PROJETO DE BANCO DE DADOS
PETCARE

Mateus Valerio, 19/0035161
Maylla Krislainy, 19/0043873

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0097 - BANCOS DE DADOS (2022.1)

mateus.valerio@aluno.unb.br, maylla.krislainy@aluno.unb.br

1. INTRODUÇÃO

O nosso projeto, nomeado como **PETCARE** tem como objetivo ser um aplicativo para funcionários de uma clinica veterinária, onde ao logar no app o usuário se conecta com o banco de dados e se ele for um atendente, poderá agendar consultas dos pets, cadastrar novos clientes e gerenciar seus dados, cadastrar e gerenciar os pets dos clientes, que serão os pacientes, gerenciar o histórico financeiro do pagamento das consultas, que contem uma soma total do valor gasto, acessando o registro clinico de cada paciente, e também tem um satus de pagamento.
Caso seja um veterinário, o usuário vera uma lista com todas as consultas agendadas com ele, durante uma consulta ele pode prescrever exames, o veterinário também tera acesso a todos os dados do pet de uma determinada consulta e ao seu registro clinico, que mostraria todas as consultas feitas na clinica, alem disso ao finalizar um consulta sempre é criado um novo registro clinico, este tem acesso aos detalhes da consulta, o pet e vet envolvidos e os medicamentos, (se houver), que foram usados durante a consulta.

2. DIAGRAMA DE ENTIDADE RELACIONAMENTO

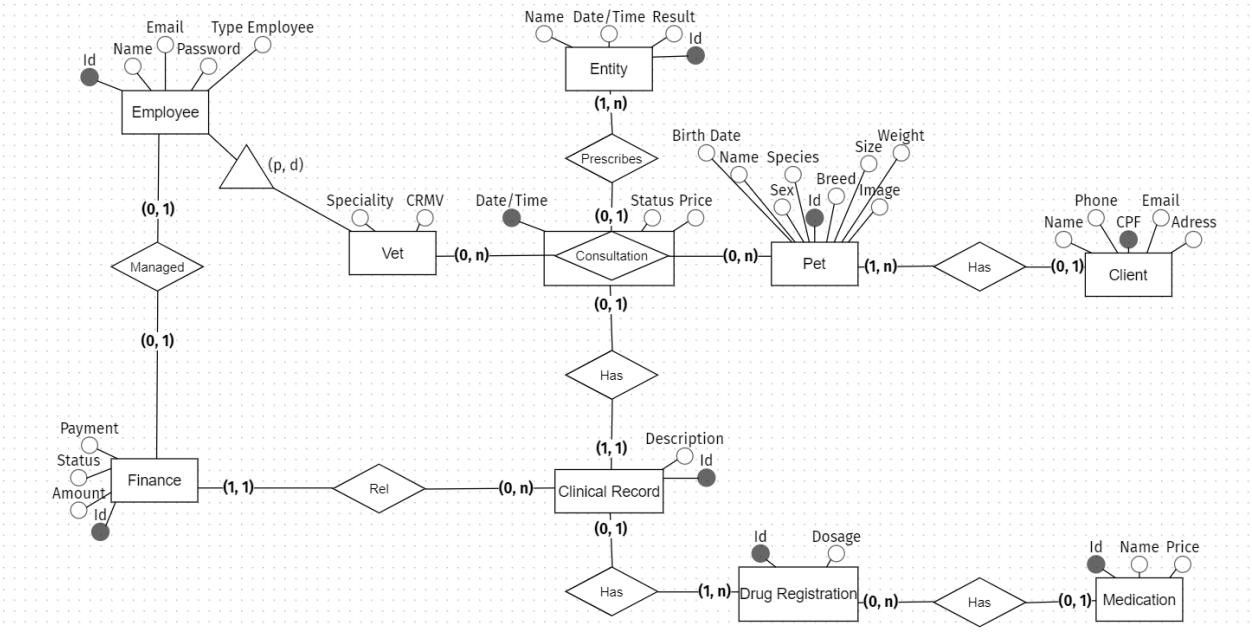


Figure 1. Modelo Entidade Relacionamento do Projeto

- Há uma especialização entre funcionário e veterinário, onde nem todo funcionário é um veterinário.
- O veterinário e o Paciente (pet) possuem um relacionamento de n pra n, sendo uma entidade associativa nomeada de consulta, onde todo veterinário pode consultar mais de um pet e um pet pode ter uma consulta com mais de um veterinário.
- um exame é prescrevido ou avaliado por um veterinário durante uma consulta.
- Todo pet tem obrigatoriamente e exclusivamente um tutor (Cliente), mas um cliente pode ser tutor de mais de um pet.
- Todo registro clinico é gerado apos a conclusão de uma consulta.
- Um registro clinico pode ou não ter um registro de medicações, mas um registro de medicação só existe se tiver um registro clinico.
- Um fechamento só existe quando também existe um registro clinico, ele é gerenciado por um funcionário que cuida da parte do financeiro da clinica.

