Ano letivo 2018/2019

Bases de Dados

Hotel Management

Desenvolvido por:

João Dias Nº89236

José Santos Nº88903

Índice

Introdução

O projeto foi criado no âmbito da cadeira de Base de Dados, o tema escolhido para o desenvolvimento deste foi a gestão de um hotel.

O objetivo principal da base de dados criada é a facilitação da gestão de um hotel, permitindo a clientes, empregados e gerente acederem e alterarem informação.

A base de dados guarda a informação sobre o cliente e todos os gastos que este efetuou durante a sua estadia. Tem também todos os quartos do hotel e a quem estes estão associados respetivamente. Podemos aceder também à lista de empregados, aceder à sua informação básica como o horário e ver que empregado esteve encarregue de fazer o quê e quando.

Descrição dos Conceitos

Customer: Contém a informação básica sobre o cliente, pode ser alterada parcialmente por ele mesmo ou totalmente pelo Manager do Hotel, pode criar uma conta nova ou entrar numa já existente. Este é identificado pelo seu *NIF*.

Account Record: Onde toda informação dos gastos do cliente é armazenada, é o account record que, após a inscrição do cliente, fica associado ao mesmo e a todas as atividades e consumismos que ele faça enquanto no hotel. É identificado pelo *account record id*, algo que é gerado quando o cliente cria uma conta.

Booking: Sempre que um cliente requisita um quarto durante um determinado período de tempo é criado um novo *Booking*, este por sua vez cria um *IS_ASSIGNED* que atribui determinado quarto(s) a um *Booking*. O *Booking* é identificado pelo *BookingNum*.

Hotel: O *Hotel* é a entidade principal do sistema, está associado aos quartos e aos empregados, é identificado pelo *hotel id*.

Room: O *Room* está associado a um *Room Type* ao qual vai buscar o tipo de quarto, este pode ter uma cama de solteiro, duas camas de solteiro, uma cama de casal, etc. O *Room* é identificado pelo *Room* nº e o *Room Type* é identificado pelo *Room Type*.

Employee: O *Employee* divide-se em 4 tipos, O *Cleaner*, *Receptionist*, *Chef* e *Bartender*. A entidade *Employee* é identificada pelo *Staff id* e está associado a um *Schedule*. Este *schedule* está dividido em três tipos, *morning shift, afternoon shift* e *evening shift*.

O *Bartender*, identificado pelo *Bartender id*, está encarregue de fazer as bebidas para os clientes e a compra das mesmas fica registada pelo *account record*. O *Chef*, identificado pelo *Chef id*, está encarregue de fazer as refeições para os clientes e tal como as bebidas, estas ficam associadas ao *account record*. O *Receptionist*, identificado pelo *receptionist id*, está encarregue de realizar o *booking* dos clientes, este porteriormente fica associado a todos os *bookings* que realizou. O *Cleaner*, identificado pelo *cleaner id,* está associado ao *Room Service*, este associa todos os *Cleaners* com os respetivos quartos que estes estão encarregues de limpar.

Esquema Relacional

O nosso esquema relacional sofreu algumas alterações ao longo do desenvolvimento da base de dados visto que nem tudo o que tínhamos em mente inicialmente poderia ser trabalhado da melhor maneira com a primeira versão do Esquema Relacional.

