**Вопрос 1. Теоретические основы веб программирования**

1. Системы контроля версий. Цели использования. Особенности работы. Локальный и глобальный репозитории на примере Git.

Система управления версиями (также используется определение «система контроля версий», от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое. Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

Разделяют хранилища на: локальные, удалённые и распределённые

У Git есть три основных состояния, в которых могут находиться ваши файлы: изменён (modified), индексирован (staged) и зафиксирован (committed):

К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Индексированный — это изменённый файл в его текущей версии, отмеченный для включения в следующий коммит.

Зафиксированный значит, что файл уже сохранён в вашей локальной базе.

Мы подошли к трём основным секциям проекта Git: рабочая копия (working tree), область индексирования (staging area) и каталог Git (Git directory).

Рабочая копия является снимком одной версии проекта. Эти файлы извлекаются из сжатой базы данных в каталоге Git и помещаются на диск, для того чтобы их можно было использовать или редактировать.

Область индексирования — это файл, обычно находящийся в каталоге Git, в нём содержится информация о том, что попадёт в следующий коммит. Её техническое название на языке Git — «индекс», но фраза «область индексирования» также работает.

Каталог Git — это то место, где Git хранит метаданные и базу объектов вашего проекта. Это самая важная часть Git и это та часть, которая копируется при клонировании репозитория с другого компьютера.

Базовый подход в работе с Git выглядит так:

1. Изменяете файлы вашей рабочей копии.

2. Выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки только этих изменений в индекс.

3. Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в ваш каталог Git.

Удалённый репозиторий:

Создаётся на сайте гитхаба. Далее вам нужно зайти в Гитбаш в вашей папке и вбить выданную строку

/чаще всего советуют переименовать ветку в main/

git remote add origin https://github.com/SergeevaP/testrepo.git

git branch -M main

git push -u origin main

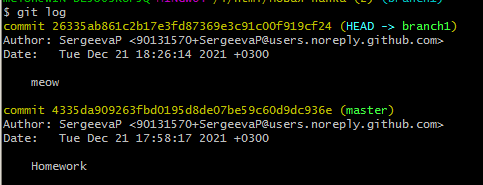
\потом загружаете коммиты с помощью git push. Иногда гит просит ввести строку

git push --set-upstream origin master

\

1. Системы контроля версий. “Перемещение” между состояниями. Сброс изменений. На примере Git.

Чтобы перемещаться по истории коммитов, нам понадобятся несколько команд.

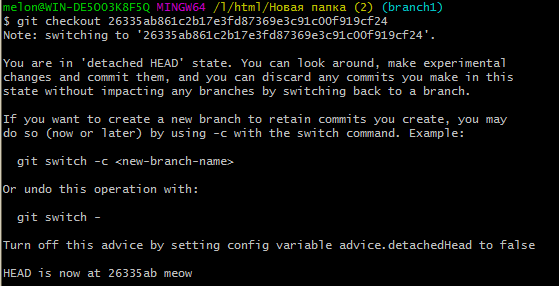


Git log

Показывает список всех коммитов, веток и тд

Аргументы:

--oneline – показывает всё в одной строке без лишней инфы



Git checkout

Позволяет перемещаться по коммитам или веткам

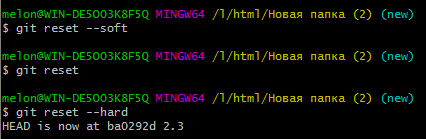
Для перемещения по коммитам используется sha(хэш. Вот тот длинны номер после слова commit), для перемещения по веткам название ветки



Git switch

Используется для создания новой ветки(git switch –c '<имя\_новой\_ветки>') или сброса переходов по коммитам (возвращается к последнему коммиту ака голове)

(прошу заметить что если сделать гит чекаут в последний коммит, то на тебя потом быканут(по стрелочке видно, т.е. если стрелочка есть, то всё ок, а то иначе быканут))



Git reset (по умолчанию --mixed)

Аргументы:

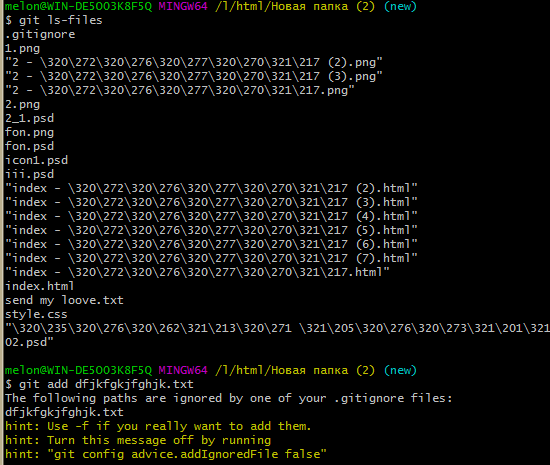
-- mixed – сбрасывает индексирование файлов (ака если ты их добавишь, то их удалят из индекса)

--soft –происходит сброс дерева коммитов, т.е изменения в файлах остаются, но удаляются коммиты.

--hard – полностью сбрасывает репозиторий до состояния определённого коммита, удаляет файлы

.gitkeep и .gitignore

Файлы, находящиеся в гитигнор будут полностью игнорироваться гитом, а когда вы попробуете их добавить – выдаст предупреждение



Создать .gitignore можно с помощью команды touch .gitignore

Пример использования

a.txt – конкретный файл

\*.txt – все файлы такого типа

И др

1. Системы контроля версий. Создание и использование ветвей разработки на примере Git.

Делать git reset очень не по-пацански, поэтому были придуманы ветви.

Ветвь – это история изменений, т.е. история коммитов. Ветви используются для сохранения изменений, которые, возможно, надо будет отменить из-за их неверности. Откатывание прогресса, то есть hard reset является не лучшим решением, так как могут возникнуть проблемы с удалённым репозиторием, поэтому следует использовать ветви.

