João Marcelo de Jesus Macedo
Matheus Reis de Souza Teixeirense
Matheus Sena Vasconcelos

Sistema hoteleiro - hospedar.com

Sistema de informação - 05 / 2020

Administração de Banco de Dados

Professor Roger Oliveira

Sumário

Sobre

Nome do projeto

Hospedar.com

Objetivo

Desenvolver um sistema web para controle de atividades hoteleiras, onde seja possível o cadastro dos quartos e seus valores, formas de pagamento, funcionários atuantes e os hóspedes do hotel. Que seja possível, também, a disponibilização de relatórios, podendo ser em arquivos, telas ou dashboards. Além disso, oferecer ao usuário do sistema um controle das reservas das hospedagens mais simples, rápida e consistente, trazendo uma visualização prévia dos quartos e hóspedes disponíveis.

Como usar

Depois de configurar e subir o sistema em um servidor, é necessário primeiro cadastrar seus funcionários, quartos, valores das diárias e formas de pagamentos disponíveis.

Quando o sistema estiver populado com esses dados básico, já pode-se usar em seu hotel. Basta cadastrar os novos hóspedes que chegarem em seu hotel e vincular esses dados à uma hospedagem.

Banco de dados utilizado

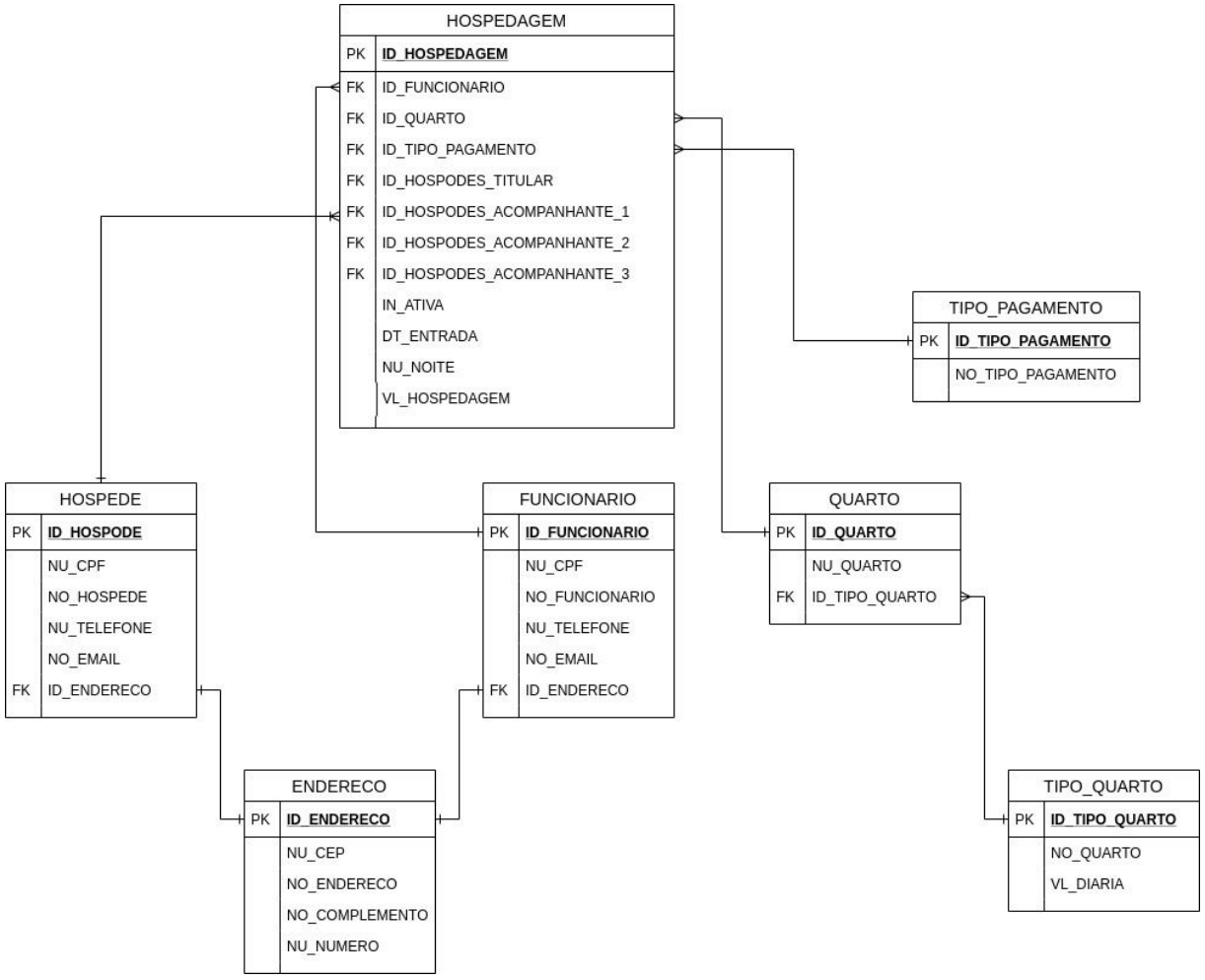
SQLite3

Links

O código de todo o projeto, desde o Banco de Dados ao back-end em Python3 e o front-end em HTML e Jinja2, podem ser visualizados no repositório público do GitHub https://github.com/senavs/website-hosting.

Estrutura do projeto

Modelo de Entidade e Relacionamento (MER)



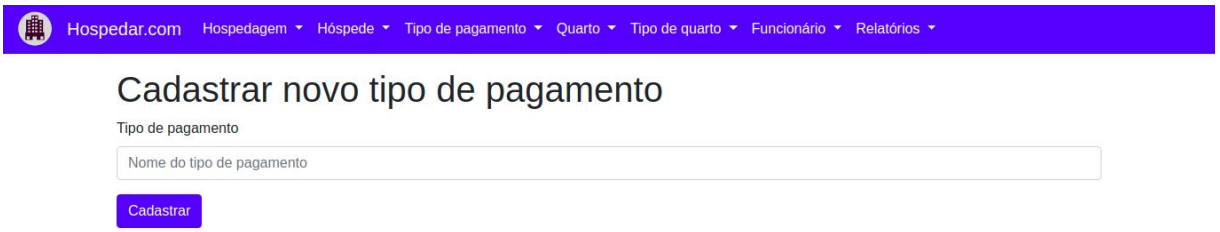
MER do hospedar.com

Tabelas, índices e suas telas

Para começar os scripts de criação das tabelas, devemos começar pelas entidades que não possuem chaves estrangeiras, ou seja, as tabelas de domínio.

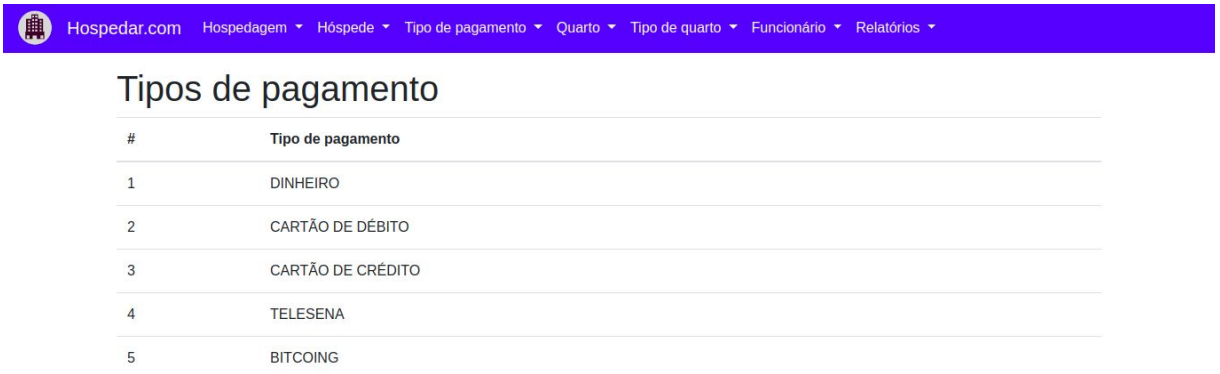
Tipo de pagamento

Tipo de pagamento é uma tabela assim. Criada simplesmente para armazenar os possíveis tipos de pagamento que o hotel pode aceitar.




```
CREATE TABLE IF NOT EXISTS "TIPO_PAGAMENTO" (
        "ID_TIPO_PAGAMENTO" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        "NO_TIPO_PAGAMENTO" VARCHAR(16) NOT NULL UNIQUE
);
```

Como o próprio nome do tipo de pagamento é único, igual à chave primária, então é possível transformá-la em índice para facilitar a consulta.