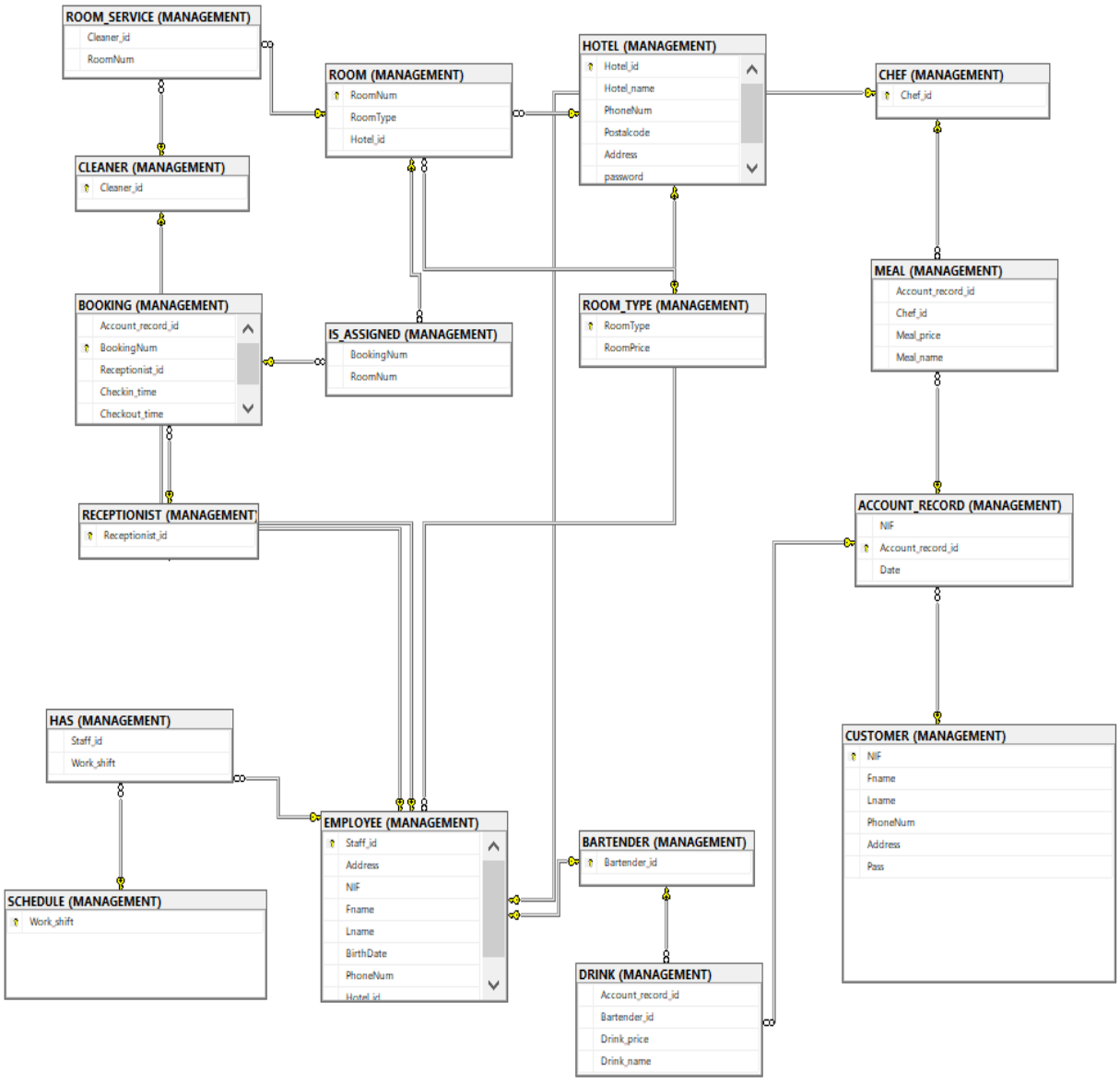
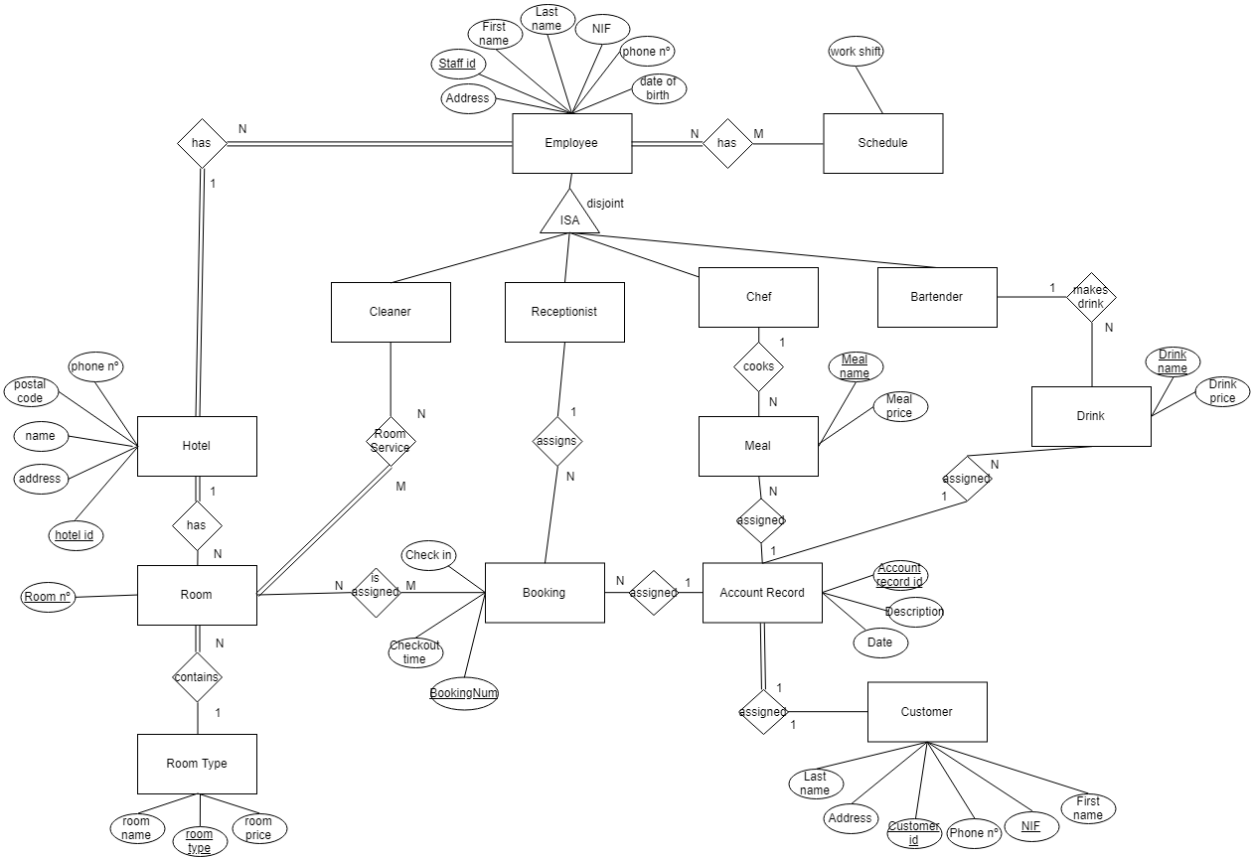


