INSTITUTO DO EMPREGO
E FORMAÇÃO PROFISSIONAL

Centro de Formação Profissional de Évora

Curso: Técnico Especialista em Tecnologias e Programação de Sistemas de Informação

UFCD 5086

Projeto de SQL

Relatório

Grupo 1

Diogo Silva
Patrícia Góis
Rita Raposo
Rúben Duarte
Victor Brito

Índice

Introdução

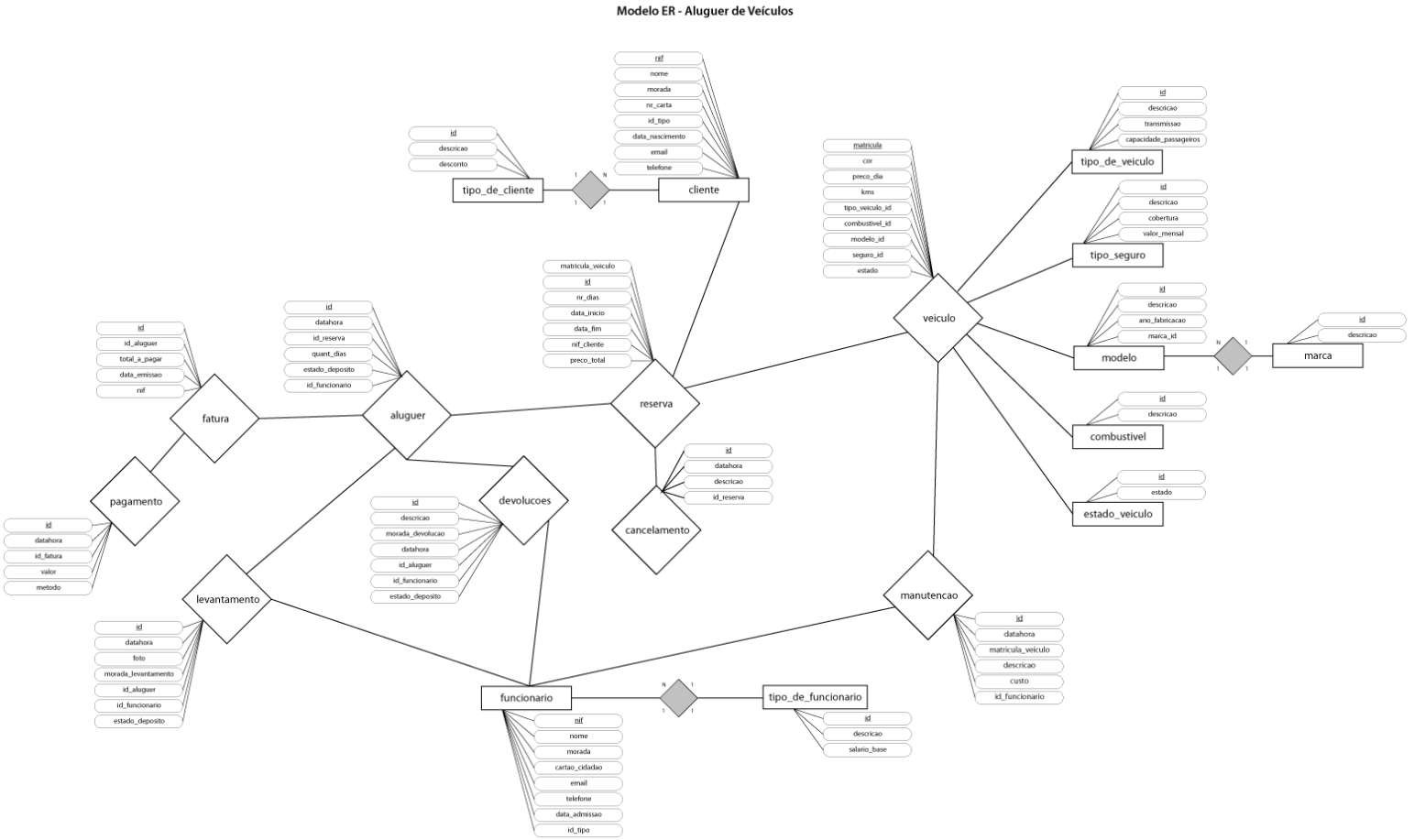
O objetivo deste projeto é desenvolver uma base de dados em SQL para gerir a frota de veículos, clientes, aluguer de veículos e funcionários administrativos de uma empresa de aluguer de veículos. A gestão eficiente desses dados é fundamental para o sucesso da empresa, pois permite uma melhor alocação de recursos, redução de custos e melhoria na satisfação dos clientes.

