Parada Tech da Comunidade

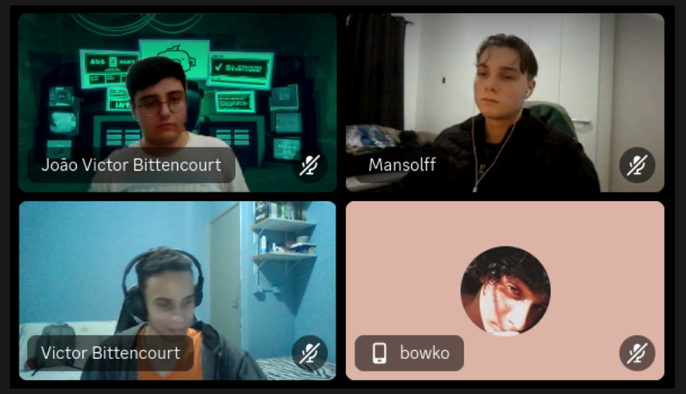
Introdução a SQL com Python

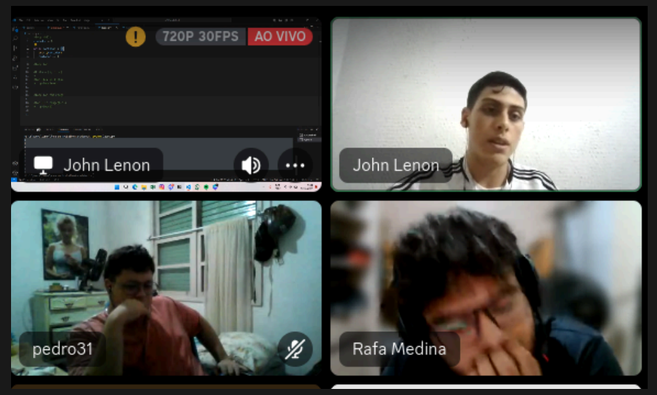
Criado por Victor Bittencourt, trilha de Python

O que é a

Parada Tech da Comunidade?

É um evento dos alunos que tenta simular as mesmas *Paradas Tech* que tivemos no decorrer do estudo assíncrono.





Trazemos introduções, resumos e tentamos tirar dúvidas de assuntos relacionados a desenvolvimento num geral.

Ferramentas Utilizadas

Programar

VSCode

Python

Com sqlite3

Manipular SQL Local

DBeaver

Manipular SQL Online

SQL IDE Online

Alunos Instrutores

Lauro Mendelski

Victor Bittencourt

Não possuímos nenhum vínculo empregatício com o Instituto Caldeira, somos somente alunos ensinando outros alunos;

Who am I?

Lauro Mendelski

18 anos, Porto Alegre
ADS no IFRS
Programador BackEnd

JavaScript Lover <3 <3 <3

Lauro Mendelski

Não possuímos nenhum vínculo empregatício com o Instituto Caldeira, somos somente alunos ensinando outros alunos;

Who I am?

Victor Bittencourt

20 anos, de Jundiaí, SP
em Gravataí, RS
Programador Fullstack

Atualmente estudo:
> React;
> Ruby;

VicourtBitt

vicourtbitt

Não possuímos nenhum vínculo empregatício com o Instituto Caldeira, somos somente alunos ensinando outros alunos;

Iniciação a SQL com Python

SQLite3

Python

Structured Query Language

O SQL é uma linguagem de consulta relacional muito utilizada em bancos de dados que precisam manter a integridade dos dados, assim como um esquema estruturado. Originalmente criado em 1974, porém, contendo diversas versões hoje em dia.

	id	first_name	last_name	gender	age	general_role	userRegisterId
1	1	Victor	Bittencourt	Masculino	20	Fullstack Developer	1
2	2	Pricila	Jose	Feminino	62	Artesã	2
3	3	Joel	Bittencourt	Masculino	72	Security Supervisor	3
4	4	Valdomiro	Quintava	Masculino	65	Mantenedor Predial	4
5	5	Rose	Flores	Feminino	59	Pedagoga	5
6	6	Steffany	Sympson	Feminino	35	Fullstack Developer	6
7	7	Julia	Quintava	Feminino	19	Auxiliar de Almoxarifac	7
8	8	Mauro	Sérgio	Masculino	45	Professor	8
9	9	Anderson	Souza	Masculino	56	Professor	9
10	10	Laura	Sadinzk	Feminino	71	Quimica	10
11	11	Emerson	Severo	Masculino	63	Técnico Mecânico	11
12	12	Rogério	Mauricio	Masculino	56	Carpinteiro	12
13	13	Marcos	Davila	Masculino	66	Técnico de Segurança	13
14	14	Joao	Vitor	Masculino	61	Programador	14
15	15	Romeu	Damasceno	Masculino	76	Soldador	15
16	16	Daniel	Barbosa	Masculino	71	Marceneiro	16
17	17	Mariana	Graça	Feminino	77	Pintora	17

Diferente das NoSQL, aqui guardamos as informações em tabelas com colunas que determinam a natureza dos valores.

