Thainara Rogério

e-Coworking

Projeto open-source disponível em:
https://github.com/thainararogerio/CoworkingSpaceProject.

BLUMENAU

2017

SUMÁRIO

Sumário

1 INTRODUÇÃO

Espaços *coworkings* vem sendo criados em médias e grandes cidades nos últimos tempos. Se tratam de instalações que foram criadas especificamente para trabalhar e que possuem salas equipadas com projetores, quadros brancos para apresentações e mesas e cadeiras para reuniões. Além disso, os espaços podem possuir uma área comum, na qual são ofertados quitutes, cafés e outros produtos e serviços.

Esses espaços são utilizados principalmente por profissionais das áreas de tecnologia, marketing digital, design, e outros, sendo funcionários de um empresa, empreendedores ou freelancers. Espaços *coworkings* possibilitam o compartilhamento de recursos e de uma estrutura robusta e de qualidade entre profissionais que não os possuem ou pequenas empresas - comumente as chamadas *start-ups* - que preferem não custear a própria estrutura. Além disso, esse tipo de modelo de trabalho gera o crescimento da rede de contatos do profissional, inclusive com profissionais de outras áreas.

Um estabelecimento como este pode possuir diversas salas, cada qual com seus equipamentos e capacidade, além de outras propriedades físicas. A cada uso, o ideal é que a sala e seus equipamentos sejam verificados e que se mantenha um registro do cliente que a utilizou, com sua identificação e informações de contato.

O presente trabalho trata justamente do protótipo de um sistema que informatiza a coleta, manutenção e consulta dessas informações. Trata-se de um software para espaços *coworking* que permite o registro e acompanhamento da utilização de salas, equipamentos, clientes e reservas. Além do cadastro dessas informações, o sistema possibilitará consultas como: reservas de determinada sala em determinada data e hora, salas livres por data e hora, salas que possuem determinado equipamento, entre outras.

2 DESENVOLVIMENTO

Neste capítulo são apresentados os requisitos do sistema, o Modelo Entidade Relacionamento, os trechos de códigos no contexto de banco de dados, seguido de imagens ilustrando o resultado das operações.

2.1 Requisitos do Sistema

O principal objetivo do sistema e-Coworking é informatizar os processos que envolvem a administração básica de um espaço coworking. A aplicação abrange principalmente o gerenciamento da infra estrutura e dos clientes e suas reservas. O sistema será no formato *Desktop* e deverá ser instalado no computador para sua utilização.

Entre os requisitos principais do sistema está a permissão do cadastro de salas, para as quais o usuário pode designar um nome, um tipo e os equipamentos que possuem (por exemplo, um notebook e um projetor). Deve ser possível cadastrar equipamentos, adicionando a eles um nome, e também cadastrar tipos de sala, com seu nome e tamanho. Além disso, o sistema deve permitir o cadastro de clientes com informações como: nome, endereço e contatos.

O cadastro principal do sistema será o de reservas, onde o usuário poderá registrar as reservas de salas feitas pelos clientes. Uma reserva deve conter: o cliente, a sala reservada, datas e horários de entrada e saída, o valor da reserva e se ela já foi paga. Depois do registro da reserva, o usuário pode ainda dar entrada a uma multa vinculada à reserva. Nesse cadastro, devem ser armazenados a reserva, o valor da multa e a data de pagamento.

Os requisitos funcionais são apresentar:

- as reservas de determinada sala em certa data/hora
- salas livres em determinada data/hora
- salas com determinado equipamento
- clientes que utilizaram determinada sala em determinada data/hora
- reservas de determinado cliente
- reservas não pagas agrupadas por cliente e ordenadas decrescentemente pelo valor
- multas não pagas agrupadas por cliente e ordenadas pelo valor decrescentemente

2.2 Modelo Entidade Relacionamento

Com base nas informações coletadas e no levantamento de requisitos funcionais foi desenvolvido o Modelo Entidade Relacionamento (MER) físico da base de dados. O MER tem como principal finalidade representar os relacionamentos de conectividade existentes para a formalização da estrutura necessária para a criação específica desta base de dados. As Figura 1 e 2 representam o diagrama resultante desenvolvido para o SGBD Oracle.

