Criação e Testes de Bancos de Dados SQL e NoSQL para um Sistema de Monitoramento para Manutenção Preditiva de Veículos Autônomos

Richard Coelho Gomes

Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Leopoldina – MG – Brazil

richard.gomes@aluno.cefetmg.br

Abstract. *This paper compares the efficiency between SQL (MySQL) and NoSQL (MongoDB) databases in a monitoring and preventive maintenance scenario for autonomous vehicles, typical of the Internet of Things (IoT). CRUD operations were simulated on a voluminous telemetry and sensor diagnostics database to evaluate performance. Results show MongoDB's superiority, with an average advantage of approximately 6 seconds in the total processing time. The NoSQL architecture proved more efficient and scalable because its document-oriented modeling stores composite data, eliminating the cost of joins and the need for normalization required by MySQL for voluminous and semi-structured IoT data.*

Resumo. *Este artigo compara a eficiência entre bancos de dados SQL (MySQL) e NoSQL (MongoDB) em um cenário de monitoramento e manutenção preventiva de veículos autônomos, típico de Internet das Coisas (IoT). Simularam-se operações CRUD em uma base volumosa de telemetria e diagnóstico de sensores para avaliar o desempenho. Os resultados mostram que o MongoDB é superior, com uma vantagem média de 6 segundos no tempo total de processamento. A arquitetura NoSQL provou-se mais eficiente e escalável, pois sua modelagem orientada a documentos armazena dados compostos, eliminando o custo de joins e a necessidade de normalização exigida pelo MySQL para dados volumosos e semi-estruturados da IoT.*

1. Introdução

A presença generalizada dos veículos no cotidiano global, essenciais para o transporte de pessoas, cargas e eventos esportivos, define um setor de alta relevância, com uma taxa de produção de mais de 93 milhões de veículos motorizados por ano (ACEA, 2024). Com o grande avanço tecnológico que está ocorrendo no século 21 (SALES; BEZERRA, 2018), surgiram os veículos autônomos, veículos que não precisam de um ser humano para realizar suas funções e que a cada dia representam uma maior fatia dos veículos produzidos no mundo (WORLD ECONOMIC FORUM, 2025). Essa categoria de veículo está em grande evidência no setor de carros de passeio, também tendo forte uso na criação de serviços de transporte autônomos com robotaxis (WORLD ECONOMIC FORUM, 2025), evidenciando a busca por conforto e segurança nos meios de transporte no mundo atual. Esses veículos são controlados a partir de vários sensores, gerando um alto volume de dados em tempo real (Big Data), predominantes do crescimento da área de Internet das Coisas (IoT), sendo eles para questões complexas como visão espacial, ou até mesmo sensores mais comuns, como sensores para medição de velocidade e nível de combustível (AYALA; MOHD, 2021), evidenciando a importância do pleno funcionamento desses componentes, assim como das outras partes

dos veículos. Dessa forma, é posto em evidência a importância da realização de manutenção preditiva, principalmente nesse tipo de veículo (RAJ; SHARMA, 2024). Com essa premissa, este artigo pretende simular os bancos de dados para um sistema de monitoramento para manutenção preditiva de veículos autônomos, com a proposta de fazer uma análise comparativa a respeito da eficiência dos bancos de dados SQL e NoSQL nesse cenário de IoT. Este artigo conta com as sessões de Introdução, Cenário da aplicação, Metodologia, Bancos de Dados Criados, Resultados e Conclusão.

2. Cenário da aplicação

O sistema proposto concentra-se na coleta contínua e volumosa de dados provenientes de sensores variados instalados em veículos autônomos. Estes sensores incluem câmeras para visão e localização espacial, receptores para localização geográfica do veículo, e componentes dedicados ao controle de velocidade, direção e freios. A natureza crítica desses componentes, aliada à alta frequência de leitura, estabelece um cenário clássico de Internet das Coisas (IoT).

Visando a manutenção preventiva, o sistema fica a cargo das leituras de integridade dos sensores, obtendo informações de status (OK, Degradado, Falha), assim como dados de temperatura, taxa de erros e código de falha no momento da leitura de telemetria do veículo. Os dados coletados geram valor imediato para proprietários e gestores de frotas, oferecendo alertas de manutenção, e também para fabricantes e engenheiros, fornecendo informações de desempenho para aprimoramento. Sendo o maior foco do projeto a aplicação dos bancos de dados, segue um repositório na plataforma Github com os arquivos de modelagem e os necessários para a execução dos testes comparativos: https://github.com/RechargeCG/projeto-banco-de-dados-iot.

3. Metodologia

Para a realização do estudo comparativo, foram criados bancos de dados locais de natureza NoSQL, utilizando o MongoDB, e de natureza SQL, utilizando o MySQL. Para a execução do estudo, foi desenvolvido um script na linguagem Python para medir o tempo de execução de cada operação, onde o próprio script, assim como os servidores dos bancos de dados, foram executados localmente para consistência de performance e temporal. Foram avaliadas as operações CRUD (*Create, Read, Update, Delete*): inserção, seleção (busca), atualização e exclusão de dados. Adicionalmente, foram criadas perguntas específicas a serem respondidas, com o objetivo de demonstrar a capacidade do sistema em gerar *knowledge* (conhecimento) para o contexto veicular, partindo de consultas que buscam *information* para análises mais complexas. O script em Python utilizará as bibliotecas pymongo[1] para a manipulação de dados no MongoDB, PyMySQL[2] para a manipulação de dados no MySQL, e Faker[3] para a criação de dados sintéticos que simularão a utilização dos bancos de dados.

[1] pymongo: https://pypi.org/project/pymongo/
[2] PyMySQL: https://pypi.org/project/PyMySQL/
[3] Faker: https://pypi.org/project/Faker/

4. Bancos de Dados Criados

Para a realização do estudo comparativo, os bancos deverão permitir a inserção e manipulação de dados como os apresentados na Tabela 1 a seguir, de forma que nem todos serão obrigatórios, que será representado pelos tipos diferentes de propulsão dos veículos, podendo ser elétricos, a combustão ou híbridos, determinando a existência de valores para um atributo de carga da bateria ou nível de combustível.

Tabela 1. Dados possíveis na simulação dos bancos de dados

Grupo	Atributo	Descrição
Veículo	Marca	Fabricante do veículo.
	Modelo	Modelo específico do ativo.
	Data de Fabricação	Data de produção.
	Identificador Único (VIN)	Número de identificação permanente do veículo.
	Nível de Autonomia	Grau de capacidade de condução autônoma (Níveis SAE 3 a 5).
	Tipo de Propulsão	Mecanismo de força (Ex: Elétrico, Combustão, Híbrido).
Configuração dos Sensores (Parte do Veículo)	Tipo de Sensor	Função específica do componente (Ex: LiDAR 360, Câmera Estéreo).
	Versão do Hardware	Código de identificação da versão do componente físico.
	Localização no Veículo	Posição física do sensor (Ex: Teto, Parabrisa, Lateral Direita).
	Última Calibração	Data em que o sensor foi ajustado e validado pela última vez.
Leitura de Telemetria	Identificador do Veículo	Referência ao veículo que gerou o registro de leitura.
	Instante da Leitura	Data e hora exata em que os dados foram coletados.

