**Gestão de Aeroporto**

QuAIRy *- Flying with class*

Bases de Dados

Ano Letivo 2018/2019

Turma 6 Grupo 6

Daniel Ferreira Brandão, up201705812, up201705812@fe.up.pt

Henrique José Santos, up201706898, up201706898@fe.up.pt

Pedro Miguel Moás, up201705208, up201705208@fe.up.pt

17 de março de 2019

Índice

[**Índice** 2](#_Toc4936000)

[Tema do trabalho 3](#_Toc4936001)

[Implementação 4](#_Toc4936002)

[Classes e Atributos 6](#_Toc4936003)

[Diagrama de classes UML 7](#_Toc4936004)

[Esquema relacional e Dependências Funcionais 8](#_Toc4936005)

[Formas Normais 10](#_Toc4936006)

[Restrições 12](#_Toc4936007)

Tema do trabalho

Este projeto tem como objetivo criar uma base de dados que conterá a informação necessária para gerir um aeroporto. Isto inclui os seus funcionários e passageiros, como também informações sobre as viagens que chegam e saem do respetivo aeroporto, para que seja possível geri-las.

Implementação

Começando pela classe **Pessoa**, sobre esta interessa saber informações básicas como nome, NIF e data de nascimento, mas para detalhes mais complexos criaram-se especializações desta, as classes **Funcionário**, que guarda o seu salário e categoria, e **Passageiro**, que apenas contém o seu número de passaporte, no caso de haver problemas de verificação.

A parte central é, naturalmente, a classe **Viagem**, que guarda a data de partida, data de chegada, duração, e se é uma partida ou chegada, dependendo do sentido da viagem. A data de chegada poderia ser determinada através dos outros atributos, mas dependeria da diferença de fuso horário entre aeroportos, sendo necessário aceder a várias outras classes sempre que fosse preciso calcular esse atributo.

Naturalmente, uma **Viagem** tem um **Avião** associado (que por sua vez pode ter várias viagens), cuja classe guarda o seu ID. Um **Avião** pertencerá a uma **Companhia Aérea** (que, certamente, poderá ter vários aviões), e está ligado a uma classe **Modelo** **Avião**, que guarda todos os modelos de aviões conhecidos, para não repetir informação no caso de haver vários com o mesmo modelo, assim como as suas capacidades.

Continuando, também é importante que haja uma gestão das bagagens que passam pelo Aeroporto. Assim, uma **Viagem** pode estar ligada a mais do que uma **Mala**. Sobre esta, guarda-se o seu ID (único) e o peso. Naturalmente, esta classe não corresponde ao objeto físico em si, pois se um passageiro trouxer a mesma mala para dois voos diferentes, certamente que o Aeroporto guardará dois IDs diferentes. Esta classe tem sempre um passageiro associado, que pode ter outras malas associadas. Para além da **Mala**, as viagens de chegada estão ainda sempre associadas a um **Terminal de Bagagem**, onde serão despachadas as suas malas, a uma hora de recolha guardada. Como seria de esperar, estes podem estar ligados a várias viagens.

Para completar as classes **Viagem** e **Passageiro**, também é importante notar que não só uma **Viagem** tem vários passageiros, como um **Passageiro** pode ter várias viagens, pois fica registado na base de dados do Aeroporto no caso provável que retorne. Para cada ligação entre um **Passageiro** e uma **Viagem**, guarda-se informação numa classe de associação Bilhete, com a informação sobre a Classe (Executiva – 0, Primeira Classe – 1, Económica – 2), o número de lugar no avião, se já fez Check-In, e se já embarcou, pois é sempre importante saber onde se encontra um passageiro, no caso de haver um eventual problema. Por isso, guardam-se dois booleanos para guardar essa informação.