Para alcançar o objetivo, elaboramos um **modelo ER (Entidade-Relação)** e um **modelo relacional** que descrevem as entidades envolvidas no negócio e como se relacionam entre si. O modelo ER permite identificar as principais entidades, atributos e relações, enquanto o modelo relacional permite definir as tabelas, campos, chaves primárias e estrangeiras necessárias para armazenar e gerir os dados.

Além disso, também foi considerada a necessidade de **validar o veículo no ato de levantamento e devolução**, garantindo que os dados sejam precisos e confiáveis. Para atender a essa necessidade, desenvolvemos um mecanismo que permite ao utilizador registar o estado do veículo nos atos de levantamento e entrega, proporcionando flexibilidade e controlo sobre a gestão da frota.

Com a base de dados implementada em HeidiDB, Sequel Pro e phpMyAdmin, pretendemos fornecer uma ferramenta eficaz e fácil de usar para a gestão da empresa de aluguer de veículos, permitindo que os funcionários administrativos consigam gerir os dados de forma eficiente e precisa, e que os clientes recebam um serviço de alta qualidade.

Modelo ER



(Disponível na sua totalidade nos Anexos)

Modelo Relacional

tipo_de_cliente (id (PK), descricao, desconto);

tipo_de_funcionario (id, descricao, salario_base);

cliente (nif (PK),nome, morada, nr_carta, id_tipo (FK), data_nasciment, email, telefone);

marca (id (PK), descricao);

modelo (id (PK), descricao, ano_fabricacao, marca_id (FK));

tipo_de_veiculo (id (PK), descricao, transmissao, capacidade_passageiros);

combustivel (id (PK), descricao);

tipo_seguro (id (PK), descricao, cobertura, valor_mensal);

estado_veiculo (id (PK), estado);

veiculo (matricula (PK), cor, preco_dia, kms, tipo_veiculo_id (FK), combustivel_id (FK), modelo_id (FK), seguro_id (FK), estado (FK));

funcionario (nif (PK), nome, morada, cartao_cidadao, email, telefone, data_admissao, id_tipo (FK));

reserva (matricula_veiculo (FK), id (PK), nr_dias, data_inicio, data_fim, nif_cliente (FK), preco_total);

aluguer (id (PK), datahora, id_reserva (FK), quant_dias, estado_deposito, id_funcionario (FK));

fatura (id (PK), id_aluguer (FK), total_a_paga, data_emissao, nif);

cancelamento (id (PK), datahora , descricao , id_reserva (FK));

devolucoes (id (PK), descricao, morada_devolucao, datahora, id_aluguer (FK), id_funcionario (FK), estado_deposito);

levantamento (id (PK), datahora, foto, morada_levantamento, id_aluguer (FK), id_funcionario (FK), estado_deposito);

manutencao (id (PK), datahora, matricula_veiculo, descricao, custo, id_funcionario);

pagamento (id (PK), datahora, id_fatura (FK), valor, metodo);

Perguntas

1 Pergunta só com critérios (uma tabela)

Qual a morada do funcionário Miguel Carvalho?

```
SELECT morada
FROM funcionario
WHERE nome like 'Miguel Carvalho';
```

Output:
morada
Rua M, 707

Justificação:
Esta consulta mostra a coluna morada da tabela funcionario para o funcionário cujo nome é exatamente 'Miguel Carvalho'.
O operador LIKE é utilizado aqui como um operador de igualdade.

1 Pergunta só com critérios (N tabelas)

Qual é o salário do funcionário João Martins, que tem a função de mecânico?

```
SELECT funcionario.nome, tipo_de_funcionario.salario_base
FROM funcionario, tipo_de_funcionario
WHERE funcionario.id_tipo = tipo_de_funcionario.id
AND funcionario.nome = 'João Martins' AND tipo_de_funcionario.descricao =
'mecanico';
```

Output:
nome;salario_base
João Martins;1800

Justificação:
Esta consulta faz uma junção entre as tabelas funcionario e tipo_de_funcionario, com base na correspondência entre id_tipo e id.
Filtra os resultados para mostrar apenas o salário do funcionário chamado 'João Martins' e para o tipo de funcionário cuja descrição é 'mecânico'.