	Tempo em Operação	Duração de atividade desde a última medição operacional.
	Distância Percorrida	Total de quilômetros registrados pelo odômetro.
	Velocidade Instantânea	Velocidade do veículo no momento da leitura.
	Carga da Bateria	Nível de energia restante (para elétricos/híbridos).
	Nível de Combustível	Volume de combustível restante (para combustão/híbridos).
Diagnóstico Operacional (Parte da Telemetria)	Nome do Sensor	Nome do componente em análise na leitura.
	Status de Funcionamento	Condição operacional atual (Ex: OK, Degradado, Falha).
	Taxa de Erros	Porcentagem de erros de leitura ou transmissão do sensor.
	Temperatura Operacional	Temperatura interna do sensor no momento da coleta.
	Código de Falha	Código específico que descreve o tipo de mau funcionamento, se houver.

Nota-se que alguns grupos representam elementos internos de outros grupos, isso se deve a alguns dos grupos terem componentes compostos, sendo a configuração de sensores um grupo que se repete para cada sensor em um mesmo veículo, assim como o diagnóstico operacional, que tem o mesmo propósito ao realizar a leitura de telemetria de um veículo.

4.1. Não relacional

Para o banco não relacional criado no MongoDB, a distribuição dos dados em *collections* foi exibida na Tabela 2 a seguir:

Tabela 2. Dados nas *collections* no MongoDB

Collections	**Atributos**		**Tipo de dado**
veiculos	_id (VIN)		STRING
	marca		STRING
	modelo		STRING
	nivel_autonomia		INT
	tipo_propulsao		STRING
	data_de_fabricacao		DATETIME
	sensores_config	tipo	STRING
		versao_hw	STRING
		local	STRING
		ult_calibracao	DATETIME
telemetrias	id_veiculo		STRING
	timestamp		DATETIME
	tempo_ativo_segundos		INT
	quilometragem		INT
	velocidade_kmh		INT
	nivel_bateria_percentual		INT
	nivel_combustivel_percentual		INT
	diagnostico_sensores	nome_sensor	STRING
		status	STRING
		taxa_erros_pct	FLOAT
		temp_operacao_c	FLOAT
		cod_falha	STRING

Onde percebe-se que os grupos compostos mencionados anteriormente estão de fato dentro da *collection* esperada.

4.2. Relacional

Para o banco relacional criado no MySQL, a distribuição dos dados em tabelas foi exibida na Tabela 3 a seguir:

Tabela 3. Dados nas tabelas no MySQL

Tabelas	Atributos	Tipo de dado
Veiculos	veiculo_id (VIN) (PK)	VARCHAR(30)
	modelo	VARCHAR(100)
	marca	VARCHAR(50)
	nivel_autonomia	TINYINT
	tipo_propulsao	ENUM('Eletrico', 'Combustao', 'Hibrido')
	data_de_fabricacao	DATETIME
Configuracao_Sensores	config_id (PK)	BIGINT
	versao_hw	STRING
	local	STRING
	ult_calibracao	DATETIME
	fk_Veiculos_veiculo_id (FK)	VARCHAR(30)
Telemetrias	leitura_id (PK)	BIGINT
	quilometragem_km	DECIMAL(10, 2)
	velocidade_kph	DECIMAL(5, 2)
	nivel_bateria_pct	TINYINT
	nivel_combustivel_pct	TINYINT
	tempo_ativo_segundos	INT
	fk_Veiculos_veiculo_id (FK)	VARCHAR(30)
Diagnostico_Sensores	diagnostico_id (PK)	BIGINT
	nome_sensor	VARCHAR(50)

	status_sensor	ENUM('OK', 'Degradado', 'Falha')
	taxa_erros_pct	DECIMAL(5, 3)
	cod_falha	VARCHAR(20)
	temp_operacao_c	DECIMAL(4, 1)
	fk_Telemetrias_leitura_id (FK)	BIGINT

Onde percebe-se que, diferente do MongoDB, os grupos compostos mencionados anteriormente precisaram ser representados em tabelas distintas e unidas através de chaves estrangeiras.

5. Resultados

Para o estudo, ambos os bancos de dados foram populados com 5000 registros de veículos e 50000 registros de telemetrias, entretanto, devido à já mencionada necessidade de dividir alguns grupos em tabelas para o MySQL, temos 4000 configurações de sensores e 400000 diagnósticos de sensores, sendo então executadas as seguintes consultas de análise:

1. Qual é a taxa de erro média dos sensores LiDAR instalados em veículos de Nível SAE 4 que estão operando em velocidade superior a 80 km/h e qual a porcentagem desses sensores que atingiu o status "Degradado" na última hora?
2. Qual a quantidade total de veículos que tiveram qualquer sensor LiDAR com erro alto (> 0.1%) nas últimas 24 horas?
3. Dado um código de falha crítico (CAM_E01), qual é o VIN do veículo, o modelo da leitura onde a falha foi registrada pela primeira vez, e há quanto tempo o veículo estava em operação (tempo_ativo_segundos) quando isso ocorreu?
4. Qual é a Taxa de Falha Crítica por Quilometragem Rodada (Falhas/1000 km) para veículos fabricados antes de 2023-01-01, comparada aos modelos fabricados posteriormente?
5. Quais são os 5 sensores que mais frequentemente reportam temperatura de operação acima de 70°C e em que marcas de veículos esses sensores estão predominantemente instalados?

Posteriormente, foram realizadas as seguintes operações de Atualização (Update) nas bases de dados:

1. Para todos os veículos de nível de autonomia 3, atualize o nível para 4. (Simulando uma atualização de software)
2. Atualiza todas as telemetrias de última hora onde o status do LiDAR_360 é 'Degradado' para 'OK'. (Apenas a fins demonstrativos)

Por fim, o ciclo de testes foi concluído com as seguintes operações de Exclusão (Delete):

1. Remove as leituras feitas a mais de 30 dias na coleção/tabela telemetrias. (Simulando uma política de retenção de dados periódicos)
2. Remove todos os veículos com tipo_propulsao: "Combustao" e suas telemetrias associadas. (Simulando fim do suporte para esse tipo de veículo)

Dessa forma, o código em Python usado para processar as queries foi disponibilizado no ANEXO 1. Em sequência, será apresentado os resultados de tempo para as operações realizadas nos bancos de dados.

5.1. Resultados das operações nos bancos de dados

Para quantificar o desempenho do modelo de dados implementado, o tempo de execução de cada operação foi registrado. As tabelas a seguir discriminam a latência obtida para cada fase do CRUD, sendo o tempo de inserção detalhado na Tabela 4, o tempo de busca na Tabela 5, o tempo de alteração na Tabela 6, e o tempo de exclusão na Tabela 7.

