

Zurrapa – Drinks & Coffee



Miguel Pereira - 43392

Bruno Monteiro - 43994

Manuel Magalhães – 44604

Covilhã – 5 Janeiro 2020

Agradecimentos

O grupo gostaria de agradecer ao professor João Muranho pelo apoio constante, orientação, sinceridade, paciência, compreensão e muito mais. Agradecemos por nos ter ensinado isso e por nos ter passado valores importantíssimos, como honestidade, trabalho árduo e especialmente a aprendizagem de trabalhar de forma autónoma. A sua motivação, conhecimento e disponibilidade ajudou ao nosso grupo a chegar a uma conclusão melhor sobre o ponto de vista que iremos ter perante alguma aplicação envolvendo a matéria de base de dados.

Resumo

- No âmbito da unidade curricular de Base de Dados, foi nos proposto desenvolver um trabalho prático que consiste na elaboração de uma base de dados e suas aplicações cujo tema abrange uma empresa de comércio de bens consumíveis: “Zurrapa - Drinks & Coffee”. Além disso além da elaboração da nossa base de dados também desenvolvemos aplicações que permitem a ligação com a nossa base de dados. Num primeiro passo desenvolvemos os modelos antes de passarmos para as nossas aplicações. Após a modelação, criamos Scripts, tabelas e dados que são inseridos no programa “SQL Server Management Studio” para que se possa criar a nossa base de dados. Após estes últimos passos, desenvolvemos por fim as aplicações que permitem a ligação e gestão da nossa base de dados.

-Palavras chave: Aplicações, Base de dados, Gestão, Ligação, Scripts.

Índice

Agradecimentos	2
Resumo	3
Lista de Abreviaturas	6
Lista de Figuras	6
1.1 Enquadramento	7
1.2 Motivação	7
1.3 Objetivos	8
1.4 Organização do documento	8
2. Desenvolvimento de aplicações cliente/servidor sobre base de dados	10
2.1 Introdução	10
2.2 Aplicações cliente/servidor	12
2.3 SQL Server	13
2.4 Configuração do acesso do servidor	14
2.5 Lazarus	16
3. Modelação	17
3.1 Introdução	17
3.2 Descrição da organização	18
3.3 Modelo Conceptual	20
3.4 Modelo Lógico	21
3.5 Considerações	22
4. Aplicação	23
4.1 Distribuição de Tarefas	23
4.1.1 Tarefa 1	23
4.1.2 Tarefa 2	23
4.1.3 Tarefa 3	23
4.2 Acesso à base de dados	24
4.3 Funcionalidade	29
4.3.1 Descrição Geral	29
4.3.2 Aplicações	30
4.3.2.1 Trata Pedido – Mesa	30
4.3.2.2 Trata Pedido – Balcão	37
4.3.2.3 Trata Armazém	46
5. Conclusões	51

Epílogo	52
Referências Bibliográficas	53
Apêndices	54
1.Script Criar Base de dados	54
2.Script Criar Tabelas.....	55
3.Script Dados Tabelas	61

Lista de Abreviaturas

TCP - Transmission Control Protocol;
IP - Internet Protocol;
IDE - Ambiente de Desenvolvimento Integrado;
SGBD – Sistema de Gestão de Base de Dados.

Lista de Figuras

Figura 1 - Modelo Cliente-Servidor	12
Figura 2 - TCP/IP	15
Figura 3 - Lazarus IDE.....	16
Figura 4 - Conceptual.....	20
Figura 5 - Modelo Lógico	21
Figura 6 - Propriedades TMSSQLTransaction	24
Figura 7 - Propriedades TSQLTransaction	25
Figura 8 - Propriedades SQLQuery (2)	26
Figura 9 - Propriedades SQLQuery (1)	26
Figura 10 - Exemplo 1	27
Figura 11 - Inserir Bar	30
Figura 12 - Página inicial "Trata Pedido-Mesa"	30
Figura 13 - Form criar novo pedido	31
Figura 14 - Consultar Estado mesa (1)	33
Figura 15 - Consultar Estado Mesa (2).....	33
Figura 16 - Pagar Pedido.....	34
Figura 17 - Marcar Mesa "Limpa"	36
Figura 18 – Bar	37
Figura 19 – Login.....	37
Figura 20 – Trata Pedido-Balcão.....	37
Figura 21 – Marcar Pedido Satisfeito	37
Figura 22 - Form Login	38
Figura 23 - Bar.....	39
Figura 24 - Página inicial "Trata Pedido-Balcão"	41
Figura 25 - Ver estados pedidos	42
Figura 26 - Mostrar detalhes do pedido.....	42
Figura 27 - Marcar Pedido como satisfeito	44
Figura 28 - Pós marcar Pedido como satisfeito	44
Figura 29 - Selecionar Pedido	45
Figura 30 - Informações do pedido.....	45
Figura 31 - Página inicial Armazém	46
Figura 32 - Consultar Stock Bar1	47
Figura 33- Consultar Valor Stock dos locais	49
Figura 34 - Consultar Despesas/Lucro dos bares	49
Figura 35 - Despesas, Recebido e lucro dos bares.....	50

1.Introdução

1.1 Enquadramento

- No âmbito da unidade curricular de Base de Dados, surge o trabalho prático, subordinado ao tema: “Zurrapa – Drinks & Coffee”. Procura-se, com a elaboração deste trabalho prático, criar uma base de dados que represente com a mais precisa fidelidade uma “situação de comércio”. Neste contexto procura-se representar, por exemplo, o comércio realizado a níveis de bares. Mas para que seja feita uma boa representação também é necessário criar aplicações que permitem uma boa gestão do nosso comércio. Procura-se então responder a todas estas questões com o desenvolvimento do nosso trabalho prático.

1.2 Motivação

- Com o trabalho prático foi nos proposto representar uma situação de comércio. Tendo isto em conta, desenvolvemos este trabalho com vista a aprofundar os nossos conhecimentos na unidade curricular de Base de Dados e também motivados a representar com maior fidelidade a empresa.

1.3 Objetivos

- 1- Elaboração de um Modelo Conceptual
- 2- Elaboração de um Modelo Lógico
- 3- Elaboração de uma base de dados usando scripts que se divide nos seguintes passos:
 - 3.1 -> Criar a base de dados;
 - 3.2 -> Criar tabelas e restrições de acordo com os nossos modelos;
 - 3.3 -> Inserir dados nas tabelas.
- 4 – Desenvolver aplicações no IDE Lazarus que procurem responder às seguintes questões:
 - 4.1 -> Aplicação que trata o Pedido-Mesa;
 - 4.2 -> Aplicação que trata o Pedido-Balcão;
 - 4.3 -> Aplicação que trata o Armazém.

1.4 Organização do documento

- O nosso relatório encontra-se estruturado da seguinte forma (por ordem de aparecimento):
 - > Capa, que possui o título do trabalho bem como os alunos que desenvolveram e em que unidade curricular foi realizada.
 - > Agradecimentos.
 - > Resumo, que possui um breve texto que visa explicar de forma breve a essência do nosso trabalho prático.
 - > Índice Geral, que indica em que páginas se encontram todos os temas e subtemas que existe no relatório.
 - > Lista de Abreviaturas
 - > Lista de Figuras

-> Introdução, que abrange diversos subtemas tais como: Enquadramento, Motivação, Objetivos e Organização do documento. Visa a contextualizar o nosso trabalho.

-> Desenvolvimento de aplicações cliente/servidor sobre bases de dados, este tema procura introduzir e explicar as aplicações que efetuam as ligações entre o cliente e o servidor bem como a configuração de acesso ao próprio servidor pelas aplicações e IDE utilizado para o desenvolvimento dessas mesmas aplicações.

-> Modelação, este tema procura explicar o processo de modelação, ou seja, explicar como realizamos os nossos modelos conceptual e lógico bem como explicar em que medida estes se enquadram no nosso trabalho.

-> Aplicação, com este tema queremos explicar as seguintes questões: Distribuição de tarefas, que inclui uma lista que representa os diversos papéis de cada elemento do grupo no desenvolvimento das aplicações; Acesso à base de dados, que procura explicar como foi efetuado o acesso à base de dados com excertos de código ilustrativos; Funcionalidade, que abrange uma descrição geral sobre a funcionalidade das aplicações no contexto geral da base de dados e uma descrição pormenorizada sobre a funcionalidade de cada aplicação.

-> Conclusões, indica o que foi conseguido e o que não foi conseguido com justificações.

-> Epílogo, que inclui uma reflexão crítica sobre a disciplina.

-> Referências Bibliográficas.

-> Apêndices, que inclui os scripts.

2. Desenvolvimento de aplicações cliente/servidor sobre base de dados

2.1 Introdução

Uma Base de Dados é uma coleção de dados partilhados, interrelacionados e usados para múltiplos objetivos. O primeiro sistema de armazenamento automático de dados foi o sistema de ficheiros.

Num sistema de ficheiros, cada aplicação cria e mantém os ficheiros com os dados necessários para a sua execução. Este sistema possui alguns problemas tais como: alto nível de redundância, inconsistência de informação, inflexibilidade, acessos concorrentes (diversas aplicações podem partilhar o acesso aos ficheiros necessários para a sua execução), isolamento e integridade dos dados e elevados custos de manutenção.

Existe um intermediário entre o nível aplicacional e a base de dados que evita a manipulação direta por parte de cada aplicação cliente. Este intermediário é o Sistema de Gestão de Base de Dados (SGBD).

Um SGBD é um conjunto de programas que permite desempenhar as tarefas de armazenamento e manipulação de dados, fornecendo aos programadores e utilizadores finais os dados tal como eles são pedidos. É a única entidade responsável pela segurança, integridade e validade dos dados armazenados.

O acesso aos dados implica obrigatoriamente a comunicação com a SGBD que reserva para si os privilégios de acesso físico à base de dados e aos ficheiros que a compõem.

Ao contrário dos sistemas de ficheiros, é possível que uma aplicação possa ser modificada, sem que isso implique alterações nos restantes programas que partilham a utilização da mesma informação.

Tipicamente as aplicações de bases de dados seguem um modelo cliente-servidor, ou seja, um servidor contém os dados e executa o Sistema de Gestão de Base de Dados e os clientes, ligados por uma rede de intercomunicações, executam os programas de aplicação que fazem a interface com o utilizador e o acesso à base de dados.

2.2 Aplicações cliente/servidor

Num modelo cliente-servidor, o servidor contém os dados e executa o SGBD e os clientes executam os programas de aplicação. Geralmente os clientes e servidores comunicam através de uma rede de computadores. Um servidor é um host que executa um ou mais serviços ou programas que compartilham recursos com os clientes. Um cliente não compartilha qualquer recurso, apenas solicita um conteúdo.

Segundo este modelo, o cliente executa os programas de aplicação, no entanto o cliente não acede de forma direta os dados, existindo um intermediário entre os dados e as aplicações que é o SGBD. O SGBD certifica-se da integridade, segurança e validade dos dados armazenados. Quando uma aplicação cliente é executada, o SGBD trata de proteger os dados dos acessos não autorizados de aplicações não fidedignas e garante que todas as aplicações são de utilizadores devidamente credenciados. Caso sejam, o SGBD permite o acesso aos dados e o cliente obtém os dados desejados.

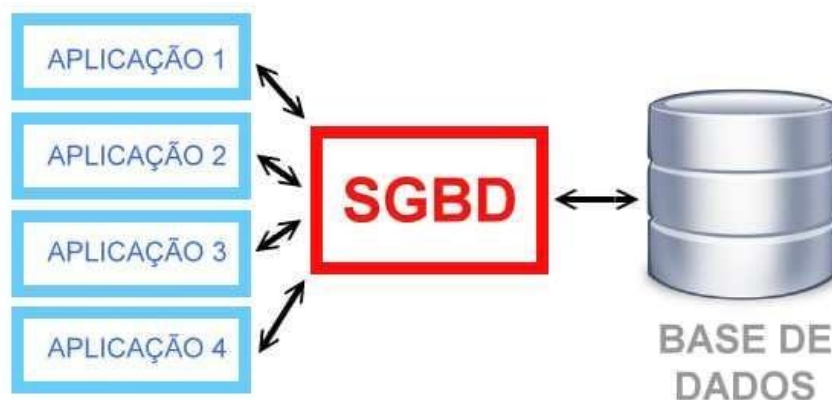


Figura 1 - Modelo Cliente-Servidor

2.3 SQL Server

No nosso trabalho prático utilizamos o Microsoft SQL Server como o nosso sistema de gestão de base de dados. Utilizamos o componente *Microsoft SQL Server Management Studio* para podermos criar a nossa base de dados com as devidas tabelas e respetivos dados.

O Microsoft SQL Server é um sistema de gestão de base de dados relacionais que suporta um número de aplicações, incluindo gestão de transações. É construído à volta da linguagem de programação SQL, que é usada para gerir bases de dados e dados de queries. A linguagem SQL é dividida em diversos subconjuntos de acordo com as operações que queremos efetuar na nossa base de dados:

- > DDL – Data Definition Language: Utilizada para definir a estrutura da base de dados e da informação a armazenar. No contexto do nosso trabalho foi utilizada no Microsoft SQL Server Management Studio, usando os scripts, para definir tabelas e elementos associados.

- > DML – Data Manipulation Language: Utilizada para obter, armazenar, alterar ou eliminar informação da base de dados. No contexto do nosso trabalho foi utilizada no Microsoft SQL Server Management Studio, usando os scripts, para inserir, atualizar ou eliminar quaisquer dados que existam nas tabelas já definidas.

