

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ  
ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

**Факультет программной инженерии и компьютерной техники**

**Кафедра вычислительной техники**

**КУРСОВАЯ РАБОТА «АВТОБУСНЫЙ ПАРК»  
ПО ДИСЦИПЛИНЕ «СИСТЕМЫ БАЗ ДАННЫХ»**

Выполнил: Милешин Андрей Александрович

Группа: Р3310

Преподаватель: Беликов Павел Андреевич

Санкт-Петербург

2017

## **Цель курсовой работы:**

Получение навыков и знаний, необходимых для создания объектно-реляционной базы данных и для создания базы данных, с использованием документо-ориентированной СУБД в произвольной предметной области. Применение этих навыков и знаний для реализации БД по выбранной исполнителями предметной области.

## **Предметная область**

В качестве предметной области был выбран автобусный парк. Общественный транспорт - важная часть городского устройства. Ввиду большой загруженности дорог, в современном мире, важно поддерживать и развивать общественный транспорт. Удобная система хранения информации позволит оптимизировать процесс составления расписаний смен, ремонта и т. п., что благоприятно повлияет на общественный транспорт.

Помимо непосредственно автобусов, важной частью автобусного парка являются работники. Кондукторы, водители и механики объединяются в бригады, по 2 водителя, 2 кондуктора и одного механика. За каждой бригадой, закреплен свой автобус, который она обслуживает. Каждый автобус, в свою очередь, закреплен за определенным маршрутом. Собственно, каждый автобус в парке, характеризуется следующими параметрами:

- Номер Автобуса
- Гос. номер
- Тип
- Бригада
- Маршрут
- Дата Последнего ТО

Гос.Номер уникален, но может возникнуть ситуация, когда он сменится. Поэтому каждый автобус обладает своим уникальным номером.

Для удобства, автобусы разделяются на типы, которые характеризуется следующими параметрами:

- Номер типа
- Марка
- Модель
- Число мест
- Характеристики

Так-же необходимо хранить информацию о работниках автобусного парка. Для этого в анкету работников заносятся следующие данные:

- Табельный номер
- Имя
- Дата рождения
- Должность

- Характеристики

У каждого работника, существует свой уникальный табельный номер. На родителях лежит ответственность за безопасность людей, поэтому стоит хранить о них дополнительную информацию:

- Номер водителя
- Табельный номер
- Дата мед. осмотра
- Характеристика водителя
- Нарушения

Водительская лицензия - очень важная вещь. Поэтому, необходимо хранить информацию о его текущей и прошлых лицензиях:

- Номер записи
- Номер водителя
- Номер лицензии
- Дата выдачи
- Дата окончания

Так-же необходимо хранить информацию о разных типах должностей:

- Номер должности
- Название
- Аббревиатура

Как уже было сказано, люди будут работать в бригадах. У каждой бригады будет название и эмблема:

- Номер бригады
- Название
- Эмблема

Одна из самых главных частей нашей предметной области - это маршруты. Им необходимо присвоить номера для однозначного отличия маршрутов.

- Номер маршрута
- Начальная остановка
- Конечная остановка
- Список остановок

Каждый имеет право на отдых, но чтобы система функционировала корректно, необходим график работы:

- Номер записи

- Номер бригады
- Начало смены
- Окончание смены

Такой график позволит оптимально распределить нагрузку для работников.

В любой технике может совершиться поломка и автобусы не исключение. Поэтому стоит хранить информацию о всех поломках:

- Номер поломки
- Номер автобуса
- Дата поломки
- Описание

и ремонтах:

- id ремонта
- Номер поломки
- Дата начала ремонта
- Дата окончания ремонта
- Табельный номер мастера
- Заключение

## **Модель БД части 1**

[illegible]

