ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Trabalho Prático Final – Fase 1

48089	André Filipe Pina Páscoa
48280	André Filipe do Pilar de Jesus
48287	Nyckollas Brandão

Orientador Engenheiro Walter Vieira

Relatório de final realizado no âmbito de Sistemas de Informação,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2021/2022

Abril de 2022

Resumo

O presente trabalho tem como principal objetivo o desenvolvimento e implementação de um modelo de dados, adequado aos requisitos de um sistema.

Numa primeira fase, realizou-se a análise dos requisitos do sistema em estudo, identificando todas as entidades, atributos, relações e restrições relevantes. Com esses dados, desenvolveram-se os modelos de dados conceptual e relacional.

Numa segunda fase, implementou-se esse modelo de dados em código PL/pgSQL (Procedural Language/PostgreSQL), criando scripts autónomos para a criação, remoção e preenchimento inicial do sistema.

Numa terceira fase, foram desenvolvidos e testados múltiplos mecanismos com o objetivo de abstrair e simplificar algumas operações sobre a base de dados.

Apresenta-se também no documento, a discussão das alternativas de modelação e as razões de escolha das soluções apresentadas.

Após a realização do trabalho, concluiu-se que os objetivos de aprendizagem foram alcançados, produzindo os resultados pretendidos e conhecimentos da matéria em estudo foram adquiridos com sucesso.

Palavras-chave: base de dados dinâmica, controlo transacional, procedimentos armazenados, funções, gatilhos, vistas.

Abstract

This work has the primary goal the development and implementation of a data model that follows the requirements of a system.

In a first phase, an analysis was conducted on the system requirements, by identifying all relevant entities, attributes, relations and restrictions. With this data, the relational and conceptual data models were developed.

In a second phase, this data model was implemented in PL/pgSQL (Procedural Language/PostgreSQL) code, creating autonomous scripts for the creation, removal, and initial filling of the system.

In a third phase, multiple mechanisms were developed and tested with the goal to abstract and simplify some operations on the database.

The current document presents a discussion on the model alternatives and the reasons for the choice of the presented solutions.

After the end of this work, it's concluded that the learning goals were reached, producing the expected results and knowledge on the studied subject was acquired successfully.

Keywords: dynamic database, transactional control, stored procedures, functions, triggers, views.

Índice

Lista de Tabelas

1. Introdução

Este trabalho teve como principal objetivo o estudo e desenvolvimento de um sistema de gestão e localização de automóveis e camiões de uma empresa, denominada "OnTrack".

Inicialmente foi realizada uma análise dos requisitos do sistema, identificando as entidades relevantes, os relacionamentos entre as mesmas, os atributos e as restrições de integridade.

É de ressaltar que todos os resultados produzidos estão em língua inglesa, à exceção do presente documento escrito em português.

Após a análise dos requisitos, iniciou-se o desenvolvimento do modelo de dados conceptual, também conhecido como entidade-associação (ER). Esse modelo representa todas as entidades do sistema e como é que elas se relacionam entre si. Posteriormente fez-se a passagem desse modelo ER para um modelo relacional, aplicando corretamente as regras de passagem na unidade curricular anterior.

Com esses modelos desenvolvidos, passou-se à implementação do modelo físico, com recurso à linguagem PostgreSQL. Inicialmente foram implementados quatro scripts autónomos com as seguintes funcionalidades: criação do modelo, remoção do modelo, inserção de alguns dados iniciais e remoção dos dados do modelo.

Após isso implementaram-se os diversos mecanismos pedidos no enunciado, com recurso a procedimentos armazenados, funções, gatilhos e vistas. Estes mecanismos têm como objetivo a simplificação e abstração de algumas operações complexas sobre a base de dados.

Todo o código desenvolvido foi devidamente comentado e testado. Foi criado um script autónomo para os testes.

2. Análise dos requisitos

Começou-se por analisar o documento de requisitos do sistema com o objetivo de identificar todos os elementos relevantes do mesmo. Durante essa análise existiu discussão de alternativas de modelação do modelo, cujas razões da escolha das mesmas estão descritas neste capítulo.

