1.   **Классы С++. Описание класса. Пример. Описание объектов. Пример.**

**Указатель this. Пример.**

**Класс** − фундаментальное понятие С++, он лежит в основе многих свойств С++. Класс предоставляет механизм для создания объектов. В классе отражены важнейшие концепции объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм.

В простейшем случае класс можно определить с помощью конструкции:

*тип\_класса имя\_класса{список\_членов\_класса}*;

*тип\_класса* – одно из служебных слов class, struct, union;

*имя\_класса* – идентификатор;

*список\_членов\_класса* – определения и описания типизированных данных и принадлежащих классу функций.

**Функции** – это методы класса, определяющие операции над объектом.

Данные – это поля объекта, образующие его структуру. Значения полей определяет состояние объекта.

Конкретные переменные типа данных "класс" называются **экземплярами** класса, или **объектами**.

Для описания объекта класса (экземпляра класса) используется конструкция

имя\_класса имя\_объекта;

monster Vasia; //Объект класса monster с параметрами по умолчанию

monster Super(200, 300);// Объект с явной инициализацией

monster stado[100];// Массив объектов с параметрами по умолчанию

/\* Динамический объект (второй параметр задается по умолчанию) \*/

monster \*beavis = new monster (10);

monster &butthead = Vasia;// Ссылка на объект

Доступ к открытым ( public ) элементам объекта аналогичен доступу к полям структуры. Для этого используются операция . (точка) при обращении к элементу через имя объекта и операция -> при обращении через указатель:

объект.поле

указатель -> поле

(\*указатель).поле

объект.метод( параметры )указатель -> метод( параметры )

(\*указатель).метод( параметры )

Обращение к открытому полю и вызов метода для массива объектов:

имя\_массива[ индекс ].поле

имя\_массива[ индекс ].метод( параметры )

Например:

int n = Vasia.get\_ammo();

stado[5].draw;

cout << beavis->get\_health();

**Указатель this.**

Когда функция-член класса вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет имя this и неявно определен в каждой функции класса следующим образом:

имя\_класса \*const this = адрес\_объекта

Указатель this является дополнительным скрытым параметром каждой нестатической компонентной функции. При входе в тело принадлежащей классу функции this инициализируется значением адреса того объекта,для которого вызвана функция. В результате этого объект становится доступным внутри этой функции.

В большинстве случаев использование this является неявным. В частности, каждое обращение к нестатической функции-члену класса неявно использует this для доступа к члену соответствующего объекта.

Выражение \*this представляет собой разыменование указателя и имеет тип определяемого класса. Обычно это выражение возвращается в качестве результата, если метод возвращает ссылку на свой класс (return \*this;).

monster & the\_best(monster &M){

if( health > M.get\_health()) {

return \*this;

}

return M;

}  
  
monster Vasia(50), Super(200);  
// Новый объект Best инициализируется значениями полей Super  
monster Best = Vasia.the\_best(Super);

2**.    Классы С++. Конструкторы. Свойства конструкторов. Конструктор**

**копирования. Пример. Деструкторы.**

Конструктор

Для инициализации объектов класса в его определение можно явно включить специальную компонентную функцию, называемую конструктором. Формат определения конструктора следующий:

имя\_класса(список\_форм\_параметров){операторы\_тела\_конструктора}

Имя этой компонентной функции по правилам языка С++ должно совпадать с именем класса. Такая функция автоматически вызывается при определении или размещении в памяти с помощью оператора new каждого объекта класса.

Конструктор выделяет память для объекта и инициализирует данные − члены класса.

**Свойства конструктора:**

• Для конструктора не определяется тип возвращаемого значения. Даже тип void не допустим.

•Указатель на конструктор не может быть определен, и соответственно нельзя получить адрес конструктора.

•Конструкторы не наследуются.

•Конструкторы не могут быть описаны с ключевыми словами virtual, static, const, mutuable, valatile.

Конструктор всегда существует для любого класса, причем, если он не определен явно, он создается автоматически. По умолчанию создается конструктор без параметров и конструктор копирования. Если конструктор описан явно, то конструктор по умолчанию не создается. По умолчанию конструкторы создаются общедоступными (public).

Параметром конструктора не может быть его собственный класс, но может быть ссылка на него (T&).

Существуют два способа инициализации данных объекта с помощью конструктора: передача значений параметров в тело конструктора, список инициализаторов данного класса.

По умолчанию создается также конструктор копирования вида T::T(const T&), где Т – имя класса. Конструктор копирования вызывается всякий раз, когда выполняется копирование объектов, принадлежащих классу. В частности он вызывается:

а) когда объект передается функции по значению;

б) при построении временного объекта как возвращаемого значения функции;

в) при использовании объекта для инициализации другого объекта.

Если класс не содержит явным образом определенного конструктора копирования, то при возникновении одной из этих трех ситуаций производится побитовое копирование объекта. Побитовое копирование не во всех случаях является адекватным. Именно для таких случаев и необходимо определить собственный конструктор копирования.

**Деструктор**

Динамическое выделение памяти для объекта создает необходимость освобождения этой памяти при уничтожении объекта. Желательно, чтобы освобождение памяти происходило автоматически. Такую возможность обеспечивает специальный компонент класса – деструктор класса. Его формат:

~имя\_класса(){операторы\_тела\_деструктора}

Имя деструктора совпадает с именем его класса, но предваряется символом “~” (тильда).

Деструктор не имеет параметров и возвращаемого значения. Вызов деструктора выполняется не явно (автоматически), как только объект класса уничтожается.

Если в классе деструктор не определен явно, то компилятор генерирует деструктор по умолчанию, который просто освобождает память, занятую данными объекта. В тех случаях, когда требуется выполнить освобождение и других объектов памяти, например область, на которую указывает.

Так же, как и для конструктора, не может быть определен указатель на деструктор.

**Примеры:**

Exam::Exam() {

cout << "// Конструктор без параметров " << this << endl;

}

Exam::Exam(char\*name, int date) {

this->name = name;

this->date = date;

cout << "// Конструктор с параметрами " << this << endl;

}

Exam::Exam(const Exam &cpexam) {

student\_name = cpexam.student\_name;

student\_mark = cpexam.student\_mark;

cout << "// Конструктор копирования " << this << endl;

}

Exam::~Exam() {

cout << "// Деструктор" << this << endl;

}

3.    Классы С++. Статические элементы класса. Статические поля. Статические

методы. Пример. Статические функции-члены. Указатель this.

**Статические члены класса.**

Такие компоненты должны быть определены в классе, как статические (static). Статические данные классов не дублируются при создании объектов, т.е. каждый статический компонент существует в единственном экземпляре. Доступ к статическому компоненту возможен только после его инициализации. Для инициализации используется конструкция

тип имя\_класса : : имя\_данного инициализатор;

Например, int complex : : count = 0;

Это предложение должно быть размещено в глобальной области после определения класса. Только при инициализации статическое данное класса получает память и становится доступным. Обращаться к статическому данному класса можно обычным образом через имя объекта

имя\_объекта.имя\_компонента

Но к статическим компонентам можно обращаться и тогда, когда объект класса еще не существует. Доступ к статическим компонентам возможен не только через имя объекта, но и через имя класса

имя\_класса :: имя\_компонента

Однако так можно обращаться только к public компонентам.

Для обращения к private статической компоненте извне можно с помощью статических компонентов-функций. Эти функции можно вызвать через имя класса.

имя\_класса :: имя\_статической\_функции

Пример.

class TPoint {

double x,y;

static int N; // статический компонент − данное: количество точек

public:

TPoint (double x1 = 0.0,double y1 = 0.0){

N++; x = x1; y = y1;

}

static int &count(){ // статический компонент-//функция

return N;

}

};

int TPoint :: N = 0; //инициализация статического //компонента-данного

void main(void){

TPoint A(1.0,2.0);

TPoint B(4.0,5.0);

TPoint C(7.0,8.0);

cout<< \nОпределены ”<<TPoint : : count()<<“точки”;

}

**Статические поля**

Статические поля применяются для хранения данных, общих для всех объектов класса, например, количества объектов или ссылки на разделяемый всеми объектами ресурс. Эти поля существуют для всех объектов класса в единственном экземпляре, то есть не дублируются.

Память под статическое поле выделяется один раз при его инициализации независимо от числа созданных объектов (и даже при их отсутствии) и инициализируется с помощью операции доступа к области действия, а не операции выбора:

class A{

public:  
static int count;

}  
A::count = 0;

Статические поля доступны как через имя класса, так и через имя объекта:  
/\* будет выведено одно и то же \*/  
A \*a, b; \* cout << A::count << a->count << b.count;

Классическое применение статических полей - подсчет объектов. Для этого в классе объявляется целочисленное поле, которое увеличивается в конструкторе и уменьшается в деструкторе.

**Статические методы**

Статические методы могут обращаться непосредственно только к статическим полям и вызывать только другие статические методы класса, поскольку им не передается скрытый указатель this. Обращение к статическим методам производится так же, как к статическим полям - либо через имя класса, либо, если хотя бы один объект класса уже создан, через имя объекта.

Статические методы не могут быть константными ( const ) и виртуальными ( virtual ).

**Указатель this.**

Когда функция-член класса вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет имя this и неявно определен в каждой функции класса следующим образом:

имя\_класса \*const this = адрес\_объекта

Указатель this является дополнительным скрытым параметром каждой нестатической компонентной функции. При входе в тело принадлежащей классу функции this инициализируется значением адреса того объекта,для которого вызвана функция. В результате этого объект становится доступным внутри этой функции.

В большинстве случаев использование this является неявным. В частности, каждое обращение к нестатической функции-члену класса неявно использует this для доступа к члену соответствующего объекта.

Выражение \*this представляет собой разыменование указателя и имеет тип определяемого класса. Обычно это выражение возвращается в качестве результата, если метод возвращает ссылку на свой класс (return \*this;).

monster & the\_best(monster &M){

if( health > M.get\_health()) {

return \*this;

}

return M;

}  
  
monster Vasia(50), Super(200);  
// Новый объект Best инициализируется значениями полей Super  
monster Best = Vasia.the\_best(Super);

4.    Дружественная функция С++. Особенности дружественной функции. Пример.

**Дружественные функции** не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private.

Для определения дружественных функций используется ключевое слово friend.

class Auto{

friend void drive(Auto &);

friend void setPrice(Auto &, int price);

public:

Auto(string autoName, int autoPrice){

name = autoName;

price = autoPrice; }

string getName(){ return name; }

int getPrice() { return price; }

private:

string name; // название автомобиля

int price; // цена автомобиля

};

void drive(Auto &a) {

cout << a.name;}

void setPrice(Auto &a, int price){

if (price > 0)

a.price = price;}

int main(){

Auto tesla("Tesla", 5000);

drive(tesla);

cout << tesla.getName() << " : " << tesla.getPrice();

setPrice(tesla, 8000);

cout << tesla.getName() << " : " << tesla.getPrice();

}

Когда мы объявляем дружественные функции, то фактически мы говорим компилятору, что это друзья класса и они имеют доступ ко всем членам этого класса, в том числе закрытым.

При этом для дружественных функций не важно, определяются они под спецификатором public или private. Для них это не имеет значения.

Определение этих функций производится вне класса. И поскольку эти функции являются дружественными, то внутри этих функций мы можем через переданную ссылку Auto обратиться ко всем его закрытым переменным.

Дружественные функции могут определяться в другом классе. Если метод класса A внутри тела класса B объявляется с модификатором friend, что указывает на то, что он является другом класса, то из него разрешен доступ ко всем членам класса B.

Например:

class A { public: int Fx();}

class B { public: friend int A::Fx();

private:

}

Объявление **дружественного класса** позволяет всем его методам получить доступ ко всем переменным и методам другого класса.

Если все функции-члены одного класса являются дружественными функциями второго класса, то можно объявить дружественный класс: friend class ИмяКласса;

class A {...};

class B

{

friend class A;

};

Все функции-члены класса А будут иметь доступ к скрытым членам класса В.

Дружественные классы не наследуются, и их дружественность не является транзитивной.

class A {int Fx();}

class B {friend class A;}

class C {friend class B;}

// Класс А не является дружественным классу C

class D : public B {}

// Класс А не является дружественным классу D

5.    **Перегрузка операций С++. Основные правила перегрузки. Перегрузка**

**унарных операций. Пример. Перегрузка бинарных операций. Пример.**

Перегрузка операторов позволяет определить действия, которые будет выполнять оператор. Перегрузка подразумевает создание функции, название которой содержит слово operator и символ перегружаемого оператора. Функция оператора может быть определена как член класса, либо вне класса.

Перегрузить можно только те операторы, которые уже определены в C++. Создать новые операторы нельзя.

Если функция оператора определена как отдельная функция и не является членом класса, то количество параметров такой функции совпадает с количеством операндов оператора. Например, у функции, которая представляет унарный оператор, будет один параметр, а у функции, которая представляет бинарный оператор, - два параметра. Если оператор принимает два операнда, то первый операнд передается первому параметру функции, а второй операнд - второму параметру. При этом как минимум один из параметров должен представлять тип класса

Для того чтобы перегрузить операцию в С++ следует объявить функцию или функцию-член класса следующего вида:

<тип> operator <символ> ([<список параметров>]);

Здесь <тип> - тип результата операции, <символ> - символьное обозначение операции, <список параметров> - параметры функции перегрузки операции.

Если предполагается, что операнд при выполнении операции не изменяется (например, операнды в операциях +, -, &&, ||), то в функцию его лучше передавать по значению или константной ссылке. Причем, если операндом является объект, занимающий большой объем памяти, то его лучше передавать по константной ссылке. Для базовых типов данных вполне подойдет передача по значению.

Если операнд изменяется при выполнении операции (например, первый операнд в операциях +=, &=, ++), то такой операнд должен передаваться по неконстантной ссылке.

Не допускается перегрузка следующих операций:

:: ?: \* . typeid sizeof

Существуют операции, которые могут быть перегружены только с использованием нестатических функций-членов классов. Это операции =, (), [], ->.

Перегрузка операторов основана на следующих **принципах**:

1. Один из аргументов должен иметь тип класс.

2. Перегружать можно только встроеные операторы, новые ввести нельзя.

3. Нельзя изменить приоритет операторов и число их аргументов.

4. Можно изменить смысл оператора.

5. Перегрузка основана на определении методов и функций специального вида

**Бинарный**

Функция operator для перегрузки (доопределения) **бинарных операторов** может быть описана двумя способами:

- как компонента-функция класса с одним аргументом, перегружаемой **внутри** класса:

<тип> operator <символ>( <тип2> <парам2>);

- как глобальная функция (функция, описанная вне класса) с двумя аргументами, перегружаемой **вне** класса:

<тип> operator <символ> (<тип1> <парам1>, <тип2> <парам2>);

При перегрузке бинарного оператора # выражение a#b может быть представлено при первом способе как a.operator#(b) или как operator #(a,b) при втором способе перегрузки.

Пример:

// перегрузка операции сравнения

int operator != (CMyString& s1, CMyString& s2){

return !( s1 == s2 );

}

**Унарный**

Унарный оператор, как и бинарный, может быть перегружен двумя способами:

- как компонента-функция без аргументов, перегружаемой **внутри** класса:

<тип> operator <символ>();

- как глобальная функция с одним аргументом, перегружаемой **вне** :

<тип> operator <символ> (<тип1> <парам1>);

Как известно, унарный оператор может быть префиксным и постфиксным. Для любого префиксного унарного оператора выражение #a может быть представлено при первом способе как a.operator#(), а при втором как #operator(a).

