облачную СУБД (это сервис DBaaS – Database as a Service ) и СУБД, запущенную на виртуальной машине (т.е. когда пользователю предоставляется виртуальная машина, в облаке, а он уже сам (или с чьей-то помощью) должен администрировать СУБД, которая, возможно, будет туда установлена). На сегодняшний день облачные СУБД – это чрезвычайно востребованная архитектура, которая освобождает как мелкие, так и крупные предприятия, от рутинной работы, связанной как с созданием инфраструктуры, так и с администрированием баз данных.

В рамках нашего курса вы ознакомитесь с несколькими СУБД, а доступ к ним будет организован именно в облачной архитектуре.

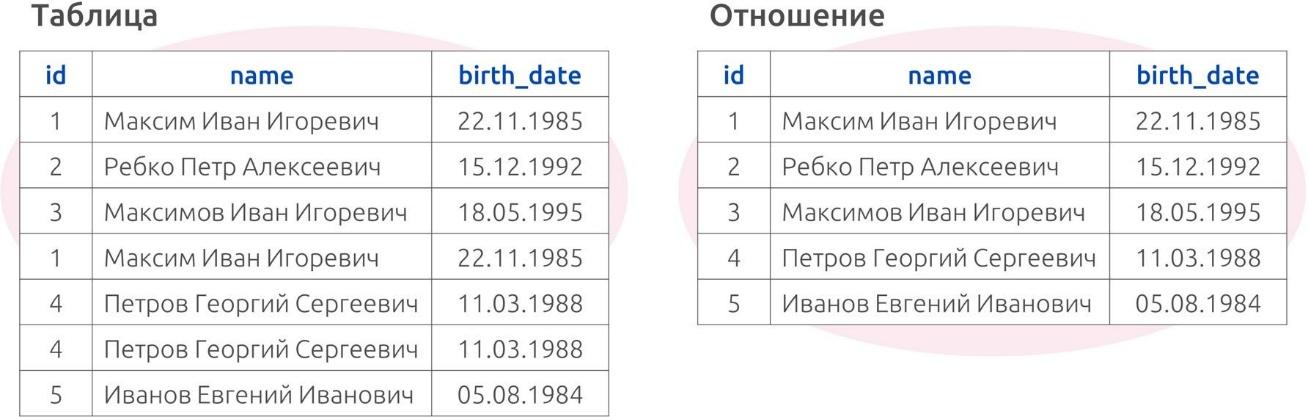
# **5. Введение в реляционные базы данных**

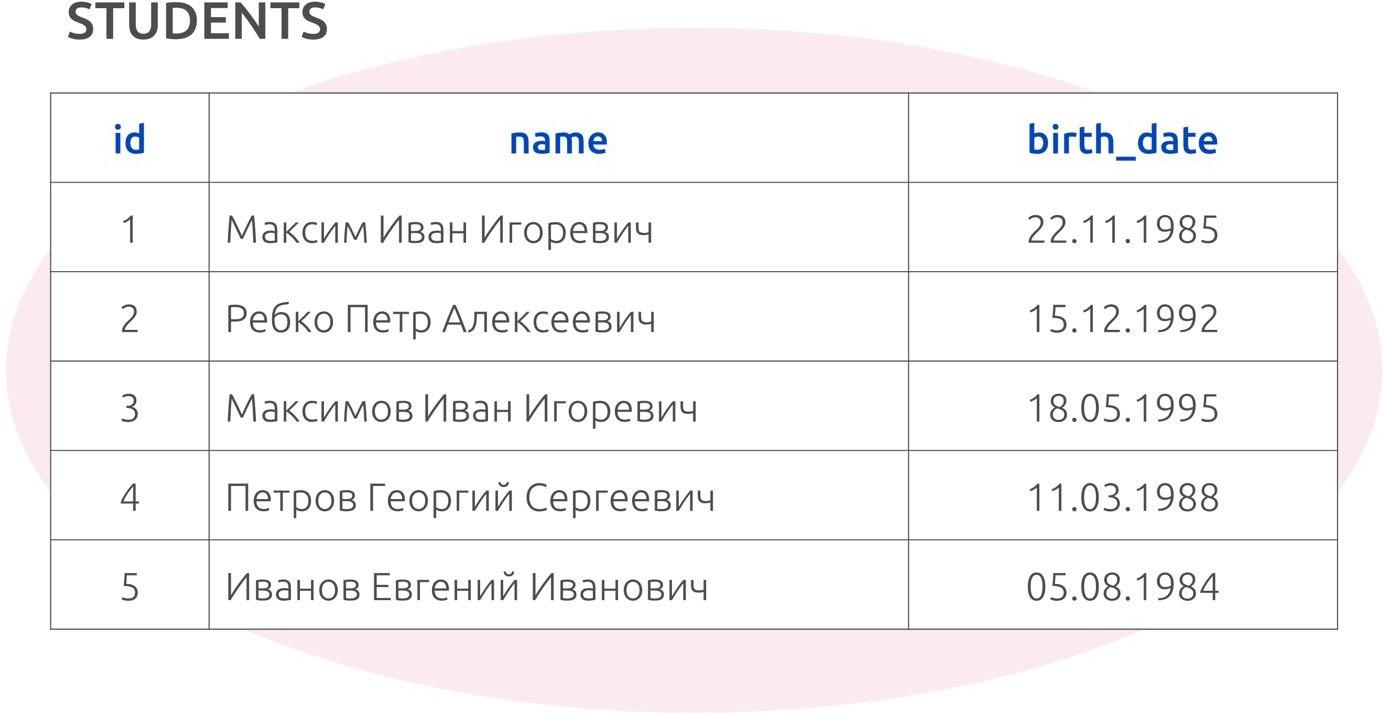
Наиболее популярной моделью хранения структурированных данных является, как мы уже говорили, реляционная модель. Реляционная модель данных была предложена в конце 60-х годов Эдгаром Коддом в работе «Реляционная модель данных для больших разделяемых банков данных» (A Relational Model of Data for Large Shared Data Banks). В дальнейшем Эдгаром Коддом были выдвинуты базовые правила, которым должна соответствовать система управления базами данных. Рассмотрим ключевые особенности, характерные для реляционной модели. Согласно Э. Кодду вся хранимая информация может быть представлена в виде совокупности таблиц, или, так называемых, отношений. Каждая таблица имеет свое уникальное имя.

В качестве примера рассмотрим простую таблицу STUDENTS, которая содержит данные студентов некоторого университета.

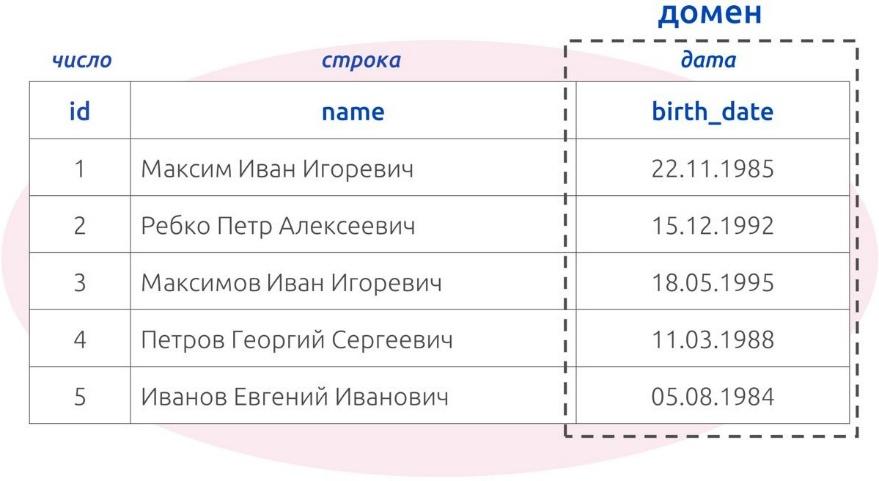
Данные записываются как строки в таблицах, каждая строка соответствует определенному элементу данных, в нашем случае – некоторому студенту. Таблица имеет определенное, заданное число столбцов. У каждого столбца есть свое уникальное имя. Как видно из примера, таблица содержит столбец с именем id, соответствующий идентификационному номеру студента, столбец name, содержащий строку с фамилией, именем и отчеством студента. В последнем столбце birth\_date хранятся даты рождений студентов. Каждая строка таблицы имеет фиксированное число элементов, соответствующее числу столбцов таблицы. В каждом столбце хранятся значения, соответствующие определенному атрибуту, или свойству объекта (в нашем случае студента).

Каждая строка таблицы имеет фиксированное число элементов, соответствующее числу столбцов таблицы. В каждом столбце хранятся значения, соответствующие определенному атрибуту, или свойству объекта (в нашем случае студента).



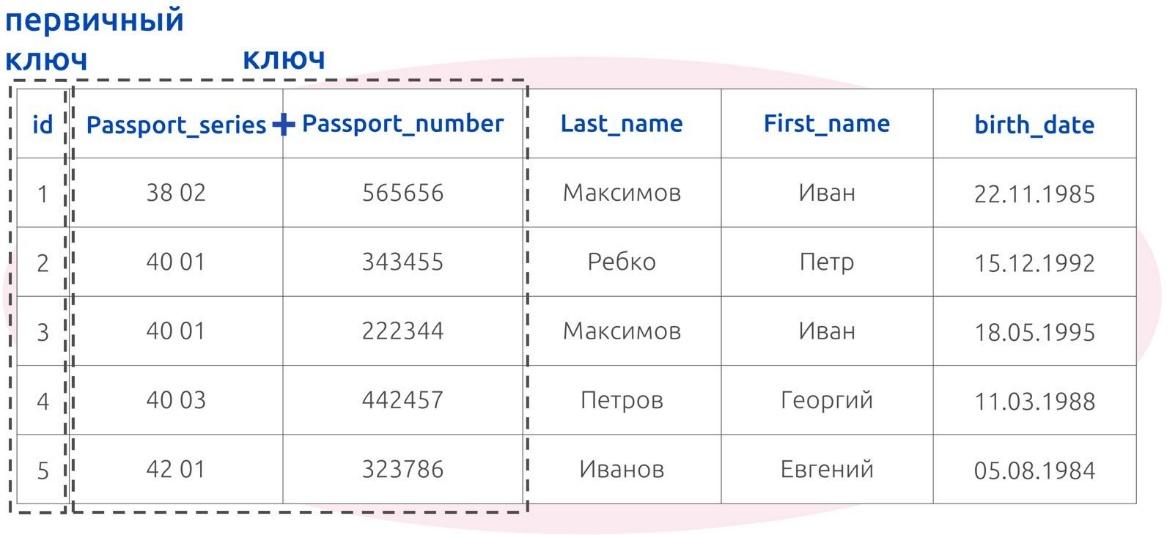
Реляционная модель основана на понятии отношения (relation). В чем отличие таблицы и отношения? Отношение можно представить как таблицу, содержащую множество значений. В отношении не может быть повторяющихся строк и порядок строк не имеет значения. Поэтому строки таблицы должны обязательно отличаться друг от друга значением хотя бы одного атрибута. Строки отношения часто называют кортежами, а столбцы – атрибутами. Вернемся к нашему примеру.

