

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №3**  
**по дисциплине «Базы данных»**  
**Тема: Реализация базы данных с использованием ORM**

Студент гр. 1304

Шаврин А.П.

Преподаватель

Заславский М.М

Санкт-Петербург

2023

## **Цель работы.**

Реализация базы данных с использованием ORM Sequelize и изучение основ работы с Sequelize

## **Задание.**

### **Вариант 4.**

Пусть требуется создать программную систему, предназначенную для организаторов выставки собак. Она должна обеспечивать хранение сведений о собаках - участниках выставки и экспертах. Для каждой собаки в БД должны храниться сведения, о том, к какому клубу она относится, кличка, порода и возраст, сведения о родословной (номер документа, клички родителей), дата последней прививки, фамилия, имя, отчество и паспортные данные хозяина. На каждый клуб отводится участок номеров, под которыми будут выступать участники выставки. Сведения об эксперте должны включать фамилию и имя, номер ринга, который он обслуживает; клуб, название клуба, в котором он состоит. Каждый ринг могут обслуживать несколько экспертов. Каждая порода собак выступает на своем ринге, но на одном и том же ринге в разное время могут выступать разные породы. Итогом выставки является определение медалистов по каждой породе. Организатор выставки должен иметь возможность добавить в базу нового участника или нового эксперта, снять эксперта с судейства, заменив его другим, отстранить собаку от участия в выставке. Организатору выставки могут потребоваться следующие сведения;

- На каком ринге выступает заданный хозяин со своей собакой?
- Какими породами представлен заданный клуб?
- Какие медали и сколько заслужены клубом?
- Какие эксперты обслуживают породу?
- Количество участников по каждой породе?

Необходимо развернуть Sequelize на своем ПК и выполнить следующие задачи:

- Описать в виде моделей Sequelize таблицы из 1-й лабораторной работы
- Написать скрипт заполнения тестовыми данными: 5-10 строк на каждую таблицу, обязательно наличие связи между ними, данные приближены к реальности.
- Написать запросы к БД, отвечающие на вопросы из 1-й лабораторной работы с использованием ORM. Вывести результаты в консоль (или иной человеко-читабельный вывод)
- Запустить в репозиторий исходный код проекта, соблюсти .gitignore, убрать исходную базу из проекта (или иные нагенерированные данные бд если они есть).
- Описать процесс запуска: команды, зависимости
- В отчете описать цель, текст задания в соответствии с вариантом, выбранную ORM, инструкцию по запуску, скриншоты (код) моделей ORM, скриншоты на каждый запрос (или группу запросов) на изменение/таблицы с выводом результатов (ответ), ссылку на PR в приложении, вывод

### Выполнение работы.

1. Сперва были настроены файлы package.json и tsconfig.json для корректной работы.
2. Затем для сокрытия данных о БД были написаны файлы .env и env.d.ts с типами переменных хранящимися в env файле для корректной работы в ts.

```

1 declare global {
2   namespace NodeJS {
3     interface ProcessEnv {
4       DB_NAME: string,
5       DB_USER: string,
6       DB_PASSWORD: string,
7       DB_HOST: string,
8       DB_DIALECT: dialect,
9       DB_PORT: number
10    }
11  }
12 }
13
14 export {}

```

Рис 1. Env.d.ts

3. После был написан файл db.ts в котором создается объект Sequelize.

```
1 import { Sequelize } from 'sequelize-typescript';
2 import * as dotenv from 'dotenv';
3
4
5 dotenv.config();
6 export const db: Sequelize = new Sequelize(
7   {
8     database: process.env.DB_NAME,
9     username: process.env.DB_USER,
10    password: process.env.DB_PASSWORD,
11    dialect: process.env.DB_DIALECT,
12    host: process.env.DB_HOST,
13    port: process.env.DB_PORT,
14  }
15 )
```

Рис 2. Db.ts

4. Затем в файле models.ts были описаны все модели БД и связи между ними.

```
14 @Table({ timestamps: false })
15 export class Owner extends Model {
16   @HasMany(() => Dog)
17   declare dogs: Dog[];
18
19   @PrimaryKey
20   @AllowNull(false)
21   @Column(DataType.STRING(11))
22   declare passport: string;
23
24   @AllowNull(false)
25   @Column(DataType.STRING(255))
26   declare surname: string;
27
28   @AllowNull(false)
29   @Column(DataType.STRING(255))
30   declare name: string;
31
32   @AllowNull(false)
33   @Column(DataType.STRING(255))
34   declare patronymic: string;
35 }
36
```

Рис 3. Owner

```
37 @Table({ timestamps: false })
38 export class Ring extends Model {
39   @HasMany(() => Breed)
40   declare breeds: Breed[];
41
42   @HasMany(() => Expert)
43   declare experts: Expert[];
44
45   @PrimaryKey
46   @AutoIncrement
47   @Column(DataType.INTEGER)
48   declare ring_number: number;
49 }
50
```

Рис 4. Ring

```

51 @Table({ timestamps: false })
52 export class Breed extends Model {
53     @BelongsTo(() => Ring)
54     declare ring: Ring;
55
56     @HasMany(() => Dog)
57     declare dogs: Dog[];
58
59     @PrimaryKey
60     @AllowNull(false)
61     @Column(DataType.STRING(255))
62     declare breed_name: string;
63
64     @ForeignKey(() => Ring)
65     @Column(DataType.INTEGER)
66     declare ring_number: number;
67 }

```

Рис 5. Breed

```

69 @Table({ timestamps: false })
70 export class Dog extends Model {
71     @BelongsTo(() => Owner)
72     declare owner: Owner;
73
74     @BelongsTo(() => Breed)
75     declare breed: Breed;
76
77     @HasMany(() => DogExpertEstimate)
78     declare dog_expert_estimates: DogExpertEstimate[];
79
80     @PrimaryKey
81     @AutoIncrement
82     @Column(DataType.INTEGER)
83     declare dog_number: number;
84
85     @ForeignKey(() => Owner)
86     @Column(DataType.STRING(11))
87     declare owner_passport: string;
88
89     @ForeignKey(() => Breed)
90     @Column(DataType.STRING(255))
91     declare breed_name: string;
92 }