- > DQL – Data Query Language : Utilizada para efetuar consultas sobre dados que pertencem a alguma tabela definida na base de dados. No contexto do nosso trabalho foi utilizada no Microsoft SQL Server Management Studio, usando os scripts, para efetuar consultas sobre os dados de uma tabela.

2.4 Configuração do acesso do servidor

- No modelo cliente – servidor podemos distinguir três entidades distintas:

1. O cliente, que com o seu software aplicacional específico é lhe possibilitado a comunicação com o servidor;
2. O servidor, que recebe a mensagem, interpreta-a e devolve uma resposta para o cliente;
3. A rede, componente de hardware que permite a comunicação entre o cliente e o servidor.

Tendo isto em conta definimos a estrutura básica do modelo cliente-servidor, no entanto para que a ligação cliente/servidor seja bem sucedida tem de existir um conjunto protocolos específicos. Este conjunto de protocolos é o TCP/IP ou Transmission Control Protocol/ Internet Protocol. Este conjunto de protocolos possui 4 camadas: Camada de Aplicação, Camada de Transporte (protocolo TCP), Camada de Internet (protocolo IP) e Interface de Rede.

Quando o cliente requisita algum tipo de dados através da aplicação, a camada de aplicação comunica com o protocolo da camada de transporte, neste caso o protocolo TCP.

Os protocolos do nível de transporte (como por exemplo o TCP) fornecem serviços que garantem uma transferência confiável de dados e aplicações entre computadores ou outros equipamentos remotos. Os programas na camada de aplicação usam os protocolos de transporte para contactar com outras aplicações. Para isso, a aplicação interage com o software do protocolo antes de ser feito o contacto. A aplicação que aguarda a conexão informa ao software do protocolo local que está pronta a aceitar a mensagem. Depois na camada de Internet, temos o Internet

Protocol (IP) que utiliza os pacotes recebidos na camada do transporte (provenientes do protocolo TCP) adiciona um endereço virtual, ou seja, adiciona o endereço do computador que está a enviar os dados e o endereço do computador que vai receber os dados. Finalmente a última camada, Interface de Rede pega os pacotes enviados pela camada de Internet e envia-os através da rede. Desta forma o cliente irá receber todos os dados que deseja.



Figura 2 - TCP/IP

2.5 Lazarus

- O nosso trabalho prático foi desenvolvido no IDE Lazarus. Lazarus é um ambiente de desenvolvimento integrado desenvolvido para o compilador Free Pascal. Free Pascal é um compilador que roda em Linux, Windows, Mac OS entre outros. Foi desenhado para compilar código com a sintaxe de Delphi ou dos dialetos Pascal do Macintosh e gerar executáveis para diferentes plataformas a partir de um mesmo código-fonte.

Lazarus possui suporte para várias bases de dados. O acesso às bases de dados pode ser feito por meio de código ou colocando componentes num formulário ou janela.



Figura 3 - Lazarus IDE

3. Modelação

3.1 Introdução

- Os modelos de dados facilitam a interação com programadores e utilizadores finais. Um modelo bem projetado promove uma melhor compreensão da organização. Construir um modelo de dados é: Identificar, analisar e registar a política da organização acerca dos dados.

A definição de um modelo de dados pode ser feita em 3 níveis:

- 1- Conceptual: Representação fiel da realidade, sem atender a quaisquer constrangimentos impostos pelo modelo informático.
- 2- Lógico: Adaptação do modelo conceptual a um modelo de dados específico, mas independente de qualquer SGBD ou outras considerações físicas.
- 3- Físico: Adaptação do modelo lógico às características do sistema informático.

Existem 2 abordagens para modelação de dados:

- 1- Abordagem da Teoria da Normalização por Codd. Adequada para projetos pequenos.
- 2- Abordagem do Modelo Entidade – Associação.

3.2 Descrição da organização

A empresa Zurrapa-Drinks & Coffee é uma empresa que se dedica exclusivamente ao comércio de vários bens consumíveis, em estabelecimentos. No contexto do enunciado assumimos que a empresa vai gerir os bares da UBI.

A empresa tem empregados que estão subdivididos em várias categorias: empregado de mesa, empregado de balcão e empregado de Armazém. Todos esses empregados possuem uma escala de trabalho, ou seja, cada empregado tem de se apresentar no seu respetivo local de trabalho num dado dia e numa dada hora. Cada local de trabalho possui um empregado responsável.

A empresa gere 3 locais: 2 bares e 1 armazém. Cada local possui um stock, é de extrema importância clarificar que os stocks dos bares e armazém são independentes uns dos outros, sendo ainda identificado o tempo (em minutos) que demora a deslocar-se de um local para o outro.

Os stocks armazenam diversos tipos de produtos que possuem uma certa unidade de medida. O stock de armazém armazena os produtos de forma agregada enquanto que os stocks dos bares os produtos são dispostos de forma individual.

Os produtos que a empresa comercializa estão identificados, possuem uma designação, preço de custo e preço de venda.

Cada bar possui várias mesas numeradas. Cada mesa possui um estado que se subdivide nas seguintes categorias: limpa, ocupada e suja.

Um cliente pode realizar vários pedidos. Cada pedido possui o número do empregado responsável pelo pedido, a mesa, a localização do bar, o dia e a hora do pedido. Cada pedido tem um e só um estado que pode vir a ser alterado, esses estados dividem-

se em: Insatisfeito, satisfeito, parcialmente pago e pago. Quando é realizado um pedido caso não haja empregado responsável pela mesa em questão o empregado responsável pelo pedido passa a ser o responsável pela mesa.

Em cada pedido o cliente pode solicitar um ou mais produtos. Num determinado produto de um dado pedido é guardada a quantidade pedida, quantidade consumida e quantidade paga. Entende-se por quantidade pedida, a quantidade que o cliente pede ao empregado no momento da realização do pedido. A quantidade consumida resume-se exclusivamente ao que o é servido ao cliente, ou seja, a quantidade que chega à mesa do mesmo. A quantidade paga refere-se à quantidade já paga, de um determinado produto, pelo cliente.

3.3 Modelo Conceptual

Um modelo conceptual de dados é a representação de um conjunto de objetos e das suas associações. Durante este processo de abstracção, objetos relevantes, associações entre eles e características (atributos) de objetos e associações são seleccionadas. Atributos de objetos e correspondentes associações têm valores específicos que pertencem a conjuntos denominados domínios. Um valor de um dado atributo pode variar ao longo do tempo, mas pertencendo sempre ao mesmo domínio. Este modelo é o modelo ao mais alto nível, é o modelo que se encontra o mais perto da realidade. Em baixo encontra-se o respetivo Diagrama de Entidade Associação do nosso projeto que foi implementado como modelo Conceptual:

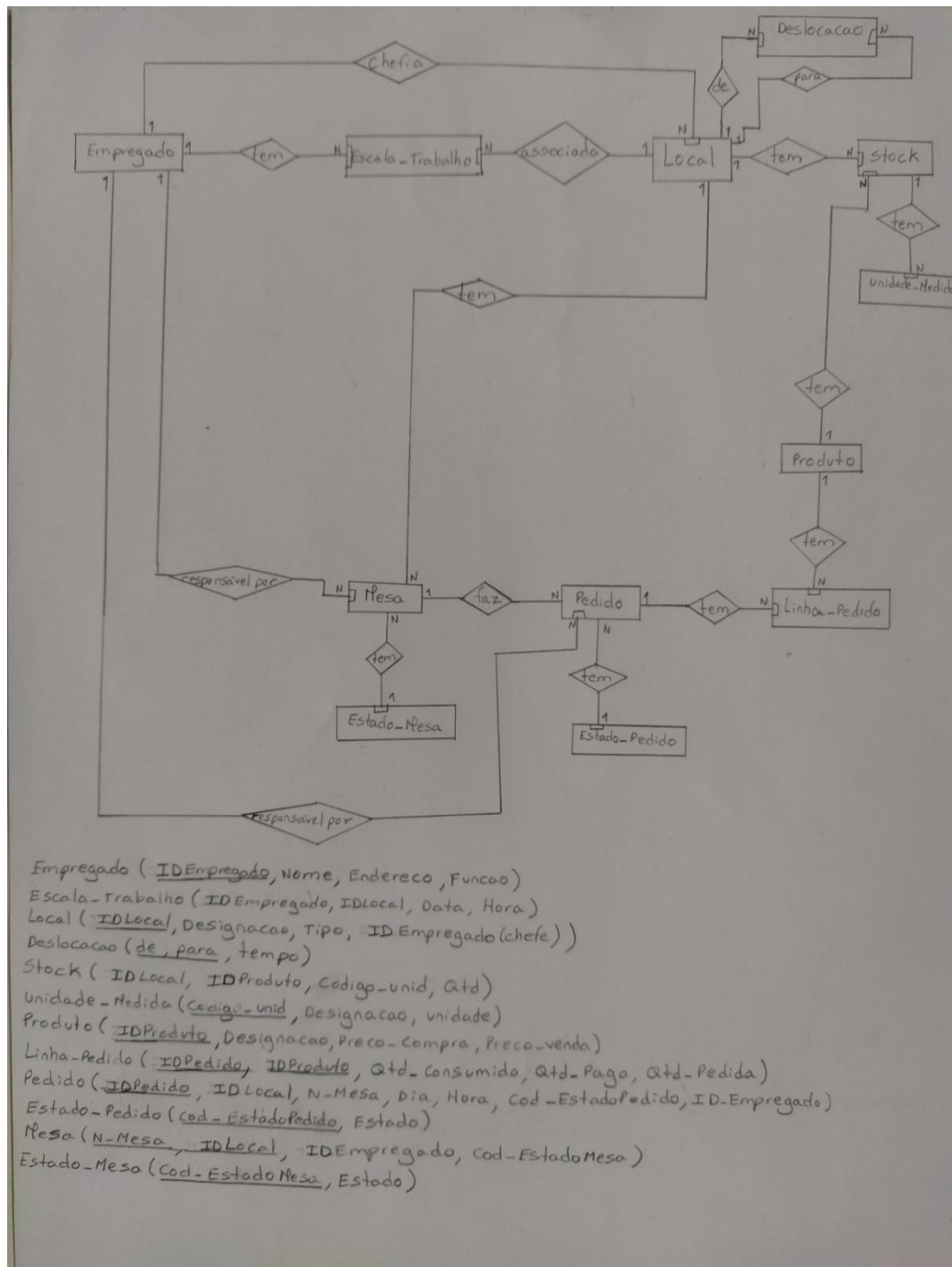


Figura 4 - Conceptual

3.4 Modelo Lógico

O modelo lógico é um modelo que mostra as ligações entre as tabelas da base de dados, as chaves primárias, chaves estrangeiras e os componentes de cada uma. É neste modelo que se aplica a técnica de normalização, com o objetivo de reduzir a redundância dos dados e evitar anomalias de implementação. Na imagem em baixo apresentamos o modelo lógico do nosso projeto.

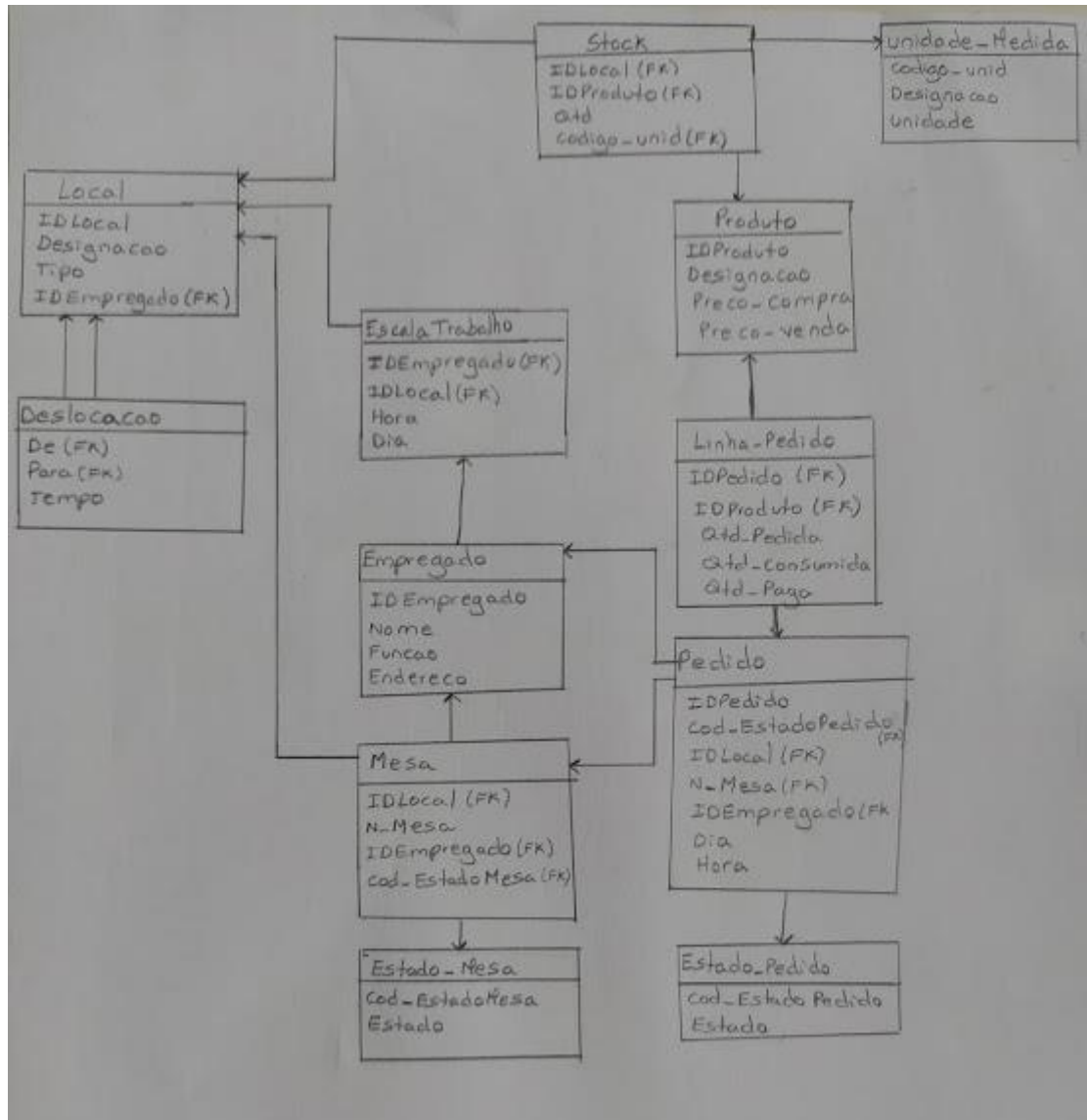


Figura 5 - Modelo Lógico

3.5 Considerações

As mesas podem possuir três estados já identificados anteriormente. A mesa estar ocupada significa que tem pedidos não pagos, ou seja, ainda se encontram clientes na mesa. Quando todos os pedidos pagos, a mesa passa para o estado “suja”. Nesse estado aguarda-se que o empregado limpe a mesa, passando esta ao estado de “limpa”. Na passagem para este estado, a mesa deixa de ter um responsável.