Создаются они с помощью команды

git branch <имяветки>

либо с помощью

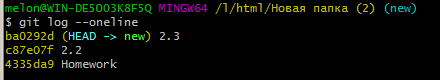
git checkout –b <branchname>

которая сразу переключает в новую ветку

Посмотреть все ветки можно с помощью git branch

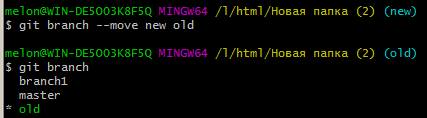


По побочным веткам тоже можно перемещаться, как по основной. (при использовании аргумента –oneline будут показаны коммиты только этой ветки)



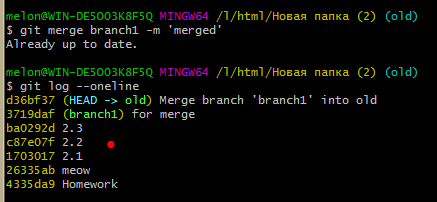
Переменовать ветку можно с помощью

git branch --move bad-branch-name corrected-branch-name

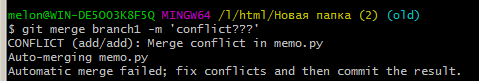


(но это исключительно локальное переменование, чтобы ветка изменила название в удалённом репозитории воспользуйтесь git push --set-upstream origin corrected-branch-name)

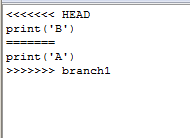
Локальные ветки можно сливать(merge) без пулл-реквеста. Можно слить ветки в мастера или друг в друга, но нельзя слить мастера в другую ветку(просто не даст)

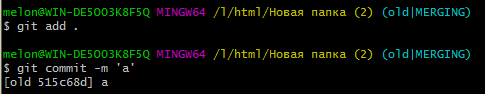


(у меня написано так, так как я уже сливала ветки. А так выведет кучу фигни, типа какой файл куда попал)



При конфликте надо исправлять файлики ВРУЧНУю... И потом уже сливать.

а ещё оно вот так в файле поругается и будет висеть в режиме соединения. После команды коммит сольёт ветки

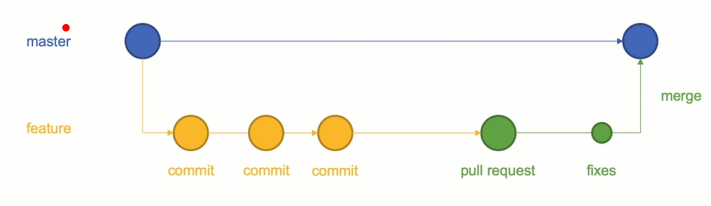


В удалённом репозитории ветви сливают с помощью пуллреквестов, которые требуют подтверждения со стороны(а в локальном вы себе хозяин барин)

Ветку можно удалить



1. Системы контроля версий. Pull Request, Rebase, cherry pick в Git. Примеры использования.

Pull Request - это та же операция слияние (merge), но путем создания запроса на добавления изменений в ветку 

Разработчик создал ветку feature, где создал некие изменения. Прежде чем слить их с веткой мастер он делает запрос, в запросе происходит обсуждение изменений, после чего происходит корректировка, после которой делается слияние веток. Зачем?

Обсуждение измений в команде

Выполнение code-review

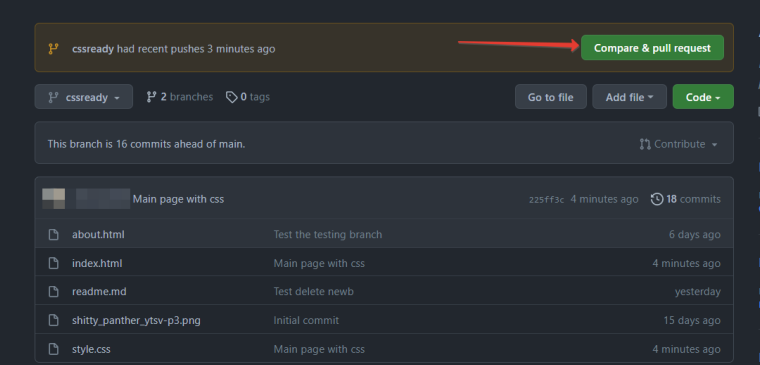
Автоматическое тестирование

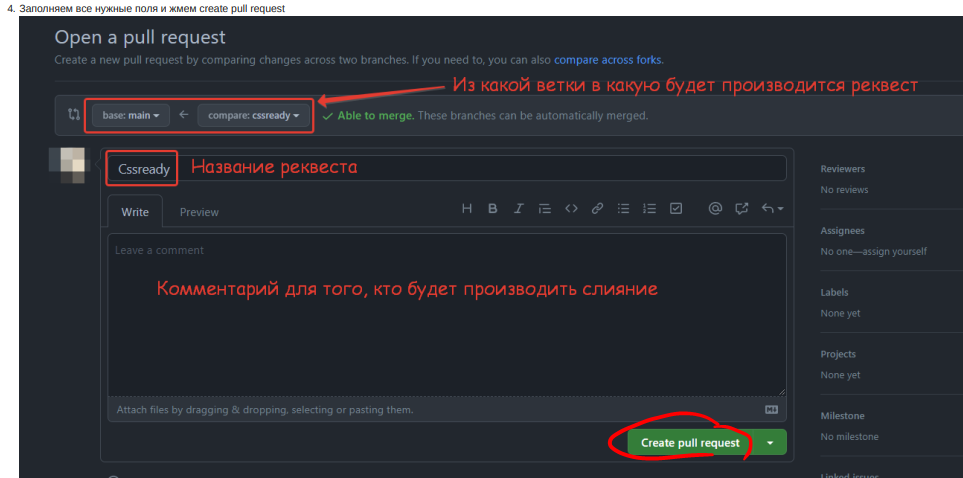
Как создать pull request:

1. Создаем ветку и сохраняем в ней изменения

2. Отправляем изменения в удаленный репозиторий

3. Заходим в удаленный репозиторий и выбираем кнопку compare & pull request

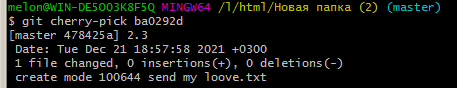




Внесение исправлений:

git cherry-pick

Команда git cherry-pick берёт изменения, вносимые одним коммитом, и пытается повторно применить их в виде нового коммита в текущей ветке. Эта возможность полезна в ситуации, когда нужно забрать парочку коммитов из другой ветки, а не сливать ветку целиком со всеми внесенными в нее изменениями.



Будет сформирован новый коммит и будут накачены изменения из указанного коммита

git rebase

git rebase — это «автоматизированный» cherry-pick . Он выполняет ту же работу, но для цепочки коммитов, тем самым как бы перенося ветку на новое место.

Допустим, мы создали ветку Experiment и сделали в нее один коммит. Как мы выяснили ранее, простейший способ выполнить слияние двух веток — это команда merge. Она осуществляет трёхстороннее слияние между двумя последними снимками сливаемых веток (C3 и C4) и самого недавнего общего для этих веток родительского снимка (C2), создавая новый снимок (и коммит). Тем не менее есть и другой способ: вы можете взять те изменения, что были представлены в C4, и применить их поверх C3. В Git это называется *перебазированием*. С помощью команды rebase вы можете взять все коммиты из одной ветки и в том же порядке применить их к другой ветке. В данном примере переключимся на ветку experiment и перебазируем её относительно ветки master следующим образом:

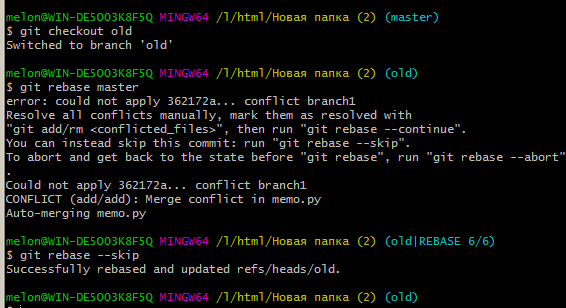
$ git checkout experiment

$ git rebase master

First, rewinding head to replay your work on top of it...