Figura 01 – Modelo Entidade Relacionamento parte 1

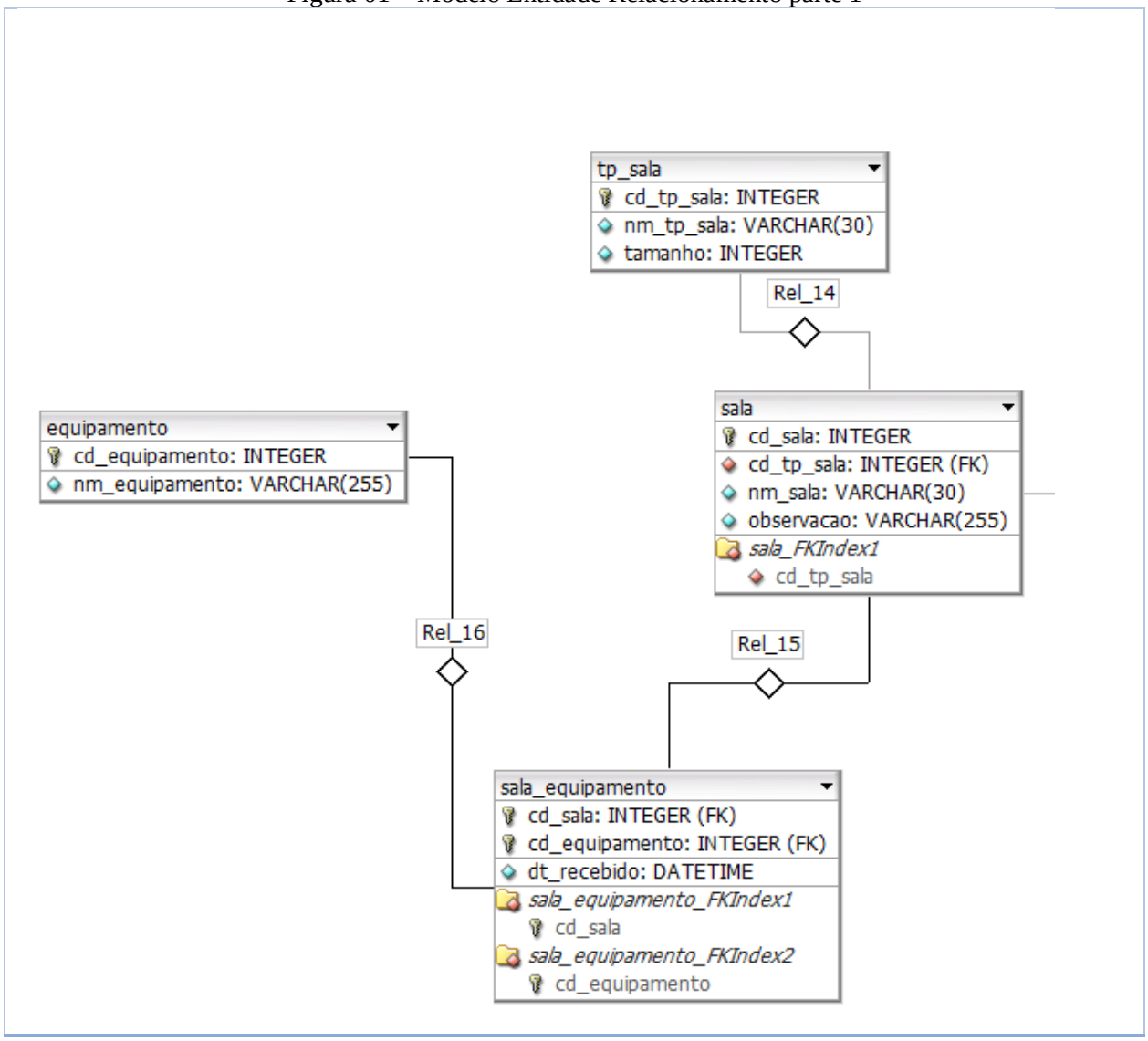
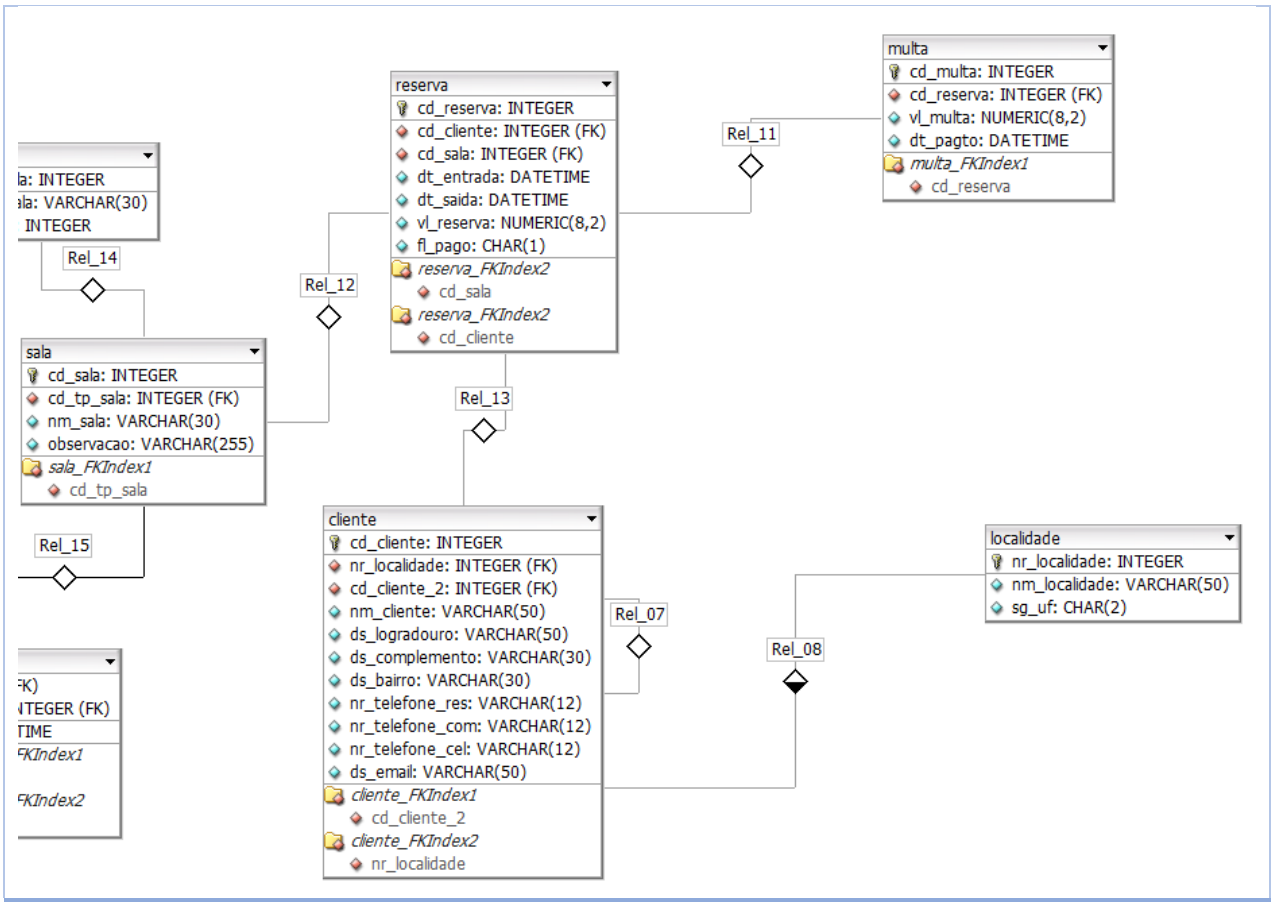


