ЦИФРОВАЯ КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

Системы управления базами данных

Высшая Школа Цифровой Культуры
Университет ИТМО
dc@itmo.ru

Содержание

1 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.

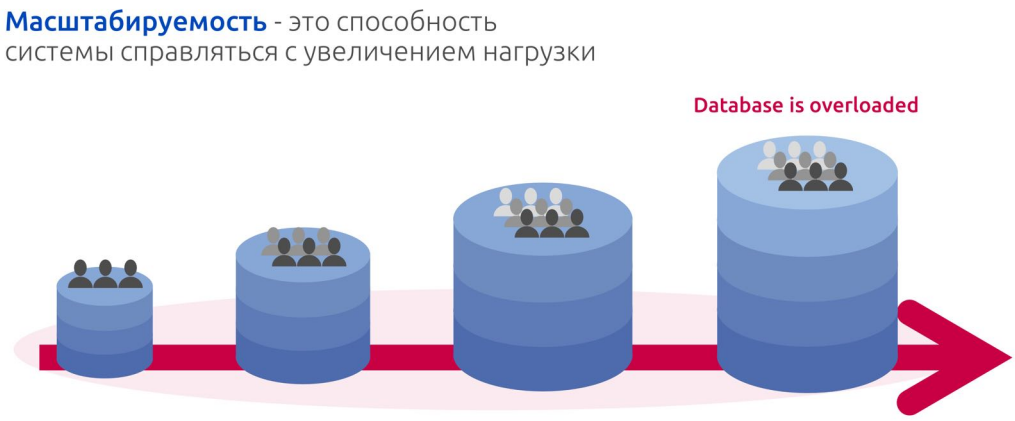
Существует множество различных требований к информационным системам. Среди основных можно выделить следующие:

- 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



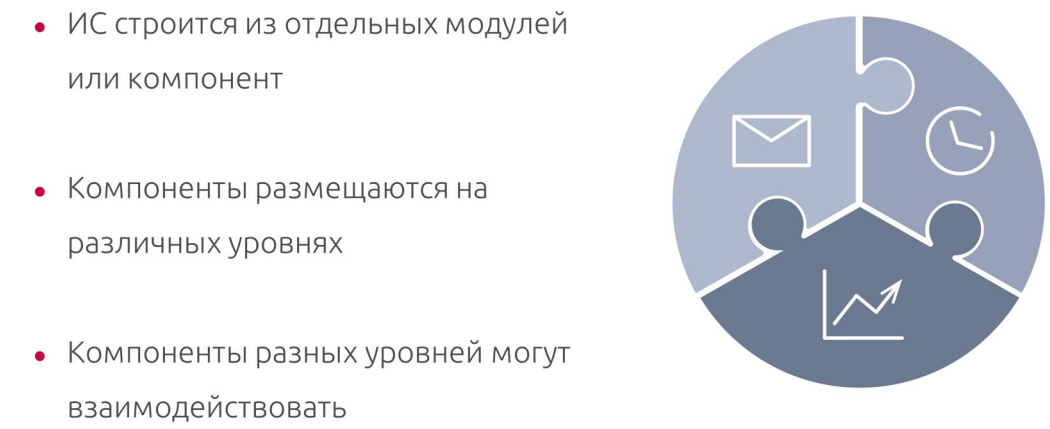
про масштабирование можно говорить не только с точки зрения аппаратных ресурсов, на которых базируется система, но и с точки зрения её программной реализации, а именно: как будут адаптироваться к увеличению нагрузки, использованные в программе алгоритмы или подключенные библиотеки.

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 разра-

ботчики использовали одни и те же согласованные и стандартизированные технологии и средства, которые могут быть изучены новыми членами команды в относительно короткое время.