Applying: added staged command

Это работает следующим образом: берётся общий родительский снимок двух веток (текущей, и той, поверх которой вы выполняете перебазирование), определяется дельта каждого коммита текущей ветки и сохраняется во временный файл, текущая ветка устанавливается на последний коммит ветки, поверх которой вы выполняете перебазирование, а затем по очереди применяются дельты из временных файлов. Часто вы будете делать так для уверенности, что ваши коммиты могут быть бесконфликтно слиты в удалённую ветку — возможно, в проекте, куда вы пытаетесь внести вклад, но владельцем которого вы не являетесь



(при наличии конфликтов может быкануть, но это можно скипнуть и отменить, либо исправить конфликт)

1. Основы HTML. Структура HTML-документа. Теги основных элементов <h1>, <p>, <img>, <a>.... На примере простейшей веб-страницы.

<http://www.astro.spbu.ru/staff/afk/Teaching/Help/Tegs.htm>

Язык HTML — язык тегов. Теги описывают структуру HTML-документа. Теги оформляются угловыми скобками , между которыми прописывается имя тега.Теги HTML документа предназначены для управления конструкциями разметки — заголовками, абзацами, списками, таблицами и картинками.

Теги бывают парные и одиночные

*Структура HTML документа*

1. Тип документа

2. Раздел head с технической информацией о странице: заголовок, описание, ключевые слова для поисковых машин, кодировка. Введенная в нем информация в основном не отображается в окне браузера, однако содержит данные, которые указывают браузеру, как следует обрабатывать страницу.

3. Раздел body, в котором уже располагаются все элементы, который видит пользователь. Именно с этим разделом мы, в основном, и будем работать.

DOCTYPE отвечает за корректное отображение веб-страницы браузером. DOCTYPE определяет не только версию HTML (например, html), но и соответствующий DTD-файл в Интернете. Элементы, находящиеся внутри тега , образуют дерево документа, так называемую объектную модель документа, DOM (document object model). При этом элемент является корневым элементом.

<h1></h1>......<h6></h6>. — теги заголовков, от самого большого к самому маленькому.

<p></p> — контейнер для абзаца.

<br> — переносит текст на другую строку без создания абзаца.

<a></a> — добавление гиперссылки в текст. Имеет обязательный атрибут href, в котором указывается ссылка или якорь. Внутри контейнера помещается текст, при нажатии на который происходит переход на другую страницу или другое место на этой же странице.

<img src='<путькфайлу>'></img> — вставка изображения. Имеет атрибут src, который указывает на адрес нужного файла.

<hr> - тэг горизонтальной линии

Стандартная структура тэга:

<название\_тега атрибут1="значение\_атрибута1"> текст внутри тега </ название\_тега >Теги <b> и <strong> внешне делают одно и то же, делая текст полужирным. Разница в том, что <strong> указывает на важность текста, а <b> просто делает текст полужирным. Так, текст в тегах <strong> устройство для чтения текста вслух будет выделять

интонацией, а <b> — нет.

Аналогично с тегами <em> и <i> , делающими текст курсивным. <em> указывает на важность текста, а <i> нет.

<small> Уменьшает размер шрифта на единицу по отношению к обычному тексту.

<sub> Используется для создания нижних индексов. Сдвигает текст ниже уровня строки, уменьшая его размер.

<sup> Используется для создания степеней и верхних индексов. Сдвигает текст выше уровня строки, уменьшая его размер.

<ins> Выделяет текст в новой версии документа, подчёркивая его.

<del> Перечёркивает текст. Используется для выделения текста, удаленного из документа.

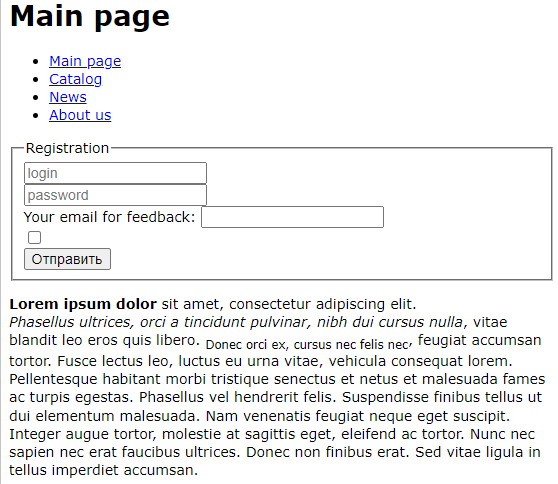
<code> Служит для выделения фрагментов программного кода. Отображается моноширинным шрифтом.

<pre> Позволяет вывести текст на экран, сохранив изначальное форматирование. Пробелы и переносы строк при этом не удаляются.

<q> Используется для выделения коротких цитат. Браузерами заключается в кавычки

Пример простенькой страницы:





1. Основы HTML. Составные теги <table>, <ul>, <ol>. Примеры использования.

**Table**:

Элемент <table> служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов <tr> и <td>. Внутри <table> допустимо использовать следующие элементы: <caption>, <col>, <colgroup>, <tbody>, <td>, <tfoot>, <th>, <thead> и <tr>.

Таблицы с невидимой границей долгое время использовались для верстки веб-страниц, позволяя разделять документ на модульные блоки. Подобный способ применения таблиц нашел воплощение на многих сайтах, пока ему на смену не пришел более современный способ верстки с помощью слоев.