```

```

93     @AllowNull(false)
94     @Column(DataType.INTEGER)
95     declare pedigree_document_number: number;
96
97     @AllowNull(false)
98     @Column(DataType.STRING(255))
99     declare mother_nickname: string;
100
101     @AllowNull(false)
102     @Column(DataType.STRING(255))
103     declare father_nickname: string;
104
105     @AllowNull(false)
106     @Column(DataType.STRING(255))
107     declare nickname: string;
108
109     @AllowNull(false)
110     @Column(DataType.INTEGER)
111     declare age: number;
112
113     @AllowNull(false)
114     @Column(DataType.DATE)
115     declare vaccination_date: Date;
116 }
117

```

Рис 6. Dog

```

118 @Table({ timestamps: false })
119 export class Club extends Model {
120     @HasMany(() => Expert)
121     declare experts: Expert[];
122
123     @HasMany(() => ClubNumber)
124     declare club_numbers: ClubNumber[];
125
126     @PrimaryKey
127     @AutoIncrement
128     @Column(DataType.INTEGER)
129     declare club_id: number;
130
131     @AllowNull(false)
132     @Column(DataType.STRING(255))
133     declare club_name: string;
134
135     @AllowNull(false)
136     @Column(DataType.INTEGER)
137     declare participants_number: number;
138 }
139

```

Рис 7. Club

```

140 @Table({ timestamps: false })
141 export class Expert extends Model {
142     @BelongsTo(() => Ring)
143     declare ring: Ring;
144
145     @BelongsTo(() => Club)
146     declare club: Club;
147
148     @HasMany(() => DogExpertEstimate)
149     declare dogs_expert_estimate: DogExpertEstimate[];
150
151     @PrimaryKey
152     @AutoIncrement
153     @Column(DataType.INTEGER)
154     declare expert_id: number;
155
156     @AllowNull(false)
157     @Column(DataType.STRING(255))
158     declare surname: string;
159
160     @AllowNull(false)
161     @Column(DataType.STRING(255))
162     declare name: string;
163
164     @ForeignKey(() => Ring)
165     @Column(DataType.INTEGER)
166     declare ring_number: number;
167
168     @ForeignKey(() => Club)
169     @Column(DataType.INTEGER)
170     declare club_id: number;
171 }

```

Рис 8. Expert

```

172
173 @Table({ timestamps: false })
174 export class ClubNumber extends Model {
175     @BelongsTo(() => Club)
176     declare club: Club;
177
178     @BelongsTo(() => Dog)
179     declare dog: Dog;
180
181     @PrimaryKey
182     @ForeignKey(() => Club)
183     @Column(DataType.INTEGER)
184     declare club_id: number;
185
186     @PrimaryKey
187     @ForeignKey(() => Dog)
188     @Column(DataType.INTEGER)
189     declare dog_number: number;
190 }

```

Рис 9. ClubNumber



```

192 @Table({ timestamps: false })
193 export class DogExpertEstimate extends Model {
194   @BelongsTo(() => Dog)
195   declare dog: Dog;
196
197   @BelongsTo(() => Expert)
198   declare expert: Expert;
199
200   @PrimaryKey
201   @ForeignKey(() => Dog)
202   @Column(DataType.INTEGER)
203   declare dog_number: number;
204
205   @PrimaryKey
206   @ForeignKey(() => Expert)
207   @Column(DataType.INTEGER)
208   declare expert_id: number;
209
210   @AllowNull(false)
211   @Column(DataType.INTEGER)
212   declare estimate: number;
213 }

```

Рис 10. DogExpertEstimate

5. Затем был создан файл fill\_db.ts, который заполняет таблицы.

```

4 export async function fill_db(){
5   // fill Rings
6   for (let i = 1; i <= 5; i++){
7     await models.Ring.create({});
8   }
9   //console.log(models.Ring.findAll())
10
11   //fill Breeds
12   let breeds = [
13     ['Немецкая овчарка', 1],
14     ['Доберман', 1],
15     ['Померанский шпиц', 2],
16     ['Йоркширский терьер', 3],
17     ['Бультерьер', 4],
18     ['Пудель', 5]
19   ];
20   await models.Breed.bulkCreate(
21     breeds.map((row) => ({ breed_name: row[0], ring_number: row[1] })),
22     { returning: false }
23   );
24   //console.log(await models.Breed.findAll());
25
26   //fill clubs
27   let clubs = [
28     ['Фаворит', 4],
29     ['Оскар', 4],
30     ['Статус', 4],
31     ['Звезда', 3],
32     ['Победа', 2]
33   ];
34   await models.Club.bulkCreate(
35     clubs.map((row) => ({ club_name: row[0], participants_number: row[1] })),
36     { returning: false }
37   );

```



```

40 //fill experts
41 let experts = [
42   [1, 1, 'Кардаш', 'Ярослав'],
43   [2, 1, 'Шаврин', 'Алексей'],
44   [2, 1, 'Дешура', 'Дмитрий'],
45   [3, 3, 'Ефремов', 'Артем'],
46   [4, 4, 'Есенина', 'Анастасия'],
47   [5, 5, 'Насекина', 'Алена'],
48   [5, 2, 'Лепс', 'Григорий'],
49 ];
50 await models.Expert.bulkCreate(
51   experts.map((row) => ({ club_id: row[0], ring_number: row[1], surname: row[2], name: row[3] })),
52   { returning: false }
53 );
54 //console.log(await models.Expert.findAll());
55
56 //fill owners
57 let owners = [
58   ['1111 222222', 'Потолов', 'Олег', 'Алексеевич'],
59   ['2222 333333', 'Зубов', 'Александр', 'Андреевич'],
60   ['3333 444444', 'Башкирев', 'Алексей', 'Владимирович'],
61   ['4444 555555', 'Иванов', 'Иван', 'Иванович'],
62   ['5555 666666', 'Добрицкий', 'Ярослав', 'Валерьевич'],
63   ['6666 777777', 'Игнатьев', 'Владимир', 'Владимирович'],
64   ['7777 888888', 'Бочаров', 'Никита', 'Игоревич'],
65   ['8888 999999', 'Данилов', 'Дмитрий', 'Семенович']
66 ];
67 await models.Owner.bulkCreate(
68   owners.map((row) => ({ passport: row[0], surname: row[1], name: row[2], patronymic: row[3] })),
69   { returning: false }
70 );