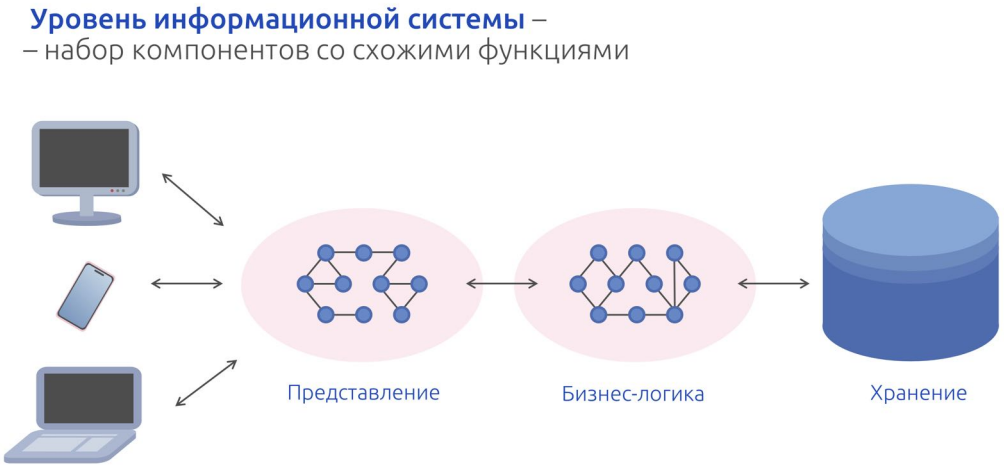
Что из этого следует? Для того, чтобы можно было реализовать эти требования на практике, систему принято строить из отдельных небольших частей, которые называются модулями или компонентами. Каждый компонент



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.

2 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.

3 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:

Обеспечение целостности данных (Integrity)

- Ограничения ссылочной целостности;
- Запрет на неопределенные значения;
- Запрет повторяющихся значений;
- Контроль возможного диапазона значений атрибутов.

ФИО студента	Предмет	Отметка
Сидоров Василий	Обработка данных	4
Алексеев 007	Алгебра	19
Семенова Анна	Прикладное искусство	?
Сидоров Василий	Обработка данных	4

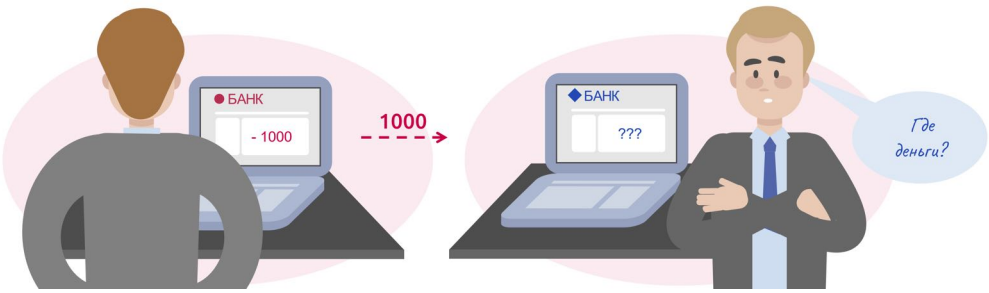
Предмет
Обработка данных
Алгебра

- 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 (Consistency)

- 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 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.

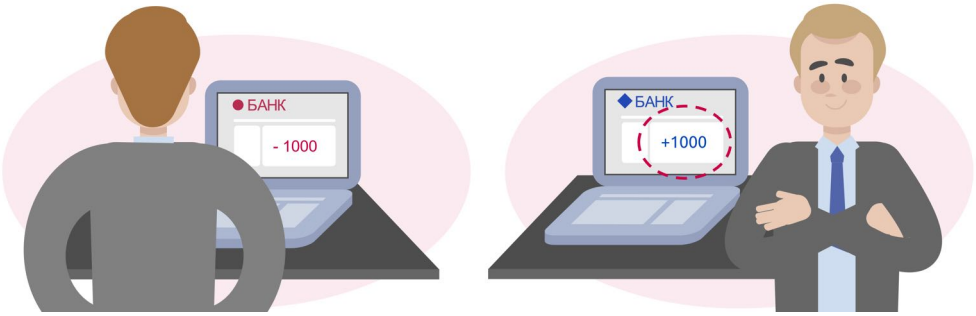
Поддержка согласованности данных тесно связано с понятием транзакции. Дадим формальное определение этому понятию.

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.

Пример транзакции (перевод средств с одного счета на другой)

- Уменьшить состояние счета первого клиента.
- Увеличить состояние счета второго клиента.



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.

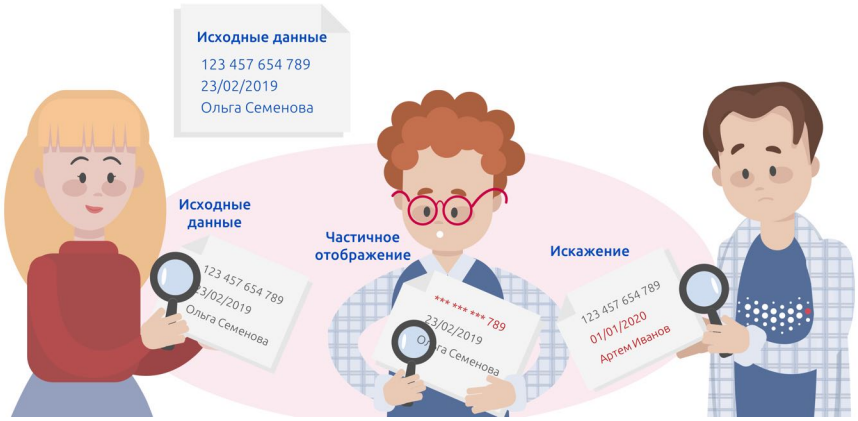
Поддержка согласованности данных со стороны СУБД означает, что после удачного или неудачного завершения работы приложения база данных в любом случае окажется в согласованном состоянии. Для транзакций это означает, что все транзакции приложения либо будут выполнены полностью, либо не оставят никаких следов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

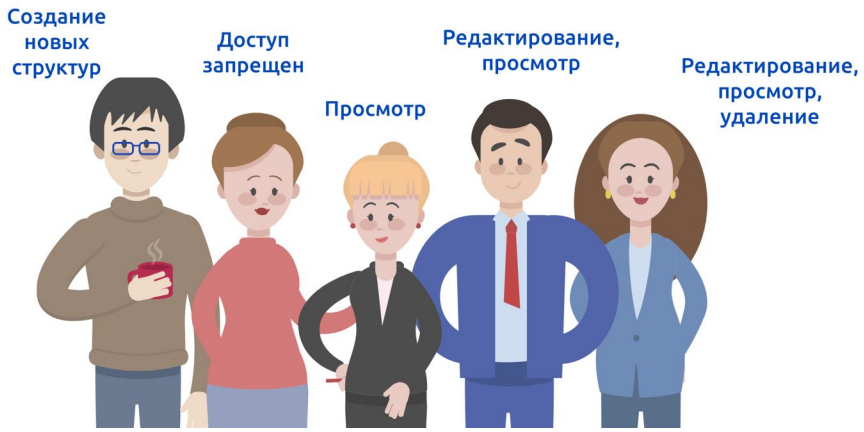
Если с данными параллельно работают несколько пользователей, то их действия могут привести данные в некорректное состояние. Например, два пользователя могут одновременно изменять одни и те же данные. Поддержка согласованности в таком случае существенно усложняется. Для обеспечения параллельной работы 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.

Примеры сокрытия данных от пользователей

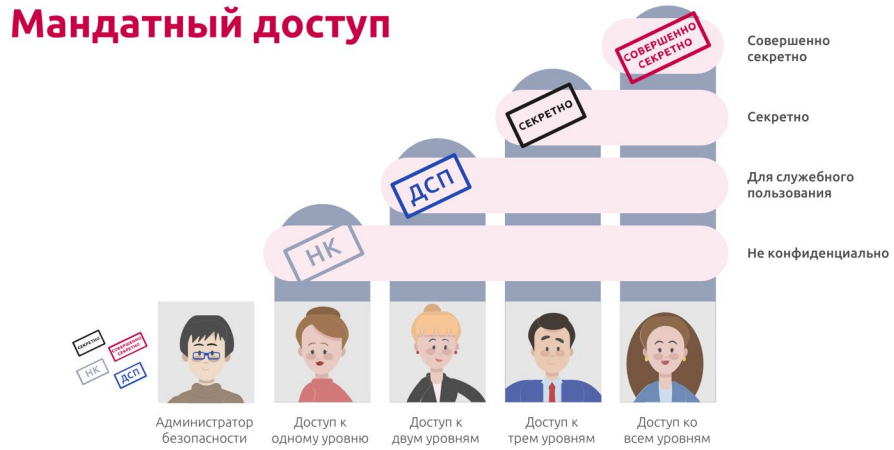


В последних версиях некоторых промышленных СУБД появляются разнообразные приемы для частичного отображения данных, а иногда и для преднамеренного, достаточно правдоподобного искаженного отображения данных. Некоторые приемы для отображения и искажения данных вы можете видеть на рисунке.



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.

Высокоуровневый язык запросов

SQL - Structured Query Language (язык структурированных запросов)

Примеры запросов:

- select * from student
- select * from student where student_id in (select student_id from marks group by student_id having min(mark) = 5)
- select * from student where student_id not in (select distinct student_id from marks where mark < 5)

С этой целью в рамках разнообразных систем управления были реализованы высокоуровневые декларативные языки для манипуляций над данным,

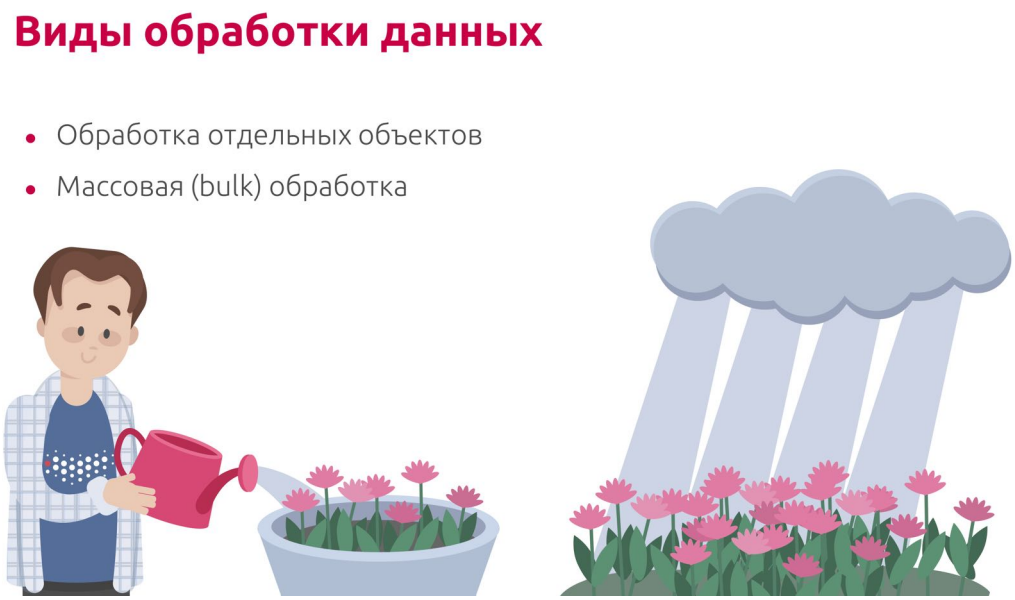
из которых особого внимания заслуживает SQL (Structured Query Language – язык структурированных запросов). Элегантность, декларативность и независимость языка от специфики СУБД, а также его поддержка ведущими производителями баз данных, сделали язык SQL основным стандартом для обработки данных. Когда 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.

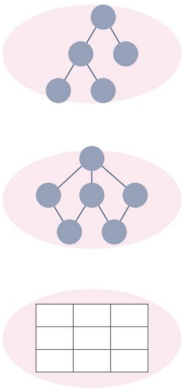


К примеру, язык SQL изначально ориентирован на массовую обработку данных. С помощью одной команды можно обработать большое количество строк. Более того, особенностью всех популярных реализаций SQL в рамках традиционных СУБД является то обстоятельство, что такие команды обработки выполняются значительно 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.

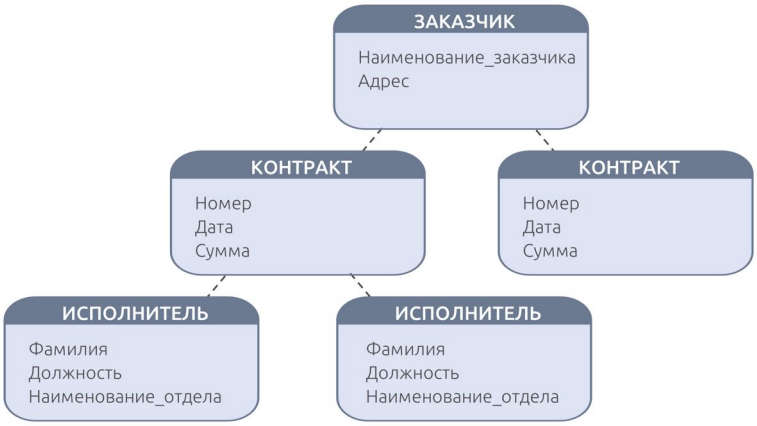
Модели данных

- Иерархические
- Сетевые
- Реляционные



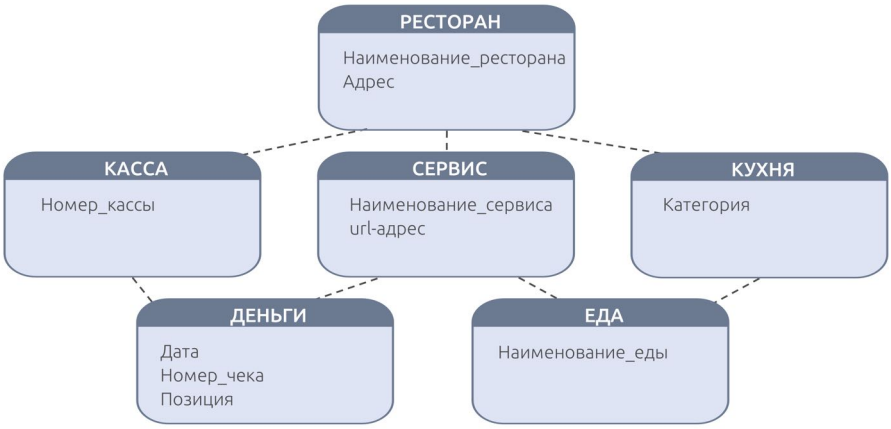
В иерархических базах данных каждая запись имеет одного родителя. Это создаёт древовидную структуру, в которой записи идентифицируются по их отношениям с цепочкой родительских записей. Пример данных, организованных таким образом вы можете видеть на рисунке.

Пример иерархической структуры



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.

Пример сетевой структуры



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.

Пример реляционной структуры

Номер личного дела	Фамилия	Имя	Отчество	Пол	Дата рождения	Специальность
16493	Сергеев	Петр	Михайлович	М	01.01.86	080104
16593	Петрова	Анна	Владимировна	Ж	15.03.85	080102
16693	Анохин	Андрей	Борисович	М	14.04.86	080104

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.

4 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 со-

ответствии 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.

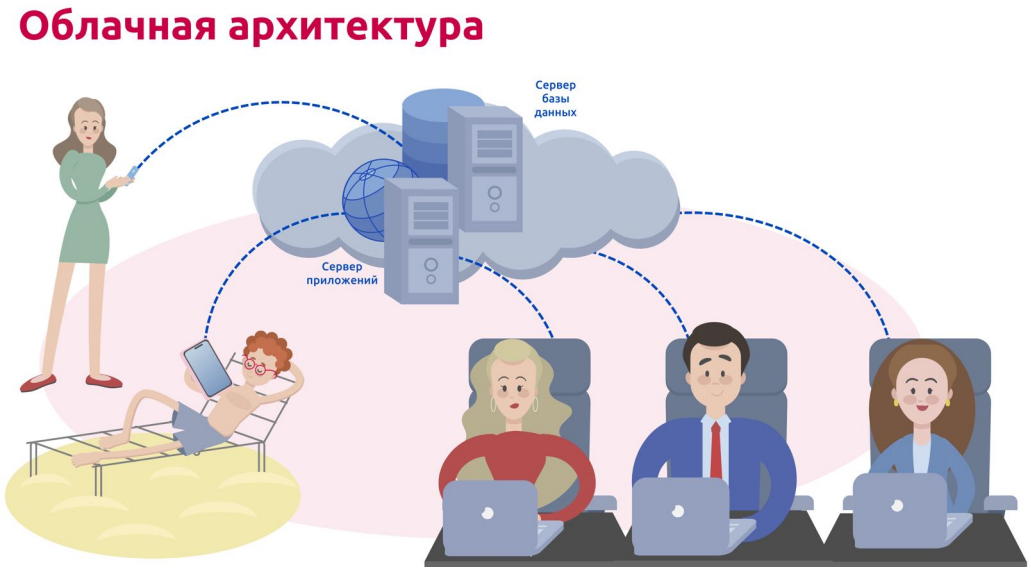
В результате работа строится следующим образом:

- 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 бизнес-логики;
- 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 модедь 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.

STUDENTS

id	name	birth_date
1	Максим Иван Игоревич	22.11.1985
2	Ребко Петр Алексеевич	15.12.1992
3	Максимов Иван Игоревич	18.05.1995
4	Петров Георгий Сергеевич	11.03.1988
5	Иванов Евгений Иванович	05.08.1984

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).

Таблица

id	name	birth_date
1	Максим Иван Игоревич	22.11.1985
2	Ребко Петр Алексеевич	15.12.1992
3	Максимов Иван Игоревич	18.05.1995
1	Максим Иван Игоревич	22.11.1985
4	Петров Георгий Сергеевич	11.03.1988
4	Петров Георгий Сергеевич	11.03.1988
5	Иванов Евгений Иванович	05.08.1984

Отношение

id	name	birth_date
1	Максим Иван Игоревич	22.11.1985
2	Ребко Петр Алексеевич	15.12.1992
3	Максимов Иван Игоревич	18.05.1995
4	Петров Георгий Сергеевич	11.03.1988
5	Иванов Евгений Иванович	05.08.1984

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.

STUDENTS

id	name	birth_date
1	Максим Иван Игоревич	22.11.1985
2	Ребко Петр Алексеевич	15.12.1992
3	Максимов Иван Игоревич	18.05.1995
4	Петров Георгий Сергеевич	11.03.1988
5	Иванов Евгений Иванович	05.08.1984

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

id	name	birth_date
1	Максим Иван Игоревич	22.11.1985
2	Ребко Петр Алексеевич	15.12.1992
3	Максимов Иван Игоревич	18.05.1995
4	Петров Георгий Сергеевич	11.03.1988
5	Иванов Евгений Иванович	05.08.1984

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

id	name	birth_date
1	Максим Иван Игоревич	22.11.1985
2	Ребко Петр Алексеевич	15.12.1992
3	Максимов Иван Игоревич	18.05.1995
4	Петров Георгий Сергеевич	11.03.1988
5	Иванов Евгений Иванович	05.08.1984

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 **первичным**

первичный ключ — составной ключ

id	Passport_series + Passport_number		Last_name	First_name	birth_date
1	38 02	565656	Максимов	Иван	22.11.1985
2	40 01	343455	Ребко	Петр	15.12.1992
3	40 01	222344	Максимов	Иван	18.05.1995
4	40 03	442457	Петров	Георгий	11.03.1988
5	42 01	323786	Иванов	Евгений	05.08.1984

ключом.

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.

STUDENTS

id	name	birth_date	gr_id
1	Максим Иван Игоревич	22.11.1985	1
2	Ребко Петр Алексеевич	15.12.1992	2
3	Максимов Иван Игоревич	18.05.1995	1
4	Петров Георгий Сергеевич	11.03.1988	2
5	Иванов Евгений Иванович	05.08.1984	2

GROUPS

gr_id	group_name
1	P3101
2	P3100
3	P3102

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`.