Tabela 4. Tempos e quantidade de inserções

Inserção		Quantidade		Tempo em segundos	
MongoDB	MySQL	MongoDB	MySQL	MongoDB	MySQL
veiculos	Veiculos	5000	5000	0.0330	0.2747
	Configuracao_Sensores		40000		1.7079
telemetrias	Telemetrias	50000	50000	0.3980	1.0514
	Diagnostico_Sensores		400000		8.0260

Tabela 5. Tempos de buscas

Pergunta	Tempo em segundos	
	MongoDB	MySQL
1	0.0961	0.2697

2	0.0350	0.2200
3	0.4230	0.2590
4	2.6710	32.8499
5	0.9049	0.3670

Tabela 6. Tempos e quantidade de atualizações

Atualização	Quantidade		Tempo em segundos	
	MongoDB	MySQL	MongoDB	MySQL
1	1639	1639	0.0260	0.0180
2	0	0	0.1180	0.5620

Tabela 7. Tempos e quantidade de remoções

Remoção	Quantidade		Tempo em segundos	
	MongoDB	MySQL	MongoDB	MySQL
1	45810	45810	0.3960	7.4754
2	1664+1367	1664+Cascata	0.0410	8.3259

Onde as quantidades da segunda remoção representam a quantidade de dados removidos para os veículos e telemetrias, respectivamente. Também é importante notar que devido à propriedade de DELETE ON CASCADE aplicado na modelagem do banco de dados MySQL permite a remoção direta dos veículos com a remoção automatizada nas tabelas dependentes da mesma, nesse caso as tabelas Configuracao_Sensores, Telemetrias e consequentemente Diagnostico_Sensores, não possuindo uma exibição direta da quantidade de elementos removidos nas mesmas.

5.2. Análise dos resultados

Com esses dados, pode-se notar que de forma geral o banco de dados NoSQL MongoDB teve uma maior eficiência, não só em tempo, onde teve uma vantagem média de aproximadamente 6 segundos, como também organizacional, permitindo a inserção flexível de dados sem a necessidade de seguir um *SCHEMA* pré-definido, em relação ao banco de dados SQL MySQL. Para o projeto proposto, a quantidade e formatação dos dados, sendo consideravelmente volumoso e com dados compostos, apresenta características típicas de sistemas IoT, sendo muito

vantajosos para bancos NoSQL, que permitem o armazenamento de dados de diferentes maneiras a depender do SGBD (Sistema de Gerenciamento de Banco de Dados) escolhido, além de permitirem dados semi-estruturados. Por outro lado, o banco de dados MySQL teve uma grande desvantagem pela necessidade de dividir os dados compostos em novas tabelas pelas regras de normalização (Configuracao_Sensores e Diagnostico_Sensores), gerando um grande número de acessos a tabelas, principalmente às tabelas Telemetrias e Diagnostico_Sensores através de operações de *JOIN*, e outras operações extras em relação às operações nas *collections* do MongoDB.

Apesar de possuir um tempo melhor em casos específicos, como na primeira *query* de atualização de dados, que apenas acessava uma tabela e alterava dados em alguns atributos em algumas *rows* (linhas), sendo mais prático que o acesso a documentos BSON pelo MongoDB, de forma geral precisou fazer muito mais consultas, tantos nas buscas quanto nas outras operações, para selecionar corretamente os dados a serem trabalhados pelas *queries*, precisando realizar muito mais trabalho que os acessos a documentos do MongoDB. Isso demonstra a importância e a utilidade dos bancos de dados NoSQL em cenários de dados volumosos e semi-estruturados, especialmente no contexto de IoT e em relação ao projeto posto neste artigo.

6. Conclusão

Este artigo teve como proposta a comparação de eficiência entre bancos de dados SQL e NoSQL, utilizando respectivamente do MySQL e do MongoDB como SGBDs, simulando a manipulação de dados de um sistema para monitoramento e manutenção preventiva de veículos autônomos, representando um cenário clássico de Internet das Coisas (IoT). A partir de operações CRUD (Create, Read, Update, Delete) em uma base volumosa de dados, conclui-se a grande eficiência dos bancos de dados NoSQL em cenários como esse, possuindo tempo médio de aproximadamente 6 segundos de vantagem em relação ao MySQL com todos serviços sendo aplicados em uma mesma máquina para manter a consistência de performance e temporal.

A superioridade do MongoDB demonstrou que, em contextos de alta volumetria e dados semi-estruturados, a modelagem orientada a documentos, que armazena os dados relacionados juntos e elimina a necessidade de JOINs complexos nas consultas de telemetria, é inerentemente mais eficiente. Esta diferença de desempenho é fundamental, pois implica que a arquitetura NoSQL oferece maior escalabilidade e menor latência para a tomada de decisão em sistemas de manutenção preditiva em tempo real.

7. Referências

ACEA. Motor vehicle production, by world region. 2024. Disponível em: https://www.acea.auto/figure/motor-vehicle-production-by-world-region/. Acesso em: 08 dez. 2025.

AYALA, R.; MOHD, T. K. Sensors in Autonomous Vehicles: A Survey. Journal of Autonomous Vehicles and Systems, v. 1, n. 3, p. 1-16, nov. 2021. Disponível em: https://www.researchgate.net/publication/356147076_Sensors_in_Autonomous_Vehicles_A_Survey. Acesso em: 08 dez. 2025. DOI: 10.1115/1.4052991.

RAJ, V.; SHARMA, D. Predictive Maintenance in Autonomous Vehicles Using Machine Learning Techniques. In: INTERNATIONAL CONFERENCE ON ELECTRICAL, MECHANICAL AND COMPUTER ENGINEERING, 2., 2024. Anais [ou Proceedings]... 2024. Disponível em: https://ieeexplore.ieee.org/document/10912235. Acesso em: 08 dez. 2025.

SALES, Lilia Maia de Morais; BEZERRA, Mário Quesado Miranda. Os avanços tecnológicos do século XXI e o desenvolvimento de habilidades necessárias ao profissional do Direito a partir das abordagens das Universidades de Harvard e Stanford. Pensar, Fortaleza, v. 23, n. 4, p. 1-13, out./dez. 2018. Disponível em: https://ojs.unifor.br/rpen/article/download/8016/pdf/31946. Acesso em: 08 dez. 2025. DOI: 10.5020/2317-2150.2018.8016.

WORLD ECONOMIC FORUM. Which countries are ahead in the global autonomous vehicle race? 2025. Disponível em: https://www.weforum.org/stories/2025/05/autonomous-vehicles-technology-future/. Acesso em: 08 dez. 2025.