Bancos de Dados SQL

MySQL

MariaDB

PostgreSQL

DuckDB

Modelo de um DB

DataBase

Users

id
first_name
last_name
gender
age

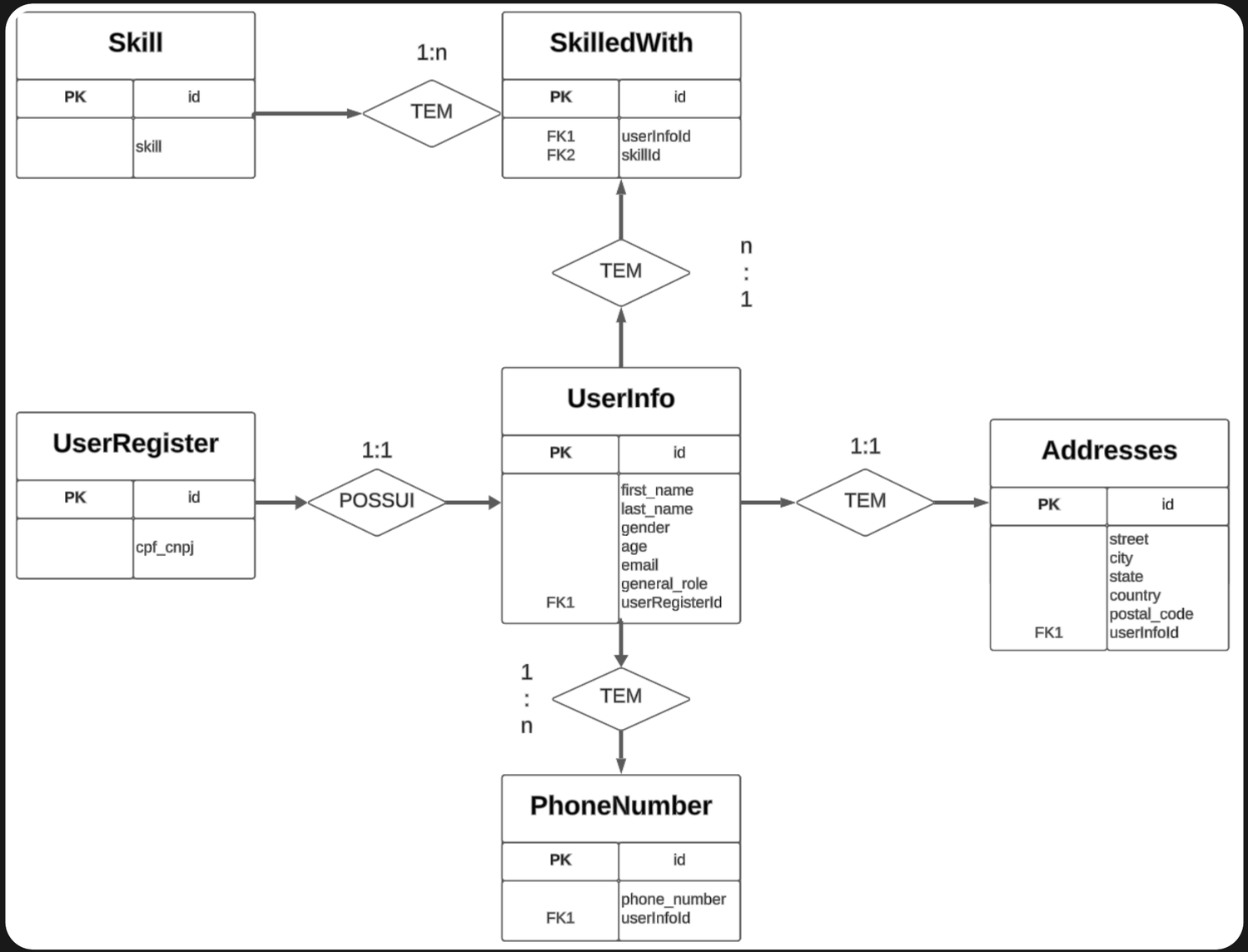
Companies

id
cmpy_name
cmpy_size
cmpy_sector
cmpy_cnpj

Courses

id
course_name
course_dur

MER/DER de um DB



Normalização

1° Norma

1x Chave Primária p/tabela

1x Valor por célula da tabela

Sem duplicatas (linha/col)

