INTERVIEW

1. Типы данных
2. Отличие == и ===
3. Приведение типов в булеан
4. Строки и числа
5. Принципы ООП

Абстракция – выделение в классе значимой информации и исключение незначимой.

Наследование – возможность одного класса перенимать свойства и методы другого, наследуя свойства и методы. Набор методов класса, доступных для использования другими классами называется интерфейсом.

class Coder extends Human {

constructor(name) {

super(name)

}

}

Инкапсуляция – сам класс является капсулой, содержащей свойства и методы, которые могут быть скрытыми. Для этого нужно указать модификатор private.

**Private** - только текущий класс будет иметь доступ к полю или методу. **Protected** - только текущий класс и подклассы этого класса будут иметь доступ к полю или методу. **Public** - любой класс может ссылаться на поле или вызывать метод.

Полиморфизм – определяем метод в классе. Создаем новый класс который будет наследовать предыдущий класс. Внутри нового класса переопределяем метод.

class Human {

constructor(name) {

this.name = name;

}

say() {

return `Hello, my name is ${this.name}, I like travelling`;

}

}

class Coder extends Human {

constructor(name) {

super(name)

}

say() {

// Переопределяем метод родителя say для отображения нового значения.

return `Hello, my name is ${this.name}, I like coding`;

}

}

1. Прототипное наследование

Возможность изменить поведение в унаследованных сущностях

JavaScript часто описывают как язык **прототипного наследования** — каждый объект, имеет **объект-прототип**, который выступает как шаблон, от которого объект наследует методы и свойства. Объект-прототип так же может иметь свой прототип и наследовать его свойства и методы и так далее. Это часто называется **цепочкой прототипов**и объясняет почему одним объектам доступны свойства и методы которые определены в других объектах.

*worker.\_\_proto\_\_ = human;*

*Object.setPrototypeOf(worker, human);*

Прототип – это как класс, шаблон. Хьюман – то прототип воркера.

1. Всплытие

Переменные созданные через ключевое слово var всплывают, через let , const – не всплывают.

console.log(k); //ReferenceError variable is not exist

let k = 1;

console.log(l); *//Undefined т к переменная созданная с помощью ключевого слова var*

*// всплывает и попадает вверх интерпретатора.*

*// Она уже существует по ей еще не присвоено значение*

var l = 1;

1. Отличие let var cоnst

*// Если переменная создана с помощью ключевого слова const тогда ее значение изменить нельзя*

*// const name = 'Nastya'*

*// name = 'NastyaSTR' будет ошибка*

*// но можно*

const obj = {};

const arr = [];

obj.key = 'a';

arr.push(1);

*// так можно. изменять элементы массива и значения ключей. нельзя присваивать новое значение*

*//  самому объекту или массиву*

*// let - блочная область видимости. var - глобальная область видимости*

{

    let test22 = 'a';

    var test21 = 'q';

    const test23 = 't';

}

*// console.log(test22); // ,будет ошибка т к let имеет блочную область видимости*

*// console.log(test21); // var имеет глобальную область видимости поэтому отработает.*

*// Но если это будет не блок а функция то тогда значение будет undefined*

*// console.log(test23); // ,будет ошибка т к const имеет блочную область видимости*

1. Фанкшн декларейшн и фанкшн экспрешн

console.log(test()); *//Отработает все ок т к функция вспылает вверх интерпретатора если она*

*//  создана с помощью объявления функции  Function Declaration*

function test() {

*return* 1;

}

*// console.log(test2()); //Не тработает т к функция не вспылает вверх интерпретатора если она*

*//  создана с помощью функционального выражения  Function Expression*

*// var let const не важно что*

var test2 = function () {

*return* 1;

};

1. Наследование в ЕS6 и прототипное наследование

Es6 – extends obj. \_\_proto\_\_= prototypeHere

1. Замыкание

function v() {

    let counter = 0;

*return* function () {

        counter++;

*return* counter;

    };

}

const x = v();

console.log(x()); *//1*

console.log(x()); *//2*

*// Когда функция имеет доступ к переменных из вышестоящего скоупа и запоминает ее*

1. Методы массивов

arr.push(...items) – добавляет элементы в конец,

arr.pop() – извлекает элемент из конца,

arr.shift() – извлекает элемент из начала,

arr.unshift(...items) – добавляет элементы в начало.