Figura 02 – Modelo Entidade Relacionamento parte 2



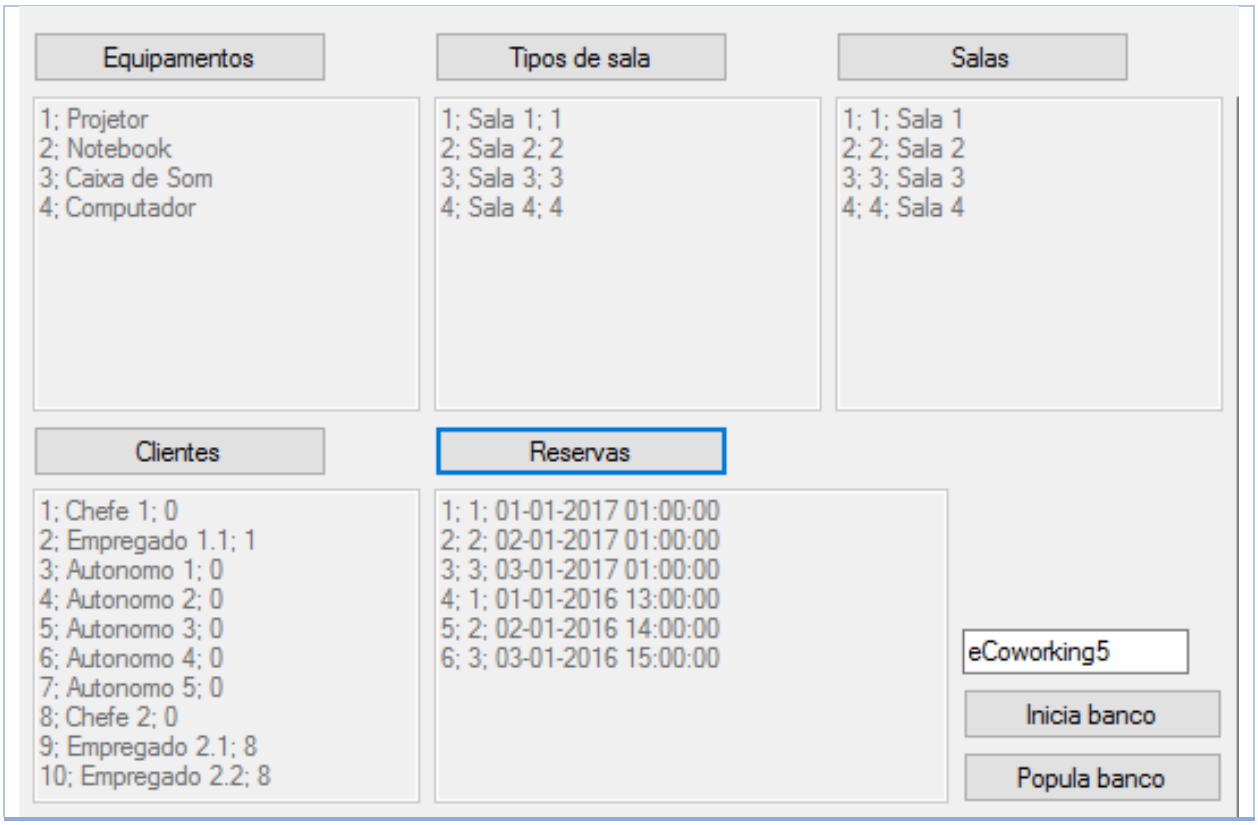
2.3 Implementação do Sistema

O sistema utiliza o banco SQL Server, conectando-se a ele com a classe SqlConnection do C# .NET. Toda a aplicação foi desenvolvida na linguagem C# framework .NET e conta com uma tela protótipo.

Para utilizar o sistema, o usuário deve configurar a base com os comandos de criação de tabelas disponibilizados no fim deste documento. Também está disponível o conjunto de comandos para população da tabela, ainda que o sistema possua uma função para isso, como poderemos ver a seguir.

Ao abrir o sistema, como mostra a Figura 3, é possível ver áreas de texto e em cada uma delas será apresentado um conjunto de registros. Em cima de cada área de texto há botões, identificados pelo tipo de registro, que buscam os registros do banco, fazendo *select* no banco.

Figura 03 – Áreas de texto para registros



Na parte inferior também temos o campo para informar o nome do banco de dados utilizado e os botões "Inicia banco", que faz com que a aplicação considere o banco informado a partir de então. Temos também o botão "Popula banco" que pode ser acionado caso se trate de um novo banco. A rotina de população do banco insere alguns registros para fins de teste das rotinas de *select*. A classe responsável por essa inclusão é a PreencheBancoUtils e o Quadro 1 ilustra um trecho de código desta classe. O trecho se trata da inclusão de registros de equipamentos.

Quadro 01 – Trecho de código da inclusão de equipamentos

```
internal void AddEquipamentos(SqlConnection conexaoSql)
{
            equipamento equipamento1 = new equipamento() { cd_equipamento = 1,
nm_equipamento = "Projetor" };
            equipamento equipamento2 = new equipamento() { cd_equipamento = 2,
nm_equipamento = "Notebook" };
            equipamento equipamento3 = new equipamento() { cd_equipamento = 3,
nm_equipamento = "Caixa de Som" };
            equipamento equipamento4 = new equipamento() { cd_equipamento = 4,
nm_equipamento = "Computador" };
            EquipamentoDAO.Add(equipamento1, conexaoSql);
            EquipamentoDAO.Add(equipamento2, conexaoSql);
            EquipamentoDAO.Add(equipamento3, conexaoSql);
```

```
            EquipamentoDAO.Add(equipamento4, conexaoSql);
}
```

Os *inserts* em si são realizados pela classe correspondente ao registro sendo incluído, portanto no exemplo do Quadro 1, os *inserts* são feitos pela EquipamentoDAO. O método Add, chamado pela PreencheBancoUtils, pode ser visualizado no Quadro 2 a seguir. O método trata da montagem do comando INSERT e da execução dele através da classe SqlCommand, do .NET.

Quadro 02 – Trecho de código do insert de equipamento

```
public static string NOME_TABELA = "equipamento";

public static void Add(equipamento novoEquip, SqlConnection conexaoSql)
{
            string sql = "INSERT INTO equipamento (" + equipamento.CD_EQUIPAMENTO +
", " + equipamento.NM_EQUIPAMENTO + ") "
                  + " values (@" + equipamento.CD_EQUIPAMENTO + ", @" +
equipamento.NM_EQUIPAMENTO + ") ";

            SqlCommand cmd = conexaoSql.CreateCommand();
            cmd.CommandText = sql;


cmd.Parameters.Add(DBUtils.criaParametro<int>(equipamento.CD_EQUIPAMENTO,
novoEquip.cd_equipamento, SqlDbType.Int));

cmd.Parameters.Add(DBUtils.criaParametro<string>(equipamento.NM_EQUIPAMENTO,
novoEquip.nm_equipamento, SqlDbType.VarChar));

            int rowCount = cmd.ExecuteNonQuery();
            AcessoBanco.comandosSqlExecutados += DBUtils.MontaComandoSql(cmd) +
"\r\n";

            Debug.Write("Linhas afetadas: " + rowCount);
}
```

A implementação foi feita de modo que também é possível adicionar um registro informado pelo usário, apesar de o sistema ainda não apresentar telas para isso. O controle de inserção de registros reais ou não é feito pela classe AcessoBanco, que verifica se foi informado um registro. Se sim, o registro informado é inserido, caso não foi informado nenhum registro, é chamada a rotina de população automática de dados fictícios.

Os botões para cada registro ilustrados pela Figura 3 acionam comandos de *select* simples na base. Os *selects* também são feitos pelas classe DAO de cada registro. O Quadro 3 mostra a rotina de *select* da tabela de salas, feito na classe SalaDAO.