1x Valor por col/linha

2° Norma

Satisfazer a 1° Norma

Não haver meia dependência

Errado **UserInfo**

UserId	UserName	StateId	StateName
1	José	16	Santa Catarina
2	Marcos	14	Rio de Janeiro
3	Pedro	20	São Paulo
4	Luana	12	Amazonas

UserInfo

UserId	UserName
1	José
2	Marcos
3	Pedro
4	Luana

Certo

States

StateId	StateName
16	Santa Catarina
14	Rio de Janeiro
20	São Paulo
12	Amazonas

3° Norma

Satisfazer a 2° Norma

Não ter dependência contextual

CREATE TABLE

```sql
CREATE TABLE registroUsuario (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    cpfCnpj VARCHAR(14) NOT NULL
)

CREATE TABLE infoUsuario (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    nome VARCHAR(40) NOT NULL,
    sobrenome VARCHAR(40) NOT NULL,
    idRegistro INTEGER,
    FOREIGN KEY (idRegistro) REFERENCES registroUsuario(ID)
);
```

É o comando inicial ao “setar” uma tabela, onde nos colocamos o nome da mesma logo após o comando e depois declaramos os atributos e suas respectivas regras dentro dos parenteses.

SELECT * FROM X

É o comando de seleção de informações, onde referênciamos as informações que queremos pegar (* pega tudo) e depois referenciamos a tabela que queremos ter como base.

A estrutura se baseia no seguinte:
SELECT
FROM
CONDIÇÕES

```
SELECT *
FROM registroUsuario u
WHERE u.id = 9
```

INSERT INTO X VALUES

É o comando para popular tabelas do banco de dados, onde referenciamos uma tabela do banco e depois damos os valores separados por caixas (parenteses)

```sql
INSERT INTO ee_jobs.UserInfos (first_name, last_name, gender, age, email, general_role, userRegisterId)
    VALUES
    ('Victor', 'Bittencourt', 'Masculino', 20, 'victor@proto.com', 'Fullstack Developer', 1),
    ('Pricila', 'Jose', 'Feminino', 62, 'pipia.pri@proto.com', 'Artesã', 2),
    ('Joel', 'Bittencourt', 'Masculino', 72, 'joel.bitt@proto.com', 'Security Supervisor', 3),
    ('Valdomiro', 'Quintava', 'Masculino', 65, 'valdomiro.pq@proto.com', 'Mantenedor Predial', 4),
    ('Rose', 'Flores', 'Feminino', 59, 'rose.flores@proto.com', 'Pedagoga', 5),
    ('Steffany', 'Sympson', 'Feminino', 35, 'steffany.sympson@proto.com', 'Fullstack Developer', 6),
    ('Julia', 'Quintava', 'Feminino', 19, 'julia.quintava@proto.com', 'Auxiliar de Almoxarifado', 7),
    ('Mauro', 'Sérgio', 'Masculino', 45, 'mauro.sergio@proto.com', 'Professor', 8),
    ('Anderson', 'Souza', 'Masculino', 56, 'anderson.souza@proto.com', 'Professor', 9),
    ('Laura', 'Sadinzk', 'Feminino', 71, 'laura.sadi@proto.com', 'Quimica', 10),
    ('Emerson', 'Severo', 'Masculino', 63, 'severo@proto.com', 'Técnico Mecânico', 11);
```

DELETE FROM X WHERE

É o comando responsável por deletar informações das tabelas do banco baseado em um filtro.

DELETE FROM tabelaEscolhida
WHERE coluna
CONDIÇÃO

```
DELETE FROM musica
WHERE musicaNome
LIKE "Construção"
```

Chave Primária (Primary Key)

Uma chave primária é um campo (ou combinação de campos) em uma tabela que serve como identificador único para cada registro da tabela. Isso significa que o valor de uma chave primária deve ser único para cada registro e não pode ser NULL. O propósito da chave primária é garantir que cada registro possa ser identificado de forma única, permitindo um acesso rápido e preciso aos dados. Além disso, a chave primária é frequentemente usada para criar relações com outras tabelas.

```
id INTEGER PRIMARY KEY
```

Chave Estrangeira (Foreign Key)

Uma chave estrangeira é um campo em uma tabela que estabelece um vínculo entre essa tabela e a chave primária de outra tabela. A chave estrangeira cria uma relação entre os dados de duas tabelas diferentes, assegurando que os dados referenciados existam na tabela de origem.