```

```

73 //fill dogs
74 let dogs = [
75   ['1111 222222', 'Немецкая овчарка', 1111111, 'Элис', 'Арчи', 'Рекс', 5, '2023-10-01'],
76   ['1111 222222', 'Доберман', 1111112, 'Меган', 'Дизель', 'Рокс', 7, '2023-09-27'],
77   ['2222 333333', 'Немецкая овчарка', 2222221, 'Меган', 'Дизель', 'Рокс', 8, '2023-09-06'],
78   ['2222 333333', 'Доберман', 2222222, 'Элис', 'Арчи', 'Рекс', 6, '2023-09-06'],
79   ['3333 444444', 'Немецкая овчарка', 3333331, 'Элли', 'Оливер', 'Раймонд', 7, '2023-09-12'],
80   ['3333 444444', 'Доберман', 3333332, 'Эмма', 'Руди', 'Алекс', 9, '2023-09-23'],
81   ['4444 555555', 'Немецкая овчарка', 4444441, 'Герда', 'Пит', 'Балу', 5, '2023-09-29'],
82   ['4444 555555', 'Пудель', 4444442, 'Витта', 'Мэверик', 'Гектор', 5, '2023-09-22'],
83   ['5555 666666', 'Бультерьер', 5555551, 'Грейт', 'Веня', 'Барон', 8, '2023-09-15'],
84   ['5555 666666', 'Пудель', 5555552, 'Инга', 'Ефим', 'Герой', 4, '2023-09-16'],
85   ['6666 777777', 'Бультерьер', 6666661, 'Лейла', 'Джанго', 'Кекс', 4, '2023-09-03'],
86   ['6666 777777', 'Пудель', 6666662, 'Скай', 'Зевс', 'Моджо', 7, '2023-09-08'],
87   ['7777 888888', 'Померанский шпиц', 7777771, 'Руни', 'Мич', 'Зефирка', 4, '2023-10-01'],
88   ['7777 888888', 'Йоркширский терьер', 7777772, 'Барби', 'Ник', 'Пряник', 4, '2023-09-28'],
89   ['8888 999999', 'Померанский шпиц', 8888881, 'Джесси', 'Зик', 'Ерошка', 6, '2023-09-20'],
90   ['8888 999999', 'Йоркширский терьер', 8888882, 'Кристи', 'Сириус', 'Цезарь', 6, '2023-09-17'],
91   ['8888 999999', 'Йоркширский терьер', 8888883, 'Фокс', 'Арчи', 'Тимка', 7, '2023-09-17']
92 ];
93 await models.Dog.bulkCreate(
94   dogs.map((row) => ({ owner_passport: row[0], breed_name: row[1], pedigree_document_number: row[2],
95     mother_nickname: row[3], father_nickname: row[4], nickname: row[5],
96     age: row[6], vaccination_date: row[7] })),
97   { returning: false }
98 );

```

```

100
101 //fill club numbers
102 let club_numbers = [
103   [1, 1], [2, 1],
104   [3, 1], [4, 1],
105   [5, 2], [6, 2],
106   [7, 2], [8, 2],
107   [9, 3], [10, 3],
108   [11, 3], [12, 3],
109   [13, 5], [14, 5],
110   [15, 4], [16, 4],
111   [17, 4]
112 ];
113 await models.ClubNumber.bulkCreate(
114   club_numbers.map((row) => ({ dog_number: row[0], club_id: row[1] })),
115   { returning: false }
116 );
117 //console.log(await models.ClubNumber.findAll());
118
119 //fill dogs experts estimates
120 let dogs_experts_estimates = [
121   [1, 1, 9], [3, 1, 6], [5, 1, 8], [7, 1, 7],
122   [1, 2, 10], [3, 2, 7], [5, 2, 9], [7, 2, 8],
123   [2, 3, 7], [4, 3, 10], [6, 3, 8],
124   [8, 6, 10], [10, 6, 10], [12, 6, 8],
125   [9, 5, 8], [11, 5, 9],
126   [14, 4, 5], [16, 4, 7], [17, 4, 8],
127   [13, 7, 10], [15, 7, 10]
128 ];
129 await models.DogExpertEstimate.bulkCreate(
130   dogs_experts_estimates.map((row) => ({ dog_number: row[0], expert_id: row[1], estimate: row[2] })),
131   { returning: false }
132 );

```

Рис 11. Fill\_db.ts

6. После создан файл запросов requests.ts.

- На каком ринге выступает заданный хозяин со своей собакой?

```

6 export async function request_1() {
7   /*
8   SELECT breed.ring_number, dog.dog_number, owner.surname, owner.name, owner.patronymic
9   FROM owner
10   INNER JOIN dog ON owner.passport = dog.owner_passport
11   INNER JOIN breed USING(breed_name)
12   ORDER BY breed.ring_number, dog.dog_number, owner.surname, owner.name, owner.patronymic;
13   */
14   await models.Dog.findAll(
15     {
16       attributes: ['dog_number'],
17       include: [
18         {
19           model: models.Owner,
20           required: true,
21           attributes: ['surname', 'name']
22         },
23         {
24           model: models.Breed,
25           required: true,
26           attributes: ['ring_number'],
27         }
28       ],
29       order: [
30         ['dog_number', 'ASC'],
31       ]
32     }
33   ).then((result) => {
34     console.log('Request 1:');
35     //console.log(result)
36     for (let dog of result) {
37       //console.log(dog)
38       console.log(dog.breed.dataValues.ring_number, dog.dataValues.dog_number, dog.owner.dataValues.surname, dog.owner.dataValues.name)
39     }
40   });
41 }
42

```

Рис 12. Request\_1.

```

Executing (default): SELECT "Dog"."dog_number", "owner"."passport" AS "owner.passport", "owner"."surname" AS "owner.surname", "owner"."name" AS "owner.name", "breed"."breed_name" AS "breed.breed_name", "breed"."ring_number" AS "breed.ring_number" FROM "Dogs" AS "Dog" INNER JOIN "Owners" AS "owner" ON "Dog"."owner_passport" = "owner"."passport" INNER JOIN "Breeds" AS "breed" ON "Dog"."breed_name" = "breed"."breed_name" ORDER BY "Dog"."dog_number" ASC;
Request 1:
1 1 Потолов Олег
1 2 Потолов Олег
1 3 Зубов Александр
1 4 Зубов Александр
1 5 Башкирев Алексей
1 6 Башкирев Алексей
1 7 Иванов Иван
5 8 Иванов Иван
4 9 Добрицкий Ярослав
5 10 Добрицкий Ярослав
4 11 Игнатьев Владимир
5 12 Игнатьев Владимир
2 13 Бочаров Никита
3 14 Бочаров Никита
2 15 Данилов Дмитрий
3 16 Данилов Дмитрий
3 17 Данилов Дмитрий

```

Рис 13. Request\_1 результат

- Какими породами представлен заданный клуб?