O pedido pode ter quatro estados indicados acima. O pedido está “insatisfeito” enquanto o empregado de balcão não o prepara. Posteriormente passa para o estado de “satisfeito”. Nesse estado o pedido é entregue à mesa. O pagamento é efetuado na aplicação “Trata pedido - mesa”. Se é pago apenas uma parte do pedido este fica no estado “parcialmente pago”. Finalmente, quando todos os produtos pedidos estão totalmente pagos, este passa para o estado “pago”.

4. Aplicação

4.1 Distribuição de Tarefas

- As tarefas do trabalho prático foram divididas por todos os elementos do grupo. Foram divididas no seguinte esquema:

4.1.1 Tarefa 1

- A tarefa 1 foi atribuída ao aluno Miguel Pereira, nº43392 e consiste na criação da aplicação que diz respeito ao tema: “Trata Armazém”. A aplicação foi desenvolvida em Lazarus.

4.1.2 Tarefa 2

- A tarefa 2 foi atribuída ao aluno Bruno Monteiro, nº43994 e consiste na criação da aplicação que diz respeito ao tema: “Trata Pedido – Mesa”. A aplicação foi desenvolvida em Lazarus.

4.1.3 Tarefa 3

-A tarefa 3 foi atribuída ao aluno Manuel Magalhães, nº44604 e consiste na criação da aplicação que diz respeito ao tema: “Trata Pedido – Balcão”. A aplicação foi desenvolvida em Lazarus. Este aluno também desenvolveu o login para cada aplicação.

4.2 Acesso à base de dados

-O acesso à base de dados efetuada através das aplicações subdivide – se em várias etapas.

Numa primeira etapa, quando um form é criado e para que este tenha acesso aos diferentes dados da base de dados irá necessitar de um `TMSSQLConnection`. Um `TMSSQLConnection` é um conector de base de dados do Microsoft SQL Server. No entanto é necessário ter em atenção que propriedades possui, ou seja, no “Object Inspector” do Lazarus deve-se verificar se as propriedades estão bem definidas. Algumas propriedades a ter atenção são: `DatabaseName`, `HostName`, `Name`, `Password`, `Transaction` e `UserName`. No nosso trabalho tivemos que introduzir alguns valores nestas propriedades para que o `TMSSQLConnection` funcionasse de acordo com os nossos objetivos. Apenas introduzimos valores nos campos `DatabaseName` e `Transaction` visto que as restantes propriedades são inseridas no momento que é efetuado o login em cada aplicação. Na imagem em baixo verifica-se as propriedades do `TMSSQLTransaction`:

Properties	Events	Favorites	Restricted
CharSet			
Connected	<input type="checkbox"/> (False)		
DatabaseName	Zurrapa		
HostName			
KeepConnection	<input type="checkbox"/> (False)		
> LogEvents	[detCustom, detPrepare, de		
LoginPrompt	<input type="checkbox"/> (False)		
Name	MSSQLConnection1		
> Options	[]		
Params	(TStrings)		
Password			
Role			
Tag	0		
> Transaction	SQLTransaction1		
UserName			

Figura 6 - Propriedades TMSSQLTransaction

Nota: É importante notar que temos uma transaction visto que para explicar o acesso à base de dados, temos de usar um exemplo, ou seja, usamos uma aplicação para podermos explicar o processo de ligação da aplicação com a base de dados. A nossa `TMSSQLConnection` fica identificada com o nome: “`MSSQLConnection1`”.

Numa segunda etapa, estabelecemos um TSQLTransaction para que possa existir uma “rede de transações” na base de dados. Desta forma será possível termos transações, sem a existência do TSQLTransaction isto não seria possível. Nas propriedades da TSQLTransaction temos que identificar a base de dados, ou seja, temos que estabelecer a base de dados para que possam existir transações. A nossa TSQLTransaction fica com o nome: “SQLTransaction1”.

Neste exemplo incluímos a base de dados definida em cima:

Properties	Events	Favorites	Restricted
Action	caRollback		
Active	<input type="checkbox"/> (False)		
> Database	MSSQLConnection1		
Name	SQLTransaction1		
> Options	[]		
Params	(TStringList)		
Tag	0		

Figura 7 - Propriedades TSQLTransaction

Finalmente caso queiramos aceder a algum tipo de dados provenientes da nossa base de dados iremos necessitar de um TSQLQuery. A partir deste poderá ser definido qualquer instrução do tipo SQL para se poder seleccionar qualquer tipo de informação proveniente da base de dados. É extremamente importante notar que sem um TSQLTransaction não seria possível termos um Query funcional, pois as transações são definidas pelo TSQLTransaction. O SQLQuery precisa de ter definido o MSSQLConnection e o SQLTransaction para que funcione corretamente.

O SQLQuery ficou com o nome: "SQLQuery1" e tem as seguintes propriedades:

Properties	Events	Favorites	Restricted
Active	<input type="checkbox"/> (False)		
AutoCalcFields	<input checked="" type="checkbox"/> (True)		
> Database	MSSQLConnection1		
DataSource			
DeleteSQL	(TStringList)		
FieldDefs	9 items		
FileName			
Filter			
Filtered	<input type="checkbox"/> (False)		
IndexDefs	0 items		
IndexFieldName			
IndexName	DEFAULT_ORDER		
InsertSQL	(TStringList)		
MaxIndexesCount	4		
Name	SQLQuery1		
> Options	[]		
PacketRecords	50		
ParamCheck	<input checked="" type="checkbox"/> (True)		
Params	0 items		
ParseSQL	<input checked="" type="checkbox"/> (True)		
ReadOnly	<input type="checkbox"/> (False)		
RefreshSQL	(TStringList)		
> Sequence	(TSQLSequence)		

Figura 9 - Propriedades SQLQuery (1)

ServerFilter	
ServerFiltered	<input type="checkbox"/> (False)
ServerIndexDefs	0 items
SQL	(TStringList)
Tag	0
Transaction	SQLTransaction1
UniDirectional	<input type="checkbox"/> (False)
UpdateMode	upWhereKeyOnly
UpdateSQL	(TStringList)
UsePrimaryKeyAsFilter	<input checked="" type="checkbox"/> (True)

Figura 8 - Propriedades SQLQuery (2)

Todas as imagens em cima foram usadas com o contexto de explicar como se processa a ligação/acesso à base de dados.

Tal como mencionado em cima para podermos explicar de forma clara o acesso à base de dados utilizamos uma aplicação para exemplificar. Foi usado o form “Trata Pedido-Balcão” para exemplificar da aplicação Trata Pedido – Balcão.

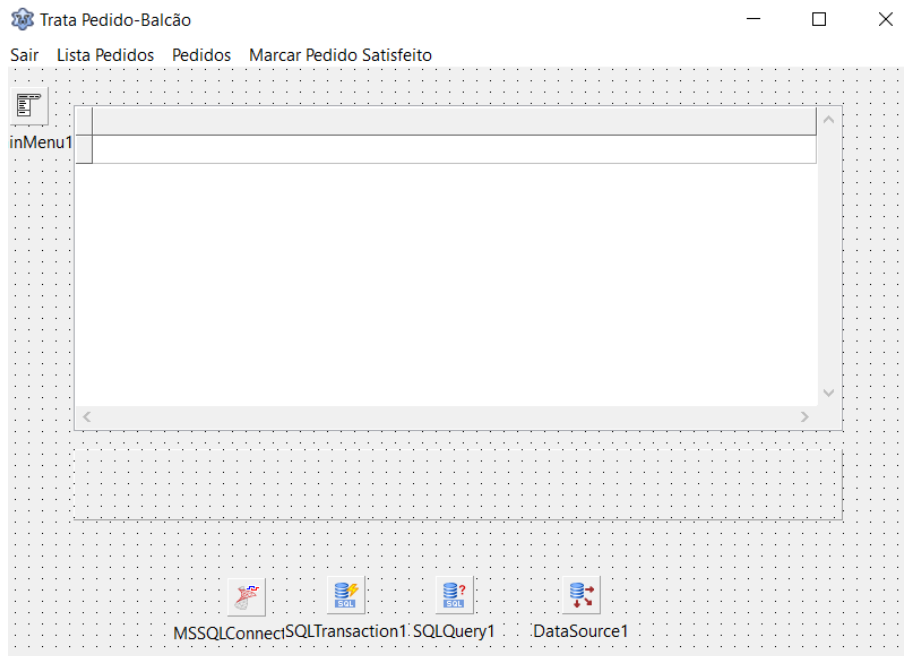


Figura 10 - Exemplo 1

Incluímos também alguns excertos que explicam a configuração de acesso à base de dados. O primeiro excerto mostra um exemplo da utilização do SQLQuery (que é utilizada para selecionar informações da base de dados).

```

if not GetNumBar(numBar) then
    Application.Terminate;
    num_Bar:=numBar;
    setnumBar(num_Bar);
    with SQLQuery1 do
        begin
            DataBase := MSSQLConnection1;
            Transaction := MSSQLConnection1.Transaction;
            SQL.Clear;
            SQL.Add('Select P.IDPedido, P.N_Mesa, Dia, Hora,
E.Estado');
            SQL.Add('From Pedido P, Estado_Pedido E');
            SQL.Add('Where P.Cod_EstadoPedido=E.Cod_EstadoPedido AND
P.IDLocal='+IntToStr(numBar));
        end;
        with SQLQuery1 do
            begin
                if Active then
                    Close;
                Open;
            end;

```

O segundo excerto mostra de forma geral como é feito o acesso à base de dados. Num primeiro passo configuramos a ligação com a base de dados (através do login). Num segundo passo configuramos o componente SQLTransaction. Num terceiro passo definimos o nome da nossa base de Dados (o hostname obtido através do login vai substituir o valor HostName do MSSQLConnection do form). Num último passo ligamos ao servidor (ligação à base de dados) e ativamos o componente de transações (SQLTransaction).

```
//Configurar o acesso à Base de Dados
With MSSQLConnection1 do
begin
    Connected := False;

    HostName := aHostName;
    UserName := aUserName;
    Password := aPassword;

    Params.Clear;
    Params.Add('AutoCommit=true');
    Params.Add('TextSize=16777216');
    Params.Add('ApplicationName=EX1');
    // Connected := True; // See below
end;

//Configurar o component SQLTransaction
with SQLTransaction1 do
begin
    if Active then
        Active := False;
        DataBase := MSSQLConnection1;

    //Base de dados...
    MSSQLConnection1.DatabaseName:= aDatabase;
end;

//Aceder aos dados
Try
    //1. Ligar ao servidor
    with MSSQLConnection1 do
    begin
        if Connected then
            Connected := False;
            Connected := True;
        end;

    //2. Activar a componente de transacções
    with SQLTransaction1 do
    begin
        if Active then
            Active := False;
            Active := True;
        end;
```

4.3 Funcionalidade

4.3.1 Descrição Geral

No contexto do projeto foram desenvolvidas três aplicações que se subdividem nos seguintes temas: Trata Pedido-Mesa, Trata Pedido-Balcão, Trata Armazém.

Na aplicação Trata Armazém, o empregado de armazém pode visualizar diferentes informações, tais como:

- consultar o stock do armazém e dos bares;
- Recarregar Stock dos Bares e armazém, o que será importante em certas situações em que o stock dos bares é insuficiente para satisfazer pedidos;
- Ver valor do Stock;
- Consultar Despesas e valor Recebido dos bares.

Na aplicação Trata Pedido-Mesa, o empregado de mesa pode gerir pedidos e mesas mais especificamente:

- Criar um novo Pedido para uma dada mesa;
- Consultar estados das mesas;
- Limpar mesas;
- Pagar pedidos.

Na aplicação Trata Pedido-Balcão, o empregado pode gerir o estado dos pedidos conforme são realizados:

- Visualizar todos os pedidos de um bar;
- Visualizar o estado desses pedidos;
- Marcar pedidos insatisfeitos como satisfeitos (fornecer os produtos que provêm do stock do bar).

Todas as aplicações possuem um menu de login que se encontra explicado no ponto 4.3.2.2.