arr.join() – превратить массив в строку

arr.splice(начиная с, удалить такое количество, на их место вставить это) – можно удалять или заменять

arr.slice(начиная с этого индекса, копировать столько элементов) – копируем элементы

arr.indexOf(element) – вернет индекс . если вернуло -1 – нет аткого элемента

Math.min.apply(num1,num2,num3) – найдет минимальное

Math.max.apply(num1,num2,num3) – найдет максимальное

Map() – перебирает массив видоизменяя каждый элемент массива

Reduce() – на основе всех значений аккумулирует какое-то одно

Sort() – сортирует массив. По умолчанию сортирует как строку, a-b – числа по возрастанию, b-a – числа по убыванию

Some() – вернет true если хотя бы один элемент соответствует условию

every() – вернет true если все элементы соответствует условию

filter() – оставит только те элементы которые проходят по условию

arr.includes(element) – есть ли в массиве элемент

arr1.concat(arr2) – объединяет 2 массива

1. Методы для работы со строками

str.split() – превратит строку в массив

str.repeat(5) –повторить строку 5 раз

str.replace(найти это, заменить на это)

str.indexOf(element) – вернет индекс

toString() –превратит в строку

trim(‘ ttttt ‘) – удалит пробелы по бокам

str.match(/g/) – ищет все сопоставления строки с таким регулярным выражением и возвращает массив

str.search(регексп) – если нужно узнать попадает ли под регулярное выражение

str.exec(регекс) – если только первое совпадение

str.charAt(index) – выведет элемент с таким индексом

str[0]

str.includes(element) – есть ли в строке элемент

str.substring(удалит из строки начиная с этого индекса) – вернет новую обрезанную строку

1. Методы работы с объектами

Object.keys() – все ключи массив

Object.values() – массв=ив значений

Object.entries() – сделает массив из массива вместо объекта

Object.assign(obj1, obj2) – в obj1 будет скопирован obj2

Object.create(student) – создастся экземпляр класса student

1. Методы работы с числами

Number() – превратит в число

Math.min(num1,num2,num3) – найдет минимальное

Math.max(num1,num2,num3) – найдет максимальное

Math.sqrt(корень из этого числа)

Math.pow(number, степень)

Number.isInteger(1) – вернет true если число целое

3.toFixed(столько чисел после запятой оставить) – округляет до сотых десятых и т п

Math.found(number) – к ближайшему целому округление

Math.ceil(number) – округление в большую сторону

Math.floor( umber) – округление в меньшую сторону

parseInt(‘123num’) //123

1. Алгоритмы поиска: линейный бинарный

Линейный ищет по порядку.Бинарный ищет разделив наполовину, если в первой половине есть искомый элемент делит еще раз наполовину и так пока не найдет элемент

1. Функция колбек

Функции, которые передают в другие функции в качестве аргумента, называют **функциями обратного вызова (callback function)**.

В JavaScript функции — это объекты. Поэтому функции могут принимать другие функции в качестве аргументов, а также функции могут возвращать функции в качестве результата. Функции, которые это умеют, называются **функциями высшего порядка**. А любая функция, которая передается как аргумент, называется **callback-функцией**.

1. Асинхронность и синхронность

Изначально Javascript синхронный, все операции идут последовательно. Пока не выполнилась предыдущая операция не будет начинать выполняться следующая. Асинхронный Javascript – это когда операции выполняются не последовательно, первая операция не ждет пока выполнится вторая, пока выполняется первая, вторая тоже будет выполнятся в фоновом режиме. Асинхронной функции нужно явно указать, что она ждёт завершения какой-то команды. Это делают командой await. Её можно применять только внутри асинхронной функции. В любых действиях где пользователи чего-то ждут прежде чем выполнить дальнейшие действия нужна асинхронность.  await - синтаксический сахар внутренней команды then.

1. Практики корошего кода(код конвеншн и т п)
2. NodeJS

* Это среда для выполнения серверного JavaScript

npm - менеджер пакетов(модульных компонентов) для node.js. Для нас менеджер npm важен в том плане, что с его помощью легко управлять пакетами.