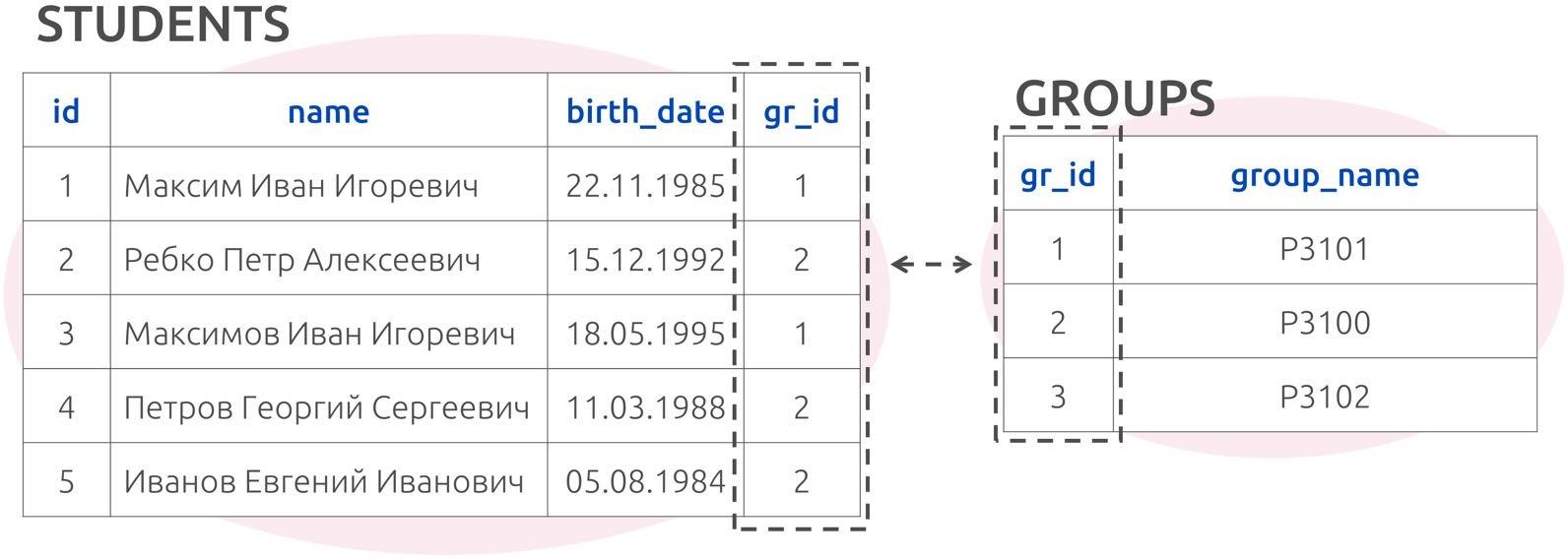
Обратите внимание, что в каждом из столбцов таблицы данные соответствуют определенному типу (это даты, строки, целые числа). В столбце id таблицы STUDENTS хранятся идентификационные номера, представленные целочисленными значениями. В столбце name хранятся имена и фамилии студентов в виде строковых значений. Подходящим типом данных для столбца с датой рождения birth\_date является тип дата. Таким образом, для каждого атрибута отношения (или столбца таблицы) существует некоторый допустимый набор значений. Такое множество значений называется доменом.

Применительно к отношениям стоит упомянуть также такие понятия, как заголовок отношения и тело отношения. Заголовок состоит из фиксированного множества атрибутов. Тело состоит из меняющегося во времени множества кортежей. В нашем случае (для отношения STUDENTS) заголовок состоит из атрибутов id, name, birth\_date. Тело отношения в данный момент состоит из 5 кортежей (в таблице 5 строк). Каждый кортеж соответствует определенному студенту. Степень отношения – это число его атрибутов. Степень отношения STUDENTS равна 3, так как отношение STUDENTS содержит три атрибута. Кардинальное число (мощность отношения) – это число его кортежей. Для таблицы STUDENT кардинальное число равно 5, так как в данный момент времени отношение содержит 5 кортежей. Кардинальное число отношения изменяется во времени.

Для идентификации кортежей используются так называемые ключи. Ключ – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый кортеж отношения. В отношении STUDENTS ключом является атрибут id – идентификационный номер студента, и эти номера не могут повторяться.

Следует сказать, что ключ не обязательно является одним атрибутом, он быть составлен из нескольких атрибутов (например, серия и номер паспорта), значения которых однозначно определяют кортеж в отношении.

Множество атрибутов ключа должно быть минимальным, и ключ не должен включать атрибуты, не обязательные для идентификации. Если в отношении есть несколько возможных вариантов ключей, то среди них определяют тот, который наиболее полезен при поиске. Его называют первичнымключом.

Важным элементом при хранении данных является организация связей между различными элементами. Особенностью реляционной модели данных является то, для представления связей также используются таблицы. Для организации связей используются ключи связываемых объектов. Продемонстрируем это на конкретном примере.

Добавим отношение GROUPS, которое будет представлять группы студентов. У этого нового отношения есть атрибут – gr\_id. Каждый кортеж имеет уникальное значение для данного атрибута. Чтобы связать студентов с их группами, нужно добавить новый атрибут в таблицу STUDENTS – назовем его gr\_id. Значения для этого атрибута будут соответствовать значениям атрибута gr\_id из отношения GROUPS. Таким образом, мы можем связывать между собой элементы разных таблиц.

Кроме самих данных, при создании таблиц можно описывать правила, которым должны соответствовать хранимые данные. Например, в таблицу STUDENTS нельзя добавить значение gr\_id, которому нет соответствия в таблице GROUPS.

# **6. Диаграммы сущность-связь (ER-диаграммы). Сущности**.

Процесс создания базы данных всегда начинается с описания предметной области. Первое, что нужно выяснить – на какие запросы пользователей предстоит ответить информационной системе. Необходимое описание структуры информационных объектов или понятий, которые будут храниться, нужно понять, как они будут связаны между собой, и какие требования предъявляются к допустимым значениям данных.

Проект базы данных надо начинать с анализа предметной области и выявления требований к ней отдельных пользователей (например, сотрудников организации, для которых создается база данных).



Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, проектировщик сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных, называют концептуальным уровнем.

Концептуальный уровень описывает полную логическую структуру хранимых данных. На последнем шаге проектирования создается внутренняя модель данных, которая, которая описывает представление концептуальной схемы в контексте выбранной СУБД.

Для представления данных на концептуальном уровне нужно удобное средство, которое позволяет наглядно отобразить структуры данных предметной области и связи между ними. Подходящим средством для этого являются диаграммы модели «Сущность-Связь», или «Entity-Relationship». Впервые описание таких диаграмм было предложено Питером Ченом в 1976 г.



В дальнейшем было разработано много подобных моделей, имеющих похожие обозначения. Все варианты диаграмм сущность-связь исходят из одной идеи – рисунок всегда нагляднее текстового описания. Рассмотрим построение диаграмм в нотации Баркера.

Элементы предметной области могут быть представлены как совокупность объектов и связей между ними. Основными элементами диаграмм являются сущности, под которыми понимают хранимые объекты или понятия, атрибуты, которые представляют свойства сущностей, и связи между сущностями.

Давайте для примера рассмотрим базу данных Экзаменационная сессия. В базе данных должна храниться информация о студентах, сданных ими экзаменах по определенным предметам. Студент характеризуется номером зачетки, ФИО, датой рождения. Может иметь несколько телефонов. Все студенты распределены по учебным группам. В каждой группе выбран ее староста.

Для группы назначаются экзамены по определенным курсам, экзаменаторы, даты проведения экзаменов и аудитории. За каждый сданный экзамен студенту ставится оценка.

Ограничения: номер зачетки – положительное число, в ФИО не бывает цифр, оценка бывает от двух до пяти.

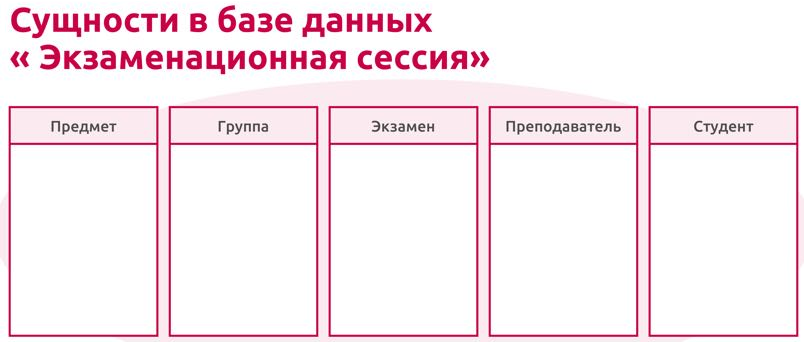
**Сущность**

Сущность – это реальный или абстрактный объект определенного вида, который может существовать независимо от других. Иногда сущностью называют класс однотипных объектов, информация о которых будет храниться в базе данных.

У каждой сущности должно быть уникальное имя, к одному и тому же имени всегда должна применяться одна и та же интерпретация. Набор экземпляров сущностей образует множество.