Agora, relativamente à classe **Funcionário**, este está ligado a um **Local de Trabalho**, que é uma generalização de **Pista** (desolagem / aterragem), **Porta** (embarque / desembarque), **Terminal de Bagagem**, e **Balcão**. Esta última é uma generalização (completa e disjunta) de **Balcão de Apoio** e **Balcão de Check-in**. A classe **Local de Trabalho** é uma generalização incompleta e disjunta, pois existem outros tipos de locais de trabalho, mas cujos detalhes não são tão cruciais. No entanto, esses funcionários ficam registados na mesma, por uma questão de fiabilidade e consistência. Sobre as classes **Pista** e **Porta**, ambas podem ter várias viagens associadas, mas uma **Viagem** apenas tem uma de cada. Por exemplo, no caso de partida do aeroporto, uma viagem está associada a uma Porta de Embarque. Um Balcão de Check-in está associado a uma ou mais companhias aéreas, e estas podem ter vários balcões também. Um **Funcionário** pode ainda ter um chefe, ou ser chefe de vários funcionários.

Por fim, existem ainda classes **País** e **Cidade**, que são usadas para não repetir informação em muitas classes desnecessariamente. Uma pessoa tem um país, e um aeroporto pertence a uma cidade. Naturalmente que uma cidade pertence a um país, e que um país pode ter uma ou mais cidades.

Classes e Atributos

**Aeroporto**

* Nome
* Local

**Avião**

* ID

**Balcão de apoio**

* Hora de abertura
* Hora de fecho

**Balcão de Check-in**

* Número

**Bilhete**

* Classe
* Número de Lugar
* FezCheckin
* Embarcou

**Cidade**

* Nome

**Companhia Aérea**

* Nome
* Contacto

**Funcionário**

* Salário
* Categoria

**Local de Trabalho**

* ID

**Mala**

* ID
* Peso

**Modelo Avião**

* Nome
* Capacidade

**País**

* Nome

**Passageiro**

* Número de passaporte

**Pessoa**

* Nome
* Data de Nascimento
* Contacto
* NIF
* /idade

**Pista**

* Número

**Porta**

* Setor
* Número
* Embarque

**Recolha de bagagem**

* Hora
* Data

**Terminal de bagagem**

* Número

**Viagem**

* Data de partida
* Data de chegada
* Duração
* Partida

Diagrama de classes UML

O seguinte modelo conceptual, que contém as diversas classes, atributos e associações, também se encontra anexado em formato pdf.