4.3.2 Aplicações

4.3.2.1 Trata Pedido – Mesa

A aplicação “Trata Pedido-Mesa” é a aplicação responsável pela criação de novos pedidos, consultar o estado dos pedidos, marcar a mesa como limpa e pagar pedidos.

Quando a aplicação é iniciada e após o login, o empregado deve inserir o número do Bar que se encontra a trabalhar. Após clicar “ok” e caso insira um número de Bar válido é apresentada a interface apresentada na imagem abaixo.

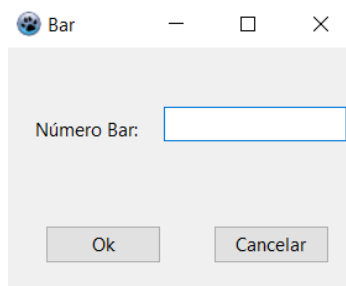


Figura 11 - Inserir Bar



Figura 12 - Página inicial "Trata Pedido-Mesa"

Inicialmente é apresentado uma tabela que contém todas as mesas daquele bar e o seu respetivo “código estado” e “estado”. Para apresentar estes valores na tabela foi realizada a seguinte instrução de código.

```
with SQLQuery1 do
begin
    DataBase := MSSQLConnection1;
    Transaction := MSSQLConnection1.Transaction;
    SQL.Clear;
    SQL.Add('Select M.N_Mesa, M.Cod_EstadoMesa, E.Estado');
    SQL.Add('From Mesa M, Estado_Mesa E');
    SQL.Add('Where M.Cod_EstadoMesa=E.Cod_EstadoMesa AND
M.IDLocal=' + IntToStr(num_Bar));
end;
with SQLQuery1 do
begin
    if Active then
        Close;
    Open;
end;
```

Ao efetuar o novo pedido é verificado se já existe um responsável pela mesa em questão. Caso isso não se verifique, o empregado que realizou o pedido irá ficar como responsável pelo pedido.

Novo Pedido

IDEmpregado:

Nº Mesa:

Dia: 05/01/2021

Hora: 16:48:23

Produto

Designacao
Café
Cerveja
Água 1.5L
Água 0.5L

ProdutoID	Qtd

Ok Cancelar

Figura 13 - Form criar novo pedido

O empregado deverá indicar qual é o seu ID e o número de mesa a que corresponde o pedido. A hora e dia do pedido é gerada automaticamente através das funções: *TimeToStr(Time)* e *DateToStr(Date)*, respetivamente. Posteriormente é possível clicar no produto que se pretende inserir e depois introduzir a quantidade que se pretende. Quando selecionados todos os produtos pretendidos clica-se no botão “ok” e o programa irá executar uma série de verificações.

Primeiramente, verifica se existe quantidade em Stock no bar para satisfazer a quantidade desejada pelo cliente. Caso não exista, o sistema verifica se o produto tem stock em armazém para indicar ao Cliente quanto tempo irá demorar a ser reposto o stock do armazém.

```
if(qtd_StockBar<qtdPedida) then
begin
    //Verifica que quantidade existe em armazém e converte em unidade
    qtd_Armazem:=qtd_Armazem*Unidade;
    if(qtd_Armazem<qtdPedida) then
    begin
        Panel1.Caption:='Stock Bar Insuficiente! Produto não existente
em armazém';
    else
        //verifica quanto tempo demor a repor o stock do bar
        with SQLQuery1 do
        begin
            DataBase := Form1.MSSQLConnection1;
            Transaction := Form1.MSSQLConnection1.Transaction;
            SQL.Clear;
            SQL.Add('Select Tempo');
            SQL.Add('From Deslocacao');
            SQL.Add('Where De = 3 AND Para='+IntToStr(num_Bar));
        end;
        with SQLQuery1 do
        begin
            if Active then
                Close;
            Open;
        end;

        tempo:= SQLQuery1.Fields[0].AsInteger;
        Panel1.Caption:='Stock Bar Insuficiente! Produto existente em
armazém demora '+IntToStr(tempo)+'min a ser reposto';
    end;
```

Seguidamente, se a quantidade do bar for suficiente para satisfazer o pedido este é realizado, mas antes verificamos se a mesa está limpa. Caso esteja marcamos a mesa como “ocupada”. Finalmente, é criado um pedido novo e na “Linha_Pedido” é introduzido todos os produtos e quantidades pedidas.

```
//Criar novo Pedido
//Prepara o SQL a submeter à BD
stSQLText := 'Insert into Pedido(IDPedido, Cod_EstadoPedido, IDLocal,
N_Mesa, IDEmpregado, Dia, Hora) ';
stSQLText:=stSQLText+'values
(:IDPedido,:Cod_EstadoPedido,:IDLocal,:N_Mesa,:IDEmpregado,:Dia,:Hora)';
SQLQuery1.sql.Text:= stSQLText;

SQLQuery1.Prepare;
SQLQuery1.ParamByName('IDPedido').AsInteger:= ID_Pedido;
SQLQuery1.ParamByName('Cod_EstadoPedido').AsInteger:= 1;
SQLQuery1.ParamByName('IDLocal').AsInteger:= num_Bar;
SQLQuery1.ParamByName('N_Mesa').AsInteger:= N_Mesa;
SQLQuery1.ParamByName('IDEmpregado').AsInteger:= ID_Empregado;
SQLQuery1.ParamByName('Dia').AsString:= DateToStr(Date);
SQLQuery1.ParamByName('Hora').AsString:= TimeToStr(Time);

//Insere na base de dados
SQLQuery1.ExecSQL;
SQLQuery1.SQLTransaction.Commit;
```



```
//Inserere Produtos relacionados com determinado pedido
stSQLText := 'Insert into Linha_Pedido(IDPedido, IDProduto,
Qtd_Pedida, Qtd_Consumida, Qtd_Paga) ' ;
stSQLText:=stSQLText+'values (:IDPedido,:IDProduto, :Qtd_Pedida,
:Qtd_Consumida, :Qtd_Paga) ' ;

SQLQuery1.sql.Text:= stSQLText;
SQLQuery1.Prepare;
SQLQuery1.ParamByName('IDPedido').AsInteger:= ID_Pedido;
SQLQuery1.ParamByName('IDProduto').AsInteger:= IDProduto;
SQLQuery1.ParamByName('Qtd_Pedida').AsInteger:= qtdPedida;
SQLQuery1.ParamByName('Qtd_Consumida').AsInteger:= 0;
SQLQuery1.ParamByName('Qtd_Paga').AsInteger:= 0;
SQLQuery1.ExecSQL;
SQLQuery1.SQLTransaction.Commit;
```

Voltando à interface inicial da aplicação, o empregado ao clicar “Consultar Estado” é indicada as mesas do Bar e os estados das mesmas.

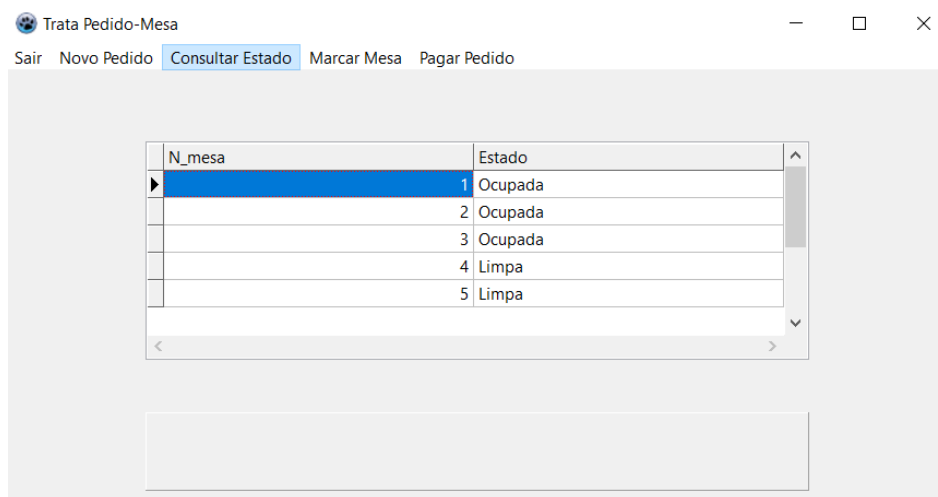


Figura 14 - Consultar Estado mesa (1)

Nesta opção caso o utilizador clique na mesa pretendida, a tabela irá ser atualizada e irá apresentar todos pedidos, em que o estado do mesmo não está marcado como “Pago”, e o respetivo estado do Pedido.

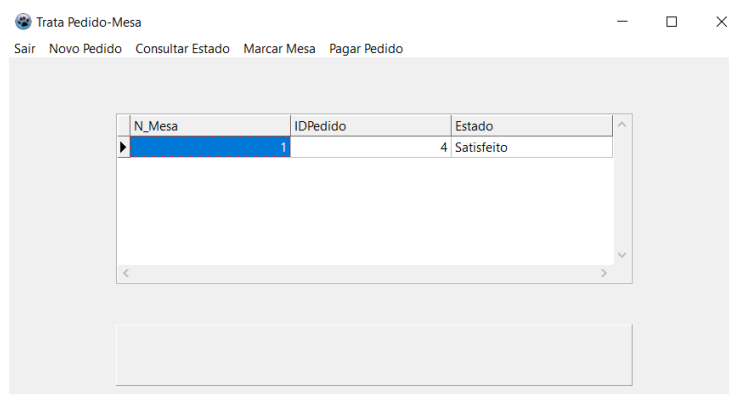


Figura 15 - Consultar Estado Mesa (2)

Outra escolha possível é “Pagar o Pedido”. Ao seleccionar esta opção é aberta uma nova página onde irá permitir ao empregado inserir a quantidade que este pretende pagar de um dado pedido. Na interface inicial é mostrada uma lista com todos os pedidos Satisfeitos, ou seja, que não estão pagos, e com todos os pedidos “Parcialmente Pagos”.

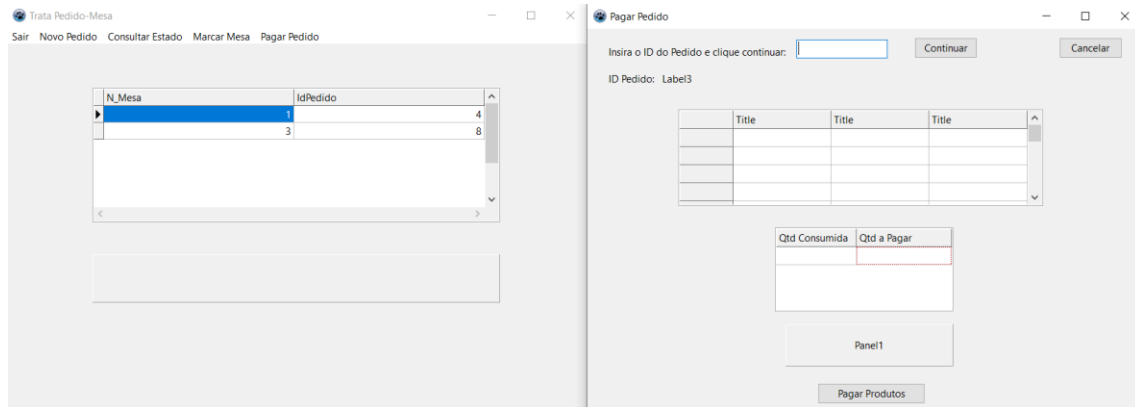


Figura 16 - Pagar Pedido

O funcionário deve introduzir o pedido e clicar “continuar”. Ao fazê-lo o programa irá mostrar todos os produtos do pedido e este poderá introduzir a quantidade a pagar. Quando já tiver introduzido tudo o que deseja deve clicar em “Pagar Produtos”. Caso a quantidade a pagar for menor que a quantidade consumida o pedido é marcado como “Parcialmente Pago”. Depois mais tarde, se pretender assinalar como pago o montante restante dos produtos basta repetir o processo descrito anteriormente. Finalmente, o pedido apenas é marcado como “Pago” quando, em todos os produtos a quantidade consumida é igual à quantidade paga. Assim, o sistema envia a instrução SQL para poder alterar o estado do pedido.

```

if(PedidoPago=indiceok) then
begin
    //altera estado pedido para pago
    stSQLText := 'UPDATE Pedido SET Cod_EstadoPedido= 4 Where
IDPedido='+IntToStr(PedidoID);
    SQLQuery1.Sql.Text:= stSQLText;
    SQLQuery1.Prepare;
    //Insere na base de dados
    SQLQuery1.ExecSQL;
    //Confirma as alterações
    SQLQuery1.SQLTransaction.Commit;

    (....)
end
else
if(PedidoPago<indiceok) then
begin
    stSQLText := 'UPDATE Pedido SET Cod_EstadoPedido= 3 Where
IDPedido='+IntToStr(PedidoID);
    SQLQuery1.Sql.Text:= stSQLText;
    SQLQuery1.Prepare;
    //Insere na base de dados
    SQLQuery1.ExecSQL;
    //Confirma as alterações
    SQLQuery1.SQLTransaction.Commit;
end;

```

A variável *indiceok* representa todos os produtos de um dado pedido, já a variável *PedidoPago* representa todos produtos de um pedido que se encontram pagos.