ANEXO 1: Script Python

```
"""Instalação das bibliotecas necessárias"""

# python -m pip install pymongo
# python -m pip install PyMySQL
# python -m pip install faker

"""Import das bibliotecas"""

from pymongo import MongoClient
import pymysql
from faker import Faker
from faker.providers import automotive
import time
import random
from datetime import datetime, timedelta, timezone

# Nessita para ser utilizado:
# Banco da dados MongoDB: ['dbname']
# Collections: ['telemetrias', 'veiculos']
# Banco da dados MySQL: ['dbname']
# SQL disponível no repositório Github:
https://github.com/RechargeCG/projeto-banco-de-dados-iot.git

uri = 'localhost:27017'
client = MongoClient( uri )
mydb = client["dbname"]
# Banco: ['dbname']
# Collections: ['telemetrias', 'veiculos']
print(mydb.list_collection_names())

mysqldb = pymysql.connect(
  host="host",
  user="user",
  password="password",
  db="dbname",
  charset="utf8"
)

#criando o cursor
mycursor = mysqldb.cursor()

mycursor.execute("show tables")
```

```
#exibindo as informações
for x in mycursor:
  print(x)

"""Criação dos dados"""

# Seed para garantir dados consistentes entre testes
seed = 13
random.seed(seed)

# Inicializa Faker
Faker.seed(seed)
fake = Faker()
fake.add_provider(automotive)

# Funções auxiliares para dados realistas dos veículos
def gerar_sensores_config():
    sensores_base = [
        # Visão Principal e Profundidade
        {'tipo': 'LiDAR_360', 'versao_hw': 'V4.5', 'local': 'Teto', 'ult_calibracao':
fake.date_time_between(start_date='-6m', end_date='now', tzinfo=None)},
        {'tipo': 'Radar_Frontal', 'versao_hw': 'V5.1', 'local': 'Parachoque', 'ult_calibracao':
fake.date_time_between(start_date='-3m', end_date='now', tzinfo=None)},
        {'tipo': 'Camera_Estereo', 'versao_hw': 'V10.2', 'local': 'Parabrisa',
'ult_calibracao': fake.date_time_between(start_date='-1y', end_date='now',
tzinfo=None)},
        {'tipo': 'Radar_Lateral_Dir', 'versao_hw': 'V5.1', 'local': 'Lateral_Dir',
'ult_calibracao': fake.date_time_between(start_date='-3m', end_date='now',
tzinfo=None)},
        # Localização e odometria
        {'tipo': 'Receptor_GPS_RTK', 'versao_hw': 'V2.2', 'local': 'Antena_Teto',
'ult_calibracao': fake.date_time_between(start_date='-2y', end_date='now',
tzinfo=None)},
        {'tipo': 'IMU', 'versao_hw': 'V8.0', 'local': 'Chassi_Central', 'ult_calibracao':
fake.date_time_between(start_date='-1y', end_date='now', tzinfo=None)},
        # Sistemas de controle
        {'tipo': 'Sensor_Torque_Volante', 'versao_hw': 'V1.1', 'local': 'Coluna_Direcao',
'ult_calibracao': fake.date_time_between(start_date='-1y', end_date='now',
tzinfo=None)},
        {'tipo': 'Sensor_Pressao_Freio', 'versao_hw': 'V2.0', 'local': 'Unid_Hidraulica',
'ult_calibracao': fake.date_time_between(start_date='-1y', end_date='now',
tzinfo=None)}
    ]
    return sensores_base
```

```
def gerar_diagnostico_sensores(sensores_config):
    diagnosticos = []
    for sensor in sensores_config:
        status = 'OK'
        # Gera valores base para OK
        taxa_erros = round(random.uniform(0.001, 0.1), 3)
        temp = round(random.uniform(30.0, 50.0), 1)
        cod_falha = None

        if random.random() < 0.05:
            status = 'Degradado'
            taxa_erros = round(random.uniform(0.1, 0.9), 3)
            temp = round(random.uniform(50.0, 75.0), 1)

        elif random.random() < 0.02:
            status = 'Falha'
            taxa_erros = 1.0
            temp = 99.9

            tipo = sensor['tipo']

            if 'Camera' in tipo:
                cod_falha = random.choice(['CAM_E01', 'CAM_L04', 'CAM_S02'])
            elif 'LiDAR' in tipo:
                cod_falha = random.choice(['LiDAR_D02', 'LiDAR_S01', 'LiDAR_T03'])
            elif 'Radar' in tipo:
                cod_falha = random.choice(['RAD_S05', 'RAD_T01', 'RAD_F07'])
            elif 'GPS' in tipo:
                cod_falha = random.choice(['GPS_N01', 'GPS_T01'])
            elif 'IMU' in tipo:
                cod_falha = random.choice(['IMU_F03', 'IMU_G01'])
            elif 'Sensor' in tipo:
                cod_falha = random.choice(['ACT_C01', 'ACT_M02'])
            else:
                cod_falha = 'GER_U99'

        diagnostico = {
            'nome_sensor': sensor['tipo'],
            'status': status,
            'taxa_erros_pct': taxa_erros,
            'temp_operacao_c': temp,
            'cod_falha': cod_falha
        }
        diagnosticos.append(diagnostico)
    return diagnosticos
```

```
# Gerando dados base
marca_modelo = {}
for i in range(100):
  marca = fake.last_name()
  marca_modelo[i] = [marca, []]
  for j in range(30):
    modelo = fake.last_name()
    marca_modelo[i][1].append(modelo)

tipos_propulsao = ['Eletrico', 'Combustao', 'Hibrido']

num_veiculos = 5000
num_leituras = 50000

# Gerando veículos
veiculos_autonomos = []
config_map = {}

for i in range(num_veiculos):
    marcaemodelo = random.choice(marca_modelo)
    marca = marcaemodelo[0]
    modelo = random.choice(marcaemodelo[1])

    vin = fake.vin()
    config = gerar_sensores_config()
    tipo_propulsao = random.choice(tipos_propulsao)

    veiculo = {
        '_id': vin,
        'marca': marca,
        'modelo': modelo,
        'nivel_autonomia': random.randint(3, 5),
        'tipo_propulsao': tipo_propulsao,
        'data_de_fabricacao': fake.date_time_between(start_date='-5y', end_date='now',
tzinfo=None),
        'sensores_config': config
    }

    veiculos_autonomos.append(veiculo)
    config_map[vin] = {'config': config, 'propulsao': tipo_propulsao}


# Gerando telemetrias
telemetrias = []
```

```
for i in range(num_leituras):
    veiculo_idx = random.randint(0, len(veiculos_autonomos) - 1)
    id_veiculo = veiculos_autonomos[veiculo_idx]['_id']

    veiculo_data = config_map[id_veiculo]
    config_base = veiculo_data['config']
    propulsao_do_veiculo = veiculo_data['propulsao']

    leitura = {
        'id_veiculo': id_veiculo,
        'tempo_ativo_segundos': random.randint(0,100000),
        'timestamp': fake.date_time_between(start_date='-1y', end_date='now',
tzinfo=None),
        'quilometragem': random.randint(0,999999),
        'velocidade_kmh': random.randint(0,180),
        'diagnostico_sensores': gerar_diagnostico_sensores(config_base)
    }

    if propulsao_do_veiculo == 'Eletrico' or propulsao_do_veiculo == 'Hibrido':
        leitura['nivel_bateria_percentual'] = random.randint(0, 100)