Inicialmente identificou-se a entidade *Client*, que representa os clientes, e é disjunta em duas sub-entidades: *PrivateClient*, que representa os clientes privados, e *InstitutionalClient*, que representa os institucionais. Um cliente, para além dos atributos indicados nos requisitos, tem também o atributo "active", que é um booleano que representa se o cliente está ativo ou não. Este atributo foi adicionado visto que é pedido que se possa remover um utilizador do sistema sem remover a sua informação. Um cliente privado pode ter sido referenciado por outro cliente, assim foi criada a relação *Refers* para esse efeito.

É de ressaltar que, para simplificação, as entidades têm um identificador (ID) único, que é do tipo SERIAL, como chave primária.

Cada cliente tem uma frota de veículos, sendo *Vehicle* mais uma entidade relevante. Um veículo tem associadas várias zonas verdes, cada uma representada por uma entidade *GreenZone*. Cada veículo tem um condutor, representado na entidade fraca Driver. Esta entidade é fraca porque um condutor só deverá existir no sistema se tiver um veículo associado, logo um condutor é também identificado pelo identificador do veículo que conduz.

Cada veículo tem um dispositivo GPS, representado pela entidade fraca *GPSDevice*. Esta é uma entidade fraca de veículo porque só faz sentido existir um dispositivo GPS no sistema se este estiver associado a um veículo. Os dispositivos GPS têm um estado, representado pela entidade *GPSDeviceState*. Este estado é representado por uma entidade e não por um atributo, porque um dos requisitos do sistema é a possibilidade da adição de outros estados, para além dos mencionados no enunciado.

Cada dispositivo GPS envia dados, como a localização e a marca temporal dessa localização, assim foi criada a entidade *GPSData* com o propósito de representar esses dados. Os dados enviados pelo dispositivo, antes de serem armazenados na tabela *GPSData*, são armazenados em duas tabelas, representadas por duas entidades, *UnprocessedGPSData*, que contém todos os dados ainda não processados, e *InvalidGPSData*, que contém todos os dados inválidos, que permanecem no sistema até 15 dias.

Os dados do GPS podem ativar um alarme, este é representado pela entidade *Alarm*. Esta entidade é fraca de *GPSData*, porque um alarme só é ativado caso os dados indiquem que o veículo está fora da zona verde. No momento da criação do alarme, o nome do condutor atual do veículo é obtido.

A Tabela 1 apresenta as entidades identificadas e os respetivos atributos.

Tabela 1 – Entidades e respetivos atributos

Entidades	Atributos
Client	id, name, nif, phone_number, address, active
PrivateClient (RI6)	citizen_card_number
InstitutionalClient	contact_name
Vehicle	id, license_plate, num_alarms
Driver (Weak)	name, phone_number
GreenZone	id, center_location (RI3), radius
GPSDevice (Weak)	id
GPSDeviceState	id, status
GPSData	id, timestamp (RI1), location
Alarm (Weak)	driver_name
RetrievedGPSData	id, timestamp, location, device_id
UnprocessedGPSData (RI4)	-
InvalidGPSData (RI5)	-

A Tabela 2 apresenta as associações identificadas e respetivas características.

Tabela 2 - Associações, entidades presentes nas mesmas e cardinalidades

Associações	Entidades presentes	Cardinalidades
Refers	Client – PrivateClient	1:N
Has	Client - Vehicle	1:N
Has	Vehicle - GreenZone	1:N
Has	Vehicle - Driver	1:1
Has	Vehicle - GPSDevice	1:1
Has	GPSDevice - GPSDeviceState	N:1
Sends (RI2)	GPSDevice - GPSData	1:N
Triggers	GPSData - Alarm	1:1

Alguns atributos da Tabela 1 e associações da Tabela 2 têm algumas restrições associadas. Estas estão apresentadas na Tabela 3.

Tabela 3 - Restrições de Integridade e descrição das mesmas

Restrições de Integridade (RI)	Descrição
RI1	Medida com precisão ao segundo
RI2	Enviado a cada 10 segundos
RI3	Com coordenadas em graus decimais de latitude e longitude
RI4	Processamento realizado a cada 5 minutos
RI5	Permanece armazenado durante 15 dias
RI6	Limitado a um máximo de 3 veículos

Após ter-se identificado as entidades, atributos, relações e restrições de integridade, construiu-se o modelo ER, apresentado no **A.1 Modelo de dados conceptual**. Contudo existem alguns requisitos e restrições que não conseguem ser garantidos nesse modelo. Estas serão implementadas posteriormente na aplicação desenvolvida na Fase 2 do projeto.