Сущности в ER-диаграммах изображаются с помощью прямоугольников, где внутри наверху указывают название сущности. Название сущности обычно имя существительное.

В базе данных Экзаменационная сессия можно выделить сущности Студент, Преподаватель, Экзамен, Группа, Предмет.



**Атрибут**

Атрибут – поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей.

Каждый атрибут имеет свой тип данных. Например, у сущности Студент атрибутами могут быть имя, фамилия, серия и номер паспорта, номер зачетки, дата рождения. Группа будет описана номером и специализацией. Преподаватель – ФИО и должность.

**Идентификация сущностей**

Экземпляры сущностей должны быть отличны друг от друга, а значит нужны идентификаторы сущностей. Экземпляры сущностей не могут быть дубликатами, поэтому как минимум, значения всех атрибутов образуют уникальную комбинацию.



Но чаще всего бывает так, что не все атрибуты нужны для идентификации сущности. Находятся один или несколько атрибутов, необходимых и достаточных для этого. Например, чтобы точно указать студента, нужно знать его номер зачетки, а дата рождения и адрес для этого не нужны.

На ER-диаграммах отображаются и ключи. Напомним, что ключом называют один или несколько атрибутов сущности, по которым можно однозначно идентифицировать сущность. Если ключом является один атрибут, то он называется простым. Ключ, составленный из нескольких атрибутов, называют составным. Иногда у сущности может быть несколько возможных ключей.

Примерами для сущности студент может быть простой ключ номер зачетки, или составной ключ – комбинация атрибутов серия и номер паспорта. При наличии нескольких возможных ключей один из них, наиболее часто используемый при поиске, называют первичным ключом. Хорошим тоном является выбор в качестве первичного ключа числового, короткого и неменяющегося в процессе жизни объекта атрибута. На диаграмме атрибуты, входящие в состав первичного ключа, подчеркивают одинарной линией.

Есть несколько возможных способов идентификации:

* естественные ключи;
* «по положению» (географическое, по порядку, по времени);
* суррогатные.

Обычно сущность обладает атрибутами, характеризующими ее свойства, которые идентифицируют каждый экземпляр сущности. Такие ключи называют естественными.

Ни один естественный идентификатор не может быть абсолютно надежен, поэтому для системы удобнее суррогатные ключи. Но они бесполезны при поиске.

Не всегда среди атрибутов сущности легко подобрать такой атрибут, а иногда его просто нет. Например, у сущности Предмет – название в качестве ключа нельзя взять, т.к. могут быть предметы с одинаковым названием, вести предмет могут разные преподаватели.

Если нет естественного ключа, придумывают искусственный – «суррогатный». Для сущности Предмет можно придумать такой числовой идентификатор.

При построении ER-диаграммы иногда трудно принять решение, где атрибут сущности, а где другая отдельная сущность.

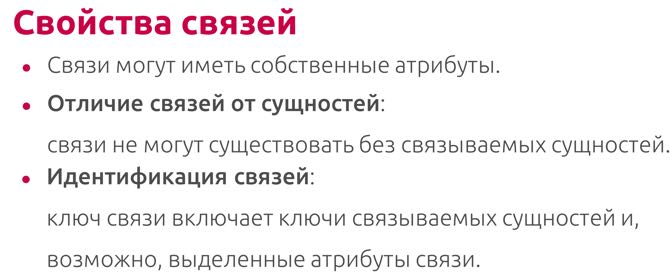
Абсолютное различие между типами сущностей и атрибутами отсутствует. Атрибут является таковым только в связи с типом сущности. В другом контексте атрибут может выступать как самостоятельная сущность. Например, для человека адрес может быть атрибутом, если нас интересует только место прописки.

Но адрес может быть и самостоятельным объектом, имеющим свои характеристики Город, улица и номер дома. Тогда мы сможем узнать не только где человек живет, но и отвечать на более сложные запросы: сколько человек живет в таком-то городе или на определенной улице. Поэтому выбор между возможным атрибутом или новой сущности нужно делать исходя из возможных запросов к данным.

# **7. Диаграммы сущность-связь (ER-диаграммы). Связи.**

Мы разобрались с проектированием сущностей, теперь рассмотрим отношения, которые возникают между сущностями – их называют связями.

Связь – ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. А так как в реальных базах данных нередко содержатся сотни или даже тысячи сущностей, то теоретически между ними может быть установлено более миллиона связей.

Наличие такого множества связей и определяет сложность инфологических моделей. У связей, как и у сущностей, есть свое название – чаще всего глагол. Например, Студент состоит в Группе, Преподаватель принимает Экзамен.

Основное отличие связей от сущностей: связи сами по себе не имеют смысла и не могут существовать без связываемых сущностей.

Связи, как и сущности, нуждаются в идентификации. Идентификатор связи включает ключи связываемых сущностей и, возможно, некоторые выделенные атрибуты связи.

Приведем пример атрибута связи. Например, если студент сдал преподавателю экзамен по некоторому предмету, то оценка, полученная за экзамен, будет атрибутом связи, а не студента, предмета и тем более преподавателя.

**Свойства связей**

Связи могут отличаться своими характеристиками, наиболее важными из которых являются:

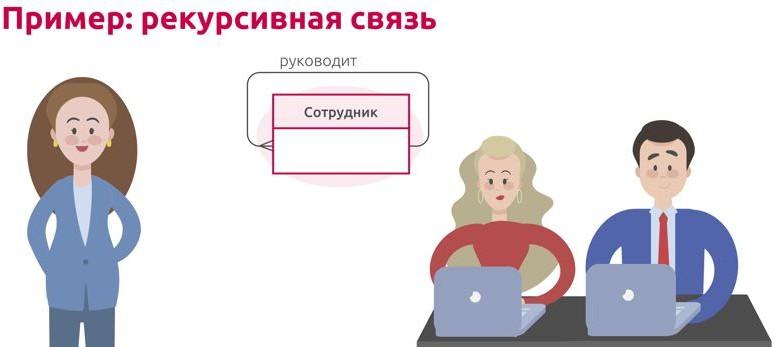
* размерность (степень связи);
* мощность (кардинальность);
* модальность

Степень связи — это количество сущностей, между которыми возникает связь. Самая типовая связь – бинарная, когда связь определена между двумя сущностями.

Если три сущности образовали связь, как в примере с экзаменом: студент, преподаватель и предмет, то связь называется тернарной. Бывают унарные, или рекурсивные связи, когда связь возникает внутри одной сущности – например, один сотрудник руководит другими. В общем случае связь между n сущностями называется n-арной.

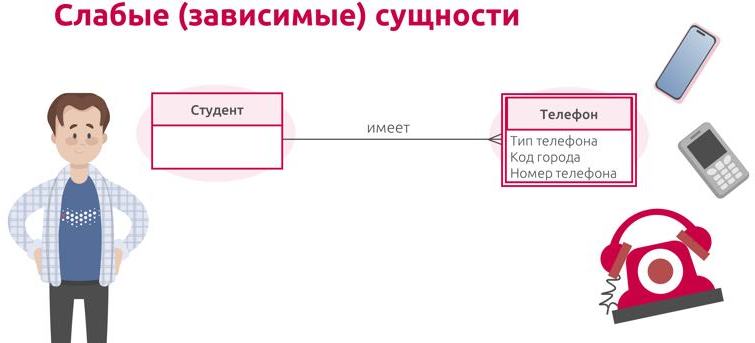
Рассмотрим бинарную связь между студентом и группой. Между двумя сущностями может быть определено несколько наборов связей. Например, студент может состоять в группе, а может являться старостой группы, это разные связи между одними и теми же сущностями.



Примером тернарной связи может быть экзамен: в этой связи участвуют три сущности – Студент, Преподаватель и Предмет. Оценка является собственным атрибутом связи. Часто для удобства вместо n-арных связей используют только бинарные. Для этого нам придется вместо связи экзамен ввести сущность Экзамен.

Приведем пример рекурсивной связи. Например, рассмотрим сущность Сотрудник. Одни сотрудники могут руководить другими, в то же время оставаясь сотрудниками. Получается, что сущность вступает в связь с другими экземплярами той же сущности.

Бывают «правильные» (независимые) и «слабые» сущности. Слабые сущности не могут существовать без связи с другими (сильными) сущностями, или находятся в зависимости от других сущностей.

Давайте представим, что нам нужно хранить информацию о телефонах студента. На первый взгляд, может показаться, что телефонный номер — это просто атрибут. Но у студента может быть несколько телефонных номеров – домашний, мобильный, а может быть, и служебный, и мобильных может быть несколько. Тогда правильнее выделить телефон в отдельную сущность. У сущности Телефон, кроме номера, могут быть еще атрибуты: тип телефона, код города и пр. У нескольких людей телефонные номера могут совпасть, – например, домашний и рабочий. Формально чтобы отличать их, нужно придумать какой-то идентификатор, или суррогатный ключ. Но навряд ли нам будет нужен телефонный номер сам по себе, без информации о том, кто его владелец.

В таком случае информация о слабой сущности хранится вместе со связью, которая указывает на независимую сущность, и суррогатный ключ не нужен. Такие сущности и связи на диаграммах иногда очерчивают двойной линией.

Мощность обозначает максимальное количество экземпляров одной сущности, связанных с экземплярами другой сущности.

Делятся на три вида в зависимости от количества участвующих в них экземпляров сущностей.

* один-к-одному 1:1
* один-ко-многим 1:N
* многие-ко-многим M:N



**Мощность** обозначает максимальное количество экземпляров одной сущности, связанных с экземплярами другой сущности. Мощность связи на диаграммах отражает конец линии, соединяющий сущности, который называют «концом связи». Связь один обозначает прямая линия на конце связи, связь много – перевернутая стрелка, которую часто называю вороньей лапкойCrow’s foot.

Иногда в СУБД на диаграммах отображают связь «один» в виде ключа, а связь «много» в виде знака бесконечность.

**Связи один-к-одному**

Наиболее просто описываемая, но реже всего встречаемая в природе, это связь вида один к одному. Она возникает, когда ровно один экземпляр одной сущности вступает в связь с одним и только одним экземпляром другой сущности.

