

SQL para criar tabelas e atributos Via VsCode

```
CREATE TABLE IF NOT EXISTS professores (  
    id SERIAL PRIMARY KEY,  
    nome VARCHAR(100) NOT NULL,  
    departamento VARCHAR(100) NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS disciplinas (  
    id SERIAL PRIMARY KEY,  
    nome VARCHAR(100) NOT NULL,  
    carga_horaria INTEGER NOT NULL,  
    professor_id INTEGER NOT NULL,  
    FOREIGN KEY (professor_id) REFERENCES professores (id) ON DELETE  
CASCADE  
);  
  
CREATE TABLE IF NOT EXISTS alunos (  
    id SERIAL PRIMARY KEY,  
    nome VARCHAR(100) NOT NULL,  
    data_nascimento DATE NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS matriculas (  
    id SERIAL PRIMARY KEY,  
    aluno_id INTEGER NOT NULL,  
    disciplina_id INTEGER NOT NULL,  
    data_matricula DATE NOT NULL,  
    FOREIGN KEY (aluno_id) REFERENCES alunos (id) ON DELETE CASCADE,  
    FOREIGN KEY (disciplina_id) REFERENCES disciplinas (id) ON DELETE  
CASCADE  
);
```

Populando essas tabelas de forma não aleatória

```
INSERT INTO professores (nome, departamento) VALUES
('João Silva', 'Matemática'),
('Maria Oliveira', 'História'),
('Pedro Souza', 'Física'),
('Ana Costa', 'Química'),
('Carla Martins', 'Biologia'),
('Luis Mendes', 'Português'),
('Paula Ramos', 'Geografia'),
('Marcos Lima', 'Artes'),
('Fernanda Dias', 'Educação Física'),
('Roberto Nunes', 'Filosofia');
```

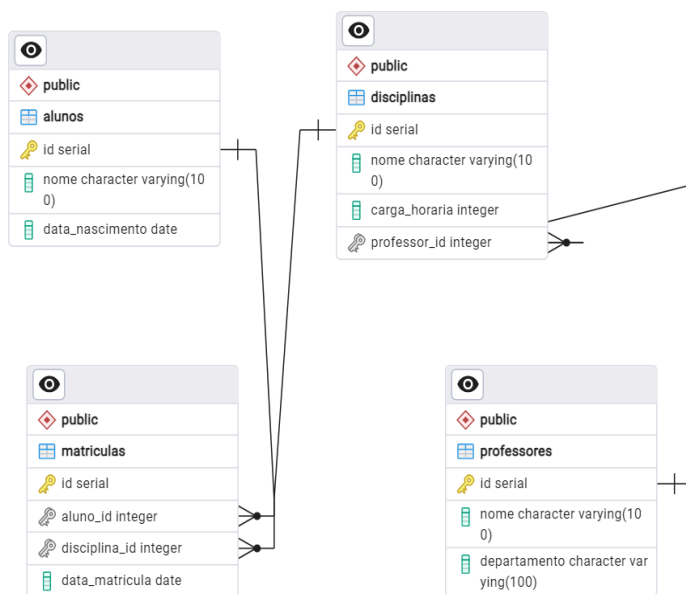
```
INSERT INTO disciplinas (nome, carga_horaria, professor_id) VALUES
('Álgebra', 60, 1),
('Geometria', 60, 1),
('História Mundial', 45, 2),
('Física I', 60, 3),
('Química Orgânica', 50, 4),
('Biologia Geral', 55, 5),
('Gramática', 40, 6),
('Geografia do Brasil', 45, 7),
('História da Arte', 35, 8),
('Filosofia Moderna', 50, 10);
```

```
INSERT INTO alunos (nome, data_nascimento) VALUES
('Carlos Souza', '2005-05-12'),
('Ana Pereira', '2006-08-22'),
('Ricardo Alves', '2005-09-15'),
('Mariana Gomes', '2006-03-08'),
('Felipe Silva', '2005-12-30'),
('Patricia Lima', '2006-07-21'),
('Eduardo Santos', '2005-11-02'),
('Bruna Castro', '2006-01-17'),
```

```
(('Gustavo Pires', '2005-04-25'),
('Sofia Rodrigues', '2006-10-10'));






INSERT INTO matriculas (aluno_id, disciplina_id, data_matricula) VALUES
(1, 1, '2025-01-10'),
(1, 3, '2025-01-11'),
(2, 2, '2025-01-12'),
(2, 4, '2025-01-13'),
(3, 5, '2025-01-14'),
(4, 6, '2025-01-15'),
(5, 7, '2025-01-16'),
(6, 8, '2025-01-17'),
(7, 9, '2025-01-18'),
(8, 10, '2025-01-19');
```