Para finalizar, caso seja inserido um valor inválido para pagar um determinado produto a janela “Pagar Pedido” é fechada e é mostrada uma mensagem de erro.

```

if ((QtdPagaAnterior+QtdPaga)> QtdConsumida) then
begin
    Form1.Panel1.Caption:='Inserida quantidade a pagar Inválida!'
end;

```

Quando todos os pedidos estão pagos, a mesa é marcada como suja, ou seja o sistema verifica se todos os pedidos de uma mesa estão marcados como pagos, o sistema atualiza a Base de dados através da instrução SQL mostrada a baixo.

```
//Altera estado da mesa para suja
if(PedidosMesaPagos=PedidosMesa) then
begin
    stSQLText := 'UPDATE M SET M.Cod_EstadoMesa = 3 From Mesa As M
INNER JOIN Pedido As P ON P.N_Mesa=M.N_Mesa Where
P.IDPedido='+IntToStr(PedidoID);
    SQLQuery1.Sql.Text:= stSQLText;
    SQLQuery1.Prepare;
    //Insere na base de dados
    SQLQuery1.ExecSQL;
    //Confirma as alterações
    SQLQuery1.SQLTransaction.Commit;
end;
```

PedidosMesasPagos representa todos os pedidos da mesa que estão pagos e *PedidosMesa* o número total de pedidos de uma dada mesa.

Outra opção que o empregado de mesa tem é de marcar a mesa como limpa, ou seja quando o funcionário limpa a mesa este necessita de a marcar como tal. Para isso, na aplicação, ao clicar “Marcar Mesa” e depois selecionar “Limpa” irá aparecer na tabela o número de todas as mesas sujas. Caso pretenda marcar alguma como limpa basta clicar na mesa pretendida e ao voltar a verificar o estado irá ser possível ver que a mesa outrora “suja” encontra-se agora como “limpa”.

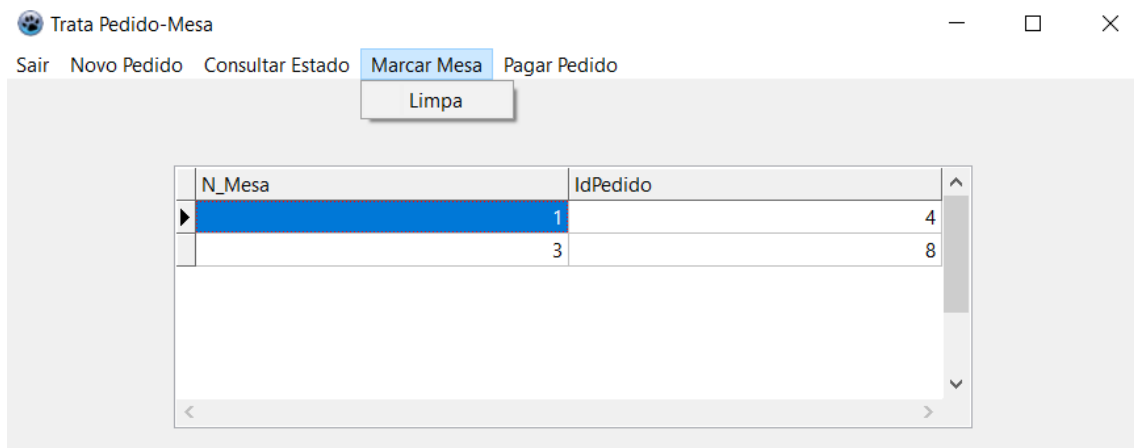


Figura 17 - Marcar Mesa "Limpa"

```
stSQLText := 'UPDATE Mesa SET Cod_EstadoMesa= 1, IDEmpregado=NULL
Where N_Mesa='+IntToStr(N_MesaMarcar);
SQLQuery1.Sql.Text:= stSQLText;
SQLQuery1.Prepare;
//Insere na base de dados
SQLQuery1.ExecSQL;
//Confirma as alterações
SQLQuery1.SQLTransaction.Commit;
```

4.3.2.2 Trata Pedido – Balcão

Esta aplicação trata de resolver o tratamento do pedido efetuado pelos clientes e a sua relação com o balcão, no contexto do problema esta aplicação procura responder aos seguintes tipos de situações:

- 1- O empregado de balcão “tratar” dos pedidos de mesa (marcar pedidos como “satisfeitos”);
- 2- O stock do bar ir atualizando conforme os pedidos ficam com o estado de “satisfeito” (os produtos sejam servidos aos clientes);
- 3- Verificar situações em que o stock é suficiente e insuficiente
- 4- Visualizar os diversos estados de pedidos (Satisfeito , Insatisfeito ,Parcialmente Pago ,Pago);

A aplicação encontra-se dividida no seguintes forms: **Trata Pedido - Balcão**, que contém uma tabela grid e um conjunto de opções a selecionar pelo utilizador, conforme a seleção de opções a tabela grid irá apresentar diversas informações, este form é o form principal da aplicação; **Bar**, que contem um form que irá pedir ao utilizador o número do bar que quer visualizar(este form “ativa-se” após um login bem sucedido e tem como principal objetivo dar ao utilizador a opção de escolher o bar para ver as respetivas informações sobre o mesmo no que diz respeito ao tema: “Trata Pedido – Balcão”; **Marcar Pedido Satisfeito**, este form foi feito com o intuito de marcar um pedido como satisfeito, ou seja, o empregado de balcão trata de preencher um pequeno formulário onde deve inserir as quantidades servidas ao cliente(no contexto do form, decidiu-se utilizar a nomeação “quantidade consumida” e finalmente o **Login** que diz respeito ao método de login, onde o utilizador terá que introduzir um certo conjunto de credenciais para poder avançar na aplicação.

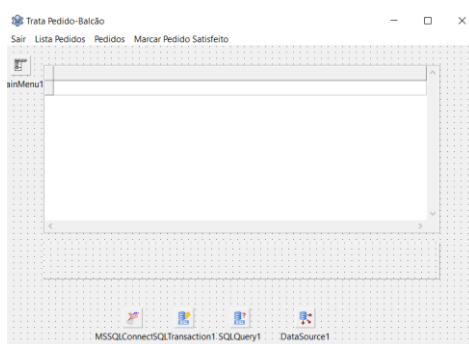


Figura 20 – Trata Pedido-Balcão

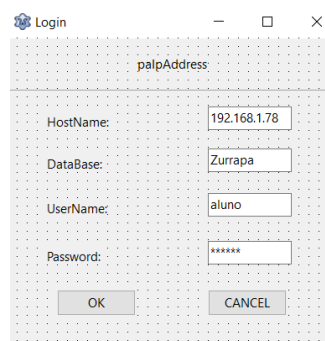


Figura 19 – Login

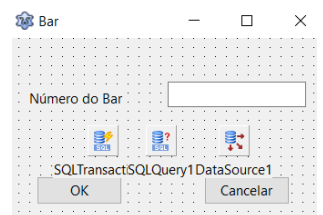


Figura 18 – Bar

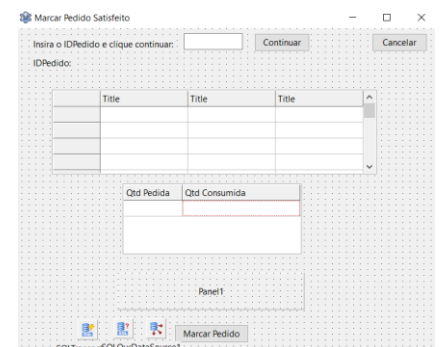


Figura 21 – Marcar Pedido Satisfeito

Após a entrada na aplicação, o utilizador terá que passar por um sistema de login, no qual terá que inserir os dados pedidos pelo form. Caso algum dado seja introduzido de forma incorreta ou tenha um valor inválido a aplicação irá apresentar uma mensagem de erro específica e irá terminar a aplicação de forma imediata. Após a execução inicial o form irá apresentar também o IP do utilizador, ele terá que efetuar um login na sua conta e se as credenciais estiverem corretas, será fornecido acesso ao resto da aplicação. A função usada para verificar as credenciais é a que se encontra no excerto de código abaixo. Esta função foi posteriormente chamada no início do form principal(Trata Pedido-Balcão).

Figura 22 - Form Login

```
Function GetLoginData(var aHostName, aDatabase, aUserName, aPassword
:String):Boolean;
begin
    //Inicializar os parâmetros
    Result := False;
    aHostName := '';
    aDatabase := '';
    aUserName := '';
    aPassword := '';
    Try
        With TForm5.Create(Nil) do
            Try
                Result := ShowModal = mrOk;
                if Result then
                    begin
                        aHostName := edHostName.Text;
                        aDatabase := edDatabase.Text;
                        aUserName := edUserName.Text;
                        aPassword := edPassword.Text;
                    end
                finally
                    Free
                end;
            except
                Result := False
            end;
        end;
    end;
```

Após um login bem sucedido, será apresentado ao utilizador um form onde deve inserir um dado bar. Pretende-se que o utilizador tenha um maior controlo no que diz respeito ao processamento do tema: “Trata Pedido-Balcão” e essa foi a principal razão pela criação deste form inicial. Assim desta forma o utilizador escolhe um dado bar e serão posteriormente apresentados num outro form, todos os dados relevantes ao tema naquele dado bar. Caso o utilizador prima o botão “Cancelar” a aplicação irá terminar imediatamente. A função GetNumBar permite que através da introdução do dado no form seja possível através de uma instrução SQL selecionar o dado bar que o utilizador escolheu. Isto será de extrema importância no outro form pois é através desta função que serão exibidos os dados desse dado bar. O excerto de código da função GetNumBar encontra-se na página seguinte.

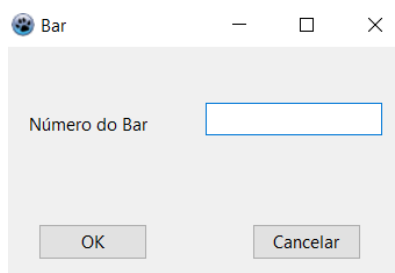


Figura 23 - Bar

```

Function GetNumBar(var numBar:integer):Boolean;
var tempBar:String;
var num_Bar:Integer;
begin
    Result:=False;
    numBar:=0;
    Try
        with TForm2.Create(Nil) do
            Try
                Result:= ShowModal=mrOk;
                if Result then
                    begin
                        num_Bar := StrToInt(Edit1.Text);
                        with SQLQuery1 do
                            begin
                                DataBase := Form1.MSSQLConnection1;
                                Transaction := Form1.MSSQLConnection1.Transaction;
                                SQL.Clear;
                                SQL.Add('Select IDLocal');
                                SQL.Add('From Local');
                                SQL.Add('Where IDLocal =' + FloatToStr(num_Bar) + ' AND
Tipo='Bar'');
                                end;
                                with SQLQuery1 do
                                    begin
                                        if Active then
                                            Close;
                                            Open;
                                        end;
                                    tempBar:=SQLQuery1.Fields[0].AsAnsiString;
                                    if (tempBar='') then
                                        Application.Terminate
                                    else
                                        numBar:=num_Bar;
                                    end;
                                finally
                                    Free
                                end;
                            except
                                Result:=False
                            end;
                        end
                    end

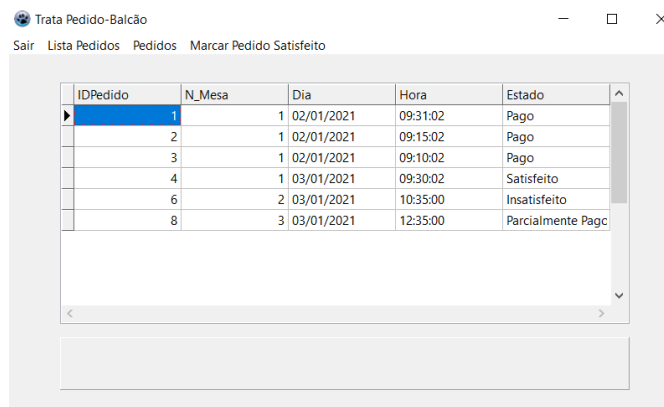
```

Posteriormente, será apresentado o form “Trata Pedido-Balcão”. Este form inclui diversas opções de escolha tais como: “**Sair**”, que quando executado a aplicação termina imediatamente; “**Lista Pedidos**”, que apresenta todos os pedidos que foram efetuados naquele dado bar; “**Pedidos**” que se subdivide em “Pedidos Satisfeitos”, “Pedidos Insatisfeitos”, “Pedidos Parcialmente Pagos” e “Pedidos Pagos”; “**Marcar Pedido Satisfeito**” que quando executado irá abrir um novo form que terá como objetivo principal apresentar um dado pedido(o id do pedido terá que ser introduzido num primeiro instante) e marcá-lo como satisfeito.

É importante referir que a função GetNumBar (tal como referido anteriormente) é de extrema importância para este form, pois a partir desta todos os dados que serão apresentados depois, conforme as opções que o utilizador escolha, serão única e

exclusivamente desse bar e não do outro. No excerto de código visualizado em baixo, pode se verificar que se caso a função `GetNumBar` não exista a aplicação é terminada, posteriormente guarda-se o valor da função numa variável chamada: “num_Bar” (a função devolve o valor do bar inserido pelo utilizador com a variável “numBar”). Depois utiliza-se a função `setnumBar`(que está definida no form do “Marcar Pedido-Satisfeito”) para atribuir um valor à variável “numBar”, ou seja, tudo o que esta função faz é igualar a variável numBar com a variável anterior, num_Bar, e posteriormente através da instrução SQL faz-se `IntToStr(numBar)` para que o número passe a string e possa ser incluído na clausula Where. A instrução SQL seleciona o ID do pedido, Número de Mesa, Dia, Hora e Estado do Pedido. Na clausula Where para se obter o estado do pedido correto, iguala-se o código de estado do pedido que se encontra no **Pedido** com o código de estado que se encontra no **EstadoPedido** e finalmente utiliza-se a função acima mencionada para converter o valor do bar para string.