Quadro 03 – Trecho de código do select de salas

```
internal static List<sala> Busca(SqlConnection conexaoSql)
{
   string sql = "SELECT * FROM " + NOME_TABELA;
   return Le(sql, conexaoSql);
}
```

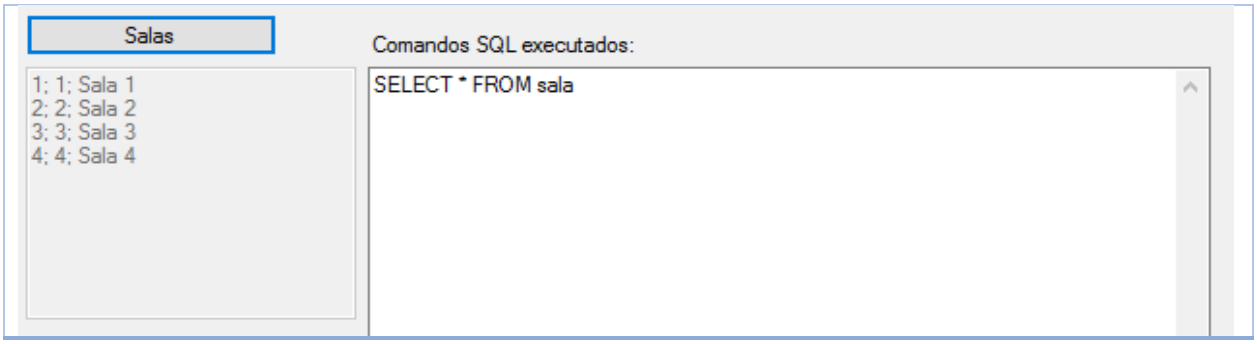
As classes DAO, responsáveis pelos comandos na base de dados em cada tabela, são:

- ClienteDAO
- EquipamentoDAO
- LocalidadeDAO
- MultaDAO
- PreencheBancoUtils
- ReservaDAO
- SalaDAO
- SalaEquipamentoDAO
- TipoSalaDAO

Os comandos *select* também são executados pela classe SqlCommand do .NET e o sistema Espaço Coworking lê cada um dos registros retornados.

O sistema também apresenta uma área de texto que mostra os comandos executados até o momento. No exemplo do *select* de salas, visto no Quadro 3, por exemplo, o sistema listaria as salas obtidas e o comando usado para buscá-las, como mostra a Figura 4 a seguir.

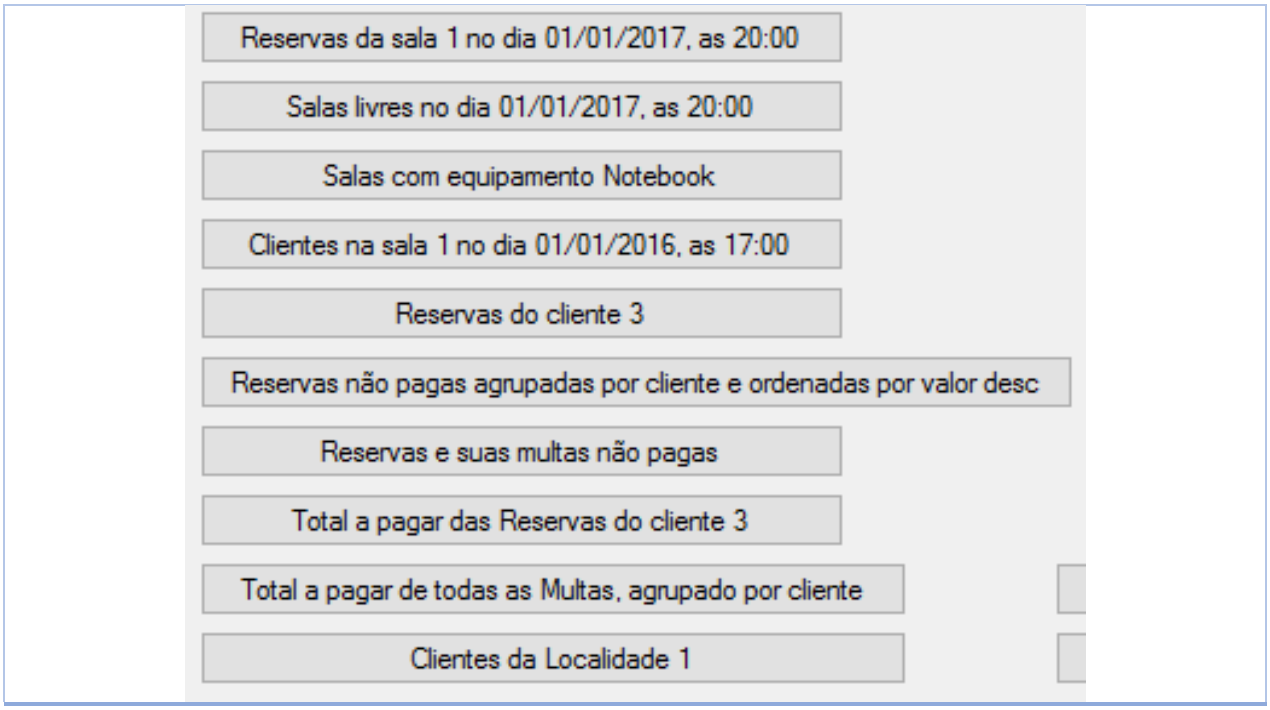
Figura 04 – Tela apresentando registros sala e o comando de *select*



2.3.1 Relatórios

O sistema também possui rotinas de relatórios e, para fins de execução dessas rotinas, foram disponibilizados botões para cada uma delas. A Figura 5 ilustra os botões, identificados por suas funções.

Figura 05 – Relatórios disponíveis



Relatório 1: Reservas de determinada sala em determinada data/hora. Seguem código fonte, comando executado e resultado da consulta.

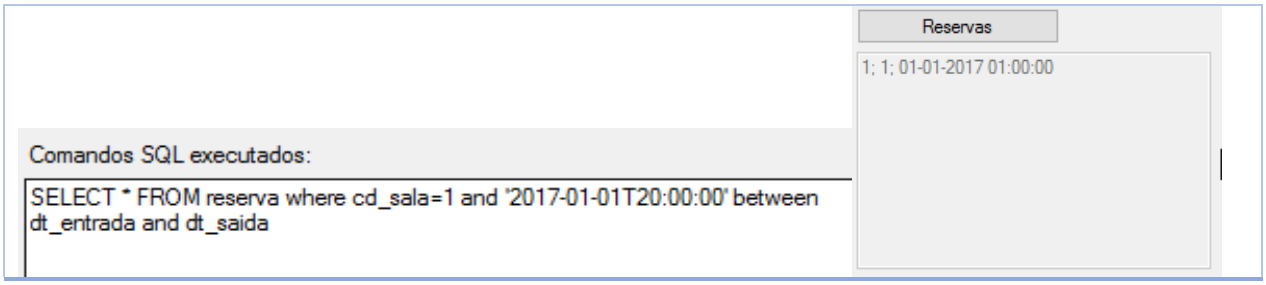
Quadro 04 – Trecho do código da busca do relatório 1

```
internal List<reserva> BuscaReservasSalaDataHora()
{
   return ReservaDAO.BuscaPor(new sala() { cd_sala = 1 }, new DateTime(2017, 01,
01, 20, 0, 0), _conexaoSql);
}

internal static List<reserva> BuscaPor(sala sala, DateTime dataHora, SqlConnection
conexaoSql)
{
            string sql = "SELECT * FROM " + NOME_TABELA;
            sql += " where cd_sala=" + sala.cd_sala;
            sql += " and '" + dataHora.ToString("yyyy-MM-ddTHH:mm:ss") + "' between
dt_entrada and dt_saida";

            return Le(sql, conexaoSql);
}
```

