1. Известно, что **a.id** является первичным ключом.
   Перепишите следующий MySQL запрос без использования подзапроса.

```sql
select *
from a
where prop is null
  or id in (select a_id from b where prop is null);
```

2. Есть следующая MySQL схема.

```sql
create schema metro collate utf8_general_ci;
use metro;

create table city
(
 id int auto_increment primary key,
 name varchar(64) not null
);

create table shop
(
 id int auto_increment primary key,
 city_id int not null,
 lat float not null,
 lng float not null,
 constraint shop_city_id_fk
   foreign key (city_id) references city (id)
);
```

   Добавьте в схему информацию о метро согласно условиям:
   - к одному магазину может быть привязано несколько станций в заданном порядке.
   - пересадочные узлы нас пока не интересуют.
   - “а еще мы хотим, чтоб у станций отображался цвет”.

3. Имеется следующий код:

```php
<?php
class a{
   private $name = 'a';
   public function getName(){
        return $this->name;
   }
}

class b extends a{
   protected $name = 'b';
}
class c extends a{
   protected $name = 'c';
}
```

   Исправьте код так, чтобы метод **getName()** корректно возвращал свойство **$name** для объектов классов **b** и **c**.
   Что можно сделать если изменение кода класса **a** недопустимо?

4. Напишите на PHP код для получения рекурсивной структуры дерева категорий из плоской.

Исходные данные:

```
[
    ['id' => 1, 'parent_id' => 0, 'sort' => 0, 'name' => 'категория 1'],
    ['id' => 2, 'parent_id' => 0, 'sort' => 1, 'name' => 'категория 2'],
    ['id' => 3, 'parent_id' => 1, 'sort' => 2, 'name' => 'категория 1-2'],
    ['id' => 4, 'parent_id' => 1, 'sort' => 1, 'name' => 'категория 1-1'],
    ['id' => 5, 'parent_id' => 3, 'sort' => 0, 'name' => 'категория 1-2-2'],
    ['id' => 6, 'parent_id' => 1, 'sort' => 3, 'name' => 'категория 1-3'],
];
```

Результат:

```
[
    [
        'id' => 1,
        'name' => 'категория 1',
        'children' =>
            [
                [
                    'id' => 4,
                    'name' => 'категория 1-1',
                    'children' =>
                        [
                        ],
                ],
                [
                    'id' => 3,
                    'name' => 'категория 1-2',
                    'children' =>
                        [
                            [
                                'id' => 5,
                                'name' => 'категория 1-2-2',
                                'children' =>
                                    [
                                    ],
                            ],
                        ],
                ],
                [
                    'id' => 6,
                    'name' => 'категория 1-3',
                    'children' =>
                        [
                        ],
                ],
            ],
    ],
    [
        'id' => 2,
        'name' => 'категория 2',
        'children' =>
            [
            ],
    ],
];
```

5. Каково типичное предназначение следующего кода?

```
<input type="hidden" name="security" value="2DkkdaG349j3K10D">
```

6. Пусть есть некая форма. Как Вы думаете, где следует производить проверку вводимых данных (бэкенд, фронтенд) и почему.

7. На клиентской стороне предстоит работать с множеством однотипных объектов имеющих уникальный **id**. Какую структуру данных Вы предпочтете для их хранения?

8. Разберите следующий код клиентской части и дайте рецензию в свободной форме (что не понравилось).

```
<html>
<head>
    <style>
        #results ul {
            list-style: none;
            padding-left: 0
        }

        #results li {
            display: flex;
            padding: 6px;
            align-items: center;
        }

        #results .color {
            border-radius: 50%;
            border: solid 1px black;
            width: 12px;
            height: 12px;
        }

        #results .name {
            margin-left: 6px;
        }
    </style>
</head>

<body>
<form id="form">
    город:
    <select id="citySelect">
        <option value="">не выбран</option>
        <option value="1">Санкт-Петербург</option>
    </select>
    станция:
    <input id="stationInput" autocomplete="off">
    <span id="status">
    </span>
    <div id="results">
    </div>
</form>

<script type="text/javascript">
```

```
    let cityId = '';
    let searchInput = '';

    onCitySelectChange = function (el) {
        displayStatus('');
        displayResults([]);
        cityId = el.target.value;
        doSearchRequest();
    };
    onStationInputChange = function (el) {
        displayStatus('');
        displayResults([]);
        searchInput = el.target.value;
        doSearchRequest();
    };
    doSearchRequest = function () {
        displayStatus('Выполняется поиск...');

        const requestCityId = cityId;
        const requestSearchInput = searchInput;
        fetch("/findStations?city=" + cityId + "&term=" + searchInput)
            .then(response => {
                return response.json();
            })
            .then(response => {
                if (requestCityId === cityId && requestSearchInput === searchInput) {
                    if (response.status !== 'ok') {
                        displayStatus(response.error || 'Неизвестная ошибка сервера')
                    } else {
                        displayStatus('Найдено: ' + response.data.results.length);
                        displayResults(response.data.results);
                    }
                }
            })
            .catch((error) => {
                displayStatus('Ошибка при обработке ответа сервера');
            });
    };
    displayStatus = function (message) {
        document.getElementById('status').innerHTML = message;
    };
    displayResults = function (results) {
        document.getElementById('results').innerHTML = '<ul>' + results.map((item) =>
{
            return '<li><div class="color"></div><div class="name">' + item.name +
'</div></li>'
        }).join('') + '</ul>';
    };

    document.getElementById('form').onsubmit = (e) => {
        e.preventDefault()
    };
    document.getElementById('citySelect').addEventListener('change',
onCitySelectChange);
    document.getElementById('stationInput').addEventListener('input',
onStationInputChange);

</script>
</body>
</html>
```

8.1. Используя схему из задачи №2, напишите соответствующую серверную часть на PHP (без использования фреймворков и сторонних библиотек) для обработки GET запросов по следующим URL:

/getCities (возвращает список городов).

/findStations (возвращает ограниченный по длине и отсортированный по релевантности список, а также общее количество станций метро указанного города, название которых содержит искомую строку. Если не указан город или искомая строка пуста - возвращает соответствующую ошибку).

Используйте механизм исключений для обработки ошибок.

8.2. Допишите клиентскую часть:

- добавьте загрузку списка городов с сервера.
- оптимизируйте количество запросов к серверу при наборе искомой строки, полагая, что типичная скорость ввода составляет ~4 зн/с, а сервер отвечает за ~0.3с.
- добавьте обработку известных ошибок (не посылать запросы, которые заведомо вернутся с ошибкой).
- измените сообщение о количестве найденных результатов на сообщение вида: “Найдено: **n** (показано: **m**)” в случае если если **n** и **m** различны.
- выделите искомую строку в результатах жирным шрифтом.