U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

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

Tema do trabalho 3

Implementação 4

Modelo Conceptual 6

Modelo Conceptual Revisto 7

Esquema relacional e Dependências Funcionais 8

  Formas Normais 11

Restrições 12

  Notas 19

Interrogações 20

Gatilhos 21

  Restrições adicionais utilizando Gatilhos 21

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 **Person**, sobre esta interessa saber informações básicas da pessoa, como SSN, nome, data de nascimento e número de telefone, mas para detalhes mais complexos criaram-se especializações desta, as classes **Employee**, que guarda o seu salário e NIF, e **Passenger**, que apenas contém o seu número de identificação, no caso de haver problemas de verificação.

A parte central é, naturalmente, a classe **Trip**, que guarda a data/hora de partida, data/hora de chegada e duração. 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. A classe **Trip** é uma generalização de **Departure** e **Arrival**, que ajudarão a separar as viagens para melhor clareza.

Naturalmente, uma **Trip** tem um **Airplane** associado (que por sua vez pode ter várias viagens), cuja classe guarda o seu nome. Um **Airplane** pertencerá a uma **Airline** (que, certamente, poderá ter vários aviões), e está ligado a uma classe **AirplaneModel**, que guarda todos os modelos de aviões conhecidos, para não repetir informação no caso de haver vários com o mesmo modelos. Para além do nome do modelo, é mantido o número de filas no avião, assim como o número de lugares por fila, possivelmente útil para a atribuição de lugares. Por fim, as viagens têm sempre uma ligação com a classe **Airport**, que guarda um código de identificação (sigla) e um nome.

Continuando, também é importante que haja uma gestão das bagagens que passam pelo aeroporto. Assim, uma **Trip** pode estar ligada a mais do que uma **Luggage**. Sobre esta última classe apenas poderá interessar saber o seu 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 duas instâncias diferentes. Esta classe tem sempre um passageiro associado, que pode ter outras malas associadas. A classe **Arrival** está ainda sempre associada a um **LuggageBelt**, onde serão despachadas as suas malas, a uma hora de recolha guardada na classe de associação **LuggageDropoff**. Como seria expectável, estes podem estar ligados a várias viagens.

Para unir as classes **Trip** e **Passenger**, criou-se uma classe **Ticket**. Esta contém o lugar do passageiro, assim como informação sobre se ele já fez check-in, se já entrou na zona de embarque e se já embarcou. Estes últimos três atributos apenas são relevantes nas viagens de partida. **Ticket** também se associa com uma classe **Class**, que apenas tem um nome (“Primeira Classe”, “Executiva”, etc.). Deste modo, um bilhete tem um único passageiro e uma única viagem, fazendo então a ligação entre **Trip** e **Passenger**.

Agora, relativamente à classe **Funcionário**, este está ligado a um **Workplace**, que é uma generalização de **Runway**, **Gate** (guarda booleano “*isBoardingGate*”), **LuggageBelt**, e **Desk**. Esta última é uma generalização (completa e disjunta) de **HelpDesk** e **CheckInDesk**. A classe **Workplace** 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 **Runway** e **Gate**, ambas podem ter várias viagens associadas, mas uma **Trip** apenas tem uma de cada. Por exemplo, no caso de partida do aeroporto, uma viagem está associada a uma Porta de Embarque. Um **CheckInDesk** está ligado a uma ou mais companhias aéreas, e estas podem ter vários balcões também. Um **Employee** pode ainda ter um chefe, ou ser chefe de vários funcionários.