1 Pergunta com GROUP BY

Mostre o numero de reserva para cada veiculo

```
SELECT matricula_veiculo, COUNT(*) as numero_de_reservas
FROM reserva
GROUP BY matricula_veiculo;
```

Output:

```
matricula_veiculo;numero_de_reservas
AA-55-33;1
FF-11-22;1
GG-22-33;1
HH-33-44;1
II-44-55;1
JJ-55-66;1
UU-67-89;1
VV-78-90;1
WW-89-01;1
```

Justificação:
Esta consulta agrupa os registo da tabela reserva por matricula_veiculo e conta quantas vezes cada veículo foi reservado.

1 Pergunta com GROUP BY (com critérios)

Mostre o número de reserva para cada veículo de cor preta.

```
SELECT matricula_veiculo, COUNT(*) as numero_de_reservas
FROM reserva, veiculo
WHERE reserva.matricula_veiculo = veiculo.matricula
AND veiculo.cor LIKE "Preto"
GROUP BY matricula_veiculo;
```

Output:

```
matricula_veiculo;numero_de_reservas
AA-55-33;1
FF-11-22;1
UU-67-89;1
VV-78-90;1
WW-89-01;1
```

Justificação:

Esta consulta junta as tabelas reserva e veiculo para contar o número de reservas por veículo, mas apenas para veículos que têm a cor preta.

1 Pergunta com NOT IN

Mostre os veiculos que nunca foram reservados

```
SELECT veiculo.matricula
FROM veiculo
WHERE veiculo.matricula NOT IN (
   SELECT matricula_veiculo
   FROM reserva
)
GROUP BY veiculo.matricula;
```

Output:
matricula
KK-66-77
LL-77-88
MM-88-99
NN-99-00
OO-00-11
PP-12-34
QQ-23-45
RR-34-56
SS-45-67
TT-56-78

Justificação:
Seleciona os veículos cujas matrículas não aparecem na tabela reserva.
Usa NOT IN para garantir que apenas veículos que não foram reservados sejam listados.

1 Pergunta com Subquery

Mostre o cliente que mais pagou. Mostre o nome e o nif

```
SELECT cliente.nome, SUM(reserva.preco_total) AS total_pago
FROM cliente, reserva, aluguer
WHERE cliente.nif = reserva.nif_cliente AND
reserva.id = aluguer.id_reserva
GROUP BY cliente.nome
HAVING SUM(reserva.preco_total) = (
   SELECT MAX(total_pago)
   FROM (
      SELECT cliente.nome, SUM(reserva.preco_total) AS total_pago
      FROM cliente, reserva, aluguer
      WHERE cliente.nif = reserva.nif_cliente AND
                  reserva.id = aluguer.id_reserva
      GROUP BY cliente.nome
   ) AS temp
);
```

Output:
nome;total_pago
Sara Martins;1400

Justificação:
Calcula o total pago por cada cliente e exibe apenas o cliente com o maior total pago.
Utiliza uma subconsulta para encontrar o valor máximo do total pago.

1 Pergunta com cálculo de máximo/mínimo

Qual o valor maximo e minimo de fatura já emitida

SELECT MAX(total_a_pagar) as valor_maximo , MIN(total_a_pagar) as valor_minimo
FROM fatura;

Output:
valor_maximo;valor_minimo
1400;85

Justificação:
Calcula o total pago por cada cliente e exibe apenas o cliente com o maior total pago.
Utiliza uma subconsulta para encontrar o valor máximo e mínimo do total pago.

1 Pergunta com UNION

Mostrar uma lista de todos os veículos e o número de reservas respectivo incluindo aqueles que nunca foram reservados.

```
SELECT veiculo.matricula as veiculo_id, COUNT(reserva.id) as
numero_de_reservas
FROM veiculo
LEFT JOIN reserva ON veiculo.matricula = reserva.matricula_veiculo
GROUP BY veiculo.matricula
UNION
SELECT matricula as veiculo_id, 0 as numero_de_reservas
FROM veiculo
WHERE matricula NOT IN (SELECT matricula_veiculo FROM reserva);
```

Output:

```
veiculo_id;numero_de_reservas
AA-55-33;1
FF-11-22;1
GG-22-33;1
HH-33-44;1
II-44-55;1
JJ-55-66;1
KK-66-77;0
LL-77-88;0
MM-88-99;0
NN-99-00;0
OO-00-11;0
PP-12-34;0
QQ-23-45;0
RR-34-56;0
SS-45-67;0
TT-56-78;0
UU-67-89;1
VV-78-90;1
WW-89-01;1
```

Justificação:
Utiliza LEFT JOIN para contar o número de reservas por veículo e combina com uma consulta UNION para incluir também veículos que não têm reservas (com contagem de 0).