Modelo ER








Consultas

```
SELECT * FROM professores;
```

Data Output Messages Notifications			
    			
	id [PK] integer	nome character varying (100)	departamento character varying (100)
1	1	João Silva	Matemática
2	2	Maria Oliveira	História
3	3	Pedro Souza	Física
4	4	Ana Costa	Química
5	5	Carla Martins	Biologia
6	6	Luis Mendes	Português
7	7	Paula Ramos	Geografia
8	8	Marcos Lima	Artes
9	9	Fernanda Dias	Educação Física
10	10	Roberto Nunes	Filosofia

```
SELECT * FROM disciplinas;
```

Data Output Messages Notifications				
    				
	id [PK] integer	nome character varying (100)	carga_horaria integer	professor_id integer
1	1	Álgebra	60	1
2	2	Geometria	60	1
3	3	História Mundial	45	2
4	4	Física I	60	3
5	5	Química Orgânica	50	4
6	6	Biologia Geral	55	5
7	7	Gramática	40	6
8	8	Geografia do Brasil	45	7
9	9	História da Arte	35	8
10	10	Filosofia Moderna	50	10

```
SELECT * FROM alunos;
```

Data Output Messages Notifications				
	id [PK] integer	nome character varying (100)	data_nascimento date	
1	1	Carlos Souza	2005-05-12	
2	2	Ana Pereira	2006-08-22	
3	3	Ricardo Alves	2005-09-15	
4	4	Mariana Gomes	2006-03-08	
5	5	Felipe Silva	2005-12-30	
6	6	Patricia Lima	2006-07-21	
7	7	Eduardo Santos	2005-11-02	
8	8	Bruna Castro	2006-01-17	
9	9	Gustavo Pires	2005-04-25	
10	10	Sofia Rodrigues	2006-10-10	

```
SELECT * FROM matriculas;
```

Data Output Messages Notifications					
	id [PK] integer	aluno_id integer	disciplina_id integer	data_matricula date	
1	1	1	1	2025-01-10	
2	2	1	3	2025-01-11	
3	3	2	2	2025-01-12	
4	4	2	4	2025-01-13	
5	5	3	5	2025-01-14	
6	6	4	6	2025-01-15	
7	7	5	7	2025-01-16	
8	8	6	8	2025-01-17	
9	9	7	9	2025-01-18	
10	10	8	10	2025-01-19	