Seguidamente, desenvolveu-se o modelo relacional, através das regras de passagem estudadas na unidade curricular precedente. O modelo está apresentado no A.2 **Modelo de dados relacional**.

3. Solução Proposta

3.1 Estrutura do projeto

A diretoria do projeto está organizada em duas subdiretorias: “data-model”, que contém os modelos de dados conceptual e relacional; e “sql” que contém todo o código PL/pgSQL desenvolvido.

Na diretoria “sql” existem duas subdiretorias, “routines”, com os procedimentos armazenados, funções e gatilhos implementados, e “views” com as vistas. Nesta diretoria também estão presentes os 5 scripts autónomos já mencionados:

- create_tables.sql – criação do modelo físico;
- remove_tables.sql – remoção do modelo físico;
- insert_data.sql – inserção de dados iniciais no modelo;
- clear_data.sql – limpeza das tabelas do modelo;
- tests.sql – testes das funcionalidades dos mecanismos implementados.

Na diretoria “routines” estão diversos ficheiros, cada um com a respetiva função ou procedimento. Cada ficheiro é o resultado de uma alínea do enunciado do trabalho.

3.2 Implementação

Este capítulo contém a descrição e explicação do código realizado nas alíneas do exercício 2 do enunciado, onde é pedido para desenvolver procedimentos armazenados, funções e gatilhos.

2.d) private_client_procedures.sql

Este ficheiro possuí os seguintes procedimentos:

- **insert_private_client**
- **remove_private_client**
- **update_private_client**

No procedimento **insert_private_client**, é feita a inserção do cliente na tabela *clients* e de seguida na tabela *private_clients*.

No procedimento **remove_private_client**, é feita a remoção do cliente da tabela private_clients e de seguida na tabela *private_clients*.

No procedimento **update_private_client**, é feita a inserção dos dados na tabela *clients* e de seguida na tabela *private_clients*.

Em todos os procedimentos, utilizou-se o nível de isolamento ***READ UNCOMMITED***. Escolheu-se este nível de isolamento, porque não foram efetuados *reads*.

2.e) get_alarms_count.sql

Nesta função, faz-se um *join* entre *gps_data* e o id do dispositivo cujo veículo tem a matrícula, obtendo os ids dos registos, se a data do registo for do ano especificado. Após isso, faz-se *join* entre *alarms* e os ids dos registos, obtendo todos os alarmes que foram feitos naquele ano e que são desse veículo. Retorna-se a contagem desses alarmes.

Se a matrícula estiver vazia, faz-se um *join* entre *alarms* e os ids dos registos cuja data é do ano especificado, obtendo todos os alarmes que foram feitos naquele ano, retornando-se a contagem desses alarmes.

Para chamar esta função, utilizou-se o nível de isolamento ***READ COMMITED***, que não permite a ocorrência de *dirty reads*. Escolheu-se este nível porque as alterações provocadas pelas funções podem causar *dirty reads* (só uma alteração efetuada antes de *commit*). Não é utilizado um nível de isolamento mais elevado porque não existe a necessidade de garantir que as *queries* são repetíveis.

2.f) process_gps_data.sql

Neste procedimento, itera-se pela tabela dos registos não processados. Cada um deles é inserido na tabela de registos processados e válidos. Se, na inserção, ocorrer uma exceção, é porque o registo falhou as restrições de integridade, logo é inválido e este é colocado na tabela de registos inválidos. Após isso, o registo é removido da tabela de registos não processados.

Para chamar este procedimento, utilizou-se o nível de isolamento ***READ UNCOMMITED***. Escolheu-se este nível, porque esta função é chamada periodicamente, e como é a única função que apaga da tabela *unprocessed_gps_data*, é garantido que as *rows* que iteramos não são alteradas.

2.g) generate_alarm_trigger.sql

Neste script é criada a função *location_inside_green_zone* que, dado o ponto central e o raio da zona verde, verifica se o ponto que representa a localização do GPS está contido na zona verde.