Propósito: A chave estrangeira é usada para manter a integridade referencial entre as tabelas, ou seja, ela garante que os relacionamentos entre as tabelas permaneçam consistentes. Se uma tabela contém uma chave estrangeira, essa chave deve corresponder a um valor existente na chave primária da tabela relacionada.

FOREIGN KEY

```
FOREIGN KEY (idRegistro) REFERENCES alunosRegistro(ID)
```

INNER JOIN

O INNER JOIN retorna apenas as linhas que têm correspondências em ambas as tabelas envolvidas na junção. Se um registro em uma tabela não tem correspondência na outra, ele será excluído dos resultados.

Ideal quando você precisa apenas dos dados que têm correspondências entre as tabelas

```
SELECT registroAluno.id, infoAluno.primeiro_nome, telefoneAluno.telefone
FROM registroAluno
INNER JOIN infoAluno ON registroAluno.id = infoAluno.registroAluno_id
INNER JOIN telefoneAluno ON registroAluno.id = telefoneAluno.registroAluno_id;
```

FULL JOIN (ou FULL OUTER JOIN)

O FULL JOIN retorna todas as linhas quando há uma correspondência em uma das tabelas. Ele combina os resultados do LEFT JOIN e do RIGHT JOIN. Se não houver correspondência em uma das tabelas, as colunas dessa tabela terão NULL.

Quando você quer todos os registros de ambas as tabelas, com ou sem correspondência.

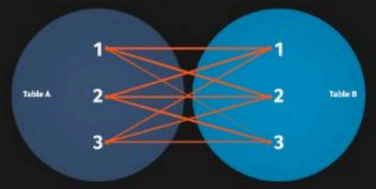
```
SELECT registroAluno.id, infoAluno.primeiro_nome, telefoneAluno.telefone
FROM registroAluno
FULL JOIN infoAluno ON registroAluno.id = infoAluno.registroAluno_id
FULL JOIN telefoneAluno ON registroAluno.id = telefoneAluno.registroAluno_id;
```

LEFT JOIN (ou LEFT OUTER JOIN)

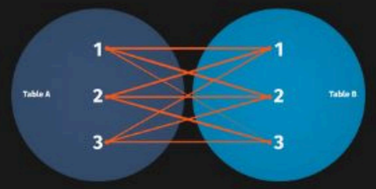
O LEFT JOIN retorna todas as linhas da tabela à esquerda (a primeira tabela mencionada na consulta) e as linhas correspondentes da tabela à direita. Se não houver correspondência na tabela à direita, o resultado será NULL para essas colunas.

Quando você quer todos os registros da tabela principal, mesmo que não haja correspondências na tabela relacionada.

```
SELECT registroAluno.id, infoAluno.primeiro_nome, telefoneAluno.telefone
FROM registroAluno
LEFT JOIN infoAluno ON registroAluno.id = infoAluno.registroAluno_id
LEFT JOIN telefoneAluno ON registroAluno.id = telefoneAluno.registroAluno_id;
```



CROSS JOIN



O CROSS JOIN retorna o produto cartesiano de duas tabelas, ou seja, combina cada linha da primeira tabela com todas as linhas da segunda tabela. O resultado é todas as possíveis combinações de linhas das duas tabelas.

É raramente usado sozinho, mas pode ser útil em situações específicas onde você precisa combinar todas as linhas de duas tabelas.