Por exemplo, num primeiro instante se o utilizador tivesse inserido o dado “1” para o número do bar, a tabela Grid iria apresentar todos os pedidos que dizem respeito ao bar 1(acontece o mesmo, caso o utilizador escolha a opção: “Lista Pedidos”):



The screenshot shows a window titled "Trata Pedido-Balcão" with a menu bar containing "Sair", "Lista Pedidos", "Pedidos", and "Marcar Pedido Satisfeito". Below the menu is a table with the following data:

IDPedido	N_Mesa	Dia	Hora	Estado
1	1	02/01/2021	09:31:02	Pago
2	1	02/01/2021	09:15:02	Pago
3	1	02/01/2021	09:10:02	Pago
4	1	03/01/2021	09:30:02	Satisfeito
6	2	03/01/2021	10:35:00	Insatisfeito
8	3	03/01/2021	12:35:00	Parcialmente Pago

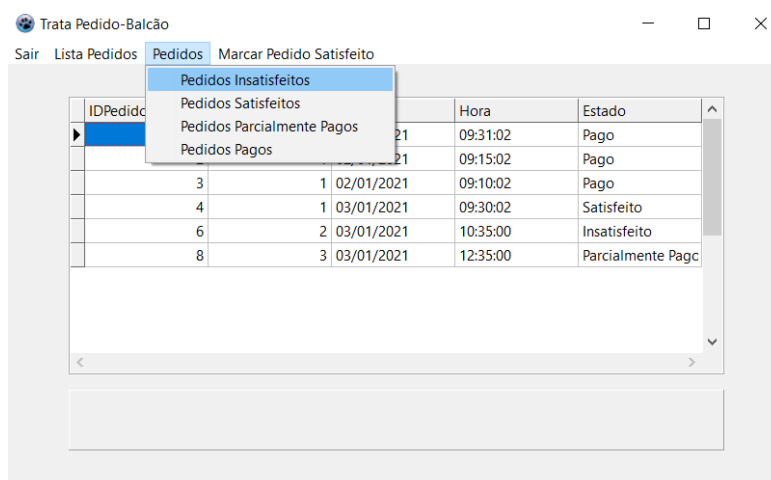
Figura 24 - Página inicial "Trata Pedido-Balcão"

```

if not GetNumBar(numBar) then
    Application.Terminate;
num_Bar:=numBar;
setnumBar(num_Bar);
with SQLQuery1 do
begin
    DataBase := MSSQLConnection1;
    Transaction := MSSQLConnection1.Transaction;
    SQL.Clear;
    SQL.Add('Select P.IDPedido, P.N_Mesa, Dia, Hora,
E.Estado');
    SQL.Add('From Pedido P, Estado_Pedido E');
    SQL.Add('Where P.Cod_EstadoPedido=E.Cod_EstadoPedido AND
P.IDLocal='+IntToStr(numBar));
end;
with SQLQuery1 do
begin
    if Active then
        Close;
    Open;
end;

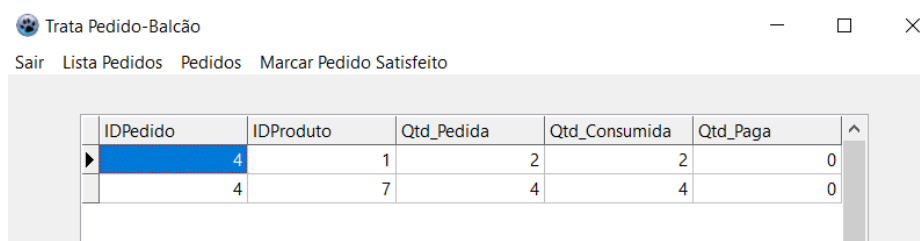
```

Em **Pedidos**, na opção “Pedidos Satisfeitos” irá ser apresentado uma tabela Grid com todos os pedidos que tenham o estado de “Satisfeito”, na opção “Pedidos Insatisfeitos” irão ser apresentados os pedidos com o estado de “Insatisfeito” e o mesmo raciocínio aplica-se aos “Pedidos Parcialmente Pagos” e “Pedidos Pagos”, no qual vão ser apresentados pedidos com o estado de “Parcialmente Pago” e “Pago” respetivamente. No que diz respeito a “Pedidos Parcialmente Pagos”, considera-se todos os pedidos que tenham uma quantidade paga inferior à quantidade consumida, ou seja, os clientes pagaram apenas uma parte do seu consumo. No que diz respeito a “Pedidos Pagos”, considera-se todos os pedidos que tenham uma quantidade paga igual à quantidade consumida, ou seja, os clientes pagaram tudo o que consumiram. No que diz respeito a “Pedidos Satisfeitos”, considera-se que a quantidade consumida seja igual à quantidade pedida. No que diz respeito a “Pedidos Insatisfeitos”, considera-se que, naquele dado momento, os clientes não têm o pedido, ou seja, os bens consumíveis não foram servidos/consumidos. Um exemplo seria escolher a subopção: “Pedidos Insatisfeitos” (a mostrar em baixo):



IDPedido	IDProduto	Qtd_Pedida	Qtd_Consumida	Qtd_Paga	Hora	Estado
21	1	2	2	0	09:31:02	Pago
21	7	4	4	0	09:15:02	Pago
3	1	2	2	0	09:10:02	Pago
4	1	2	2	0	09:30:02	Satisfeito
6	2	2	2	0	10:35:00	Insatisfeito
8	3	3	3	0	12:35:00	Parcialmente Pago

Figura 25 - Ver estados pedidos



IDPedido	IDProduto	Qtd_Pedida	Qtd_Consumida	Qtd_Paga
4	1	2	2	0
4	7	4	4	0

Figura 26 - Mostrar detalhes do pedido

O seguinte excerto de código mostra como foi apresentado na tabela Grid, caso o utilizador escolha a subopção “Pedidos Insatisfeitos”. Basicamente após estabelecida a conexão com a base de dados e inicializada a transação, através de uma instrução SQL, será selecionada o ID do pedido, dia, hora e estado do pedido. Na clausula Where o ID

Local fica com o valor da variável “num_Bar” (que antes é convertida para String), iguala-se o Código de estado do pedido do **Pedido** com o código do estado do pedido do **EstadoPedido** e o código do estado de pedido do **EstadoPedido** tem que ter como valor 1, pois este valor refere-se ao estado “Insatisfeito”.

```
procedure TForm1.MenuItem2Click(Sender: TObject);
begin
    Panel1.Caption:='';
    with SQLQuery1 do
        begin
            DataBase := MSSQLConnection1;
            Transaction := MSSQLConnection1.Transaction;
            SQL.Clear;
            SQL.Add('Select P.IDPedido, P.Dia, P.Hora, E.Estado');
            SQL.Add('From Pedido P, Estado_Pedido E');
            SQL.Add('Where P.IDLocal='+IntToStr(num_Bar)+ 'AND
P.Cod_EstadoPedido=E.Cod_EstadoPedido AND E.Cod_EstadoPedido=1' );

            end;
        with SQLQuery1 do
            begin
                if Active then
                    Close;
                Open;
            end;
        end;
end;
```

Por último, caso o utilizador escolha a opção: “Marcar Pedido Satisfeito” irá ser redirecionado para um outro form chamado “Marcar Pedido Satisfeito”. O propósito deste form consiste em introduzir inicialmente um ID Produto, caso este exista, será apresentado na tabela Grid deste form o ID do produto, a quantidade pedida e a quantidade consumida. Caso ele não exista irá ser apresentado uma mensagem de erro no panel. Existe também uma tabela onde o utilizador pode gerir o pedido, ou seja, conforme uma quantidade pedida deve inserir uma quantidade consumida. No exemplo abaixo se o utilizador inserir o valor 4 na coluna “Qtd Consumida” da segunda tabela e premir o botão marcar pedido, o pedido irá mudar de estado de insatisfeito para satisfeito. Resumidamente, desta forma o utilizador pode “converter” um pedido com o estado de insatisfeito para o estado satisfeito.

Antes de “Marcar Pedido”:

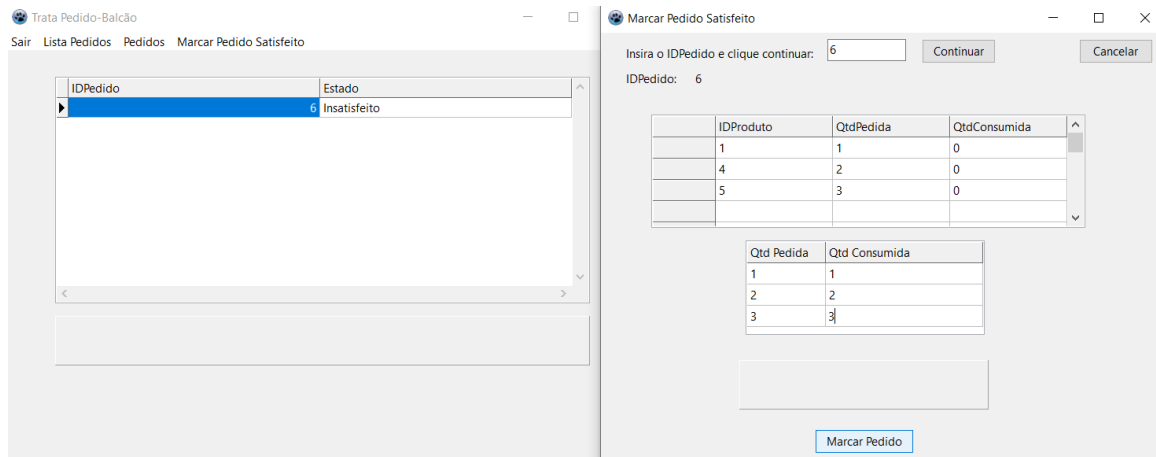


Figura 27 - Marcar Pedido como satisfeito

Depois de “Marcar Pedido”:



Figura 28 - Pós marcar Pedido como satisfeito

É também importante referir que existe uma dinâmica interessante após a escolha da opção “Lista Pedidos”. Quando o utilizador escolhe esta opção poderá verificar todos os detalhes acerca de um Pedido se premir um ID de Pedido, ou seja, se o utilizador clicar no pedido 3, por exemplo, será apresentado no Grid todas as informações acerca desse pedido (no que diz respeito a quantidades pagas, consumidas, pedidas e que produtos se encontram nesse pedido). Verifica-se a seguinte dinâmica nas duas imagens em baixo:

O utilizador seleciona o ID 3, neste caso:

Trata Pedido-Balcão

Sair Lista Pedidos Pedidos Marcar Pedido Satisfeito

IDPedido	N_Mesa	Dia	Hora	Estado
1	1	02/01/2021	09:31:02	Pago
2	1	02/01/2021	09:15:02	Pago
3	1	02/01/2021	09:10:02	Pago
4	1	03/01/2021	09:30:02	Satisfeito
6	2	03/01/2021	10:35:00	Satisfeito
8	3	03/01/2021	12:35:00	Parcialmente Pago

Figura 29 - Selecionar Pedido

A tabela Grid irá apresentar todas as informações acerca do pedido 3:

Trata Pedido-Balcão

Sair Lista Pedidos Pedidos Marcar Pedido Satisfeito

IDPedido	IDProduto	Qtd_Pedida	Qtd_Consumida	Qtd_Paga
3	5	1	1	1

Figura 30 - Informações do pedido

4.3.2.3 Trata Armazém

A aplicação armazém foi feita com a intuição de dar suporte ao armazém através de diversos comandos, inicialmente apresentada com a seguinte estrutura:

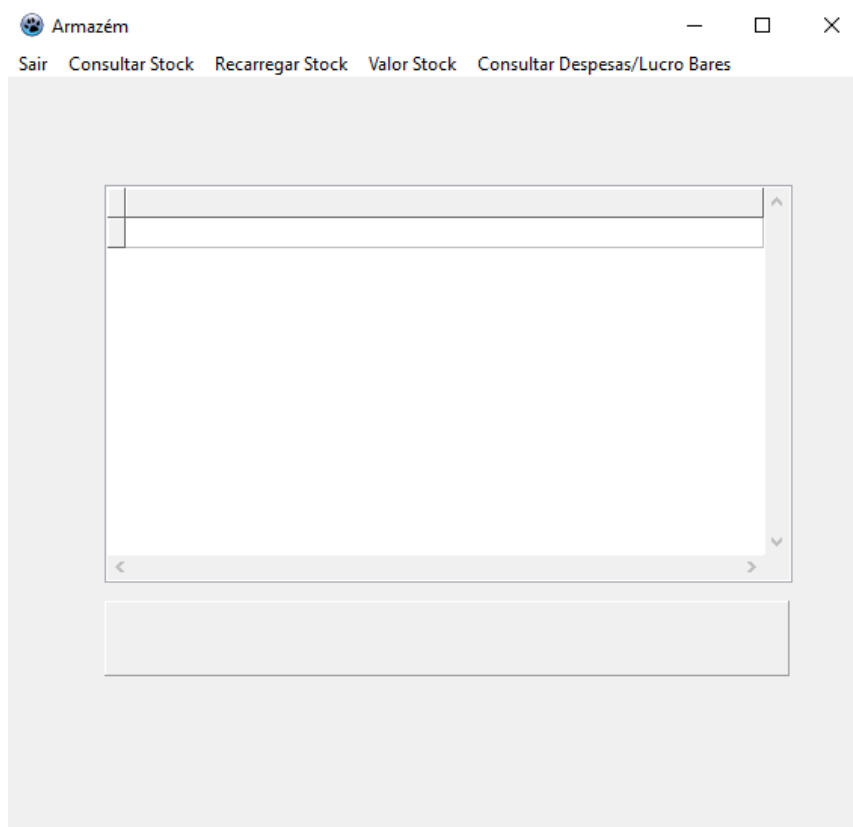


Figura 31 - Página inicial Armazém

Em primeiro lugar, o botão “Sair”, consiste exclusivamente para sair do programa.