    if propulsao_do_veiculo == 'Combustao' or propulsao_do_veiculo == 'Hibrido':
        leitura['nivel_combustivel_percentual'] = random.randint(0, 100)

    telemetrias.append(leitura)

"""Etapa de inserção MongoDB"""

#########################################################################
############
#                 Inserção no MongoDB - com marcação de tempo              #
#########################################################################
############

mycol_veiculos = mydb["veiculos"]
mycol_telemetrias = mydb["telemetrias"]
mycol_veiculos.delete_many({})
mycol_telemetrias.delete_many({})
print("Dados do banco removidos.")

################################## INSERT VEICULOS
##################################

# Marca o início
inicio = time.time()
```

```
#realiza o insert
mycol_veiculos.insert_many(veiculos_autonomos)
# Marca o fim
fim = time.time()
# Tempo
tempo_execucao = fim - inicio
print(f"Inserção na Collection veiculos de {len(veiculos_autonomos)} documentos
durou {tempo_execucao:.4f} segundos")


################################ INSERT TELEMETRIAS
################################

# Marca o início
inicio = time.time()
#realiza o insert
# mycol_telemetrias.insert_many(telemetrias)
batch_size = 5000
for i in range(0, len(telemetrias), batch_size):
    batch = telemetrias[i:i + batch_size]
    mycol_telemetrias.insert_many(batch)
    print(f"Lote {i//batch_size + 1} de {len(telemetrias)//batch_size} inserido.")
# Marca o fim
fim = time.time()
# Tempo
tempo_execucao = fim - inicio
print(f"Inserção na Collection devices de {len(telemetrias)} documentos durou
{tempo_execucao:.4f} segundos")

"""Etapa de inserção MySQL"""

def normalizar_para_mysql(veiculos_autonomos, telemetrias):
    sql_veiculos = []
    sql_configuracao_sensores = []
    sql_telemetria = []
    sql_diagnostico_sensores = []

    leitura_id_counter = 1
    config_id_counter = 1
    diagnostico_id_counter = 1

    for veiculo in veiculos_autonomos:
        vin = veiculo['_id']

        sql_veiculos.append({
```

```
            'veiculo_id': vin,
            'marca': veiculo['marca'],
            'modelo': veiculo['modelo'],
            'nivel_autonomia': veiculo['nivel_autonomia'],
            'tipo_propulsao': veiculo['tipo_propulsao'],
            'data_fabricacao': veiculo['data_de_fabricacao'].strftime('%Y-%m-%d')
        })

        for sensor_config in veiculo['sensores_config']:
            sql_configuracao_sensores.append({
                'config_id': config_id_counter,
                'fk_Veiculos_veiculo_id': vin,
                'nome_sensor': sensor_config['tipo'],
                'versao_hardware': sensor_config['versao_hw'],
                'ultima_calibracao': sensor_config['ult_calibracao'].strftime('%Y-%m-%d
%H:%M:%S')
            })
            config_id_counter += 1

    for leitura in telemetrias:
        vin = leitura['id_veiculo']
        current_leitura_id = leitura_id_counter

        sql_telemetria.append({
            'leitura_id': current_leitura_id,
            'fk_Veiculos_veiculo_id': vin,
            'timestamp': leitura['timestamp'].strftime('%Y-%m-%d %H:%M:%S'),
            'tempo_ativo_segundos': leitura.get('tempo_ativo_segundos'),
            'quilometragem_km': leitura['quilometragem'],
            'velocidade_kph': leitura['velocidade_kmh'],
            'nivel_bateria_pct': leitura.get('nivel_bateria_percentual'),
            'nivel_combustivel_pct': leitura.get('nivel_combustivel_percentual')
        })
        leitura_id_counter += 1

        for diagnostico in leitura['diagnostico_sensores']:
            sql_diagnostico_sensores.append({
                'diagnostico_id': diagnostico_id_counter,
                'fk_Telemetrias_leitura_id': current_leitura_id,
                'nome_sensor': diagnostico['nome_sensor'],
                'status_sensor': diagnostico['status'],
                'taxa_erros_pct': diagnostico['taxa_erros_pct'],
                'temp_operacao_c': diagnostico['temp_operacao_c'],
                'cod_falha': diagnostico['cod_falha']
            })
```

```
            diagnostico_id_counter += 1

    print(f"Normalização concluída: {len(sql_veiculos)} Veículos,
{len(sql_telemetria)} Telemetrias, {len(sql_configuracao_sensores)} Configurações,
{len(sql_diagnostico_sensores)} Diagnósticos.")
    return sql_veiculos, sql_configuracao_sensores, sql_telemetria,
sql_diagnostico_sensores

def inserir_em_lote_mysql(nome_tabela, lista_dados):
    if not lista_dados:
        print(f"Lista de dados para {nome_tabela} está vazia. Pulando.")
        return

    colunas = lista_dados[0].keys()

    colunas_sql = ", ".join(colunas)
    placeholders = ", ".join(["%s"] * len(colunas))

    query = f"INSERT INTO {nome_tabela} ({colunas_sql}) VALUES
({placeholders})"

    dados_para_insercao = [tuple(d.values()) for d in lista_dados]

    print(f"Inserindo {len(dados_para_insercao)} linhas na tabela {nome_tabela}...")

    return query, dados_para_insercao

res = normalizar_para_mysql(veiculos_autonomos,telemetrias)

#########################################################################
###########
#                Inserção no MySQL - com marcação de tempo              #
#########################################################################
###########

mycursor.execute("DELETE FROM Diagnostico_Sensores")
mycursor.execute("DELETE FROM Telemetrias")
mycursor.execute("DELETE FROM Configuracao_Sensores")
mycursor.execute("DELETE FROM Veiculos")
mysqldb.commit()
print("Dados do banco removidos.")

################################ INSERT VEICULOS
################################
```

```
query, dados_para_insercao = inserir_em_lote_mysql("Veiculos", res[0])
# Marca o início
inicio = time.time()
#realiza o insert
mycursor.executemany(query, dados_para_insercao)
mysqldb.commit()
# Marca o fim
fim = time.time()
# Tempo
tempo_execucao = fim - inicio
print(f"Inserção na tabela Veiculos de {len(res[0])} documentos durou
{tempo_execucao:.4f} segundos")

################################ INSERT CONFIGURACOES_SENSORES
################################

query, dados_para_insercao = inserir_em_lote_mysql("Configuracao_Sensores",
res[1])
# Marca o início
inicio = time.time()
#realiza o insert
mycursor.executemany(query, dados_para_insercao)
mysqldb.commit()
# Marca o fim
fim = time.time()
# Tempo
tempo_execucao = fim - inicio
print(f"Inserção na tabela Configuracao_Sensores de {len(res[1])} documentos durou