Diagrama Entidade-Relação

Tal como o esquema relacional, o diagrama entidade-relação também sofre algumas alterações ao longo da construção da base de dados, essas alterações foram feitas por razões de lógica e simplicidade de funcionalidades.



Formas Normais

Após feito o modelo, conseguimos observar que em todas as relações existe um ID que permite chegar aos restantes atributos de cada relação. Estes Ids são chaves de entidades e relações. Nas relações que contêm mais que uma dependência funcional, podemos observar que os atributos também permitem chegar aos referidos Ids. Com isto, todas as relações se encontram de acordo com a forma normal *BCNF*. Sabendo que a 3ª forma normal é um super set da *BCNF*, estas também se encontram na 3ª forma normal.

DDL (Data Definition Language)

A utilização de DDL foi essencial para os passos inicias de criação de tabelas (CREATE TABLE) e alteração das mesmas (ALTER TABLE).

```sql
/*Completed*/
CREATE TABLE MANAGEMENT.CUSTOMER(
    NIF             CHAR(9) NOT NULL,
    Fname           VARCHAR(20) NOT NULL,
    Lname           VARCHAR(20) NOT NULL,
    PhoneNum        INT NOT NULL CHECK(PhoneNum > 0),
    [Address]       VARCHAR(50),
    Pass            VARCHAR(30) NOT NULL CHECK(LEN(Pass) > 7),
    PRIMARY KEY(NIF)
);

/*Completed*/
CREATE TABLE MANAGEMENT.HOTEL(
    Hotel_id    CHAR(10) NOT NULL,
    Hotel_name  VARCHAR(20) NOT NULL,
    PhoneNum    INT NOT NULL CHECK(PhoneNum > 0),
    Postalcode  CHAR(8) NOT NULL,
    [Address]   VARCHAR(30) NOT NULL,
    PRIMARY KEY(Hotel_id)

);
```