Поскольку group в findAll в Sequelize работает плохо, используется запрос query

```

export async function request_2(){
  /*
  SELECT club.club_id, dog.breed_name
  FROM club
  INNER JOIN club_numbers USING(club_id)
  INNER JOIN dog USING(dog_number)
  GROUP BY club.club_id, dog.breed_name
  ORDER BY club.club_id, dog.breed_name;
  */

  await db.query(
    `SELECT club.club_id, dog.breed_name
    FROM "Clubs" AS club
    INNER JOIN "ClubNumbers" AS club_numbers USING(club_id)
    INNER JOIN "Dogs" AS dog USING(dog_number)
    GROUP BY club.club_id, dog.breed_name
    ORDER BY club.club_id, dog.breed_name;`,
    { type: QueryTypes.SELECT }
  ).then((result) => {
    console.log('Request 2:');
    console.log(result);
  })
}

```

Рис 14. Request\_2

```

Executing (default): SELECT club.club_id, dog.breed_name
                      FROM "Clubs" AS club
                      INNER JOIN "ClubNumbers" AS club_numbers USING(club_id)
                      INNER JOIN "Dogs" AS dog USING(dog_number)
                      GROUP BY club.club_id, dog.breed_name
                      ORDER BY club.club_id, dog.breed_name;
Request 2:
[
  { club_id: 1, breed_name: 'Доберман' },
  { club_id: 1, breed_name: 'Немецкая овчарка' },
  { club_id: 2, breed_name: 'Доберман' },
  { club_id: 2, breed_name: 'Немецкая овчарка' },
  { club_id: 2, breed_name: 'Пудель' },
  { club_id: 3, breed_name: 'Бультерьер' },
  { club_id: 3, breed_name: 'Пудель' },
  { club_id: 4, breed_name: 'Йоркширский терьер' },
  { club_id: 4, breed_name: 'Померанский шпиц' },
  { club_id: 5, breed_name: 'Йоркширский терьер' },
  { club_id: 5, breed_name: 'Померанский шпиц' }
]

```

Рис 15. Request\_2 результат

- Какие медали и сколько заслужены клубом?

Поскольку group в findAll в Sequelize работает плохо и не поддерживает with, используется запрос query

```

96 export async function request_3() {
97   /*
98   WITH dog_places AS (
99     SELECT dog.breed_name, dog.dog_number, ROUND(AVG(dog_expert_estimate.estimate), 2) AS Estimate,
100     DENSE_RANK() OVER (PARTITION BY dog.breed_name ORDER BY ROUND(AVG(dog_expert_estimate.estimate), 2) DESC) AS place
101     FROM dog
102     INNER JOIN dog_expert_estimate USING(dog_number)
103     GROUP BY dog.breed_name, dog.dog_number
104     ORDER BY dog.breed_name, Estimate DESC
105   )
106   SELECT club.club_id, club.club_name, dog_places.place, COUNT(dog_places.place)
107   FROM club
108   INNER JOIN club_numbers USING(club_id)
109   INNER JOIN dog_places USING(dog_number)
110   GROUP BY club.club_id, club.club_name, dog_places.place
111   ORDER BY club.club_id, dog_places.place;
112   */
113
114   await db.query(
115     `WITH dog_places AS (
116       SELECT dog.breed_name, dog.dog_number, ROUND(AVG(dog_expert_estimate.estimate), 2) AS Estimate,
117       DENSE_RANK() OVER (PARTITION BY dog.breed_name ORDER BY ROUND(AVG(dog_expert_estimate.estimate), 2) DESC) AS place
118       FROM "Dogs" AS dog
119       INNER JOIN "DogExpertEstimates" AS dog_expert_estimate USING(dog_number)
120       GROUP BY dog.breed_name, dog.dog_number
121       ORDER BY dog.breed_name, Estimate DESC
122     )
123     SELECT club.club_id, club.club_name, dog_places.place, COUNT(dog_places.place)
124     FROM "Clubs" AS club
125     INNER JOIN "ClubNumbers" AS club_numbers USING(club_id)
126     INNER JOIN dog_places USING(dog_number)
127     GROUP BY club.club_id, club.club_name, dog_places.place
128     ORDER BY club.club_id, dog_places.place;`
129     , { type: QueryTypes.SELECT }
130   ).then((result) => {
131     console.log('Request 3:');
132     console.log(result);
133   });
134 }

```

Рис 16. Request\_3



```

Executing (default): WITH dog_places AS (
  SELECT dog.breed_name, dog.dog_number, ROUND(AVG(dog_expert_estimate.estimate), 2) AS Estimate,
    DENSE_RANK() OVER (PARTITION BY dog.breed_name ORDER BY ROUND(AVG(dog_expert_estimate.estimate), 2) DESC) AS place
  FROM "Dogs" AS dog
  INNER JOIN "DogExpertEstimates" AS dog_expert_estimate USING(dog_number)
  GROUP BY dog.breed_name, dog.dog_number
  ORDER BY dog.breed_name, Estimate DESC
)
SELECT club.club_id, club.club_name, dog_places.place, COUNT(dog_places.place)
FROM "Clubs" AS club
  INNER JOIN "ClubNumbers" AS club_numbers USING(club_id)
  INNER JOIN dog_places USING(dog_number)
GROUP BY club.club_id, club.club_name, dog_places.place
ORDER BY club.club_id, dog_places.place;

Request 3:
[
  { club_id: 1, club_name: 'Фаворит', place: '1', count: '2' },
  { club_id: 1, club_name: 'Фаворит', place: '3', count: '1' },
  { club_id: 1, club_name: 'Фаворит', place: '4', count: '1' },
  { club_id: 2, club_name: 'Оскар', place: '1', count: '1' },
  { club_id: 2, club_name: 'Оскар', place: '2', count: '2' },
  { club_id: 2, club_name: 'Оскар', place: '3', count: '1' },
  { club_id: 3, club_name: 'Статус', place: '1', count: '2' },
  { club_id: 3, club_name: 'Статус', place: '2', count: '2' },
  { club_id: 4, club_name: 'Звезда', place: '1', count: '2' },
  { club_id: 4, club_name: 'Звезда', place: '2', count: '1' },
  { club_id: 5, club_name: 'Победа', place: '1', count: '1' },
  { club_id: 5, club_name: 'Победа', place: '3', count: '1' }
]