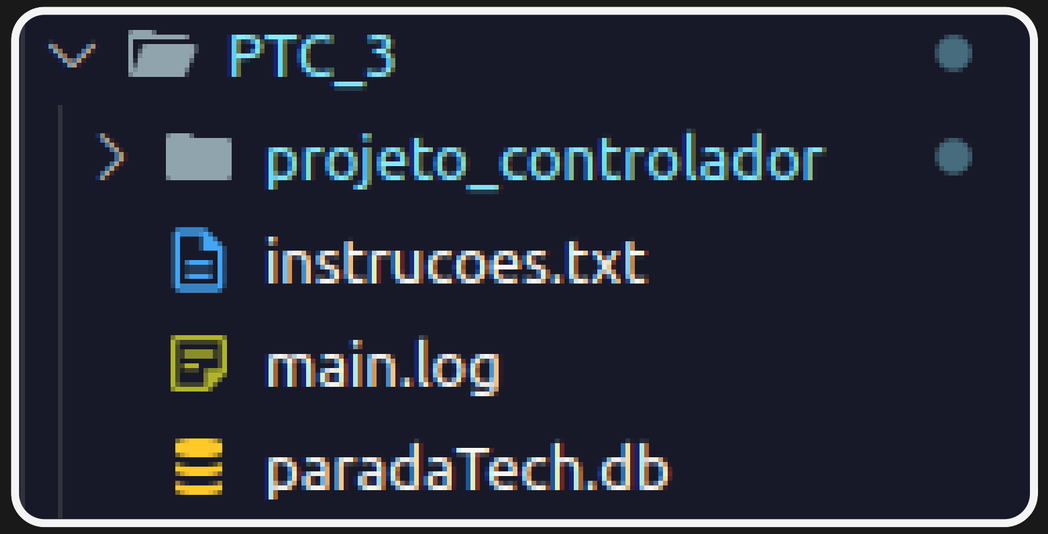
```
SELECT infoAluno.primeiro_nome, cursos.css, cursos.python, cursos.javascript, cursos.sql
FROM infoAluno
CROSS JOIN cursos;
```

SQLite3 & Python

```python
import sqlite3

conn = sqlite3.connect('paradaTech.db')

c = conn.cursor()
```



Para protótipos, aplicações locais ou até aplicações pequenas, podemos utilizar o sqlite3, um banco de dados SQL que não precisa de servidor dedicado para estar funcionando.

SQLite3 não precisa ser instalado, visto que, ele é nativo nas bibliotecas do Python desde o Python 2.7 (*)

(*) A informação pode estar errada quando a versão, alguns indícios apontam desde a 2.5 e outras desde a 2.7

SQLite3 & Python

Importação do módulo

```python
import sqlite3

conn = sqlite3.connect('paradaTech.db')

c = conn.cursor()
```

Criação de um cursor, para guiar as query's

Comando para estabelecer a conexão com DB

Caso não haja nenhum arquivo ".db" na raiz do projeto, ao usar o .connect(nome) um arquivo será criado

Classe para conexão

Podemos criar aplicações com sqlite3 utilizando somente funções (como na documentação do próprio modulo), mas hoje vamos criar esse “Controlador” utilizando OOP.

Também utilizaremos algumas práticas novas (como TypeHint ou Log Storage)

```python
class SQLiteConnection:
    """Classe responsável por se conectar a database, permitindo
    ações como select, insert, update e delete. Tudo com DocStrings
    e seus parametros adicionais conforme documentação."""

    def __init__(self, database):
        self._database = sqlite3.connect(database)

    def criarEstudante(self, valuesToInsert: tuple):
        """ Query com docstring literal para postar
        o novo estudante no sistema. Alunos também
        será a tabela default
        > Example:
        - sql = '''INSERT INTO alunos (firstName, lastName) VALUES (?, ?)'''
        - valuesToInsert = ('Victor', 'Bittencourt')
        - c = cursor.execute(sql, valuesToInsert) """

        try:
            # QUERY LITERAL
            sql = """
                INSERT INTO alunos (firstName, lastName)
                VALUES (?, ?)
            """

            cur = self._database.cursor()
            cur.execute(sql, valuesToInsert)

            # ENVIO DOS VALORES DE FORMA PERMANENTE
            self._database.commit()
            return cur.lastrowid

        except (ValueError, TypeError) as e:
            addIntoLog('error', e)
```

(*) A informação pode estar errada quando a versão, alguns indícios apontam desde a 2.5 e outras desde a 2.7

PROJETO PRÁTICO

```
SEJA BEM VINDO AO CADASTRO DE ALUNOS

Ações > [NewUser] - [AllUsers] - [DeleteUser]
Deseja fazer o que?
```

id	firstName	lastName	createdAt
1	Victor	Bittencourt	2024-08-26 15:03:50.977
2	Lauro	Mendelski	2024-08-26 15:37:09.144
3	Pedro	Barbosa	2024-08-26 18:51:17.362

Caso não haja nenhum arquivo ".db" na raiz do projeto, ao usar o .connect(nome) um arquivo será criado

Chegamos ao fim!

Deixe seu feedback ou sugestão!