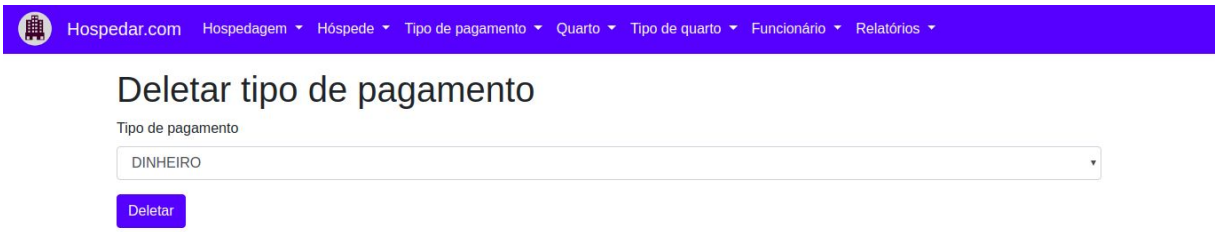



```
CREATE UNIQUE INDEX IF NOT EXISTS "IX_NOME_TIPO_PAGAMENTO" ON "TIPO_PAGAMENTO" (
        "NO_TIPO_PAGAMENTO"
);
```

Para popular a tabela dos tipos de pagamento, o seguinte código DML foi utilizado:

```
INSERT INTO "TIPO_PAGAMENTO" ("NO_TIPO_PAGAMENTO")
VALUES ("DINHEIRO");

INSERT INTO "TIPO_PAGAMENTO" ("NO_TIPO_PAGAMENTO")
VALUES ("CARTÃO DE DÉBITO");

INSERT INTO "TIPO_PAGAMENTO" ("NO_TIPO_PAGAMENTO")
VALUES ("CARTÃO DE CRÉDITO");

INSERT INTO "TIPO_PAGAMENTO" ("NO_TIPO_PAGAMENTO")
VALUES ("TELESENA");

INSERT INTO "TIPO_PAGAMENTO" ("NO_TIPO_PAGAMENTO")
VALUES ("BITCOING");
```

Além disso, é possível deletar um tipo de pagamento previamente cadastrado:



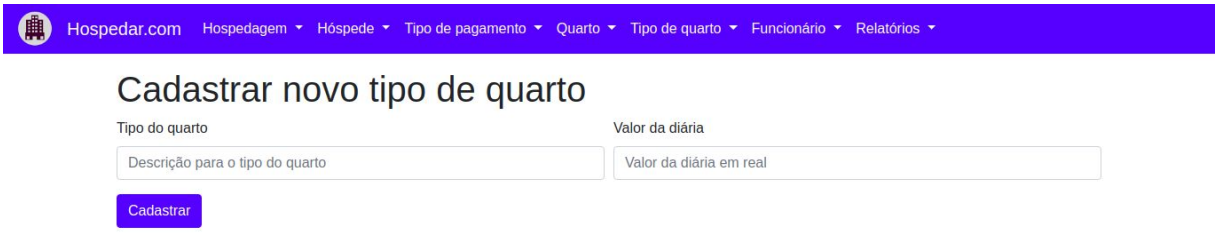

```
DELETE FROM "TIPO_PAGAMENTO"
WHERE "ID_TIPO_PAGAMENTO" = <ID_TIPO_PAGAMENTO>;
```

Para mostrar todos os tipos de pagamentos disponíveis para deletar, é necessário fazer uma consulta que busque todos os dados dessa tabela.

```
SELECT ID_TIPO_PAGAMENTO,
       NO_TIPO_PAGAMENTO
FROM TIPO_PAGAMENTO;
```

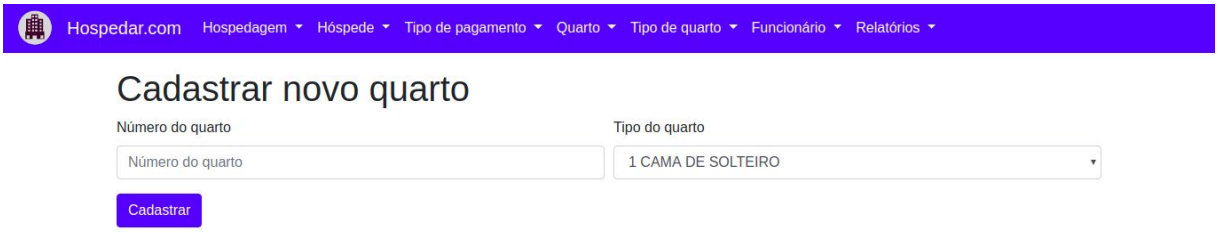
Quarto e tipo do quarto

Para cadastro do quarto é preciso de duas tabelas. A primeira é a tabela domínio para armazenar os tipos de quarto e seus respectivos valores.




```
CREATE TABLE IF NOT EXISTS "TIPO_QUARTO" (
        "ID_TIPO_QUARTO" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        "NO_TIPO_QUARTO" VARCHAR(16) NOT NULL UNIQUE,
        "VL_DIARIA" REAL NOT NULL
);
```

A outra tabela seria o próprio quarto, sendo necessário armazenar o número do quarto e vinculá-lo com o tipo do quarto através da chave estrangeira.

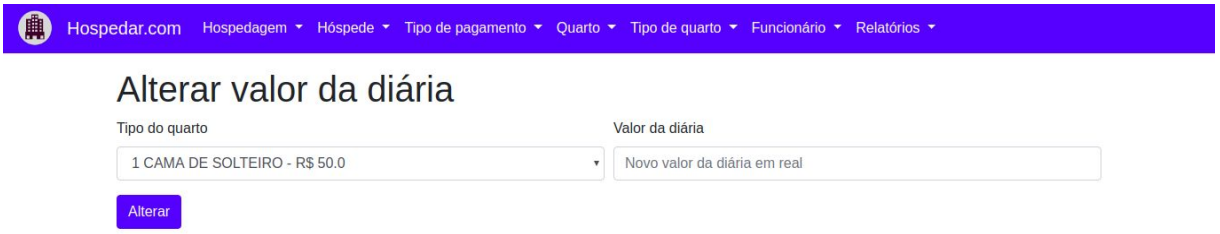



```
CREATE TABLE IF NOT EXISTS "QUARTO" (
        "ID_QUARTO" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        "NU_QUARTO" INTEGER NOT NULL UNIQUE,

        "ID_TIPO_QUARTO" INTEGER NOT NULL,

        FOREIGN KEY ("ID_TIPO_QUARTO") REFERENCES "TIPO_QUARTO" ("ID_TIPO_QUARTO")
);
```

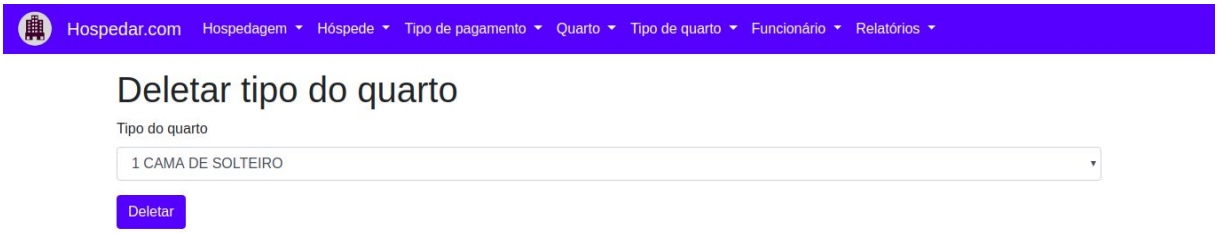
Do mesmo jeito do tipo de pagamento, a tabela quarto possui a coluna Número do Quarto que contém valores únicos, podendo ser criados índices.


```
CREATE UNIQUE INDEX IF NOT EXISTS "IX_QUARTO_NUMERO" ON "QUARTO" (
        "NU_QUARTO"
);
```

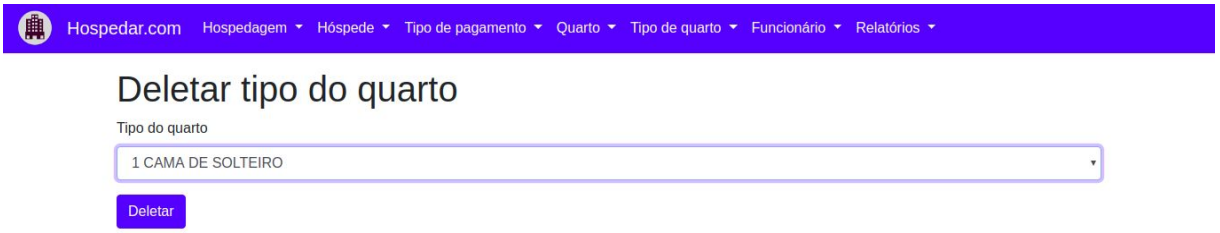
Além dessas telas, ainda possui telas para alterar o valor da diária e deletar o quarto e o tipo de quarto.




```
UPDATE "TIPO_PAGAMENTO"
SET "VL_DIARIA" = <NOVO_VALOR_DIAROA>
WHERE "ID_TIPO_PAGAMENTO" = <ID_TIPO_PAGAMENTO>
```




```
DELETE FROM "QUARTO"
WHERE "NU_QUARTO" = <NU_QUARTO>;
```




```
DELETE FROM "TIPO_QUARTO"
WHERE "ID_TIPO_QUARTO" = <ID_TIPO_QUARTO>;
```

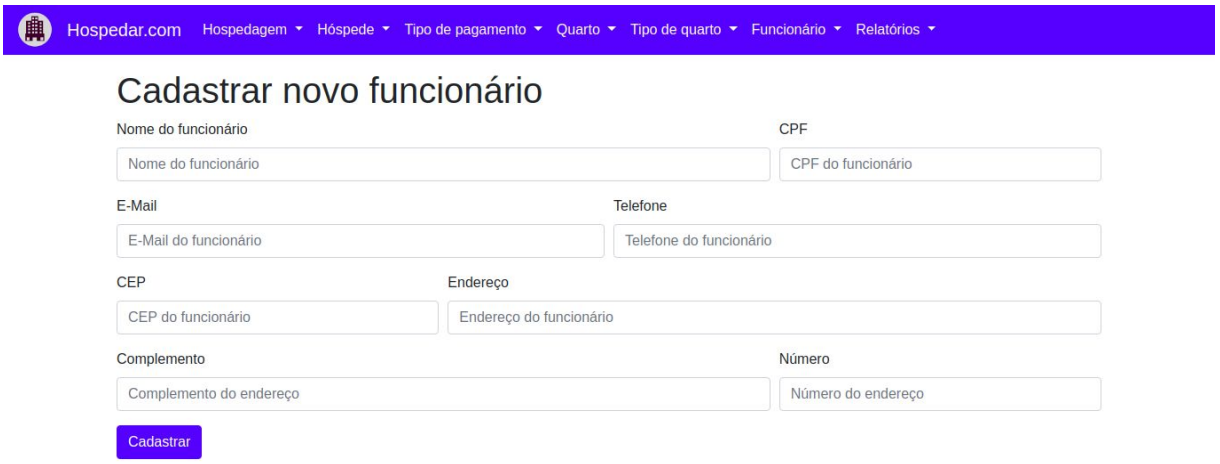
Nessas últimas telas acima, é necessário fazer uma consulta para buscar todos os dados presentes nas tabelas quarto e tipo do quarto e preencher o dropdown com os quartos e tipos disponíveis.

```
SELECT T1.ID_QUARTO,
       T1.NU_QUARTO,
       T2.ID_TIPO_QUARTO,
       T2.NO_TIPO_QUARTO
FROM QUARTO T1
LEFT JOIN TIPO_QUARTO T2;
```