{tempo_execucao:.4f} segundos")

################################ INSERT TELEMETRIAS
################################

query, dados_para_insercao = inserir_em_lote_mysql("Telemetrias", res[2])
# Marca o início
inicio = time.time()
#realiza o insert
mycursor.executemany(query, dados_para_insercao)
mysqldb.commit()
# Marca o fim
fim = time.time()
# Tempo
tempo_execucao = fim - inicio
print(f"Inserção na tabela Telemetrias de {len(res[2])} documentos durou
{tempo_execucao:.4f} segundos")
```

```
################################ INSERT DIAGNOSTICO_SENSORES
################################

query, dados_para_insercao = inserir_em_lote_mysql("Diagnostico_Sensores",
res[3])
# Marca o início
inicio = time.time()
#realiza o insert
mycursor.executemany(query, dados_para_insercao)
mysqldb.commit()
# Marca o fim
fim = time.time()
# Tempo
tempo_execucao = fim - inicio
print(f"Inserção na tabela Diagnostico_Sensores de {len(res[3])} documentos durou
{tempo_execucao:.4f} segundos")

"""Etapa de buscas MongoDB

"""

# Qual é a taxa de erro média dos sensores LiDAR instalados em veículos de Nível
SAE 4
# que estão operando em velocidade superior a 80 km/h e qual a porcentagem desses
sensores
# que atingiu o status "Degradado" na última hora?
mycol_telemetrias = mydb['telemetrias']
inicio = time.time()
res = (list(mycol_telemetrias.aggregate([
    {
        '$match': {
            'timestamp': { '$gte': datetime.now() - timedelta(hours=1) }
        }
    },
    {
        '$lookup': {
            'from': 'veiculos',
            'localField': 'id_veiculo',
            'foreignField': '_id',
            'as': 'dados_veiculo'
        }
    },
    {
        '$unwind': '$dados_veiculo'
```

```
    },
    {
        '$match': {
            'dados_veiculo.nivel_autonomia': 4,
            'velocidade_kmh': { '$gt': 80 }
        }
    },
    {
        '$unwind': '$diagnostico_sensores'
    },
    {
        '$match': {
            'diagnostico_sensores.nome_sensor': 'LiDAR_360'
        }
    },
    {
        '$group': {
            '_id': None,
            'taxa_erro_media': { '$avg': '$diagnostico_sensores.taxa_erros_pct' },
            'total_sensores': { '$sum': 1 },
            'sensores_degradados': {
                '$sum': {
                    '$cond': [ { '$eq': ['$diagnostico_sensores.status', 'Degradado'] }, 1, 0 ]
                }
            }
        }
    },
    {
        '$project': {
            '_id': 0,
            'taxa_erro_media': { '$round': ['$taxa_erro_media', 4] },
            'porcentagem_degradada': {
                '$round': [
                    { '$multiply': [ { '$divide': ['$sensores_degradados', '$total_sensores'] },
100 ] }, 2
                ]
            }
        }
    }
]))
fim = time.time()
tempo_execucao = fim - inicio
print(f"Taxa de erro media LiDAR SAE 4 > 80km/h: (Tempo: {tempo_execucao:.4f}
segundos)")
print(res)
```

```
# ------------------------------------------------------------------------

# Qual a quantidade total de veículos que tiveram qualquer sensor LiDAR com erro alto (> 0.1%)
# nas últimas 24 horas?
mycol_telemetrias = mydb['telemetrias']
inicio = time.time()
res = (list(mycol_telemetrias.aggregate([
    {
        '$match': {
            'timestamp': { '$gte': datetime.now() - timedelta(hours=24) }
        }
    },
    {
        '$unwind': '$diagnostico_sensores'
    },
    {
        '$match': {
            'diagnostico_sensores.nome_sensor': 'LiDAR_360',
            'diagnostico_sensores.taxa_erros_pct': { '$gt': 0.1 }
        }
    },
    {
        '$group': {
            '_id': '$id_veiculo'
        }
    },
    {
        '$group': {
            '_id': None,
            'total_veiculos': { '$sum': 1 }
        }
    },
    {
        '$project': {
            '_id': 0,
            'total_veiculos': 1
        }
    }
])))
fim = time.time()
tempo_execucao = fim - inicio
print(f"Total de veículos (LiDAR com erro alto, última 24h): (Tempo: {tempo_execucao:.4f} segundos)")
```

```
print(res)

# ---------------------------------------------------------------------------

# Dado um código de falha crítico (CAM_E01), qual é o VIN do veículo, o modelo da
# leitura onde a falha foi registrada pela primeira vez, e há quanto tempo o veículo
# estava em operação (tempo_ativo_segundos) quando isso ocorreu?
mycol_telemetrias = mydb['telemetrias']
inicio = time.time()
res = (list(mycol_telemetrias.aggregate([
    {
      '$unwind': '$diagnostico_sensores'
    },
    {
      '$match': {
        'diagnostico_sensores.cod_falha': 'CAM_E01'
      }
    },
    {
      '$sort': {
        'timestamp': 1
      }
    },
    {
      '$limit': 1
    },
    {
      '$lookup': {
        'from': 'veiculos',
        'localField': 'id_veiculo',
        'foreignField': '_id',
        'as': 'dados_veiculo'
      }
    },
    {
      '$unwind': '$dados_veiculo'
    },
    {
      '$project': {
        '_id': 0,
        'VIN': '$id_veiculo',
        'modelo_veiculo': '$dados_veiculo.modelo',
        'tempo_ativo_segundos': '$tempo_ativo_segundos',
        'data_falha': '$timestamp'
      }
```

```
    }
])))
fim = time.time()
tempo_execucao = fim - inicio
print(f"Dados da primeira falha crítica (CAM_E01): (Tempo: {tempo_execucao:.4f}
segundos)")
print(res)

# ---------------------------------------------------------------------------

# Qual é a Taxa de Falha Crítica por Quilometragem Rodada (Falhas/1000 km)
# para veículos fabricados antes de 2023-01-01, comparada aos modelos fabricados
posteriormente?
mycol_telemetrias = mydb['telemetrias']
inicio = time.time()
res_nova = (list(mycol_telemetrias.aggregate([
    {
        '$lookup': {
            'from': 'veiculos',
            'localField': 'id_veiculo',
            'foreignField': '_id',
            'as': 'dados_veiculo'
        }
    },
    {
        '$unwind': '$dados_veiculo'
    },
    {
        '$unwind': '$diagnostico_sensores'
    },
    {
        '$group': {
            '_id': '$id_veiculo',
            'data_fabricacao': { '$first': '$dados_veiculo.data_de_fabricacao' },
            'quilometragem': { '$max': '$quilometragem' },
            'total_falhas_criticas': {
                '$sum': {
                    '$cond': [ { '$eq': ['$diagnostico_sensores.status', 'Falha'] }, 1, 0 ]
                }
            }
        }
    },
    {
        '$group': {
            '_id': {
```

```