Figura 06 – Comando executado e resultado da consulta para o relatório 1



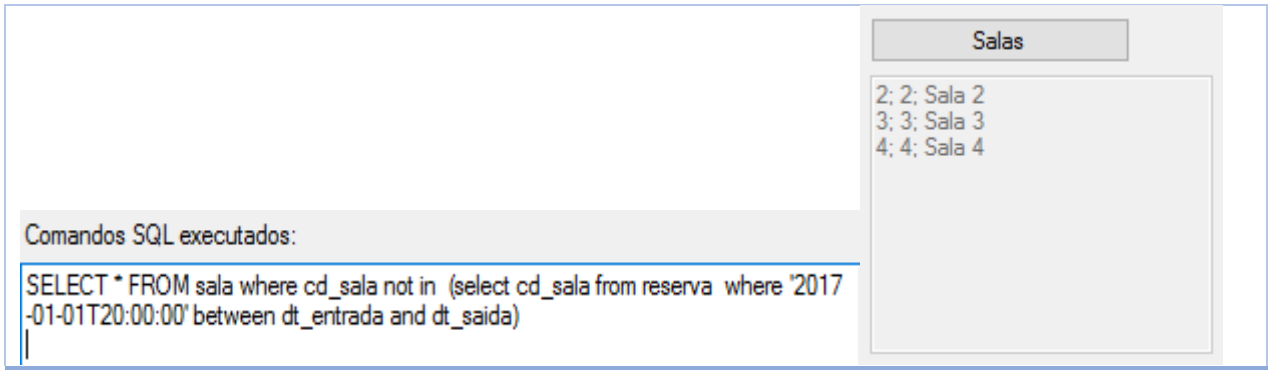
Relatório 2: Salas livres em determinada data/hora.

Quadro 05 – Trecho do código da busca do relatório 2

```
internal List<sala> BuscaSalasLivresDataHora()
{
            return SalaDAO.BuscaSalasLivresEm(new DateTime(2017, 01, 01, 20, 0,
0), _conexaoSql);
}
internal static List<sala> BuscaSalasLivresEm(DateTime dataHora, SqlConnection
conexaoSql)
{
            string sql = "SELECT * FROM " + NOME_TABELA;
            sql += " where cd_sala not in ";
            sql += " (select cd_sala from reserva ";
            sql += " where '" + dataHora.ToString("yyyy-MM-ddTHH:mm:ss") + "'
between dt_entrada and dt_saida)";

            return Le(sql, conexaoSql);
}
```

Figura 07 – Comando executado e resultado da consulta para o relatório 2



Relatório 3: Salas com determinado equipamento.

Quadro 06 – Trecho do código da busca do relatório 3

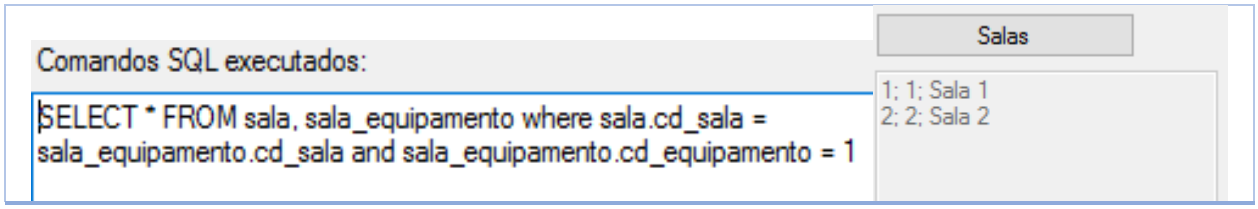
```
internal List<sala> BuscaSalasComEquipamento()
{
```

```
  return SalaDAO.BuscaSalasComEquipamento(1, _conexaoSql);
}

internal static List<sala> BuscaSalasComEquipamento(int equipamento, SqlConnection
conexaoSql)
{
            string sql = "SELECT * FROM " + NOME_TABELA + ", " +
SalaEquipamentoDAO.NOME_TABELA;
            sql += " where sala.cd_sala = sala_equipamento.cd_sala";
            sql += " and sala_equipamento.cd_equipamento = " + equipamento;

            return Le(sql, conexaoSql);
}
```

Figura 08 – Comando executado e resultado da consulta para o relatório 3



Relatório 4: Clientes que utilizaram determinada sala em determinados data/hora.

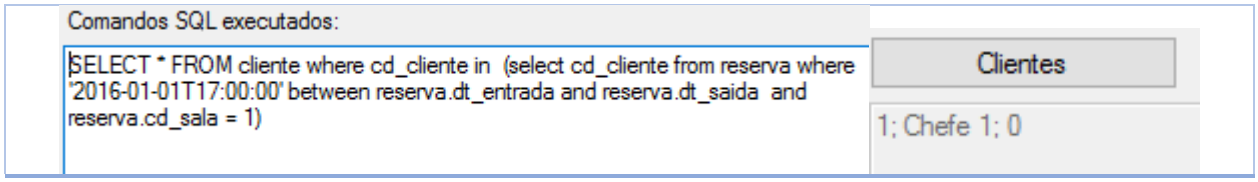
Quadro 07 – Trecho do código da busca do relatório 4

```
internal List<cliente> BuscaNaSalaDataHora()
{
  return ClienteDAO.BuscaClienteSalaDataHora(1, new DateTime(2016, 01, 01, 17, 0,
0), _conexaoSql);
}

internal static List<cliente> BuscaClienteSalaDataHora(int sala, DateTime dataHora,
SqlConnection conexaoSql)
{
            string sql = "SELECT * FROM " + NOME_TABELA;
            sql += " where cd_cliente in ";
            sql += " (select cd_cliente from reserva where '";
            sql += dataHora.ToString("yyyy-MM-ddTHH:mm:ss") + "' between
reserva.dt_entrada and reserva.dt_saida ";
            sql += " and reserva.cd_sala = " + sala + ")";

            return Le(conexaoSql, sql);
}
```

Figura 09 – Comando executado e resultado da consulta para o relatório 4



Relatório 5: Reservas de determinado cliente.