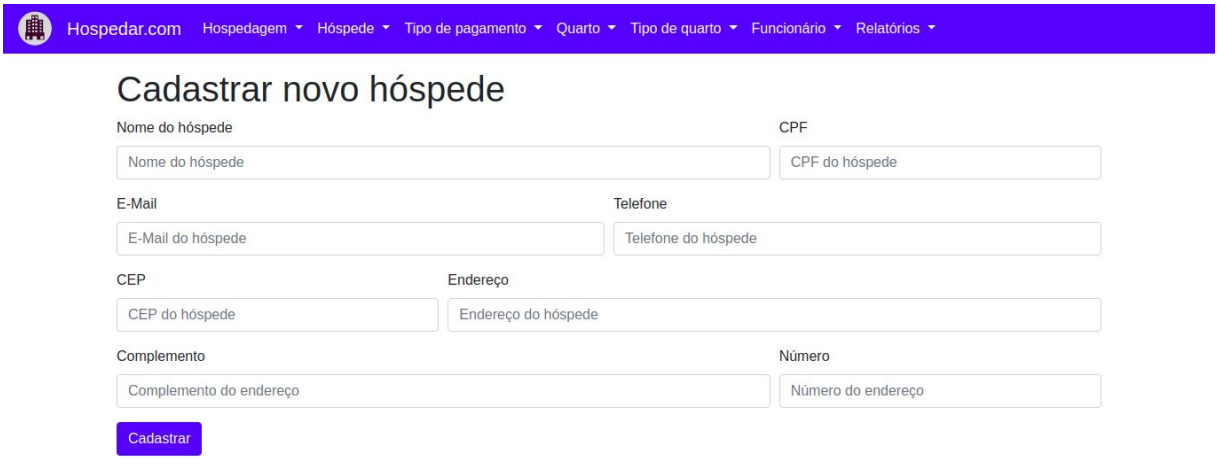
Hóspedes, funcionários e seus endereços

No MER, observa-se que os Hóspedes e Funcionários estão conectados à mesma tabela: Endereço. Então, já que essas duas tabelas possuem chaves estrangeiras da tabela endereço, é necessário construí-la primeiro e, logo em seguida, as tabelas de funcionário e hóspede.

```
CREATE TABLE IF NOT EXISTS "ENDERECO" (
        "ID_ENDERECO" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        "NU_CEP" CHAR(8) NOT NULL,
        "NO_ENDERECO" VARCHAR(64) NOT NULL,
        "NO_COMPLEMENTO" VARCHAR(32) NOT NULL,
        "NU_NUMERO" INTEGER NOT NULL
);
```




```
CREATE TABLE IF NOT EXISTS "FUNCIONARIO" (
        "ID_FUNCIONARIO" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        "NU_CPF" CHAR(11) NOT NULL UNIQUE,
        "NO_FUNCIONARIO" VARCHAR(64) NOT NULL,
        "NU_TELEFONE" CHAR(11) NOT NULL UNIQUE,
        "NO_EMAIL" VARCHAR(32) NOT NULL UNIQUE,
        "ID_ENDERECO" INTEGER NOT NULL,

        FOREIGN KEY ("ID_ENDERECO") REFERENCES "ENDERECO" ("ID_ENDERECO")
);
```



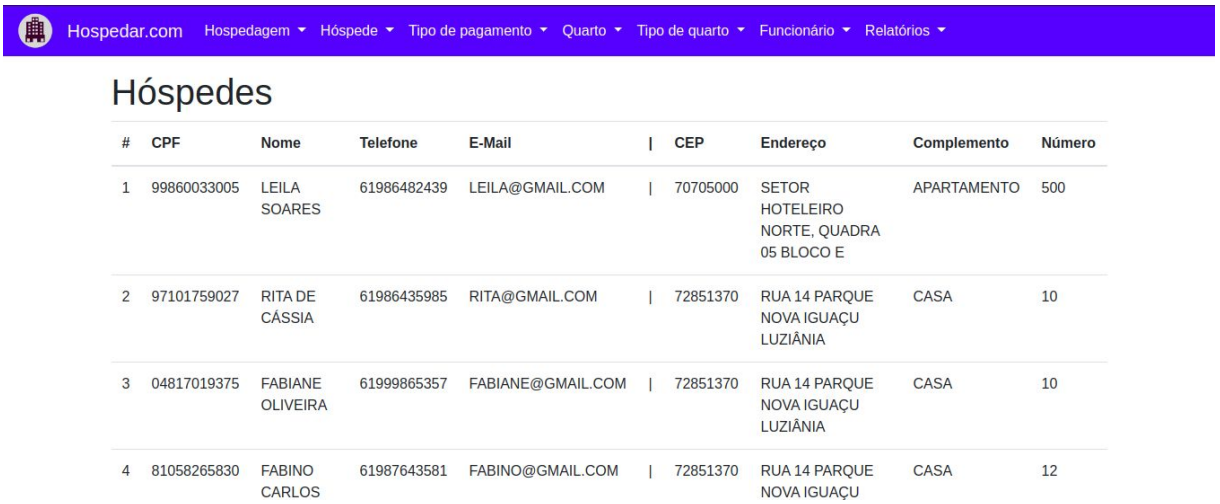
```
CREATE TABLE IF NOT EXISTS "HOSPEDE" (
        "ID_HOSPEDE" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        "NU_CPF" CHAR(11) NOT NULL UNIQUE,
        "NO_HOSPEDE" VARCHAR(64) NOT NULL,
        "NU_TELEFONE" CHAR(11) NOT NULL UNIQUE,
        "NO_EMAIL" VARCHAR(32) NOT NULL UNIQUE,
        "ID_ENDERECO" INTEGER NOT NULL,

        FOREIGN KEY ("ID_ENDERECO") REFERENCES "ENDERECO" ("ID_ENDERECO")
);
```

Nessas duas telas, em cada uma, é necessário fazer duas inserções no banco de dados. Primeiramente, é necessário cadastrar o endereço do hóspede/funcionário. Caso a inserção ocorra com sucesso, podemos inserir a pessoa passando a chave estrangeira do endereço. Caso uma das duas inserções falhem, é necessário fazer o rollback.

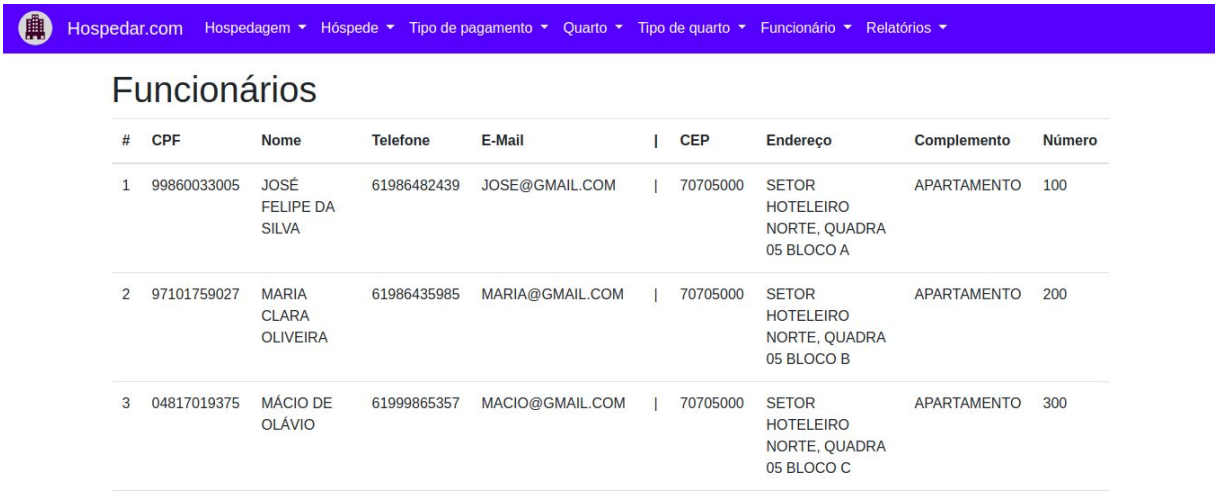
```
INSERT INTO "ENDERECO" ("NU_CEP", "NO_ENDERECO", "NO_COMPLEMENTO", "NU_NUMERO")
VALUES ("70705000", "SETOR HOTELEIRO NORTE, QUADRA 05 BLOCO A", "APARTAMENTO", 100);
INSERT INTO "FUNCIONARIO" ("NU_CPF", "NO_FUNCIONARIO", "NU_TELEFONE", "NO_EMAIL", "ID_ENDERECO")
VALUES ("99860033005", "JOSÉ FELIPE DA SILVA", 61986482439, "JOSE@GMAIL.COM", 1);
```

```
INSERT INTO "ENDERECO" ("NU_CEP", "NO_ENDERECO", "NO_COMPLEMENTO", "NU_NUMERO")
VALUES ("72851370", "RUA 14 PARQUE NOVA IGUAÇU LUZIÂNIA", "CASA", 10);
INSERT INTO "HOSPEDE" ("NU_CPF", "NO_HOSPEDE", "NU_TELEFONE", "NO_EMAIL", "ID_ENDERECO")
VALUES ("99860033005", "LEILA SOARES", 61986482439, "LEILA@GMAIL.COM", 6);
```

Do mesmo jeito das outras tabelas listadas acima, as tabelas Hóspede e Funcionários possuem colunas de valores únicos, como por exemplo o CPF. Então, para uma maior velocidade das consultas à essas tabelas, podemos criar índices.