```

Рис 17. Request\_3 результат

- Какие эксперты обслуживают породу?

Поскольку group в findAll в Sequelize работает плохо, используется запрос query

```

137 export async function request_4() {
138   /*
139   SELECT dog.breed_name, expert.expert_id, expert.name, expert.surname
140   FROM dog
141   INNER JOIN dog_expert_estimate USING(dog_number)
142   INNER JOIN expert USING(expert_id)
143   GROUP BY dog.breed_name, expert.expert_id
144   ORDER BY dog.breed_name;
145   */
146
147   await db.query(
148     `SELECT dog.breed_name, expert.expert_id, expert.name, expert.surname
149     FROM "Dogs" AS dog
150     INNER JOIN "DogExpertEstimates" AS dog_expert_estimate USING(dog_number)
151     INNER JOIN "Experts" AS expert USING(expert_id)
152     GROUP BY dog.breed_name, expert.expert_id
153     ORDER BY dog.breed_name;`,
154     { type: QueryTypes.SELECT }
155   ).then((result) => {
156     console.log('Request 4:');
157     console.log(result);
158   });
159   /*

```

Рис 18. Request\_4

```

Executing (default): SELECT dog.breed_name, expert.expert_id, expert.name, expert.surname
FROM "Dogs" AS dog
    INNER JOIN "DogExpertEstimates" AS dog_expert_estimate USING(dog_number)
    INNER JOIN "Experts" AS expert USING(expert_id)
GROUP BY dog.breed_name, expert.expert_id
ORDER BY dog.breed_name;

```

Request 4:

```

[
  {
    breed_name: 'Бультерьер',
    expert_id: 5,
    name: 'Анастасия',
    surname: 'Есенина'
  },
  {
    breed_name: 'Доберман',
    expert_id: 3,
    name: 'Дмитрий',
    surname: 'Дешура'
  },
  {
    breed_name: 'Йоркширский терьер',
    expert_id: 4,
    name: 'Артём',
    surname: 'Ефремов'
  },
]

```

```

{
  breed_name: 'Немецкая овчарка',
  expert_id: 1,
  name: 'Ярослав',
  surname: 'Кардаш'
},
{
  breed_name: 'Немецкая овчарка',
  expert_id: 2,
  name: 'Алексей',
  surname: 'Шаврин'
},
{
  breed_name: 'Померанский шпиц',
  expert_id: 7,
  name: 'Григорий',
  surname: 'Лепс'
},
{
  breed_name: 'Пудель',
  expert_id: 6,
  name: 'Алена',
  surname: 'Насекина'
}

```

Рис 19. Request\_4 результат

- Количество участников по каждой породе?

```

194 export async function request_5(){
195     /*
196     SELECT dog.breed_name, COUNT(*)
197     FROM dog
198     GROUP BY dog.breed_name
199     ORDER BY dog.breed_name;
200     */
201
202     await models.Dog.findAll(
203     {
204         attributes: ['breed_name', [db.fn('COUNT', db.col('dog_number')), 'count']],
205         group: ['breed_name'],
206         order: [['breed_name', 'ASC']]
207     }
208     ).then((result) => {
209         console.log('Request 5:');
210         for (let dog of result) {
211             console.log(dog.dataValues.breed_name, dog.dataValues.count)
212         }
213     })
214 }

```

Рис 20. Request\_5

```

Executing (default): SELECT "breed_name", COUNT("dog_number") AS "count" FROM "Dogs" AS "Dog" GROUP BY "breed_name" ORDER BY "Dog"."breed_name" ASC;
Request 5:
Бультерьер 2
Доберман 3
Йоркширский терьер 3
Немецкая овчарка 4
Померанский шпиц 2
Пудель 3

```

Рис 21. Request\_5 результат

7. Самым последним был написан файл index.ts в котором осуществляется подключение к БД, добавление и синхронизация моделей

```

1 import { db } from "../db.js"
2 import * as models from "../models/models.js"
3 import { fill_db } from "../fill_db.js"
4 import * as requests from "../requests.js"
5
6
7 try{
8     await db.authenticate();
9     console.log('authenticate: done')
10
11     db.addModels([models.Ring, models.Breed, models.Expert, models.Club, models.ClubNumber, models.Owner, models.Dog, models.DogExpertEstimate])
12     console.log('models add: done')
13
14     await db.sync({ force: true })
15     console.log('db sync: done')
16
17     await fill_db();
18     console.log('fill db: done')
19
20     await requests.request_1();
21     await requests.request_2();
22     await requests.request_3();
23     await requests.request_4();
24     await requests.request_5();
25     console.log('requests: done')
26 } catch(error) {
27     console.log('authenticate: error', error)
28 }

```

Рис 22. Index.ts

Разработанный программный код см. в приложении А.

Ссылка на PR см. в приложении В.