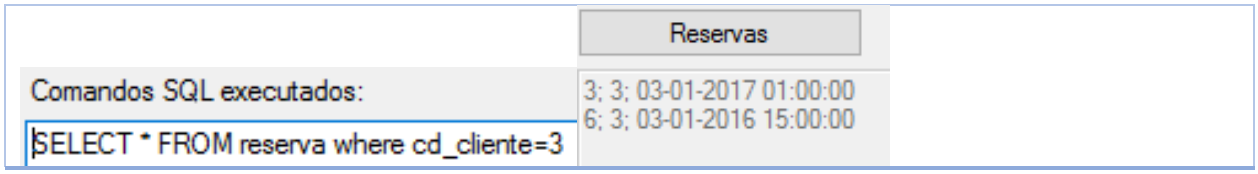
Quadro 08 – Trecho do código da busca do relatório 5

```
internal List<reserva> BuscaReservasCliente()
{
    return ReservaDAO.BuscaPor(new cliente() { cd_cliente = 3 }, _conexaoSql);
}

internal static List<reserva> BuscaPor(cliente cliente, SqlConnection conexaoSql)
{
    string sql = "SELECT * FROM " + NOME_TABELA;
    sql += " where cd_cliente=" + cliente.cd_cliente;

    return Le(sql, conexaoSql);
}
```

Figura 10 – Comando executado e resultado da consulta para o relatório 5



Relatório 6: Reservas não pagas agrupadas por cliente e ordenadas por valor decrescente.

Quadro 09 – Trecho do código da busca do relatório 6

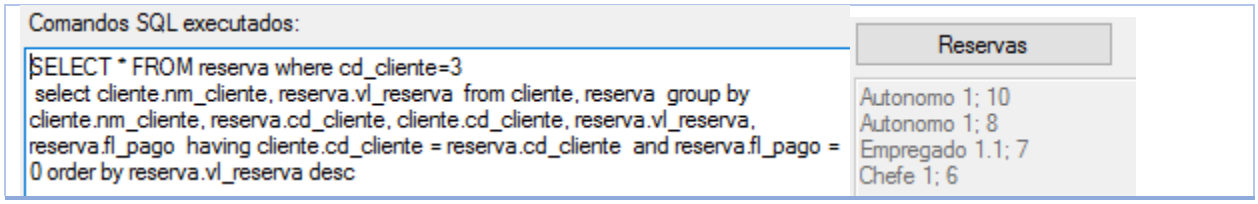
```
internal List<cliente_reserva> BuscaReservasNaoPagas()
{
   return DAOGenerico.BuscaReservasAgrupCliente(new reserva() { fl_pago = false },
SortOrder.Descending, _conexaoSql);
}

internal static List<cliente_reserva> BuscaReservasAgrupCliente(reserva reserva,
SortOrder order, SqlConnection conexaoSql)
{
            string sql = " select cliente.nm_cliente, reserva.vl_reserva ";
            sql += " from cliente, reserva ";
            sql += " group by cliente.nm_cliente, reserva.cd_cliente,
cliente.cd_cliente, reserva.vl_reserva, reserva.fl_pago ";
```

```
            sql += " having cliente.cd_cliente = reserva.cd_cliente ";
            sql += " and reserva.fl_pago = " + (reserva.fl_pago ? "1" : "0");
            sql += " order by reserva.vl_reserva " + (order ==
SortOrder.Ascending ? "asc" : "desc");

            return LeClienteReserva(sql, conexaoSql);
}
```

Figura 11 – Comando executado e resultado da consulta para o relatório 6



Relatório 7: Reservas e suas multas não pagas.
Para este relatório, foi criada uma view chamada ReservasMultasNaoPagas.

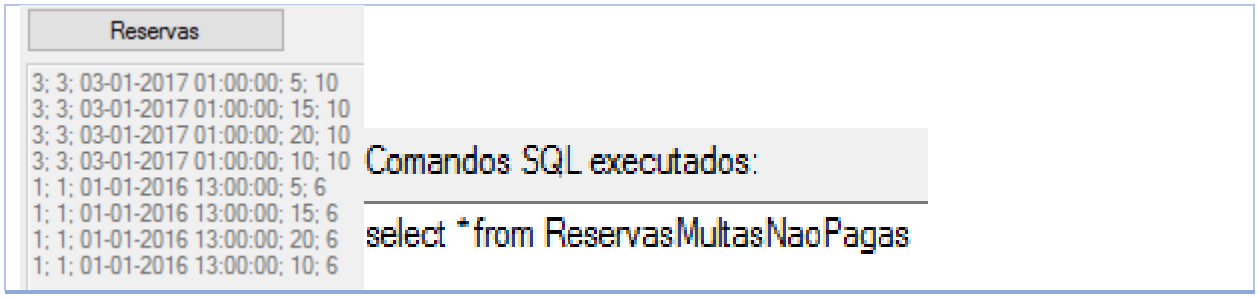
Quadro 10 – Trecho do código da busca do relatório 7

```
internal List<reserva_multa> BuscaReservasComMultasNaoPagas()
        {
            return DAOGenerico.BuscaReservasMultaNaoPaga(_conexaoSql);
        }

internal static List<reserva_multa> BuscaReservasMultaNaoPaga(SqlConnection
conexaoSql)
        {
            string sql = "select * from ReservasMultasNaoPagas";

            return LeReservaMulta(sql, conexaoSql);
        }
```

Figura 12 – Comando executado e resultado da consulta para o relatório 7



Relatório 8: Soma dos valores das reservas não pagas de determinado cliente.

Quadro 11 – Trecho do código da busca do relatório 8

```
internal float BuscaSomaReservasCliente()
{
```

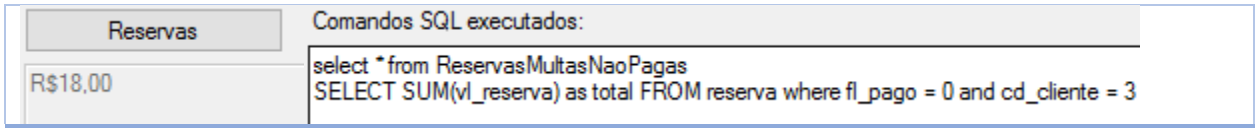
```
            return ReservaDAO.BuscaTotalPendentePorCliente(new cliente()
{ cd_cliente = 3 }, _conexaoSql);
}

internal static float BuscaTotalPendentePorCliente(cliente cliente, SqlConnection
conexaoSql)
{
            float total = 0;

            string sql = "SELECT SUM(vl_reserva) as total FROM " + NOME_TABELA;
            sql += " where fl_pago = 0 and cd_cliente = " + cliente.cd_cliente;

           ... leitura do resultado...
}
```

Figura 13 – Comando executado e resultado da consulta para o relatório 8



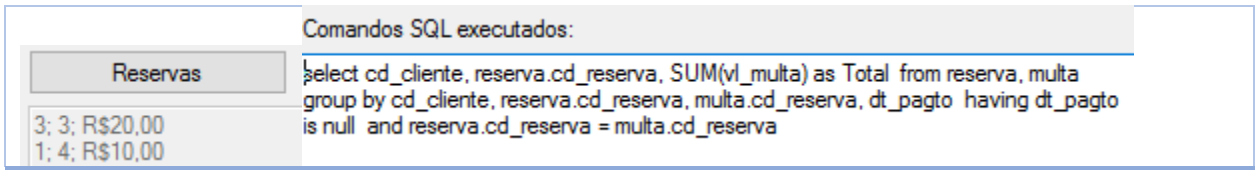
Relatório 9: Soma dos valores de todas as multas não pagas agrupadas por cliente.