### Синтаксис

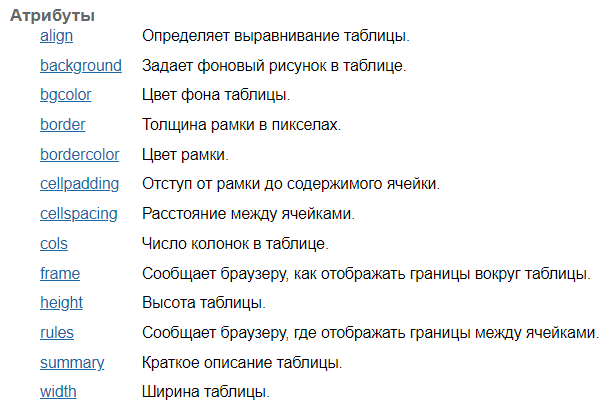
**<table>**

**<tr>**

**<td>**...**</td>**

**</tr>**

**</table>**



<tr> - cтрока таблицы, в которой хранятся ячейки

<td> - непостредственно ячейки таблицы

<th> - ячейка заголовка таблицы

Пример:

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Таблица размеров обуви**</title>**

**</head>**

**<body>**

**<table** border="1"**>**

**<caption>**Таблица размеров обуви**</caption>**

**<tr>**

**<th>**Россия**</th>**

**<th>**Великобритания**</th>**

**<th>**Европа**</th>**

**<th>**Длина ступни, см**</th>**

**</tr>**

**<tr><td>**34,5**</td><td>**3,5**</td><td>**36**</td><td>**23**</td></tr>**

**<tr><td>**35,5**</td><td>**4**</td><td>**36⅔**</td><td>**23–23,5**</td></tr>**

**<tr><td>**36**</td><td>**4,5**</td><td>**37⅓**</td><td>**23,5**</td></tr>**

**<tr><td>**36,5**</td><td>**5**</td><td>**38**</td><td>**24**</td></tr>**

**<tr><td>**37**</td><td>**5,5**</td><td>**38⅔**</td><td>**24,5**</td></tr>**

**<tr><td>**38**</td><td>**6**</td><td>**39⅓**</td><td>**25**</td></tr>**

**<tr><td>**38,5**</td><td>**6,5**</td><td>**40**</td><td>**25,5**</td></tr>**

**<tr><td>**39**</td><td>**7**</td><td>**40⅔**</td><td>**25,5–26**</td></tr>**

**<tr><td>**40**</td><td>**7,5**</td><td>**41⅓**</td><td>**26**</td></tr>**

**<tr><td>**40,5**</td><td>**8**</td><td>**42**</td><td>**26,5**</td></tr>**

**<tr><td>**41**</td><td>**8,5**</td><td>**42⅔**</td><td>**27**</td></tr>**

**<tr><td>**42**</td><td>**9**</td><td>**43⅓**</td><td>**27,5**</td></tr>**

**<tr><td>**43**</td><td>**9,5**</td><td>**44**</td><td>**28**</td></tr>**

**<tr><td>**43,5**</td><td>**10**</td><td>**44⅔**</td><td>**28–28,5**</td></tr>**

**<tr><td>**44**</td><td>**10,5**</td><td>**45⅓**</td><td>**28,5–29**</td></tr>**

**<tr><td>**44,5**</td><td>**11**</td><td>**46**</td><td>**29**</td></tr>**

**<tr><td>**45**</td><td>**11,5**</td><td>**46⅔**</td><td>**29,5**</td></tr>**

**<tr><td>**46**</td><td>**12**</td><td>**47⅓**</td><td>**30**</td></tr>**

**<tr><td>**46,5**</td><td>**12,5**</td><td>**48**</td><td>**30,5**</td></tr>**

**<tr><td>**47**</td><td>**13**</td><td>**48⅔**</td><td>**31**</td></tr>**

**<tr><td>**48**</td><td>**13,5**</td><td>**49⅓**</td><td>**31,5**</td></tr>**

**</table>**

**</body>**

**</html>**

**Нумерованный список:**

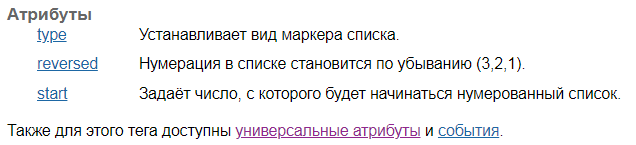
Тег <ol> устанавливает нумерованный список. Каждый элемент списка должен начинаться с тега <li>. Если к тегу <ol> применяется таблица стилей, то элементы <li> наследуют эти свойства.

### Синтаксис

**<ol>**

**<li>**элемент нумерованного списка**</li>**

**<li>**элемент нумерованного списка**</li>**



**Пример**

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Тег OL**</title>**

**</head>**

**<body>**

**<ol>**

**<li>**Чебурашка**</li>**

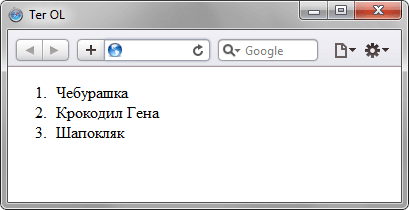
**<li>**Крокодил Гена**</li>**

**<li>**Шапокляк**</li>**

**</ol>**

**</body>**

**</html>**



**Маркированный список:**

Тег <ul> устанавливает маркированный список. Каждый элемент списка должен начинаться с тега <li>. Если к тегу <ul> применяется таблица стилей, то элементы <li> наследуют эти свойства.

### Синтаксис

**<ul>**

**<li>**элемент маркированного списка**</li>**

**</ul>**

**Пример**

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Тег UL**</title>**

**</head>**

**<body>**

**<ul>**

**<li>**Чебурашка**</li>**

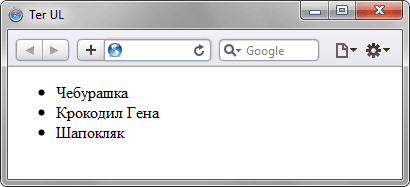
**<li>**Крокодил Гена**</li>**

**<li>**Шапокляк**</li>**

**</ul>**

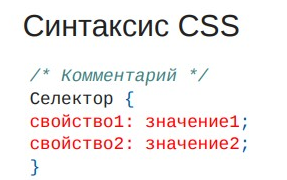
**</body>**

**</html>**



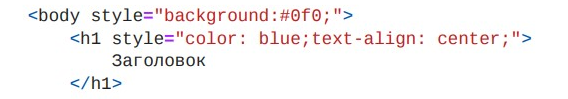
1. Основы CSS. Способы подключения css в html документ. Селектор, способы оформления для каждого вида подключения. Пример страницы с использованием каждого способа.

CSS– это каскадные листы стилей, которые применяются для описания внешнего вида веб-документа, написанного при помощи языка разметки HTML.(т.е. отвечает за оформление документа).



3 способа: (по приоритету )

* Inline-стили (тэг style)



* Стили в разделе head (в атрибуте type)

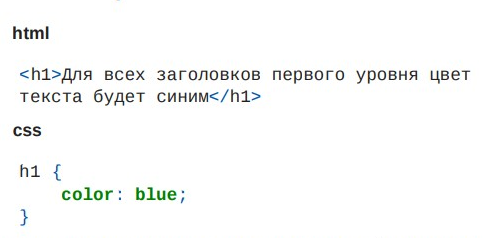


Внешний CSS файл(Создаем файл с расширением .css. Обычно, так же как и в случае с картинками, все css файлы размещают в отдельной папке.)

