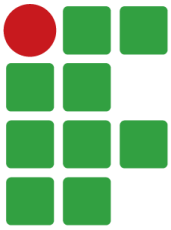


INSTITUTO FEDERAL
Minas Gerais
Campus Formiga

DOCUMENTAÇÃO TÉCNICA

Sistema hotel

Ryan William Fonseca

Mateus Coutinho Ferreira

Formiga - Minas Gerais

2021

RESUMO

Este documento visa descrever os detalhes envolvidos na programação do sistema, desde o planejamento até implementação. É citado o padrão de projeto utilizado, bem como a análise das entidades relevantes para posterior elaboração do diagrama de entidade-relacionamento. A estrutura de arquivos do projeto e funções utilitárias também são abordadas.

SUMÁRIO

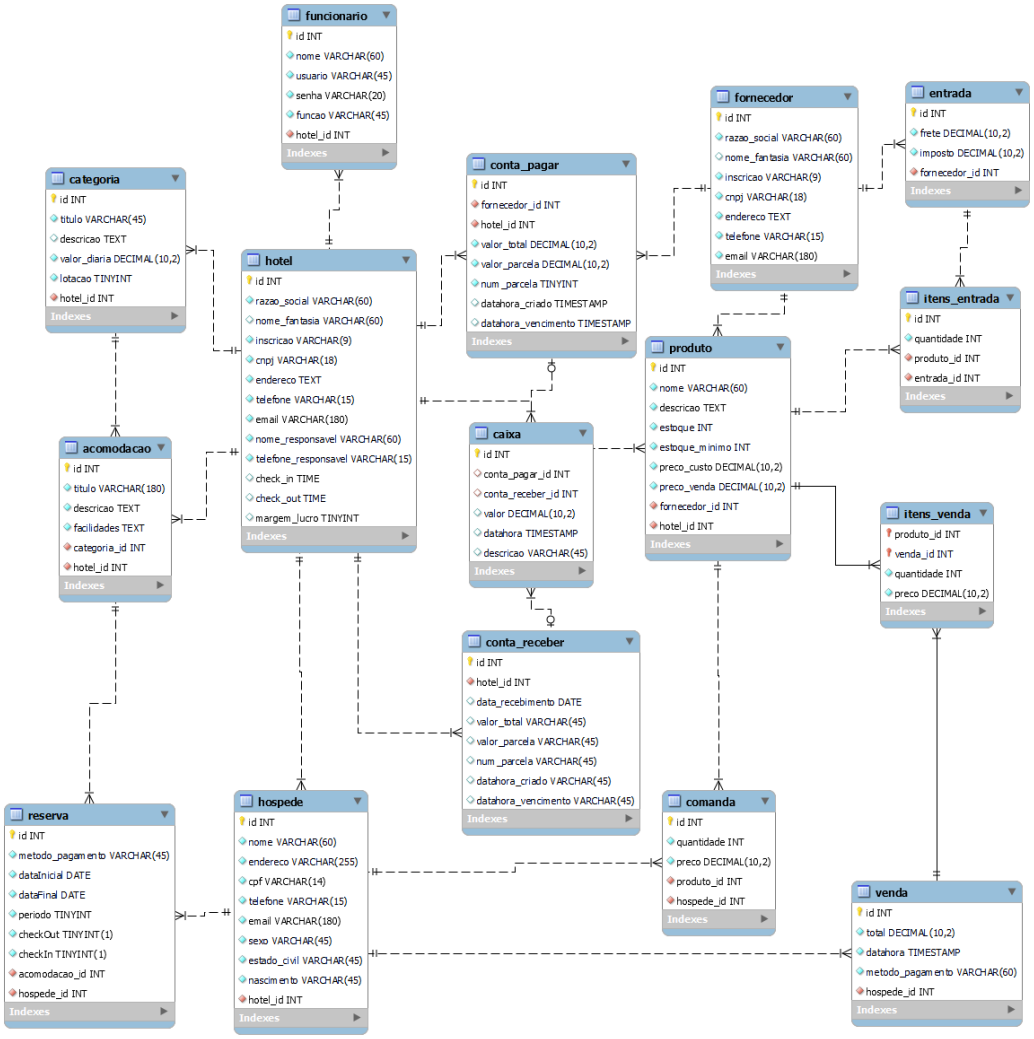
1. Padrão de projeto adotado:

MVC é o acrônimo para Model-View-Controller, que em português pode ser traduzido para Modelo-Visão-Controlador. Corresponde a um padrão de arquitetura que divide o software em três camadas interconectadas: uma para apresentação dos dados (Visão), uma para manipulação dos dados (Modelo) e uma para comunicação entre as camadas anteriores (Controlador). Tornou-se especialmente popular no desenvolvimento de aplicações para web, mas também é utilizado em projetos mobile e desktop.

2. Modelagem de dados:

Foi feita uma análise das entidades e fenômenos relevantes no contexto de um hotel e com base no enunciado do trabalho para posterior elaboração do modelo conceitual de entidade relacionamento, também conhecido como MER. Esse modelo documentado em um diagrama nos traz a representação gráfica das tabelas e seus atributos, bem como o relacionamento com outras entidades.

O diagrama de entidade-relacionamento proporciona uma melhor visualização da estrutura de dados, facilitando a discussão de ideias e tomada de decisões. A visão completa do diagrama é representada na figura 1.