**Файл package.json**

Для более удобного управления конфигурацией и пакетами приложения в npm применяется файл конфигурации **package.json**. Так, добавим в папку проекта modulesapp новый файл *package.json*:

Npm init – инициализация package.json

Npm install – установка всех зависимостей

Npm uninstall mocha – удаление зависисмости

--save-dev

Информация обо всех добавляемых пакетах, которые используются при работе приложения, добавляется в секцию **dependencies**. Кроме пакетов, которые применяются в приложении, когда оно запущено и находится в рабочем состояни, например, express, то есть в состоянии "production", есть еще пакеты, которые применяются при разработке приложения и его тестировании. Такие пакеты, как правило, добавляются в другую секцию - **devDependencies**.

В секции скрипт прописыват=ют команды которые затем можно вводить в терминал

**-S, --save:** Пакет будет отображается в ваших dependencies

**-D, --save-dev:** пакет будет отображаться в ваших devDependencies

-g - установит глобально на локальный компьютер

{

  "name": "modulesapp",

  "version": "1.0.0",

  "scripts" : {

    "start" : "node app.js",

    "dev" : "node app.js Tom 26"

  }

}

(npm start, npm dev)

1. Промиссы
2. Методы call apply bind

*let human = {*

*name: 'humanName',*

*sur: 'lala',*

*sayHello(surname) {*

*return `My name is ${this.name} ${surname}`;*

*},*

*};*

*let worker = {*

*name: 'Sasha',*

*};*

*console.log(human.sayHello.call(worker, 'Ivanov'));*

*console.log(human.sayHello.apply(worker, ['Petrov']));*

*console.log(human.sayHello.bind(worker, 'Sidorov')());*

person.testName('all', 'Andrew');

person.testName.call(john, 'nothing', 'Max'); *//позволит использовать контекст john*

person.testName.apply(john, ['nothing', 'Max']); *//то же самое просто 2й и 3й аргмент передаются как массив*

person.testName.bind(john, 'nothing', 'Max')(); *// отличие от предыдущих методов в том что он не вызывает*

*// функцию а возвращает ее поэтому ее надо вызвать или засунуть функцию в переменную и вызвать переменную*

1. Agile, Waterfall
2. Объекты как хранятся и как хранятся примитивы

Объекты по ссылке

*// Примитивы передаются по значению. Объекты по ссылке*

let a = 1;

let b = 1;

console.log(a === b); *//true*

let c = [1, 2];

let d = [1, 2];

console.log(c === d); *//false*

let e = d;

e.push(4);

console.log(e); *//[1,2,4]*

console.log(d); *//[1,2,4] d также изменился чтобы предотварить надо было написать let e = d.concat();*

let j = c.concat();

j.push(4);

console.log(j); *//[1,2,4]*

console.log(c); *////[1,2]// сработало значение не поменялось*

1. Scrum , Kanban
2. Что такое контекст

Код всегда выполняется в некотором **контексте** – окружении, хранящем все параметры, необходимые для правильной работы программы. Когда мы вызываем функцию создается контекст.

1. Page Object Model

* Разделение тестового сценария и контента страниц
* DSL Domain Specific Language. Язык специфичный для домена. Наименование классов и методов должно иметь ту самую специфику предметной области с которой мы работаем(не clickToCart а addItemToCart).
* Do not repeat yourself() DRY . Код который будем писать разделить на модули с уникальным содержимым.
* Keep It Simple Stupid. Упрощай до невозможности. Лучше реализовать много маленьких PO чем один большой.
* You are not Gonna Need It. Не стоит сразу с ходу реализовывать весь функционал, писать только то, что нужно на данный момент. Ориентироваться только на тот функционал который требуется сейчас.

Делим все на объекты из которых состоит страница. Позволяет переиспользоват код, делает его более читаемым, поддерживать легче. Из минусов требуется больше времени на разработку и люди владеющие навыками.