É criado o gatilho *generate_alarm_trigger*, chamado sempre que um novo registo é criado, e que gera um alarme se o GPS está fora de alguma das zonas verdes do dispositivo. Neste gatilho declaram-se 4 variáveis, *green_zone*, *device_status_id*, *v_device_status* e *v_driver_name*.

Faz-se um *join* entre *green_zones* e os veículos (apenas um) cujo dispositivo GPS é o do registo, obtendo todas as zonas verdes do dispositivo.

Para cada uma destas zonas verdes, obtém-se o id do estado do dispositivo ao buscar de *gps_devices* o estado do dispositivo do registo. Após isto, obtém-se o estado ao buscar de status o estado com o id. Obtém-se também o nome do condutor do veículo.

Se o estado do dispositivo não é "AlarmPause", então verifica-se se a localização está em alguma zona verde ao chamar *location_inside_green_zone*. Se não estiver, insere-se uma nova entrada em *alarms*, passando o nome do condutor do veículo, e retorna-se porque é desnecessário continuar a ver o resto das zonas verdes.

Na utilização deste gatilho, utilizou-se o nível de isolamento ***REPEATABLE READ***, que não permite a ocorrência de *dirty reads*, *non-repeatable reads* nem de *phantom reads*. Escolheu-se este nível, porque dentro do *loop* presente na função, é necessário garantir que as *queries* executados sejam repetíveis.

2.h) create_vehicle.sql

Neste script é criado o procedimento *create_vehicle*, que cria um veículo e associa-o a um cliente.

Neste procedimento, obtém-se o número de carros do cliente ao buscar da tabela *vehicles* a contagem dos veículos cujo cliente é o especificado.

O veículo só é criado se o cliente é institucional (existe uma entrada em *institutional_clients* em que o id é o id do cliente), ou tem contagem de veículos menor que 3. Se são passados como argumentos a o ponto central e o raio da zona verde, então adiciona-se uma nova zona verde em *green_zones*.

Para chamar este procedimento, utilizou-se o nível de isolamento ***SERIALIZABLE***, que não permite a ocorrência de *dirty reads*, *non-repeatable reads*, *phantom reads* nem de anomalias de serialização. Escolheu-se este nível, porque durante a execução da função, é executada uma operação de contagem sobre uma tabela, que pode originar um *phantom read*.

2.i) list_alarms.sql

Neste script, é criada uma vista que lista todos os alarmes, juntamente com informação sobre o id do dispositivo, matrícula do veículo, localização e tempo.

Faz-se um *join* entre *gps_data* e *vehicles_and_drivers*, chamando esta tabela de *gps_data_and_vehicle*. Após isso, faz-se um *join* entre *alarms* e *gps_data_and_vehicle*, obtendo todos os alarmes juntamente com o resto da informação relevante.

2.j) insert_into_list_alarms_trigger.sql

Neste script é criado o gatilho *insert_into_list_alarms_trigger* que permite, ao fazer a operação INSERT na vista *list_alarms*, inserir o registo e o alarme nas suas devidas tabelas.

Faz-se um *left join* entre as tabelas *vehicles* e drivers, de modo a obter o veículo cuja matrícula é a especificada e o nome do condutor. O uso do *left* join é para, no caso de não existir um condutor associado, as informações do veículo serem obtidas independentemente.

Insere-se um novo registo em *gps_data*.

Como o gatilho *alarm_gps_data_trigger* está em funcionamento, se realmente existirem condições para o alarme ser criado (localização fora de uma zona verde), este será criado automaticamente ao criar-se um registo.

Na utilização deste trigger, utilizou-se o nível de isolamento ***REPEATABLE READ***, que não permite a ocorrência de *dirty reads*, *non-repeatable reads* nem de *phantom reads*. Escolheu-se este nível, porque durante a execução da função é necessário garantir que a *query* que obtém o veículo seja repetível (devido a ser utilizado para uma inserção, o que poderia provocar um problema de integridade caso os dados desse veículo tivessem sido alterados).