В качестве примеров можно привести связь между столичным городом и страной, где город является столицей, ректор – вуз для связи Руководит. Ромб связи и прямоугольник объекта соединяются прямой линией.

**Связи один-ко-многим**

Один-ко-многим: каждому экземпляру первой сущности может соответствовать несколько экземпляров другой сущности, но каждому экземпляру второй сущности соответствует не более одного экземпляра первой сущности.

Например: в каждом отделе может быть множество сотрудников, но каждый сотрудник работает только в одном отделе.

**Связи многие-ко-многим**

Встречаются также связи вида многие-ко-многим: каждому экземпляру первой сущности может соответствовать несколько экземпляров другой сущности, и наоборот.

**Модальность связей**

Последнее свойство связей, которое мы рассмотрим, называется модальность Необязательные связи (условные) – модальность «может» означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром.

* Пассажир может иметь билет.
* Человек может иметь автомобиль.
* Студент может сдать экзамен.

Обязательные – модальность «должен» означает, что экземпляр одной сущности обязан быть связан не менее чем с одним экземпляром другой сущности.

Каждый курс лекций должен иметь преподавателя.

У каждой кафедры должен быть заведующий.

Каждый билет выписан на имя конкретного пассажира.

Если рассмотреть связь между студентом и экзаменом, то тут должна быть модальность «может», т.к. студент может не сдать ни одного экзамена, и может найтись экзамен, который никто еще не сдал.

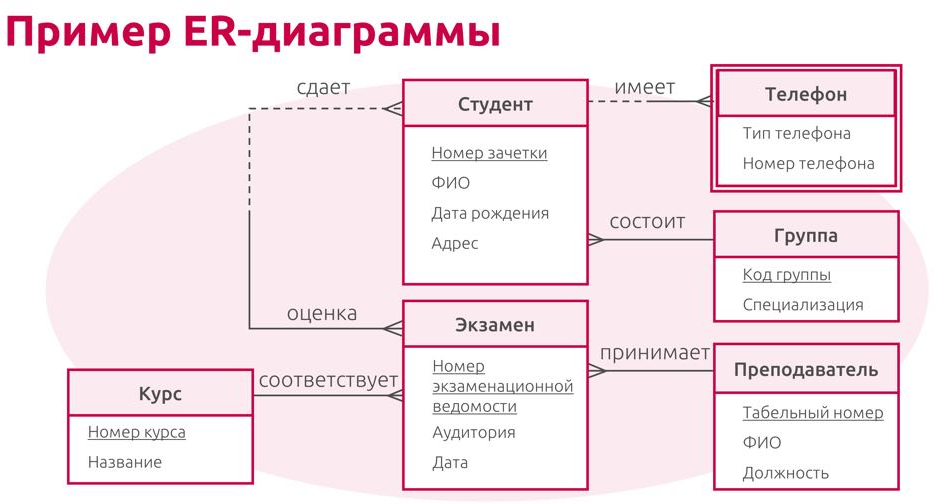
Например: каждый сотрудник может участвовать в нескольких проектах, и в каждом проекте участвуют несколько сотрудников. Связь студент-курс: каждый студент изучает множество курсов, и каждый курс изучается многими студентами.

Таким образом мы рассмотрели все элементы ER-диаграмм.

Итак, чтобы построить диаграмму, нужно сделать следующие шаги:

* Определить сущности.
* Определить атрибуты сущностей.
* Определить первичные ключи.
* Определить отношения между сущностями.
* Определить кардинальность.
* Нарисовать ER-диаграмму.
* Проверить ER-диаграмму.

Вспомним предметную область, которую мы рассматривали: Экзаменационная сессия. В нашей базе данных должна храниться информация о студентах, сданных им экзаменах по определенным предметам. Студент характеризуется номером зачетки, ФИО, датой рождения. Может иметь несколько телефонов. Все студенты распределены по учебным группам. Для группы назначается экзамен по определенному предмету, назначается экзаменатор. Указывается дата экзамена и аудитория. За каждый сданный экзамен студенту ставится оценка. Ограничения: номер зачетки – положительное число, в ФИО не бывает цифр, оценка бывает от двух до пяти. Студент состоит в одной учебной группе, в один день может сдать только один экзамен.



# **8.** **Преобразование ER-модели в реляционную базу данных**

Мы научились определять структуры данных в терминах модели сущность-связь при помощи ER-диаграмм. Следующий шаг после этого выбор конкретной СУБД, поддерживающей некоторую модель данных, и отображение диаграмм на структуры, принятые в выбранной СУБД. Если выбором будет реляционная СУБД, то необходимо преобразовать структуры данных ER-модели в отношения.

Давайте вспомним, что отношения — это объекты базы данных, в которых хранится вся доступная пользователю информация.

Отношение имеет заголовок, или схему с описанием структуры таблицы, состоящий из атрибутов, или названий столбцов, и тело отношения, составленное из строк, или кортежей – в них содержатся непосредственно данные. Значения каждого атрибута содержатся в допустимом множестве возможных значений, которое называется доменом.

Схема отношения – упорядоченное множество атрибутов. Каждый атрибут имеет имя, уникальное в рамках отношения, и для каждого из которых задан тип данных.

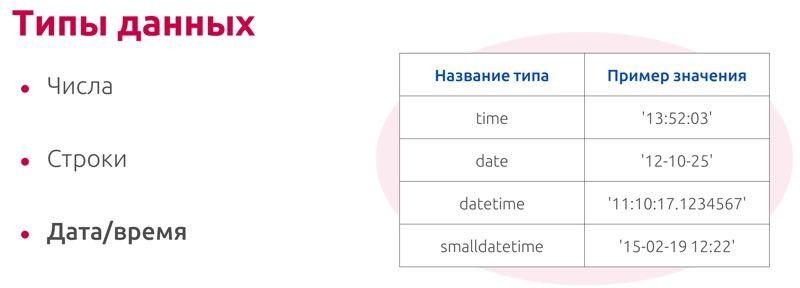


**Типы данных**

В разных СУБД могут быть некоторые отличия в поддерживаемых типах данных и в их наименовании, но в любой СУБД можно хранить числа, строки и информацию о дате и времени.

Числовые типы данных могут быть целыми и дробными, заданы с разной точностью, т.е. можно явно указывать количество знаков в целой и дробной частях числа.

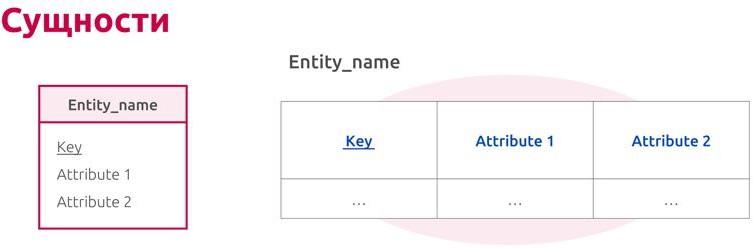
Символьные данные могут быть представлены строками фиксированной или переменной длины. При описании типа в скобках указывают размер. Для строк переменной длины этот размер задает максимально возможную длину строки, а для строк фиксированной длины – точную длину строки. Если не указать размер, то в некоторых СУБД длина поля будет равна одному символу, в некоторых – строке переменной длины максимально возможного размера.

Тип данных дата и время также используется в любой СУБД, но именно для этого типа чаще всего проявляются особенности реализации. Для даты и времени есть много разных форматов, например, хранение в одном типе даты и времени или по отдельности.

В ряде СУБД предусмотрена специальные типы данных для хранения данных в других форматах: пространственных и иерархических данных, данных в формате XML и JSON и другие.

Для каждой сущности, описанной в ER-модели, в базе данных нужно создать отдельную таблицу. Имя сущности становится именем таблицы. Каждый атрибут становится столбцом. Для имен таблиц и столбцов принято использовать латинские буквы. Для каждого столбца определяется подходящий тип данных. Идентификатор сущности превращаются в ключ таблицы.

Напомним, что возможным ключом называют минимальный набор атрибутов, по которому можно определить все остальные. В качестве первичного ключа выбирают один из возможных ключей, обычно чаще всего используемый при поиске.

Например, в базе данных Экзаменационная сессия нужно создать таблицу для сущности Студент с атрибутами номер зачетки, ФИО, серия и номер паспорта, дата рождения и адрес. Создадим таблицу STUDENT с полями:

* Номер зачетки – StudentId – целое число;
* ФИО – StudentName – строка переменной длины до 100 символов;
* Серия паспорта – Pass\_s – строка до 4 символов;
* Номер паспорта – Pass\_num – строка до 6 символов;
* Дата рождения – BirthDate – формат Дата;
* Адрес – Address – строка переменной длины до 100 символов.

Возможные ключи – номер зачетки и серия и номер паспорта. Выберем в качестве ключа поле StudentId.

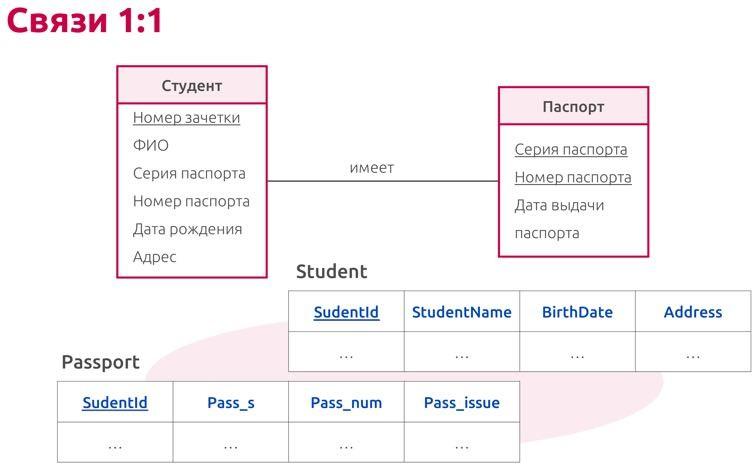
Создадим таблицу для сущности Группа назовем ST\_GROUP. Полями будут:

* Код группы – GroupCode – строка до 32 символов;
* Специализация – Specialization – строка переменной длины до 100 символов.