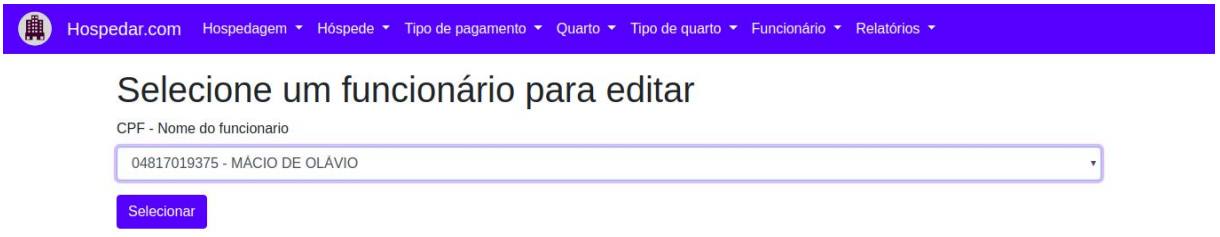
```
CREATE UNIQUE INDEX IF NOT EXISTS "IX_HOSPEDE_CPF" ON "HOSPEDE" (
        "NU_CPF"
);
```



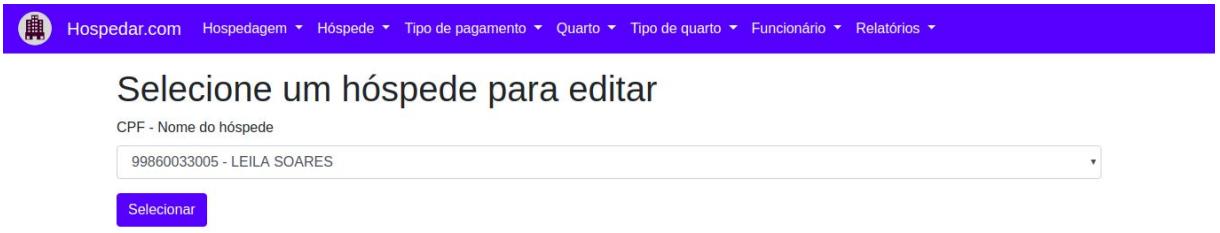
#	CPF	Nome	Telefone	E-Mail	|	CEP	Endereço	Complemento	Número
1	99860033005	JOSÉ FELIPE DA SILVA	61986482439	JOSE@GMAIL.COM	|	70705000	SETOR HOTELEIRO NORTE, QUADRA 05 BLOCO A	APARTAMENTO	100
2	97101759027	MARIA CLARA OLIVEIRA	61986435985	MARIA@GMAIL.COM	|	70705000	SETOR HOTELEIRO NORTE, QUADRA 05 BLOCO B	APARTAMENTO	200
3	04817019375	MÁCIO DE OLÁVIO	61999865357	MACIO@GMAIL.COM	|	70705000	SETOR HOTELEIRO NORTE, QUADRA 05 BLOCO C	APARTAMENTO	300

```
CREATE UNIQUE INDEX IF NOT EXISTS "IX_FUNCIONARIO_CPF" ON "FUNCIONARIO" (
        "NU_CPF"
);
```

Para alterar os dados dos funcionários e dos hóspedes, é necessário também alterar a tabela de endereço. Primeiramente, Listamos todos as pessoas disponíveis para essa alteração, fazer uma busca de todos os registros dentro do banco de dados.

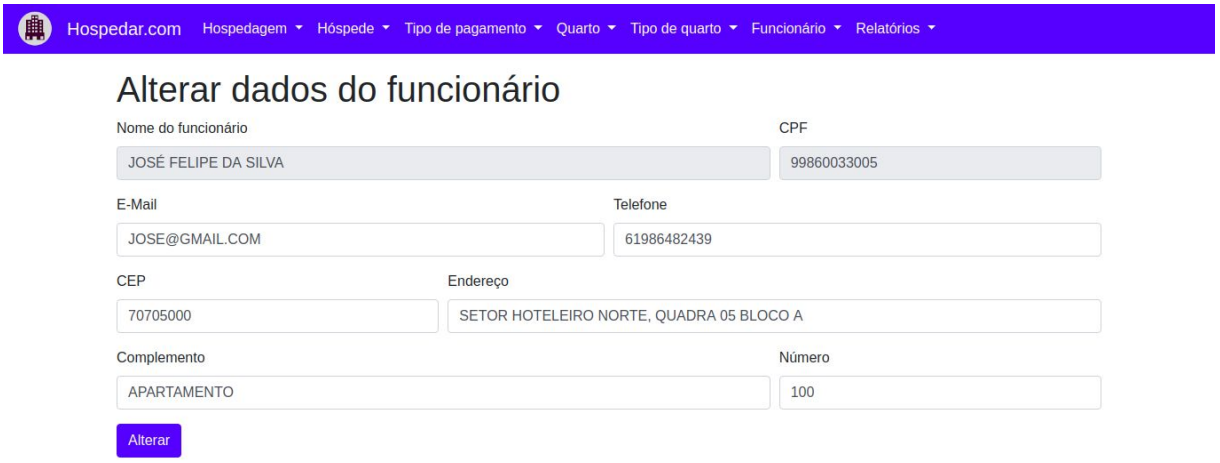



```
SELECT ID_FUNCIONARIO,
       NO_FUNCIONARIO,
       NU_CPF
FROM FUNCIONARIO;
```

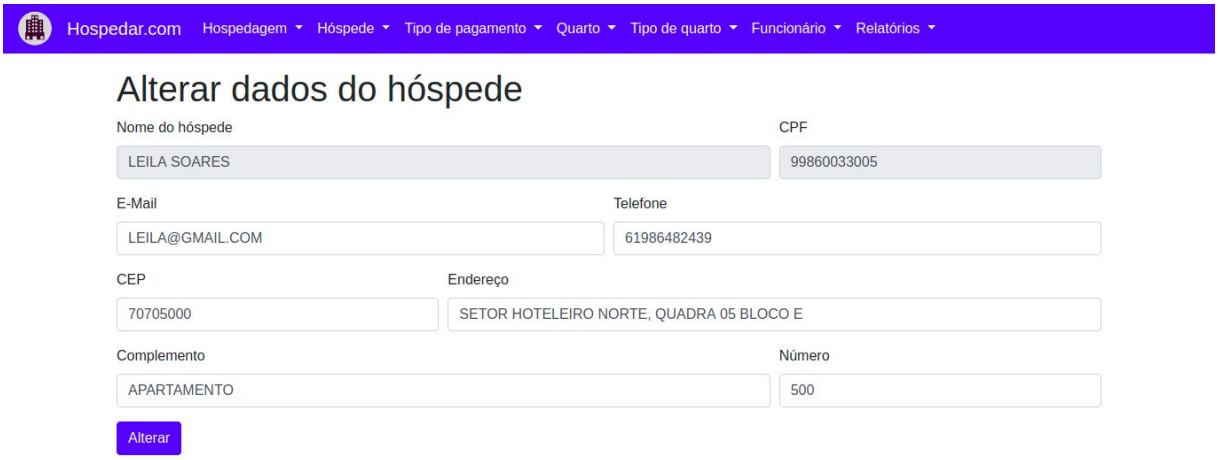



```
SELECT ID_HOSPEDE,
       NO_HOSPEDE,
       NU_CPF
FROM HOSPEDE;
```

Depois de selecionado, outra busca é realizada, porém juntando com a tabela de endereço e filtrando pelo hóspede/funcionário selecionado. Com o resultado da consulta é possível construir uma tela que disponibiliza esses dados consultados e preenchidos. Caso o usuário selecione o botão alterar, todos os dados são alterados.