Em segundo lugar o programa apresenta um botão que através da introdução do número do local que desejamos, conseguiremos verificar o stock do mesmo. Implementada no código abaixo, quando inserido o número do local, irá disponibilizar o ID do produto, a quantidade que há em stock e o código unidade (produtos guardados como agregados, em que têm um código de medida para transformar em unidades disponíveis para bar).

```
begin
  Panel1.Caption:= '';
  Form4.showmodal;
  StringLocal:=Form4.Edit1.Text;
  if(StringLocal<>'') then
  begin
    Local:=StrToInt(StringLocal);
    with SQLQuery1 do
    begin
      DataBase := MSSQLConnection1;
      Transaction := MSSQLConnection1.Transaction;
      SQL.Clear;
      SQL.Add('Select IDProduto, Qtd, Codigo_Unid');
      SQL.Add('From Stock ');
      SQL.ADD('Where IDLocal='+IntToStr(Local));
    end;
    with SQLQuery1 do
    begin
      if Active then
        Close;
      Open;
    end;
  end;
  Form4.Edit1.Clear;
end;
```

Armazém

Sair Consultar Stock Recarregar Stock Valor Stock Consultar Despesas/Lucro Bares

IDProduto	Qtd	Codigo_Unid
1	270	9
2	140	9
3	50	9
4	50	9
5	70	9
6	80	9
7	45	9
8	50	9
9	50	9
10	40	9
11	40	9

Figura 32 - Consultar Stock Bar1

O Botão seguinte permite ao utilizador recarregar o stock de cada local, neste caso temos os locais 1, 2 e 3, sendo o 1 e 2 correspondentes aos bares e o 3 ao armazém. Relativamente à recarga do armazém, está representado abaixo o seguinte código que o faz. Após o primeiro excerto de código apresentado está a atualização do stock na base de dados.

```
if(Local=3) then
begin
    qtdColocada:=20;
    Panell.Caption:='';
    with SQLQuery1 do
        begin
            DataBase := MSSQLConnection1;
            Transaction := MSSQLConnection1.Transaction;
            SQL.Clear;
            SQL.Add('Select Qtd');
            SQL.Add('From Stock ');
            SQL.Add('Where IDLocal = 3 AND
            IDProduto='+FloatToStr(i));
        end;
    end;
```

```
if(Local=3) then
begin
    qtdColocada:=20;
    Panell.Caption:='';
    with SQLQuery1 do
        begin
            DataBase := MSSQLConnection1;
            Transaction := MSSQLConnection1.Transaction;
            SQL.Clear;
            SQL.Add('Select Qtd');
            SQL.Add('From Stock ');
            SQL.Add('Where IDLocal = 3 AND
            IDProduto='+FloatToStr(i));
        end;
    end;
```

O botão “Valor Stock” consiste em calcular o valor exato do valor total de cada local, inicialmente iremos selecionar o preço de compra de um determinado produto

```
begin
    DataBase := MSSQLConnection1;
    Transaction := MSSQLConnection1.Transaction;
    SQL.Clear;
    SQL.Add('Select Preco_Compra');
    SQL.Add('From Produto');
    SQL.Add('Where IDProduto='+FloatToStr(i));
end;
```

Após selecionarmos o preço de compra dos produtos iremos verificar o preço de compra (preço que a empresa compra aos fornecedores):


```
preco_custo:=SQLQuery1.Fields[0].AsInteger;
```

Por fim iremos calcular o valor total:

```
valorTotal:=valorTotal+(preco_custo)*qtd;
```

Logo após todas as instruções acima um exemplo do output que aparece é o seguinte:

Valor do Local 1 (Preço custo): 220 €
Valor do Local 2 (Preço custo): 335 €
Valor do Local 3 (Preço custo): 480 €

Figura 33- Consultar Valor Stock dos locais

O último botão é para consultar as despesas e o lucro dos bares em questão, sendo essa consulta feita através do dia pretendido. No display do programa irá aparecer a lista de dias disponíveis para a consulta das despesas e do lucro.

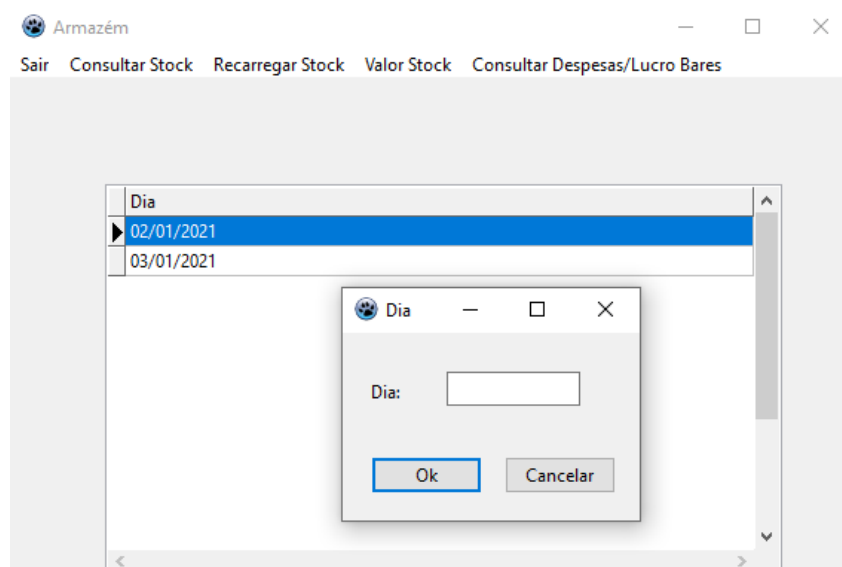


Figura 34 - Consultar Despesas/Lucro dos bares

Após o sistema receber o dia, através dos seguintes códigos e da conta final irá aparecer numa parte do display todos os dados pretendidos.

```
begin
    DataBase := MSSQLConnection1;
    Transaction := MSSQLConnection1.Transaction;
    SQL.Clear;
    SQL.Add('Select IDProduto,Qtd_Paga, Qtd_Pedida');
    SQL.Add('From Pedido P, Linha_Pedido L');
    SQL.Add('Where P.IDLocal = '+ IntToStr(o)+'AND
P.Cod_EstadoPedido=4 AND P.IDPedido=L.IDPedido AND Dia =
'''+data+'''' AND IDProduto='+IntToStr(i));
end;
```

```
begin
    DataBase := MSSQLConnection1;
    Transaction := MSSQLConnection1.Transaction;
    SQL.Clear;
    SQL.Add('Select Preco_Compra, Preco_Venda');
    SQL.Add('From Produto');
    SQL.Add('Where IDProduto= '+ FloatToStr(produto));
end;
```

```
begin
    DataBase := MSSQLConnection1;
    Transaction := MSSQLConnection1.Transaction;
    SQL.Clear;
    SQL.Add('Select Preco_Compra, Preco_Venda');
    SQL.Add('From Produto');
    SQL.Add('Where IDProduto= '+ FloatToStr(produto));
end;
```

O output irá ser:

Bares: - Despesas: 1,1€ - Recebido: 3,8€ - Lucro: 2,7€

Figura 35 - Despesas, Recebido e lucro dos bares

5. Conclusões

Com a conclusão do trabalho o grupo conseguiu resolver as tarefas a fazer, à exceção de algumas restrições na inserção de valores devido à falta de tempo, mesmo trabalhando arduamente quase todos os dias, a estudar tanto para esta cadeira tanto para outras.

Epílogo

Em geral, o professor João Muranho colocou a avaliação correta na frequência/caderno autónomo que os alunos fizeram durante o semestre, não concordando apenas na avaliação do trabalho que neste caso está avaliado com um máximo de 5 valores. Esse valor está muito abaixo em relação ao trabalho dispensado pelos alunos, ou seja, o trabalho vale pouco relativamente à complexidade do trabalho. Um dos pontos positivos foi a adição do caderno autónomo, em que ajuda muito os alunos a manterem a matéria em dia e “obriga-nos” a ter de estudar para a unidade curricular.

Um dos pontos mais importantes era sermos avaliados através de duas frequências e não apenas uma, pois se a matéria for dividida por duas frequências os alunos tiram mais proveito e conseguem tirar melhores notas.

Referências Bibliográficas

Lazarus(ADI) (9/12/2020) [https://pt.wikipedia.org/wiki/Lazarus_\(ADI\)](https://pt.wikipedia.org/wiki/Lazarus_(ADI))

Modelo cliente-servidor (10/12/2020)

https://pt.wikipedia.org/wiki/Modelo_cliente%E2%80%93servidor#Tipos_ou_Modelos_de_Client/Server

Rob Lerner, *The Basics of a Microsoft SQL Server Architecture* (12/12/2020)

<https://www.inap.com/blog/microsoft-sql-server-architecture/>

O modelo cliente/servidor (12/12/2020) https://web.fe.up.pt/~goii2000/M1/2_1-clienteservidor.htm

Transmission Control Protocol (15/12/2020)

https://pt.wikipedia.org/wiki/Transmission_Control_Protocol#Utiliza%C3%A7%C3%A3o_do_IP_para_entrega_de_dados

Modelagem de dados (4/01/2020) https://pt.wikipedia.org/wiki/Modelagem_de_dados

J. Muranho, H. Proença, P. Prata, R. Cardoso; *Notas de apoio às aulas*: Versão 4.03

Apêndices

1. Script Criar Base de dados

```
USE master
---Create the DataBase

IF ( EXISTS( SELECT * FROM [dbo].[sysdatabases] Where name =
'Zurrapa') )
Begin
    DROP DATABASE Zurrapa
end

IF (NOT EXISTS( SELECT * FROM [dbo].[sysdatabases] Where name =
'Zurrapa') )
Begin

    CREATE DATABASE Zurrapa
    ON
        ( NAME = 'Projecto_dat',

        FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\Projdat.mdf',

        SIZE = 10,
        MAXSIZE = 50,
        FILEGROWTH = 5 )
    LOG ON
        ( NAME = 'Projecto_log',

        FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\Projlog.ldf',

        SIZE = 5MB,
        MAXSIZE = 25MB,
        FILEGROWTH = 5MB )
end
```

2. Script Criar Tabelas

```
USE Zurrapa
--

-- Criar as tabelas

if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Empregado]')) )
begin
    CREATE TABLE Empregado (
        IDEmpregado int NOT NULL
        CHECK (IDEmpregado >= 1), -- constraint type:
check
        Nome nvarchar(30) NOT NULL,
        Funcao nvarchar(30) NOT NULL,
        Endereco nvarchar(30) NOT NULL
        DEFAULT 'Covilha', -- Default definition

        CONSTRAINT PK_IDEmpregado PRIMARY KEY (IDEmpregado), -- constraint type:
primary key
    );
end

--

.....
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Local]')) )
begin
    CREATE TABLE Local (
        IDLocal int NOT NULL
        CHECK (IDLocal >= 1),
        Designacao nvarchar (30) NOT NULL,
        Tipo nvarchar (30) NOT NULL,

        IDEmpregado int NOT NULL
        CHECK (IDEmpregado >= 1),

        CONSTRAINT PK_IDLocal PRIMARY KEY (IDLocal), -- constraint type:
primary key
    );
end
```

```

if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Deslocacao]')) )
begin
    CREATE TABLE Deslocacao (
        De int NOT NULL,
        Para int NOT NULL,
        Tempo int NOT NULL,

        CONSTRAINT FK_De FOREIGN KEY (De)
            REFERENCES Local(IDLocal)
            ON UPDATE CASCADE,

        CONSTRAINT FK_Para FOREIGN KEY (Para)
            REFERENCES Local(IDLocal)
            ON UPDATE NO ACTION,

        CONSTRAINT PK_Deslocacao PRIMARY KEY (De, Para), --
constraint type: primary key
    );
end
--
.....
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[EscalaTrabalho]')) )
begin
    CREATE TABLE EscalaTrabalho (

        IDEmpregado int NOT NULL
            CHECK (IDEmpregado >= 1),
        IDLocal int NOT NULL
            CHECK (IDLocal >= 1),
        Hora nvarchar (50) NOT NULL ,
        Dia nvarchar (30) NOT NULL ,           -- NULL Column

        CONSTRAINT FK_IDEmpregado FOREIGN KEY (IDEmpregado)
            REFERENCES Empregado(IDEmpregado)
            ON UPDATE CASCADE,

        CONSTRAINT FK_IDLocal FOREIGN KEY (IDLocal)
            REFERENCES Local(IDLocal)
            ON UPDATE CASCADE

    );
end