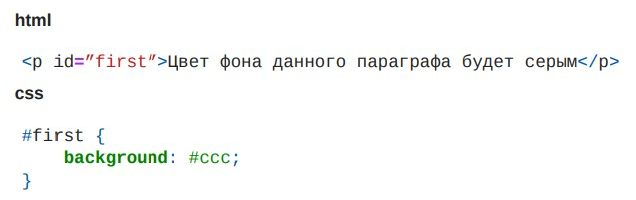


*Селекторы в CSS*

* Селекторы тегов (применяется ко всем указанным тегам!)

**

* Селекторы идентификаторов

**

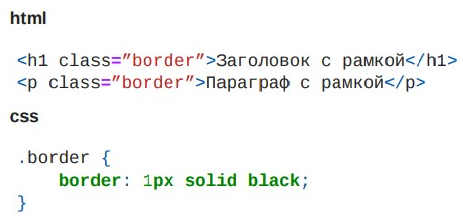
!идентификатор должен быть уникальным, т.е. нельзя задавать одно и то же имя двум и более элементам.

*#собственное имя*

* Селекторы классов

Классы отличаются от идентификаторов тем, что применять один и тот же стиль к разным элементам.

*.название*

**

* Селекторы атрибутов

В данном примере, сначала указывается стиль для всех картинок, у которых присутствует атрибут title . Для этого название атрибута указывается в квадратных скобках, сразу после названия тега.

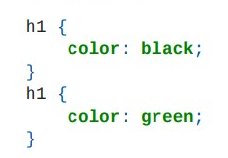
Второй пример. Стиль будет применяться для всех тегов <input>, в значении атрибута type которого присутствует значение text , т.е. для всех обычных текстовых полей ввода.

**

1. Основы CSS. Каскадная модель работы CSS. Модификатор !important. Блочная структура страницы. Тег <div>, <span>. Пример страницы с использованием и дизайном.

Существует такое понятие, как каскадирование, которое применяется тогда, когда одному и тому же элементу пытаются присвоить одни и те же стили. Например, мы всем параграфам пытаемся присвоить сначала черный цвет, а потом зеленый.

Каскад таблицы стилей, если говорить упрощённо, означает, что порядок следования правил в CSS имеет значение; когда применимы два правила, имеющие одинаковую специфичность, используется то, которое идёт в CSS последним.



Итог: зеленый цвет

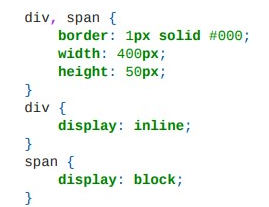
Рассмотрим приоритеты стилей автора проекта. Самым важным свойством является то, у которого после значения свойства установлена директива !important.



Итог: чёрный цвет

* Тег div является блочным элементом, который предназначен для выделения фрагмента документа и затем изменением его содержимого.
* Тег span является строчным элементом, который предназначен для выделения фрагмента текста внутри других тегов, таких как <p>,<table> или <div>.

При помощи css можно изменить тип элемента, т.е. блочный тег можно сделать строчным, а строчный - блочным. Для этого существует css свойство - display . Вернёмся к предыдущему примеру и для элементов div задать значение свойства display: inline, а для - значение display: block;



Блочный элемент (display: block;) создает разрыв строки перед тегом и после него.

-Блочные элементы могут содержать внутри себя элементы любого типа.

-Нельзя размещать блочные элементы внутри строчных, за исключением элемента img. -Для блочных элементов можно задавать margin и padding .

-Свойства width и height устанавливают ширину и высоту области содержимого элемента.

* Внутренний отступ или поле элемента (padding) добавляет отступы внутри элемента, между его основным содержимым и его границей. Не может быть отрицательным.
* Внешний отступ (margin) добавляет отступы за границами элемента, создавая тем самым промежутки между элементами. Они всегда остаются прозрачными, и через них виден фон родительского элемента. Значения padding и margin задаются в следующем порядке: верхнее, правое, нижнее и левое.
* Граница или рамка элемента задается с помощью свойства border. Если цвет рамки не задан, она принимает цвет основного содержимого элемента, например, текста.
* По умолчанию все равно 0.

*4 вида позиционирования:*

* Абсолютное позиционирование

- положение задаётся относительно краёв браузера через значение absolute свойства position.

- не меняет своё исходное положение.

- свойства left и top имеют более высокий приоритет по сравнению с right и bottom.

- width, hight имеют более высокий приоритет.

- перемещается вместе с документом при его прокрутке.

* Относительное позиционирование

- relative - элемент будет смещаться относительно его определенного в настоящее время положения.

- добавление свойств left , top , right и bottom изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения.

- Положительное значение left определяет сдвиг вправо от левой границы элемента, отрицательное — сдвиг влево.(right наоборот)

- Положительное значение top задаёт сдвиг элемента вниз, отрицательное — сдвиг вверх. (bottom наоборот)

- При смещении элемента относительно исходного положения, место, которое занимал элемент, остаётся пустым и не заполняется ниже или вышележащими элементами.

* Фиксированное положение

- Фиксированное положение слоя задаётся значением fixed свойства position.

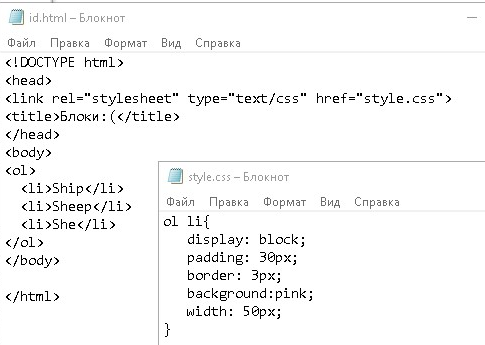
- не меняет своего положения при прокрутке веб-страницы.

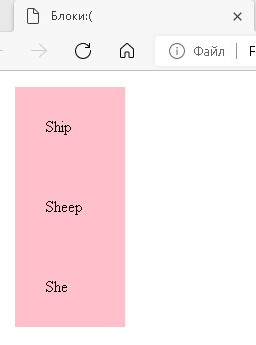
* Z-index

- Каждый элемент может находиться как ниже, так и выше других объектов веб-страницы, их размещением по z-оси и управляет z-index. Это свойство работает только для элементов, у которых значение position задано как absolute , fixed или relative.

- Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше.

- При равном значении z-index, на переднем плане находится тот элемент, который в коде HTML описан ниже.





1. Основы CSS. Медиа запросы. Цели применения, синтаксис. Пример работы.