3. Estrutura do projeto

Em seguida, é detalhado como o projeto foi organizado em arquivos e funções relevantes do sistema.

3.1 Rotas

O arquivo de rotas registra os menus principais do sistema, indicando o caminho para realizar cada operação.

3.2 Controllers

Os controladores organizam a lógica de cadastro, visualização, edição, remoção e relatório de dados de uma entidade num mesmo arquivo.

3.3 Modelos

3.3.1 Tables

Esses tipos foram criados para fornecer ao programa, em tempo de execução, ciência acerca das estruturas de dados usadas para armazenar os registros das tabelas, afim de permitir a escrita de funções mais genéricas por parte do programador.

O tipo Table é um vetor de ponteiros terminado em NULL: o primeiro ponteiro aponta para um TableState, o segundo para um TableInfo, e os demais para ColumnMetas. É importante notar que Table não armazena os registros da tabela, mas dados que permitem recuperar tais registros.

A struct TableState armazena informações geradas e usadas pela própria implementação, como o tamanho total de cada registro, posição do ponteiro no arquivo, etc.

A struct TableInfo armazena informações sobre a tabela, como o nome do arquivo, a tag XML que a representa, etc.

As structs ColumnMeta armazenam informações sobre cada coluna da tabela, como o tamanho/alinhamentos do campo na struct, o rótulo exibido para o usuário, a tag XML que a representa, flags que conferem algum comportamento específico, etc.

3.3.2 Database

Esse tipo foi criado afim de mimicar o que é uma 'Interface' numa linguagem orientada a objetos, de modo a armazenar tanto as funções do armazenamento binário quanto do armazenamento XML, de modo padronizado, para que o programador que esteja usando as funções de armazenamento não precise se preocupar com detalhes internos.

As funções definidas são as mais simples possíveis, como 'open' e 'close' para abrir e fechar o arquivo da tabela, 'rewind' para voltar o cursor ao início do arquivo, 'next' para recuperar o próximo registro e avançar o cursor (como num Iterator), 'insert' para inserir um registro e mover o cursor para o fim do arquivo, 'update' para atualizar o registro anterior ao cursor, e 'delete' para apagá-lo.

Munido destas funções e dos metadados pôde ser construído funções mais avançadas como 'DATABASE_forEach' e 'DATABASE_findBy', de forma genérica, sem ter de se importar com detalhes da implementação do armazenamento binário e XML.

4. Funções utilitárias

4.1 Form

Exibe um formulário com os campos de acordo com os metadados das colunas das tabelas.

Sintaxe:

```
form(2, Reservas, &res);
```

Parâmetros:

- **Mode:** define a operação realizada pelo formulário, em que 0 = inserir, 1 = visualizar, 2 = editar.
- **Table:** trata-se da tabela que contém as colunas
- **Ptr:** ponteiro para struct onde os dados serão lidos e/ou armazenados.

Retorno:

Um array de valores booleanos em que cada índice é um campo do form indicando se este foi preenchido.

O método form fui extensivamente utilizado no módulo de cadastro para realizar as operações de inserção, edição e visualização dos dados, evitando a repetição de código.

4.2 Menu

Exibe um menu com opções selecionáveis que permite que o usuário navegue pressionando TAB e selecione um item clicando em ENTER.

Sintaxe:

```
int option = menu($f, 3, "Incluir", "Próximo", "Finalizar venda");
```

Parâmetros:

- **nItems:** indica a quantidade de opções do menu
- **options:** lista dinâmica de strings que serão exibidas como opções selecionáveis do menu

Retorno:

Retorna o índice da opção selecionada

4.3 Printval

Faz a leitura de dados com base nos metadados da coluna informada.

O método menu foi utilizado em todas as operações para construir as interfaces interagíveis.

Sintaxe:

```
readVal(stdin, '\n', &(ColumnMeta) {.type = COL_TYPE_UINT}, &quantidade);
```

Parâmetros:

- **stream:** Arquivo ou dispositivo de onde será feita a leitura dos dados
- **delimiter:** Caractere que determina o final da leitura
- **colMeta:** Metadados da coluna relativa ao valor sendo lido
- **ptr:** Ponteiro para a variável onde o valor será armazenado

Retorno:

Retorna a quantidade de caracteres impressos

4.4 Readval

Lê um valor com base nos metadados da coluna informada

Sintaxe:

```
printVal(stdout, colMeta, ptr);
```

Parâmetros:

- **stream:** Arquivo ou dispositivo de onde será feita a leitura dos dados
- **delimiter:** Caractere que determina o final da leitura
- **colMeta:** Metadados da coluna relativa ao valor sendo lido
- **ptr:** Ponteiro para a variável onde o valor será armazenado

Retorno:

Retorna um valor booleano que indica se a operação de leitura foi bem sucedida

4.5 compareFields

Verifica se os registros passados como parâmetros são iguais

Sintaxe:

```
compareFields(Hospedes, &hosp, &tempHosp, filtroHospede);
```

Parâmetros:

- **table:** A tabela que contém as colunas dos registros sendo comparados
- **oneReg:** O primeiro registro a ser comparado
- **otherReg:** O segundo registro a ser comparado
- **fieldsToFilter:** Um array de boolean que indicam quais campos devem ser comparados

Retorno:

Retorna um valor booleano que indica se os campos são iguais