# Trabalho Prático SBD - Parte Final - Etapa 3

## Alunos:

Gabriel Ribeiro Bernardi - 11821BCC036
Gabriel Solis Corrêa - 11711BCC032
Guilherme Soares Correa - 11821BCC026

## 2.f.i)

A aplicação escolhida foi BD de uma Universidade, presente no arquivo L2_ER.pdf, no tópico II.4).
Para o desenvolvimento da aplicação foi usado NodeJS juntamente com uma biblioteca que conectava a aplicação com o Banco de Dados (DB) utilizado, que no caso foi o MySQL. Essa biblioteca é o Knex JS. Os trechos de código mostrados abaixo e o código completo estão presentes no github:
https://github.com/gabrielrbernardi/SBD-TrabalhoFinal

Às interfaces de conexão da aplicação com o DB para inclusão, alteração, exclusão e consulta seguem abaixo:

Observação: em relação à inclusão e/ou alteração de um registro, o mesmo receberá alguns parâmetros a serem adicionados ou alterados.

### Inclusão

Inclusão de uma disciplina

```
async createDisciplina(request: Request, response: Response){
    const {siglaDisciplina, nomeDisciplina, nroCreditos, idTurma} = request.body;
    const disciplina = {
        siglaDisciplina,
        nomeDisciplina,
        nroCreditos,
        idTurma,
    }
    await knex("Disciplina").insert(disciplina).then(responseDiscipline => {
        if(responseDiscipline){
            return response.json({createdDsicipline: true});
        }else{
            return response.json({createdDsicipline: false, error: "Não foi possível criar a Disciplina."});
        }
    })
}
```

Inclusão de um aluno

```
async createAluno(request: Request, response: Response){
    const {nome, dataNascimento, cra, telefone, siglaFaculdade} = request.body;
    const aluno = {
        nomeAluno: nome,
        dataNascimento,
        cra,
        telefone,
        siglaFaculdade
    }
    await knex("Aluno").insert(aluno).then(responseStudent => {
        if(responseStudent){
            return response.json({createdStudent: true});
        }else{
            return response.json({createdStudent: false, error: "Não foi possível criar o Aluno."});
        }
    })
}
```

## Alteração

### Alteração de uma disciplina

```
//Alteracao
async updateDisciplina(request: Request, response: Response){
    const {siglaDisciplina, nomeDisciplina, nroCreditos} = request.body;
    const disciplinaDB = await knex('Disciplina').where('siglaDisciplina', siglaDisciplina).update({
        nomeDisciplina,
        nroCreditos
    });
    if (disciplinaDB) {
        return response.json({ updatedDisciplina: true, disciplinaDB });
    } else {
        return response.json({
            updatedDisciplina: false,
            error: "Não foi possível alterar as informações da Disciplina.",
        });
    }
}
```

### Alteração de um aluno

```
async updateAluno(request: Request, response: Response){
    const {nome, dataNascimento, cra, telefone} = request.body;
    const {id} = request.params;
    const idInt = parseInt(id)
    const alunoDB = await knex('Aluno').where('idAluno', idInt).update({
        nomeAluno: nome,
        dataNascimento,
        cra,
        telefone
    });
    if (alunoDB) {
        return response.json({ updatedAluno: true, alunoDB });
    } else {
        return response.json({
            updatedAluno: false,
            error: "Não foi possível alterar as informações do Aluno.",
        });
    }
}
```

## Exclusão

### Exclusão de uma disciplina

```
//Exclusao
async deleteDisciplina(request: Request, response: Response){
    const {siglaDisciplina} = request.body;
    const disciplinaDB = await knex('Disciplina').where('siglaDisciplina', siglaDisciplina).del();
    if(disciplinaDB){
        return response.json({deletedDisciplina: true});
    }else{
        return response.json({deletedDisciplina: false});
    }
}
```

### Exclusão de um aluno

```
async deleteAluno(request: Request, response: Response){
    const {id} = request.params;
    const idInt = parseInt(id);
    const alunoDB = await knex('Aluno').where('idAluno', idInt).del();
    if(alunoDB){
        return response.json({deletedAluno: true});
    }else{
        return response.json({deletedAluno: false});
    }
}
```

## Consultas

### Consulta de alunos

```
//Consultas
async showAlunos(request: Request, response: Response){
    const alunos = await knex("Aluno").select("*");
    return response.json(alunos);
}
```