Quadro 12 – Trecho do código da busca do relatório 9

```
internal List<cliente_multa> BuscaTotalPagarMultasCliente()
        {
            return DAOGenerico.BuscaClienteMultaTotalPagar(_conexaoSql);
        }
internal static List<cliente_multa> BuscaClienteMultaTotalPagar(SqlConnection
conexaoSql)
        {
            string sql = "select cd_cliente, reserva.cd_reserva, SUM(vl_multa) as
Total ";
            sql += " from reserva, multa ";
            sql += " group by cd_cliente, reserva.cd_reserva, multa.cd_reserva,
dt_pagto ";
            sql += " having dt_pagto is null ";
            sql += " and reserva.cd_reserva = multa.cd_reserva ";

            return LeClienteMulta(sql, conexaoSql);
        }
```

Figura 14 – Comando executado e resultado da consulta para o relatório 9



Relatório 10: Clientes de determinada localidade.

Quadro 13 – Trecho do código da busca do relatório 10

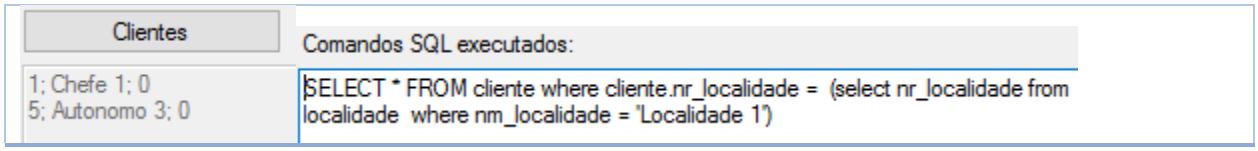
```
List<cliente> clientes = _acessoBanco.BuscaClientes(new localidade()
{ nm_localidade = "Localidade 1" });

internal List<cliente> BuscaClientes(localidade localidade)
{
            return ClienteDAO.Busca(localidade, _conexaoSql);
}

internal static List<cliente> Busca(localidade localidade, SqlConnection
conexaoSql)
{
            string sql = "SELECT * FROM " + NOME_TABELA;
            sql += " where cliente.nr_localidade = ";
            sql += " (select nr_localidade from localidade ";
            sql += " where nm_localidade = '" + localidade.nm_localidade + "') ";

            return Le(conexaoSql, sql);
}
```

Figura 15 – Comando executado e resultado da consulta para o relatório 10



3 CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentado um sistema de informatização de um tipo de negócio conhecido como Espaço Coworking, que é um estabelecimento com uma infraestrutura adequada para profissionais e empresas que não desejam ter o custo de montar seu próprio ambiente. É possível ver que o sistema não está funcional ainda para um estabelecimento como este, porém possui a estrutura de busca e de inserção toda pronta por trás. O sistema é funcional em suas buscas, porém não é possível hoje ainda inserir registros pela interface gráfica, exceto por registros fictícios pré estabelecidos.

Haja vista que a implementação do sistema já foi feita com o intuito de ter como produto um sistema com as funcionalidades de cadastro e de consultas de relatórios, é necessário somente adaptar sua interface gráfica para que permita ao usuário realizar inserções. Além disso,

deve-se aperfeiçoar as visualizações das consultas para que tragam informações completas de registros de chave estrangeira, por exemplo, e não somente a chave primária destes.

No que concerne à tecnologia utilizada, conclui-se que atende bem aos requisitos e ao objetivo principal do sistema. A linguagem C# com o framework .NET possuem suporte para a conexão e execução de comandos em bancos SQL Server. O banco SQL Server, por sua vez, também atende à solução satisfatoriamente.

REFERÊNCIAS BIBLIOGRÁFICAS

VALDAMERI, A. R. **Materiais sobre Banco de Dados.**. Disponível em: <http://ava.furb.br/ava2/FURB/inicial/>. Acesso em: 01 mar. 2017.

Comandos de criação de tabelas

```
create table equipamento
(
    cd_equipamento int not null primary key,
    nm_equipamento varchar(255)
);

create table tp_sala
(
    cd_tp_sala int not null primary key,
    nm_tp_sala varchar(30),
    tamanho int
);

create table sala
(
    cd_sala int not null primary key,
    cd_tp_sala int not null references tp_sala(cd_tp_sala),
    nm_sala varchar(30),
    observacao varchar(255)
);

create table sala_equipamento
(
    cd_sala int not null references sala(cd_sala),
    cd_equipamento int not null references equipamento(cd_equipamento),
    primary key (cd_sala, cd_equipamento),
    dt_recebido datetime
);

create table localidade
```

```
(
    nr_localidade int not null primary key,
    nm_localidade varchar(50),
    sg_uf char(2)
);

create table cliente
(
    cd_cliente int not null primary key,
    nr_localidade int not null references localidade(nr_localidade),
    cd_responsavel int references cliente(cd_cliente),
    nm_cliente varchar(50),
    ds_logradouro varchar(50),
    ds_complemento varchar(30),
    ds_bairro varchar(30),
    nr_telefone_res varchar(12),
    nr_telefone_com varchar(12),
    nr_telefone_cel varchar(12),
    ds_email varchar(50)
);

create table reserva
(
    cd_reserva int not null primary key,
    cd_cliente int not null references cliente(cd_cliente),
    cd_sala int not null references sala(cd_sala),
    dt_entrada datetime,
    dt_saida datetime,
    vl_reserva numeric(8,2),
    fl_pago char(1)
);

create table multa
(
    cd_multa int not null primary key,
    cd_reserva int not null references reserva(cd_reserva),
    vl_multa numeric(8,2),
    dt_pagto datetime
);

create view ReservasMultasNaoPagas as
select reserva.cd_cliente, reserva.cd_sala, reserva.dt_entrada, reserva.vl_reserva,
multa.vl_multa
from reserva, multa
where reserva.cd_reserva  in
(select cd_reserva from multa
where dt_pagto is null)
```