Ключ – GroupCode.

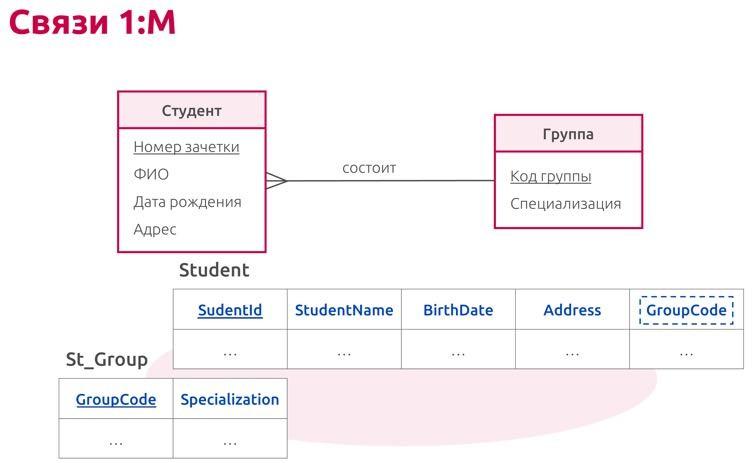
Связи также хранятся в отношениях. Для отображения связи необходимы ключевые атрибуты объектов, участвующих связи, и атрибуты связи, если таковые есть. Связи вида один к одному принято хранить в одной таблице, столбцы которой соответствуют атрибутам обеих сущностей.

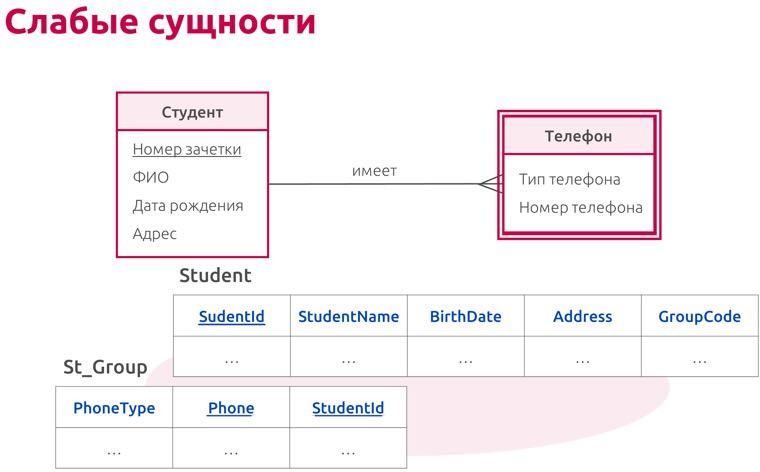
Рассмотрим простейший пример связи вида 1:1. Предположим, что требуется хранить информацию о паспортных данных студентов. Тогда мы бы ввели отдельную сущность – номер паспорта с атрибутами серия, номер и дата выдачи.

У каждого студента ровно один паспорт, и каждый паспорт соответствует ровно одному человеку. В таком случае можно было эту информацию можно представить в одной таблице Студент с атрибутами Идентификатор студента, ФИО, Серия паспорта, Номер паспорта, Дата выдачи паспорта, Дата рождения, Адрес. Ключом отношения может быть Идентификатор студента или серия и номер паспорта.

Иногда, наоборот, атрибуты одной сущности хранят в двух разных таблицах. Например, из соображений безопасности мы могли бы хранить информацию о номерах паспортов студентов в отдельной таблице.

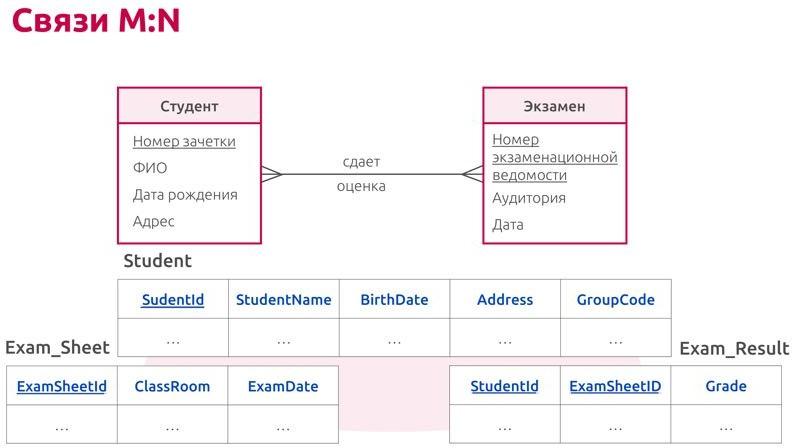
Тогда в одной таблице были бы поля Номер зачетки – StudentId, StudentName – имя студента, дата рождения и адрес. Ключ – StudentID. В другой таблице хранились бы Pass\_s серия паспорта и Pass\_num номер паспорта и StudentId – ключ.

Если между сущностями связь вида один ко многим, как например, между сущностями Группа и Студент – в каждой группе учится один или более (много) студентов, и каждый студент учится ровно в одной группе, то реализовать эту связь можно следующим образом: в таблицу сущности с концом связи «много» добавляем ключ сущности с концом связи «один».



Также хранится информация о связях с слабыми сущностями. У студента может быть несколько телефонных номеров, и у некоторых номера могут совпасть, например, домашние.

Мы добавим к таблице с информацией о номерах телефона ключ сущности Студент – StudentID. Ключ отношения будет составным: Phone и StudentID.

Для хранения связей вида многие ко многим приходится создавать отдельную таблицу, которая служит для отображения связи. Схема данного отношения составляется из ключевых атрибутов объектов, участвующих в связи, и атрибутов связи, если таковые есть. Например, связь между сущностью Студент и Экзамен имеет вид многие-ко-многим. За экзамен студент получает оценку – атрибут оценка является свойством связи. Создадим таблицу Результат Экзамена с полями Идентификатор студента, Идентификатор экзамена и Оценка – целые числа.

# **9. Нормализация в базах данных**

Проектирование данных – очень важная задача. И плохо спроектированная схема создает сложности и с доступом к данным, и пониманием отдельных объектов и связей, и с дальнейшим расширением структуры. Для улучшения схемы данных на помощь придет нормализация.

Нормализация базы данных — это метод проектирования реляционных данных, который помогает изменить структуру таблиц для четкого представления информации и взаимосвязей, без избыточности и потери данных. Но сначала давайте рассмотрим проблемы, которые могут возникнуть, если структура данных оказалась не оптимальна.

* Трудности с извлечением некоторых данных
* Избыточность
* Аномалии изменения
* Аномалии удаления
* Аномалии добавления

В одном столбце оказались данные сразу о нескольких школьниках, которые учатся в одной классе. В такой структуре легко будет выполняться, например, такие запросы:

* Какие классы есть в школе?
* Сколько классов в школе?
* Кто учится в 4а классе?

Но вот примеры других запросов, ответы на которые получить непросто: по фамилии школьника определить его класс, сосчитать количество школьников каждом классе, или общее количество школьников во всех классах.

Мы видим явную избыточность данных – фамилия классного руководителя и его телефон повторяются многократно. Если в таблице всего три строки, то проблема не кажется большой. Но представим, что у нас сотни школьников, и у десятков из них повторяются одни и те же фамилии классного руководителя и телефоны. Тогда избыточность превратится уже в весомые цифры.

Но проблема не только в том, что мы дублируем одну и ту же информацию. При изменении значения данных нам придется изменять данные во множестве строк. Например, у классного руководителя Певцова изменился телефон.

Другие аномалии могут возникнуть при удалении данных.



Например, если в таблице Школьники мы удалим Черняева из 6б, то исчезнет информация о классном руководителе класса в целом. Это называется аномалией удаления.

И при добавлении могут возникнуть аномалии. Давайте представим, что в 1б класс назначили классного руководителя, но пока в ни одного школьника в класс не зачислили. Как добавить информацию о классном руководителе? Если оставить поля Табельный номер и ФИО школьника пустыми, то эту пустую строку надо будет удалить при добавлении школьников в класс.