### Consulta de faculdades

```
async showFaculdades(request: Request, response: Response){
    const faculdades = await knex("Faculdade").select("*");
    return response.json(faculdades);
}
```

### Consulta de turmas

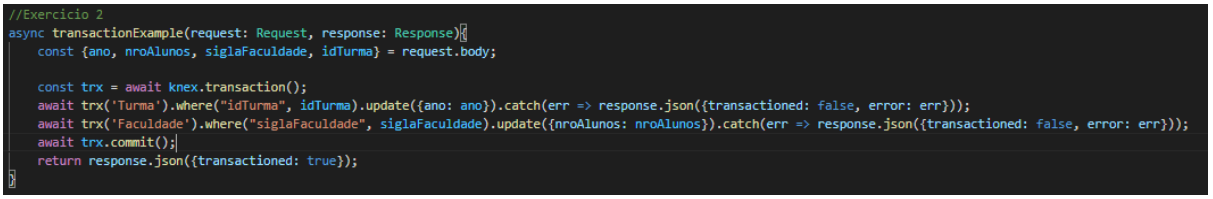
```
async showTurma(request: Request, response: Response){
    const turmas = await knex("Turma").select("*");
    return response.json(turmas);
}
```

### Consulta de disciplina

```
async showDisciplina(request: Request, response: Response){
    const disciplinas = await knex("Disciplina").select("*");
    return response.json(disciplinas);
}
```

## 2.f.v)

A transação escolhida foi relacionada a atualização do ano de uma determinada turma e a atualização da quantidade de alunos de uma determinada faculdade.



```
//Exercicio 2
async transactionExample(request: Request, response: Response){
    const {ano, nroAlunos, siglaFaculdade, idTurma} = request.body;

    const trx = await knex.transaction();
    await trx('Turma').where("idTurma", idTurma).update({ano: ano}).catch(err => response.json({transactioned: false, error: err}));
    await trx('Faculdade').where("siglaFaculdade", siglaFaculdade).update({nroAlunos: nroAlunos}).catch(err => response.json({transactioned: false, error: err}));
    await trx.commit();
    return response.json({transactioned: true});
}
```

## 2.f.vi)

Para a realização deste tópico foi feito um script SQL utilizando algumas técnicas para verificar a integridade dos dados, a utilização de *loops* e a utilização de triggers. O código segue abaixo e também ficará disponível no arquivo Trigger.sql no github citado anteriormente.

```
CREATE OR REPLACE FUNCTION verifica_aluno()
RETURNS TRIGGER AS $$
DECLARE
    tamanho INTEGER := 0;
    fileira aluno%rowtype;

    erro BOOLEAN := false;
    textoerro VARCHAR(50);
BEGIN
    FOR fileira IN SELECT * FROM aluno
    LOOP
        IF fileira.idAluno IS NULL THEN
            erro := true;
            textoerro := "Id de Aluno " || fileira.nomeAluno || " não
pode ser núlo.";
            RETURN
        ELSE
            IF fileira.idAluno < 0 THEN
                erro := true;
                textoerro := "Id de Aluno " || fileira.nomeAluno || "
Inválido.";
            END IF;
        END IF;

        IF fileira.cra IS NULL THEN
            erro := true;
            textoerro := "CRA de Aluno " || fileira.nomeAluno || " não
pode ser núlo.";
```

```
        ELSE
            IF fileira.cra < 0 THEN
                erro := true;
                textoerro := "CRA de Aluno " || fileira.nomeAluno || "
Inválido.";
            END IF;
        END IF;

        tamanho := LENGTH(fileira.telefone);

        IF tamanho IS NULL THEN
            erro := true;
            textoerro := "Telefone de Aluno " || fileira.nomeAluno || "
não pode ser núlo.";
        ELSE
            IF tamanho < 0 THEN
                erro := true;
                textoerro := "Telefone de Aluno " || fileira.nomeAluno
|| " Inválido.";
            END IF;
        END IF;

        IF erro THEN

            RAISE EXCEPTION "Após a query foi possível encontrar este
erro na tabela Aluno: " || textoerro;
    END LOOP;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER verifica_aluno
AFTER INSERT OR UPDATE OR DELETE ON aluno
FOR EACH STATEMENT EXECUTE PROCEDURE verifica_aluno();
```