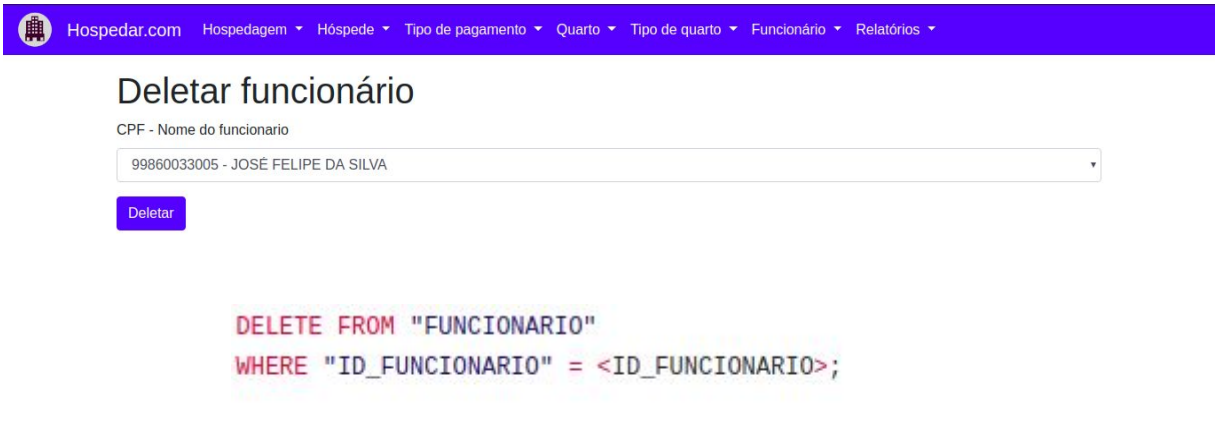



```
UPDATE "FUNCIONARIO"
SET "NO_EMAIL" = <NOVO_EMAIL>,
    "NU_TELEFONE" = <NOVO_TELEFONE>
WHERE "ID_FUNCIONARIO" = <ID_FUNCIONARIO>

UPDATE "ENDERECO"
SET "NO_CEP" = <NOVO_CEP>,
    "NO_ENDERECO" = <NOVO_ENDERECO>,
    "NO_COMPLEMENTE" = <NOVO_COMPLEMENTO>,
    "NU_NUMERO" = <NOVO_NUMERO>
WHERE "ID_ENDERECO" = <ID_ENDERECO>
```

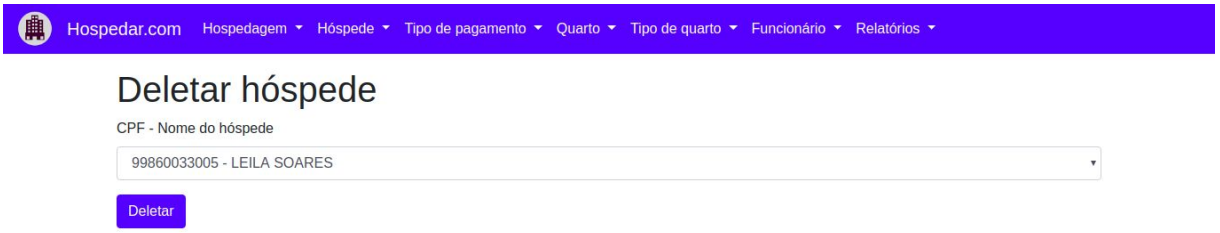



```
UPDATE "HOSPEDE"
SET "NO_EMAIL" = <NOVO_EMAIL>,
    "NU_TELEFONE" = <NOVO_TELEFONE>
WHERE "ID_HOSPEDE" = <ID_HOSPEDE>

UPDATE "ENDERECO"
SET "NO_CEP" = <NOVO_CEP>,
    "NO_ENDERECO" = <NOVO_ENDERECO>,
    "NO_COMPLEMENTE" = <NOVO_COMPLEMENTO>,
    "NU_NUMERO" = <NOVO_NUMERO>
WHERE "ID_ENDERECO" = <ID_ENDERECO>
```

Também é possível deletar as pessoas do sistema. Para deletar um hóspede/funcionário, também é necessário deletar seu endereço.




```
DELETE FROM "FUNCIONARIO"
WHERE "ID_FUNCIONARIO" = <ID_FUNCIONARIO>;

DELETE FROM "ENDERECO"
WHERE "ID_ENDERECO" = <ID_ENDERECO_DO_FUNCIONARIO>;
```



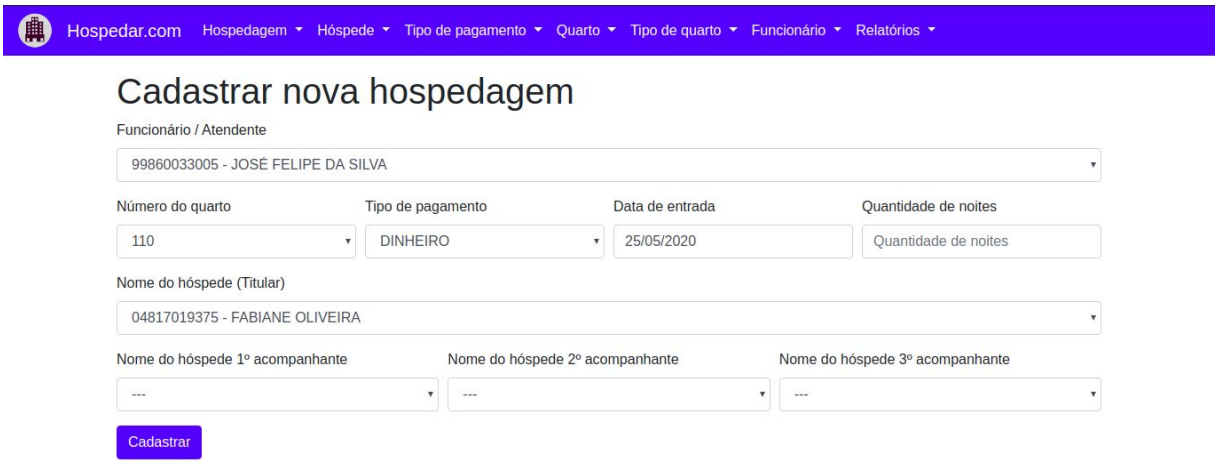

```
DELETE FROM "HOSPEDE"
WHERE "ID_HOSPEDE" = <ID_HOSPEDE>;

DELETE FROM "ENDERECO"
WHERE "ID_ENDERECO" = <ID_ENDERECO_DO_HOSPEDE>;
```

Hospedagem

A tabela Hospedagem, que é o centro da aplicação, armazena todas as chaves primárias das outras tabelas. É necessário guardar a chave primária do funcionário que efetuou a reserva, a chave primária do quarto e do tipo de pagamento selecionado e as chaves primárias dos hóspedes que estarão na reserva. Os hóspedes acompanhantes não são obrigatórios, pois, para reservar um quarto, basta um hóspede. Além disso, é necessário armazenar se a atual hospedagem está ativa (o valor default é 1), a data que a reserva foi criada e quantas noites os hóspedes vão ficar. O valor da hospedagem não é necessário preencher, pois será preenchido pela trigger mais para frente.

```sql
CREATE TABLE IF NOT EXISTS "HOSPEDAGEM" (
        "ID_HOSPEDAGEM" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        "ID_FUNCIONARIO" INTEGER NOT NULL,
        "ID_QUARTO" INTEGER NOT NULL,
        "ID_TIPO_PAGAMENTO" INTEGER NOT NULL,
        "ID_HOSPEDE_TITULAR" INTEGER NOT NULL,
        "ID_HOSPEDE_ACOMPANHANTE_1" INTEGER,
        "ID_HOSPEDE_ACOMPANHANTE_2" INTEGER,
        "ID_HOSPEDE_ACOMPANHANTE_3" INTEGER,
        "IN_ATIVO" BOOLEAN NOT NULL DEFAULT 1, -- BOOLEAN 0/1
        "DT_ENTRADA" CHAR(10) NOT NULL,
        "NU_NOITE" INTEGER NOT NULL,
        "VL_HOSPEDAGEM" REAL,

        FOREIGN KEY ("ID_FUNCIONARIO") REFERENCES "FUNCIONARIO" ("ID_FUNCIONARIO"),
        FOREIGN KEY ("ID_QUARTO") REFERENCES "QUARTO" ("ID_QUARTO"),
        FOREIGN KEY ("ID_HOSPEDE_TITULAR") REFERENCES "HOSPEDE" ("ID_HOSPEDE"),
        FOREIGN KEY ("ID_HOSPEDE_ACOMPANHANTE_1") REFERENCES "HOSPEDE" ("ID_HOSPEDE"),
        FOREIGN KEY ("ID_HOSPEDE_ACOMPANHANTE_2") REFERENCES "HOSPEDE" ("ID_HOSPEDE"),
        FOREIGN KEY ("ID_HOSPEDE_ACOMPANHANTE_3") REFERENCES "HOSPEDE" ("ID_HOSPEDE"),
        FOREIGN KEY ("ID_TIPO_PAGAMENTO") REFERENCES "TIPO_PAGAMENTO" ("ID_TIPO_PAGAMENTO")
);
```

Para a tela de cadastrar hospedagem, algumas consultas são realizadas para preencher os dropdown.