```



```
--
.....
...
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Estado_Mesa]')) )
begin
    CREATE TABLE Estado_Mesa (

        Cod_EstadoMesa int NOT NULL,
        Estado nvarchar (30) NOT NULL ,

        --CONSTRAINT PK_Mesa PRIMARY KEY (IDLocal,NMesa), --
constraint type: primary key
        CONSTRAINT PK_Cod_EstadoMesa PRIMARY KEY (Cod_EstadoMesa),

    );
end
--
.....
...
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Mesa]')) )
begin
    CREATE TABLE Mesa (

        IDLocal int NOT NULL,
        IDEmpregado int,
        N_Mesa int NOT NULL ,           -- NULL Column
        Cod_EstadoMesa int NOT NULL,

        CONSTRAINT FK_IDEmpregado1 FOREIGN KEY (IDEmpregado)
            REFERENCES Empregado (IDEmpregado)
            ON UPDATE CASCADE,

        CONSTRAINT FK_IDLocal11 FOREIGN KEY (IDLocal)
            REFERENCES Local (IDLocal)
            ON UPDATE CASCADE,

        CONSTRAINT FK_Cod_EstadoMesa FOREIGN KEY (Cod_EstadoMesa)
            REFERENCES Estado_Mesa (Cod_EstadoMesa)
            ON UPDATE CASCADE,

        --CONSTRAINT PK_Mesa PRIMARY KEY (IDLocal,NMesa), --
constraint type: primary key
        CONSTRAINT PK_Mesa PRIMARY KEY (N_Mesa, IDLocal),

    );
end
```

```
--
.....
.....
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Estado_Pedido]')) )
begin
    CREATE TABLE Estado_Pedido (

        Cod_EstadoPedido int NOT NULL,
        Estado nvarchar (30) NOT NULL ,

        --CONSTRAINT PK_Mesa PRIMARY KEY (IDLocal,NMesa), --
        constraint type: primary key
        CONSTRAINT PK_Cod_EstadoMesa1 PRIMARY KEY (Cod_EstadoPedido),

    );
end
--
.....
.....
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Pedido]')) )
begin
    CREATE TABLE Pedido (

        IDPedido int NOT NULL,
        Cod_EstadoPedido int NOT NULL,
        IDLocal int NOT NULL,
        N_Mesa int NOT NULL ,
        IDEmpregado int NOT NULL ,
        Dia nvarchar (30) NOT NULL ,
        Hora nvarchar (30) NOT NULL ,

        CONSTRAINT FK_IDEmpregado2 FOREIGN KEY (IDEmpregado)
            REFERENCES Empregado (IDEmpregado)
            ON UPDATE CASCADE,

        CONSTRAINT FK_NMesa FOREIGN KEY (N_Mesa, IDLocal)
            REFERENCES Mesa (N_Mesa, IDLocal)
            ON UPDATE NO ACTION,

        CONSTRAINT FK_Cod_EstadoPedido FOREIGN KEY (Cod_EstadoPedido)
            REFERENCES Estado_Pedido (Cod_EstadoPedido)
            ON UPDATE NO ACTION,

        CONSTRAINT PK_IDPedido PRIMARY KEY (IDPedido),

    );
end
```

```
--
.....
.....
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Produto]')) )
begin
    CREATE TABLE Produto (

        IDProduto int NOT NULL,
        Designacao nvarchar (30) NOT NULL,
        Preco_Compra decimal(5,2) NOT NULL,
        Preco_Venda decimal(5,2) NOT NULL,

        CONSTRAINT ID_Produto PRIMARY KEY (IDProduto),

    );
end

--
.....
.....
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Linha_Pedido]')) )
begin
    CREATE TABLE Linha_Pedido(

        IDPedido int NOT NULL,
        IDProduto int NOT NULL,
        Qtd_Pedida int,
        Qtd_Consumida int,
        Qtd_Paga int,

        CONSTRAINT FK_IDPedido2 FOREIGN KEY (IDPedido)
            REFERENCES Pedido(IDPedido)
            ON UPDATE CASCADE,

        CONSTRAINT FK_IDProduto FOREIGN KEY (IDProduto)
            REFERENCES Produto(IDProduto)
            ON UPDATE CASCADE,

        CONSTRAINT PK_Linha_Pedido PRIMARY KEY (IDPedido,IDProduto),

    );
end

--
.....
.....
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Unidade_Medida]')) )
begin
    CREATE TABLE Unidade_Medida (

        Codigo_Unid int NOT NULL,
        Designacao nvarchar (30) NOT NULL,
        Unidade int NOT NULL,

        CONSTRAINT PK_Codigo_Unid PRIMARY KEY (Codigo_Unid),

    );
end
```

```
if not exists (select * from dbo.sysobjects
               where id = object_id(N'[dbo].[Stock]')) )
begin
    CREATE TABLE Stock (

        IDLocal int NOT NULL,
        IDProduto int NOT NULL,
        Qtd int,
        Codigo_Unid int NOT NULL,

        CONSTRAINT FK_IDLocal2 FOREIGN KEY (IDLocal)
            REFERENCES Local(IDLocal)
            ON UPDATE CASCADE,

        CONSTRAINT FK_IDProduto1 FOREIGN KEY (IDProduto)
            REFERENCES Produto(IDProduto)
            ON UPDATE CASCADE,

        CONSTRAINT FK_Codigo_Unid FOREIGN KEY (Codigo_Unid)
            REFERENCES Unidade_Medida(Codigo_Unid)
            ON UPDATE CASCADE,

    );
end
```

3.Script Dados Tabelas

```
USE Zurrapa
----Empregado----
INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (1, 'Bruno Monteiro', 'Balcao', 'Covilhã');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (2, 'Manuel Magalhães', 'Mesa', 'Covilhã');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (3, 'Miguel Pereira', 'Armazem', 'Covilhã');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (4, 'João Dias', 'Balcao', 'Fundão');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (5, 'Afonso Rodrigues', 'Mesa', 'Fundão');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (6, 'Ângela Soares', 'Mesa', 'Fundão');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (7, 'António Silva', 'Mesa', 'Fundão');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (8, 'Beatriz Pinto', 'Mesa', 'Fundão');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (9, 'Joaquim Ferreira', 'Mesa', 'Fundão');

INSERT INTO Empregado (IDEmpregado, Nome, Funcao, Endereco)
VALUES (10, 'Filipe Carvalho', 'Mesa', 'Fundão');
-----

----Local----
INSERT INTO Local (IDLocal, Designacao, IDEmpregado, Tipo)
VALUES (1, 'Bar 6 fase', 2, 'Bar');

INSERT INTO Local (IDLocal, Designacao, IDEmpregado, Tipo)
VALUES (2, 'Bar 4 fase', 1, 'Bar');

INSERT INTO Local (IDLocal, Designacao, IDEmpregado, Tipo)
VALUES (3, 'Armazém Central', 3, 'Armazem');
-----

----Deslocacao----
INSERT INTO Deslocacao (De, Para, Tempo)
VALUES (3, 1, 35);

INSERT INTO Deslocacao (De, Para, Tempo)
VALUES (3, 2, 25);

INSERT INTO Deslocacao (De, Para, Tempo)
VALUES (1, 3, 36);

INSERT INTO Deslocacao (De, Para, Tempo)
VALUES (2, 3, 24);

INSERT INTO Deslocacao (De, Para, Tempo)
VALUES (1, 2, 10);

INSERT INTO Deslocacao (De, Para, Tempo)
VALUES (2, 1, 12);
-----
```

```

-----EscalaTrabalho-----
INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (3,3, '9h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,1, '9h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,2, '9h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,5, '10h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,6, '12h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,7, '13h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (2,4, '9h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (2,8, '9h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (2,9, '9h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (2,10, '10h', '03/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (3,3, '9h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,1, '9h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,2, '9h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,5, '10h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,6, '12h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (1,7, '13h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (2,4, '9h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (2,8, '9h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (2,9, '9h', '05/01/2021');

INSERT INTO EscalaTrabalho (IDLocal, IDEmpregado, Hora, Dia)
VALUES (2,10, '10h', '05/01/2021');
-----*****-----

```

```

----Estado_Mesa----
Insert INTO Estado_Mesa(Cod_EstadoMesa, Estado)
values(1, 'Limpa')

Insert INTO Estado_Mesa(Cod_EstadoMesa, Estado)
values(2, 'Ocupada')

Insert INTO Estado_Mesa(Cod_EstadoMesa, Estado)
values(3, 'Suja')
----***----

----Mesa----
INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(1,2,1, 2)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(1,5,2, 2)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(1,5,3, 2)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(1,NULL,4, 1)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(1,NULL,5, 1)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(2,8,1, 3)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(2,9,2, 2)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(2,10,3, 2)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(2,8,4, 2)

INSERT INTO Mesa(IDLocal, IDEmpregado, N_Mesa, Cod_EstadoMesa)
Values(2,NULL,5, 1)
----***----

----Estado_Pedido----
INSERT INTO Estado_Pedido(Cod_EstadoPedido, Estado)
Values(1, 'Insatisfeito')

INSERT INTO Estado_Pedido(Cod_EstadoPedido, Estado)
Values(2, 'Satisfeito')

INSERT INTO Estado_Pedido(Cod_EstadoPedido, Estado)
Values(3, 'Parcialmente Pago')

INSERT INTO Estado_Pedido(Cod_EstadoPedido, Estado)
Values(4, 'Pago')
----***----

```

```

-----Pedidos-----
INSERT INTO Pedido(IDPedido, Cod_EstadoPedido, N_Mesa,
IDLocal, IDEmpregado, Dia, Hora)
Values(1,4,1,1,2, '02/01/2021', '09:31:02')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido, N_Mesa,
IDLocal, IDEmpregado, Dia, Hora)
Values(2,4,1,1,2, '02/01/2021', '09:15:02')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido, N_Mesa,
IDLocal, IDEmpregado, Dia, Hora)
Values(3,4,1,1,2, '02/01/2021', '09:10:02')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido, N_Mesa,
IDLocal, IDEmpregado, Dia, Hora)
Values(4,2,1,1,2, '03/01/2021', '09:30:02')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido,
N_Mesa, IDLocal, IDEmpregado, Dia, Hora)
Values(5,4,1,2,8, '03/01/2021', '9:35:17')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido, N_Mesa,
IDLocal, IDEmpregado, Dia, Hora)
Values(6,1,2,1,5, '03/01/2021', '10:35:00')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido,
N_Mesa, IDLocal, IDEmpregado, Dia, Hora)
Values(7,3,2,2,9, '03/01/2021', '10:35:02')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido,
N_Mesa, IDLocal, IDEmpregado, Dia, Hora)
Values(8,3,3,1,5, '03/01/2021', '12:35:00')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido, N_Mesa,
IDLocal, IDEmpregado, Dia, Hora)
Values(9,2,3,2,10, '03/01/2021', '12:32:15')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido, N_Mesa,
IDLocal, IDEmpregado, Dia, Hora)
Values(10,1,4,2,8, '03/01/2021', '12:37:02')

INSERT INTO Pedido(IDPedido, Cod_EstadoPedido, N_Mesa,
IDLocal, IDEmpregado, Dia, Hora)
Values(11,1,4,2,10, '03/01/2021', '12:37:02')
----***-----

-----Produtos-----
INSERT INTO Produto(IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values(1, 'Café', 0.30, 0.60)

INSERT INTO Produto(IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values(2, 'Cerveja', 0.80, 1.20)

INSERT INTO Produto(IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values(3, 'Água 1.5L', 0.10, 1.50)

INSERT INTO Produto(IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values(4, 'Água 0.5L', 0.05, 0.50)

INSERT INTO Produto(IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values(5, 'Milka Oreo', 0.10, 1.0)

```



```

INSERT INTO Produto (IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values (6, 'Coca-cola', 0.20, 1.10)

INSERT INTO Produto (IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values (7, 'Pão', 0.05, 0.15)

INSERT INTO Produto (IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values (8, 'Croissant', 0.10, 0.70)

INSERT INTO Produto (IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values (9, 'Brownie', 0.10, 0.70)

INSERT INTO Produto (IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values (10, 'Red Bull', 0.70, 3.10)

INSERT INTO Produto (IDProduto, Designacao, Preco_Compra, Preco_Venda)
Values (11, 'Lasanha', 1.2, 4.10)
-----****-----

----Linha Pedidos-----
INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (1, 1, 3, 3, 3)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (2, 4, 2, 2, 2)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (3, 5, 1, 1, 1)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (4, 7, 4, 4, 0)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (4, 1, 2, 2, 0)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (5, 2, 3, 3, 3)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (6, 1, 1, 0, 0)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (6, 5, 3, 0, 0)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (6, 4, 2, 0, 0)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (7, 6, 2, 2, 1)

```

```

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (7, 3, 2, 2, 1)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (8, 9, 3, 3, 2)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (9, 10, 2, 2, 0)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (10, 11, 1, 0, 0)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (10, 1, 3, 0, 0)

INSERT INTO
Linha_Pedido (IDPedido, IDProduto, Qtd_Pedida, Qtd_Consumida, Qtd_Paga)
Values (11, 8, 2, 0, 0)
-----

---Unidade Medida---
INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (1, 'Saco de café', 50)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (2, 'Grade Cerveja', 24)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (3, 'Embalagem Água', 6)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (4, 'Embalagem Chocolates', 10)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (5, 'Embalagem Refrigerante', 12)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (6, 'Saco Pães', 5)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (7, 'Saco Bolos', 6)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (8, 'Embalagem energéticos', 4)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (9, 'Unidade', 1)

INSERT INTO Unidade_Medida (Codigo_Unid, Designacao, Unidade)
Values (10, 'Embalagem Lasanhas', 4)

```

```

----Stock----
INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,1,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,2,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,3,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,4,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,5,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,6,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,7,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,8,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,9,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,10,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(1,11,20,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,1,15,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,2,10,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,3,5,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,4,10,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,5,7,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,6,10,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,7,6,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,8,7,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,9,5,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,10,3,9)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(2,11,2,9)

```

```
INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,1,15,1)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,2,15,2)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,3,15,3)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,4,15,3)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,5,15,4)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,6,15,5)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,7,15,6)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,8,15,7)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,9,15,7)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,10,15,8)

INSERT INTO Stock(IDLocal,IDProduto,Qtd,Codigo_Unid)
Values(3,11,15,10)
```