Aplicação Crud interligada com o PostGress para gerenciamento de dados

```
from flask import Flask, jsonify, request
import psycopg2
from psycopg2.extras import RealDictCursor

app = Flask(__name__)

DB_CONFIG = {
    "dbname": "SistemaEscola",
    "user": "postgres",
    "password": "lalalala",
    "host": "localhost",
    "port": "5432"
}

def get_db_connection():
    conn = psycopg2.connect(**DB_CONFIG, cursor_factory=RealDictCursor)
    return conn

@app.route('/')
def home():
    try:
        conn = get_db_connection()
        conn.close()
        return jsonify({"message": "Conexão com o BD estabelecida com sucesso!"})
    except Exception as e:
        return jsonify({"message": "Falha na conexão com o BD.",
                        "error": str(e)}), 500

### CRUD PARA PROFESSORES ###

@app.route('/professores', methods=['GET'])
def get_professores():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM professores;")
    professores = cur.fetchall()
```

```

        cur.close()
        conn.close()
        return jsonify(professores)

@app.route('/professores/<int:id>', methods=['GET'])
def get_professor(id):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM professores WHERE id = %s;", (id,))
    professor = cur.fetchone()
    cur.close()
    conn.close()
    if professor:
        return jsonify(professor)
    return jsonify({'message': 'Professor não encontrado'}), 404

@app.route('/professores', methods=['POST'])
def create_professor():
    data = request.get_json()
    nome = data.get('nome')
    departamento = data.get('departamento')
    if not nome or not departamento:
        return jsonify({'message': 'Nome e departamento são obrigatórios'}), 400

    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("INSERT INTO professores (nome, departamento) VALUES (%s, %s) RETURNING *;",
                (nome, departamento))
    novo_professor = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    return jsonify(novo_professor), 201

@app.route('/professores/<int:id>', methods=['PUT'])
def update_professor(id):
    data = request.get_json()
    nome = data.get('nome')
    departamento = data.get('departamento')
    conn = get_db_connection()
    cur = conn.cursor()

```

```

        cur.execute("UPDATE professores SET nome = %s, departamento = %s
WHERE id = %s RETURNING *;",
                    (nome, departamento, id))
    professor_atualizado = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    if professor_atualizado:
        return jsonify(professor_atualizado)
    return jsonify({'message': 'Professor não encontrado'}), 404

@app.route('/professores/<int:id>', methods=['DELETE'])
def delete_professor(id):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("DELETE FROM professores WHERE id = %s RETURNING *;",
(id,))
    professor_deletado = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    if professor_deletado:
        return jsonify({'message': 'Professor deletado com sucesso'})
    return jsonify({'message': 'Professor não encontrado'}), 404

### CRUD PARA DISCIPLINAS ###

@app.route('/disciplinas', methods=['GET'])
def get_disciplinas():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM disciplinas;")
    disciplinas = cur.fetchall()
    cur.close()
    conn.close()
    return jsonify(disciplinas)

@app.route('/disciplinas/<int:id>', methods=['GET'])
def get_disciplina(id):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM disciplinas WHERE id = %s;", (id,))
    disciplina = cur.fetchone()

```



```

        cur.close()
        conn.close()
        if disciplina:
            return jsonify(disciplina)
        return jsonify({'message': 'Disciplina não encontrada'}), 404

@app.route('/disciplinas', methods=['POST'])
def create_disciplina():
    data = request.get_json()
    nome = data.get('nome')
    carga_horaria = data.get('carga_horaria')
    professor_id = data.get('professor_id')
    if not nome or not carga_horaria or not professor_id:
        return jsonify({'message': 'Nome, carga horária e professor_id são obrigatórios'}), 400

    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
        INSERT INTO disciplinas (nome, carga_horaria, professor_id)
        VALUES (%s, %s, %s) RETURNING *;
    """, (nome, carga_horaria, professor_id))
    nova_disciplina = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    return jsonify(nova_disciplina), 201

@app.route('/disciplinas/<int:id>', methods=['PUT'])
def update_disciplina(id):
    data = request.get_json()
    nome = data.get('nome')
    carga_horaria = data.get('carga_horaria')
    professor_id = data.get('professor_id')
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
        UPDATE disciplinas SET nome = %s, carga_horaria = %s,
professor_id = %s
        WHERE id = %s RETURNING *;
    """, (nome, carga_horaria, professor_id, id))
    disciplina_atualizada = cur.fetchone()
    conn.commit()