1 Pergunta com IF

Para cada cliente, mostrar a quantidade de cancelamento com, SE tiver 3 ou mais

---

```
SELECT nif_cliente, if(COUNT(*) >= 3, COUNT(*) , "Nao tem 3") as
quantidade_de_cancelamentos
FROM reserva, cancelamento
WHERE cancelamento.id_reserva = reserva.id
GROUP BY nif_cliente;
```

Output:

```
nif_cliente;quantidade_de_cancelamentos
112233445;Nao tem 3
```

Justificação:
Conta o número de cancelamentos por cliente e usa IF para mostrar a contagem apenas se for 3 ou mais. Caso contrário, exibe 'Nao tem 3'.

1 Pergunta com subquery + cálculo

Qual o valor total gasto por cada cliente em reservas, apenas considerando os veículos que possuem mais de 500 quilômetros rodados?

```
SELECT reserva.nif_cliente, SUM(reserva.preco_total) as total_gasto
FROM reserva, veiculo
WHERE reserva.matricula_veiculo = veiculo.matricula
AND veiculo.kms > 500
GROUP BY reserva.nif_cliente;
```

Output:

```
nif_cliente;total_gasto
112233445;550
223344557;1040
332211446;945
334455668;85
445566779;570
556677889;400
667788990;1400
778899002;280
998877664;375
```

Justificação:

Calcula o total gasto por cliente para reservas associadas a veículos que têm mais de 500 km.

1 View com aplicação prática funcional para o enunciado proposto

Mostrar os veículos disponíveis (visao cliente, com informação tratada)

CREATE VIEW veiculos_disponiveis AS
SELECT veiculo.matricula AS matricula, modelo.descricao AS modelo, marca.descricao AS marca, veiculo.cor AS cor, veiculo.kms AS kms
FROM veículo, modelo, marca
WHERE veiculo.modelo_id = modelo.id
AND modelo.marca_id = marca.id
AND veiculo.estado = 1;

Output:
matricula;modelo;marca;cor;kms
FF-11-22;Corolla;Toyota;Preto;27000
KK-66-77;Camaro;Chevrolet;Amarelo;32000
LL-77-88;Elantra;Hyundai;Verde;21000
MM-88-99;Corolla;Toyota;Laranja;33000
NN-99-00;Camaro;Chevrolet;Roxo;25000
OO-00-11;Elantra;Hyundai;Bege;28000
PP-12-34;Golf;Volkswagen;Cinza;26000
QQ-23-45;Altima;Nissan;Azul;22000
RR-34-56;Camaro;Chevrolet;Branco;31000
SS-45-67;Elantra;Hyundai;Preto;30000
TT-56-78;C-Class;Mercedes-Benz;Verde;25000
UU-67-89;Corolla;Toyota;Preto;24000

Justificação:
Cria uma visão chamada veiculos_disponiveis para listar veículos disponíveis (com estado = 1 "Disponível"), incluindo as descrições de modelo e marca.

1 Trigger com aplicação prática funcional para o enunciado proposto

Fatura.

```
DELIMITER $$
CREATE TRIGGER auto_fatura
AFTER INSERT ON aluguer
FOR EACH ROW
BEGIN
   DECLARE montante_fatura DOUBLE;
   DECLARE contribuinte INT;
   DECLARE desconto DOUBLE;

   SET montante_fatura = (SELECT preco_total FROM reserva WHERE id =
NEW.id_reserva);
   SET contribuinte = (SELECT nif_cliente FROM reserva WHERE id =
NEW.id_reserva);
   SET desconto = (SELECT tipo_de_cliente.desconto
            FROM cliente
            JOIN tipo_de_cliente ON cliente.id_tipo = tipo_de_cliente.id
            WHERE cliente.nif = contribuinte);

   SET montante_fatura = montante_fatura - (montante_fatura * desconto);

   INSERT INTO fatura (id_aluguer, total_a_pagar, nif)
   VALUES (NEW.id, montante_fatura, contribuinte);
END$$
DELIMITER ;
```

Justificação:
Cria um trigger que insere automaticamente uma fatura na tabela fatura sempre que um novo registo é inserido na tabela aluguer.
Recupera o total_a_pagar e o nif_cliente da tabela reserva para preencher a nova fatura.
Relaciona e aplica o desconto do respectivo tipo de cliente.

