Trabalho Final de Aspectos e Implementação de Banco de Dados

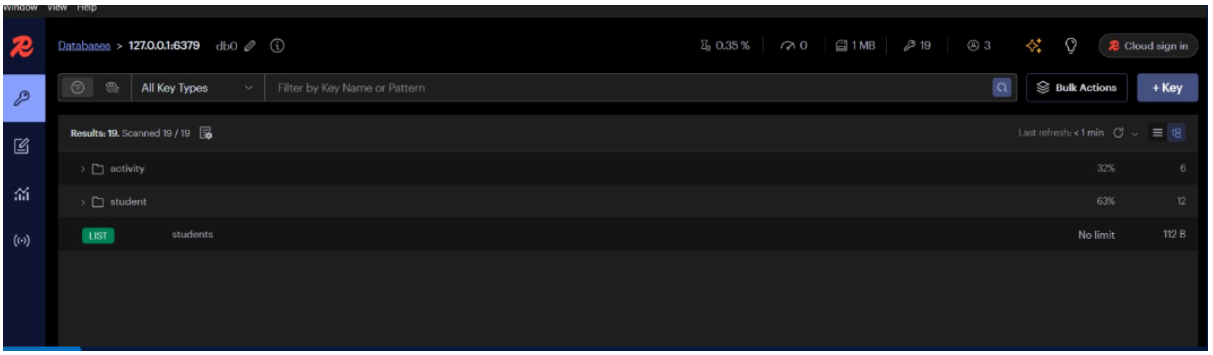
Raissa Barbosa dos Santos - RA: 148551, Tainara Oliveira Santos - RA: 148486, Victor Hugo Teodoro Pimentel - RA: 150976

Universidade Federal de São Paulo (UNIFESP) – São José dos Campos – SP – Brasil

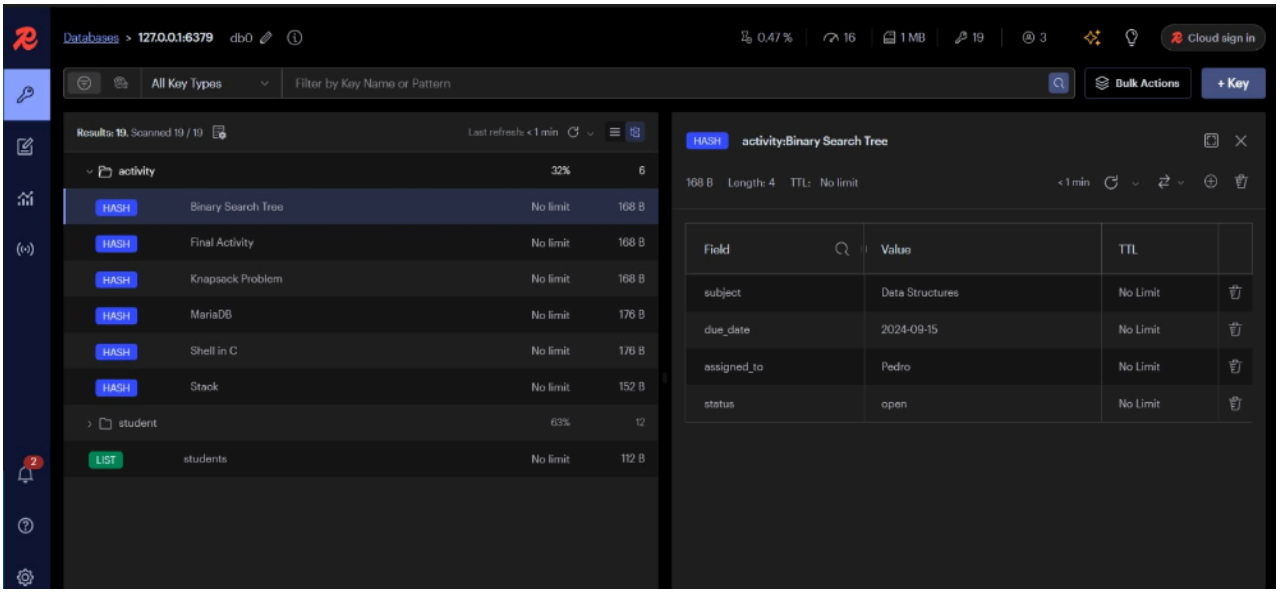
raissa.barbosa@unifesp.br, santos.oliveira@unifesp.br, victor.pimentel16@unifesp.br

1. Descrição

O SGBD escolhido para a realização do trabalho foi o Redis, junto com as tecnologias Python 3.8 e a biblioteca Flask. Em relação ao SGBD, é um banco de dados que armazena dados de chave-valor e, diferente dos bancos tradicionais, o Redis não utiliza tabelas e colunas, mas sim um modelo de dados flexível, em que as informações são armazenadas em estruturas de dados como strings, listas, conjuntos e hashes. Para o trabalho, optamos trabalhar com hashes e listas. Implementamos um sistema que gerencia dados de estudantes e informações acadêmicas da UNIFESP.