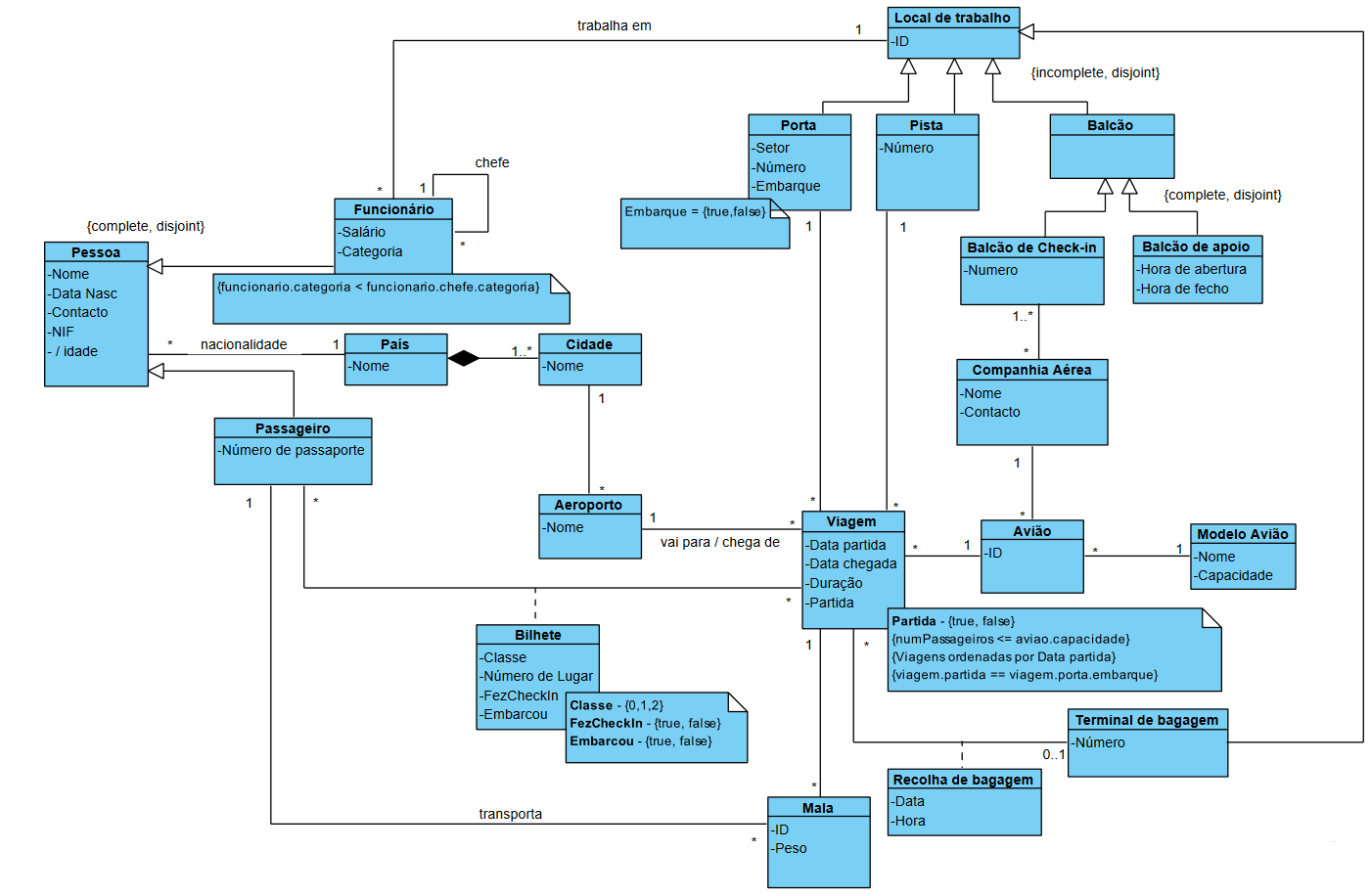


Figura 1 - Diagrama de classes UML

Esquema relacional e Dependências Funcionais

**Person** (PersonID, SSN, PersonName, BirthDate, PhoneNumber, Country->Country)

1. {PersonID} -> {SSN, PersonName, BirthDate, PhoneNumber, Country }
2. {SSN} -> {PersonID}
3. {PhoneNumber} -> {PersonID}

**Employee** (Person->Person, Salary, NIF, Workplace->Workplace)

1. {Person} -> {Salary , NIF , Workplace }
2. {NIF} -> {Person}

**Passenger** (Person->Person, IDnumber)

1. {Person}->{IDnumber}
2. {IDnumber}->{Person}

**IsBoss** (BossID->Employee, BossedID->Employee)

1. {BossedID} -> {BossID}

**Country** (CountryID, CountryName)

1. {CountryID} -> {CountryName}

**City** (CityID, CityName, Country->Country)

1. {CityID} -> {CityName, Country}
2. {CityName, Country} -> {CityID}

**Airport** (AirportCode, City->City, AirportName)

1. {AirportCode}->{City, AirportName}
2. {City, AirportName}->{AirportCode}

**Trip**(TripID)

**Departure** (Trip->Trip, Airport->Airport, DepartureDate, DepartureTime, ArrivalDate, ArrivalTime, DurationHours, DurationMinutes, Gate->Gate, Runway->Runway, Airplane->Airplane)

1. {Trip} -> {Airport, DepartureDate, DepartureTime, ArrivalDate, ArrivalTime, DurationHours, DurationMinutes, Gate, Runway, Airplane}
2. {DepartureDate, DepartureTime, Runway} -> {Trip}
3. {DepartureDate, DepartureTime, Gate} -> {Trip}
4. {DepartureDate, DepartureTime, Airplane}->{Trip}

**Arrival** (Trip->Trip, Airport->Airport, DepartureDate, DepartureTime, ArrivalDate, ArrivalTime, DurationHours, DurationMinutes, Gate->Gate, Runway->Runway, Airplane->Airplane, LuggageBelt->LuggageBelt, DropoffDate, DropoffTime)