Primeiramente, trazer todos os funcionários cadastrado no sistema:

```sql
SELECT ID_FUNCIONARIO,
       NO_FUNCIONARIO,
       NU_CPF
FROM FUNCIONARIO;
```

Depois, é preciso listar somente os quartos disponíveis, ou seja, quartos que não estão com a hospedagem aiva. Para isso, podemos criar uma view, pois usaremos essa mesma consulta para a criação da dashboard do sistema (espécie de relatório). Primeiro criamos uma view para listar todos os quarto indisponíveis, para aí sim, buscar os quartos disponíveis através da diferença.

```
-- QUARTO INDISPONIVEL E SEU TIPO
CREATE VIEW IF NOT EXISTS VW_QUARTO_INDISPONIVEL AS
SELECT  T2.*,
                T3.NO_TIPO_QUARTO,
                T3.VL_DIARIA
FROM HOSPEDAGEM T1
LEFT JOIN QUARTO T2 ON T1.ID_QUARTO = T2.ID_QUARTO
LEFT JOIN TIPO_QUARTO T3 ON T2.ID_TIPO_QUARTO = T3.ID_TIPO_QUARTO
WHERE T1.IN_ATIVO = 1;

-- QUARTO DISPONIVEL E SEU TIPO
CREATE VIEW IF NOT EXISTS VW_QUARTO_DISPONIVEL AS
SELECT  T1.*,
                T3.NO_TIPO_QUARTO,
                T3.VL_DIARIA
FROM QUARTO T1
LEFT JOIN VW_QUARTO_INDISPONIVEL T2 ON T1.ID_QUARTO = T2.ID_QUARTO
LEFT JOIN TIPO_QUARTO T3 ON T1.ID_TIPO_QUARTO = T3.ID_TIPO_QUARTO
WHERE T2.ID_QUARTO IS NULL;
```

Além disso, listar todos os tipos de pagamentos disponíveis:

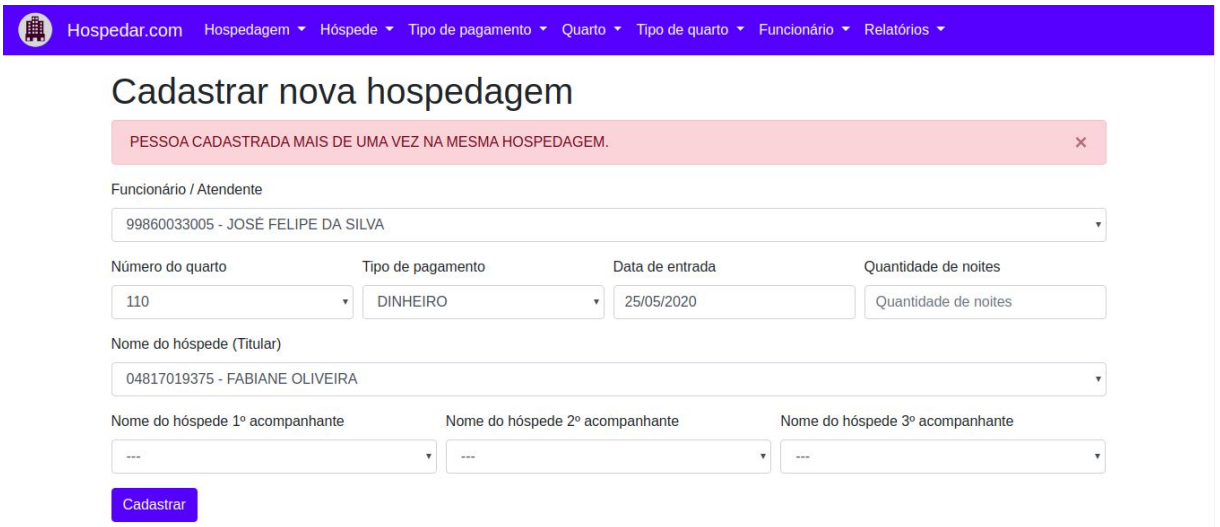
```
SELECT ID_TIPO_PAGAMENTO,
       NO_TIPO_PAGAMENTO
FROM TIPO_PAGAMENTO;
```

A data é preenchida automaticamente com a data de hoje, porém o usuário pode alterar. Depois de digitar a quantidade de noites que os hóspedes irão ficar, o usuário pode selecionar os hóspedes. Para isso, o sistema deve mostrar os hóspedes que não estão em hospedagens ativas. As seguintes views fazem esse papel:

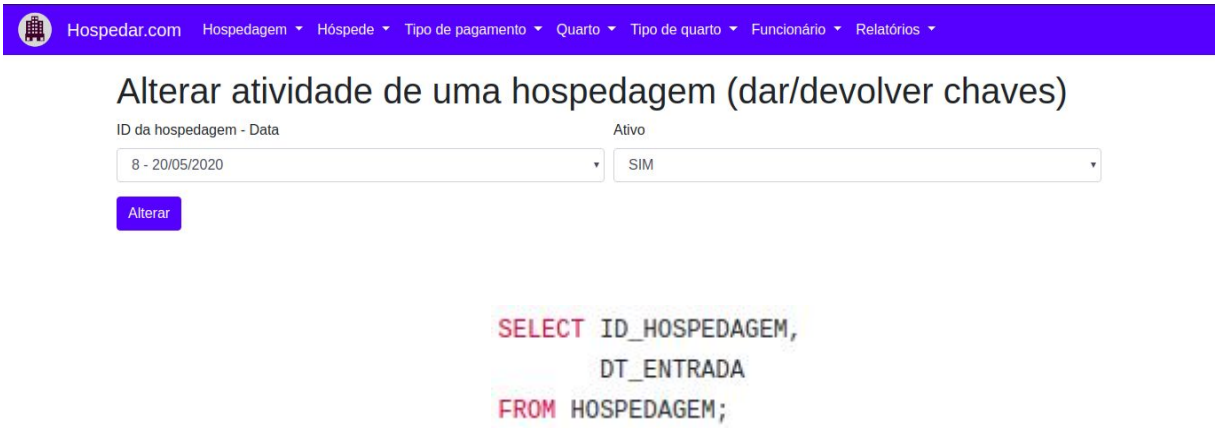
```sql
-- HOSPEDAGEM COMPLETA
CREATE VIEW IF NOT EXISTS VW_HOSPEDAGEM_COMPLETA AS
SELECT  T1.ID_HOSPEDAGEM,
                T2.ID_FUNCIONARIO,
                T2.NU_CPF NU_CPF_FUNCIONARIO,
                T2.NO_FUNCIONARIO,
                T3.ID_QUARTO,
                T3.NU_QUARTO,
                T9.ID_TIPO_QUARTO,
                T9.NO_TIPO_QUARTO,
                T9.VL_DIARIA,
                T4.ID_TIPO_PAGAMENTO,
                T4.NO_TIPO_PAGAMENTO,

                T1.ID_HOSPEDE_TITULAR,
                T5.NU_CPF NU_CPF_TITULAR,
                T5.NO_HOSPEDE NO_HOSPEDE_TITULAR,

                T1.ID_HOSPEDE_ACOMPANHANTE_1,
                T6.NU_CPF NU_CPF_ACOMPANHANTE_1,
                T6.NO_HOSPEDE NO_HOSPEDE_ACOMPANHANTE_1,

                T1.ID_HOSPEDE_ACOMPANHANTE_2,
                T7.NU_CPF NU_CPF_ACOMPANHANTE_2,
                T7.NO_HOSPEDE NO_HOSPEDE_ACOMPANHANTE_2,

                T1.ID_HOSPEDE_ACOMPANHANTE_3,
                T8.NU_CPF NU_CPF_ACOMPANHANTE_3,
                T8.NO_HOSPEDE NO_HOSPEDE_ACOMPANHANTE_3,

                T1.IN_ATIVO,
                T1.DT_ENTRADA,
                T1.NU_NOITE,
                T1.VL_HOSPEDAGEM
FROM HOSPEDAGEM T1
LEFT JOIN FUNCIONARIO T2 ON T1.ID_FUNCIONARIO = T2.ID_FUNCIONARIO
LEFT JOIN QUARTO T3 ON T1.ID_QUARTO = T3.ID_QUARTO
LEFT JOIN TIPO_PAGAMENTO T4 ON T1.ID_TIPO_PAGAMENTO = T4.ID_TIPO_PAGAMENTO
LEFT JOIN HOSPEDE T5 ON T1.ID_HOSPEDE_TITULAR = T5.ID_HOSPEDE
LEFT JOIN HOSPEDE T6 ON T1.ID_HOSPEDE_ACOMPANHANTE_1 = T6.ID_HOSPEDE
LEFT JOIN HOSPEDE T7 ON T1.ID_HOSPEDE_ACOMPANHANTE_2 = T7.ID_HOSPEDE
LEFT JOIN HOSPEDE T8 ON T1.ID_HOSPEDE_ACOMPANHANTE_3 = T8.ID_HOSPEDE
LEFT JOIN TIPO_QUARTO T9 ON T3.ID_TIPO_QUARTO = T9.ID_TIPO_QUARTO;
```

```sql
-- HOSPEDES QUE JÁ ESTÃO EM UMA HOSPEDAGEM (NÃO ESTÃO DISPONÍVEIS PARA NOVA HOSPEDAGEM)
CREATE VIEW IF NOT EXISTS VW_HOSPEDE_INDISPONIVEL_HOSPEDAGEM AS
WITH
	UNION_HOSPEDE_TITULAR AS (
		SELECT  T1.ID_HOSPEDE_TITULAR ID_HOSPEDE,
				T1.NU_CPF_TITULAR NU_CPF,
				T1.NO_HOSPEDE_TITULAR NO_HOSPEDE,
				T1.IN_ATIVO
		FROM VW_HOSPEDAGEM_COMPLETA T1
	),
	UNION_HOSPEDE_ACOMPANHANTE_1 AS (
		SELECT  T1.ID_HOSPEDE_ACOMPANHANTE_1 ID_HOSPEDE,
				T1.NU_CPF_ACOMPANHANTE_1 NU_CPF,
				T1.NO_HOSPEDE_ACOMPANHANTE_1 NO_HOSPEDE,
				T1.IN_ATIVO
		FROM VW_HOSPEDAGEM_COMPLETA T1
	),
	UNION_HOSPEDE_ACOMPANHANTE_2 AS (
		SELECT  T1.ID_HOSPEDE_ACOMPANHANTE_2 ID_HOSPEDE,
				T1.NU_CPF_ACOMPANHANTE_2 NU_CPF,
				T1.NO_HOSPEDE_ACOMPANHANTE_2 NO_HOSPEDE,
				T1.IN_ATIVO
		FROM VW_HOSPEDAGEM_COMPLETA T1
	),
	UNION_HOSPEDE_ACOMPANHANTE_3 AS (
		SELECT  T1.ID_HOSPEDE_ACOMPANHANTE_3 ID_HOSPEDE,
				T1.NU_CPF_ACOMPANHANTE_3 NU_CPF,
				T1.NO_HOSPEDE_ACOMPANHANTE_3 NO_HOSPEDE,
				T1.IN_ATIVO
		FROM VW_HOSPEDAGEM_COMPLETA T1
	),
	HOSPEDE_INDISPONIVEL AS (
		SELECT * FROM UNION_HOSPEDE_TITULAR
		UNION ALL
		SELECT * FROM UNION_HOSPEDE_ACOMPANHANTE_1
		UNION ALL
		SELECT * FROM UNION_HOSPEDE_ACOMPANHANTE_2
		UNION ALL
		SELECT * FROM UNION_HOSPEDE_ACOMPANHANTE_3
	)
SELECT DISTINCT *
FROM HOSPEDE_INDISPONIVEL T1
WHERE T1.IN_ATIVO = 1 AND T1.ID_HOSPEDE IS NOT NULL;
```

Obs: o próprio sistema verifica se o um mesmo hóspede disponível foi utilizado mais de uma vez na mesma hospedagem através das trigger (listada mais adiante).