1 - Estrutura das Pastas no Redis. Fonte: Os Autores.



2 - Estrutura das Pastas no Redis. Fonte: Os Autores.

2. Modelo de Dados

Para o modelo de dados temos uma lista de estudantes, sendo composta por:

- **Chave:** students
  - **Tipo:** Lista
  - **Valores:** nomes dos estudantes(ex:'Tainara')

Além disso, temos os detalhes dos estudantes, sendo composto por:

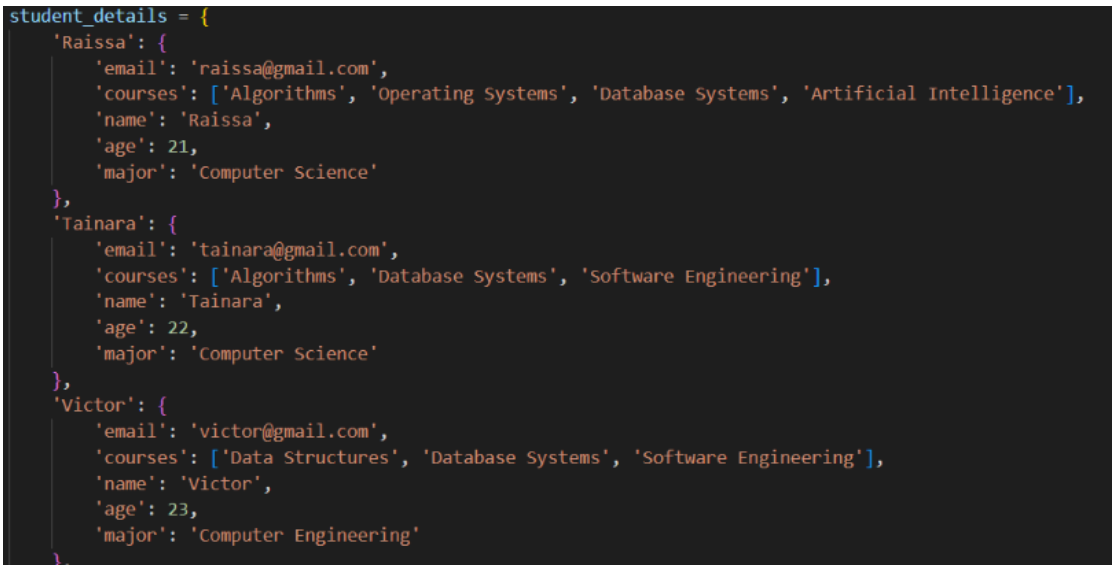
- **Chave:** courses
  - **Tipo:** Lista
  - **Valores:** Curso associados ao estudante [ex: Algorithms', 'Operating Systems', 'Database Systems', 'Artificial Intelligence']
- **Chave:** details
  - **Tipo:** Hash
  - **Campos e Valores:**

    name: 'Raissa'

    age: 21

    major: 'Computer Science'

    email: 'raissa@gmail.com'



3 - Imagem da Lista de Estudantes. Fonte: Os Autores.

Ademais, o banco de dados conta com os detalhes das atividades da UNIFESP, sendo assim:

- **Chave:** activity_name
  - **Tipo:** Hash
  - **Campos e Valores:**

    subject: Data Structures

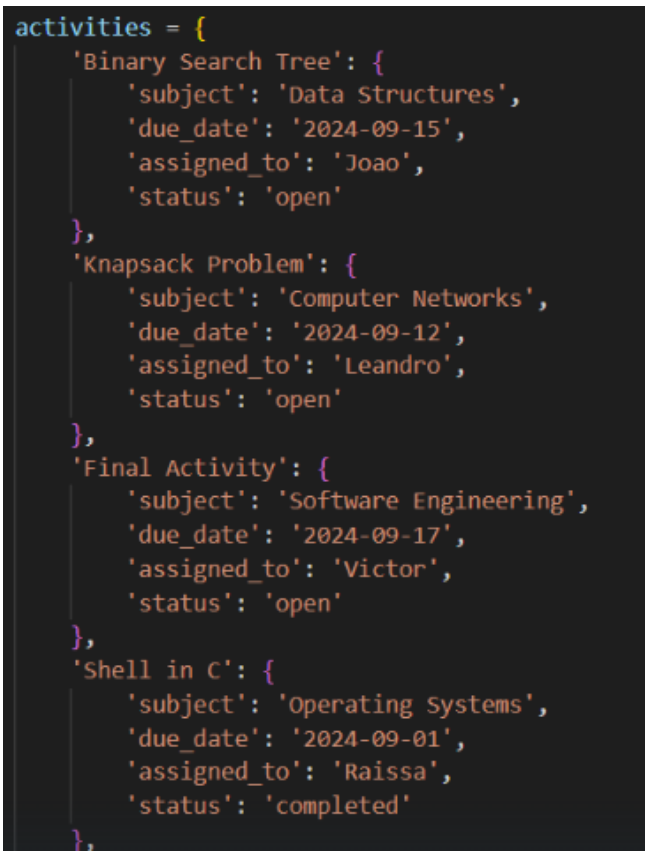
    due_date: ‘2024-09-04’

    assigned_to: ‘Victor’

    status: ‘open’
- **Chave:** activities
  - **Tipo:** Lista
  - **Valores:** Nomes das atividades [ex:” 'Binary Search Tree', 'Knapsack Problem', 'Final Activity', 'Shell in C', 'Knn', 'MariaDB’"]