```

```

        cur.close()
        conn.close()
        if disciplina_atualizada:
            return jsonify(disciplina_atualizada)
        return jsonify({'message': 'Disciplina não encontrada'}), 404

@app.route('/disciplinas/<int:id>', methods=['DELETE'])
def delete_disciplina(id):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("DELETE FROM disciplinas WHERE id = %s RETURNING *;",
(id,))
    disciplina_deletada = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    if disciplina_deletada:
        return jsonify({'message': 'Disciplina deletada com sucesso'})
    return jsonify({'message': 'Disciplina não encontrada'}), 404

### CRUD PARA ALUNOS ###

@app.route('/alunos', methods=['GET'])
def get_alunos():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM alunos;")
    alunos = cur.fetchall()
    cur.close()
    conn.close()
    return jsonify(alunos)

@app.route('/alunos/<int:id>', methods=['GET'])
def get_aluno(id):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM alunos WHERE id = %s;", (id,))
    aluno = cur.fetchone()
    cur.close()
    conn.close()
    if aluno:
        return jsonify(aluno)
    return jsonify({'message': 'Aluno não encontrado'}), 404

```

```

@app.route('/alunos', methods=['POST'])
def create_aluno():
    data = request.get_json()
    nome = data.get('nome')
    data_nascimento = data.get('data_nascimento') # Formato:
    'YYYY-MM-DD'
    if not nome or not data_nascimento:
        return jsonify({'message': 'Nome e data_nascimento são obrigatórios'}), 400

    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
        INSERT INTO alunos (nome, data_nascimento)
        VALUES (%s, %s) RETURNING *;
    """, (nome, data_nascimento))
    novo_aluno = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    return jsonify(novo_aluno), 201

@app.route('/alunos/<int:id>', methods=['PUT'])
def update_aluno(id):
    data = request.get_json()
    nome = data.get('nome')
    data_nascimento = data.get('data_nascimento')
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
        UPDATE alunos SET nome = %s, data_nascimento = %s
        WHERE id = %s RETURNING *;
    """, (nome, data_nascimento, id))
    aluno_atualizado = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    if aluno_atualizado:
        return jsonify(aluno_atualizado)
    return jsonify({'message': 'Aluno não encontrado'}), 404

@app.route('/alunos/<int:id>', methods=['DELETE'])

```

```

def delete_aluno(id):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("DELETE FROM alunos WHERE id = %s RETURNING *;", (id,))
    aluno_deletado = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    if aluno_deletado:
        return jsonify({'message': 'Aluno deletado com sucesso'})
    return jsonify({'message': 'Aluno não encontrado'}), 404

### CRUD PARA MATRÍCULAS ###

@app.route('/matriculas', methods=['GET'])
def get_matriculas():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM matriculas;")
    matriculas = cur.fetchall()
    cur.close()
    conn.close()
    return jsonify(matriculas)

@app.route('/matriculas/<int:id>', methods=['GET'])
def get_matricula(id):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM matriculas WHERE id = %s;", (id,))
    matricula = cur.fetchone()
    cur.close()
    conn.close()
    if matricula:
        return jsonify(matricula)
    return jsonify({'message': 'Matrícula não encontrada'}), 404

@app.route('/matriculas', methods=['POST'])
def create_matricula():
    data = request.get_json()
    aluno_id = data.get('aluno_id')
    disciplina_id = data.get('disciplina_id')
    data_matricula = data.get('data_matricula') # Formato:
'YYYY-MM-DD'