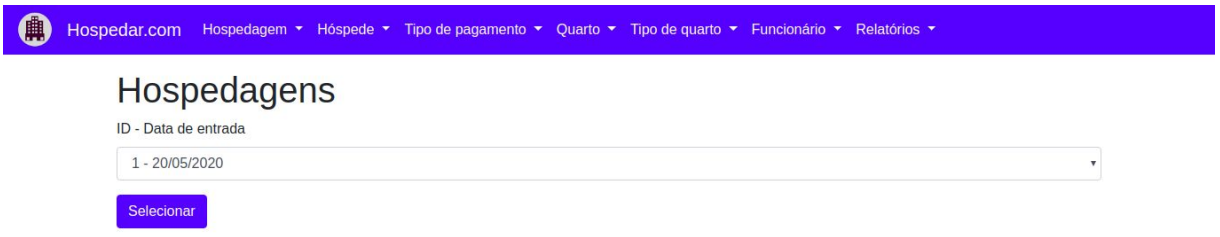
Para a hospedagem, a alteração é somente se ela está ativa ou desativa. Para isso, uma tela que mostra todas as hospedagens e seus identificadores faz a seguinte consulta:

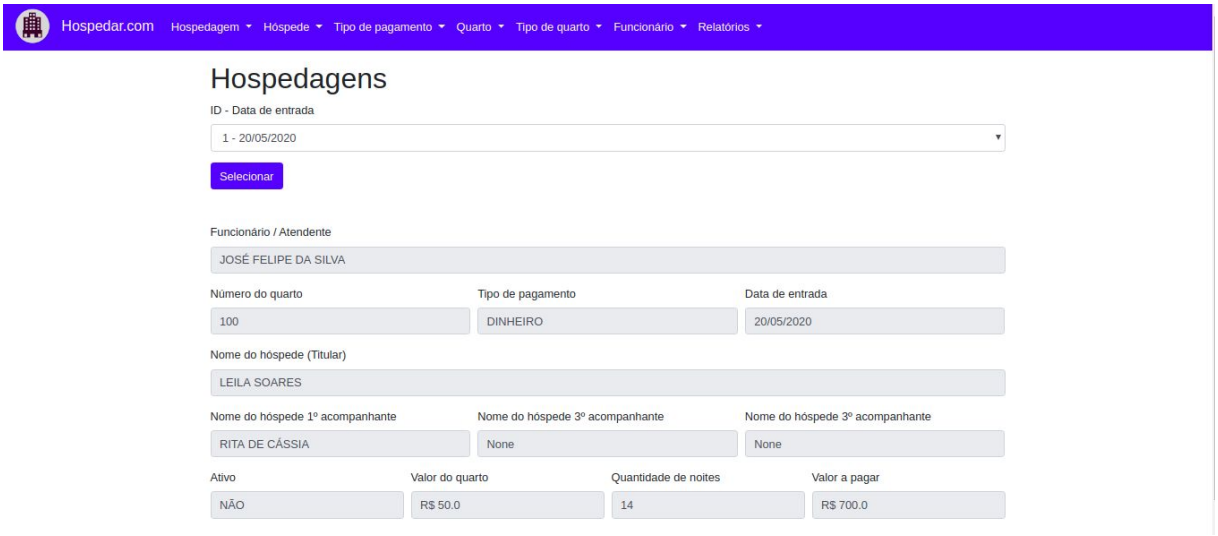



```
SELECT ID_HOSPEDAGEM,
       DT_ENTRADA
FROM HOSPEDAGEM;
```

Depois, o valor é alterado com utilizando a cláusula UPDATE.

Para verificar dados da hospedagem já cadastrada anteriormente, a mesma consulta é utilizada para mostrar as hospedagens disponíveis. Após selecionado, a seguinte tela é gerada a partir das views de hospedagem construídas anteriormente.





Triggers

Validação dos dados

Em algumas colunas, é necessário verificar o tipo ou o formato dos dados que estão entrando nas tabelas. Por exemplo, na tabela *Endereço* o campo CEP não pode aceitar qualquer formato, é necessário que seja somente um campo texto de 8 dígitos de números. Podemos garantir essa regra com as seguintes triggers.

```
CREATE TRIGGER IF NOT EXISTS TR_ENDERECO_BEFORE_INSERT_VALIDA_DADOS
   BEFORE INSERT ON "ENDERECO"
BEGIN
    SELECT
        CASE
            WHEN NEW.NU_CEP NOT LIKE "________" THEN
                RAISE (ABORT, "NUMERO DE CEP INVALIDO")
        END;
END;

CREATE TRIGGER IF NOT EXISTS TR_ENDERECO_BEFORE_UPDATE_VALIDA_DADOS
   BEFORE UPDATE ON "ENDERECO"
BEGIN
    SELECT
        CASE
            WHEN NEW.NU_CEP NOT LIKE "________" THEN
                RAISE (ABORT, "NUMERO DE CEP INVALIDO")
        END;
END;
```

Duas triggers foram criadas para garantir a mesma lógica durante a inserção e durante a alteração.

O mesma acontece nas tabelas *Funcionários* e *Hóspedes*. Devemos garantir que os dados de CPF, E-Mail e telefone sejam preenchidos corretamente.

```
CREATE TRIGGER IF NOT EXISTS TR_FUNCIONARIO_BEFORE_INSERT_VALIDA_DADOS
   BEFORE INSERT ON "FUNCIONARIO"
BEGIN
    SELECT
        CASE
            WHEN NEW.NU_CPF NOT LIKE "___________" THEN
                RAISE (ABORT, "NUMERO DE CPF INVALIDO")
            WHEN NEW.NU_TELEFONE NOT LIKE "___________" THEN
                RAISE (ABORT, "NUMERO DE TELEFONE INVALIDO")
            WHEN NEW.NO_EMAIL NOT LIKE '%_@__%.__%' THEN
                RAISE (ABORT,'EMAIL INVALIDO')
        END;
END;

CREATE TRIGGER IF NOT EXISTS TR_FUNCIONARIO_BEFORE_UPDATE_VALIDA_DADOS
   BEFORE UPDATE ON "FUNCIONARIO"
BEGIN
    SELECT
        CASE
            WHEN NEW.NU_CPF NOT LIKE "___________" THEN
                RAISE (ABORT, "NUMERO DE CPF INVALIDO")
            WHEN NEW.NU_TELEFONE NOT LIKE "___________" THEN
                RAISE (ABORT, "NUMERO DE TELEFONE INVALIDO")
            WHEN NEW.NO_EMAIL NOT LIKE '%_@__%.__%' THEN
                RAISE (ABORT,'EMAIL INVALIDO')
        END;
END;
```

```
CREATE TRIGGER IF NOT EXISTS TR_HOSPEDE_BEFORE_INSERT_VALIDA_DADOS
   BEFORE INSERT ON "HOSPEDE"
BEGIN
    SELECT
        CASE
            WHEN NEW.NU_CPF NOT LIKE "___________" THEN
                RAISE (ABORT, "NUMERO DE CPF INVALIDO")
            WHEN NEW.NU_TELEFONE NOT LIKE "___________" THEN
                RAISE (ABORT, "NUMERO DE TELEFONE INVALIDO")
            WHEN NEW.NO_EMAIL NOT LIKE '%_@__%.__%' THEN
                RAISE (ABORT,'EMAIL INVALIDO')
        END;
END;

CREATE TRIGGER IF NOT EXISTS TR_HOSPEDE_BEFORE_UPDATE_VALIDA_DADOS
   BEFORE UPDATE ON "HOSPEDE"
BEGIN
    SELECT
        CASE
            WHEN NEW.NU_CPF NOT LIKE "___________" THEN
                RAISE (ABORT, "NUMERO DE CPF INVALIDO")
            WHEN NEW.NU_TELEFONE NOT LIKE "___________" THEN
                RAISE (ABORT, "NUMERO DE TELEFONE INVALIDO")
            WHEN NEW.NO_EMAIL NOT LIKE '%_@__%.__%' THEN
                RAISE (ABORT,'EMAIL INVALIDO')
        END;
END;
```