            '$lt': ['$data_fabricacao', datetime(2023, 1, 1)]
          },
          'total_km_rodado': { '$sum': '$quilometragem' },
          'soma_total_falhas': { '$sum': '$total_falhas_criticas' }
       }
    },
    {
       '$project': {
          '_id': 0,
          'grupo': {
             '$cond': [ '$_id', 'Frota Antiga (Antes de 2023)', 'Frota Nova (2023 em
diante)' ]
          },
          'falhas_por_1000km': {
             '$round': [
                { '$multiply': [ { '$divide': ['$soma_total_falhas', '$total_km_rodado'] },
1000 ] }, 4
             ]
          }
       }
    }
])))
fim = time.time()
tempo_execucao = fim - inicio
print(f"Taxa de falha crítica por 1000 km (comparativo por idade): (Tempo:
{tempo_execucao:.4f} segundos)")
print(res_nova)

# ------------------------------------------------------------------------

# Quais são os 5 sensores que mais frequentemente reportam temperatura de operação
acima de 70°C
# e em que marcas de veículos esses sensores estão predominantemente instalados?
mycol_telemetrias = mydb['telemetrias']
inicio = time.time()
res = (list(mycol_telemetrias.aggregate([
    {
       '$unwind': '$diagnostico_sensores'
    },
    {
       '$match': {
          'diagnostico_sensores.temp_operacao_c': { '$gt': 70.0 }
       }
    },
    {
```

```
        '$lookup': {
            'from': 'veiculos',
            'localField': 'id_veiculo',
            'foreignField': '_id',
            'as': 'dados_veiculo'
        }
    },
    {
        '$unwind': '$dados_veiculo'
    },
    {
        '$group': {
            '_id': {
                'sensor': '$diagnostico_sensores.nome_sensor',
                'marca': '$dados_veiculo.marca'
            },
            'frequencia_superaquecimento': { '$sum': 1 }
        }
    },
    {
        '$sort': {
            'frequencia_superaquecimento': -1
        }
    },
    {
        '$limit': 5
    },
    {
        '$project': {
            '_id': 0,
            'sensor': '$_id.sensor',
            'marca_veiculo': '$_id.marca',
            'contagem': '$frequencia_superaquecimento'
        }
    }
])))
fim = time.time()
tempo_execucao = fim - inicio
print(f"Top 5 sensores com superaquecimento por marca: (Tempo:
{tempo_execucao:.4f} segundos)")
print(res)

"""Etapa de buscas MySQL"""

mycursor = mysqldb.cursor()
```

```
# Assumindo que 'mycursor' e 'mydb' estão definidos e a conexão está ativa.

# A data de corte de 2023-01-01 deve ser passada como string no SQL.
DATA_CORTE_FROTA_ANTIGA = '2023-01-01'

# Qual é a taxa de erro média dos sensores LiDAR instalados em veículos de Nível SAE 4
# que estão operando em velocidade superior a 80 km/h e qual a porcentagem desses sensores
# que atingiu o status "Degradado" na última hora?
inicio = time.time()
query_sql_1 = """
SELECT
    ROUND(AVG(d.taxa_erros_pct), 4) AS taxa_erro_media,
    ROUND(SUM(CASE WHEN d.status_sensor = 'Degradado' THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) AS porcentagem_degradada
FROM Telemetrias t
JOIN Veiculos v ON t.fk_Veiculos_veiculo_id = v.veiculo_id
JOIN Diagnostico_Sensores d ON t.leitura_id = d.fk_Telemetrias_leitura_id
WHERE
    t.timestamp >= DATE_SUB(NOW(), INTERVAL 1 HOUR) AND
    v.nivel_autonomia = 4 AND
    t.velocidade_kph > 80 AND
    d.nome_sensor = 'LiDAR_360';
"""
mycursor.execute(query_sql_1)
res = mycursor.fetchall()
fim = time.time()
tempo_execucao = fim - inicio
print(f"Taxa de erro media LiDAR SAE 4 > 80km/h: (Tempo: {tempo_execucao:.4f} segundos)")
print(res)

# ------------------------------------------------------------------------

# Qual a quantidade total de veículos que tiveram qualquer sensor LiDAR com erro alto (> 0.1%)
# nas últimas 24 horas?
inicio = time.time()
query_sql_2 = """
SELECT
    COUNT(DISTINCT t.fk_Veiculos_veiculo_id) AS total_veiculos
FROM Telemetrias t
JOIN Diagnostico_Sensores d ON t.leitura_id = d.fk_Telemetrias_leitura_id
WHERE
```

```
    t.timestamp >= DATE_SUB(NOW(), INTERVAL 24 HOUR) AND
    d.nome_sensor = 'LiDAR_360' AND
    d.taxa_erros_pct > 0.1;
"""
mycursor.execute(query_sql_2)
res = mycursor.fetchall()
fim = time.time()
tempo_execucao = fim - inicio
print(f"Total de veículos (LiDAR com erro alto, última 24h): (Tempo:
{tempo_execucao:.4f} segundos)")
print(res)

# ----------------------------------------------------------------------

# Dado um código de falha crítico (CAM_E01), qual é o VIN do veículo, o modelo da
# leitura onde a falha foi registrada pela primeira vez, e há quanto tempo o veículo
# estava em operação (tempo_ativo_segundos) quando isso ocorreu?
na Telemetria, ou será omitida.
inicio = time.time()
query_sql_3 = """
SELECT
    v.veiculo_id AS VIN,
    v.modelo,
    t.tempo_ativo_segundos,
    t.timestamp AS data_falha

FROM Telemetrias t
JOIN Veiculos v ON t.fk_Veiculos_veiculo_id = v.veiculo_id
JOIN Diagnostico_Sensores d ON t.leitura_id = d.fk_Telemetrias_leitura_id
WHERE
    d.cod_falha = 'CAM_E01'
ORDER BY
    t.timestamp ASC
LIMIT 1;
"""
mycursor.execute(query_sql_3)
res = mycursor.fetchall()
fim = time.time()
tempo_execucao = fim - inicio
print(f"Dados da primeira falha crítica (CAM_E01): (Tempo: {tempo_execucao:.4f}
segundos)")
print(res)

# ----------------------------------------------------------------------
```

```
# Qual é a Taxa de Falha Crítica por Quilometragem Rodada (Falhas/1000 km)
# para veículos fabricados antes de 2023-01-01, comparada aos modelos fabricados posteriormente?
inicio = time.time()
query_sql_4 = f"""
SELECT
   CASE
      WHEN V.data_fabricacao < '{DATA_CORTE_FROTA_ANTIGA}' THEN 'Frota Antiga (Antes de 2023)'
      ELSE 'Frota Nova (2023 em diante)'
   END AS grupo,
   ROUND(SUM(T1.total_falhas_criticas) * 1000.0 / SUM(T1.quilometragem_max), 4) AS falhas_por_1000km
FROM Veiculos V
JOIN (

   SELECT
      T.fk_Veiculos_veiculo_id,
      MAX(T.quilometragem_km) AS quilometragem_max,
      SUM(CASE WHEN D.status_sensor = 'Falha' THEN 1 ELSE 0 END) AS total_falhas_criticas
   FROM Telemetrias T
   JOIN Diagnostico_Sensores D ON T.leitura_id = D.fk_Telemetrias_leitura_id
   GROUP BY T.fk_Veiculos_veiculo_id
) T1 ON V.veiculo_id = T1.fk_Veiculos_veiculo_id
GROUP BY grupo;
"""
mycursor.execute(query_sql_4)
res_nova = mycursor.fetchall()
fim = time.time()
tempo_execucao = fim - inicio
print(f"Taxa de falha crítica por 1000 km (comparativo por idade): (Tempo: {tempo_execucao:.4f} segundos)")
print(res_nova)