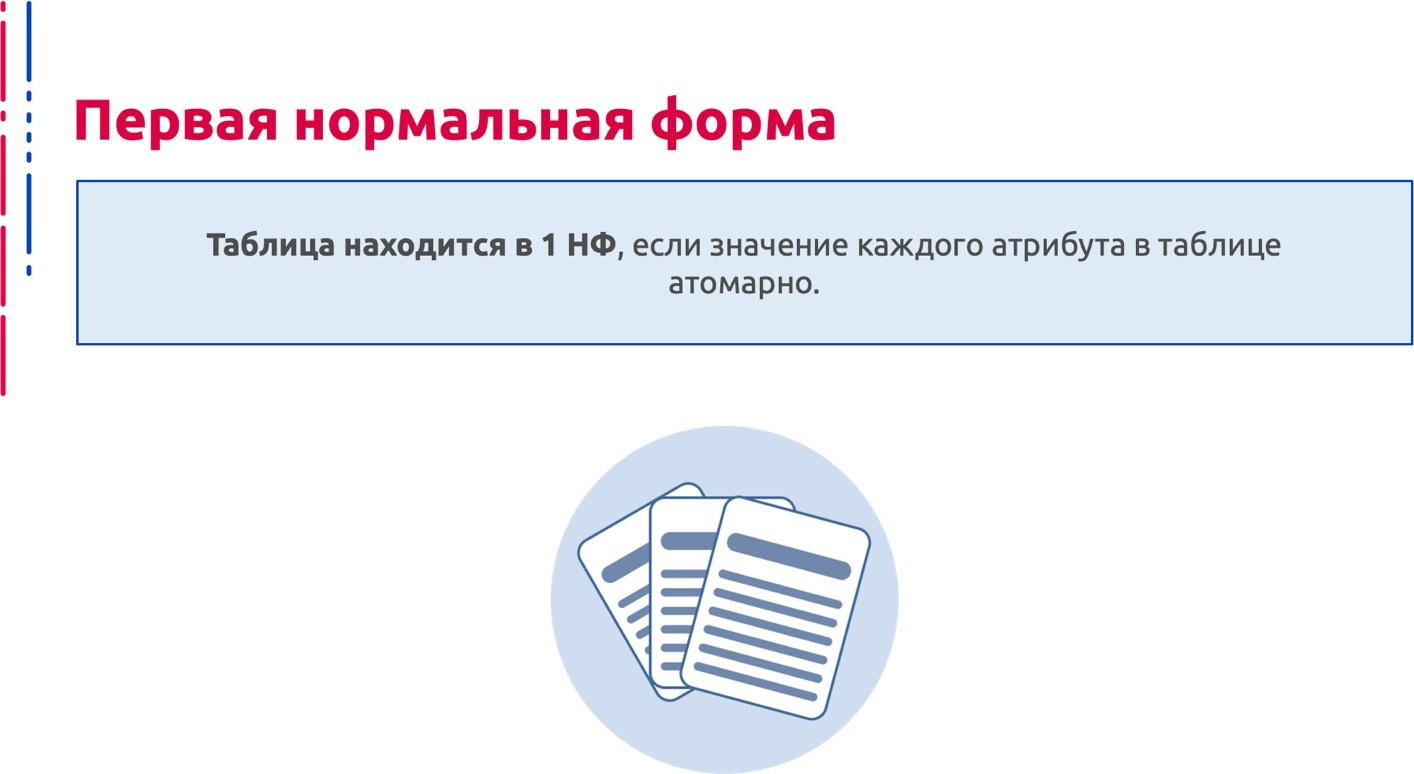


Чтобы избежать всех перечисленных аномалий и оптимизировать структуру таблиц были придуманы нормальные формы отношений. Нормальные формы – это правила, которым должны соответствовать таблицы. Нормальные формы имеют номера, начиная с первой, и каждая следующая усиливает требования предыдущей.

Чтобы понять, что в нашей схеме базы данных все хорошо, надо проверить, соответствуют ли таблицы нормальным формам. И если не соответствуют, то преобразовать схему так, чтобы НФ выполнялись.

В процессе нормализации данные преобразуют, чтобы они занимали меньше места, а поиск по элементам был быстрым и результативным. Для этого создают дополнительные таблицы и связывают друг с другом ключами — колонками, в которых нет повторяющихся элементов.

Нормализация - это итеративный процесс. То есть сначала надо привести таблицу к 1 НФ, и затем таблицу, которая находится в 1 НФ, можно приводить ко 2НФ. Каждый последующий шаг разбивает таблицу на более легкую в управлении информацию, чем повышается общая логичность системы и простота работы с данными.

Не всегда базу данных приводят в соответствие всем нормальным формам, так сказать до «идеального» состояния, так это может привести к созданию очень большого количества связанных таблиц. Но обычно до 3 НФ данные нормализуют.

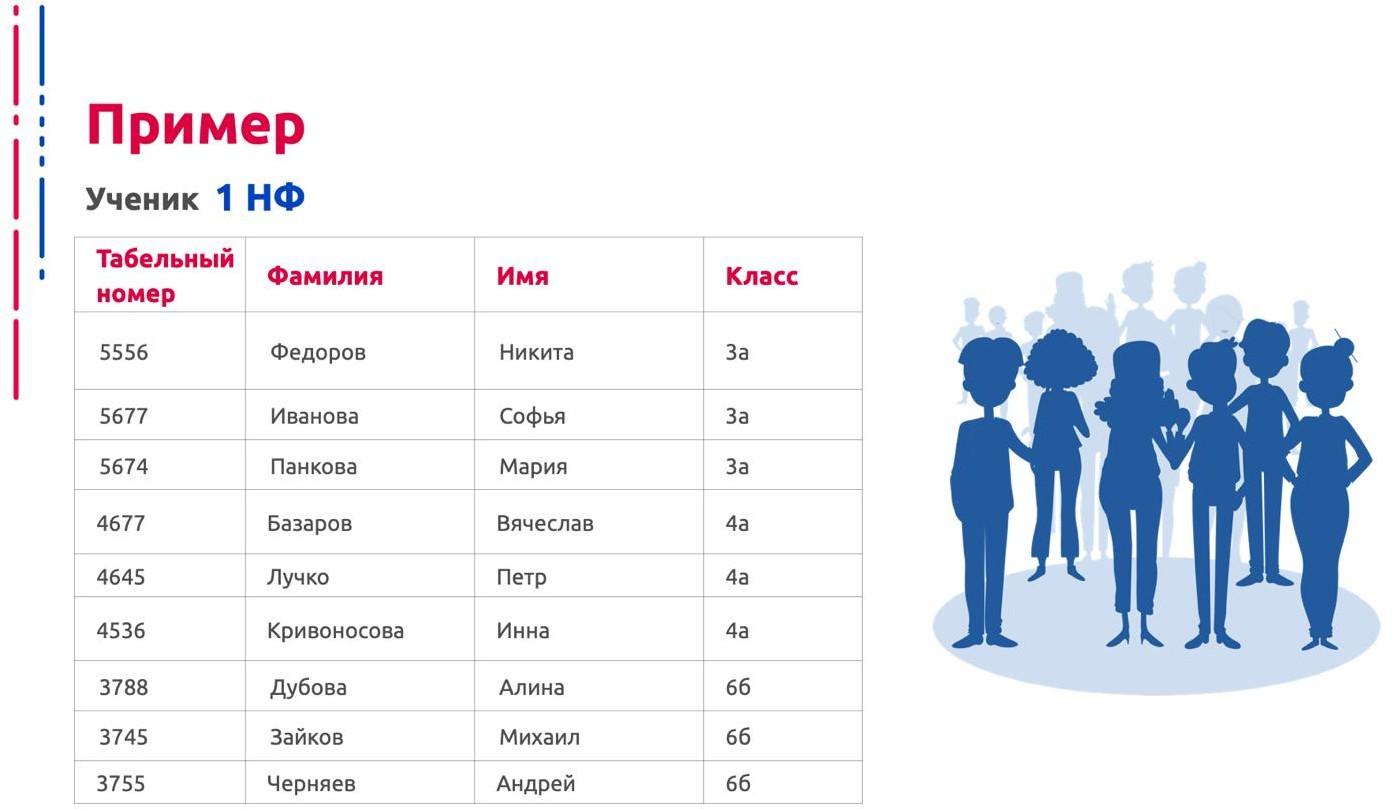
1 НФ – значение каждого атрибута в таблице должно быть атомарно. То есть значение атрибута не может быть списком, множеством, вложенной таблицей.

Рассмотрим еще раз таблицу про учебные группы.

1 НФ нарушена, так как в столбце список группы содержатся множества значений, каждое из которых может представлять интерес для поиска и анализа.

Как можно преобразовать структуру таблицы? Создадим таблицу из трех атрибутов – номер группы, номер зачетки. ФИО студента. Теперь для каждого свойства объекта есть отдельная ячейка, или значение атрибута.

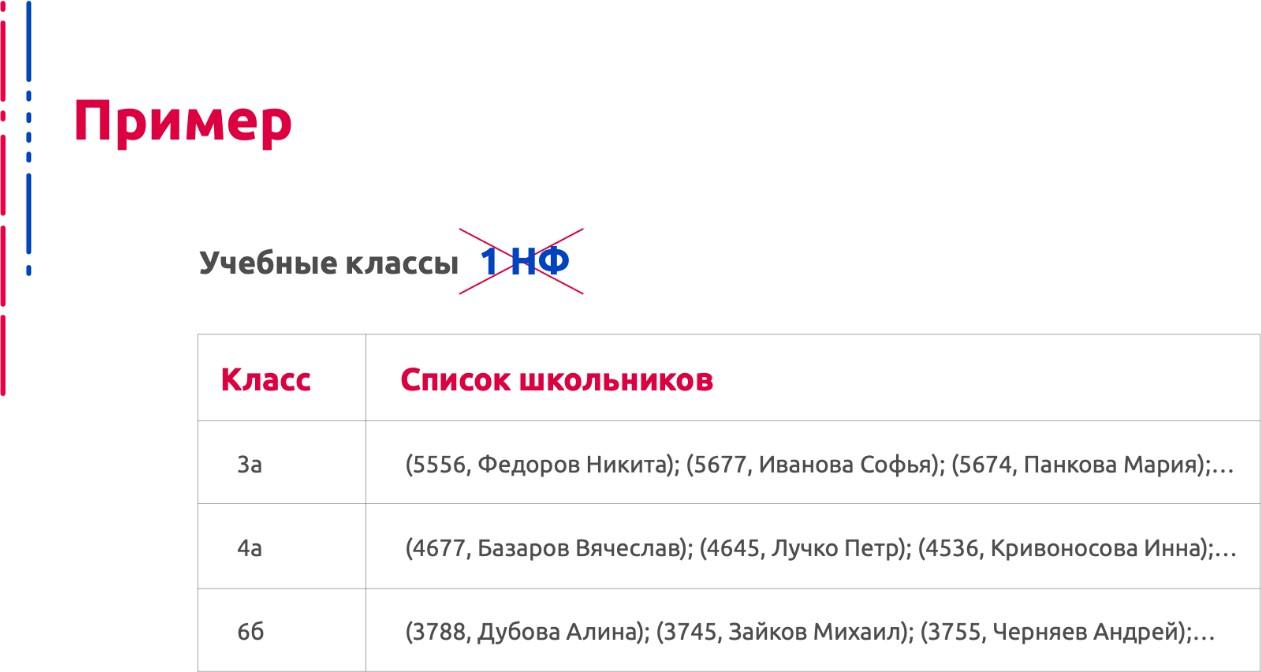
Эта таблица находится в 1 НФ.

Давайте посмотрим еще на столбец ФИО, который хранит фамилию и имя школьника. Надо ли этот столбец разбить на два и хранить фамилии и имена отдельно?