Figura 1: Exemplo do uso de DDL

DML(Data Manipulation Language)

Da mesma maneira que DDL nos permitiu a criação base da base de dados, DML deixa-nos popular a mesma e interagir com os dados inseridos e mostrá-los na nossa interface gráfica.

```
INSERT INTO MANAGEMENT.EMPLOYEE VALUES
-- staff_id, address, nif, fname, lname, bday, phoneNum, hotel_id --
(3265485596,NULL,569422565,'Rita','Costa','1977-07-22',928856178,8695647822),--Cleaner --
(4924297067,'Rua Maria Azevinho Costa',569621038,'Joana','Martins','1980-08-02',936544891,8695647822), --Cleaner --
(5378235358,'Rua Alice Rosada Sensas',364412500,'Pedro','Rosaldo','1987-08-23',966648551,8695647822), --Cleaner--
(6829855222,'Rua Maria Olivia Rosa',469531259,'Pedro','Amarante','1990-02-12',936584420,8695647822), --Cleaner--
(2824650926,'Rua Vertigo Santinha',841233265,'João','Silva','1990-03-16',966548832,8695647822), --Cleaner --
(7257432533,'Rua Olivia Vertigo',841233223,'José','Costa','1969-04-09',935465201,8695647822),--Cleaner --
(2393595733,'Rua Andrade Santos',512366540,'José','Silva','1973-09-21',911568987,8695647822),--Cleaner --
(2340978646,'Rua Augusto Dionísio',132002508,'Ricardo','Almeida','1986-01-19',967897256,8695647822),--Cleaner --
(6162059924,'Rua Almirante Dias',123123002,'Tomás','Santos','1990-06-02',913655898,8695647822),--Cleaner --
(8136962830,'Rua Almeida Rosa',789865200,'André','Rosa','1996-11-15',936557981,8695647822),--Cleaner --
(9921060057,NULL,789558903,'Maria','Almeida','1996-08-29',912085610,8695647822),--Cleaner --
(8603913353,'Rua Vertigo Santinha',456565502,'Maria','Dias','1999-09-30',961782028,8695647822),--Cleaner --
(4497828304,'Rua Almeida Rosa',465200965,'Flávia','Figueiredo','2000-12-24',235698120,8695647822),--Cleaner --
(9539421033,'Rua Santigo',865233014,'Bernardo','Faria','1970-06-12',935454412,8695647822),--Cleaner --
(7924268842,'Rua Vagar do Monte',456888879,'João','Abrantes','1976-03-02',969954112,8695647822),--Chef --
(4172196483,'Rua Mirtilo da Esquina',656478809,'Ricardo','Pastor','1993-08-26',935464218,8695647822),--Chef --
(1142298274,'Rua José Emanuel',326998754,'Alexandre','Pintor','1992-10-17',919256362,8695647822),--Chef --
(5343285525,'Rua Direita',456995633,'Jorge','Martins','1993-02-13',968953204,8695647822),--Chef--
(9392233957,'Rua Rosaldo Rosa',112796658,'Xavier','Silva','1990-09-03',917757632,8695647822),--Chef--
(3242349318,'Rua Rita Olívias',369965899,'Francisco','Costa','1985-03-01',933564920,8695647822),--Chef--
(8555075982,'Rua Carolinias',178230555,'Dinis','Correia','1980-04-22',966598222,8695647822),--Chef--
(4732578707,NULL,789880263,'Guilherme','Correia','1986-03-22',936654456,8695647822),--Chef--
(8032496247,NULL,558965478,'Ana','Fernandes','1993-02-22',933654210,8695647822),--Bartender --
(2010761690,'Rua da Universidade',898226600,'Ana','Dias','1976-03-15',964565209,8695647822), --Bartender
```

Figura 2: Exemplo do uso de DML

Stored Procedures

Stored Procedures são um bom meio de abstração e a sua execução pode ser bastante rápida. Como os *Stored Procedures* são desenvolvidos pelo developer da base de dados é menos provável que tenha erros de integridade que SQL code ad hoc.