**Выводы.**

В ходе выполнения работы, реализована база данных с использованием ORM Sequelize, изучены основы работы с Sequelize.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: env.d.ts

```
declare global {
  namespace NodeJS {
    interface ProcessEnv {
      DB_NAME: string,
      DB_USER: string,
      DB_PASSWORD: string,
      DB_HOST: string,
      DB_DIALECT: dialect,
      DB_PORT: number
    }
  }
}
export {}
```

Название файла: db.ts

```
import { Sequelize } from 'sequelize-typescript';
import * as dotenv from 'dotenv';

dotenv.config();
export const db: Sequelize = new Sequelize(
  {
    database: process.env.DB_NAME,
    username: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    dialect: process.env.DB_DIALECT,
    host: process.env.DB_HOST,
    port: process.env.DB_PORT,
  }
)
```

Название файла: models.ts

```
import {
  Table,
  Column,
  Model,
  PrimaryKey,
  AllowNull,
  DataType,
  AutoIncrement,
  Default,
} from 'sequelize-typescript';
import { BelongsTo, ForeignKey, HasMany } from 'sequelize-typescript'

@Table({ timestamps: false })
export class Owner extends Model {
  @HasMany(() => Dog)
```

```

declare dogs: Dog[];

@PrimaryKey
@AllowNull(false)
@Column(DataType.STRING(11))
declare passport: string;

@AllowNull(false)
@Column(DataType.STRING(255))
declare surname: string;

@AllowNull(false)
@Column(DataType.STRING(255))
declare name: string;

@AllowNull(false)
@Column(DataType.STRING(255))
declare patronymic: string;
}

@Table({ timestamps: false })
export class Ring extends Model {
    @HasMany(() => Breed)
    declare breeds: Breed[];

    @HasMany(() => Expert)
    declare experts: Expert[];

    @PrimaryKey
    @AutoIncrement
    @Column(DataType.INTEGER)
    declare ring_number: number;
}

@Table({ timestamps: false })
export class Breed extends Model {
    @BelongsTo(() => Ring)
    declare ring: Ring;

    @HasMany(() => Dog)
    declare dogs: Dog[];

    @PrimaryKey
    @AllowNull(false)
    @Column(DataType.STRING(255))
    declare breed_name: string;

    @ForeignKey(() => Ring)
    @Column(DataType.INTEGER)
    declare ring_number: number;
}

@Table({ timestamps: false })
export class Dog extends Model {
    @BelongsTo(() => Owner)
    declare owner: Owner;

    @BelongsTo(() => Breed)

```

```

declare breed: Breed;

@HasMany(() => DogExpertEstimate)
declare dog_expert_estimates: DogExpertEstimate[];

@PrimaryKey
@AutoIncrement
@Column(DataType.INTEGER)
declare dog_number: number;

@ForeignKey(() => Owner)
@Column(DataType.STRING(11))
declare owner_passport: string;

@ForeignKey(() => Breed)
@Column(DataType.STRING(255))
declare breed_name: string;

@AllowNull(false)
@Column(DataType.INTEGER)
declare pedigree_document_number: number;

@AllowNull(false)
@Column(DataType.STRING(255))
declare mother_nickname: string;

@AllowNull(false)
@Column(DataType.STRING(255))
declare father_nickname: string;

@AllowNull(false)
@Column(DataType.STRING(255))
declare nickname: string;

@AllowNull(false)
@Column(DataType.INTEGER)
declare age: number;

@AllowNull(false)
@Column(DataType.DATE)
declare vaccination_date: Date;
}

@Table({ timestamps: false })
export class Club extends Model {
  @HasMany(() => Expert)
  declare experts: Expert[];

  @HasMany(() => ClubNumber)
  declare club_numbers: ClubNumber[];

  @PrimaryKey
  @AutoIncrement
  @Column(DataType.INTEGER)
  declare club_id: number;

  @AllowNull(false)
  @Column(DataType.STRING(255))

```

```

        declare club_name: string;

        @AllowNull(false)
        @Column(DataType.INTEGER)
        declare participants_number: number;
    }

@Table({ timestamps: false })
export class Expert extends Model {
    @BelongsTo(() => Ring)
    declare ring: Ring;

    @BelongsTo(() => Club)
    declare club: Club;

    @HasMany(() => DogExpertEstimate)
    declare dogs_expert_estimate: DogExpertEstimate[];

    @PrimaryKey
    @AutoIncrement
    @Column(DataType.INTEGER)
    declare expert_id: number;

    @AllowNull(false)
    @Column(DataType.STRING(255))
    declare surname: string;

    @AllowNull(false)
    @Column(DataType.STRING(255))
    declare name: string;

    @ForeignKey(() => Ring)
    @Column(DataType.INTEGER)
    declare ring_number: number;

    @ForeignKey(() => Club)
    @Column(DataType.INTEGER)
    declare club_id: number;
}

@Table({ timestamps: false })
export class ClubNumber extends Model {
    @BelongsTo(() => Club)
    declare club: Club;

    @BelongsTo(() => Dog)
    declare dog: Dog;

    @PrimaryKey
    @ForeignKey(() => Club)
    @Column(DataType.INTEGER)
    declare club_id: number;

    @PrimaryKey
    @ForeignKey(() => Dog)
    @Column(DataType.INTEGER)
    declare dog_number: number;
}

```

```

@Table({ timestamps: false })
export class DogExpertEstimate extends Model {
  @BelongsTo(() => Dog)
  declare dog: Dog;

  @BelongsTo(() => Expert)
  declare expert: Expert;

  @PrimaryKey
  @ForeignKey(() => Dog)
  @Column(DataType.INTEGER)
  declare dog_number: number;

  @PrimaryKey
  @ForeignKey(() => Expert)
  @Column(DataType.INTEGER)
  declare expert_id: number;

  @AllowNull(false)
  @Column(DataType.INTEGER)
  declare estimate: number;
}

```

### Название файла: fill\_db.ts

```

import * as models from "../models/models.js"

export async function fill_db() {
  // fill Rings
  for (let i = 1; i <= 5; i++) {
    await models.Ring.create({});
  }

  //fill Breeds
  let breeds = [
    ['Немецкая овчарка', 1],
    ['Доберман', 1],
    ['Померанский шпиц', 2],
    ['Йоркширский терьер', 3],
    ['Бультерьер', 4],
    ['Пудель', 5]
  ];
  await models.Breed.bulkCreate(
    breeds.map((row) => ({ breed_name: row[0], ring_number:
row[1] })),
    { returning: false }
  );

  //fill clubs
  let clubs = [
    ['Фаворит', 4],
    ['Оскар', 4],
    ['Статус', 4],
    ['Звезда', 3],
    ['Победа', 2]
  ];
}