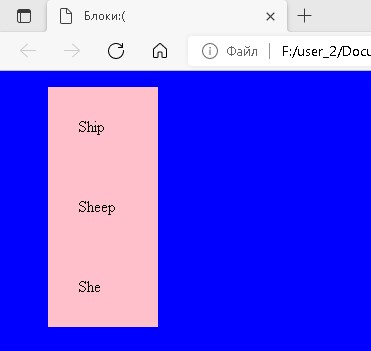
Наряду с типами носителей, в CSS3 включена поддержка различных технических параметров устройств, на основе которых требуется загружать те или иные стили.

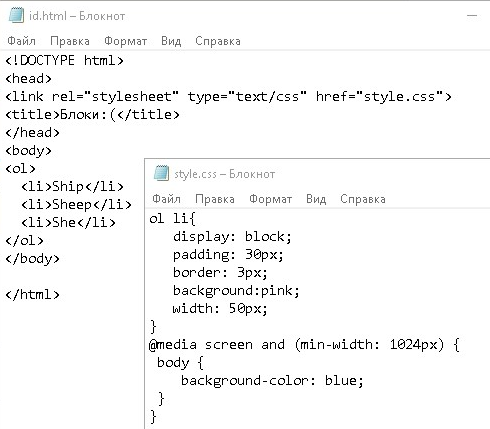
К примеру, можно определить смартфон с максимальным разрешением 640 пикселей и для него установить одни стилевые свойства, а для остальных устройств другие.

Возможности медиа-запросов не ограничиваются выявлением мобильных устройств, с их помощью можно создавать адаптивный макет. Такой макет подстраивается под разрешение монитора и окна браузера, меняя при необходимости ширину макета, число колонок, размеры изображений и текста. Медиа-запросы ограничивают ширину макета и, при достижении этого значения (к примеру, за счёт уменьшения окна или при просмотре на устройстве с указанным размером), уже применяется другой стиль.

*Синтаксис*

Все запросы начинаются с правила @media , после чего следует условие, в котором используются типы носителей, логические операторы и медиа-функции.





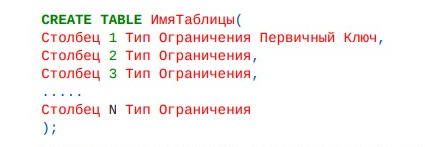
1. Основы SQL. Создание базы данных. Создание таблицы. Ограничение полей (unique, not null). Типы данных. Удаление таблицы. На примере SQLite.

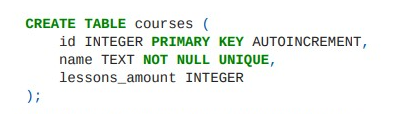
База данных — это структурированный набор данных, а СУБД — приложение, которое обеспечивает работу с базами данных.

Для работы с базами данных (добавлять, изменять, получать требуемую информацию) используется структурированный язык запросов — SQL.

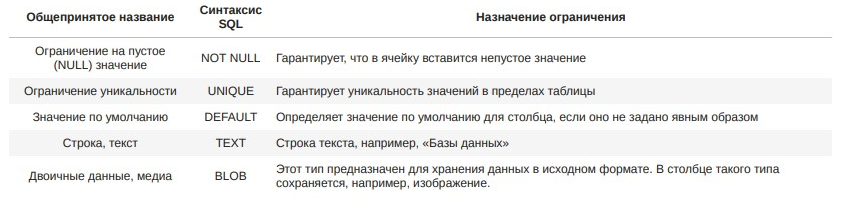
SQLite — это компактная кросс-платформенная встраиваемая СУБД, разработка которой ведётся как проект с открытым кодом.

*Создание таблицы*

**

**

*Ограничение полей*

**

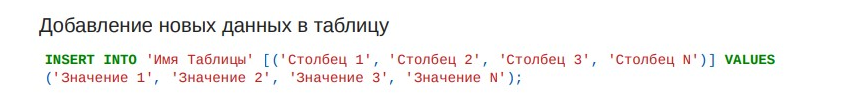
*Типы данных*

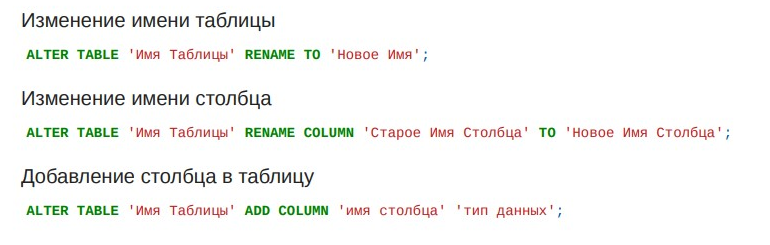
**

*Удаление таблицы*

DROP TABLE 'Имя Таблицы';

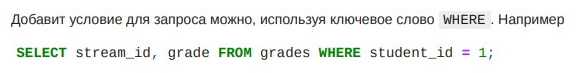
1. Основы SQL. Добавление данных. Изменение данных. Простейшие запросы на выборку данных из таблицы. На примере SQLite.