1. {Trip}-> {DepartureDate, DepartureTime, ArrivalDate, ArrivalTime, DurationHours, Gate, Runway, Airplane, Airport, LuggageBelt, DropoffDate, DropoffTime}
2. {ArrivalDate, ArrivalTime, Runway} -> {Trip}
3. {DepartureDate, DepartureTime, Gate} -> {Trip}
4. {DepartureDate, DepartureTime, Airplane}->{Trip}

**Class** (ClassID, ClassName)

1. {ClassID} -> {ClassName}
2. {ClassName} -> {ClassID}

**Ticket** (Passenger->Passenger, Trip->Trip, SeatRow, SeatLetter, HasCheckedIn, HasBoarded, HasEnteredBoardingZone, Class->Class)

1. {Passenger, Trip}->{SeatRow, SeatLetter, HasCheckedIn, HasBoarded, HasEnteredBoardingZone, Class}
2. {SeatRow, SeatLetter, Trip}->{Passenger, Trip}

**Luggage** (LuggageID, Weight, Trip->Trip, Passenger->Passenger)

1. {LuggageID}->{Weight, Trip, Passenger}

**Airplane** (AirplaneID, AirplaneName, Airline->Airline, AirplaneModel->AirplaneModel)

1. {AirplaneID}->{AirplaneName, Airline->Airline, AirplaneModel->AirplaneModel}
2. {AirplaneName, Airline->Airline}->{AirplaneID}

**Airline** (AirlineID, AirlineName, PhoneNumber)

1. {AirplaneID} -> {AirlineName, PhoneNumber}
2. {PhoneNumber} -> {AirlineID}
3. {AirlineName} -> {AirlineID}

**AirplaneModel** (ModelID, ModelName, SeatsPerRow, NumRows)

1. {ModelID} -> {ModelName, SeatsPerRow, NumRows}

**Workplace** (WorkplaceID, WorkplaceName)

1. {WorkplaceID} -> {WorkplaceName}
2. {WorkplaceName} -> {WorkplaceID}

**Gate** (Workplace->Workplace, GateName, IsBoardingGate)

1. {Workplace}->{GateName, IsBoardingGate}
2. {GateName} -> {Workplace}

**Runway** (Workplace->Workplace, RunwayNum)

1. {Workplace} -> {RunwayNum}
2. {RunwayNum} -> {Workplace}

**Desk** (Workplace->Workplace)

**CheckInDesk** (Desk->Desk, CheckInNum)

1. {Desk}->{CheckInNum}
2. {CheckInNum}->{Desk}

**HelpDesk** (Desk->Desk, OpenTime, CloseTime)

1. {Desk}->{OpenTime,CloseTime}

**LuggageBelt** (Workplace->Workplace, BeltNum)

1. {Workplace} -> {BeltNum}
2. {BeltNum}->{Workplace}

**HasDesk** (Airline -> Airline, CheckInDesk->CheckInDesk)

Formas Normais

Analisando então as dependências funcionais listadas, podemos ver que, em praticamente todas as relações, existem um ou mais IDs que permite chegar a todos os outros atributos da relação, sendo esses IDs chaves das relações. Assim, para todas as relações com apenas uma dependência funcional listada, conseguimos concluir que estão na forma normal de Boyce-Codd, pois o lado esquerdo contém sempre o ID da relação, que é uma chave (no caso de Airport não guardamos um número mas sim um código único, o que terá o mesmo efeito).

Para as relações que têm mais do que uma dependência funcional, consegue-se observar que os atributos do lado esquerdo permitem inferir os tais IDs, que são chaves das relações, sendo trivial concluir que esses atributos são, por sua vez, chaves da relação.

Relativamente às relações sem dependências funcionais, a chave corresponderá a todos os atributos.

Concluindo, como para todas as dependências funcionais, os atributos do lado esquerdo correspondem a sempre chaves da relação respetiva, o modelo relacional está na Forma Normal de Boyce-Codd, o que implica que esteja também na 3ª Forma Normal, não havendo violações a nenhuma destas Formas Normais.