```

```

    await models.Club.bulkCreate(
      clubs.map((row) => ({ club_name: row[0], participants_number:
row[1] })),
      { returning: false }
    );

    //fill experts
    let experts = [
      [1, 1, 'Кардаш', 'Ярослав'],
      [2, 1, 'Шаврин', 'Алексей'],
      [2, 1, 'Дешура', 'Дмитрий'],
      [3, 3, 'Ефремов', 'Артем'],
      [4, 4, 'Есенина', 'Анастасия'],
      [5, 5, 'Насекина', 'Алена'],
      [5, 2, 'Лепс', 'Григорий'],
    ];
    await models.Expert.bulkCreate(
      experts.map((row) => ({ club_id: row[0], ring_number: row[1],
surname: row[2], name: row[3] })),
      { returning: false }
    );

    //fill owners
    let owners = [
      ['1111 222222', 'Потолов', 'Олег', 'Алексеевич'],
      ['2222 333333', 'Зубов', 'Александр', 'Андреевич'],
      ['3333 444444', 'Башкирев', 'Алексей', 'Владимирович'],
      ['4444 555555', 'Иванов', 'Иван', 'Иванович'],
      ['5555 666666', 'Добрицкий', 'Ярослав', 'Валерьевич'],
      ['6666 777777', 'Игнатъев', 'Владимир', 'Владимирович'],
      ['7777 888888', 'Бочаров', 'Никита', 'Игоревич'],
      ['8888 999999', 'Данилов', 'Дмитрий', 'Семенович']
    ];
    await models.Owner.bulkCreate(
      owners.map((row) => ({ passport: row[0], surname: row[1],
name: row[2], patronymic: row[3] })),
      { returning: false }
    );

    //fill dogs
    let dogs = [
      ['1111 222222', 'Немецкая овчарка', 1111111, 'Элис', 'Арчи',
'Рекс', 5, '2023-10-01'],
      ['1111 222222', 'Доберман', 1111112, 'Меган', 'Дизель',
'Рокс', 7, '2023-09-27'],
      ['2222 333333', 'Немецкая овчарка', 2222221, 'Меган',
'Дизель', 'Рокс', 8, '2023-09-06'],
      ['2222 333333', 'Доберман', 2222222, 'Элис', 'Арчи', 'Рекс',
6, '2023-09-06'],
      ['3333 444444', 'Немецкая овчарка', 3333331, 'Элли', 'Оливер',
'Раймонд', 7, '2023-09-12'],
      ['3333 444444', 'Доберман', 3333332, 'Эмма', 'Руди', 'Алекс',
9, '2023-09-23'],
      ['4444 555555', 'Немецкая овчарка', 4444441, 'Герда', 'Пит',
'Балу', 5, '2023-09-29'],
      ['4444 555555', 'Пудель', 4444442, 'Витта', 'Мэверик',
'Гектор', 5, '2023-09-22'],

```



```

        ['5555 666666', 'Бультерьер', 5555551, 'Грейт', 'Веня',
'Барон', 8, '2023-09-15'],
        ['5555 666666', 'Пудель', 5555552, 'Инга', 'Ефим', 'Герой',
4, '2023-09-16'],
        ['6666 777777', 'Бультерьер', 6666661, 'Лейла', 'Джанго',
'Кекс', 4, '2023-09-03'],
        ['6666 777777', 'Пудель', 6666662, 'Скай', 'Зевс', 'Моджо',
7, '2023-09-08'],
        ['7777 888888', 'Померанский шпиц', 7777771, 'Руни', 'Мич',
'Зефирка', 4, '2023-10-01'],
        ['7777 888888', 'Йоркширский терьер', 7777772, 'Барби', 'Ник',
'Пряник', 4, '2023-09-28'],
        ['8888 999999', 'Померанский шпиц', 8888881, 'Джесси', 'Зик',
'Ерошка', 6, '2023-09-20'],
        ['8888 999999', 'Йоркширский терьер', 8888882, 'Кристи',
'Сириус', 'Цезарь', 6, '2023-09-17'],
        ['8888 999999', 'Йоркширский терьер', 8888883, 'Фокс', 'Арчи',
'Тимка', 7, '2023-09-17']
    ];
    await models.Dog.bulkCreate(
        dogs.map((row) => ({ owner_passport: row[0], breed_name:
row[1], pedigree_document_number: row[2],
                                mother_nickname: row[3], father_nickname:
row[4], nickname: row[5],
                                age: row[6], vaccination_date:
row[7] })),
        { returning: false }
    );

    //fill club numbers
    let club_numbers = [
        [1, 1], [2, 1],
        [3, 1], [4, 1],
        [5, 2], [6, 2],
        [7, 2], [8, 2],
        [9, 3], [10, 3],
        [11, 3], [12, 3],
        [13, 5], [14, 5],
        [15, 4], [16, 4],
        [17, 4]
    ];
    await models.ClubNumber.bulkCreate(
        club_numbers.map((row) => ({ dog_number: row[0], club_id:
row[1] })),
        { returning: false }
    );

    //fill dogs experts estimates
    let dogs_experts_estimates = [
        [1, 1, 9], [3, 1, 6], [5, 1, 8], [7, 1, 7],
        [1, 2, 10], [3, 2, 7], [5, 2, 9], [7, 2, 8],
        [2, 3, 7], [4, 3, 10], [6, 3, 8],
        [8, 6, 10], [10, 6, 10], [12, 6, 8],
        [9, 5, 8], [11, 5, 9],
        [14, 4, 5], [16, 4, 7], [17, 4, 8],
        [13, 7, 10], [15, 7, 10]
    ];
    await models.DogExpertEstimate.bulkCreate(

```

```

        dogs_experts_estimates.map((row) => ({ dog_number: row[0],
expert_id: row[1], estimate: row[2] })),
        { returning: false }
    );
}