UPDATE

Estado do veículo

```
UPDATE veiculo SET estado=1 WHERE matricula = "AA-55-33"
```

Justificação:
Atualiza o estado do veículo (constante o número inteiro para “estado”).

TRIGGERS

after_reserva_insert :

```
DELIMITER $$
CREATE TRIGGER after_reserva_insert
AFTER INSERT ON reserva
FOR EACH ROW
BEGIN
   UPDATE veiculo
   SET estado = 4
   WHERE matricula = NEW.matricula_veiculo;
END$$
DELIMITER ;
```

Justificação:
Após a reserva ter sido efetuada, o estado do veículo é atualizado para “Reservado”.

auto_fatura:

```
DELIMITER $$
CREATE TRIGGER auto_fatura
AFTER INSERT ON aluguer
FOR EACH ROW
BEGIN
   DECLARE montante_fatura DOUBLE;
   DECLARE contribuinte INT;

   SET montante_fatura = (SELECT preco_total FROM reserva WHERE NEW.id_reserva =
reserva.id);
   SET contribuinte = (SELECT nif_cliente FROM reserva WHERE NEW.id_reserva =
reserva.id);

   INSERT INTO fatura (id_aluguer, total_a_pagar, nif)
   VALUES (NEW.id, montante_fatura, contribuinte);
END$$
DELIMITER ;
```

Justificação:
Após o aluguer ter sido efetuado, existe um mecanismo de emissão de fatura automática. Aplica o desconto do tipo de cliente.

after_cancelamento_insert:

```
DELIMITER $$
CREATE TRIGGER after_cancelamento_insert
AFTER INSERT ON cancelamento
FOR EACH ROW
BEGIN
   DECLARE v_matricula VARCHAR(8);

   SET v_matricula = (SELECT matricula_veiculo
                      FROM reserva
                      WHERE id = NEW.id_reserva);

   UPDATE veiculo
   SET estado = 1
   WHERE matricula = v_matricula;
END$$
DELIMITER ;
```

Justificação:
Após o cancelamento ter sido efetuado, o estado do veículo é atualizado para “Disponível”.

after_devolucao_insert:

```
DELIMITER $$
CREATE TRIGGER after_devolucao_insert
AFTER INSERT ON devolucoes
FOR EACH ROW
BEGIN
   DECLARE v_matricula VARCHAR(8);

    SET v_matricula = (SELECT matricula_veiculo
                      FROM reserva
                      WHERE id = (SELECT id_reserva
                                  FROM aluguer
                                  WHERE id = NEW.id_aluguer));

   UPDATE veiculo
   SET estado = 1
   WHERE matricula = v_matricula;
END$$
DELIMITER ;
```

Justificação:
Após a devolução ter sido efetuada, o estado do veículo é atualizado para “Disponível”.

after_levantamento_insert:

```
DELIMITER $$
CREATE TRIGGER after_levantamento_insert
AFTER INSERT ON levantamento
FOR EACH ROW
BEGIN
   DECLARE v_matricula VARCHAR(8);

    SET v_matricula = (SELECT matricula_veiculo
                FROM reserva
                WHERE id = (SELECT id_reserva
                        FROM aluguer
                        WHERE id = NEW.id_aluguer));

   UPDATE veiculo
   SET estado = 2
   WHERE matricula = v_matricula;
END$$
DELIMITER ;
```

Justificação:
Após o levantamento ter sido efetuado, o estado do veículo é atualizado para “Alugado”.

after_manutencao_insert:

```
DELIMITER $$
CREATE TRIGGER after_manutencao_insert
AFTER INSERT ON manutencao
FOR EACH ROW
BEGIN
   UPDATE veiculo
   SET estado = 3
   WHERE matricula = NEW.matricula_veiculo;
END$$
DELIMITER ;
```

Justificação:
Após a entrada de um veículo para manutenção, o estado do veículo é atualizado para “Em Manutenção”.