## Примеры CRUD кода первой части.

```
private static void createMethod(Integer n) throws IllegalArgumentException, ParseException, SQLException {
    Object entity=null;
    int maxId=0;
    hql = "SELECT f FROM "+tables.get(n)+"Entity f";
    TypedQuery<Object[]> query = entityManager.createQuery(hql,Object[].class);
    List<Object[]> results = query.getResultList();
    for(Object someObject: results)
        for (Field field : someObject.getClass().getDeclaredFields()) {
            field.setAccessible(true); // You might want to set modifier to public first.
            Object value = field.get(someObject);
            if(field.getName().equals("id"))
                if(maxId< ((Integer) value))
                    maxId=((Integer) value);
        }
    int len = maxId;
    len++;
    switch (n) {
        case 1:
            entity = new PositionsEntity(true,len);
            break;
        case 2:
            entity = new PeopleEntity(true,len);
            break;
        case 3:
            entity = new DriversEntity(true,len);
            break;
    }
}
```

```

        case 4:
            entity = new BrigadesEntity(true,len);
            break;
        case 5:
            entity = new RotesEntity(true,len);
            break;
        case 6:
            entity = new SheduleEntity(true,len);
            break;
        case 7:
            entity = new BusesEntity(true,len);
            break;
        case 8:
            entity = new BreakdownEntity(true,len);
            break;
        case 9:
            entity = new RepairsEntity(true,len);
            break;
        case 10:
            entity = new DrivingLicenceEntity(true,len);
            break;
        case 11:
            entity = new TypesOfBusesEntity(true,len);
            break;

        default:
            entity = null;
    }
    if (entity != null) {
        try {
            entityManager.getTransaction().begin();
            entityManager.persist(entity);
            entityManager.getTransaction().commit();
        }
        catch ( Exception ex) {
            System.out.println("EXCEPTION: " + ex.getMessage());
            entityManager.getTransaction().rollback();
        }
    }
}

private static void readMethod(Integer n) throws IllegalAccessException, SQLException {
    hql = "SELECT f FROM "+tables.get(n)+"Entity f";
    TypedQuery<Object[]> query = entityManager.createQuery(hql,Object[].class);
    List<Object[]> results = query.getResultList();
    for(Object someObject: results) {
        for (Field field : someObject.getClass().getDeclaredFields()) {
            field.setAccessible(true); // You might want to set modifier to public first.
            Object value = field.get(someObject);
            if (value != null) {
                System.out.println(field.getName() + "=" + value);
            }
        }
    }
    System.out.println();
}

private static void updateMethod(Integer n) throws IllegalAccessException, ParseException {
    int pk;

```

```

System.out.println("Введите id");
while(true) {
    if (sc.hasNextInt()) {
        pk = sc.nextInt();
        break;
    }
    else{
        System.out.println("Введите корректный id");
    }
}
}

```

## Схема БД части 2

### Buses.js

```

var schema = mongoose.Schema({
    number_plate: {type: String, required: true, unique: true},
    vehicle_inspection_date: {type: Date, required: true},
    working_status: {type: Boolean, required: true},
    types_of_buses: {
        brand: {type: String, required: false},
        model: {type: String, required: false},
    },
    seats: {type: Number, required: true},
    routes: {
        start: {type: String, required: false},
        stop: {type: String, required: false},
        list_of_station: {type: [String], required: false}
    }
});

```

### People.js

```

var schema = mongoose.Schema({
    id : {type: Number, unique: true, required: true},
    surname: {type: String, required: true},
    name: {type: String, required: true},
    middle_name: {type: String, required: true},
    date_of_birth: {type: Date, required: true},
    position: {
        name: {type: String, required: true},
        abb: {type: String, required: true}
    },
    drivers: {
        medical_check_up: {type: String, required: false},
        id_licence : {type: Number, required: false},
        date_of_issuance: {type: Date, required: false},
        valid_until: {type: Date, required: false},
    }
});

```

### brigades.js

```

var schema = mongoose.Schema({
    name: {type: String, required: true, unique: true},
    emblem: {type: Buffer, required: false},
    people: [{

```

```

    id : {type: Number, unique: true, required: true},
    surname: {type: String, required: true},
    name: {type: String, required: true},
    middle_name: {type: String, required: true},
  }
});

```

### Пример CRUD кода 2 части

```

'add': function(splittedinput) {
    var schema = getSchemaByName(splittedinput[1]);
    if(schema == null) return;
    model = fillFields(schema, null);
    console.log(model);
    model.save(function(err){
        if(err) return console.log(err);
    });
},
'read': function(splittedinput) {
    var schema = getSchemaByName(splittedinput[1]);
    if(schema == null) return;
    schema.model.find({}).exec(function(err, docs){
        if(err) return console.log(err);
        console.log(docs)
    });
},
'findById': function(splittedinput) {
    var schema = getSchemaByName(splittedinput[1]);
    if(schema == null) return;
    var id = rl.question(" _id = ");
    schema.model.findById(id, function(err, docs){
        if(err) return console.log(err);
        console.log(docs)
    });
},

```

### Вывод:

В ходе выполнения курсовой работы были получены и применены на практике навыки и знания, для создания базы данных, с использованием документо-ориентированной СУБД MongoDB в выбранной предметной области “автобусный парк”. Полученные знания могут быть использованы в дальнейшем изучении дисциплины и работе в этой области.