2.k) clear_invalid_gps_data.sql

Neste script é criado um procedimento que remove todos os registos inválidos da tabela *invalid_gps_data* que existam há mais de 15 dias.

Para chamar este procedimento, utilizou-se o nível de isolamento ***READ UNCOMMITTED***. Escolheu-se este nível, porque durante a execução do procedimento não são efetuados *reads*.

2.l) delete_client_trigger.sql

Neste script é criado um gatilho chamado *delete_client_trigger*, que é executado sempre que é feita a operação DELETE na tabela *clients*, e que permite, ao invés de remover um cliente, simplesmente atualiza o seu atributo *active* para false.

Realiza-se uma operação de UPDATE em *clients*, colocando *active* a false no cliente com o id passado.

Na utilização deste gatilho, utilizou-se o nível de isolamento ***READ UNCOMMITTED***. Escolheu-se este nível, porque durante a execução do procedimento não são efetuados *reads*.

2.m) increment_vehicle_count_trigger.sql

Neste script é criado um gatilho chamado *increment_vehicle_count_trigger*, que é executado sempre que um alarme é criado, e que permite atualizar a contagem de alarmes do veículo.

Declara-se a variável *vehicle_id* que guardará o id do veículo.

Realiza-se um *join* entre *gps_data* e *vehicles*, obtendo-se o id do veículo e atribuindo-o à variável *vehicle_id*.

Realiza-se uma operação de UPDATE na tabela *vehicles*, incrementando *num_alarms* no veículo com o id *vehicle_id*.

Na utilização deste gatilho, utilizou-se o nível de isolamento ***REPEATABLE READ***, que não permite a ocorrência de *dirty reads nem de non-repeatable reads*. Escolheu-se este nível, porque durante a execução da função é necessário garantir que a *query* para obter o *vehicle* id é repetível, devido à sua utilização no *update* do número de alarmes desse veículo (veículo poderia ter sido apagado).

2.n) tests.sql

Neste script foram desenvolvidos testes para todos os scripts das alíneas 2.d) a 2.m) para garantir o seu correto funcionamento. Sempre que um teste falha, uma exceção é levantada.

4. Conclusões

Em suma podemos concluir que os objetivos definidos para a realização deste trabalho foram atingidos. Consideramos que desenvolvemos um modelo de dados adequado que cumpre com todos os requisitos do sistema estudado.

Durante a implementação, adquirimos conhecimentos relativos ao desenvolvimento de um modelo de dados, ao controlo transacional, aos níveis de isolamento, a vistas, procedimentos armazenados, funções e gatilhos. Ainda aprendemos a utilizar o software DataGrip, que foi o IDE escolhido por nós para a manutenção e desenvolvimento da base de dados.

Este projeto proporcionou-nos a oportunidade de utilizar conhecimentos que viemos a adquirir nas aulas da unidade curricular, e durante o nosso estudo autónomo.

Software utilizado

Segue-se uma lista do software utilizado na realização deste trabalho, juntamente com uma breve descrição sobre a sua utilização:

- **diagrams.net**: desenvolvimento de diagramas;
- **PostgreSQL**: linguagem utilizada para a implementação e manutenção do modelo de dados físico (base de dados);
- **DataGrip** 2021.3.4: ambiente de manutenção da base de dados e de desenvolvimento de código PL/pgSQL;
- **Git**/**GitHub**: controlo de versões e armazenamento do projeto num repositório;
- **LaTeX**: desenvolvimento do modelo de dados relacional;
- **Microsoft Word**: escrita do presente documento.