VIEWS

vw_aluguer_detalhes:

```
CREATE VIEW vw_aluguer_detalhes AS
SELECT
    aluguer.id,
    aluguer.datahora,
    aluguer.id_funcionario,
    funcionario.nome AS nomefuncionario,
    cliente.nome AS nomecliente
FROM aluguer
JOIN funcionario ON aluguer.id_funcionario = funcionario.nif
JOIN reserva ON aluguer.id_reserva = reserva.id
JOIN cliente ON reserva.nif_cliente = cliente.nif;
```

---

veiculos_disponiveis :

```
CREATE VIEW veiculos_disponiveis AS
SELECT
    veiculo.matricula AS matricula,
    modelo.descricao AS modelo,
    marca.descricao AS marca,
    veiculo.cor AS cor,
    veiculo.kms AS kms
FROM veiculo, modelo, marca
WHERE veiculo.modelo_id = modelo.id
AND modelo.marca_id = marca.id
AND veiculo.estado = 1;
```

Guia do Modelo

Entidades Principais e Relacionamentos

Cliente: Cada cliente possui um identificador único, nome, morada, dados de contacto e, possivelmente, um histórico de alugueres.
Veículo: Cada veículo possui uma matrícula única, modelo, marca, tipo de combustível, tipo de seguro e um histórico de manutenções e alugueres.
Aluguer: Representa um evento de aluguer, ligando um cliente a um veículo específico por um determinado período. Inclui informações sobre data de início e fim, valor total, forma de pagamento e outros detalhes.
Pagamento: Registra os pagamentos realizados pelos clientes, associando-os a um aluguer específico.
Manutenção: Registra as manutenções realizadas nos veículos, incluindo data, descrição e custo.
Funcionário: Representa os funcionários da empresa, com informações como nome, morada, contacto e função.
Reserva: Representa uma reserva de um veículo por um cliente, antes do aluguer efetivo.

Relacionamentos

Um cliente pode fazer muitos alugueres.
Um veículo pode ser alugado muitas vezes.
Um aluguer envolve um cliente e um veículo.
Um pagamento está associado a um aluguer.
Um aluguer está associado a uma reserva.
Uma manutenção está associada a um veículo.
Tanto o funcionário como o cliente podem realizar muitos alugueres, mas possuem características diferentes.
Uma reserva pode levar a um aluguer ou a um cancelamento.
Uma fatura é emitida após um aluguer ser adicionado via *trigger*.
Cada entidade possui uma chave primária (geralmente um campo "id") que a identifica de forma única.

Automatização

Os triggers fornecidos automatizam diversas tarefas relacionadas ao sistema, garantindo a integridade dos dados e a consistência do estado dos veículos.

Descrição Detalhada dos Triggers

1. after_reserva_insert
Objetivo: Atualizar o estado do veículo para "Reservado" quando uma nova reserva é criada.
Funcionamento:
Ao inserir uma nova reserva, o trigger identifica o veículo envolvido.
Altera o estado do veículo na tabela veículo para "Reservado", indicando que ele não está disponível para outros alugueres.

2. auto_fatura
Objetivo: Criar automaticamente uma fatura ao finalizar um aluguer.
Funcionamento:
Após a inserção de um novo registo na tabela aluguer, o trigger calcula o valor total do aluguer com base na reserva correspondente.
Insere uma nova linha na tabela fatura, registando o aluguer , o valor total e o número de contribuinte do cliente.

3. after_cancelamento_insert
Objetivo: Alterar o estado do veículo para novos alugueres quando uma reserva é cancelada.
Funcionamento:
Ao registar um cancelamento, o trigger identifica o veículo associado à reserva cancelada.
Altera o estado do veículo na tabela veiculo para "Para Alugar", indicando que ele está disponível para novas reservas.

4. after_devolucao_insert
Objetivo: Alterar o estado do o veículo para novos alugueres após a devolução.
Funcionamento:
Após o registo de uma devolução, o trigger identifica o veículo associado ao aluguer devolvido.
Altera o estado do veículo na tabela veiculo para "Para Alugar", disponibilizando-o para novas reservas.

5. after_levantamento_insert
Objetivo: Alterar o estado do o veículo para novos alugueres após o levantamento.
Funcionamento:
Similar ao trigger after_devolucao_insert, o trigger liberta o veículo para novos alugueres após o registo de um levantamento.

6. after_manutencao_insert
Objetivo: Indicar que um veículo está em manutenção.
Funcionamento:
Ao registar uma nova manutenção, o trigger altera o estado do veículo na tabela veiculo para "Em Manutenção".