3. MODELO RELACIONAL

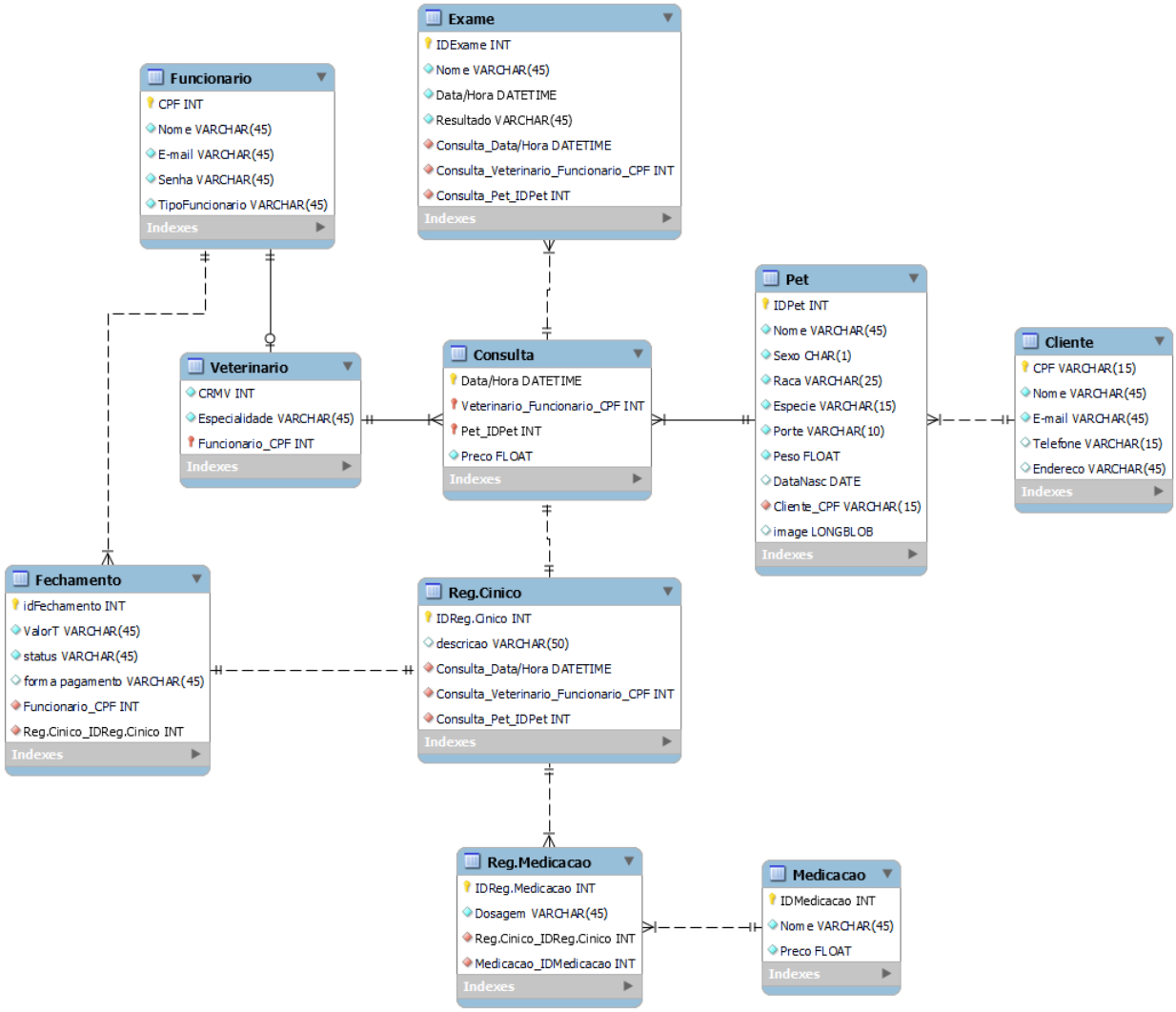


Figure 2. Modelo Relacional do Projeto

4. ÁLGEBRA RELACIONAL

a) Consultation Data

Qual o nome e dados do pet consultado? Nome e id do veterinário que o atendeu? Cpf e nome do tutor do pet? Data e status da consulta?