# ------------------------------------------------------------------------

# Quais são os 5 sensores que mais frequentemente reportam temperatura de operação acima de 70°C
# e em que marcas de veículos esses sensores estão predominantemente instalados?
inicio = time.time()
query_sql_5 = """
SELECT
   d.nome_sensor AS sensor,
   v.marca AS marca_veiculo,
```

```
    COUNT(*) AS contagem
FROM Diagnostico_Sensores d
JOIN Telemetrias t ON d.fk_Telemetrias_leitura_id = t.leitura_id
JOIN Veiculos v ON t.fk_Veiculos_veiculo_id = v.veiculo_id
WHERE
    d.temp_operacao_c > 70.0
GROUP BY
    d.nome_sensor, v.marca
ORDER BY
    contagem DESC
LIMIT 5;
"""
mycursor.execute(query_sql_5)
res = mycursor.fetchall()
fim = time.time()
tempo_execucao = fim - inicio
print(f"Top 5 sensores com superaquecimento por marca: (Tempo:
{tempo_execucao:.4f} segundos)")
print(res)

"""Etapa de atualizações MongoDB"""

from datetime import datetime, timedelta, timezone
import time

# Para todos os veículos de nível de autonomia 3, atualize o nível para 4. (Simulando
uma atualização de software)
mycol_veiculos = mydb['veiculos']
inicio = time.time()
res = mycol_veiculos.update_many(
    {
        "nivel_autonomia": 3
    },
    {
        "$set": {
            "nivel_autonomia": 4
        }
    }
)
fim = time.time()
tempo_execucao = fim - inicio
print(f"Atualização do Nível de Autonomia 3 para 4. (Tempo: {tempo_execucao:.4f}
segundos)")
print(f"Documentos modificados: {res.modified_count}")
```

```
# Atualiza todas as telemetrias da última hora onde o status do LiDAR_360 é
'Degradado'.
mycol_telemetrias = mydb['telemetrias']
inicio = time.time()
data_corte_1_hora = datetime.now(timezone.utc) - timedelta(hours=1)

res = mycol_telemetrias.update_many(
   {
      "timestamp": {"$gte": data_corte_1_hora},
      "diagnostico_sensores.status": "Degradado",
      "diagnostico_sensores.nome_sensor": "LiDAR_360"
   },
   {
      "$set": {
         "diagnostico_sensores.$[elem].status": "Checked OK",
         "diagnostico_sensores.$[elem].cod_falha": None
      }
   },
   array_filters=[
      {
         "elem.nome_sensor": "LiDAR_360",
         "elem.status": "Degradado"
      }
   ]
)
fim = time.time()
tempo_execucao = fim - inicio
print(f"Correção automatizada de status 'Degradado' em Telemetrias. (Tempo:
{tempo_execucao:.4f} segundos)")
print(f"Documentos modificados: {res.modified_count}")

"""Etapa de atualizações MySQL"""

# Para todos os veículos de nível de autonomia 3, atualize o nível para 4. (Simulando
uma atualização de software)
inicio = time.time()
query_sql_A = """
UPDATE Veiculos
SET nivel_autonomia = 4
WHERE nivel_autonomia = 3;
"""
mycursor.execute(query_sql_A)
modified_count_A = mycursor.rowcount
mysqldb.commit()
```

```
fim = time.time()
tempo_execucao = fim - inicio
print(f"Atualização do Nível de Autonomia 3 para 4. (Tempo: {tempo_execucao:.4f}
segundos)")
print(f"Documentos modificados: {modified_count_A}")

# Atualiza todas as telemetrias da última hora onde o status do LiDAR_360 é
'Degradado'.
inicio = time.time()
query_sql_B = """
UPDATE Diagnostico_Sensores d
JOIN Telemetrias t ON d.fk_Telemetrias_leitura_id = t.leitura_id
SET
   d.status_sensor = 'Checked OK',
   d.cod_falha = NULL
WHERE
   t.timestamp >= %s AND
   d.nome_sensor = 'LiDAR_360' AND
   d.status_sensor = 'Degradado';
"""
mycursor.execute(query_sql_B,data_corte_1_hora)
modified_count_B = mycursor.rowcount
mysqldb.commit()

fim = time.time()
tempo_execucao = fim - inicio
print(f"Correção automatizada de status 'Degradado' em Telemetrias. (Tempo:
{tempo_execucao:.4f} segundos)")
print(f"Documentos modificados: {modified_count_B}")

"""Etapa de remoção MongoDB"""

leituras_col = mydb['telemetrias']
veiculos_col = mydb['veiculos']

# Remove as leituras feitas a mais de 30 dias na coleção telemetrias. (Simulando uma
política de retenção de dados periódicos)
data_corte_30_dias = datetime.now(timezone.utc) - timedelta(days=30)
inicio = time.time()
res_leituras_antigas = leituras_col.delete_many({"timestamp": {"$lt":
data_corte_30_dias}})
deleted_leituras_count_antigas = res_leituras_antigas.deleted_count

fim = time.time()
tempo_execucao = fim - inicio
```

```
print(f"Remoção de Telemetrias Antigas (> 30 dias): (Tempo: {tempo_execucao:.4f}
segundos)")
print(f"Leituras excluídas: {deleted_leituras_count_antigas}")

# Remove todos os veículos com tipo_propulsao: "Combustao" e suas telemetrias
associadas. (Simulando fim do suporte para esse tipo de veículo)
inicio = time.time()
veiculo_ids_combustao = veiculos_col.distinct("_id", {"tipo_propulsao":
"Combustao"})

res_veiculos_combustao = veiculos_col.delete_many({"_id": {"$in":
veiculo_ids_combustao}})
deleted_veiculos_count_combustao = res_veiculos_combustao.deleted_count

res_leituras_combustao = leituras_col.delete_many({"id_veiculo": {"$in":
veiculo_ids_combustao}})
deleted_leituras_count_combustao = res_leituras_combustao.deleted_count

fim = time.time()
tempo_execucao = fim - inicio
print(f"Remoção de frota à combustão: (Tempo: {tempo_execucao:.4f} segundos)")
print(f"Veículos excluídos: {deleted_veiculos_count_combustao}")
print(f"Telemetrias excluídas: {deleted_leituras_count_combustao}")

"""Etapa de remoção MySQL"""

# Remove as leituras feitas a mais de 30 dias na coleção telemetrias. (Simulando uma
política de retenção de dados periódicos)
inicio = time.time()

query_sql_1 = """
DELETE FROM Telemetrias
WHERE timestamp < %s;
"""
mycursor.execute(query_sql_1,(data_corte_30_dias))
deleted_leituras_count_antigas = mycursor.rowcount
mysqldb.commit()

fim = time.time()
tempo_execucao = fim - inicio
print(f"Remoção de Telemetrias Antigas (> 30 dias): (Tempo: {tempo_execucao:.4f}
segundos)")
print(f"Leituras excluídas: {deleted_leituras_count_antigas}")

# Remove todos os veículos com tipo_propulsao: "Combustao" e suas telemetrias
```

```
associadas. (Simulando fim do suporte para esse tipo de veículo)
inicio = time.time()

# (Graças ao ON DELETE CASCADE, basta deletar o veículo na tabela pai)
query_sql_2 = """
DELETE FROM Veiculos
WHERE tipo_propulsao = 'Combustao';
"""
mycursor.execute(query_sql_2)
deleted_veiculos_count_combustao = mycursor.rowcount
mysqldb.commit()

fim = time.time()
tempo_execucao = fim - inicio
print(f"Remoção de frota à combustão: (Tempo: {tempo_execucao:.4f} segundos)")
print(f"Veículos excluídos: {deleted_veiculos_count_combustao}")
print(f"Telemetrias excluídas: {"Devido à cascata, não é possível visualizar
diretamente."}")
```