Comandos de população do banco

```
INSERT INTO equipamento (cd_equipamento, nm_equipamento)  values (1, 'Projetor')
INSERT INTO equipamento (cd_equipamento, nm_equipamento)  values (2, 'Notebook')
INSERT INTO equipamento (cd_equipamento, nm_equipamento)  values (3, 'Caixa de
Som')
```

```
INSERT INTO equipamento (cd_equipamento, nm_equipamento)  values (4, 'Computador')
INSERT INTO tp_sala (cd_tp_sala, nm_tp_sala, tamanho)  values (1, 'Sala 1', 1)
INSERT INTO tp_sala (cd_tp_sala, nm_tp_sala, tamanho)  values (2, 'Sala 2', 2)
INSERT INTO tp_sala (cd_tp_sala, nm_tp_sala, tamanho)  values (3, 'Sala 3', 3)
INSERT INTO tp_sala (cd_tp_sala, nm_tp_sala, tamanho)  values (4, 'Sala 4', 4)
INSERT INTO sala (cd_sala, cd_tp_sala, nm_sala, observacao)  values (1, 1, 'Sala
1', 'tem vista para rua')
INSERT INTO sala (cd_sala, cd_tp_sala, nm_sala, observacao)  values (2, 2, 'Sala
2', 'possui lixeira')
INSERT INTO sala (cd_sala, cd_tp_sala, nm_sala, observacao)  values (3, 3, 'Sala
3', 'voltada para lateral do espaço')
INSERT INTO sala (cd_sala, cd_tp_sala, nm_sala, observacao)  values (4, 4, 'Sala
4', 'nos fundos')
INSERT INTO sala_equipamento (cd_sala, cd_equipamento, dt_recebido)  values (1, 1,
'2017-01-03T01:00:00')
INSERT INTO sala_equipamento (cd_sala, cd_equipamento, dt_recebido)  values (1, 2,
'2017-02-03T02:00:00')
INSERT INTO sala_equipamento (cd_sala, cd_equipamento, dt_recebido)  values (1, 3,
'2017-03-03T03:00:00')
INSERT INTO sala_equipamento (cd_sala, cd_equipamento, dt_recebido)  values (1, 4,
'2017-04-03T04:00:00')
INSERT INTO sala_equipamento (cd_sala, cd_equipamento, dt_recebido)  values (2, 1,
'2017-05-03T05:00:00')
INSERT INTO localidade (nr_localidade, nm_localidade, sg_uf)  values (1,
'Localidade 1', 'SC')
INSERT INTO localidade (nr_localidade, nm_localidade, sg_uf)  values (2,
'Localidade 2', 'PR')
INSERT INTO localidade (nr_localidade, nm_localidade, sg_uf)  values (3,
'Localidade 3', 'PN')
INSERT INTO localidade (nr_localidade, nm_localidade, sg_uf)  values (4,
'Localidade 4', 'DF')
INSERT INTO cliente (cd_cliente, nr_localidade, nm_cliente, ds_logradouro,
ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com, nr_telefone_cel,
ds_email)  values (1, 1, 'Chefe 1', 'Log. Chefe 1', null, null, null, null, null,
null)
INSERT INTO cliente (cd_cliente, cd_responsavel, nr_localidade, nm_cliente,
ds_logradouro, ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com,
nr_telefone_cel, ds_email)  values (2, 1, 2, 'Empregado 1.1', 'Log. Emp. 1.1',
null, null, null, null, null, null)
INSERT INTO cliente (cd_cliente, nr_localidade, nm_cliente, ds_logradouro,
ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com, nr_telefone_cel,
ds_email)  values (3, 3, 'Autonomo 1', 'Log. Auton. 1', null, null, null, null,
null, null)
INSERT INTO cliente (cd_cliente, nr_localidade, nm_cliente, ds_logradouro,
ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com, nr_telefone_cel,
ds_email)  values (4, 4, 'Autonomo 2', 'Log. Auton. 2', null, null, null, null,
null, null)
INSERT INTO cliente (cd_cliente, nr_localidade, nm_cliente, ds_logradouro,
ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com, nr_telefone_cel,
ds_email)  values (5, 1, 'Autonomo 3', 'Log. Auton. 3', null, null, null, null,
null, null)
INSERT INTO cliente (cd_cliente, nr_localidade, nm_cliente, ds_logradouro,
ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com, nr_telefone_cel,
ds_email)  values (6, 2, 'Autonomo 4', 'Log. Auton. 4', null, null, null, null,
null, null)
```

```
INSERT INTO cliente (cd_cliente, nr_localidade, nm_cliente, ds_logradouro,
ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com, nr_telefone_cel,
ds_email)  values (7, 3, 'Autonomo 5', 'Log. Auton. 5', null, null, null, null,
null, null)
INSERT INTO cliente (cd_cliente, nr_localidade, nm_cliente, ds_logradouro,
ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com, nr_telefone_cel,
ds_email)  values (8, 4, 'Chefe 2', 'Log. Chefe 2', null, null, null, null, null,
null)
INSERT INTO cliente (cd_cliente, cd_responsavel, nr_localidade, nm_cliente,
ds_logradouro, ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com,
nr_telefone_cel, ds_email)  values (9, 8, 3, 'Empregado 2.1', 'Log. Emp. 2.1',
null, null, null, null, null, null)
INSERT INTO cliente (cd_cliente, cd_responsavel, nr_localidade, nm_cliente,
ds_logradouro, ds_complemento, ds_bairro, nr_telefone_res, nr_telefone_com,
nr_telefone_cel, ds_email)  values (10, 8, 4, 'Empregado 2.2', 'Log. Emp.
2.2',null ,null ,null ,null ,null ,null )
INSERT INTO reserva (cd_reserva, cd_cliente, cd_sala, dt_entrada, dt_saida,
vl_reserva, fl_pago)  values (1, 1, 1, '2017-01-01T01:00:00', '2017-01-
01T23:00:00', 10, 1)
INSERT INTO reserva (cd_reserva, cd_cliente, cd_sala, dt_entrada, dt_saida,
vl_reserva, fl_pago)  values (2, 2, 2, '2017-01-02T01:00:00', '2017-01-
02T23:00:00', 10, 1)
INSERT INTO reserva (cd_reserva, cd_cliente, cd_sala, dt_entrada, dt_saida,
vl_reserva, fl_pago)  values (3, 3, 3, '2017-01-03T01:00:00', '2017-01-
03T23:00:00', 10, 0)
INSERT INTO reserva (cd_reserva, cd_cliente, cd_sala, dt_entrada, dt_saida,
vl_reserva, fl_pago)  values (4, 1, 1, '2016-01-01T13:00:00', '2016-01-
01T17:00:00', 6, 0)
INSERT INTO reserva (cd_reserva, cd_cliente, cd_sala, dt_entrada, dt_saida,
vl_reserva, fl_pago)  values (5, 2, 2, '2016-01-02T14:00:00', '2016-01-
02T20:00:00', 7, 0)
INSERT INTO reserva (cd_reserva, cd_cliente, cd_sala, dt_entrada, dt_saida,
vl_reserva, fl_pago)  values (6, 3, 3, '2016-01-03T15:00:00', '2016-01-
03T18:00:00', 8, 0)
INSERT INTO multa (cd_multa, cd_reserva, vl_multa, dt_pagto)  values (1, 1, 5,
'2017-02-01T12:00:00')
INSERT INTO multa (cd_multa, cd_reserva, vl_multa, dt_pagto)  values (2, 2, 15,
'2017-03-01T13:00:00')
```