$x \leftarrow \sigma_{(employee.idEmployee=consultation.idVet) \wedge (vet.idVet=consultation.idVet)}(consultation \times employee \times vet)$
$\rho_{idPetCons(consultation.idPet)}$
$\rho_{vetName(employee.name)}$
$cons \leftarrow \pi_{(dPetCons,consultationDateTime,status,price,consultation.idVet,vetName,speciality)}(x)$
$\sigma_{pet.idPet=CONS.i}(pet \times cons)$

b) Finance Data

Os dados do fechamento, valor total, status, forma de pagamento, id do funcionário responsável pelo pagamento, id do registro clinico relacionado, cpf do tutor responsável pelo pet consultado.

$\rho_{cr(clinical-record)}$
$x \leftarrow \sigma_{(efinance.idClinicalRecord=cr.idClinicalRecord) \wedge (cr.idPet=pet.idPet)}(finance \times cr \times pet)$
$\pi_{(idFinance,amount,status,payment,idEmployee,finance.idClinicalRecord,consultationDateTime,cpfTutor)}(x)$

c) Clinical Record

Todos os dados do Registro Clinico, e dados do registro de medicação (se tiver) e da medicação.

$\rho_{cr(clinical-record)}$
$\rho_{dr(drug-registration)}$
$x \leftarrow \sigma_{(dr.idMedication=medication.idMedication)}(dr \times medication)$
$y \leftarrow \pi_{(idClinicalRecord,dr.idMedication,medication.name,price,dosage)}(x)$
$\sigma_{cr.idClinicalRecord=y.idClinicalRecord}(cr \times y)$

d) Vet Consultations

Todos os dados de uma consulta relacionada a um vet especifico e os dados deste vet.

$\rho_{idPetCons(idPet)}$
$\rho_{vetName(employee.name)}$
$x \leftarrow \sigma_{(employee.idEmployee=consultation.idVet) \wedge (vet.idVet=consultation.idVet)}(consultation \times employee)$
$y \leftarrow \pi_{(idPetCons,consultationDateTime,status,price,consultation.idVet,vetName,speciality)}(x)$
$\sigma_{pet.idPet=y.idPetCons}(pet \times y)$

e) Exam All Dates

Todos os dados de um exame, os dados do registro clinico relacionado a consulta na qual esta relacionada com o exame, e os dados do veterinário responsável pela consulta.

$\rho_{cr(clinical-record)}$
$\rho_{exName(exam.name)}$
$\rho_{vetName(employee.name)}$

$\rho_{consDT(exam.consultationDateTime)}$
$x \leftarrow \sigma_{(exam.idVet=vet.idVet) \wedge (vet.idVet=employee.idEmployee)}(exam \times employee \times vet)$
$y \leftarrow \pi_{(idExam,exName,examDateTime,result,consDT,exam.idVet,idPet,vetName,crmv,speciality)}(x)$
$\sigma_{(cr.consultationDateTime=y.consDT) \times (cr.idPet=y.idPet) \times (cr.idVet=y.idVet)}(cr \times y)$

5. FORMAS NORMAIS

Comanda de Atendimento	
Cpf Cliente	**ID Pet**
Nome Cliente	Nome Pet
Email Cliente	Sexo Pet
Telefone Cliente	DataNasc Pet
Endereço Cliente	Espécie Pet
Serviço Executado (Descrição)	Raça Pet
ID Comanda	Porte Pet
Data	Peso Pet
Valor	**ID Profissional**
Valor Total	Profissional

1° Forma Normal

$TAB(\underline{cpfCli, idPet, Data, idVet, idComanda}, nomeCli, emailCli, telefoneCli, endereoCli, nomePet, SexoPet, DataNascPet, EspciePet, PortePet, RaaPet, PesoPet, nomeVet, Descrio, Data, Valor, valorTotal)$

2° Forma Normal

$\underline{cpfCli} \rightarrow nomeCli, emailCli, telefoneCli, endereoCli$
$\underline{idPet} \rightarrow nomePet, sexoPet, dataNascPet, especiePet, raaPet, portePet, pesoPet$
$\underline{idVet} \rightarrow nomeVet$
$\underline{idPet, Data, idVet} \rightarrow valor, endereoCli$
$\underline{idPet, Data, idVet, idComanda} \rightarrow valorTotal, Descrio$

3° Forma Normal

Não há necessidade de modificação pois as tabelas já estão na 3° Forma Normalizada.