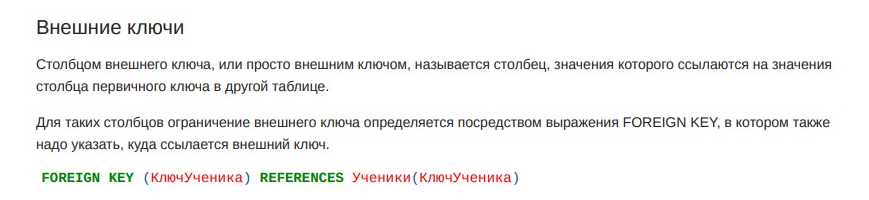


*Выборка данных*

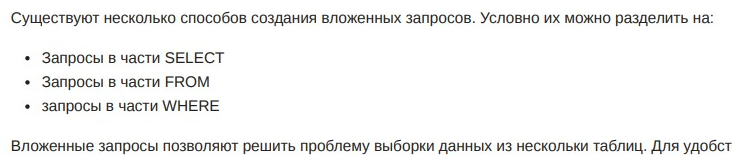
**

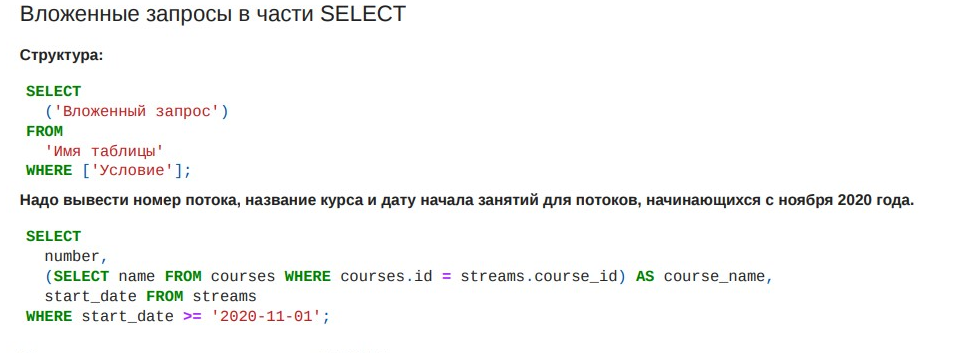


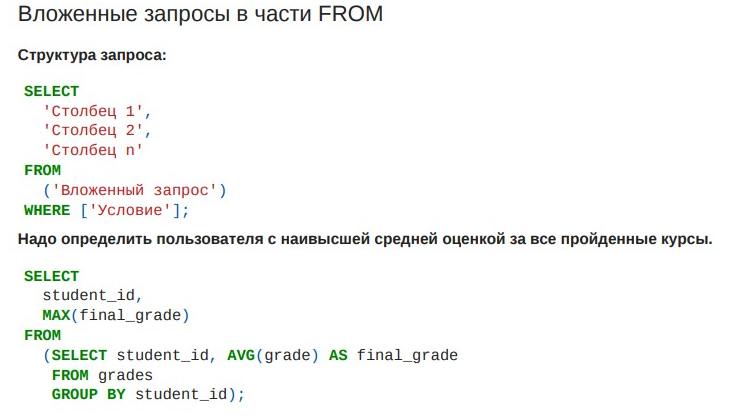
1. Основы SQL. Внешние ключи. Вложенные запросы к базе данных для выборки данных из нескольких таблиц. Создание индексов для полей б.д. На примере SQLite.

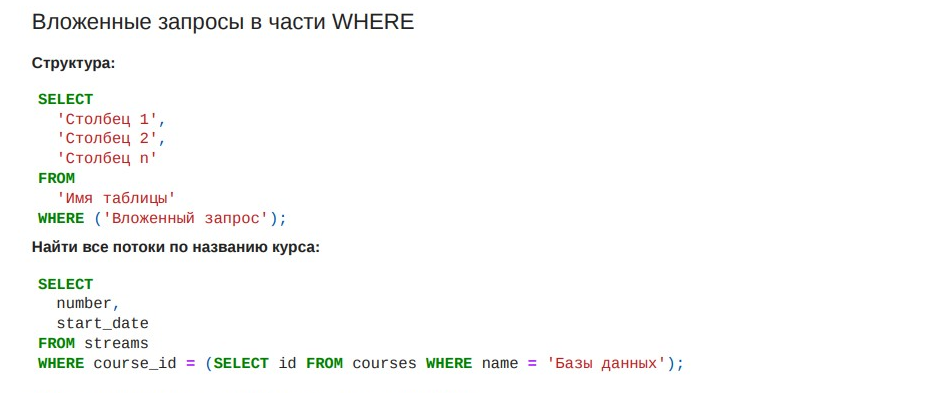


Вложенные запросы





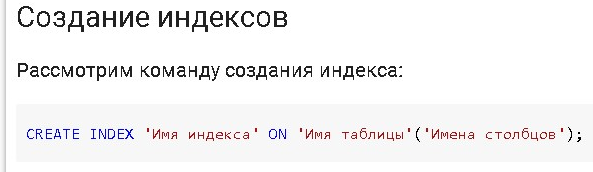




*Индексы*

Индекс — структура базы данных, в которой хранятся отсортированные значения столбца или нескольких столбцов, с указателями на место в физическом файле, где хранится соответствующая строка таблицы.

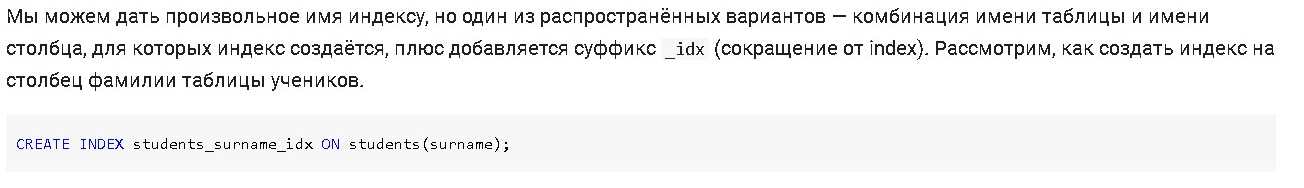
Если элементы отсортированы, значит можно применить например бинарный поиск, что сильно ускорит выборку данных.



- на создание индексов, их хранение и поддержание в актуальном состоянии требуются ресурсы — процессорное время, место на носителях информации, оперативная память системы.

- Когда надо создавать индексы? Общепринятым подходом считается создавать индексы на этапе эксплуатации системы, когда уже понятно, какие запросы выполняются наиболее часто и значительно нагружают базу данных.

Пример



1. Основы SQL. Объединение таблиц с JOIN, UNION. Транзакции. На примере SQLite.

Объединение данных через UNION



Пример

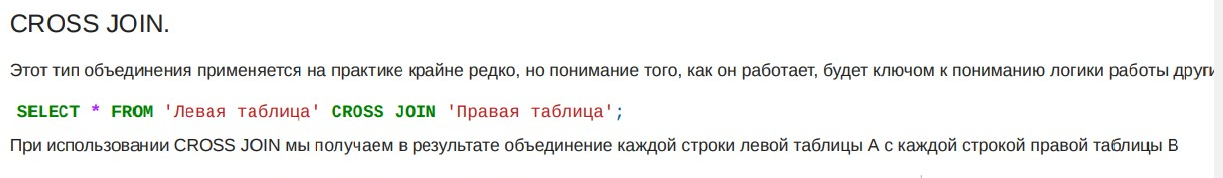


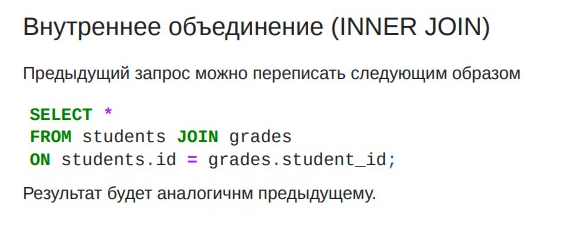
!!!При помощи UNION мы получаем объединение (как при работе со множествами) результатов запросов. В таком случае результаты будут уникальные. Если нужно получить результаты с повторением, используется оператор UNION ALL.

Для объединения требуется:

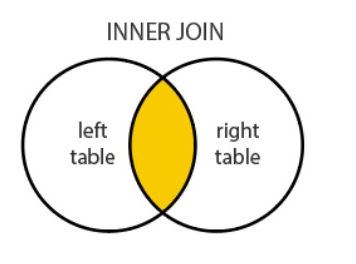
* совпадение количества выводимых столбцов каждого запроса;
* одинаковые по типу выводимые данные каждого столбца.







Такой тип объединения называется внутренним, так как в результате появятся те строки двух таблиц, между которыми найдено соответствие, определённое в части запроса WHERE . При использовании JOIN рекомендуется определять условие объединения таблиц, выбирая часть выражения ON .





****

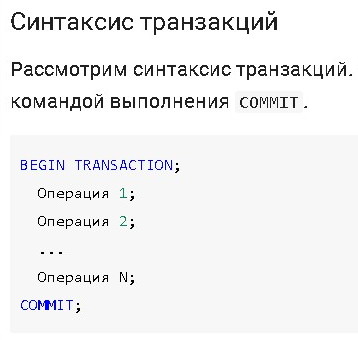
Right JOIN не поддерживается в SQLite, но его можно реализовать через LEFT JOIN наоборот.

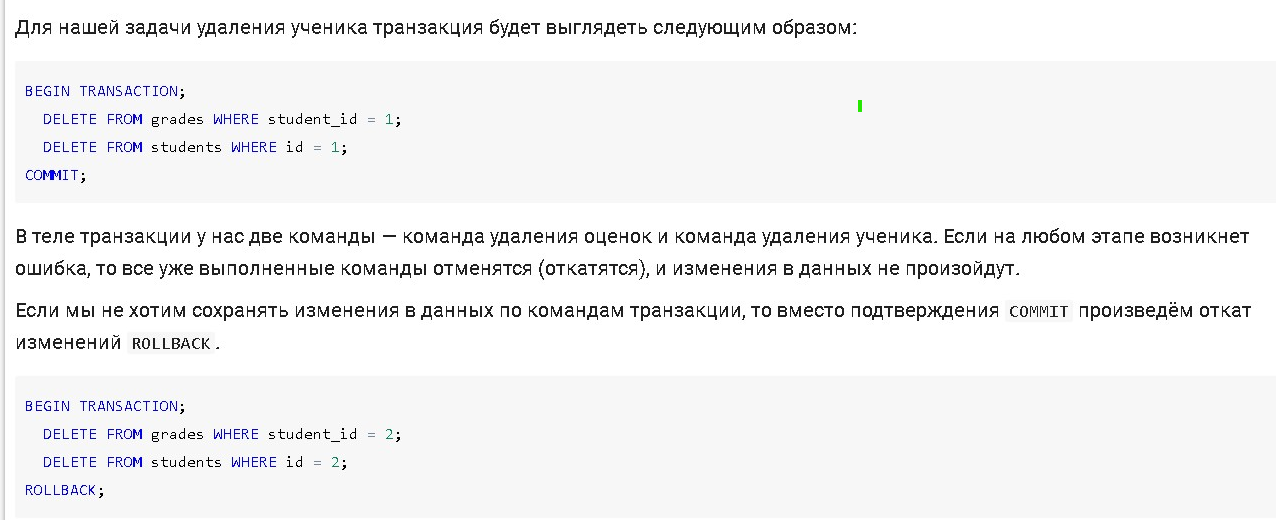
*Транзакции*

**Транзакция** — набор команд, которые выполняются только вместе. Если при выполнении любой команды возникает ошибка, то для всех уже выполненных команд выполняется откат, и данные остаются в исходном состоянии. (Например, когда нужно удалить из таблицы элемент, тогда данные нужно удалять сразу их нескольких таблиц).

4 базовых свойства:

1. Атомарность — обеспечивает выполнение правила «всё или ничего». Выполняются все команды, или ни одна из них, третьего варианта нет.
2. Целостность — обеспечивает корректные изменения в данных при успешном выполнении транзакции и гарантирует отсутствие нарушений связности данных различных таблиц.
3. Изолированность — обеспечивает независимое выполнение транзакций, транзакции не влияют друг на друга.
4. Устойчивость — обеспечивает предсказуемость поведения транзакции, когда возникают сбои системы.





Вопрос 2. Практические основы веб программирования

Задания для HTML, CSS и SQL на примере одной системы.

Иван Петрович уже давно работает в школе села Айтиево учителем информатики. В рамках программы информатизации в школе нужно внедрить систему обработки новостей с таргетированием по категориям - ученики, родители, педагоги.

В системе должны быть следующие элементы:

* Регистрация ученика / педагога / администрации школы
* Страница новостей (для каждой из групп отдельная)
* Главная страница

Помогите Ивану Петровичу написать систему.

1. Написать главную страницу сайта. Предусмотреть форму для входа, меню сайта, с учетом т.з. из описания, хедер и футер. Оформить с использованием CSS. Страница должна быть responsive.
2. Написать страницу новостей для одной из категорий пользователей из т.з. Предусмотреть меню сайта, с учетом т.з. из описания, хедер и футер. Оформить с использованием CSS. Страница должна быть responsive.
3. Написать страницу регистрации. Структура должна давать возможность выбора одной из категорий пользователей из т.з. Предусмотреть меню сайта, с учетом т.з. из описания, хедер и футер. Оформить с использованием CSS. Страница должна быть responsive
4. Создать таблицы пользователей и категорий пользователей из т.з. Продумать поля таблиц, настроить связь между таблицами. Заполнить каждую таблицу тремя или более записями, убедиться, что все работает при помощи запросов. Создание, заполнение и тестирование выполняются на Python. Тестирование при помощи UnitTest
5. Создать таблицы новостей и групп пользователей из т.з. Продумать поля таблиц, настроить связь между таблицами. Заполнить каждую таблицу тремя или более записями, убедиться, что все работает при помощи запросов. Создание, заполнение и тестирование выполняются на Python. Тестирование при помощи UnitTest
6. Создать таблицы пользователей и новостей из т.з. Продумать поля таблиц, настроить связь между таблицами. Заполнить каждую таблицу тремя или более записями, убедиться, что все работает при помощи запросов. Создание, заполнение и тестирование выполняются на Python. Тестирование при помощи UnitTest
7. Создать таблицы пользователей и категорий пользователей из т.з. Продумать поля таблиц, настроить связь между таблицами. Заполнить каждую таблицу тремя или более записями, убедиться, что все работает при помощи запросов. Создание, заполнение и тестирование выполняются на Python. Тестирование при помощи UnitTest
8. Создать таблицы пользователей из т.з. Продумать поля таблицы. Создать индекс для подходящего поля. Заполнить таблицу тремя или более записями, убедиться, что все работает при помощи запросов. Создание, заполнение и тестирование выполняются на Python. Тестирование при помощи UnitTest.

Вопрос 3. Задача