Isso impede que o veículo seja reservado ou alugado enquanto estiver em manutenção.

Justificação de Decisões

Escolha dos eventos triggers

Após inserção: A escolha de acionar os triggers após a inserção de novos registos nas tabelas reserva, aluguer, cancelamento, devoluções, levantamento e manutenção garante que as ações de atualização de estado do veículo sejam executadas imediatamente após a ocorrência de um evento relevante.
Justificação: Essa decisão assegura a consistência dos dados em tempo real, evitando discrepâncias entre o estado de um veículo e as informações registadas nas outras tabelas.

Atualização do estado do veículo

Centralização da lógica: A decisão de centralizar a lógica de atualização do estado do veículo nos triggers simplifica a manutenção do sistema. Ao ocorrer uma alteração no fluxo de trabalho, é necessário modificar apenas os triggers, em vez de procurar e alterar o código em diversos pontos do sistema.
Justificação: Essa centralização facilita a compreensão do sistema e reduz o risco de erros.

Criação automática de faturas

Automatização de processos: A criação automática de faturas após a finalização de um aluguer elimina a necessidade de intervenção manual, reduzindo o tempo e os erros.
Justificação: A automatização desse processo aumenta a eficiência e a precisão do sistema.

Uso de variáveis declaradas

Melhoria da legibilidade: A utilização de variáveis declaradas para armazenar valores intermediários torna o código mais legível e fácil de entender.
Justificação: Variáveis bem nomeadas facilitam a depuração e a manutenção do código.

Consulta de dados relacionados

Relacionamento entre tabelas: Os triggers consultam as tabelas reserva e aluguer para obter informações relevantes sobre o veículo e o cliente.
Justificação: Essa consulta permite estabelecer a relação entre as diferentes entidades do sistema e garantir a consistência dos dados.

Importância das decisões

Garantia da integridade dos dados: As decisões tomadas garantem que os dados do sistema estejam sempre consistentes e atualizados, evitando dados duplicados e informações incorretas.
Automatização de processos: A automatização de tarefas como a criação de faturas e a atualização do estado dos veículos aumenta a eficiência e reduz a carga de trabalho manual.
Facilidade de manutenção: A centralização da lógica e a utilização de variáveis bem nomeadas facilitam a compreensão e a manutenção do sistema.
Flexibilidade: Os triggers podem ser facilmente adaptados para atender a novas necessidades do sistema, como a inclusão de novos tipos de eventos ou a alteração do fluxo de trabalho.

Conclusão

A base de dados apresentada é uma representação robusta e detalhada de um sistema de aluguer de veículos, com várias entidades inter-relacionadas que cobrem diferentes aspectos do processo de aluguer. A estrutura da base de dados demonstra uma atenção cuidadosa aos detalhes, assegurando que todas as etapas do aluguer, desde a reserva e levantamento até à devolução e faturação, sejam devidamente registadas e acompanhadas.

As tabelas de suporte, como **tipo de veículo**, **tipo de cliente** e **tipo de seguro**, permitem uma flexibilidade na definição de diferentes parâmetros e características dos veículos e clientes, o que facilita a adaptação do sistema a diferentes necessidades operacionais.

Adicionalmente, a inclusão de tabelas como **manutenção**, **combustível** e **estado do veículo** evidencia a preocupação em manter um controlo rigoroso sobre a condição e a disponibilidade dos veículos, garantindo assim a satisfação do cliente e a eficiência operacional. A tabela de funcionários e respetivas funções também são bem definidas, permitindo um acompanhamento claro das responsabilidades e atividades de cada funcionário envolvido no processo de aluguer, o que é crucial para uma operação suave e eficiente. Este nível de detalhe reflete uma compreensão profunda das necessidades operacionais de uma empresa de aluguer de veículos.