```

```

        if not aluno_id or not disciplina_id or not data_matricula:
            return jsonify({'message': 'aluno_id, disciplina_id e
data_matricula são obrigatórios'}), 400

    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
        INSERT INTO matriculas (aluno_id, disciplina_id,
data_matricula)
        VALUES (%s, %s, %s) RETURNING *;
    """, (aluno_id, disciplina_id, data_matricula))
    nova_matricula = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    return jsonify(nova_matricula), 201

@app.route('/matriculas/<int:id>', methods=['PUT'])
def update_matricula(id):
    data = request.get_json()
    aluno_id = data.get('aluno_id')
    disciplina_id = data.get('disciplina_id')
    data_matricula = data.get('data_matricula')
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
        UPDATE matriculas SET aluno_id = %s, disciplina_id = %s,
data_matricula = %s
        WHERE id = %s RETURNING *;
    """, (aluno_id, disciplina_id, data_matricula, id))
    matricula_atualizada = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    if matricula_atualizada:
        return jsonify(matricula_atualizada)
    return jsonify({'message': 'Matrícula não encontrada'}), 404

@app.route('/matriculas/<int:id>', methods=['DELETE'])
def delete_matricula(id):
    conn = get_db_connection()
    cur = conn.cursor()

```

```

    cur.execute("DELETE FROM matriculas WHERE id = %s RETURNING *;",
(id,))
    matricula_deletada = cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    if matricula_deletada:
        return jsonify({'message': 'Matrícula deletada com sucesso'})
    return jsonify({'message': 'Matrícula não encontrada'}), 404

if __name__ == '__main__':
    app.run(debug=True)

```

Esta aplicação é uma API RESTful construída em Python utilizando o Flask e o PostgreSQL (através da biblioteca `psycopg2`). Ela permite gerenciar dados de um sistema escolar. A seguir, um resumo das funcionalidades:

1. Verificação da Conexão com o Banco de Dados:

- O endpoint raiz (/) testa se a aplicação consegue se conectar ao banco de dados, retornando uma mensagem de sucesso ou erro.

2. Operações CRUD (Create, Read, Update, Delete):

- **Professores:**
 - Listar todos os professores: `GET /professores`
 - Obter um professor específico: `GET /professores/<id>`
 - Criar um novo professor: `POST /professores`
 - Atualizar um professor existente: `PUT /professores/<id>`

- Deletar um professor: DELETE
/professores/<id>

- Disciplinas:

- Listar todas as disciplinas: GET /disciplinas
- Obter uma disciplina específica: GET
/disciplinas/<id>
- Criar uma nova disciplina: POST /disciplinas
- Atualizar uma disciplina existente: PUT
/disciplinas/<id>
- Deletar uma disciplina: DELETE
/disciplinas/<id>

- Alunos:

- Listar todos os alunos: GET /alunos
- Obter um aluno específico: GET /alunos/<id>
- Criar um novo aluno: POST /alunos
- Atualizar um aluno existente: PUT /alunos/<id>
- Deletar um aluno: DELETE /alunos/<id>

- Matrículas:

- Listar todas as matrículas: GET /matriculas
- Obter uma matrícula específica: GET
/matriculas/<id>
- Criar uma nova matrícula: POST /matriculas
- Atualizar uma matrícula existente: PUT
/matriculas/<id>

■ Deletar uma matrícula: DELETE
/matriculas/<id>

3. Relacionamento entre as Entidades:

- Cada disciplina está associada a um professor.
- Matrículas representam a associação entre alunos e disciplinas, implementando uma relação muitos-para-muitos entre essas entidades.

Resumo do Que a Aplicação Pode Fazer:

- **Gerenciamento de Dados:**

Permite a inserção, consulta, atualização e remoção de registros de professores, disciplinas, alunos e matrículas.

- **Integração com o Banco de Dados:**

Conecta-se ao PostgreSQL para persistir e manipular os dados do sistema escolar.

- **Testes e Validação:**

Através dos endpoints, é possível testar todas as operações CRUD, facilitando o desenvolvimento, manutenção e eventual integração com outras interfaces (por exemplo, um front-end web).

- **Extensibilidade:**

A estrutura da API possibilita a adição de novos endpoints e funcionalidades conforme as necessidades do sistema escolar evoluírem.

Em resumo, essa aplicação serve como base para um sistema de gerenciamento escolar, permitindo a administração dos elementos fundamentais (professores, disciplinas, alunos e matrículas) e facilitando a integração de futuras funcionalidades.