Em termos de segurança é útil para os logins e outras utilidades na aplicação, visto que, o user apenas utiliza o *Stored Procedure* para aceder à tabela.

Os *Stored Procedures* proporcionaram-nos um grande apoio no desenvolvimento da nossa aplicação pois facilitavam eliminação de tabelas em "cascata". Foram usados também para obtenção de dados com as uniões de várias tabelas

```
CREATE PROCEDURE [MANAGEMENT].[DELETE_CHEF]
    @chefId char(10),
    @outputResult INT OUTPUT
AS
BEGIN
    BEGIN TRANSACTION
        BEGIN TRY
            DELETE FROM MANAGEMENT.MEALS WHERE MANAGEMENT.MEALS.Chef_id = @chefId
            DELETE FROM MANAGEMENT.CHEF WHERE MANAGEMENT.CHEF.Chef_id = @chefId
            DELETE FROM MANAGEMENT.HAS WHERE MANAGEMENT.HAS.Staff_id = @chefId
            COMMIT

            SET @outputResult = 0
        END TRY
        BEGIN CATCH
            ROLLBACK;
            SET @outputResult = 1
        END CATCH
    DELETE FROM MANAGEMENT.EMPLOYEE WHERE MANAGEMENT.EMPLOYEE.Staff_id = @chefId
    RETURN @outputResult
END
```

Figuras 3 : Exemplo do uso de Stored Procedures na eliminação de informação

```
ALTER PROCEDURE MANAGEMENT.GET_GASTOS @NIF char(9)
AS
    DECLARE @auxTable TABLE(Drink_name varchar(20), Drink_price int, Meal_name varchar(20), Meal_price int, RoomNum int, Checkin_time datetime2, Checkout_time datetime2,
    INSERT INTO @auxTable (Drink_name, Drink_price, Meal_name, Meal_price, RoomNum, Checkin_time, Checkout_time, Room_price)
    SELECT D.Drink_name, D.Drink_price, M.Meal_name, M.Meal_price, R.RoomNum, B.Checkin_time, B.Checkout_time, RT.RoomPrice FROM MANAGEMENT.CUSTOMER C
    INNER JOIN MANAGEMENT.ACCOUNT_RECORD AC ON C.NIF = AC.NIF
    INNER JOIN MANAGEMENT.DRINK D ON D.Account_record_id = AC.Account_record_id
    INNER JOIN MANAGEMENT.MEAL M ON M.Account_record_id = AC.Account_record_id
    INNER JOIN MANAGEMENT.BOOKING B ON B.Account_record_id = AC.Account_record_id
    INNER JOIN MANAGEMENT.IS_ASSIGNED A ON A.BookingNum = B.BookingNum
    INNER JOIN MANAGEMENT.ROOM R ON R.RoomNum = A.RoomNum
    INNER JOIN MANAGEMENT.ROOM_TYPE RT ON RT.RoomType = R.RoomType
    WHERE C.NIF = 195175302

    INSERT INTO @auxTable (Total_meal_price) SELECT SUM(Meal_price) FROM @auxTable
    INSERT INTO @auxTable (Total_drink_price) SELECT SUM(Drink_price) FROM @auxTable

    DECLARE @NR_DAYS INT;

    DECLARE @Checkin_time datetime2;
    DECLARE @Checkout_time datetime2;

    SELECT TOP 1 @Checkin_time = Checkin_time, @Checkout_time = Checkout_time FROM @auxTable

    SELECT @NR_DAYS = DATEDIFF(day, @Checkin_time, @Checkout_time)
    PRINT @NR_DAYS
    INSERT INTO @auxTable (Total_room_price) SELECT @NR_DAYS * Room_price FROM @auxTable

    SELECT * FROM @auxTable
GO
```