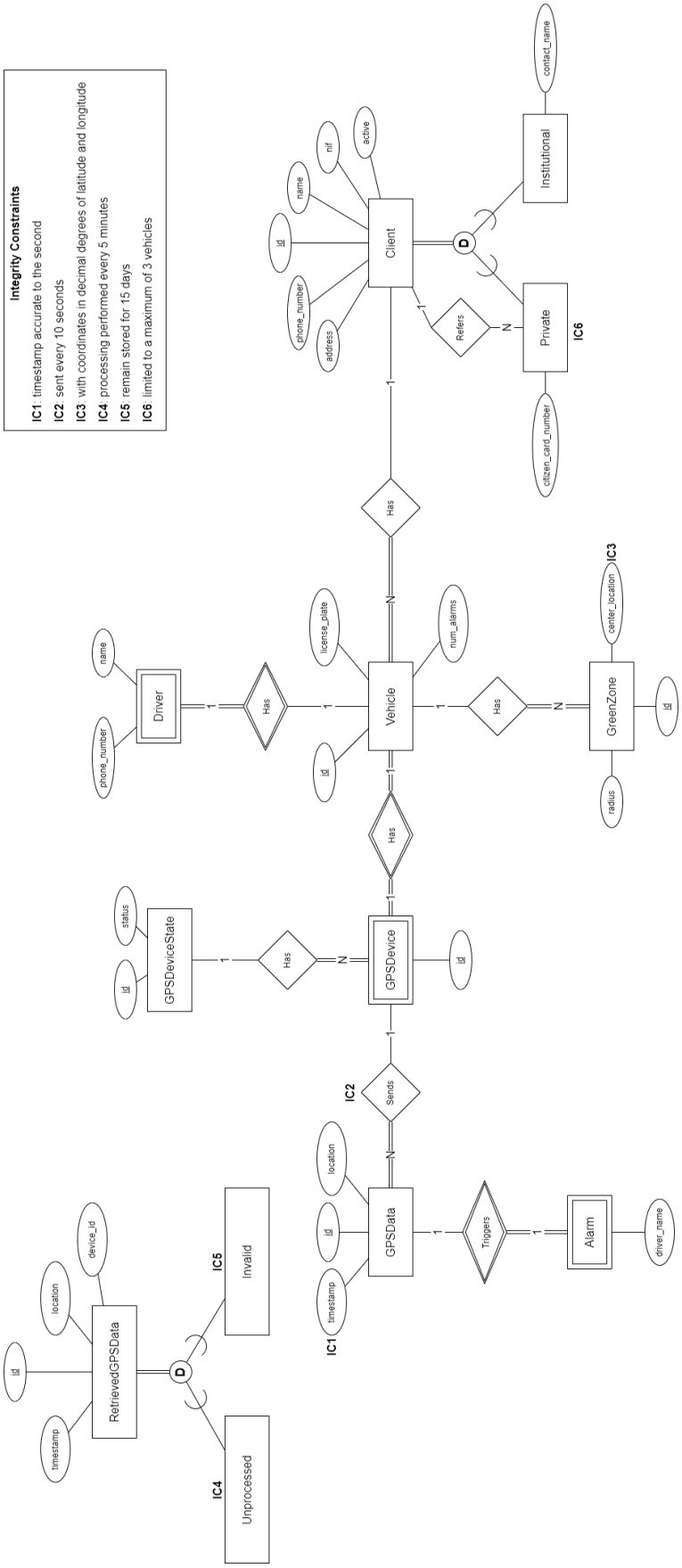
Referências

[1] Walter Vieira. *SisInf_M1_Transacções(v6)*.

[2] Walter Vieira. *SisInf_M2_SP_Trig_Func*(v2).

[3] *Documentation*. PostgreSQL. (n.d.). Retrieved April 16, 2022, from https://www.postgresql.org/docs/

A.1 Modelo de dados conceptual



A.2 Modelo de dados relacional

InstitutionalClient(id, referral, name, phone_number, nif, address, active, contact_name)
FK: { referral } de PrivateClient.id

PrivateClient(id, referral, name, phone_number, nif, address, active, citizen_card_number)
FK: { referral } de PrivateClient.id

GreenZone(id, vehicle_id, center_location, radius) FK: { vehicle_id } de Vehicle.id

Driver(vehicle_id, name, phone_number)
FK: { vehicle_id } de Vehicle.id

Vehicle(id, gps device_id, license_plate, num_alarms) FK: { gps_device_id } de GPSDevice.id

GPSDevice(id, device_status)
FK: { device_status } de GPSDeviceState

GPSDeviceState(id, status)

GPSData(id, device_id, timestamp, location)
FK: { device_id } de GPSDevice.id

Alarm(gps_data_id, driver_name)
FK: { gps_data_id } de GPSData.id

UnprocessedGPSData(id, timestamp, location, device_id)

InvalidGPSData(id, timestamp, location, device_id)