1. Промиссы

**Промис (promise)** - это объект, представляющий результат успешного или неудачного завершения асинхронной операции:resolved, reject, pending). Обравботка промиса если он выполнен успешно – then. Если ошибка – catch. Finally выполнится в любом случае.

const myPromise = new Promise(function(){

    console.log("Выполнение асинхронной операции");

});

Так создается новый промис. В агргумент идет callback

1. Bisness Object
2. Cлои test automation framework

PO

Тестовые сценарии

Бизнес объекты(тестовые данные)

Инструменты используемые для автоматизации: фреймворк для автоматизации, unit framework, инструкменты для репортеров

Интеграция CI. GitLabCI с помощью yaml . Jenkins

1. Виды test automation framework

* Линейный – например Selenium IDE. Не обязательно знать программирование. Просто записывается последовательность действий с помощью специальных инструментов и в дальнейшем этот тест можно будет воспроизвести. Плюсы: такие тесты дешевые и не требуют знаний программирования
* Модульный – есть сохраненные действия и они могут быть переиспользованы
* Data Driver –тестовые данные отделяются от тестовых сценариев и хранятся в специальных таблицах. Например login – “natasha”, password – ‘12345’
* Keyword Driven – существует база ключевых слов которая хравнится где либо. Представляет собой таблицу, которая читается построчно и содержит ключевые слова на каждом шаге. Таблица является тест кейсом
* Behavior Driven – используя BDD подход можно задействовать людей, которые не разбираются в программировании, но они будут понимать что будет происходить в тестах и даже сами могут писать сценарии. Клиенты, бизнес-менеджеры, менеджеры проектов – все они смогут понимать о чем идет речь
* Hybrid – комбинация должна быть такой чтобы слабые стороны одного фреймворка компенсировались сильной стороной другого

1. Test automation framework какие задачи решает и что это

* Минимальное смешательство человека а значит снижении ошибок
* Переиспользование кода
* Автоматические репорты
* Максимально тестовое покрытие
* Включение тестов в CI СD уменьшает зависимость от комманд

1. Какие факторы определяют эффективность автоматизированных тестов?(Экономи времени – автом запуск, автоматическая генерация репортов, подключение в CI)
2. Локаторы. Практики хороших локаторов

* Использовать уникальные классы, айди, атрибуты для поиска элементов которые отражают функциональность, а не стилевое оформление
* Избегать автогенерируемых айди, классов и т д
* Избегать использования индекса элементов, даже если есть четкие требования о структуре документа лучше все равно избегать
* Если айти кнопки “ok” но она называется “approve” тогда возможно ее будут править и это важно предугадать

1. Что может xpath и не может css

Xpath в отличие от css может искать элемент по содержащему тексту и перемещаться вперед назад. Css только вперед

1. Ожидания явные и неявные explicit/implicit

Implicit wait указывает один раз и он применится к поиску всех элементов в DOM.

Explicit wait тут мы пишет конкретное условие и webdriver будет ждать пока оно не выполнится.(например кнопка кликабельна станет или ждать пока текст не исчезнет).

1. Гит. Стратегии работы в команде

* Гитфлоу. Все разработчики работают в одном репозитории.Когда создают фичу, то создают отдельную ветку, туда пушат код, и создают пул реквест и если все ок то пушат в главную ветку. В главной ветке всегда будет стабильная версия.
* Integration Workfrow. Каждый разработчик имеет собственный репозиторий.Также есть главный репозиторий для всех разработчиков, откуда те подтягивают стабильную версию.Разработчик закончил писать код, коммитит и пушит в репозиторий человека, который отвечает за интеграцию. Тот человек просматривает и если все ок то пушит в главный репозиторий.
* Диктатор и лейтенант. Есть главный репозиторий. Один репозиторий диктатора и много репозиториев лейтенантов. Такая стратегия используется для очень крупных проектов. Разработчики пишут код, отправляют в репозиторий лейтенанта, у каждого разработчика свой лейтенант. Далее если код хороший то они отправляют его в репозиторий диктатора и если все ок то диктатор отправляет это в главный репозиторий.
* Forking Workfow. Разработчик форкает репозиторий. Вносит изменения в своем репозитории, отправляет их в репозиторий оригинальный. Автор оригинального репозитория смотрит и если все ок то пушит эти изменения.

1. Синхронизация в вебдрайвер

- это согласованность выполнения кода теста с событиями происходящими на странице

1. Как отключить тест в МОКе один. Как наоборот прогнать один тест

it('logs a', function(done) {

console.log('a');

done();

});

it('logs b', function(done) {

console.log('b');

done();

});

mocha –grep 'logs a' – запустит один тест , название передаем в команду

Если надо запустить 1 тестовый файл то – npm run test -- --spec testTitle

1. Mocha

Npm I mochawesome –save-dev – это для автоотчетов о тестировании.