6. CAMADA DE MAPEAMENTO

No projeto petcare para a disciplina de Banco de Dados, foi utilizado o modelo de negócio DAO (Data Acess Object), na qual tem o objetivo de encapsular a camada de acesso a origem dos dados.

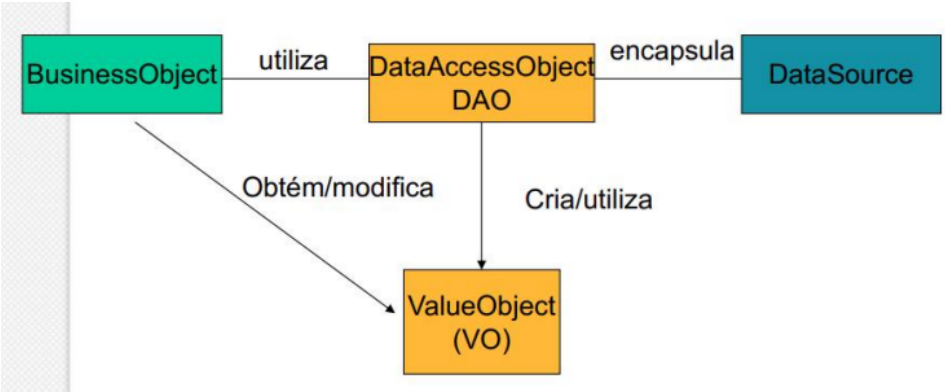


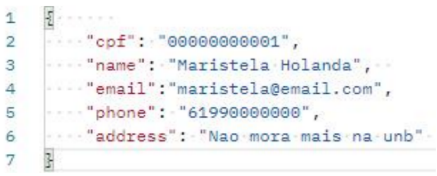
Figure 3. DAO

A imagem acima exemplifica como funciona o DAO na prática. O **BusinessObject** é o cliente que faz uso e manipula dos dados, ou seja, a nossa aplicação de fato. O DAO é então o intermediador entre o cliente e o **DataSource** (fonte de dados), sendo dessa forma, o responsável por manter uma comunicação entre o cliente e o banco de dados. Por fim, o **ValueObject** é o dados que está sendo manipulado de fato. Dessa forma, o cliente realiza modificações com esse dado, envia os dados para o DAO que então estabelece uma comunicação com o banco de dados afim de persistir essa informação. O fluxo contrário também ocorre, pois o cliente pode solicitar um dados já existente e o DAO será responsável por recuperar essa informação do banco de dados.

Em nosso aplicativo, o cliente é a aplicação em si. A imagem abaixo mostra a tela de cadastro de cliente, na qual é uma interface gráfica que facilita a apresentação para o usuário que deseja criar, atualizar, ler ou deletar um dado. Vamos chamar essa tela de camada de Apresentação.

Figure 4. APK

Quando o usuário terminar as alterações desejadas, ele irá clicar no botão confirma, na qual é responsável por chamar a lógica necessária que faz a conexão entre o cliente (O aplicativo) e o DAO para o envio do ValueObject. Em nosso caso, a comunicação é feita pelo protocolo HTTP através de requisições HTTP, que é facilitado por uma módulo do javascript chamado **axios**.



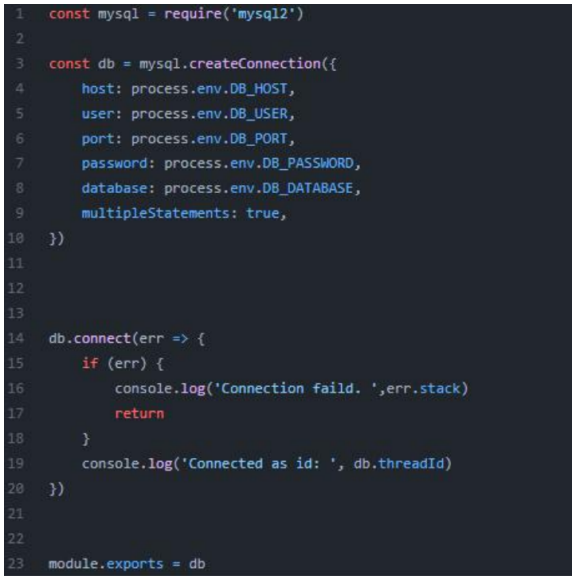
(a) Value Object

(b) Requisição Post

A nossa camada de persistência foi construída com outro módulo do node, o **express**. O express é responsável por criar uma API que recebe requisições HTTP (que o nosso aplicativo envia por meio do axios), pode ou não tratar essas requisições e então envia alguma mensagem de resposta ao cliente. A imagem a seguir mostra a abertura dessa API servidor que está escutando na porta 3003.