Restrições

**Person:**

PersonID é a chave primária (PRIMARY KEY)

SSN e PhoneNumber são atributos únicos (UNIQUE)

BirthDate tem um comprimento obrigatório de 10 caractéres (CHECK)

Country é uma chave estrangeira (REFERENCES) de CountryID da classe Country

**Employee:**

PersonID é a chave primária (PRIMARY KEY) e chave estrangeira (REFERENCES) de PersonID da classe Person

Salary é um número positivo superior a 600 (CHECK) euros

WorkplaceID é uma chave estrangeira (REFERENCES) de WorkplaceID da classe Workplace

**IsBoss:**

BossID e BossedID são chaves estrangeiras (REFERENCES) de PersonID da classe Employee

\*\*\*Moas – explica o cascade \*\*\*

Um employee não pode ser chefe de si (CHECK) mesmo (BossID != BossedID)

**Passenger:**

PersonID é a chave primária (PRIMARY KEY)

IDNumber é um atributo único (UNIQUE)

**Country:**

CountryID é a chave primária (PRIMARY KEY)

**City:**

CityID é a chave primária (PRIMARY KEY)

CountryID é uma chave estrangeira (REFERENCES) de CountryID da classe Country

\*\*\*Moas, cascade :((( \*\*\*

**Airport:**

AirportCode é uma chave primária (PRIMARY KEY) de comprimento obrigatório de 3 caractéres (CHECK)

CityID é uma chave estrangeira (REFERENCES) de CityID da classe City

**Trip:**

TripID é a chave primária (PRIMARY KEY)

**Departure:**

TripID é a chave primária (PRIMARY KEY) e chave estrangeira (REFERENCES) de TripID da classe Trip

AirportCode é uma chave estrangeira (REFERENCES) de AirportCode da classe Airport

DepartureTime e ArrivalTime têm comprimento obrigatório de 5 caractéres (CHECK)

DepartureDate e ArrivalDate têm comprimento obrigatório de 10 caractéres (CHECK)

DurationHours é um número positivo (CHECK)

DurationMinutes é um número entre 0 e 59 (CHECK)

GateID é uma chave estrangeira (REFERENCES) de GateID da classe Gate

RunwayID é uma chave estrangeira (REFERENCES) de RunwayID da classe Runway

AirplaneID é uma chave estrangeira (REFERENCES) de AirplaneID da classe Airplane

Não podem existir dois Departures com a mesma DepartureDate, DepartureTime e usar o mesmo GateID, RunwayID ou AirplaneID (UNIQUE)

Pelo menos um de DurationHours ou DurationMinutes tem de ser positivo (CHECK)

**Arrival:**

TripID é a chave primária (PRIMARY KEY) e chave estrangeira (REFERENCES) de TripID da classe Trip

AirportCode é uma chave estrangeira (REFERENCES) de AirportCode da classe Airport

DepartureTime e ArrivalTime têm comprimento obrigatório de 5 caractéres (CHECK)

DepartureDate e ArrivalDate têm comprimento obrigatório de 10 caractéres (CHECK)

DurationHours é um número positivo (CHECK)

DurationMinutes é um número entre 0 e 59 (CHECK)

GateID é uma chave estrangeira (REFERENCES) de GateID da classe Gate

RunwayID é uma chave estrangeira (REFERENCES) de RunwayID da classe Runway

AirplaneID é uma chave estrangeira (REFERENCES) de AirplaneID da classe Airplane

BeltID é uma chave estrangeira (REFERENCES) de WorkplaceID da classe LuggageBelt

Não podem existir dois Departures com a mesma DropoffDate, DropoffTime e usar o mesmo BeltID (UNIQUE)

Não podem existir dois Departures com a mesma DepartureDate, DepartureTime e usar o mesmo GateID, RunwayID ou AirplaneID (UNIQUE)

Pelo menos um de DurationHours ou DurationMinutes tem de ser positivo (CHECK)

**Class:**

ClassID é a chave primária (PRIMARY KEY)

**Ticket:**

SeatRow, SeatLetter e TripID são a chave primária (PRIMARY KEY)