Npm I nyc –save-dev – это для анализа тестового покрытия => npm run coverage

Npm I husky –save-dev – можно установить првило для коммитов например коммитить только если function 80% покрыто

1. Cucumber

Feature – название фичи

Background – предусловие(общее для нескольких сценариев)

Scenario – тест кейс

Given – предусловие типа открыть страницу и т п

When - действия пользователя на странице

Then – описание ожидаемого результата

Шаги представлены регулярными выражениями. Встроена библиотека чай с expect

1. Webdriver Selenium
2. WebdriverIO/Cypress
3. Отличие findElement / findElements

findElement – возвращает 1 элемент, если не найдет то выбрасывает эксепшн. findElements – не выбрасывает эксепшн если не находит возвращает пустой массив, если находит то массив элементов

1. Yargs
2. TDD/BDD
3. Паттерны для автоматизации: PO, PageFactory, разделение тестового сценария и контента страниц, включение тестов в Ci CD
4. Continius Integration/ Continius Delivery.
5. Крон синтаксис
6. Что такое автоматизированный тестовый сценарий? Автоматизированное тестирование
7. Среды тестирования

* Среда разработка (Development Env) – это среда, в которой работаю программисты. Здесь они занимаются написанием и отладкой кода, а также выполняют модульное тестирование
* Среда тестирования (Test Env) – это окружение, в котором работает команда QA. Здесь выполняется проверки функциональности и регресс с использованием тестовых данных. Как правило эта среда не связана или частично связана с внешними системами (нет полноценной интеграционной схемы)
* Продуктивная среда (Production Env) – это окружение, в котором развернуто ПО, где продукт доступен пользователям  
  Source: https://doitsmartly.ru/all-articles/sw-testing/89-test-environment.html

1. Сборка(билд) и релиз что это

**Build - это версия программного обеспечения, которую команда разработчиков передает команде тестирования для целей тестирования, а Release - это программное обеспечение, которое команда тестирования передает клиенту.**

1. SQL
2. Что такое баг. Обязательные поля
3. Жизненный цикл бага
4. Severity/Priority
5. Что такое требования
6. Что такое тест-кейс. Обязательные поля
7. Что такое чек лист
8. Цели тестирования
9. Виды тестирования
10. Уровни тестирования
11. Пирамида тестирования
12. QA /QC/ Testing
13. Тест план структура
14. Отчет о тестировании структура
15. Тестировая документация виды
16. Тест-дизайн: классы эквивалентности, граничные значения, таблица принятия решений, попарное тестирование, блок-схема, mindmap
17. Верификация и валидация
18. Толстый и тонкий клиент
19. Среды тестирования
20. Rest / soup
21. API
22. Веб сервис
23. http/https
24. Методы запроса
25. Коды ответа
26. Кеш/куки
27. Package.json
28. Принципы хорошего кода
29. Клиент –сервер – база данных
30. Сервер-балансировщик
31. Прогресс\регресс
32. Какие тестовые случаи не подходят для автоматизации?(Те которые будут подвержены постоянным изменениям)
33. Какие факторы определяют эффективность автоматизированных тестов?(Экономи времени – автом запуск, автоматическая генерация репортов, подключение в CI)
34. Преищущества АТ(времяб эффективность)

Минимальное смешательство человека а значит снижении ошибок

Переиспользование кода

Автоматические репорты

Максимально тестовое покрытие

Включение тестов в CI СD уменьшает зависимость от комманд

1. Page Factory

Нужен для работы с тестовыми данными

Мы создаем базовый класс, содержащий в конструкторе различные свойства. Далее создаем класс, который будет содержать объект с ключами и значениями(полями и данными) и возвращать тот базовый класс, в параметр которого передаем объект с данными. Получается что будет создан как раз объект с тестовыми данными. Его создаем в самих тестах

1. Commands: git bash
2. Разница интерфейса и абстрактного класса.

Интерфейс содержит только методы. Абстрактный класс содержит и свойства и методы.

1. **Что такое качество ПО?** Степень близости продукта к ожидаемому результату (требованиям).
2. **Как понять, что тестирование закончено?** Достигнут дедлайн, процент успешно выполненных тестов, все дефекты исправлены, все тест-кейсы пройдены, бюджет исчерпан.
3. Что такое класс. Что такое объект
4. Прокси-сервер.VPN
5. TCP/iP

**Базы данных:**

Всех интересует только SQL, и ничего кроме него.