Figure 5. Servidor API com express

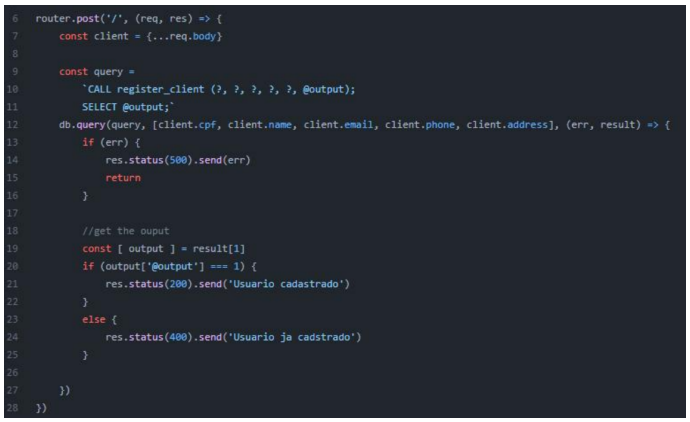
É também na nossa API que é realizada a comunicação com o banco de dados mysql. Então o módulo **mysql2** provem um **Hanlder** que nos permite realizar consultar SQL para manipular e persistir os dados no banco de dados



```
const mysql = require('mysql2')

const db = mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    port: process.env.DB_PORT,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_DATABASE,
    multipleStatements: true,
})



db.connect(err => {
    if (err) {
        console.log('Connection faild. ',err.stack)
        return
    }
    console.log('Connected as id: ', db.threadId)
})


module.exports = db
```

Figure 6. Handler com Banco de dados

Com isso, a camada de persistência já está pronta e configurada de tal forma que o cliente que acessa a API não se preocupa em como os dados serão entregues para ele, ou seja, o acesso aos dados foi encapsulado conforme o modelo DAO. Voltando para a parte em que o cliente realizou uma requisição do tipo POST na imagem 6), ele acessa a API que receberá essa requisição, poderá fazer ou não algum tipo de tratamento no dado, realiza a comunicação com o banco de dados e manipula esses dados por meio de comandos SQL.

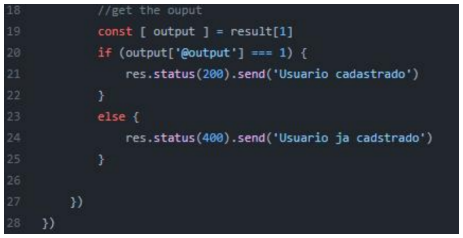


```
router.post('/', (req, res) => {
    const client = {...req.body}

    const query =
        `CALL register_client (?, ?, ?, ?, ?, @output);
        SELECT @output;`
    db.query(query, [client.cpf, client.name, client.email, client.phone, client.address], (err, result) => {
        if (err) {
            res.status(500).send(err)
            return
        }

        //get the ouput
        const [ output ] = result[1]
        if (output['@output'] === 1) {
            res.status(200).send('Usuario cadastrado')
        }
        else {
            res.status(400).send('Usuario ja cadstrado')
        }

    })
})
```

Figure 7. API (DAO)

Então, a API receberá informações a respeito do comando SQL do banco de dados e enviará resposta de erro com código HTTP 500 caso algo deu errado durante a consulta por parte do banco de dados, resposta com código 400 caso o dado que o cliente forneceu foi incorreto ou resposta com código 200 se os dados foram extraídos ou inseridos com sucesso.

(a) Value Object



(b) Requisição Post

References

1. Slides da Professora Maristela Holanda para ministrar a disciplina de Banco de Dados na Universidade de Brasília - Semestre 2022.1
2. Código fonte camada de mapeamento (BACK-END) do Projeto 6: Clique Aqui
3. BR MODELO WEB: ferramenta de modelagem para o MER 2
4. MySQL Workbench Models: ferramenta de modelagem para o Modelo Relacional 3
5. Scrip SQL das Querys e Comandos que geraram o banco de dados: Clique Aqui
6. Código do FRONT-END do Projeto: Clique Aqui