Figuras 4: Exemplo do uso de Stored Procedures na obtenção de informação

UDFs

UDFs têm os mesmos benefícios dos Stored Procedures e podem ser utilizados para incorporar uma lógica mais complexa dentro de uma consulta. Utilizámos os UDFs Inline Table_Valued para poder aceder e mostrar as tabelas que necessitamos de uma maneira rápida e eficiente (fig.5).

Fizemos uso de UDFs Escalares para podermos retornar valores únicos (fig. 6). No login de cada employee também foi usado um UDF para poder distinguir cada tipo de employee.

```
------------------------cleaner-----------------------------------------------
--- Getting the cleaner info -----
CREATE FUNCTION MANAGEMENT.get_cleaner_information(@staff_id Char(10))
RETURNS TABLE
AS
RETURN
    SELECT
        Work_shift,
        RoomNum
FROM
    HAS has,
    ROOM_SERVICE rs
WHERE
    has.Staff_id = @staff_id AND rs.Cleaner_id = @staff_id
```

Figura 5: Exemplo do uso de UDF Inline Table-Valued

```sql
----- Verificar tipo de Employee -----
CREATE FUNCTION MANAGEMENT.get_type_of_employee(@staff_id Char(10))
RETURNS INT
BEGIN
IF EXISTS(SELECT * FROM MANAGEMENT.CLEANER WHERE MANAGEMENT.CLEANER.Cleaner_id = @staff_id)
BEGIN
    RETURN 1
END
IF EXISTS(SELECT * FROM MANAGEMENT.RECEPTIONIST WHERE MANAGEMENT.RECEPTIONIST.Receptionist_id = @staff_id)
BEGIN
    RETURN 2
END
IF EXISTS(SELECT * FROM MANAGEMENT.BARTENDER WHERE MANAGEMENT.BARTENDER.Bartender_id = @staff_id)
BEGIN
    RETURN 3
END
IF EXISTS(SELECT * FROM MANAGEMENT.CHEF WHERE MANAGEMENT.CHEF.Chef_id = @staff_id)
BEGIN
    RETURN 4
END
    RETURN 0 -- Caso de erro
END
```

Figura 6: Exemplo do uso de UDF Escalar

Conceitos não implementados

Index: Permite uma maior rapidez na pesquisa em tabelas com muitas entradas, visto que é algo que não necessitámos no nosso projeto, não foi necessária a sua implementação.

Triggers: Durante a realização do projeto todos os casos onde poderiam ser usados *triggers* foram usadas funções devido à simplicidade que isso ia proporcionar.

Como usar

As tabelas estão todas populadas, logo há dezenas de customers e employees que se podem usar. Aqui estão alguns exemplos de uso

Na janela do Manager: Hotel id: 8695647822 / Password: password

Na janela do *Employee*: *Cleaner* Example: Staff id: 5378235358

Chef Example: Staff id: 4732578707

Bartender Example: Staff id: 6963947624

Receptionist Example Staff id: 7993251489

Na janela do *Customer*: NIF: 596312599 / Password: 4590689130

Conclusão

A gestão de hotel é um conceito básico e muitas vezes usado como exemplo em aulas de base de dados, daí o incentivo a termos escolhido o tema porque nos ia permitir “inserir” algo mais complexo sobre um conceito básico, por exemplo, os tipos de empregados, refeições, bebidas, etc.

À superfície o conceito que pretendíamos realizar foi bem-sucedido, mas não sem as suas falhas, ainda há funcionalidades que gostaríamos de num futuro, poder implementar, tal como a adição de employee, reviews de refeições, personalização de refeições, uma interface mais apelativa, etc.

Por fim, posso dizer que, aplicar muitos dos conceitos que fomos aprendendo nas aulas para realizar este projeto levou a que após a conclusão do mesmo me sentisse bastante realizado