1. Что такое join, какие они бывают и зачем?
2. Что делает group by? Как отсортировать результат по возрастанию? Как получить результаты которые содержат строку поиска, но не обязательно равны ей?
3. Часто предлагают сходу на листе бумаги написать селект. Обычно есть 2-3 таблицы, и нужно джойнами объединить их и получить какую-нибудь строку.

INTERVIEW

1. Типы данных
2. Отличие == и ===
3. Приведение типов в булеан
4. Строки и числа
5. Принципы ООП
6. Всплытие
7. Отличие let var cоnst
8. Фанкшн декларейшн и фанкшн экспрешн
9. Наследование в ЕS6 и прототипное наследование
10. Замыкание
11. Методы массивов
12. Методы для работы со строками
13. Алгоритмы поиска: линейный бинарный
14. Функция колбек
15. Функции высшего порядка
16. Асинхронность и синхронность
17. Практики корошего кода(код конвеншн и т п)
18. Промиссы
19. Методы call apply bind
20. Agile, Waterfall
21. Scrum , Kanban
22. Page Object Model
23. Bisness Object
24. Cлои test automation framework
25. Виды test automation framework
26. Test automation framework какие задачи решает и что это
27. Какие факторы определяют эффективность автоматизированных тестов?
28. Локаторы. Практики хороших локаторов
29. Ожидания явные и неявные explicit/implicit
30. Гит. Стратегии работы в команде
31. Как отключить тест в МОКе один. Как наоборот прогнать один тест
32. Mocha
33. Cucumber
34. Webdriver Selenium
35. WebdriverIO/Cypress
36. Отличие findElement / findElements
37. Yargs
38. TDD/BDD
39. Паттерны для автоматизации
40. Continius Integration/ Continius Delivery
41. Что такое автоматизированный тестовый сценарий? Авттоматизированное тестирование
42. Среды тестирования
43. Сборка и билд что это
44. SQL
45. Что такое баг. Обязательные поля
46. Жизненный цикл бага
47. Severity/Priority
48. Что такое требования
49. Что такое тест-кейс. Обязательные поля
50. Что такое чек лист
51. Цели тестирования
52. Виды тестирования
53. Уровни тестирования
54. Пирамида тестирования
55. QA /QC/ Testing
56. Тест план структура
57. Отчет о тестировании структура
58. Тестировая документация виды
59. Тест-дизайн: классы эквивалентности, граничные значения, таблица принятия решений, попарное тестирование, блок-схема, mindmap
60. Верификация и валидация
61. Толстый и тонкий клиент
62. Среды тестирования
63. Rest / soup
64. API
65. Веб сервис
66. http/https
67. Методы запроса
68. Коды ответа
69. Кеш/куки
70. Package.json
71. Принципы хорошего кода

Соблюдать код конвенш

Пользоваться инструментами для автоанализа кода

Попарное программирование(код ревью)

Создавать стандарт кодинга в команде

ESLINTER

Коммитов в коде не должно быть

Принципы проектирования: KISS DRY , Слепой копипаст

“Перепишу потом на чистый код”

1. Клиент –сервер – база данных
2. Сервер-балансировщик
3. Прогресс\регресс
4. Какие тестовые случаи не подходят для автоматизации?(Те которые будут подвержены постоянным изменениям)
5. Преищущества АТ(времяб эффективность)
6. Page Factory Page Object
7. Commands: git bash
8. Разница интерфейса и абстрактного класса.
9. **Что такое качество ПО?** Степень близости продукта к ожидаемому результату (требованиям).
10. **Как понять, что тестирование закончено?** Достигнут дедлайн, процент успешно выполненных тестов, все дефекты исправлены, все тест-кейсы пройдены, бюджет исчерпан.
11. Что такое класс. Что такое объект
12. Прокси-сервер.VPN
13. TCP/iP
14. Крон синтаксис

**Базы данных:**

Всех интересует только SQL, и ничего кроме него.

1. Что такое join, какие они бывают и зачем?
2. Что делает group by? Как отсортировать результат по возрастанию? Как получить результаты которые содержат строку поиска, но не обязательно равны ей?
3. Часто предлагают сходу на листе бумаги написать селект. Обычно есть 2-3 таблицы, и нужно джойнами объединить их и получить какую-нибудь строку.