Para a tabela hospedagem não é diferente, é preciso fazer as validações dos dados. Porém, a única validação necessária é verificar se na mesma hospedagem o hóspede não está sendo registrado duas vezes.

```
CREATE TRIGGER IF NOT EXISTS TR_HOSPEDAGEM_BEFORE_INSERT_VALIDA_DADOS
   BEFORE INSERT ON "HOSPEDAGEM"
BEGIN
    SELECT
        CASE
            WHEN NEW.DT_ENTRADA NOT LIKE "__/__/____" THEN
                RAISE (ABORT, "DATA DE ENTRADA INVALIDA")
            WHEN NEW.ID_HOSPEDE_TITULAR IN (NEW.ID_HOSPEDE_ACOMPANHANTE_1, NEW.ID_HOSPEDE_ACOMPANHANTE_2, NEW.ID_HOSPEDE_ACOMPA
                RAISE (ABORT, "PESSOA CADASTRADA MAIS DE UMA VEZ NA MESMA HOSPEDAGEM")
            WHEN NEW.ID_HOSPEDE_ACOMPANHANTE_1 IN (NEW.ID_HOSPEDE_TITULAR, NEW.ID_HOSPEDE_ACOMPANHANTE_2, NEW.ID_HOSPEDE_ACOMPA
                RAISE (ABORT, "PESSOA CADASTRADA MAIS DE UMA VEZ NA MESMA HOSPEDAGEM")
            WHEN NEW.ID_HOSPEDE_ACOMPANHANTE_2 IN (NEW.ID_HOSPEDE_ACOMPANHANTE_1, NEW.ID_HOSPEDE_TITULAR, NEW.ID_HOSPEDE_ACOMPA
                RAISE (ABORT, "PESSOA CADASTRADA MAIS DE UMA VEZ NA MESMA HOSPEDAGEM")
            WHEN NEW.ID_HOSPEDE_ACOMPANHANTE_3 IN (NEW.ID_HOSPEDE_ACOMPANHANTE_1, NEW.ID_HOSPEDE_ACOMPANHANTE_2, NEW.ID_HOSPEDE
                RAISE (ABORT, "PESSOA CADASTRADA MAIS DE UMA VEZ NA MESMA HOSPEDAGEM")
                END;
END;
```

Calculando o valor da hospedagem

Outra trigger necessária na tabela hospedagem é para calcular o valor toda da hospedagem. A seguinte trigger verifica a quantidade de noites que os hóspedes vão ficar e multiplica pelo valor da diária do quarto selecionado. Após calculado, o valor é inserido na tabela.

```
CREATE TRIGGER IF NOT EXISTS TR_HOSPEDAGEM_AFTER_INSERT_PEENCHE_VALOR_HOSPEDAGEM
   AFTER INSERT ON "HOSPEDAGEM"
BEGIN
    UPDATE "HOSPEDAGEM"
    SET "VL_HOSPEDAGEM" = (
        SELECT SUM(NEW.NU_NOITE * T2.VL_DIARIA)
        FROM QUARTO T1
        LEFT JOIN TIPO_QUARTO T2 ON T1.ID_TIPO_QUARTO = T2.ID_TIPO_QUARTO
        WHERE T1.ID_QUARTO = NEW.ID_QUARTO
    ) WHERE ID_HOSPEDAGEM = NEW.ID_HOSPEDAGEM;
END;
```

Views e os relatórios

Na tela inicial do sistema possuem algumas tabelas com dados. Essa é a dashboard do sistema. **Cada cada valor marcado em amarelo nas imagens abaixo é uma consulta ou view**.

Overview

Primeiramente, temos a seção Overview. Nela é possível ter uma noção geral do hotel.

Para os dados da hospedagem, são listados: Total de hospedagem cadastradas, hospedagem ativas, total a receber em real (hospedagens ativas) e total já recebido em real (hospedagens desativadas). As seguintes fazem esse papel:

```
-- HOSPEDAGEM TOTAL
CREATE VIEW IF NOT EXISTS VW_HOSPEDAGEM_TOTAL AS
SELECT COUNT(ID_HOSPEDAGEM) TOTAL
FROM HOSPEDAGEM;
```

```
-- HOSPEDAGEM TOTAL DE ATIVOS
CREATE VIEW IF NOT EXISTS VW_HOSPEDAGEM_TOTAL_ATIVO AS
SELECT COUNT(VL_HOSPEDAGEM) TOTAL_ATIVO
FROM HOSPEDAGEM
WHERE IN_ATIVO = 1;
```

```
-- HOSPEDAGEM TOTAL A RECEBER
CREATE VIEW IF NOT EXISTS VW_HOSPEDAGEM_TOTAL_A_RECEBER AS
SELECT SUM(VL_HOSPEDAGEM) TOTAL_A_RECEBER
FROM HOSPEDAGEM
WHERE IN_ATIVO = 1;
```

```
-- HOSPEDAGEM TOTAL RECEBIDO
CREATE VIEW IF NOT EXISTS VW_HOSPEDAGEM_TOTAL_RECEBIDO AS
SELECT SUM(VL_HOSPEDAGEM) TOTAL_RECEBIDO
FROM HOSPEDAGEM
WHERE IN_ATIVO = 0;
```

Ainda na seção Overview, temos um bloco para visualização dos quartos disponíveis. Novamente, views foram criadas para buscar esses dados do banco de dados.

```
-- QUARTO E SEU TIPO
CREATE VIEW IF NOT EXISTS VW_QUARTO_TIPO_QUARTO AS
SELECT  T1.ID_QUARTO,
                T1.NU_QUARTO,
                T1.ID_TIPO_QUARTO ID_TIPO_QUARTO,
        T2.NO_TIPO_QUARTO,
        T2.VL_DIARIA
FROM QUARTO T1
LEFT JOIN TIPO_QUARTO T2 ON T1.ID_TIPO_QUARTO = T2.ID_TIPO_QUARTO;
```

```sql
-- QUARTO INDISPONIVEL E SEU TIPO
CREATE VIEW IF NOT EXISTS VW_QUARTO_INDISPONIVEL AS
SELECT  T2.*,
                T3.NO_TIPO_QUARTO,
                T3.VL_DIARIA
FROM HOSPEDAGEM T1
LEFT JOIN QUARTO T2 ON T1.ID_QUARTO = T2.ID_QUARTO
LEFT JOIN TIPO_QUARTO T3 ON T2.ID_TIPO_QUARTO = T3.ID_TIPO_QUARTO
WHERE T1.IN_ATIVO = 1;


-- QUARTO DISPONIVEL E SEU TIPO
CREATE VIEW IF NOT EXISTS VW_QUARTO_DISPONIVEL AS
SELECT  T1.*,
                T3.NO_TIPO_QUARTO,
                T3.VL_DIARIA
FROM QUARTO T1
LEFT JOIN VW_QUARTO_INDISPONIVEL T2 ON T1.ID_QUARTO = T2.ID_QUARTO
LEFT JOIN TIPO_QUARTO T3 ON T1.ID_TIPO_QUARTO = T3.ID_TIPO_QUARTO
WHERE T2.ID_QUARTO IS NULL;
```

Para a dashboard dos hóspedes e funcionários não é diferente.

```sql
-- TODOS FUNCIONARIOS
CREATE VIEW IF NOT EXISTS VW_FUNCIONARIO_TOTAL AS
SELECT COUNT(ID_FUNCIONARIO)
FROM VW_FUNCIONARIO_ENDERECO;


-- TODOS HOSPEDES
CREATE VIEW IF NOT EXISTS VW_HOSPEDE_TOTAL AS
SELECT COUNT(ID_HOSPEDE)
FROM VW_HOSPEDE_ENDERECO;
```

Obs: O banco de dados utilizado (SQLite3), em sua sintaxe, não é necessário utilizar a cláusula GROUP BY juntamente com a cláusula COUNT, quando selecionado apenas uma tabela.

Detalhado

Na seção Detalhado da dashboard são apresentados dados das hospedagens e dos quartos, mostrando as chaves primárias (identificadores) das hospedagens ativas e os números dos quartos disponíveis e indisponíveis.

As views utilizadas nessa seção foram as mesma da Overview, porém, sem a contagem.

Rank

Além disso, a dashboard traz uma seção para ranquear os funcionários que mais atenderam e os quartos mais solicitados.

Esses resultados podem ser obtidos através das seguintes views:

```
-- RANK DE FUNCIONÁRIOS
CREATE VIEW IF NOT EXISTS VW_RANK_FUNCIONARIO AS
SELECT  T1.ID_FUNCIONARIO,
                T1.NU_CPF,
                T1.NO_FUNCIONARIO,
                (COUNT(T1.ID_FUNCIONARIO) - 1) QT_HOSPEDAGEM
FROM FUNCIONARIO T1
LEFT JOIN HOSPEDAGEM T2 ON T1.ID_FUNCIONARIO = T2.ID_FUNCIONARIO
GROUP BY T1.NO_FUNCIONARIO
ORDER BY QT_HOSPEDAGEM DESC;
```

```sql
-- RANK DE QUARTOS MAIS REQUISITADOS
CREATE VIEW IF NOT EXISTS VW_RANK_QUARTO AS
SELECT  T1.ID_QUARTO,
                T1.NU_QUARTO,
                COUNT(T2.ID_QUARTO) QT_HOSPEDAGEM
FROM QUARTO T1
LEFT JOIN HOSPEDAGEM T2 ON T1.ID_QUARTO = T2.ID_QUARTO
GROUP BY T1.ID_QUARTO
ORDER BY QT_HOSPEDAGEM DESC;
```

Outros

O sistema também disponibiliza relatórios detalhados de cada entidade do sistema, conforme pode ser visto na seção Tabelas, índices e views do documento.

Estimativa da quantidade de linhas durante 1 ano

Como o sistema foi projetado para controle das hospedagem de um hotel, as tabelas que mais poderão receber dados seriam a de Hóspedes e de Hospedagem. Considerando que o hotel receba, em um mês, 30 novos hóspedes e registre 55 hospedagens, as tabelas de Hóspedes e Hospedagem estariam, ao final de um ano, com 360 e 660 registro, respectivamente.

Então, um banco de dados como por exemplo o MySQL com um plano de hospedagem de dados básico, conseguiria atender o sistema por, pelo menos, 10 anos.