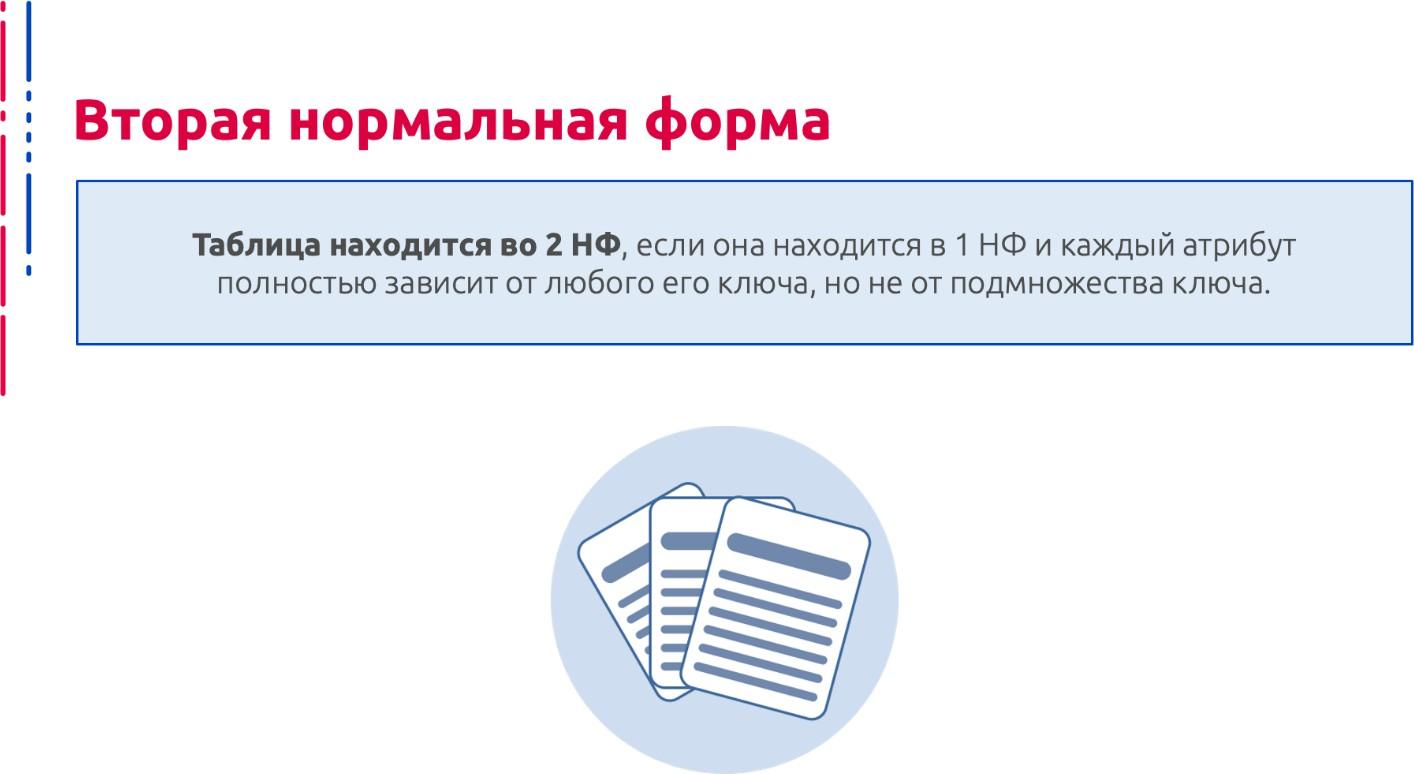
Это зависит от запросов, которые мы будет выполнять. Если нам нужно подсчитывать, например, количество однофамильцев, или искать учеников только по имени, то столбец ФИО нужно разбить на два отдельных. Если мы будем искать школьников, задавая фамилию или вместе фамилию и имя, можно оставить столбец ФИО.

Теперь перейдем ко 2 НФ. Для этого нам потребуется ввести понятие зависимости атрибутов (функциональной зависимости).

Например, по ИНН точно можно определить адрес проживания. По регистрационному номеру и региону можно определить марку автомобиля и год выпуска. А зная серию и номер паспорта, можно узнать ФИО и прописку.

Через функциональную зависимость можно дать определение ключа таблицы.





Рассмотрим еще один пример

Здесь ключом, то есть атрибутом, от которого зависят все остальные, являются табельный номер и название предмета. Зная эти атрибуты, мы узнаем ФИО студента и оценку. При этом чтобы узнать оценку, нужны оба ключевых атрибута. А для того, чтобы узнать ФИО студента, нам нужен только табельный номер. Такая зависимость от ключа называется неполной, или частичной.

Как можно улучшить такую таблицу Учебный год? Догадаемся пока интуитивно – надо разбить на две таблицы. В одной таблицы будут столбцы Табельный номер, ФИО и Класс, а в другой – Табельный номер, предметы и оценки.

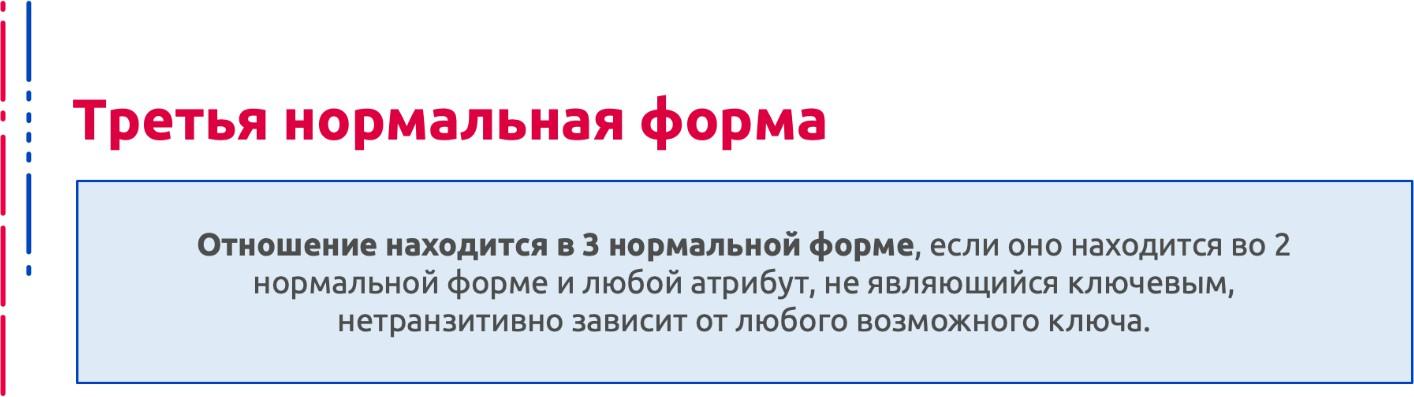
# **10. 3 нормальная форма**

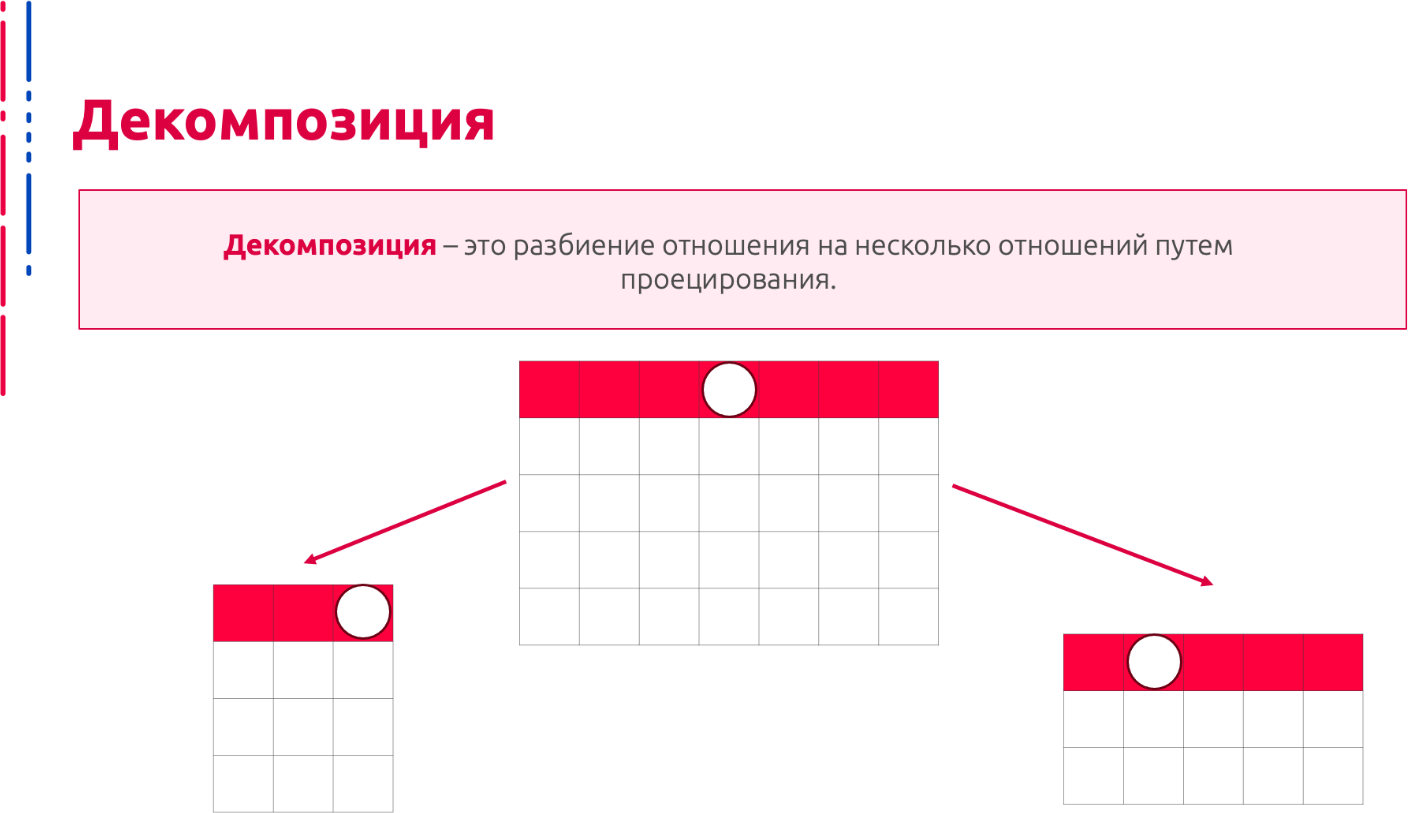
Давайте еще раз рассмотрим таблицу Школьники

Зная Табельный номер, мы узнаем ФИО школьника, а также класс, а по номеру класса найдем классного руководителя и его телефон. Получается, что ключ – это табельный номер, так как по нему можно определить все остальные атрибуты. Но на самом деле, чтобы найти классного руководителя, нам нужен класс.