```

### Название файла: requests.ts

```

import * as models from "../models/models.js"
import { db } from "../db.js"
import { QueryTypes } from 'sequelize'

export async function request_1() {
    /*
        SELECT    breed.ring_number,    dog.dog_number,    owner.surname,
owner.name, owner.patronymic
        FROM owner
            INNER JOIN dog ON owner.passport = dog.owner_passport
            INNER JOIN breed USING(breed_name)
        ORDER BY breed.ring_number, dog.dog_number, owner.surname,
owner.name, owner.patronymic;
    */
    await models.Dog.findAll(
        {
            attributes: ['dog_number'],

            include: [
                {
                    model: models.Owner,
                    required: true,
                    attributes: ['surname', 'name']
                },
                {
                    model: models.Breed,
                    required: true,
                    attributes: ['ring_number'],
                }
            ],
            order: [
                ['dog_number', 'ASC'],
            ]
        }
    ).then((result) => {
        console.log('Request 1:');
        for (let dog of result) {
            console.log(dog.breed.dataValues.ring_number,
dog.dataValues.dog_number, dog.owner.dataValues.surname,
dog.owner.dataValues.name)
        }
    });
}

export async function request_2(){
    /*
        SELECT club.club_id, dog.breed_name
        FROM club
    */
}

```

```

        INNER JOIN club_numbers USING(club_id)
        INNER JOIN dog USING(dog_number)
    GROUP BY club.club_id, dog.breed_name
    ORDER BY club.club_id, dog.breed_name;
    */

    await db.query(
        `SELECT club.club_id, dog.breed_name
        FROM "Clubs" AS club
            INNER JOIN "ClubNumbers" AS club_numbers
USING(club_id)
            INNER JOIN "Dogs" AS dog USING(dog_number)
        GROUP BY club.club_id, dog.breed_name
        ORDER BY club.club_id, dog.breed_name;`,
        { type: QueryTypes.SELECT }
    ).then((result) => {
        console.log('Request 2:');
        console.log(result);
    })
}

export async function request_3() {
    /*
        WITH dog_places AS (
            SELECT
                dog.breed_name,
                dog.dog_number,
                ROUND(AVG(dog_expert_estimate.estimate), 2) AS Estimate,
                DENSE_RANK() OVER (PARTITION BY dog.breed_name ORDER BY
                ROUND(AVG(dog_expert_estimate.estimate), 2) DESC) AS place
            FROM dog
                INNER JOIN dog_expert_estimate USING(dog_number)
            GROUP BY dog.breed_name, dog.dog_number
            ORDER BY dog.breed_name, Estimate DESC
        )
        SELECT
            club.club_id,
            club.club_name,
            dog_places.place,
            COUNT(dog_places.place)
        FROM club
            INNER JOIN club_numbers USING(club_id)
            INNER JOIN dog_places USING(dog_number)
        GROUP BY club.club_id, club.club_name, dog_places.place
        ORDER BY club.club_id, dog_places.place;
    */

    await db.query(
        `WITH dog_places AS (
            SELECT
                dog.breed_name,
                dog.dog_number,
                ROUND(AVG(dog_expert_estimate.estimate), 2) AS Estimate,
                DENSE_RANK() OVER (PARTITION BY dog.breed_name
                ORDER BY ROUND(AVG(dog_expert_estimate.estimate), 2) DESC) AS place
            FROM "Dogs" AS dog
                INNER JOIN "DogExpertEstimates" AS
dog_expert_estimate USING(dog_number)
            GROUP BY dog.breed_name, dog.dog_number
            ORDER BY dog.breed_name, Estimate DESC
        )
        SELECT
            club.club_id,
            club.club_name,
            dog_places.place,
            COUNT(dog_places.place)
        FROM "Clubs" AS club
    `
    )
}

```

```

            INNER JOIN "ClubNumbers" AS club_numbers
USING(club_id)
            INNER JOIN dog_places USING(dog_number)
            GROUP BY club.club_id, club.club_name, dog_places.place
            ORDER BY club.club_id, dog_places.place;`,
        { type: QueryTypes.SELECT }
    ).then((result) => {
        console.log('Request 3:');
        console.log(result);
    });
}

export async function request_4() {
    /*
        SELECT      dog.breed_name,      expert.expert_id,      expert.name,
expert.surname
        FROM dog
            INNER JOIN dog_expert_estimate USING(dog_number)
            INNER JOIN expert USING(expert_id)
        GROUP BY dog.breed_name, expert.expert_id
        ORDER BY dog.breed_name;
    */

    await db.query(
        `SELECT      dog.breed_name,      expert.expert_id,      expert.name,
expert.surname
        FROM "Dogs" AS dog
            INNER JOIN      "DogExpertEstimates"      AS
dog_expert_estimate USING(dog_number)
            INNER JOIN "Experts" AS expert USING(expert_id)
        GROUP BY dog.breed_name, expert.expert_id
        ORDER BY dog.breed_name;`,
        { type: QueryTypes.SELECT }
    ).then((result) => {
        console.log('Request 4:');
        console.log(result);
    });
}

export async function request_5(){
    /*
        SELECT dog.breed_name, COUNT(*)
        FROM dog
        GROUP BY dog.breed_name
        ORDER BY dog.breed_name;
    */

    await models.Dog.findAll(
        {
            attributes:      ['breed_name',      [db.fn('COUNT',
db.col('dog_number')), 'count']],
            group: ['breed_name'],
            order: [['breed_name', 'ASC']]
        }
    ).then((result) => {
        console.log('Request 5:');
    });
}

```

```

        for (let dog of result) {
            console.log(dog.dataValues.breed_name,
dog.dataValues.count)
        }
    })
}

```

### Название файла: index.ts

```

import { db } from "../db.js"
import * as models from "../models/models.js"
import { fill_db } from "../fill_db.js"
import * as requests from "../requests.js"

try{
    await db.authenticate();
    console.log('authenticate: done')

    db.addModels([models.Ring,      models.Breed,      models.Expert,
models.Club,      models.ClubNumber,      models.Owner,      models.Dog,
models.DogExpertEstimate])
    console.log('models add: done')

    await db.sync({ force: true })
    console.log('db sync: done')

    await fill_db();
    console.log('fill db: done')

    await requests.request_1();
    await requests.request_2();
    await requests.request_3();
    await requests.request_4();
    await requests.request_5();
    console.log('requests: done')
} catch(error) {
    console.log('authenticate: error', error)
}

```

## **ПРИЛОЖЕНИЕ Б**

### **ССЫЛКИ**

<https://github.com/moevm/sql-2023-1304/pull/57>