При перегрузке унарного оператора, используемого в постфиксной форме, выражение вида a# может быть представлено при первом способе как a.operator#(int) или как operator#(a,int) при втором способе. При этом аргумент типа int не существует и используется для отличия префиксной и постфиксной форм при перегрузке.

Пример:

// перегрузка операции ++A

void operator++(int x){

x++;

}

6.    **Перегрузка операций С++. Операторы, которые нельзя перегружать.**

Операторы, которые можно перегружать в качестве методов

Для того чтобы перегрузить операцию в С++ следует объявить функцию или функцию-член класса следующего вида:

<тип> operator <символ> ([<список параметров>]);

Здесь <тип> - тип результата операции, <символ> - символьное обозначение операции, <список параметров> - параметры функции перегрузки операции. Количество параметров в списке зависит от того является ли операция унарной или бинарной, а также от способа перегрузки операции. Перегрузка операторов не изменяет порядок выполнения операций и их приоритет. Унарный оператор не может использоваться для переопределения бинарной операции так же, как и бинарный оператор не переопределит унарную операцию.

-В случае, когда перегрузка выполняется функцией вне пользовательского класса, количество параметров функции совпадает с количеством операндов перегружаемой операции.

-В случае, когда перегрузка выполняется функцией-членом класса количество параметров функции на единицу меньше числа операндов перегружаемой операции. В этом случае первый из операндов операции передается в функцию-член класса в качестве указателя this и отсутствует в списке параметров функции.

Операция, в которой первый аргумент не является объектом пользовательского класса, не может быть перегружена функцией-членом класса, а только внешней по отношению к классу функцией. К сожалению, ввести новое символьное обозначение для операции нельзя. Также нельзя перегрузить оператор для базовых типов данных.

Не допускается перегрузка следующих операций:

:: ?: \* . typeid sizeof

Существуют операции, которые могут быть перегружены только с использованием нестатических функций-членов классов. Это операции =, (), [], ->.

Перегрузка операторов основана на следующих **принципах**:

1. Один из аргументов должен иметь тип класс.

2. Перегружать можно только встроеные операторы, новые ввести нельзя.

3. Нельзя изменить приоритет операторов и число их аргументов.

4. Можно изменить смысл оператора.

5. Перегрузка основана на определении методов и функций специального вида

7.    **Перегрузка операций С++. Перегрузка операции вызова функции. Пример.**

Перегрузка операции присваивания. Пример.

Для того чтобы перегрузить операцию в С++ следует объявить функцию или функцию-член класса следующего вида:

<тип> operator <символ> ([<список параметров>]);

Здесь <тип> - тип результата операции, <символ> - символьное обозначение операции, <список параметров> - параметры функции перегрузки операции. Количество параметров в списке зависит от того является ли операция унарной или бинарной, а также от способа перегрузки операции. Перегрузка операторов не изменяет порядок выполнения операций и их приоритет. Унарный оператор не может использоваться для переопределения бинарной операции так же, как и бинарный оператор не переопределит унарную операцию.

Перегрузка операторов основана на следующих **принципах**:

1. Один из аргументов должен иметь тип класс.

2. Перегружать можно только встроеные операторы, новые ввести нельзя.

3. Нельзя изменить приоритет операторов и число их аргументов.

4. Можно изменить смысл оператора.

5. Перегрузка основана на определении методов и функций специального вида

**Перегрузка операции присваивания**

Операция присваивания определена в любом классе по умолчанию как поэлементное копирование. Эта операция вызывается каждый раз, когда одному существующему объекту присваивается значение другого. Операция-функция должна возвращать ссылку на объект, для которого она вызвана, и принимать в качестве параметра единственный аргумент - ссылку на присваиваемый объект:

Операцию присваивания можно определять только в теле класса. Она не наследуется.

Пример

Class first\_Class{

private:

    int x;

int y;

public:

// Перегрузка оператора присваивания

first\_Class & operator= (const first\_Class &M){

// Выполняем копирование значений

x = M.x;

y = M.y;

// Возвращаем текущий объект, чтобы иметь возможность связать в цепочку выполнение нескольких операций присваивания

return \*this;

}

};

**Перегрузка операции вызова функции**

Операция присваивания рассматривается как бинарная операция, второй аргумент которой представляет совой список параметров вызова(содержащий ноль или более параметров) . Перегружается операция как функция-член класса.

Пример

class Matrix

{

private:

    double data[5][5];

public:

    Matrix(){

// Присваиваем всем элементам массива значение 0.0

        for (int i = 0; i < 5; ++i)

            for (int j = 0; j < 5; ++j)

                data[i][j] = 0.0;

     }

};

double& Matrix::operator()(int row, int col)

{

    assert(col >= 0 && col < 5);

    assert(row >= 0 && row < 5);

    return data[row][col];

}

8.    **Наследование. Ключи доступа. Простое наследование С++. Множественное**

**наследование С++.**

**Наследование** − это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются базовыми, а новые – производными. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

В иерархии производный объект наследует разрешенные для наследования компоненты всех базовых объектов (public, protected).

Допускается множественное наследование – возможность для некоторого класса наследовать компоненты нескольких никак не связанных между собой базовых классов. В иерархии классов соглашение относительно доступности компонентов класса следующее:

**private** – член класса может использоваться только функциями –членами данного класса и функциями – “друзьями” своего класса. В производном классе он недоступен.

**protected** – то же, что и private, но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями – “друзьями” классов, производных от данного.

**public** – член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к public - членам возможен доступ извне через имя объекта.

Следует иметь в виду, что объявление friend не является атрибутом доступа и не наследуется.

Синтаксис определения производного класса:

class имя\_класса : [private | protected | public] список\_базовых\_классов

{список\_компонентов\_класса};

В производном классе унаследованные компоненты получают статус доступа private, если новый класс определен с помощью ключевого слова class, и статус public, если с помощью struct.

Явно изменить умалчиваемый статус доступа при наследовании можно с помощью атрибутов доступа – private, protected и public, которые указываются непосредственно перед именами базовых классов.

Объекты класса конструируются снизу-вверх: сначала базовый, потом компоненты-объекты (если они имеются), а потом сам производный класс. Таким образом, объект производного класса содержит в качестве подобъекта объект базового класса.

Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.

**Простым** называется наследование, при котором производный класс имеет одного родителя. Для различных элементов класса существуют разные правила наследования.

**Множественное** наследование означает, что класс имеет несколько базовых классов. При этом, если в базовых классах есть одноименные элементы, может произойти конфликт идентификаторов, который устраняется с помощью операции доступа к области видимости. Если у базовых классов есть общий предок, это приведет к тому, что производный от этих базовых класс унаследует два экземпляра полей предка, что чаще всего является нежелательным. Чтобы избежать такой ситуации, требуется при наследовании общего предка определить его как виртуальный класс.

9**.    Наследование, виртуальные функции и полиморфизм в С++.**

**Наследование** − это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются базовыми, а новые – производными. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

**Виртуальные функции.**

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются полиморфными и играют особую роль в ООП.

Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания. Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово virtual.

Пример.

class base

{

public:

virtual void print(){cout<<“\nbase”;}

...

};

class dir : public base

{

public:

void print(){cout<<“\ndir”;}

};

void main()

{

base B,\*bp = &B;

dir D,\*dp = &D;

base \*p = &D;

bp –>print(); // base

dp –>print(); // dir

p –>print(); // dir

}

Таким образом, интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов.

Выбор того, какую виртуальную функцию вызвать, будет зависеть от типа объекта, на который фактически (в момент выполнения программы) направлен указатель, а не от типа указателя.

Виртуальными могут быть только нестатические функции-члены.

Виртуальность наследуется. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию,причем спецификатор virtual может не использоваться.

Конструкторы не могут быть виртуальными, в отличие от деструкторов. Практически каждый класс, имеющий виртуальную функцию, должен иметь виртуальный деструктор.

Полиморфизм — это способность обьекта использовать методы производного класса, который не существует на момент создания базового.

10**.  Наследование и спецификаторы доступа в С++. Спецификатор доступа при**

**наследовании базового класса.**

**Наследование** − это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются базовыми, а новые – производными. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

**Спецификаторы доступа:**

В С++ члены класса классифицируются в соответствии с правами доступа на три категории:

– public (публичные) – любая функция программы имеет к ним доступ;

– private (частные) – доступ к ним имеют только функции-члены класса или дружественные функции;

– protected (защищенные) – доступ к ним имеют функции-члены класса или дружественные функции, а также производные классы.

При описании класса в его заголовке перечисляются (через запятую) все классы, которые являются для него базовыми. Возможность обращения к элементам этих классов регулируется с помощью спецификаторов доступа.

**Спецификатор доступа при наследовании базового класса:**

Элементы protected при наследовании с ключом private становятся в производном классе private, в остальных случаях права доступа к ним не изменяются.

Доступ к элементам public при наследовании становится соответствующим ключу доступа.

Если базовый класс наследуется с ключом private, можно выборочно сделать некоторые его элементы доступными в производном классе, объявив их в секции public производного класса с помощью операции доступа к области видимости.

**Передача параметров в базовый класс:**

В случае, когда базовый класс содержит конструктор с аргументами, производные классы должны явно передавать базовому классу необходимые аргументы.

Для этого используется расширенная форма конструкторов производных классов, в которые передаются аргументы конструкторам базового класса.

Эта форма выглядит следующим образом:

порожденый\_конструктор(список\_аргументов):базовый\_класс1(список\_аргументов), базовый\_класс2(список\_аргументов), . . ., базовый\_классN(список\_аргументов)

{

. . . тело класса

};

**Ссылки на производные классы**

Предположим, например, что имеется базовый класс B\_class и его производный класс D\_class. В С++ любой указатель типа B\_class\* может также указы­вать на объект типа D\_class. Например, если имеются следующие объявления переменных:  
  
B\_class \*р; // указатель на объект типа B\_class  
B\_class B\_ob; // объект типа B\_class  
D\_class D\_ob; // объект типа D\_class

то следующие присвоения абсолютно законны:р = &В\_оb; // р указывает на объект типа B\_class  
р = &D\_ob; /\* р указывает на объект типа D\_class, являющийся объектом, порожденным от B\_class \*/

11. **Виртуальные функции в С++. Чисто виртуальные функции и абстрактные**

**типы. Виртуальный базовый класс.**

**Виртуальные функции.**

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются полиморфными и играют особую роль в ООП.

Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания. Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово virtual.

Пример.

class base

{

public:

virtual void print(){cout<<“\nbase”;}

...

};

class dir : public base

{

public:

void print(){cout<<“\ndir”;}

};

void main()

{

base B,\*bp = &B;

dir D,\*dp = &D;

base \*p = &D;

bp –>print(); // base

dp –>print(); // dir

p –>print(); // dir

}

Таким образом, интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов.

Выбор того, какую виртуальную функцию вызвать, будет зависеть от типа объекта, на который фактически (в момент выполнения программы) направлен указатель, а не от типа указателя.

Виртуальными могут быть только нестатические функции-члены.

Виртуальность наследуется. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию,причем спецификатор virtual может не использоваться.

Конструкторы не могут быть виртуальными, в отличие от деструкторов. Практически каждый класс, имеющий виртуальную функцию, должен иметь виртуальный деструктор.

**Абстрактные классы.**

Абстрактным называется класс, в котором есть хотя бы одна чистая(пустая) виртуальная функция.

**Чистой виртуальной функцией** называется компонентная функция, которая имеет следующее определение:

virtual тип имя\_функции (список\_формальных\_параметров) = 0;

Чистая виртуальная функция ничего не делает и недоступна для вызовов. Ее назначение – служить основой для подменяющих ее функций в производных классах. Абстрактный класс может использоваться только в качестве базового для производных классов.

Механизм абстрактных классов разработан для представления общих понятий, которые в дальнейшем предполагается конкретизировать. При этом построение иерархии классов выполняется по следующей схеме. Во главе иерархии стоит абстрактный базовый класс. Он используется для наследования интерфейса. Производные классы будут конкретизировать и реализовать этот интерфейс. В абстрактном классе объявлены чистые виртуальные функции, которые по сути есть абстрактные методы.

Пример.

class Base{

public:

Base(); // конструктор по умолчанию

Base(const Base&); // конструктор копирования

virtual ~Base(); // виртуальный деструктор

virtual void Show()=0; // чистая виртуальная функция

// другие чистые виртуальные функции

protected: // защищенные члены класса

private:

// часто остается пустым, иначе будет мешать будущим //разработкам

};

class Derived: virtual public Base{

public:

Derived(); // конструктор по умолчанию

Derived(const Derived&); // конструктор копирования

Derived(параметры); // конструктор с параметрами

virtual ~Derived(); // виртуальный деструктор

void Show(); // переопределенная //виртуальная функция

// другие переопределенные виртуальные функции

// другие перегруженные операции

protected:

// используется вместо private, если ожидается наследование

private:

// используется для деталей реализации

};

Объект абстрактного класса не может быть формальным параметром функции, однако формальным параметром может быть указатель на абстрактный класс. В этом случае появляется возможность передавать в вызываемую функцию в качестве фактического параметра значение указателя на производный объект, заменяя им указатель на абстрактный базовый класс. Таким образом мы получаем полиморфные объекты.

**Виртуальный базовый класс**

**Виртуальный базовый класс** — это класс, объект которого является общим для использования всеми дочерними классами. Вот пример создания общего родительского класса:

|  |
| --- |
| class PoweredDevice {      PoweredDevice(int power) {  std::cout << "PoweredDevice: " << power << '\n';      }  };    class Scanner: public PoweredDevice{      Scanner(int scanner, int power) : PoweredDevice(power) {  std::cout << "Scanner: " << scanner << '\n';      }  };    class Printer: public PoweredDevice {      Printer(int printer, int power) : PoweredDevice(power) {  std::cout << "Printer: " << printer << '\n';      }  };    class Copier: public Scanner, public Printer {      Copier(int scanner, int printer, int power): Scanner(scanner, power), Printer(printer, power)      {    }  };  Диаграмма наследования:  https://ravesli.com/wp-content/uploads/2018/10/inheriting-diagram.jpg |

Теперь, при создании класса Copier, мы получим только одну копию PoweredDevice, которая будет общей как для Scanner, так и для Printer.

12. **Объявление класса в отдельном файле. Отделение интерфейса от реализации**

**в С++.**

Подключение файлов выполняется с помощью препроцессорной директивы #include. Но даже, если мы сможем подключить файл с классом, появится новая проблема — так как в файле с классом уже есть функция main(), то при построении проекта компилятор выдаст ошибку. Суть ошибки: «В проекте найдено несколько main() — функций.» Именно поэтому класс необходимо объявлять в отдельном файле, чтобы его можно было неоднократно использовать. Ранее мы объявляли в отдельном файле функции, таким же образом размещается класс в отдельном файле.

Для этого необходимо выполнить 3 шага:

1)добавить в проект заголовочный файл \*.h;

2)в заголовочном файле объявить пользовательский класс, в нашем случае — CppStudio;

3)подключить заголовочный файл к программе, в нашем случае — #include "CppStudio.h".

**Отделение интерфейса от реализации**

Интерфейс класса — конструкция, определяющая методы и свойства, предоставляемые классом. Реализация класса — это способ осуществления работоспособности класса. До этого мы не отделяли интерфейс класса от его реализации, то есть реализация методов осуществлялась внутри класса. Отделение интерфейса от реализации класса выполняется для того, чтобы скрыть способ осуществления работоспособности класса. Отделение интерфейса от реализации выполняется за 5 шагов:

1)добавить в проект заголовочный файл \*.h;

2)определить интерфейс класса в заголовочном файле;

3)добавить в проект исполняемый файл \*.cpp;

4)в исполняемом файле выполнить реализацию класса;

5)подключить заголовочный файл к программе.

Имена заголовочному и исполняемому файлам даются, как правило, одинаковые.

13**. Информационная система. Данные. База данных. Ведение базы данных. Система управления базами данных. Классификация АИС(автоматизированные информационные системы) по типу хранимых данных**

**Информационная система** (ИС) — [система](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), предназначенная для хранения, поиска и обработки [информации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F), и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию ([ISO](https://ru.wikipedia.org/wiki/ISO)/IEC 2382:2015)[[1]](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0#cite_note-ISO/IEC_2382-1.IS-1).

ИС предназначена для своевременного обеспечения надлежащих людей надлежащей [информацией](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F)[[2]](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0#cite_note-_de9948b04b7226a8-2), то есть для удовлетворения конкретных информационных потребностей в рамках определённой [предметной области](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C), при этом результатом функционирования информационных систем является *информационная продукция* — документы, информационные массивы, [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и информационные услуги[[3]](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0#cite_note-gost-7.0-3).

**База данных** (БД) – это специальным образом организованное хранение информационных ресурсов в виде интегрированной совокупности файлов, обеспечивающей удобное взаимодействие между ними и быстрый доступ к данным.

**База данных** – это динамичный объект, меняющий значения при изменении состояния отражаемой предметной области (внешних условий по отношению к базе). Под предметной областью понимается часть реального мира (объектов, процессов), которая должна быть адекватно, в полном информационном объеме представлена в базе данных. Данные в базе организуются в единую целостную систему что обеспечивает более производительную работу пользователей с большими объемами данных.

Основные ***функции*** СУБД:

* непосредственное управление данными во внешней памяти;
* управление буферами оперативной памяти;
* управление транзакциями;
* журнализация;
* поддержка языков БД.

**Система управления базами данных**(СУБД) – это система, обеспечивающая поиск, хранение, корректировку данных, формирование ответов на запросы. Система обеспечивает сохранность данных, их конфиденциальность, перемещение и связь с другими программными средствами.

**Системы управления базами данных** – это программные средства, с помощью которых можно создавать базы данных, наполнять их и работать с ними.

Обычно СУБД работает с несколькими видами объектов:

* ***Таблицы*** – основные объекты базы данных. Реляционная база данных может иметь много взаимосвязанных таблиц.
* ***Запросы*** – это специальные структуры, предназначенные для обработки данных базы. С помощью запросов данные упорядочивают, фильтруют, отбирают, изменяют, объединяют, то есть обрабатывают. Существует много различных видов запросов, и самые простые из них и, к тому же наиболее часто используемые – это запросы на выборку. Цель запроса на выборку состоит в создании результирующей таблицы, в которой отображаются только нужные по условию запроса данные из базовых таблиц.
* ***Формы*** – это объекты с помощью которых в базу вводят новые данные или просматривают имеющиеся.
* ***Отчеты*** – это объекты, с помощью которых данные выдают на принтер в удобном и наглядном виде.
* ***Макросы*** – это макрокоманды. Если какие-то операции с базой производятся особенно часто, то имеет смысл сгруппировать несколько команд в один макрос и назначить его выделенной комбинации клавиш.
* ***Модули*** – это программные процедуры, написанные на каком-либо языке программирования.

***втоматизированная информационная система*** (Automated information system, AIS) - это совокупность программных и аппаратных средств, предназначенных для хранения и (или) управления данными и информацией, а также для производства вычислений.

Основная цель АИС - хранение, обеспечение эффективного поиска и передачи информации по соответствующим запросам для наиболее полного удовлетворения информационных запросов большого числа пользователей. К основным принципам автоматизации информационных процессов относят: окупаемость, надежность, гибкость, безопасность, дружественность, соответствие стандартам.

*По назначению функционирующей информации* ИС делятся на: государственные, юридические (законодательные), деловые, финансовые, научно-технические, учебные, социальные, развлекательные и другие

*По отраслям применения* выделяют деловую, профессиональную, потребительскую информацию и электронную коммерцию.

*По уровню управления* выделяют стратегические, тактические и оперативные информационные системы.

*По уровню применения технических средств* ИС делят на автоматизированные и неавтоматизированные. При этом автоматизированные подразумевают автоматизацию от отдельных процессов и задач до уровня автоматизации предприятий, учреждений и их совокупности в масштабах территории.

*По типам информации* - документальные, фактографические и документально-фактографические ИС.

*Документальные ИС* включают информационно-поисковые системы (ИПС), информационно-логические и информационно-семантические системы.

*Фактографические ИС* делятся на две категории:

1) системы обработки данных (СОД),

2) автоматизированные информационные системы (АИС) и автоматизированные системы управления (АСУ).

*Документально-фактографические ИС* содержат:

1) автоматизированные документально-фактографические информационно-поисковые системы научно-технической информации (АДФИПС НТИ) и

2) автоматизированные документально-фактографические информационно-поисковые системы в автоматизированной системе нормативно-методического обеспечения управления (АДФИПС в АСНМОУ).

Выделяют и такие ИС, как: бухгалтерские, банковские, ИС рынка ценных бумаг, ИС управления (ИСУ), системы поддержки принятия решений (СППР), экспертные системы (ЭС), гибридные экспертные системы (ГЭС), ИС мониторинга (ИСМ) и др.

Выделяют четыре типа АИС:

1) Охватывающий один процесс (операцию) в одной организации;

2) Объединяющий несколько процессов в одной организации;

3) Обеспечивающий функционирование одного процесса в масштабе нескольких взаимодействующих организаций;

4) Реализующий работу нескольких процессов или систем в масштабе нескольких организаций.

14. **Уровни представления данных. Трехуровневая архитектура ANSI/SPARC**

обобщенная трехуровневая модель архитектуры СУБД, включающая концептуальный, внешний и внутренний уровни (рис. 1.4).

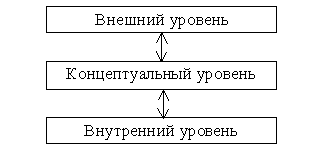


Рис.1.4. Уровни представления данных

**Концептуальный уровень**архитектуры ANSI/SPARC служит для поддержки единого взгляда на базу данных, общего для всех её приложений и независимого от них. Концептуальный уровень представляет собой формализованную информационно-логическую модель ПО. Описание этого представления называется *концептуальной схемой*.

**Внутренний уровень**архитектуры поддерживает представление БД в среде хранения – хранимую базу данных. На этом архитектурном уровне БД представлена в полностью “материализованном” виде, тогда как на других уровнях идёт работа на уровне отдельных экземпляров или множества экземпляров записей. Описание БД на внутреннем уровне называется *внутренней схемой*или*схемой хранения.*

**Внешний уровень**архитектуры БД предназначен для различных групп пользователей. Описания таких представлений называются *внешними схемами.*В системе БД могут одновременно поддерживаться несколько внешних схем для различных групп пользователей или задач.

Совокупность схем всех уровней называется *схемой базы данных*

15. **Трехуровневая архитектура ANSI/SPARC. Внутренняя схема базы  данных.**

обобщенная трехуровневая модель архитектуры СУБД, включающая концептуальный, внешний и внутренний уровни (рис. 1.4).

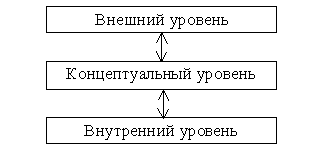


Рис.1.4. Уровни представления данных

**Концептуальный уровень**архитектуры ANSI/SPARC служит для поддержки единого взгляда на базу данных, общего для всех её приложений и независимого от них. Концептуальный уровень представляет собой формализованную информационно-логическую модель ПО. Описание этого представления называется *концептуальной схемой*.

**Внутренний уровень**архитектуры поддерживает представление БД в среде хранения – хранимую базу данных. На этом архитектурном уровне БД представлена в полностью “материализованном” виде, тогда как на других уровнях идёт работа на уровне отдельных экземпляров или множества экземпляров записей. Описание БД на внутреннем уровне называется *внутренней схемой*или*схемой хранения.*

**Внешний уровень**архитектуры БД предназначен для различных групп пользователей. Описания таких представлений называются *внешними схемами.*В системе БД могут одновременно поддерживаться несколько внешних схем для различных групп пользователей или задач.

Совокупность схем всех уровней называется *схемой базы данных*

16. **Трехуровневая архитектура ANSI/SPARC. Концептуальный уровень.**

обобщенная трехуровневая модель архитектуры СУБД, включающая концептуальный, внешний и внутренний уровни (рис. 1.4).

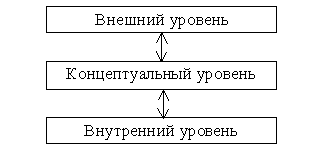


Рис.1.4. Уровни представления данных

**Концептуальный уровень**архитектуры ANSI/SPARC служит для поддержки единого взгляда на базу данных, общего для всех её приложений и независимого от них. Концептуальный уровень представляет собой формализованную информационно-логическую модель ПО. Описание этого представления называется *концептуальной схемой*.

**Внутренний уровень**архитектуры поддерживает представление БД в среде хранения – хранимую базу данных. На этом архитектурном уровне БД представлена в полностью “материализованном” виде, тогда как на других уровнях идёт работа на уровне отдельных экземпляров или множества экземпляров записей. Описание БД на внутреннем уровне называется *внутренней схемой*или*схемой хранения.*

**Внешний уровень**архитектуры БД предназначен для различных групп пользователей. Описания таких представлений называются *внешними схемами.*В системе БД могут одновременно поддерживаться несколько внешних схем для различных групп пользователей или задач.

Совокупность схем всех уровней называется *схемой базы данных*

17. **Трехуровневая архитектура ANSI/SPARC. Внешний уровень.**

обобщенная трехуровневая модель архитектуры СУБД, включающая концептуальный, внешний и внутренний уровни (рис. 1.4).

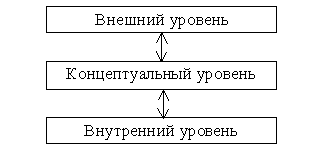


Рис.1.4. Уровни представления данных

**Концептуальный уровень**архитектуры ANSI/SPARC служит для поддержки единого взгляда на базу данных, общего для всех её приложений и независимого от них. Концептуальный уровень представляет собой формализованную информационно-логическую модель ПО. Описание этого представления называется *концептуальной схемой*.

**Внутренний уровень**архитектуры поддерживает представление БД в среде хранения – хранимую базу данных. На этом архитектурном уровне БД представлена в полностью “материализованном” виде, тогда как на других уровнях идёт работа на уровне отдельных экземпляров или множества экземпляров записей. Описание БД на внутреннем уровне называется *внутренней схемой*или*схемой хранения.*

**Внешний уровень**архитектуры БД предназначен для различных групп пользователей. Описания таких представлений называются *внешними схемами.*В системе БД могут одновременно поддерживаться несколько внешних схем для различных групп пользователей или задач.

Совокупность схем всех уровней называется *схемой базы данных*

18. **Предметная область. Сущность. Атрибут сущности. Экземпляр сущности. Связь. Концептуальная схема базы данных.**

**Предме́тная о́бласть** — множество всех предметов, свойства которых и отношения между которыми рассматриваются в научной теории. В логике — подразумеваемая **область** возможных значений **предметных** переменных логического языка. **Предметная область**представляется множеством фрагментов, например, предприятие - цехами, дирекцией, бухгалтерией и т. д.

**Сущность** в **базе данных** – это любой объект в **базе данных**, который можно выделить исходя из сути предметной области для которой разрабатывается эта **база данных**. Разработчик **базы данных** должен уметь правильно определять **сущности**.

**Атрибут** представляет свойство, которое описывает некоторую характеристику объекта. Каждый столбец должен хранить один **атрибут сущности**. А каждая строка представляет отдельный объект или экземпляр **сущности**.

**Связь** работает путем сопоставления **данных** в ключевых столбцах, обычно столбцах (или полях), которые имеют одно и то же имя в обеих таблицах. В большинстве случаев **связь** соединяет основной ключ или уникальный столбец идентификатора для каждой строки, от одной таблицы к полю в другой таблице.

**Концептуальная модель** - это отражение предметной области, для которой разрабатывается база данных. Не вдаваясь в теорию, отметим, что это некая диаграмма с принятыми обозначениями элементов. Так, все объекты, обозначающие вещи, обозначаются в виде прямоугольника. Атрибуты, характеризующие объект - в виде овала, а связи между объектами - ромбами. Мощность связи обозначаются стрелками (в направлении, где мощность равна многим - двойная стрелка, а со стороны, где она равна единице - одинарная).

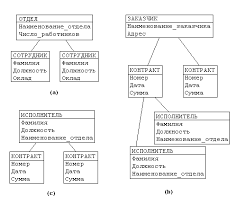
19. **Основные модели данных. Модель данных. Иерархическая модель данных. Пример иерархической базы данных. Достоинства/недостатки.**

Логические модели:

* [Иерархическая модель](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* [Сетевая модель](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* [Реляционная модель](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)

модель данных есть формальная теория представления и обработки данных в системе управления базами данных, которая включает, по меньшей мере, три аспекта: аспект структуры: методы описания типов и логических структур данных в базе данных;

**Иерархическая модель данных** — это **модель данных**, где используется представление базы **данных** в виде древовидной (**иерархической**) структуры, состоящей из объектов (**данных**) различных уровней. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня.



**Основными достоинствами иерархической модели данных являются:**

* эффективное использование памяти ЭВМ;
* высокая скорость выполнения основных операций над данными;
* удобство работы с иерархически упорядоченной информацией;
* простота при работе с небольшим объемом данных так как, иерархический принцип соподчиненности понятий является естественным для многих задач.

**К недостаткам иерархической модели представления данных относятся:**

* громоздкость такой модели для обработки информации с достаточно сложными логическими связями;
* трудность в понимании ее функционирования обычным пользователем.
* трудность в применении к данным со сложной внутренней взаимосвязью
* исключительно навигационный принцип доступа к данным

20. **Основные модели данных. Модель данных. Сетевая модель. Пример сетевой модели. Достоинства/недостатки.**

Логические модели:

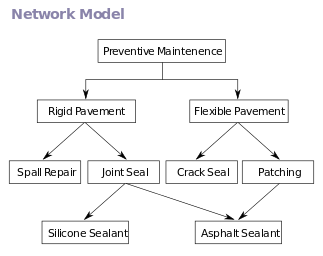
* [Иерархическая модель](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* [Сетевая модель](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* [Реляционная модель](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* модель данных есть формальная теория представления и обработки данных в системе управления базами данных, которая включает, по меньшей мере, три аспекта: аспект структуры: методы описания типов и логических структур данных в базе данных;

**Сетевая модель данных** — логическая [модель данных](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), являющаяся расширением [иерархического подхода](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных.

Разница между [иерархической моделью данных](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и сетевой состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков.

Сетевая БД состоит из набора экземпляров определенного типа записи и набора экземпляров определенного типа связей между этими записями.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

* каждый экземпляр типа записи P является предком только в одном экземпляре типа связи L;
* каждый экземпляр типа записи C является потомком не более чем в одном экземпляре типа связи L.
* 

## **Достоинства**

Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности.

## **Недостатки**

Недостатком сетевой модели данных являются высокая сложность и жесткость схемы БД, построенной на её основе. Поскольку логика процедуры выборки данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения. Другими словами, если необходимо изменить структуру данных, то нужно изменить и приложение.

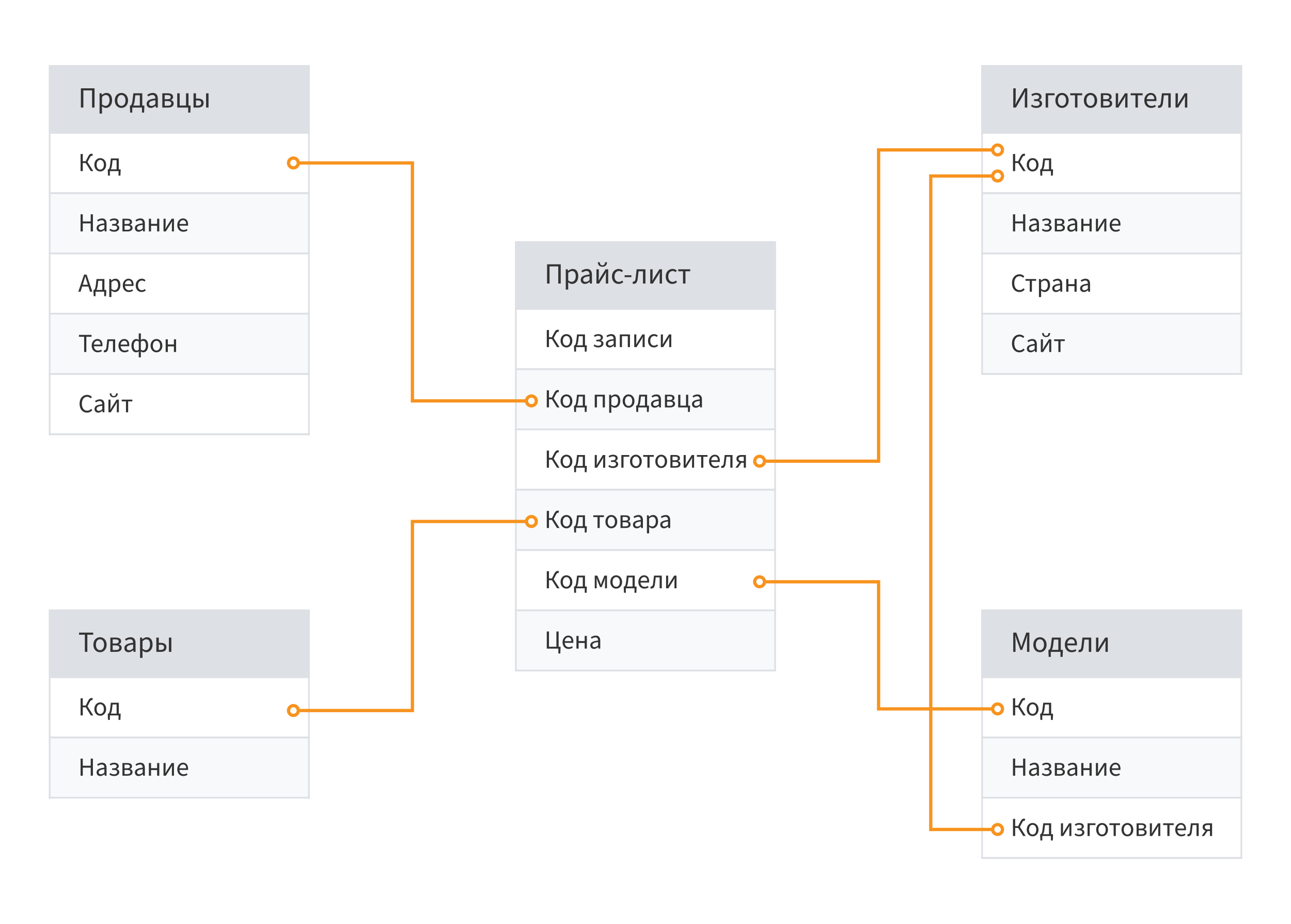
21. Основные модели данных. Модель данных. Реляционная модель данных. Пример реляционной модели базы данных. Достоинства/недостатки.

Логические модели:

* [Иерархическая модель](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* [Сетевая модель](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* [Реляционная модель](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)

модель данных есть формальная теория представления и обработки данных в системе управления базами данных, которая включает, по меньшей мере, три аспекта: аспект структуры: методы описания типов и логических структур данных в базе данных;

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).



|  |  |
| --- | --- |
| **Достоинства** | **Недостатки** |
| ·        Эта модель данных отображает информацию в наиболее простой для пользователя форме  ·        Основана на развитом математи-ческом аппарате, который позволяет достаточно лаконично описать основные операции над данными.  ·        Позволяет создавать языки манипулирования данными не процедурного типа.  ·        Манипулирование данными на уровне выходной БД и возможность изменения. | ·        Самый медленный доступ к данным.  ·        Трудоемкость разработки |

22. Реляционная модель данных. Сущность. Отношение. Кортеж. Атрибуты. Степень отношения. Кардинальное число отношения. Домен. Тип данных.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

**Сущность** в **базе данных** – это любой объект в **базе данных**, который можно выделить исходя из сути предметной области для которой разрабатывается эта **база данных**. Разработчик **базы данных** должен уметь правильно определять **сущности**.

Отношение обычно имеет простую графическую интерпретацию в виде таблицы, столбцы которой соответствуют атрибутам, а строки — кортежам, а в «ячейках» находятся значения атрибутов в кортежах. Тем не менее, в строгой реляционной модели *отношение* не является *таблицей*, *кортеж* — это не *строка*, а *атрибут* — это не *столбец*[[2]](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)#cite_note-DinD-2)[[3]](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)#cite_note-3).

**Атрибут** представляет свойство, которое описывает некоторую характеристику объекта. Каждый столбец должен хранить один **атрибут сущности**. А каждая строка представляет отдельный объект или экземпляр **сущности**.

В реляционных **базах данных кортеж** — это элемент отношения, строка таблицы;

количество кортежей называют **кардинальным числом отношения**(кардинальностью), или мощностью **отношения**. Количество атрибутов называют степенью, или «арностью» **отношения**; **отношение** с одним атрибутом называется унарным, с двумя — бинарным и т. ... В отношении нет двух одинаковых элементов (кортежей).

**Домен** в [реляционной модели данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) — [тип данных](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), то есть [множество](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE) допустимых значений[[1]](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BC%D0%B5%D0%BD_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)#cite_note-DateWritings-1).

**В поля БД можно вводить данные следующих типов.**

* Текстовый. В поля такого **типа** помещают текст или комбинацию текстовых и числовых значений. ...
* Поле MEMO. Длинный текст (до 65535) символов.
* Числовой. ...
* Дата/время. ...
* Денежный. ...
* Счетчик. ...
* Логический. ...
* Поле объекта OLE.

23. Реляционная модель данных. Ключи отношений. Типы ключей. Пример применения ключей.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

* первичный ключ;
* внешний ключ;
* простой и составной ключ;
* отношение, типы отношений;
* искусственный и естественный ключи;
* главная (master) и подчиненная (detail) таблицы.

**Типы ключей баз данных**

* Foreign Key. Оператор SQL FOREIGN KEY (внешний **ключ**) служит для указания в одной таблице на первичный **ключ** (оператор SQL PRIMARY KEY) в другой. Оператор SQL FOREIGN KEY имеет следующий синтаксис: ...
* Unique Key. UNIQUE KEY позволяет однозначно идентифицировать каждую запись в таблице.

CREATE TABLE books(

Id INTEGER PRIMARY KEY,

title TEXT NOT NULL,

count\_page INTEGER NOT NULL CHECK (count\_page >0),

price REAL CHECK (price >0),

auth\_id INTEGER NOT NULL,

FOREIGN KEY (auth\_id) REFERENCES auth(id)

);

CREATE TABLE auth(

id INTEGER PRIMARY KEY,

name TEXT NOT NULL,

age INTEGER  CHECK (age >16)

);

24. Реляционная модель данных. Отношение, схема отношения, схема БД. Множество элементов отношения.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

*Схема отношения* - это именованное множество пар {имя атрибута – имя домена (или типа, если понятие домена не поддерживается)}. Степень или "арность" схемы отношения - мощность этого множества. Степень отношения СОТРУДНИКИ равна четырем, то есть оно является 4-арным. Если все атрибуты одного отношения определены на разных доменах, осмысленно использовать для именования атрибутов имена соответствующих доменов (не забывая, конечно, о том, что это является всего лишь удобным способом именования и не устраняет различия между понятиями домена и атрибута).

**Отноше́ние** — фундаментальное понятие [реляционной модели данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). По этой причине модель и называется *реляционной*

25. Реляционная модель данных. Свойства реляционных отношений.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

Основные свойства отношения[[1]](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)" \l "cite_note-Date-1)[[2]](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)#cite_note-DinD-2):

* В отношении нет двух одинаковых элементов (кортежей).
* Порядок кортежей в отношении не определён.
* Порядок атрибутов в заголовке отношения не определён.

Подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости), называется [потенциальным ключом](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87). Поскольку все кортежи в отношении по определению уникальны, в любом отношении должен существовать по крайней мере один потенциальный ключ.

26. Реляционная модель данных. Виды реляционных отношений. Связывание таблиц.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

*Реляционные базы* данных представляют собой [базы данных](https://www.oracle.com/ru/database/what-is-database.html), которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. Реляционные базы данных основаны на реляционной модели — интуитивно понятном, наглядном табличном способе представления данных. Каждая строка, содержащая в таблице такой базы данных, представляет собой запись с уникальным идентификатором, который называют *ключом*. Столбцы таблицы имеют атрибуты данных, а каждая запись обычно содержит значение для каждого атрибута, что дает возможность легко устанавливать взаимосвязь между элементами данных.

Она обеспечила стандартный способ представления данных и отправки запросов, которые могли быть использованы в любых приложениях. Разработчики уяснили, что таблицы являются ключевым преимуществом реляционных баз данных, так как обеспечивают интуитивно понятный, эффективный и гибкий способ хранения структурированной информации и получения к ней доступа.

**Виды и типы связей между таблицами в реляционных базах данных**

Давайте теперь рассмотрим то, **как могут быть связаны таблицы в реляционных базах данных**. Сразу скажу, что всего существует **три вида связей между таблицами баз данных**:

• связь один к одному;

**Связь один к одному** – самая редко встречаемая связь между таблицами. В 97 случаях из 100, если вы видите такую связь, вам необходимо объединить две таблицы в одну.

• связь один ко многим;

**Связь один ко многим в реляционных базах данных** реализуется тогда, когда объекту А может принадлежать или же соответствовать несколько объектов Б, но объекту Б может соответствовать только один объект А.

• связь многие ко многим.

**Связь многие ко многим** реализуется в том случае, когда нескольким объектам из таблицы А может соответствовать несколько объектов из таблицы Б, и в тоже время нескольким объектам из таблицы Б соответствует несколько объектов из таблицы А.

27. Реляционная модель данных. Целостность данных. Основные требования. Ограничение целостности. Значение NULL.Целостность сущностей. Целостность связей.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

В теории баз данных под целостностью понимают свойство соответствия структуры и содержания базы данных предметной области. В реляционной модели данных определяются **два основных требования**, при которых обеспечивается целостность данных: **целостность сущностей** и **целостность ссылок**.

Каждый объект или наблюдение представляется в [реляционной базе](https://wiki.loginom.ru/articles/relational-database.html) как группа взаимосвязанных элементов данных — кортеж некоторого отношения. Требование целостности сущностей заключается в том, чтобы каждый кортеж любого отношения отличался от другого кортежа этого отношения (т.е. любое отношение должно обладать первичным ключом).

В теории реляционных баз данных принято выделять четыре типа ограничений целостности[[2]](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#cite_note-%D0%94%D0%B5%D0%B9%D1%82_%D0%92%D0%B2%D0%A1%D0%91%D0%94_8-2):353:

* *Ограничением базы данных* называется ограничение на значения, которые разрешено принимать указанной базе данных.
* *Ограничением* [*переменной отношения*](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)) называется ограничение на значения, которые разрешено принимать указанной переменной отношения.
* *Ограничением* [*атрибута*](https://ru.wikipedia.org/w/index.php?title=%D0%90%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)&action=edit&redlink=1) называется ограничение на значения, которые разрешено принимать указанному атрибуту.
* *Ограничение* [*типа*](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BC%D0%B5%D0%BD_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) представляет собой ни что иное, как определение множества значений, из которых состоит данный тип.

Примером распространённого ограничения уровня переменной отношения является [потенциальный ключ](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87); примером распространённого ограничения уровня базы данных является [внешний ключ](https://ru.wikipedia.org/wiki/%D0%92%D0%BD%D0%B5%D1%88%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87).

Значение NULL указывает на то, что поле может быть пустым.

**Целостность сущностей**

Первое ограничение целостности касается первичных ключей базовых отношений. В базовом отношении ни один атрибут первичного , ключа не может содержать отсутствующих значений, обозначаемых как NULL.

Пустое значение: Указывает, что значение атрибута в настоящий момент неизвестно или неприемлемо для этого кортежа.

По определению, первичный ключ — это минимальный идентификатор, который используется для уникальной идентификации кортежей. Это значит, что никакое подмножество первичного ключа не может быть достаточным для уникальной идентификации кортежей. Если допустить присутствие NULL в любой части первичного ключа, это равносильно утверждению, что не все его атрибуты необходимы для уникальной идентификации кортежей, что противоречит определению первичного ключа.

**Ссылочная целостность**

Второе ограничение целостности касается внешних ключей.

Ссылочная целостность - необходимое качество реляционной базы данных, заключающееся в отсутствии в любом её отношении внешних ключей, ссылающихся на несуществующие кортежи.

Если в отношении существует внешний ключ, то значение внешнего ключа должно либо соответствовать значению потенциального ключа некоторого кортежа в его базовом отношении либо внешний ключ должен полностью состоять из значений NULL.

Например, считается допустимым создание записи с информацией о новом сотруднике с указанием NULL вместо номера отделения, в котором этот сотрудник работает. Такая ситуация может иметь место в том случае, когда сотрудник зачислен в штат компании, но еще не приписан к какому-то конкретному отделению.

28. Реляционная модель данных. Операции над отношениями. Объединение. Пример объединения.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

**тип данных. Операции над множествами.**

Типа данных задается таким свойством множеств как ограниченность. Для ***однотипных*** элементов появляется возможность сравнения. Над такими однотипными множествами возможны операции:***объединение, пересечение, вычитание (дополнение*), *декартово произведение.***

*Объединение* выполняется над двумя совместными отношениями*R1,R2*с идентичной структурой*.*В результате операции строится новое отношение*R =R1UR2*, которое имеет тот же состав атрибутов и совокупность кортежей исходных отношений. В результирующее отношение по определению не включаются дубликаты кортежей. Ниже приведены исходные отношения:*R1*(табл. 2) и*R2*(табл. 3) и результат объединения –*R*(табл. 4).

Изображение выглядит как текст, снимок экрана, компьютер, ноутбук

Автоматически созданное описание

29. Реляционная модель данных. Операции над отношениями. Пересечение. Пример применения пересечения.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

**тип данных. Операции над множествами.**

Типа данных задается таким свойством множеств как ограниченность. Для ***однотипных*** элементов появляется возможность сравнения. Над такими однотипными множествами возможны операции:***объединение, пересечение, вычитание (дополнение*), *декартово произведение.***

*Пересечение* выполняется над двумя совместными отношениями*R1,R2.*Результирующее отношение*RP=R1)R2*содержит кортежи, которые есть в каждом из исходных. Результат имеет тот же состав атрибутов, что и исходные отношения. Пересечение отношений*R1*и*R2*дает отношение*RP*(табл. 5).

Изображение выглядит как текст, снимок экрана, компьютер, ноутбук

Автоматически созданное описание

30. Реляционная модель данных. Операции над отношениями. Вычитание. Пример применения вычитания.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

**тип данных. Операции над множествами.**

Типа данных задается таким свойством множеств как ограниченность. Для ***однотипных*** элементов появляется возможность сравнения. Над такими однотипными множествами возможны операции:***объединение, пересечение, вычитание (дополнение*), *декартово произведение.***

*Вычитание* выполняется над двумя совместными отношениями*R1,R2.*В результате строится новое отношение*RV=R1-R2*с идентичным набором атрибутов, содержащее кортежи первого отношения*R1*, которые не входят в отношение*R2.*Вычитание отношения*R2*из*R1*дает отношение*RV*(табл. 6).

Изображение выглядит как текст, снимок экрана, компьютер, ноутбук

Автоматически созданное описание

31. Реляционная модель данных. Операции над отношениями. Декартово произведение. Пример применения декартового произведения.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

**Операции над множествами.**

Типа данных задается таким свойством множеств как ограниченность. Для ***однотипных*** элементов появляется возможность сравнения. Над такими однотипными множествами возможны операции: ***объединение, пересечение, вычитание (дополнение*), *декартово произведение.***

**ДЕКАРТОВО ПРОИЗВЕДЕНИЕ МНОЖЕСТВ** [Cartesian product] — множество А × В всех упорядоченных пар элементов (**a, b**), из которых **a** принадлежит множеству A, **b** — множеству B.

Соединения - это подмножества декартова произведения. Так как декартово произведение n таблиц - это таблица, содержащая все возможные строки r, такие, что r является сцеплением какой-либо строки из первой таблицы, строки из второй таблицы, ... и строки из n-й таблицы

*Декартово произведение* выполняется над двумя отношениями*R1*и*R2,*имеющими в общем случае разный состав атрибутов. В результате образуется новое отношение*RD*=*R1 x R2,* которое включает все атрибуты исходных отношений. Результирующее отношение состоит из всевозможных сочетаний кортежей исходных отношений*.* Число кортежей (мощность) отношения-произведения равно произведению мощностей исходных отношений.

Декартово произведение отношений *R1*(табл. 7) и*R2*(табл. 8) дает новое отношение*RD*(табл. 9), которое содержит все атрибуты исходных отношений. В него целесообразно добавить атрибут*Оценка*для записи результатов экзамена.

Таблица 7.

|  |  |
| --- | --- |
| Код дисциплины | Наименование |
| Д1 | Математика |
| Д2 | Информатика |

Таблица 8.

|  |  |
| --- | --- |
| Номер студента | ФИО студента |
| 11 | Иванов И.И. |
| 12 | Петров П.П. |
| 13 | Сидоров С.С. |

Таблица 9.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер студента | ФИО студента | Код дисциплины | Наименование |
| 11 | Иванов И.И. | Д1 | Математика |
| 12 | Петров П.П. | Д1 | Математика |
| 13 | Сидоров С.С. | Д1 | Математика |
| 11 | Иванов И.И. | Д2 | Информатика |
| 12 | Петров П.П. | Д2 | Информатика |
| 13 | Сидоров С.С. | Д2 | Информатика |

32. Реляционная модель данных. Специальные реляционные операторы. Выборка. Пример применения выборки.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

*Выборка*выполняется над одним отношением*R.*Для отношения по заданному условию (предикату) осуществляется выборка подмножества кортежей. Результирующее отношение имеет ту же структуру, что и исходное, но число его кортежей будет меньше (или равно) числа кортежей исходного отношения. Например, выбрать студентов, сдавших математику на отлично*(Код дисциплины = Д1)AND(Оценка = 5)*(табл. 10).

Таблица 10.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер студента | ФИО студента | Код дисциплины | Наименование |
| 11 | Иванов И.И. | Д1 | Математика |
| 13 | Сидоров С.С. | Д1 | Математика |

33. Реляционная модель данных. Специальные реляционные операторы. Проекция. Пример применения проекция.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

*Проекция* выполняется над одним отношением *R.* Операция формирует новое отношение *RPR*с заданным подмножеством атрибутов исходного отношения *R.* Оно может содержать меньше кортежей, так как после отбрасывания в исходном отношении *R* части атрибутов (и возможного исключения первичного ключа) могут образоваться кортежи-дубли, которые из результирующего отношения исключаются по определению.

Ниже приведен пример исходного отношения *R*(табл. 11) и результат проекции этого отношения на два его атрибута*- Должность*и*Номер отдела*(табл. 12).

Таблица 11.

|  |  |  |
| --- | --- | --- |
| ФИО | Номер отдела | Должность |
| Иванов И.И. | 01 | Инженер |
| Петров П.П. | 02 | Инженер |
| Нестеров Н.Н. | 01 | Инженер |

Таблица 12.

|  |  |
| --- | --- |
| Номер отдела | Должность |
| 01 | Инженер |
| 02 | Инженер |
| 03 | Лаборант |

34. Реляционная модель данных. Специальные реляционные операторы. Соединение. Пример применения соединения.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

*Соединение* выполняется для заданного условия соединения над двумя логически связанными отношениями. Исходные отношения*R1*и*R2*имеют разные структуры, в которых есть одинаковые атрибуты – внешние ключи. Операция соединения формирует новое отношение, структура которого является совокупностью всех атрибутов исходных отношений. Результирующие кортежи формируются соединением каждого кортежа из*R1*с теми кортежами*R2,*для которых выполняется условие соединения. В зависимости от этого условия соединение называется:*естественным –*равенство значений общих атрибутов отношений*R1*и*R2*;*эквисоединением –*равенство значений атрибутов, входящих в условие соединения;*тета-соединением –*другой знак сравнения.

Операция соединения имеет большое значение для РБД, так как в процессе нормализации отношений исходное отношение разбивается на несколько более мелких отношений, которые при выполнении запросов пользователя требуется, как правило, вновь соединять для восстановления исходного отношения.

35. Реляционная модель данных. Специальные реляционные операторы. Деление. Пример применения  деления.

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы).

*Деление* выполняется над двумя отношениями*R1*и*R2,*имеющими в общем случае разные структуры и часть одинаковых атрибутов. В результате образуется новое отношение, содержащее атрибуты 1-го операнда, отсутствующие во 2-м операнде, и кортежи 1-го операнда, которые совпали с кортежами 2-го. Для выполнения этой операции 2-й операнд должен содержать лишь атрибуты, совпадающие с атрибутами 1-го*.*

Например, чтобы узнать, кто из студентов получил по математике 5 и по информатике 4, надо разделить отношения *Экзаменационная ведомость*на вспомогательное отношение*Мат5Физ4 (Наименование, Оценка)*с двумя кортежами:*Математика, 5*и*Информатика, 4*. В результате получим отношение*Итог (Номер студента, ФИО студента, Код дисциплины)*с одним кортежем –*13, Сидоров, Д1*.

36. Язык SQL. Свойства таблиц. Ограничения на содержимое таблиц. Основы синтаксиса языка SQL.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

**Свойства** обычной **таблицы**

**Таблицы** содержат строки. Каждая строка представляет собой одну запись. В пространственной таблице одна строка соответствует одному объекту. Все строки в таблице имеют одни и те же столбцы.

Когда вы создаете таблицу ( или, когда вы ее изменяете ), вы можете помещать ограничение на значения которые могут быть введены в пол. Если вы это сделали, SQL будет отклонять любые значения которые нарушают критерии которые вы определили. Имеется два основных типа ограничений - ограничение столбца и ограничение таблицы. Различие между ними в том, что ограничение столбца применяется только к индивидуальным столбцам, в то врем как ограничение таблицы применяется к группам из одного и более столбцов.

SELECT \* FROM 'cities'

INSERT INTO 'cities' SET 'name' = 'Санкт-Петербург'

С помощью SQL можно не только добавлять и читать данные, но и:

* удалять и обновлять записи в таблицах;
* создавать и редактировать сами таблицы;
* производить операции над данными: считать сумму, получать самое большое или малое значение, и так далее;
* настраивать работу сервера СУБД.

CREATE TABLE weather\_log (

id INT AUTO\_INCREMENT PRIMARY KEY,

city\_id INT,

day DATE,

temperature INT,

cloud TINYINT DEFAULT 0

);

37. Язык SQL. Типы данных SQL.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

#### Числовые типы данных

* BIT: хранит значение 0 или 1. Фактически является аналогом булевого типа в языках программирования. Занимает 1 байт.
* TINYINT: хранит числа от 0 до 255. Занимает 1 байт. Хорошо подходит для хранения небольших чисел.
* SMALLINT: хранит числа от –32 768 до 32 767. Занимает 2 байта
* INT: хранит числа от –2 147 483 648 до 2 147 483 647. Занимает 4 байта. Наиболее используемый тип для хранения чисел.
* BIGINT: хранит очень большие числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, которые занимают в памяти 8 байт.
* DECIMAL: хранит числа c фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой.

#### Типы данных, представляющие дату и время

* DATE: хранит даты от 0001-01-01 (1 января 0001 года) до 9999-12-31 (31 декабря 9999 года). Занимает 3 байта.
* TIME: хранит время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт.

Может иметь форму TIME(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.

* DATETIME: хранит даты и время от 01/01/1753 до 31/12/9999. Занимает 8 байт.
* DATETIME2: хранит даты и время в диапазоне от 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999. Занимает от 6 до 8 байт в зависимости от точности времени.

Может иметь форму DATETIME2(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.

* SMALLDATETIME: хранит даты и время в диапазоне от 01/01/1900 до 06/06/2079, то есть ближайшие даты. Занимает от 4 байта.
* DATETIMEOFFSET: хранит даты и время в диапазоне от 0001-01-01 до 9999-12-31. Сохраняет детальную информацию о времени с точностью до 100 наносекунд. Занимает 10 байт.

#### Строковые типы данных

* CHAR: хранит строку длиной от 1 до 8 000 символов. На каждый символ выделяет по 1 байту. Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом CHAR(10) будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.

* VARCHAR: хранит строку. На каждый символ выделяется 1 байт. Можно указать конкретную длину для столбца - от 1 до 8 000 символов, например, VARCHAR(10). Если строка должна иметь больше 8000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб: VARCHAR(MAX).

Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

В отличие от типа CHAR если в столбец с типом VARCHAR(10) будет сохранена строка в 5 символов, то в столце будет сохранено именно пять символов.

* NCHAR: хранит строку в кодировке Unicode длиной от 1 до 4 000 символов. На каждый символ выделяется 2 байта. Например, NCHAR(15)
* NVARCHAR: хранит строку в кодировке Unicode. На каждый символ выделяется 2 байта.Можно задать конкретный размер от 1 до 4 000 символов: . Если строка должна иметь больше 4000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб.

38. Язык SQL. Создание таблицы. Описание столбца таблицы. Описание первичного ключа. Описание вторичного ключа. Практические примеры.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

CREATE TABLE название\_таблицы

(название\_столбца1 тип\_данных атрибуты\_столбца1,

 название\_столбца2 тип\_данных атрибуты\_столбца2,

 ................................................

 название\_столбцаN тип\_данных атрибуты\_столбцаN,

 атрибуты\_таблицы

)

CREATE TABLE Customers

(

    Id INT,

    Age INT,

    FirstName NVARCHAR(20),

    LastName NVARCHAR(20),

    Email VARCHAR(30),

    Phone VARCHAR(20)

)

Структура создаваемой базы данных и наименования таблиц показаны на Рис. Ж.5. Имя базы данных, создаваемой по умолчанию – «TagsDb». Далее в документе приведено краткое описаниетаблиц БД.

Оператор SQL PRIMARY KEY (Первичный ключ) — это параметр, который устанавливается для однозначной идентификации той или иной записи в таблице. Значения SQL PRIMARY KEY должны быть всегда уникальны, а так же не содержать значений NULL.

CREATE TABLE table\_name (

Id int NOT NULL,

PRIMARY KEY (Id)

)

CREATE TABLE Planets (

ID int NOT NULL,

PlanetName varchar(10),

Radius float (10),

SunSeason float(10),

OpeningYear int,

HavingRings bit,

Opener varchar(30)

PRIMARY KEY (ID)

)

**Внешние ключи** позволяют установить связи между таблицами. **Внешний ключ** устанавливается для столбцов из зависимой, подчиненной таблицы, и указывает на один из столбцов из главной таблицы. Как правило, **внешний ключ** указывает на первичный **ключ** из связанной главной таблицы.

CREATE TABLE Customers

(

    Id INT PRIMARY KEY AUTO\_INCREMENT,

    Age INT,

    FirstName VARCHAR(20) NOT NULL,

    LastName VARCHAR(20) NOT NULL,

    Phone VARCHAR(20) NOT NULL UNIQUE

);

CREATE TABLE Orders

(

    Id INT PRIMARY KEY AUTO\_INCREMENT,

    CustomerId INT,

    CreatedAt Date,

    FOREIGN KEY (CustomerId)  REFERENCES Customers (Id)

);

39. Язык SQL. Модификация таблицы. Добавление нового столбца. Удаление первичного ключа. Изменение/удаление значения по умолчанию. Практические примеры.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

|  |  |
| --- | --- |
| INSERT | Добавление записей (строк) в таблицу БД |
| UPDATE | Обновление данных в столбце таблицы БД |
| DELETE | Удаление записей из таблицы БД |

1. CREATE TABLE product
2. (
3. maker char (1) NOT NULL,
4. model varchar (4) NOT NULL,
5. type varchar (7) NOT NULL
6. );

Оператор **INSERT** вставляет новые записи в таблицу. При этом значения столбцов могут представлять собой литеральные константы, либо являться результатом выполнения подзапроса. В первом случае для вставки каждой строки используется отдельный оператор **INSERT**; во втором случае будет вставлено столько строк, сколько возвращается подзапросом.

INSERT INTO Product

VALUES ('B', 1157, 'PC');

INSERT INTO Product\_D (model, maker)

VALUES (1157, 'B');

Удаление первичного ключа

ALTER TABLE user\_customer\_permission DROP PRIMARY KEY;

Назначить столбцу новое значение по умолчанию можно так:

ALTER TABLE products ALTER COLUMN price SET DEFAULT 7.77;

Заметьте, что это никак не влияет на существующие строки таблицы, а просто задаёт значение по умолчанию для последующих команд INSERT.

Чтобы удалить значение по умолчанию, выполните:

ALTER TABLE products ALTER COLUMN price DROP DEFAULT;

При этом по сути значению по умолчанию просто присваивается NULL. Как следствие, ошибки не будет, если вы попытаетесь удалить значение по умолчанию, не определённое явно, так как неявно оно существует и равно NULL.

40. Язык SQL. Модификация таблицы. Удаление таблицы. Добавление строк в таблицу.  Добавление строк в таблицу с использованием списка имен столбцов.  Добавление строк в таблицу по результатам запроса к БД. Практические примеры.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

|  |  |
| --- | --- |
| INSERT | Добавление записей (строк) в таблицу БД |
| UPDATE | Обновление данных в столбце таблицы БД |
| DELETE | Удаление записей из таблицы БД |

Оператор **DELETE** удаляет строки из временных или постоянных базовых таблиц, представлений или курсоров, причем в двух последних случаях действие оператора распространяется на те базовые таблицы, из которых извлекались данные в эти представления или курсоры. Оператор удаления имеет простой синтаксис:

DELETE FROM <имя таблицы >

[WHERE <предикат>];

41. Язык SQL. Модификация таблицы. Выборка данных из таблиц. Оператор SELECT. Описание.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

Для получения данных применяется команда SELECT. В упрощенном виде она имеет следующий синтаксис:

|  |  |
| --- | --- |
| 1 | SELECT список\_столбцов FROM имя\_таблицы |

42. Язык SQL. Арифметические функции. Строковые функции. Операторы и функции, возвращающие логическое значение. Практические примеры.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

43. Язык SQL. Модификация таблицы. Упорядочивание и группирование строк результирующей таблицы.  Выборка из нескольких таблиц. Практические примеры.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

Оператор ORDER BY позволяет отсортировать извлекаемые значения по определенному столбцу:

|  |  |
| --- | --- |
| 1  2  3 | SELECT \*  FROM Products  ORDER BY ProductName |

.  Выборка из нескольких таблиц

SELECT \* FROM table\_1, table\_2 WHERE table\_1.id > table\_2.user\_id

44. Язык SQL. Модификация таблицы. Удаление строк. Изменение содержимого строк.

**SQL** ([ˈɛsˈkjuˈɛl](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *structured query language* — «язык структурированных запросов») — [декларативный](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), применяемый для создания, модификации и управления данными в [реляционной базе данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Изначально SQL был основным способом работы пользователя с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

Для удаления таблицы из БД применяется команда DROP TABLE, после которой указывается название удаляемой таблицы. Например, удалим таблицу Clients:

|  |  |
| --- | --- |
| 1 | DROP TABLE Clients; |

**Добавление нового столбца**

Добавим в таблицу Customers новый столбец Address:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  ADD Address VARCHAR(50) NULL; |

В данном случае столбец Address имеет тип VARCHAR и для него определен атрибут NULL.

**Удаление столбца**

Удалим столбец Address из таблицы Customers:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  DROP COLUMN Address; |

**Изменение значения по умолчанию**

Установим в таблице Customers для столбца Age значение по умолчанию 22:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  ALTER COLUMN Age SET DEFAULT 22; |

**Изменение типа столбца**

Изменим в таблице Customers тип данных у столбца FirstName на CHAR(100) и установим для него атрибут NULL:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  MODIFY COLUMN FirstName CHAR(100) NULL; |

**Добавление и удаление внешнего ключа**

Пусть изначально в базе данных будут добавлены две таблицы, никак не связанные:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | CREATE TABLE Customers  (      Id INT PRIMARY KEY AUTO\_INCREMENT,      Age INT,      FirstName VARCHAR(20) NOT NULL,      LastName VARCHAR(20) NOT NULL  );  CREATE TABLE Orders  (      Id INT PRIMARY KEY AUTO\_INCREMENT,      CustomerId INT,      CreatedAt Date  ); |

Добавим ограничение внешнего ключа к столбцу CustomerId таблицы Orders:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Orders  ADD FOREIGN KEY(CustomerId) REFERENCES Customers(Id); |

При добавлении ограничений мы можем указать для них имя, используя оператор CONSTRAINT, после которого указывается имя ограничения:

|  |  |
| --- | --- |
| 1  2  3 | ALTER TABLE Orders  ADD CONSTRAINT orders\_customers\_fk  FOREIGN KEY(CustomerId) REFERENCES Customers(Id); |

В данном случае ограничение внешнего ключа называется orders\_customers\_fk. Затем по этому имени мы можем удалить ограничение:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Orders  DROP FOREIGN KEY orders\_customers\_fk; |

**Добавление и удаление первичного ключа**

Добавим в таблицу Products первичный ключ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE Products  (      Id INT,      Model VARCHAR(20)  );    ALTER TABLE Products  ADD PRIMARY KEY (Id); |

Теперь удалим первичный ключ:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Products  DROP PRIMARY KEY; |

**Обновление данных в базе (UPDATE)**

Функция обновления "UPDATE" (переводится с английского как "ОБНОВИТЬ") довольно часто используется в проектах сайтов. Как и в случае с функцией "DELETE", фкнция обновления не успокоится до тех пор, пока не обновит все поля, которые подходят под условия, если нет лимита на выборку. Поэтому необходимо задавать однозначные условия, чтобы вместо одной строки нечаянно не обновить половину таблицы. Приведём пример использования команды "UPDATE":

UPDATE `USERS` SET `NAME` = 'Мышь' WHERE `ID` = 3 LIMIT 1;

В этом примере, в таблие "USERS" будет установлено значение "Мышь" в столбец "NAME" у строки, в столбце "ID" которой стоит значение "3". Можно обновить сразу несколько столбцов у одной записи, передав значения через запятую. Попробуем обновить не только значение с толбце "NAME", но и значение в столбце "FOOD" используя один запрос:

UPDATE `USERS` SET `NAME` = 'Мышь', `FOOD` = 'Сыр' WHERE `ID` = 3 LIMIT 1;

Если не поставить никаких лимитов LIMIT и условий WHERE, то все записи таблицы будут обновлены без исключений.

**1 Основные типы данных. Неизменяемые типы**

К основным встроенным типам относятся: Логические переменные (**True, False**), Числа(**int, float, complex**), Списки(**list, tuple, range**), Строки(**str**), Множества(**set, frozen set**), Словари(**dict**).

В Python существуют изменяемые и неизменяемые типы.

К неизменяемым типам относятся: целые числа (*int*),  числа с плавающей точкой (*float*), комплексные числа (*complex*), логические переменные (*true, false*), кортежи (*tuple*), строки (*str*) и неизменяемые множества (*frozen set*).

К изменяемым типам относятся: списки (*list*), множества (*set*), словари (*dict*).

**2 Числовые типы в Python. Библиотека random**

**Числовой тип данных в Python** предназначен для хранения числовых значений. Это *неизменяемый тип данных*, что означает, что изменение значения числового типа данных приведет к созданию нового объекта в памяти (и удалению старого)

В **Python** есть четыре вида числового типа данных:

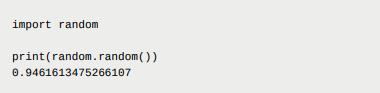
* *int* (целое число)
* *long* (длинное целое число [может быть представлено в восьмеричной или шестнадцатеричной системе исчисления])
* *float* (число с плавающей точкой: -0.2, 0.0, 3.14159265 и т.д.)
* *complex* (комплексное число)

Модуль **random** позволяет генерировать случайные числа. Прежде чем использовать модули, необходимо подключить его с помощью инструкции:

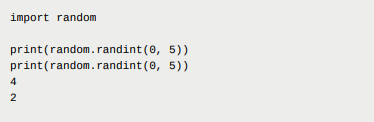


Список **методов модуля random** в Python:

random.random() — возвращает псевдослучайное число от 0.0 до 1.0



random.randint(a,b) — возвращает псевдослучайное целое число в диапазоне от a до b:



Это те что я использовал в ходе выполнения лабораторных работ, их там конечно гораздо больше, но для примера хватит и этих с другими можно ознакомится на сайте metanit.com и pythonworld.ru

**3 Библиотека math. Основные константы и операции**

Встроенный модуль **math** в Python предоставляет набор функций для выполнения математических, тригонометрических и логарифмических операций.

Некоторые из основных **операций** модуля:

* **pow(a, b):** возведение числа a в степень b
* **sqrt(a):** квадратный корень числа a
* **factorial(a):** факториал числа
* **degrees(rad):** перевод из радиан в градусы
* **radians(grad):** перевод из градусов в радианы
* **cos(rad):** косинус угла в радианах
* **sin(rad):** синус угла в радианах
* **tan(rad):** тангенс угла в радианах
* **acos(rad):** арккосинус угла в радианах
* **asin(rad):** арксинус угла в радианах
* **atan(rad):** арктангенс угла в радианах
* **log(a, base):** логарифм числа a по основанию base
* **log10(a):** десятичный логарифм числа a

Основных **константы** модуля:

* **math.pi** - число пи
* **math.e** - натуральный логарифм

**4 Строки в Python. Основные операции. Особенности**

Под **строками в Python** подразумевается набор символов между кавычками. В **Python** можно использовать пары одинарных либо двойных кавычек. Из строк можно взять подстроку используя оператор нарезки ( [ ] и [ : ] ) с индексами от нуля для первого символа строки и до последнего. Так же можно использовать обратную индексацию от -1 для последнего символа до начала.

Оператор плюс ( + ) для строк соединяет две строки в одну, звездочка ( \* ) оператор повторения

Встроенные **функции** строк:

* chr() - преобразует целое число в символ
* ord() - преобразует символ в целое число
* len() - возвращает длину строки
* str() - изменяет тип объекта на string

**5 Списки в Python. Основные операции. Вложение, виды вложений**

**Список** (list) представляет тип данных, который хранит набор или изменяемую последовательность элементов. Для создания списка в квадратных скобках ([]) через запятую перечисляются все его элементы. Например, определим список чисел: numbers = [1, 2, 3, 4, 5]. Список необязательно должен содержать только однотипные объекты. Мы можем поместить в один и тот же список одновременно строки, числа, объекты других типов данных: objects = [1, 2.6, "Hello", True]. Списки кроме стандартных данных типа строк, чисел, также могут содержать другие списки: users = [["Tom", 29], ["Alice", 33], ["Bob", 27]]

Встроенные **функции** списка:

* len(list) - возвращает длину списка
* sorted(list, [key]) - возвращает отсортированный список
* min(list) - возвращает наименьший элемент списка
* max(list) - возвращает наибольший элемент списка

Для управления элементами списки имеют целый ряд **методов**:

* append(item): добавляет элемент item в конец списка
* insert(index, item): добавляет элемент item в список по индексу index
* remove(item): удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError
* clear(): удаление всех элементов из списка
* index(item): возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError
* pop([index]): удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.
* count(item): возвращает количество вхождений элемента item в список
* sort([key]): сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.
* reverse(): расставляет все элементы в списке в обратном порядке

Вложенный список - это список списков. Вложенный список — это всегда один элемент вне зависимости от количества его элементов.

Виды вложений не нашел

**6 Словари в Python. Основные операции. Вложение, виды вложений**

**Словарь** — неупорядоченный набор пар ключ-значение. Они используются, когда нужно сопоставить каждому из ключей значение и иметь возможность быстро получать доступ к значению, зная ключ.

Определение словаря имеет следующий синтаксис:

* dictionary = { ключ1:значение1, ключ2:значение2, ....}

Определим пару словарей:

* users = {1: "Tom", 2: "Bob", 3: "Bill"}
* elements = {"Au": "Золото", "Fe": "Железо", "H": "Водород", "O": "Кислород"

Для управления элементами словари имеют целый ряд методов.

* dict.clear() - очищает словарь.
* dict.copy() - возвращает копию словаря.
* dict.fromkeys(seq[, value]) - создает словарь с ключами из seq и значением value (по умолчанию None).
* dict.get(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).
* dict.items() - возвращает пары (ключ, значение).
* dict.keys() - возвращает ключи в словаре.
* dict.pop(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).
* dict.popitem() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.
* dict.setdefault(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).
* dict.update([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).
* dict.values() - возвращает значения в словаре.

**Вложенные словари** – это словари, содержащие другие словари. Мы можем создать вложенный словарь так же, как мы создаем обычный словарь, используя фигурные скобки.

Видов тоже нет, мне кажется он их по приколу добавил

**7 Кортежи в Python. Основные операции. Особенности**

**Кортеж** (tuple) представляет последовательность элементов, которая во многом похожа на список за тем исключением, что кортеж является неизменяемым типом. Поэтому мы не можем добавлять или удалять элементы в кортеже, изменять его.

Для создания кортежа используются круглые скобки, в которые помещаются его значения, разделенные запятыми. Если вдруг кортеж состоит из одного элемента, то после единственного элемента кортежа необходимо поставить запятую:

user = ("Tom", 23)

user = "Tom", 23

user = ("Tom",)

Обращение к элементам в кортеже происходит так же, как и в списке по индексу.

Преимущества **кортежей**:

* Защита от дурака. То есть кортеж защищен от изменений, как намеренных, так и случайных.
* Меньший размер

Основные операции **кортежей**:

* Операция присваивания кортежа. =
* Операция tuple() – создание кортежа из итерированного объекта
* Операция T[i]. Получить элемент по индексу
* Операция T[i][j]. Получить доступ к составному элементу в кортеже
* Операция T[i:j]. Взятие среза в кортеже
* Сложение. +
* Повторение. \*
* Операция in. Проверка вхождения элемента в кортеж

**Методы** работы с кортежами:

* index() - поиск позиции элемента в кортеже
* count() - количество вхождений элемента в кортеж

**8 Множества в Python. Основные операции. Особенности**

**Множества** — это неупорядоченный список, то есть элементы внутри него располагаются в случайном порядке. Как списки и словари, множество относится к изменяемым типам данных. Но при этом невозможен поиск по индексу, так как элементы в случайном порядке. Зато множества дают высокую скорость поиска по ним. Например, это удобно, когда порядок вам не важен, а нужно просто быстро искать нужные элементы.

Основные **операции**:

* Создание множеств
* Доступ к элементам множеств
* Добавление элементов во множество
* Удаление элементов из множеств
* Объединение множеств
* Пересечение множеств
* Разница множеств
* Сравнение множеств

**Методы:**

* copy() - этот метод возвращает копию множества
* isdisjoint() - этот метод проверяет, является ли множество пересечением или нет
* len() - этот метод возвращает длину множества

**9 Файлы в Python. Основные операции. Unicode и двоичные файлы**

**Файл** — это всего лишь набор данных, сохраненный в виде последовательности битов на компьютере. Информация хранится в куче данных (структура данных) и имеет название «имя файла» (filename).

В Python существует два типа файлов:

* Текстовые
* Бинарные

**Текстовые файлы**

Это файлы с человекочитаемым содержимым. В них хранятся последовательности символов, которые понимает человек. Блокнот и другие стандартные редакторы умеют читать и редактировать этот тип файлов.

Текст может храниться в двух форматах: ( .txt ) — простой текст и ( .rtf ) — «формат обогащенного текста»

**Бинарные файлы**

В бинарных файлах данные отображаются в закодированной форме (с использованием только нулей (0) и единиц (1) вместо простых символов). В большинстве случаев это просто последовательности битов.

Они хранятся в формате  .bin

Открытие файла производится с помощью **метода open()**

f = open(file\_name, access\_mode)

Где, file\_name  = имя открываемого файла, а access\_mode  = режим открытия файла. Он может быть: для чтения, записи и т. д. По умолчанию используется режим чтения ( r ). Например: (r-только для чтения, w-только для записи. Создаст новый файл, если не найдет с указанным именем, и так далее)

Закрытие файла производится с помощью **метода close()**

Основные **функции**:

* read() - функция чтения
* readline() - функция построчного чтения
* write() - функция записи
* rename() - функция переименовывания

Основные **методы**:

* file.write(str) - добавляет строку str в файл
* file.next() - возвращает следующую строку файла
* file.read(n) - чтение первых n символов файла

Остальное описано выше

10 Типизация в Python. Понятие динамической типизации

**Типизация** — это то, как язык распознаёт типы переменных. Типизация определяет, нужно ли вам писать тип, или язык «поймёт» его сам, и насколько свободно можно с типами работать

**Динами́ческая типиза́ция** — приём, используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов.

**11 Операторы в Python. Перечислить и описать**

**Оператор в Python** — это символ, который выполняет операцию над одним или несколькими операндами. Операндом выступает переменная или значение, над которыми проводится операция.

Операторы Python бывают 7 типов:

* **Арифметические операторы**

1. Сложение (+)

2. Вычитание (-)

3. Умножение (\*)

4. Деление (/)

5. Возведение в степень (\*\*)

6. Деление без остатка (//)

7. Деление по модулю (остаток от деления) (%)

* **Операторы сравнения**

1. Меньше ()

2. Больше (>)

3. Меньше или равно (<=)

4. Больше или равно (>=)

5. Равно (==)

6. Не равно (!=)

* **Операторы присваивания**

1. Присваивание (=)

2. Сложение и присваивание (+=)

3. Вычитание и присваивание (-=)

4. Деление и присваивание (/=)

5. Умножение и присваивание (\*=)

6. Деление по модулю и присваивание (%=)

7. Возведение в степень и присваивание (\*\*=)

8. Деление с остатком и присваивание (//=)

* **Логические операторы**

1. И (and)

2. Или (or)

3. Не (not)

* **Операторы принадлежности**

1. В (in)

2. Нет в (not in)

* **Операторы тождественности**

1. Это (is)

2. Это не (is not)

* **Битовые операторы**

1. Бинарное И (&)

2. Бинарное ИЛИ (|)

3. Бинарное ИЛИ НЕТ (^)

4. Инвертирующий оператор (~)

5. Бинарный сдвиг влево (<>)

**12 Функции в Python. Назначение и применение.**

**Функции** представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы. Формальное определение функции:

def имя\_функции ([параметры]):

инструкции

Для вызова функции указывается имя функции, после которого в скобках идет передача значений для всех ее параметров. Например:

def say\_hello(name):

print("Hello,",name)

say\_hello("Tom")

Некоторые параметры функции мы можем сделать необязательными, указав для них **значения по умолчанию** при определении функции. Например:

def say\_hello(name="Tom"):

print("Hello,", name)

say\_hello()

say\_hello("Bob")

При передаче значений функция сопоставляет их с параметрами в том порядке, в котором они передаются. Использование **именованных параметров** позволяет переопределить порядок передачи:

def display\_info(name, age):

print("Name:", name, "\t", "Age:", age)

display\_info(age=22, name="Tom")

Функция может **возвращать результат**. Для этого в функции используется оператор return, после которого указывается возвращаемое значение. Поскольку функция возвращает значение, то мы можем присвоить это значение какой-либо переменной и затем использовать ее.

def exchange(usd\_rate, money):

result = round(money/usd\_rate, 2)

return result result1 = exchange(60, 30000)

print(result1)

В Python функция может возвращать сразу несколько значений:

def create\_default\_user():

name = "Tom"

age = 33

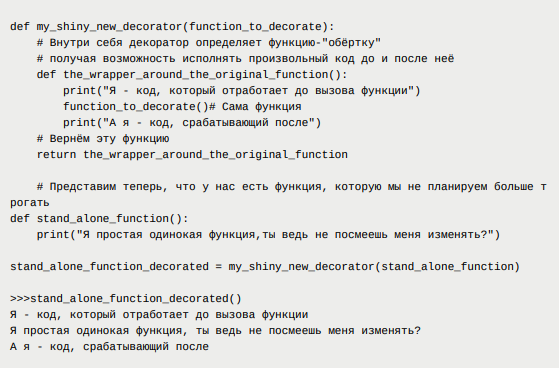
return name, age

**13 Декораторы в Python. Применение и особенности**

**Декоратор** – это функция, которая принимает другую функцию в качестве аргумента. **Декоратор** модифицирует или улучшает принятую функцию и выдает измененную. Это значит, что когда вы вызываете декорированную функцию, вы получите функцию, которая может иметь небольшие отличия, в виде дополнительных функций, совмещенных с базовым определением.

**Особенности:**

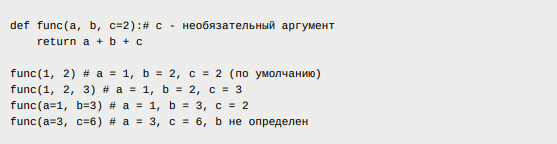
* Декораторы несколько замедляют вызов функции, не забывайте об этом.
* Вы не можете "раздекорировать" функцию. Безусловно, существуют трюки, позволяющие создать декоратор, который можно отсоединить от функции, но это плохая практика. Правильнее будет запомнить, что если функция декорирована — это не отменить.
* Декораторы оборачивают функции, что может затруднить отладку.



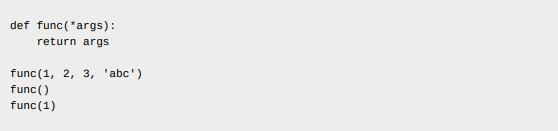
**14 Аргументы функций. Варианты передачи. Аннотирование типов**

Функция может принимать произвольное количество аргументов или не принимать их вовсе.

Варианты передачи:



Функция также может принимать переменное количество позиционных аргументов, тогда перед именем ставится \*

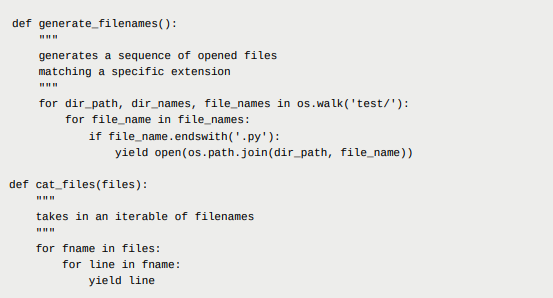


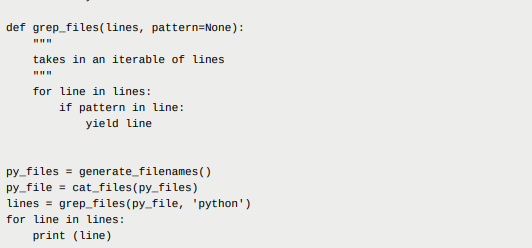
Как видно из примера, args - это кортеж из всех переданных аргументов функции, и с переменной можно работать также, как и с кортежем.

**15 Генераторы в Python. Применение и особенности**

**Генераторы** — это функции, которые можно приостанавливать и возобновлять во время их выполнения, при этом они возвращают объект, который можно итерировать.

Генераторы идеально подходят для чтения большого количества больших файлов, поскольку они выдают данные по одному фрагменту за раз, независимо от размера входного потока. Они также могут привести к более чистому коду путем разделения процесса итерации на более мелкие компоненты.





**16 Классы в Python. Стандартные методы классов.**

Класс — это пользовательский тип, состоящий из методов и атрибутов. Класс является шаблоном или формальным описанием объекта, а объект представляет экземпляр этого класса, его реальное воплощение.

Создание класса:

class название\_класса:

методы\_класса

Чтобы создать экземпляры классов, нужно вызвать класс с использованием его имени и передать аргументы, которые принимает метод **init**.

Для создания объекта (экземпляра) класса используется следующий синтаксис:

название\_объекта = название\_класса([параметры])

Каждая такая запись генерирует свой объект класса. Отличие от C++ в том, что в C++ описание класса — это лишь объявление, а в питоне — это создание объекта.

Внутри классов вы можете определить функции или методы, которые являются частью этого класса.

В классе метод определяется с помощью оператора def. Первый параметр каждого метода всегда ссылается на сам объект. В соответствии с общепринятым соглашением для этого параметра применяется имя self. Все операции, затрагивающие атрибуты объекта, должны явно ссылаться на переменную self.

Стандартные **методы**:

* \_\_init\_\_(self, [...) инициализатор класса
* \_\_del\_\_(self) деструктор
* \_\_cmp\_\_(self, other) базовый метод сравнения
* \_\_add\_\_(self, other) сложение

Про методы вообще информации не много

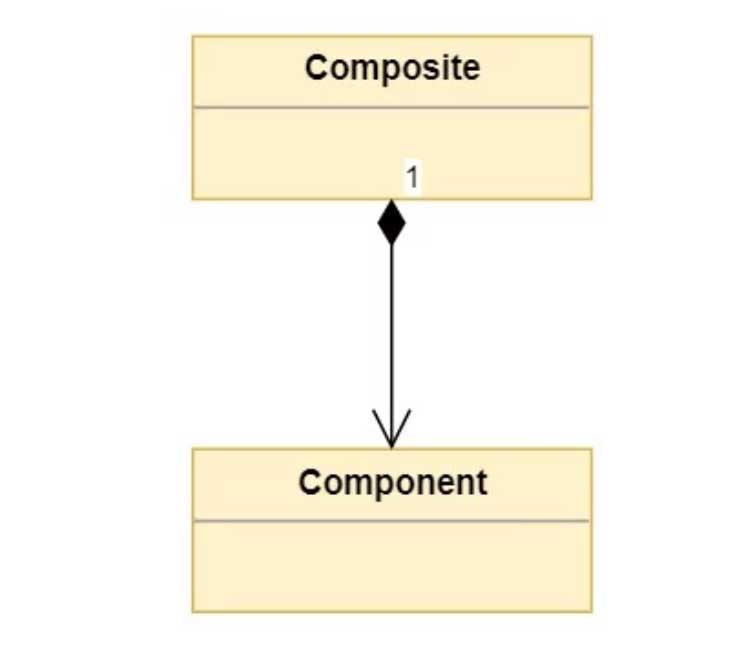
**17 Наследование, композиция, агрегация в Python**

Наследование: можно создавать специализированные классы на основе базовых. Это позволяет нам избегать написания повторного кода.

Ключевыми понятиями наследования являются подкласс и суперкласс. Производный (дочерний) наследует от базового (родительского) все публичные атрибуты и методы.

Класс называется абстрактным, если он предназначен только для наследования. Экземпляры абстрактного класса обычно не имеют большого смысла.

Композиция — это концепция, которая моделирует отношения. Она позволяет создавать сложные типы, комбинируя объекты других типов. Это означает, что класс Composite может содержать объект другого класса Component.



Отдельный класс задач, который не может обойтись без циклов, называется агрегированием данных. К таким задачам относятся поиск максимального, минимального, суммы, среднего арифметического и т.п. Их главная особенность в том, что результат зависит от всего набора данных. Для расчёта суммы нужно сложить все числа, для вычисления максимального нужно сравнить все числа.

sum\_numbers\_from\_range(5, 7) # 5 + 6 + 7 = 18

sum\_numbers\_from\_range(1, 2) # 1 + 2 = 3

# [1, 1] диапазон с одинаковым началом и концом – тоже диапазон

# он в себя включает ровно одно число – саму границу диапазона

sum\_numbers\_from\_range(1, 1) # 1

sum\_numbers\_from\_range(100, 100) # 100

1**8 Numpy. Назначение и основные возможности**

**NumPy** (сокращенно от *Numerical Python*)— библиотека с [открытым](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [исходным кодом](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) для [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [Python](https://ru.wikipedia.org/wiki/Python).

Возможности:

* поддержка многомерных [массивов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2) (включая [матрицы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)));
* поддержка [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) математических функций, предназначенных для работы с многомерными массивами.

Назначение:

Математические алгоритмы, реализованные на [интерпретируемых языках](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (например, Python), часто работают гораздо медленнее тех же алгоритмов, реализованных на [компилируемых языках](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (например, [Фортран](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D1%82%D1%80%D0%B0%D0%BD), [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Java](https://ru.wikipedia.org/wiki/Java)). Библиотека NumPy предоставляет реализации [вычислительных](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) [алгоритмов](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) (в виде функций и операторов), оптимизированные для работы с многомерными массивами. В результате любой алгоритм, который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием NumPy, работает так же быстро, как эквивалентный код, выполняемый в [MATLAB](https://ru.wikipedia.org/wiki/MATLAB).

**19 Типы данных в Numpy**

В NumPy доступны следующие типы данных:

bool\_ Логический тип (истина или ложь), хранящийся в виде байта.

int\_ Целочисленный тип установленный по умолчанию (такой же, как C long, как правило это либо int64 либо int32).

intc Идентичен C int (int32 или int64).

intp Целочисленный тип, используемый для индексирования (такой же, как C ssize\_t, как правило это либо int64 либо int32).

int8 Целые числа в диапазоне от -128 по 127 (числа размером 1 байт).

int16 Целые числа в диапазоне от -32768 по 32767, (числа размером 2 байта).

int32 Целые числа в диапазоне от -2147483648 по 2147483647, (числа размером 4 байта).

int64 Целые числа в диапазоне от -9223372036854775808 по 9223372036854775807, (числа размером 8 байт).

uint8 Целые числа в диапазоне от 0 по 255 (числа размером 1 байт).

uint16 Целые числа в диапазоне от 0 по 65535 (числа размером 2 байта).

uint32 Целые числа в диапазоне от 0 по 4294967295 (числа размером 4 байта).

uint64 Целые числа в диапазоне от 0 по 18446744073709551615 (числа размером 8 байт).

float\_ То же самое что и float64.

float16 Вещественные числа половинной точности: 1 бит знака, 5 бит экспоненты, 10 бит мантисы (числа размером 2 байта).

float32 Вещественные числа одинарной точности: 1 бит знака, 8 бит экспоненты, 23 бита мантисы (числа размером 4 байта).

float64 Вещественные числа двойной точности: 1 бит знака, 11 бит экспоненты, 52 бита мантисы (числа размером 8 байт).

complex\_ То же самое что и complex128.

complex64 Комплексные числа в которых действительная и мнимая части представлены двумя вещественными числами типа float32.

complex128 Комплексные числа в которых действительная и мнимая части представлены двумя вещественными числами типа float64.

В NumPy существует 5 базовых числовых типов: булевы числа (тип bool: 0 - ложь и 1 - истина), целые числа (тип int), беззнаковые целые числа (тип uint), вещественные числа (числа с плавающей запятой, тип float) и комплексные числа (тип complex). У некоторых после указания типа следует количество бит необходимое для хранения такого числа в памяти (16, 32, 64 или 128), но некоторые, такие как int\_ или intp зависят от используемой платформы (32 или 64-разрядные машины).

**20 Основные операции над массивами и скалярами в Numpy**

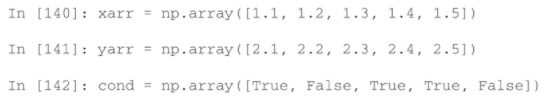
NumPy включает в себя набор пакетов для решения специализированных задач, например:

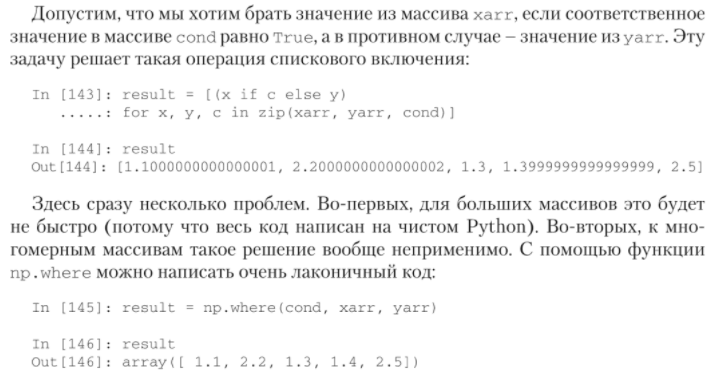
* numpy.linalg — реализует операции линейной алгебры (простое умножение векторов и матриц есть в базовом варианте);
* numpy.random — реализует функции для работы со случайными величинами;
* numpy.fft — реализует прямое и обратное преобразование Фурье.

Основные операции:

copyto(dst, src[, casting, where]) Копирует данные из одного массива в другой с выполнением транслирования если это необходимо.  
**21 Запись логических условий в виде операций с массивами**







**22 Индексирование и вырезание. Булево индексирование**

Механизм индексирования участвует в выполнении запросов; он вызывается в соответствии с планом, построенным на этапе оптимизации. Оптимизатор, перебирая и оценивая различные пути выполнения запроса, должен понимать возможности всех методов доступа, которые потенциально можно применить. Механизм индексирования позволяет PostgreSQL одинаково работать с самыми разными методами доступа, учитывая их возможности.

Основные способы сканирования:

1) Индексное сканирование

2) Сканирование по битовой карте

Индексное сканирование хорошо работает, когда речь идет всего о нескольких значениях. Однако при увеличении выборки возрастают шансы, что придется возвращаться к одной и той же табличной странице несколько раз. Поэтому в таком случае оптимизатор переключается на сканирование по битовой карте.

3) Последовательное сканирование

4) Покрывающие индексы

Как правило, основная задача метода доступа — вернуть идентификаторы подходящих строк таблицы, чтобы механизм индексирования мог прочитать из них необходимые данные. Но что, если индекс уже содержит все необходимые для запроса данные? Такой индекс называется покрывающим (covering), и в этом случае оптимизатор может применить исключительно индексное сканирование.

5) Null

Неопределенные значения играют важную роль в реляционных базах данных как удобный способ представления того факта, что значение не существует или не известно.

6) Индексы по нескольким полям

Как правило, если на первое поле не наложено условие, индекс использоваться не будет. Но в некоторых случаях оптимизатор может счесть, что это все-таки выгоднее последовательного сканирования.

7) Индексы по выражениям

Функциональный индекс создается не по полю таблицы, а по произвольному выражению; оптимизатор будет принимать во внимание такой индекс для условий вида «индексированное-выражение оператор выражение».

8) Частичные индексы

9) Сортировка

10) Параллельное построение

Обычно построение индекса требует установки блокировки типа SHARE на таблицу. Такая блокировка позволяет читать данные из таблицы, но запрещает любые изменения, пока строится индекс.

**23 Срезы в Numpy**

Срезы позволяют извлекать части массива для создания новых массивов. Когда вы используете срезы для списков Python, результирующие массивы — это копии, но в NumPy они являются представлениями одного и того же лежащего в основе буфера.

В зависимости от части массива, которую необходимо извлечь, нужно использовать синтаксис среза; это последовательность числовых значений, разделенная двоеточием (:) в квадратных скобках.

Чтобы получить, например, часть массива от второго до шестого элемента, необходимо ввести индекс первого элемента — 1 и индекса последнего — 5, разделив их :.

>>> a = np.arange(10, 16)

>>> a

array([10, 11, 12, 13, 14, 15])

>>> a[1:5]

array([11, 12, 13, 14])

Если нужно извлечь элемент из предыдущего отрезка и пропустить один или несколько элементов, можно использовать третье число, которое представляет собой интервал последовательности. Например, со значением 2 результат будет такой:

>>> a[1:5:2]

array([11, 13])

Чтобы лучше понять синтаксис среза, необходимо рассматривать и случаи, когда явные числовые значения не используются. Если не ввести первое число, NumPy неявно интерпретирует его как 0 (то есть, первый элемент массива). Если пропустить второй — он будет заменен на максимальный индекс, а если последний — представлен как 1. То есть, все элементы будут перебираться без интервалов.

>>> a[::2]

array([10, 12, 14])

>>> a[:5:2]

array([10, 12, 14])

>>> a[:5:]

array([10, 11, 12, 13, 14]

**24 Векторизованные операции в Numpy**

Функция Numpy vectorize принимает функцию python (py func) и возвращает векторизованную версию функции.

Векторизованная версия функции принимает последовательность объектов или массивов NumPy в качестве входных данных и вычисляет функцию Python над каждым элементом входной последовательности. Векторизация Numpy по существу функционирует подобно python map (), но с дополнительной функциональностью – механизмом вещания NumPy.

Необходимые параметры:

py func : Функция, которую мы хотим применить к последовательности объектов

Дополнительные параметры:

типы : Выходные типы функции могут быть указаны в виде строки или списка типов данных. Если типы не указаны, а кэш установлен в True , тип вывода определяется вызовом первого элемента ввода.

doc : Для указания строки документа созданного. Если не указано, будет использоваться исходная строка документа функции( если таковая имеется).

кэш : Если True , то кэшируйте первый вызов функции, определяющий количество выходов, если o типов не предусмотрено.

**25 Pandas. Назначение и основные возможности**

Рandas это высокоуровневая Python библиотека для анализа данных. Построена она поверх более низкоуровневой библиотеки NumPy (написана на Си), что является большим плюсом в производительности. В экосистеме Python, pandas является наиболее продвинутой и быстроразвивающейся библиотекой для обработки и анализа данных.

Чтобы эффективно работать с pandas, необходимо освоить самые главные структуры данных библиотеки: DataFrame и Series.

Основные возможности библиотеки:

* Объект *DataFrame* для манипулирования индексированными массивами двумерных данных[[4]](https://ru.wikipedia.org/wiki/Pandas#cite_note-4)
* Инструменты для обмена данными между структурами в памяти и файлами различных форматов
* Встроенные средства совмещения данных и способы обработки отсутствующей информации
* Переформатирование наборов данных, в том числе создание сводных таблиц
* Срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных
* Вставка и удаление столбцов данных
* Возможности группировки позволяют выполнять трёхэтапные операции типа «разделение, изменение, объединение» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *split-apply-combine*).
* Слияние и объединение наборов данных
* Иерархическое индексирование позволяет работать с данными высокой размерности в структурах меньшей размерности
* Работа с временными рядами: формирование временных периодов и изменение интервалов и так далее

**26 Типы данных Dataframe и Series в Pandas**

Ядром pandas являются две структуры данных, в которых происходят все операции:

Series и Dataframes

Series — это структура, используемая для работы с последовательностью одномерных данных, а Dataframe — более сложная и подходит для нескольких измерений.

Пусть они и не являются универсальными для решения всех проблем, предоставляют отличный инструмент для большинства приложений. При этом их легко использовать, а множество более сложных структур можно упросить до одной из этих двух.

Однако особенности этих структур основаны на одной черте — интеграции в их структуру объектов index и labels (метки). С их помощью структурами становится очень легко манипулировать.

Series — это объект библиотеки pandas, спроектированный для представления одномерных структур данных, похожих на массивы, но с дополнительными возможностями. Его структура проста, ведь он состоит из двух связанных между собой массивов. Основной содержит данные (данные любого типа NumPy), а в дополнительном, index, хранятся метки.

Dataframe — это табличная структура данных, напоминающая таблицы из Microsoft Excel. Ее главная задача — позволить использовать многомерные Series. Dataframe состоит из упорядоченной коллекции колонок, каждая из которых содержит значение разных типов (числовое, строковое, булевое и так далее).

В отличие от Series у которого есть массив индексов с метками, ассоциированных с каждым из элементов, Dataframe имеет сразу два таких. Первый ассоциирован со строками (рядами) и напоминает таковой из Series. Каждая метка ассоциирована со всеми значениями в ряду. Второй содержит метки для каждой из колонок.

**27 Базовая функциональность Pandas. Доступ по индексу, выборка и фильтрация**

Базовые функции DataFrame

T Переставляет строки и столбцы.

axes Возвращает список с метками оси строк и метками оси столбцов в качестве единственных членов.

dtypes Возвращает типы в этом объекте.

empty Истинно, если NDFrame полностью пуст [нет элементов]; если какая-либо из осей имеет длину 0.

ndim Количество осей / размеров массива.

shape Возвращает кортеж, представляющий размерность DataFrame.

size Количество элементов в NDFrame.

values Numpy представление NDFrame.

head() Возвращает первые n строк.

tail() Возвращает последние n строк.

Выбирать отдельные элементы можно по принципу обычных массивов numpy, используя для этого индекс. Или же можно выбрать метку, соответствующую положению индекса. Таким же образом можно выбрать несколько элементов массива numpy с помощью следующей команды:

>>> s[0:2]

a 12

b -4

dtype: int64

Благодаря тому что основной библиотекой в pandas является NumPy, многие операции, применяемые к массивам NumPy, могут быть использованы и в случае с Series. Одна из таких — фильтрация значений в структуре данных с помощью условий.

Например, если нужно узнать, какие элементы в Series больше 8, то можно написать следующее:

>>> s[s > 8]

a 12

d 9

dtype: int64  
**28 Обработка отсутствующих данных в Pandas**

Pandas - это библиотека Python для анализа и обработки данных. Почти все операции в pandas вращаются вокруг DataFrame s, абстрактной структуры данных, специально созданной для обработки метрической тонны данных.

Первым шагом к правильному анализу данных является очистка и организация данных.

Если мы хотим отобразить список значений, которые должны рассматриваться как отсутствующие значения во всех столбцах, мы можем передать список кандидатов, которые мы хотим глобально рассматривать как отсутствующие значения, в параметр na\_values

После того, как мы определили все недостающие значения в DataFrame и правильно аннотировали их, есть несколько способов обработки недостающих данных.

Один из подходов - удалить все строки, содержащие пропущенные значения. Это легко сделать с помощью специально предназначенной для этого функции dropna()

inplace = True вносит все изменения в существующий DataFrame не возвращая новый. Без него вам пришлось бы переназначить DataFrame самому себе.

axis указывает, работаете ли вы со строками или столбцами: 0 - строки, а 1 - столбцы.

Вы можете контролировать , хотите ли вы удалить строки , содержащие по меньшей мере , 1 NaN или все NaN значения, установив , how параметр в dropna методе.

Заполнение недостающих значений:

1) Можно использовать функции .median() , .mode() и .mean() для столбца и указать их в качестве значения заполнения.

2) Вы также можете решить заполнить значения с пометкой NA постоянным значением. Например, вы можете ввести специальную строку или числовое значение.

3) Прямое заполнение отсутствующих значений фрейма данных.

df['Salary'].fillna(method='ffill', inplace=**True**)

4) Обратное заполнение отсутствующих значений фрейма данных

df['Salary'].fillna(method='bfill', inplace=**True**)

5) Этот метод использует математическую интерполяцию, чтобы определить, какое значение было бы на месте отсутствующего значения:

df['Salary'].interpolate(method='polynomial')

Очистка и предварительная обработка данных - очень важная часть каждого анализа данных и каждого проекта в области науки о данных.

**!!!!!29 Статистика в Pandas. Уникальные значения, счётчики значений**

Большое количество методов совместно вычисляет описательную статистику и другие связанные операции над DataFrame. Большинство из них являются агрегатами, такими как **sum (), mean (),** но некоторые из них, например **sumsum ()** , создают объект одинакового размера. Вообще говоря, эти методы принимают аргумент **оси** , как и *ndarray. {Sum, std, …},* но ось может быть указана по имени или целому числу.  
**30 Базовая функциональность Pandas.**

Объекты pandas ( Index , Series , DataFrame ) можно рассматривать как контейнеры для массивов, которые содержат фактические данные и выполняют фактические вычисления. Для многих типов базовым массивом является numpy.ndarray . Однако pandas и сторонние библиотеки могут расширять систему типов NumPy, добавляя поддержку настраиваемых массивов.

Основные функции серии

**axes** Возвращает список меток оси строк

dtype Возвращает dtype объекта.

empty Возвращает True, если серия пуста.

ndim Возвращает количество измерений базовых данных по определению 1.

size Возвращает количество элементов в базовых данных.

values Возвращает серию как ndarray.

**head()** Возвращает первые n строк.

tail() Возвращает последние n строк.

Базовые функции DataFrame

T Переставляет строки и столбцы.

axes Возвращает список с метками оси строк и метками оси столбцов в качестве единственных членов.

dtypes Возвращает типы в этом объекте.

empty Истинно, если NDFrame полностью пуст [нет элементов]; если какая-либо из осей имеет длину 0.

ndim Количество осей / размеров массива.

shape Возвращает кортеж, представляющий размерность DataFrame.

size Количество элементов в NDFrame.

values Numpy представление NDFrame.

head() Возвращает первые n строк.

tail() Возвращает последние n строк.

**31 Переиндексация и удаление элементов в Pandas**

Переиндексирование df.reindex()

После объявления в структуре данных объект Index нельзя менять. Но с помощью операции переиндексирования это можно решить.

>>> ser = pd.Series([2,5,7,4], index=['one','two','three','four']) >>> ser

one 2

two 5

three 7

four 4

dtype: int64

Для того чтобы провести переиндексирование объекта Series библиотека pandas предоставляет функцию reindex(). Она создает новый объект Series со значениями из другого Series, которые теперь переставлены в соответствии с новой последовательностью меток.

При операции переиндексирования можно поменять порядок индексов, удалить некоторые из них или добавить новые. Если метка новая, pandas добавит NaN на место соответствующего значения.

>>> ser.reindex(['three','four','five','one'])

three 7.0

four 4.0

five NaN

one 2.0

dtype: float64

Как видно по выводу, порядок меток можно поменять полностью. Значение, которое раньше соответствовало метке two, удалено, зато есть новое с меткой five.

Удаление

Удалить строку или колонку не составит труда, потому что метки используются для обозначения индексов и названий колонок.

В этом случае pandas предоставляет специальную функцию для этой операции, которая называется drop(). Метод возвращает новый объект без элементов, которые необходимо было удалить.

Например, возьмем в качестве примера случай, где из объекта нужно удалить один элемент. Для этого определим базовый объект Series из четырех элементов с 4 отдельными метками.

>>> ser = pd.Series(np.arange(4.), index=['red','blue','yellow','white'])

>>> ser

red 0.0

blue 1.0

yellow 2.0

white 3.0

dtype: float64

Теперь, предположим, необходимо удалить объект с меткой yellow. Для этого нужно всего лишь указать ее в качестве аргумента функции drop().

>>> ser.drop('yellow')

red 0.0

blue 1.0

white 3.0

dtype: float64

Для удаления большего количества элементов, передайте массив с соответствующими индексами.