Вспомним определение транзитивной зависимости.

Определение: Функциональная зависимость A->C называется транзитивной, если существует такой атрибут B, что имеются функциональные зависимости A->B и B->C и отсутствует функциональная зависимость C->A.

В нашем отношении Школьники атрибуты Классный руководитель и Телефон зависят от класса, а класс – от табельного номера. При этом класс не зависит от Классного руководителя, а тем более от его телефона. Получается, что атрибуты Классный руководитель и Телефон транзитивно зависят от атрибута Табельный номер, т.е. через другой атрибут (в данном случае класс).

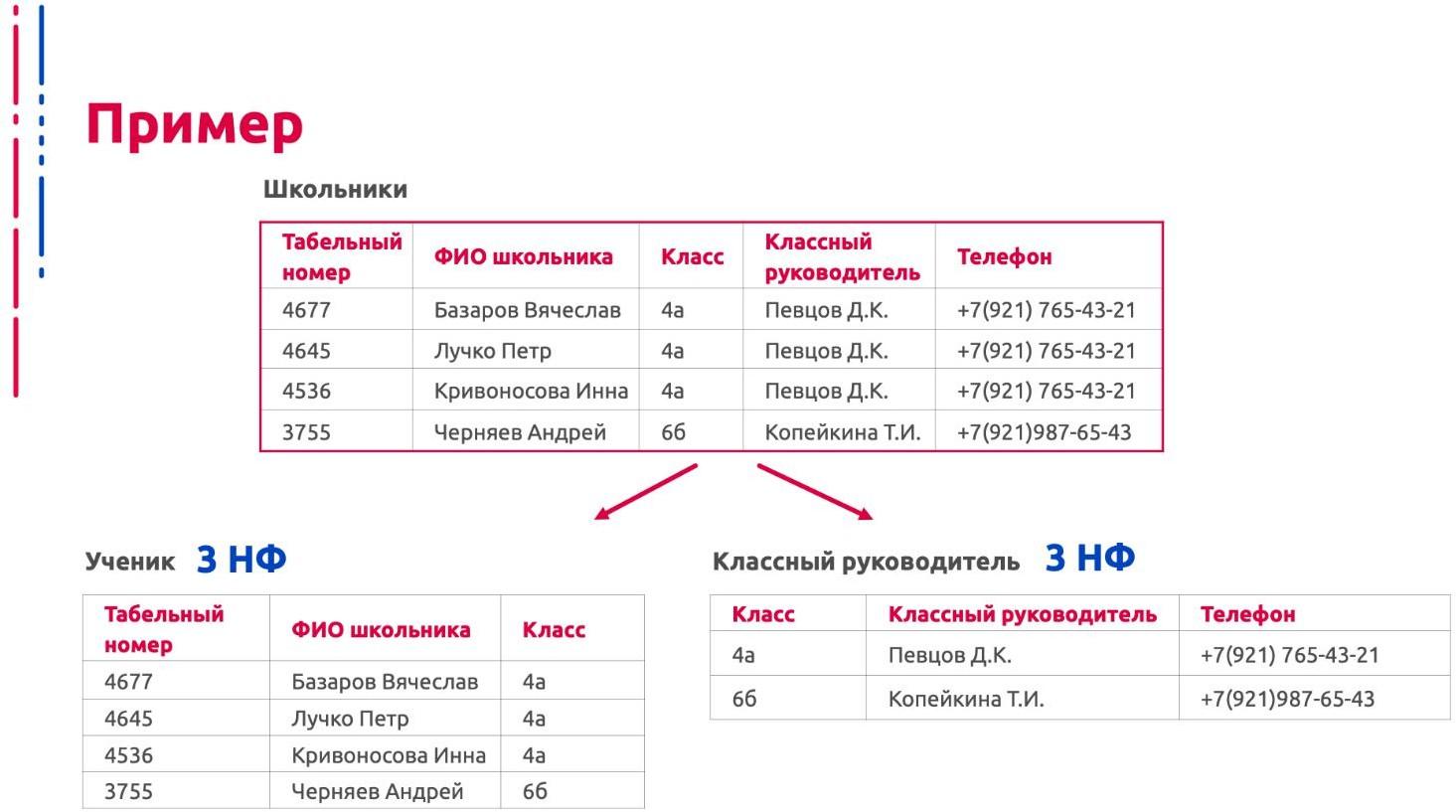
Как привести к 3 НФ отношение Школьники? Снова разбить на две таблицы. Давайте уже действовать не интуитивно, а по правилу. Нам надо разбить отношение на несколько (обычно два) отношений, чтобы избежать избыточности, но не потерять никаких данных, т. е. чтобы исходное отношение можно было восстановить.

Разбиение отношения на несколько отношений путем проецирования называется декомпозицией. Рассмотрим декомпозицию этой таблицы на две. Для этого надо разбить множество атрибутов на две части – одна часть атрибутов станет атрибутами первой таблицы, вторая часть – второй. Эти множества атрибутов обязательно должны пересекаться, то есть иметь хоть один общий атрибут, который войдет в обе новые таблицы. Именно с помощью этого атрибута можно будет потом восстановить исходное отношение методом естественного соединения, то есть соединения по равенству одноименных атрибутов.

Теперь давайте разберемся, как же разбивать исходное множество атрибутов отношения на части.



Попробуем произвести декомпозицию таблицы Школьники

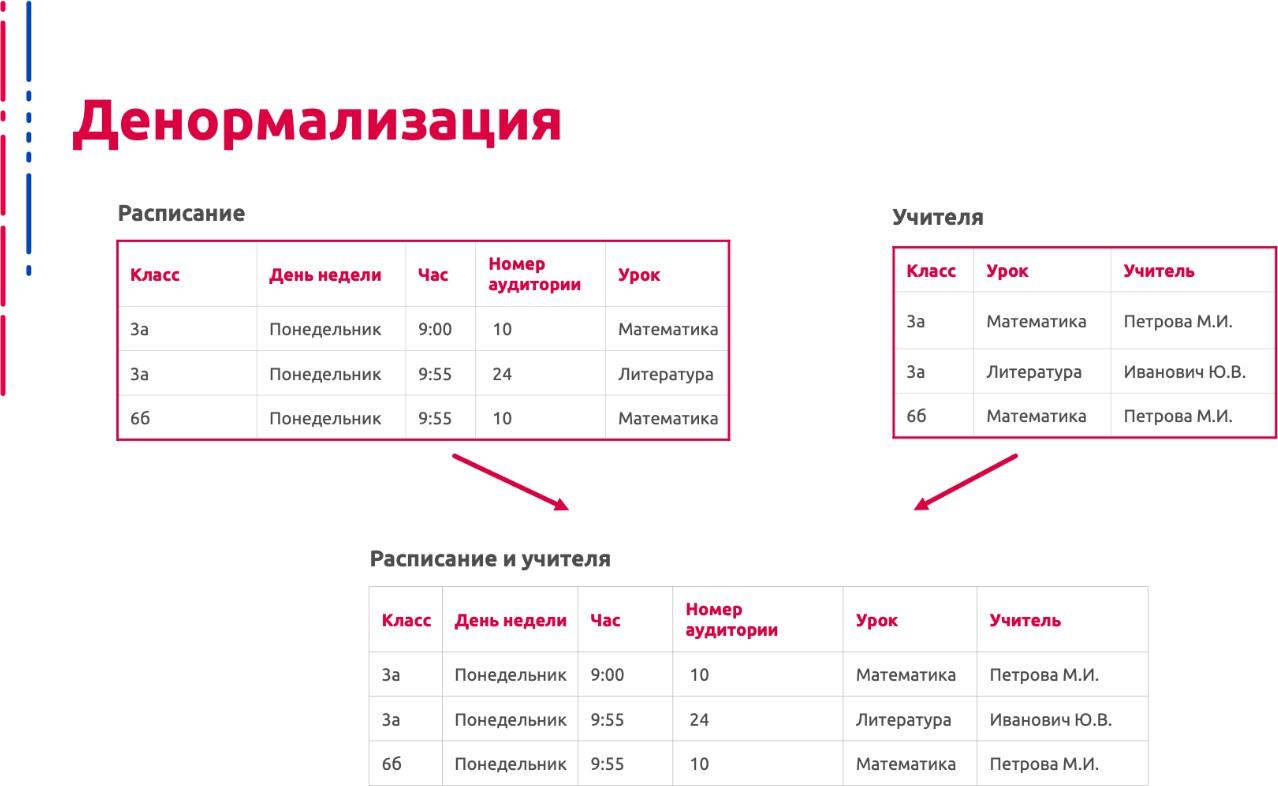


Подведем итоги: мы разбили таблицу на две, в которых удалось избежать избыточности данных. Например, куратор Певцов встречается теперь только один раз. А каждая из новых таблиц находится в 3 НФ.

# **12. Денормализация отношений**

Есть еще нормальные формы 5 и 6, устраняющие более сложные многозначные зависимости. Но надо ли всегда проводить нормализацию до наивысшей формы? Ведь экономя память и избегая аномалий, мы жертвуем производительностью системы, так как для поиска информации нам придется производить операции соединения таблиц.

Так что чаще всего нормализацию проводят лишь до 3 нормальной формы. А иногда после нормализации отношений проводят их денормализацию.

В нашем примере с расписанием уроков мы выделили учителей в отдельную таблицу. А если в системе часто возникают запросы, когда, и где и кто проводит урок, то можно вернуть в расписание учителя, соединив таблицы Расписание и Учителя

Итог: