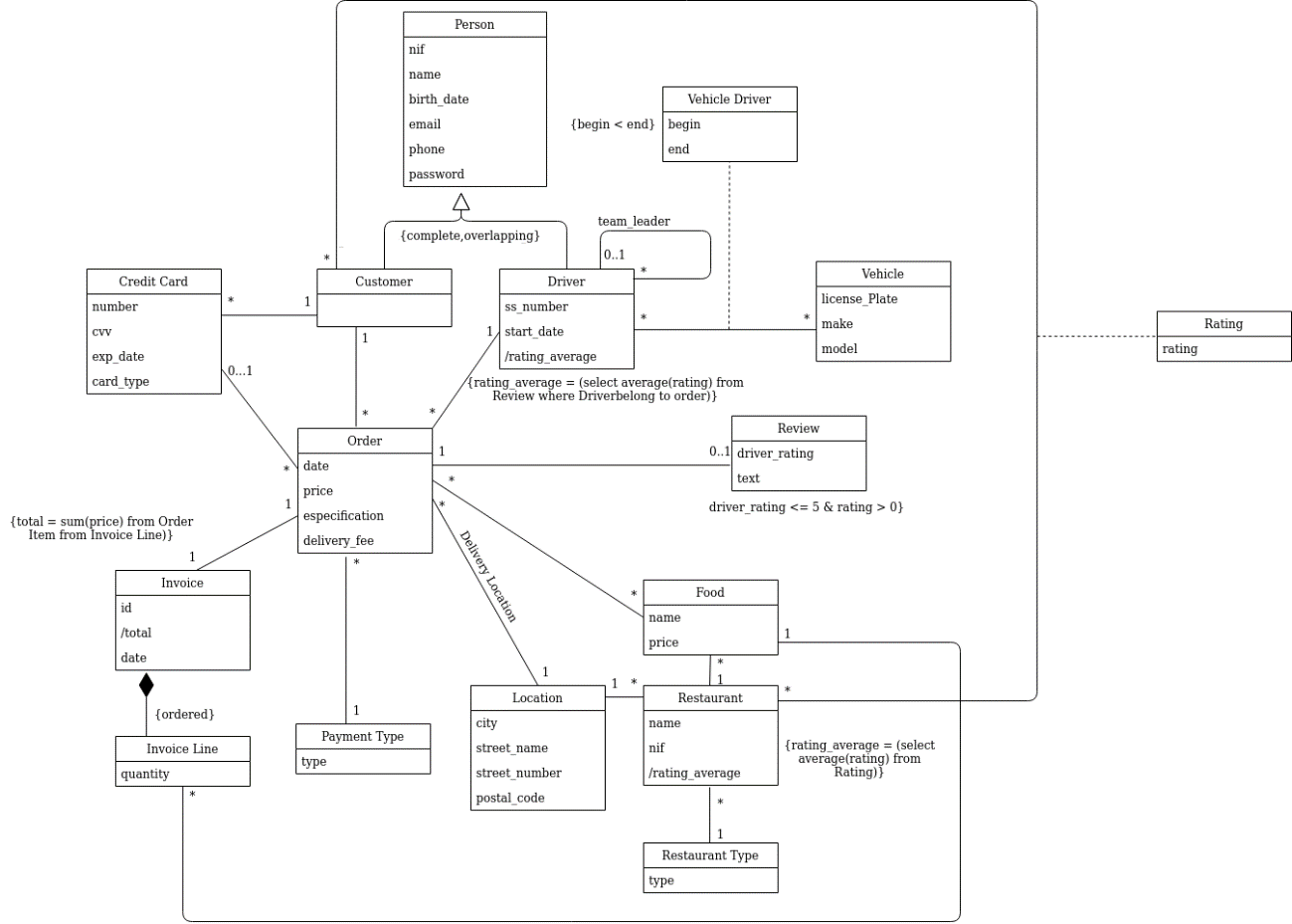
BDAD – Relatório: Empresa de Entregas

João Martins, José Miguel Maçães e Miguel Charchalis

**Esquema UML:**



**Contexto:**

De cada cliente interessa saber o nome, o NIF, o email, a data de nascimento, o número de telemóvel e a palavra passe. Isto tudo também é informação necessária relativa a cada condutor, para além do número de segurança social e a data em que começou a trabalhar na plataforma. Para além disso interessa saber as horas de início e de fim da sessão de trabalho do condutor.

Cada cliente necessita de ter um cartão de crédito, do qual deve ser conhecido o número, o CVV, a data de validade e a rede (VISA, MasterCard, etc.)

Cada condutor deve ter a si associado um veículo, identificado pela matrícula, do qual interessa saber a marca e o modelo. Um condutor pode ser chefe de equipa, sendo responsável por outros condutores.

O cliente pode efetuar o pedido, que será entregue por um condutor e é constituído por uma data, um modo de pagamento e um preço, calculado a partir do preço da comida e da taxa de entrega. Para além disso terá uma avaliação que terá uma classificação, entre 1 e 5, e poderá ou não ter um texto. As classificações são usadas para calcular a média de classificações do condutor. O pedido estará também associado a um local de entrega, caracterizado pela cidade, nome da rua, número da rua, código postal.

O Restaurante tem como elementos identificativos o nome, o tipo, a localização, da qual interessa saber o mesmo que o local de entrega, e a classificação, é a média de classificações de clientes. Disponibiliza pratos, cada um com um nome e preço, que constituem os itens pedidos, juntamente com a quantidade de cada prato. Estes itens fazem parte do pedido e estão presentes na fatura, também associada ao pedido composta por um número identificativo, os itens pedidos e a sua quantidade, o preço total e a data.

Modelo Relacional:

Person (NIF, name, birth\_date, email, phone, password)

Person\_FD = {NIF -> name, birth\_date, email, phone, password}

NIF -> {name, birth\_date, email, phone, password}

Não viola BCNF porque todos os atributos à direita são não primos.

Customer (customerNIF -> Person)

Customer\_FD = {customerNIF}

Só um atributo, não viola BCNF.

Driver (driverNIF -> Person, ss\_number, start\_date, team\_leader, /rating\_average)

FD = {driverNIF -> ss\_number, start\_date, team\_leader, /rating\_average}

driverNIF -> {ss\_number, start\_date, team\_leader, /rating\_average}

Não viola BCNF porque todos os atributos à direita são não primos.

Vehicle (license\_plate, make, model)

FD = {license\_plate -> make, model}

license\_plate -> {make, model}

Não viola BCNF porque todos os atributos à direita são não primos.

VehicleDriver (driverNIF -> Driver, vehicle\_license\_plate -> Vehicle, begin, end)

FD = {(driverNIF,vehicle\_license\_plate) -> begin, end}

//Violação – Usar o “driverNIF” e “vehicle\_license\_plate” como primary key leva à repetição da primary key e a mesma key se relacionar com diferentes valores. Uma possível solução é trocar “vehicle\_license\_plate” com “begin”, pois é impossível o mesmo condutor começar a conduzir duas vezes no mesmo tempo, torna a primary key única.

VehicleDriver (driverNIF -> Driver, begin, vehicle\_license\_plate -> Vehicle, end)

FD = {(driverNIF, begin) -> vehicle\_license\_plate, end}

driverNIF -> end

begin -> vehicle\_license\_plate

{driverNIF, begin} -> {vehicle\_license\_plate, end}

Não viola BCNF porque nenhum atributo no lado direito depende de outro no lado direito: end não depende de vehicle\_license\_plate, nem vice-versa. Apesar da chave ser única e respeitar a normalização, não impede que um condutor tenha períodos sobrepostos.

Team (driverNIF -> Driver, leaderNIF -> Driver)

FD = {diverNIF -> leaderNIF}

diverNIF -> leaderNIF

Não tem dependências funcionais. Até que pode existir um driver sem leader, sendo ele um leader.

CreditCard (number, cvv, exp\_date, card\_type, customerNIF -> Customer)

FD = {(number, customerNIF) -> cvv, exp\_date, card\_type}

number -> {cvv, exp\_date, card\_type}

{number, customerNIF}-> {cvv, exp\_date, card\_type}

Viola BCNF pois duas pessoas podem usar o mesmo cartão, então a informação está repetida. Neste caso não é pretendido o update nem delete geral devido à proteção de dados: se um cliente mudar algum dado, por exemplo o cvv, não se pretende que mude também para outros clientes que estão a usar o mesmo cartão noutra conta, pois devem ser os mesmos a mudar.

Demand (demandID, date, price, specification, delivery\_fee, customerNIF -> Customer, driverNIF -> Driver, locationID -> Location, paymentTypeID -> PaymentType, creditCardID -> CreditCard)

FD = {demandID -> date, price, specification, delivery\_fee, customerNIF, driverNIF, locationID, paymentTypeID, creditCardID; date -> delivery\_fee}

date -> delivery\_fee

demandID -> {date, price, specification, delivery\_fee, customerNIF, driverNIF, locationID, paymentTypeID, creditCardID}

Viola BCNF, mas não viola 3NF porque existe atributos à direita dependem de outra key.

PaymentType (paymentTypeID, type)

FD = {paymentTypeID -> type}

paymentTypeID -> type

Não viola BCNF porque todos os atributos à direita são não primos e só tem um atributo de cada lado.

Review (reviewID, rating, text, demandID -> Demand)

FD = {reviewID -> rating, text, demandID; demandID -> reviewID}

reviewID -> {rating, text, demandID}

Não viola BCNF porque só pode existir uma review por demand e se a demand for apagada então a review também devia ser.

Rating (ratingID, rating, restaurantID -> Restaurant) - Class associação.

FD = {ratingID -> rating, restaurantID}

ratingID -> rating, restaurantID

Não viola BCNF porque todos os atributos à direita são não primos e só tem um atributo de cada lado.

Food (foodID, name, price, restaurantID -> Restaurant)

FD = {foodID -> name, price, restaurantID}

foodID -> {name, price}

foodID -> {name, price, restaurantID}

Não viola BCNF porque só pode existir uma review por demand e se a demand for apagada então a review também devia ser.

Demanded (foodID -> Food, demandID -> Demand, quantity)

FD = {demandID -> foodID, quantity}

{demandID, foodID} -> quantity

Não viola BCNF porque a quantidade depende exclusivamente da demand e da comida.

Restaurant (restaurantID, name, NIF, locationID ->Location, /rating\_average, restaurantTypeID -> RestaurantType)

FD = {restaurantID -> name, NIF, locationID, /rating\_average, restaurantTypeID}

restaurantID -> {name, NIF, /rating\_average}

restaurantID -> {name, NIF, locationID, /rating\_average, restaurantTypeID}

Não viola BCNF porque todos os atributos à direita dependem do restaurantID.

RestaurantType (restaurantTypeID, type)

FD = {restaurantTypeID -> type}

restaurantTypeID -> RestaurantType

Não viola BCNF porque todos os atributos à direita são não primos e só tem um atributo de cada lado.

Location (locationID, city, street\_name, street\_number, postal\_code)

FD = {locationID -> city, street\_name, street\_number, postal\_code}

locationID -> {city, street\_name, street\_number, postal\_code}

Não viola BCNF porque todos os atributos à direita são não primos.

Invoice (id, /total, date, demandID→Demand)

FD = {id- > /total, date, demandID}

id- > {/total, date, demandID}

Viola BCNF porque a date depende da demandID, mas não viola 3NF porque existe atributos à direita que são parte de outra key.

InvoiceLine (invoice\_lineID, quantity, Demanded, invoiceID -> Invoice)

FD = {invoice\_lineID -> quantity, Demanded, invoiceID}

invoice\_lineID -> {quantity, Demanded, invoiceID}

Viola BCNF porque a quantity depende de Demanded, mas não viola 3NF porque existe atributos à direita que são parte de outra key.