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

Modelo Conceptual

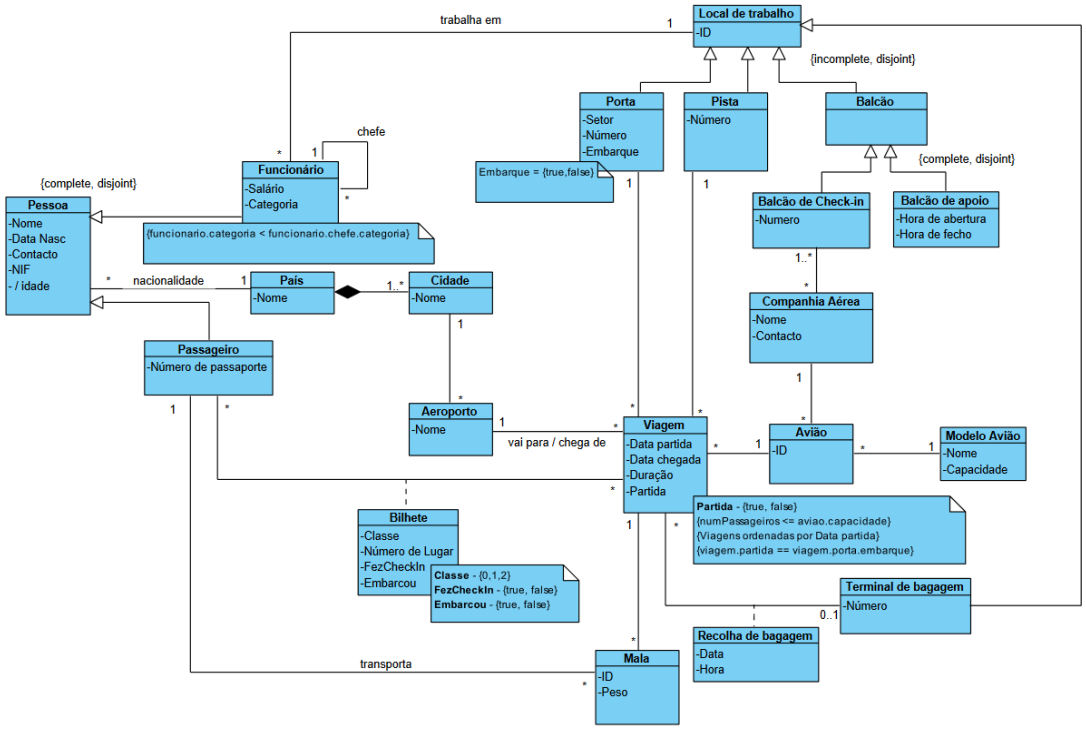

Figura 1 - Diagrama de classes UML

Modelo Conceptual Revisto

O diagrama anterior foi revisto, estando o seguinte anexado em formato pdf.

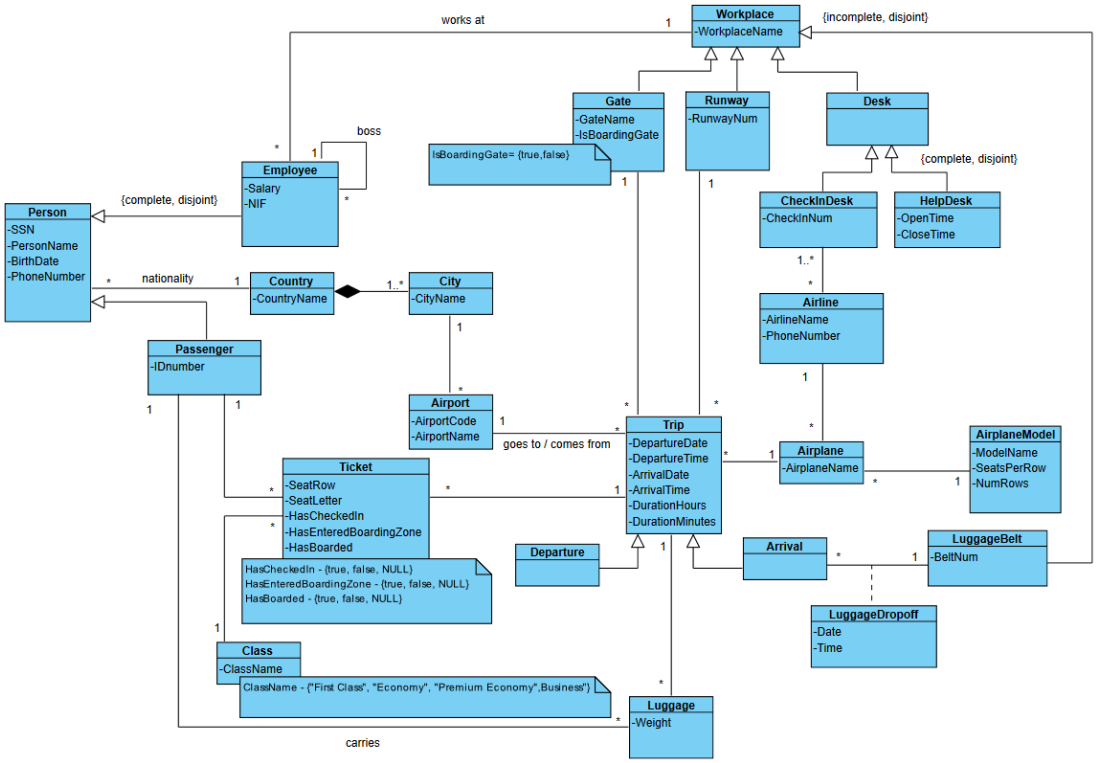

Figura 2 - Diagrama de classes UML (revisto)

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}

Chaves: {PersonID}, {SSN}, {PhoneNumber}

Employee (PersonID->Person, Salary, NIF, WorkplaceID->Workplace)

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

Chaves: {PersonID}, {NIF}

Passenger (PassengerID->Person, IDnumber)

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

Chaves: {PersonID}, {IDnumber}

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

1. {BossedID} -> {BossID}

Chaves: {BossedID}

Country (CountryID, CountryName)

1. {CountryID} -> {CountryName}
2. {CountryName} -> {CountryID}

Chaves: {CountryID}, {CountryName}

City (CityID, CityName, CountryID->Country)

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

Chaves: {CityID}, {CityName, Country}

Airport (AirportCode, CityID->City, AirportName)

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

Chaves: {AirportCode},{City , AirportName}

Trip (TripID, AirportCode->Airport, DepartureDate, DepartureTime, ArrivalDate, ArrivalTime, DurationHours, DurationMinutes, GateID->Gate, RunwayID->Runway, AirplaneID->Airplane)

1. {TripID} -> {AirportCode, DepartureDate, DepartureTime, ArrivalDate, ArrivalTime, DurationHours, DurationMinutes, GateID, RunwayID, AirplaneID}

Chaves: {TripID}

Departure (TripID->Trip)

Chaves: {TripID}

Arrival (TripID>Trip, BeltID->LuggageBelt, DropoffDate, DropoffTime)

1. {TripID} -> {BeltID, DropoffDate, DropoffTime}
2. {BeltID, DropoffDate, DropoffTime} -> {TripID}

Chaves: {TripID}, {LuggageBelt, DropoffDate, DropoffTime}

Class (ClassID, ClassName)

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

Chaves: {ClassID}, {ClassName}

Ticket (PassengerID->Passenger, TripID->Trip, SeatRow, SeatLetter, HasCheckedIn, HasBoarded, HasEnteredBoardingZone, ClassID->Class)

1. {SeatRow, SeatLetter, TripID} -> {PassengerID, HasCheckedIn, HasBoarded, HasEnteredBoardingZone, ClassID}

Chaves: {SeatRow, SeatLetter, TripID}

Luggage (LuggageID, Weight, TripID->Trip, PassengerID->Passenger)

1. {LuggageID} -> {Weight, TripID, PassengerID}

Chaves: {LuggageID}

Airplane (AirplaneID, AirplaneName, AirlineID->Airline, ModelID->AirplaneModel)

1. {AirplaneID} -> {AirplaneName, AirlineID, ModelID}

Chaves: {AirplaneID}

Airline (AirlineID, AirlineName, PhoneNumber)

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

Chaves: {AirplaneID}, {PhoneNumber}, {AirlineName}

AirplaneModel (ModelID, ModelName, SeatsPerRow, NumRows)

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

Chaves: {ModelID}, {ModelName}

Workplace (WorkplaceID, WorkplaceName)

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

Chaves: {WorkplaceID}, {WorkplaceName}

Gate (WorkplaceID->Workplace, GateName, IsBoardingGate)

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

Chaves: {WorkplaceID}, {GateName}

Runway (WorkplaceID->Workplace, RunwayNum)

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

Chaves: {WorkplaceID}, {RunwayNum}

CheckInDesk (WorkplaceID -> Workplace, CheckInNum)

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

Chaves: {WorkplaceID}, {CheckInNum}

HelpDesk (WorkplaceID -> Workplace, OpenTime, CloseTime)

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

Chaves: {WorkplaceID}

LuggageBelt (WorkplaceID ->Workplace, BeltNum)

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

Chaves: {WorkplaceID}, {BeltNum}

HasCheckInDesk (AirlineID -> Airline, DeskID->CheckInDesk)

Chaves: {Airline, DeskID}

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 sempre a 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), não podendo haver duas pessoas com o mesmo ID.

SSN e PhoneNumber são atributos únicos, não podendo haver SSN repetidos, nem PhoneNumber repetidos (UNIQUE)

PersonName não pode ser nulo (NOT NULL)

BirthDate tem um comprimento obrigatório de 10 carateres (CHECK)

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

Passenger

PersonID é a chave primária (PRIMARY KEY), não podendo haver dois passageiros com o mesmo ID, e chave estrangeira (REFERENCES) de PersonID da classe Person (ON DELETE CASCADE, ON UPDATE CASCADE)

Não pode haver dois passageiros com o mesmo IDnumber (UNIQUE).

IDnumber não pode ser nulo (NOT NULL)

Employee

PersonID é a chave primária (PRIMARY KEY), não podendo haver dois funcionários com o mesmo ID, e chave estrangeira (REFERENCES) de PersonID da classe Person (ON DELETE CASCADE, ON UPDATE CASCADE)

Não pode haver dois funcionários com o mesmo NIF (UNIQUE).

NIF não pode ser nulo (NOT NULL)

Salary é um número real igual ou superior a 600 (CHECK) euros

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

IsBoss

BossedID é chave primária (PRIMARY KEY), não podendo haver um funcionário com mais do que um chefe.

BossID e BossedID são chaves estrangeiras (REFERENCES) de PersonID da classe Employee (Para ambas: ON DELETE CASCADE, ON UPDATE CASCADE)

BossID não pode ser nulo (NOT NULL)

Um Employee não pode ser chefe de si mesmo (CHECK) (BossID =/= BossedID)

Country

CountryID é a chave primária (PRIMARY KEY), não podendo haver dois países com o mesmo ID.

Não pode haver dois países com o mesmo CountryName (UNIQUE)

CountryName não pode ser nulo (NOT NULL)

City

CityID é a chave primária (PRIMARY KEY), não podendo haver duas cidades com o mesmo ID.

CountryID é uma chave estrangeira (REFERENCES) de CountryID da classe Country (ON DELETE CASCADE, ON UPDATE CASCADE)

CityName não pode ser nulo (NOT NULL)

CityID e CountryID são únicos para uma cidade (UNIQUE)

Airport

AirportCode é uma chave primária (PRIMARY KEY) de comprimento obrigatório de 3 carateres (CHECK), não podendo haver dois aeroportos com o mesmo código.

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

Não podem existir dois aeroportos com os mesmos AirportName e CityID (UNIQUE).

Trip

TripID é a chave primária (PRIMARY KEY), não podendo haver duas viagens com o mesmo ID.

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

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

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

DurationHours é um número não negativo (CHECK)

DurationMinutes é um número entre 0 e 59 (CHECK)

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

Nenhum atributo de Trip pode ser nulo (NOT NULL)

Departure

TripID é a chave primária (PRIMARY KEY), não podendo haver duas partidas com o mesmo TripID, e chave estrangeira (REFERENCES) de TripID da classe Trip (ON DELETE CASCADE ON UPDATE CASCADE).

Arrival

TripID é a chave primária (PRIMARY KEY), não podendo haver duas chegadas com o mesmo TripID, e chave estrangeira (REFERENCES) de TripID da classe Trip (ON DELETE CASCADE ON UPDATE CASCADE).

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

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

Nenhum atributo de Arrival pode ser nulo (NOT NULL)

Class

ClassID é a chave primária (PRIMARY KEY), não havendo duas classes com o mesmo ID.

Não podem existir duas classes com o mesmo ClassName (UNIQUE).

Ticket

SeatRow, SeatLetter e TripID são a chave primária (PRIMARY KEY), não sendo possível haver dois bilhetes com os três atributos iguais.

PassengerID é uma chave estrangeira (REFERENCES) de PersonID da classe Passenger

PassengerID e ClassID não podem ser nulos (NOT NULL)

SeatRow é um inteiro positivo (CHECK), e SeatLetter tem comprimento de apenas um caráter (CHECK)

HasCheckedIn, HasEnteredBoardingZone e HasBoarded são booleanos, podendo ter os valores 0 (falso), 1 (verdadeiro) ou NULL (caso seja um bilhete de Arrival) (CHECK)

ClassID é uma chave estrangeira (REFERENCES) de ClassID da classe Class

Luggage

LuggageID é a chave primária (PRIMARY KEY), não podendo haver duas malas com o mesmo ID.

Weight é superior a 0 (CHECK)

TripID é uma chave estrangeira (REFERENCES) de TripID da classe Trip

PersonID é uma chave estrangeira (REFERENCES) de PersonID da classe Passanger

Nenhum atributo de Luggage pode ser nulo (NOT NULL)

Airplane

AirplaneID é a chave primária (PRIMARY KEY), não podendo haver dois aviões com o mesmo ID.

AirlineID é uma chave estrangeira (REFERENCES)de AirlineID da classe Airline

ModelID é uma chave estrangeira (REFERENCES) de ModelID da classe AirplaneModel

Nenhum atributo de Airplane pode ser nulo (NOT NULL)

Airline

AirlineID é a chave primária (PRIMARY KEY)

Não podem haver duas Airlines com o mesmo AirlineName (UNIQUE)

Não podem haver duas Airlines com o mesmo PhoneNumber (UNIQUE)

Nenhum atributo de Airline pode ser nulo (NOT NULL)

AirplaneModel

ModelID é a chave primária (PRIMARY KEY), por isso não há dois modelos IDs iguais.

SeatsPerRow e NumRows são inteiros superiores a 0 (CHECK)

Nenhum atributo de AirplaneModel pode ser nulo (NOT NULL)

Workplace

WorkplaceID é a chave primária (PRIMARY KEY), não podendo haver dois Workplaces com o mesmo ID.

Não podem existir dois Workplaces com o mesmo WorkplaceName (UNIQUE).

Runway

WorkplaceID é a chave primária (PRIMARY KEY), não podendo haver duas Runways com o mesmo ID, e chave estrangeira (REFERENCES) de WorkplaceID da classe Workplace (ON DELETE CASCADE ON UPDATE CASCADE)

Não podem existir duas Runways com o mesmo RunwayNum (UNIQUE).

Nenhum atributo de Runway pode ser nulo (NOT NULL).

LuggageBelt

WorkplaceID é a chave primária (PRIMARY KEY), não podendo haver dois LuggageBelts com o mesmo ID, e chave estrangeira (REFERENCES) de WorkplaceID da classe Workplace (ON DELETE CASCADE ON UPDATE CASCADE)

Não podem existir dois LuggageBelts com o mesmo BeltNum (UNIQUE).

Nenhum atributo de LuggageBelt pode ser nulo (NOT NULL).

CheckInDesk

WorkplaceID é a chave primária (PRIMARY KEY), não podendo haver dois CheckInDesks com o mesmo ID, e chave estrangeira (REFERENCES) de WorkplaceID da classe Workplace (ON DELETE CASCADE ON UPDATE CASCADE)

Não podem existir dois CheckInDesks com o mesmo CheckInName (UNIQUE).

Nenhum atributo de CheckInDesk pode ser nulo (NOT NULL).

HelpDesk

WorkplaceID é a chave primária (PRIMARY KEY), não podendo haver dois HelpDesks com o mesmo ID, e chave estrangeira (REFERENCES) de WorkplaceID da classe Workplace (ON DELETE CASCADE ON UPDATE CASCADE)

OpenTime e CloseTime têm comprimento obrigatório de 5 carateres (CHECK).

OpenTime é uma hora anterior a CloseTime (CHECK).

Nenhum atributo de HelpDesk pode ser nulo (NOT NULL).

HasCheckInDesk

AirlineID e DeskID são a chave primária (PRIMARY KEY), logo não podem haver dois tuplos na tabela com o mesmo par de AirlineID e DeskID.

AirlineID é chave estrangeira (REFERENCES) de AirlineID da classe Airline (ON DELETE CASCADE ON UPDATE CASCADE).

DeskID é chave estrangeira (REFERENCES) de DeskID da classe CheckInDesk (ON DELETE CASCADE ON UPDATE CASCADE).

Gate

WorkplaceID é a chave primária (PRIMARY KEY), não podendo haver duas Gates com o mesmo ID, e chave estrangeira (REFERENCES) de WorkplaceID da classe Workplace (ON DELETE CASCADE ON UPDATE CASCADE)

Não podem existir duas Gates com o mesmo GateName (UNIQUE).

IsBoardingGate é um booleano e, como tal, pode assumir os valores 0 (falso) ou 1 (true) (CHECK).

Nenhum atributo de Gate pode ser nulo (NOT NULL).

Notas

Os atributos classificados como chaves primárias (PRIMARY KEY) ou únicos (UNIQUE) não podem aparecer duas vezes repetidas na mesma tabela.

As chaves primárias não podem ser nulas.

Os atributos classificados como chaves estrangeiras têm de obrigatoriamente fazer parte da tabela referenciada.

As chaves estrangeiras que contêm indicações ON UPDATE/ON DELETE, especificam a ação que deve ocorrer perante uma alteração/remoção do tuplo referenciado. No caso de SET NULL, as chaves estrangeiras passarão a NULL, no caso de CASCADE, os tuplos que referenciam atributos alterados/removidos serão também alterados/removidos. A ação por omissão é RESTRICT, ou seja, impede que tal alteração/remoção ocorra.

Interrogações

As 10 interrogações guardadas em ficheiro .sql, estão abaixo listadas em linguagem natural, pela mesma ordem.

1. Listagem do peso total usado pelas bagagens em cada viagem.
2. Listagem de passageiros registados que sempre viajaram em 1ª classe.
3. Listagem de passageiros (com todas as datas e horas de viagem) que, no mês atual, fizeram check-in mas não embarcaram em viagens de partida.
4. Passageiros que, no mesmo dia, chegaram ao aeroporto e partiram do mesmo (com respetivas horas de desembarque e embarque).
5. Número de ligações (países: partidas e chegadas) de cada companhia aérea, ordenado por numero de ligações, de seguida pelo nome da companhia aérea e por fim alfabeticamente pelo país de destino.
6. Listagem do número de voos no aeroporto por hora.
7. Listagem de todas as viagens do aeroporto (tipo, data e hora) ordenadas por data e hora (útil para representação nos painéis de voos)
8. Média de salários e média de idades dos trabalhadores de cada local de trabalho (ordenado por salário e depois por idades).
9. Luggage belts disponíveis para cada voo de chegada.
10. Listagem de balcões de check-in atribuídos a cada viagem de partida, assim como o número de bilhetes comprados para cada uma.

Gatilhos

1. **NovoTicket**

Após ser criado um novo Ticket (*AFTER INSERT*), os seus atributos: HasCheckedIn, HasEnteredBoardingZone e HasBoarded são inicializados dependendo do tipo de viagem. Caso seja uma Departure, são colocados a false (0), pois sendo bilhetes novos, ainda não podem ter passado por nenhuma fase de embarque do aeroporto. Se for um Arrival, ficarão a NULL.

2. **RemoverEmployee**

Quando um Employee é removido (*AFTER DELETE*), é averiguado se nenhum outro Employee trabalha no seu antigo Workplace e também se exitem Workplaces em que trabalham mais do que uma pessoa. Caso seja verificado, este gatilho procura o Workplace em que mais pessoas trabalham e transfere um Employee desse local para o que ficou vazio.

3. **AdicionarDepartureTrip e AdicionarArrivalTrip**

Antes de ser adicionada uma nova viagem (Departure ou Arrival) (*BEFORE INSERT*), é verificado, entre todas as outras viagens (Departures e Arrivals), se alguma utiliza a mesma Gate ou Runway à mesma hora do mesmo dia em que a nova viagem a iria utilizar. Caso alguma esteja a ser utilizada, este gatilho impede a inserção da Trip na tabela e retorna uma mensagem de erro.

Restrições adicionais utilizando Gatilhos

As seguintes restrições também poderiam ser utilizadas adicionando outros gatilhos:

1. Antes de ser adicionado uma nova viagem(Departure ou Arrival), verificar se a o tipo de Gate corresponde ao tipo de viagem, ou seja, caso seja uma Departure, o parametro da sua gate isBoardingGate tem de ser TRUE e caso seja um Arrival deve ser FALSE.

2. Antes de ser adicionado uma nova viagem(Departure ou Arrival), verificar se o seu tripID não existe como tripID de uma viagem do tipo contrário.

3. Antes de ser adicionado um novo bilhete, certificar que existe uma viagem (Departure ou Arrival) com o tripID correspondente.