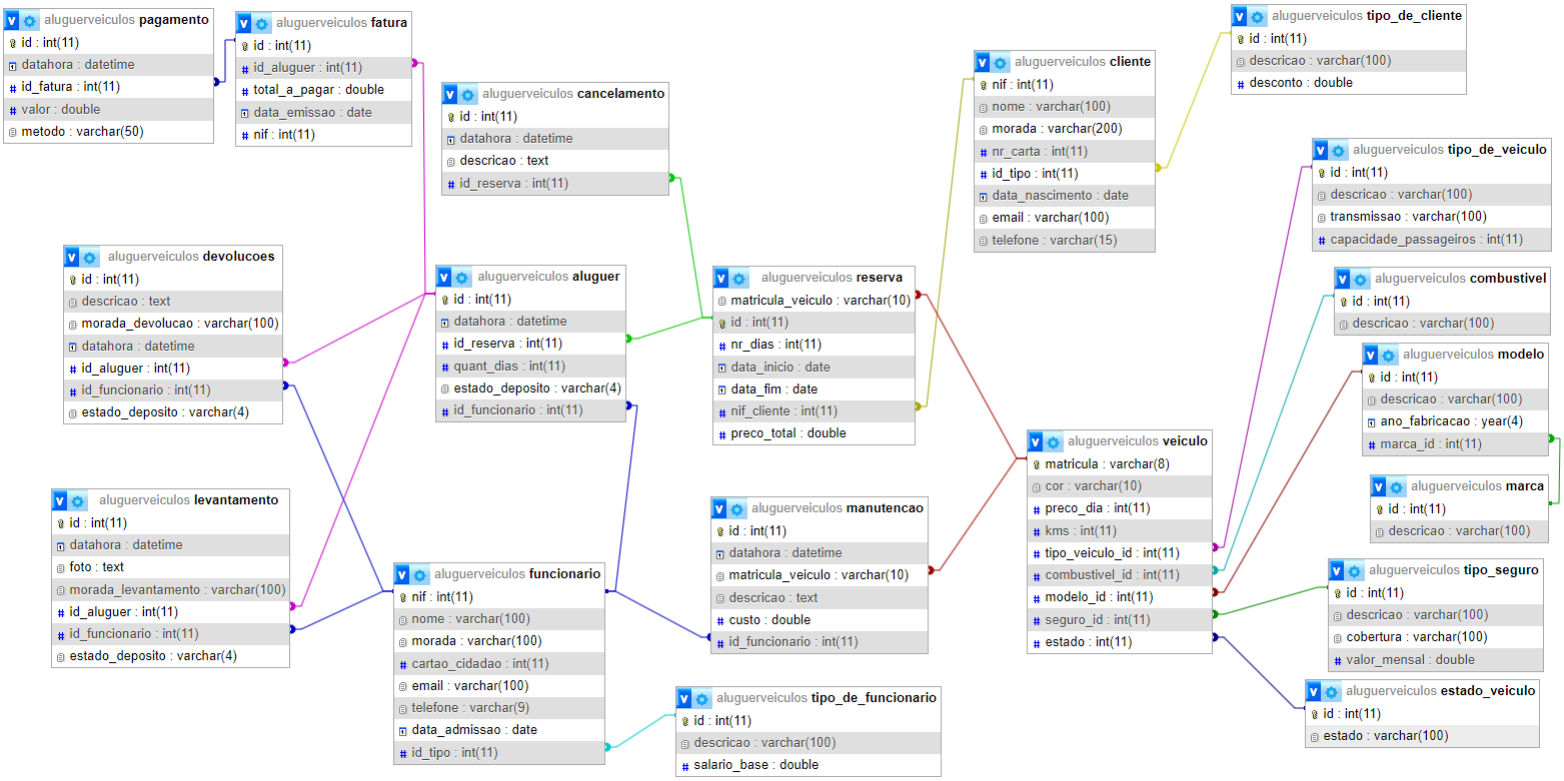
Para concluir, a base de dados desenvolvida proporciona uma solução abrangente e bem estruturada para a **gestão de um sistema de aluguer de veículos**. A interligação lógica entre as tabelas e a consideração de vários aspectos operacionais demonstram uma abordagem meticulosa e bem planeada. Esta base de dados não facilita apenas o acompanhamento detalhado de cada processo de aluguer, mas também assegura que todas as **informações relevantes** sejam facilmente acessíveis e geríveis, promovendo assim a eficiência, precisão e qualidade no serviço oferecido pela empresa de aluguer de veículos.

Para o futuro, a base de dados pode saber responder a:

- **Anulação de faturas**, permitir que o cliente saiba **quanto tempo de aluguer ainda tem** (caso não tenha usufruído na totalidade do aluguer).
- Apenas permitir a **devolução** se o estado do depósito seja igual ao estado do depósito do levantamento.
- Poderia também ser disponibilizada uma interface de **serviços promocionais**, aplicando o desconto no tipo de cliente (por exemplo).

Anexos

phpMyAdmin desenhador:



Modelo ER - Aluguer de Veículos