Por fim, temos os dados dos cursos:

- **Chave:** courses
  - **Tipo:** Lista
  - **Valores:** Lista de todos os cursos únicos dos estudantes [ex: “'Algorithms', 'Operating Systems', 'Database Systems', 'Artificial Intelligence', 'Data Structures', 'Computer Networks'"]



4 - Imagem das Atividades dos Estudantes. Fonte: Os Autores.

3. Implementação

No projeto, utilizamos o arquivo app.py como ponto central de conexão e comunicação. Inicialmente, o app.py estabelece uma conexão com o banco de dados Redis, popula-o com os dados necessários e, em seguida, conecta-se ao front-end utilizando o Flask.

Quando o usuário realiza uma ação no front-end, como clicar em um botão que dispara uma das funções implementadas, uma requisição é enviada para uma rota específica no back-end. Essa rota, que está associada a uma função Python, executa a query correspondente no Redis. Após a execução, a informação resultante é retornada ao front-end, onde é exibida para o usuário.

Ao todo foram 1 query de inserção e 10 de busca de dados implementadas ao sistema, sendo elas:

1. Adicionar Estudante à Lista

- Descrição: Adiciona um estudante à lista `students` no Redis.
- Função que a executa: `set_value('nome_do_estudante') '

2. Visualização Geral de Dados

- Descrição: Retorna todas as chaves armazenadas no Redis.
- Função que a executa: `view_all()`

3. Obter Estudantes

- Descrição: Retorna a lista de todos os estudantes.
- Função que a executa: `get_students()`

4. Obter Progresso do Estudante

- Descrição: Retorna o progresso do estudante (atividades completas e incompletas).
- Função que a executa: `get_student_course_progress(student_name)`

5. Atividades Futuras de Estudantes

- Descrição: Varre todas as atividades e verifica quais possuem datas de vencimento futuras.
- Função que a executa: `get_students_with_upcoming_activities()`

6. Cursos com Atividades Incompletas

- Descrição: Varre todas as atividades e verifica quais ainda não estão concluídas, retornando os cursos que possuem atividades incompletas.
- Função que a executa: `get_courses_with_incomplete_activities()`

7. Média de Idade por Curso

- Descrição: Calcula a média de idade dos estudantes por curso.
- Função que a executa: `calculate_average_age_per_course()`

8. Atividades Atrasadas por Estudante

- Descrição: Varre todas as atividades para identificar as que estão atrasadas (data de vencimento no passado e não concluídas).
- Função que a executa: `find_overdue_activities_per_student()`

9. Top 3 Estudantes de Ciência da Computação por Atividades Concluídas

- Descrição: Conta o número de atividades concluídas por estudante no curso de Ciência da Computação e retorna os três alunos com mais atividade concluídas.
- Função que a executa: `top_3_cs_students_by_completed_activities()`

10. Curso com Maior Proporção de Atividades Atrasadas

- Descrição: Calcula a proporção de atividades atrasadas para cada curso e retorna o curso com a maior proporção de atrasos.
- Função que a executa: `course_with_highest_overdue_ratio()`

11. Atividade Mais Atrasada por Curso e Número de Estudantes Atrasados

- Descrição: Retorna a atividade mais atrasada em cada curso e o número de estudantes atrasados para essa atividade.
- Função que a executa: `most_overdue_activity_per_course_and_num_students()`

5 - Aplicação funcionando. Fonte: Os Autores.

4. Conclusões Finais

A partir desse trabalho foi possível observar que o Redis garante flexibilidade, uma vez que trabalha com estruturas como listas e hashes, o que é adequado para armazenar e consultar informações de estudantes, cursos e atividades de maneira eficiente. O sistema desenvolvido gerencia dados acadêmicos da UNIFESP, oferecendo um conjunto de funcionalidades, incluindo a adição de estudantes, monitoramento de atividades, cálculo de médias por curso, além da detecção de atividades atrasadas. As queries implementadas atenderam as necessidades do sistema, tendo em vista que o Redis permite consultas rápidas e operações simultâneas. Portanto, o uso do Redis mostrou-se adequado para o

gerenciamento de dados de chave-valor no contexto acadêmico, e a integração com Flask para a construção do back-end garantiu a comunicação eficiente entre banco de dados e interface web, resultando em uma aplicação ágil e funcional.