Monitoramento de Erros no PostgreSQL, MySQL e SQL Server com Notificação via Microsoft Teams

Empresas que dependem de bancos de dados críticos precisam de soluções robustas para detectar e responder rapidamente a erros, garantindo a continuidade operacional e minimizando impactos negativos.

Este artigo apresenta uma solução completa de monitoramento que abrange os principais bancos de dados usados no mercado: **PostgreSQL**, **MySQL** e **SQL Server**.

O sistema detecta erros, registra logs e envia notificações para o **Microsoft Teams** via Webhooks.

A solução utiliza **Java** e **C#** como linguagens de implementação, oferecendo flexibilidade e fácil integração em ambientes corporativos.

Estrutura e Requisitos

Uma tabela comum, presente nos bancos monitorados, armazena informações de status dos processos. Exemplo:

id	status	bot	timestamp
1	erro	RPA1	2025-01-16 12:00:00
2	warning	RPA2	2025-01-16 12:05:00

Funcionalidades

1. Consulta periódica para identificar registros com **status** "erro" ou "warning".
2. Registro em **arquivos de log** com timestamp.
3. Envio de mensagens personalizadas ao **Microsoft Teams** via Webhooks.

Requisitos Técnicos

1. Bancos de dados:
    - PostgreSQL (porta padrão: 5432).
    - MySQL (porta padrão: 3306).
    - SQL Server (porta padrão: 1433).
2. Ambiente de desenvolvimento:
    - **Java**: JDK 8+.
    - **C#**: .NET SDK 6.0+.
3. Webhook configurado no Microsoft Teams para envio de notificações.
4. Credenciais para acesso aos bancos de dados.

Implementação em Java

PostgreSQL

Este código monitora erros e os notifica no Microsoft Teams.

```
import java.sql.*;
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;

public class PostgreSQLErrorMonitor {
    private static final String DB_URL = "jdbc:postgresql://<HOST>:<PORT>/<DATABASE>";
    private static final String DB_USER = "<USER>";
    private static final String DB_PASSWORD = "<PASSWORD>";
    private static final String TEAMS_WEBHOOK_URL = "<URL_DO_WEBHOOK>";

    public static void main(String[] args) {
        while (true) {
            checkForErrors();
            try {
                Thread.sleep(600000); // Executa a cada 10 minutos
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private static void checkForErrors() {
        String query = "SELECT bot, status FROM sua_tabela WHERE status ILIKE '%erro%' OR
status ILIKE '%warning%'";
        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()) {
                String bot = rs.getString("bot");
                String status = rs.getString("status");

                saveToLog(bot, status);
                sendToTeams(bot, status);
            }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void saveToLog(String bot, String status) {
        try (FileWriter writer = new FileWriter("log_postgresql.txt", true)) {
            writer.write("Bot: " + bot + " - Erro: " + status + " - Timestamp: " + new java.util.Date() +
"\n");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void sendToTeams(String bot, String status) {
        try {
            URL url = new URL(TEAMS_WEBHOOK_URL);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type", "application/json");
            conn.setDoOutput(true);

            String jsonPayload = String.format("{\"text\": \"Erro detectado no RPA: %s\\nStatus:
%s\"}", bot, status);

            try (OutputStream os = conn.getOutputStream()) {
                os.write(jsonPayload.getBytes());
                os.flush();
            }

            if (conn.getResponseCode() != 200) {
                System.out.println("Falha ao enviar para o Teams: " + conn.getResponseCode());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

MySQL

O código para MySQL é similar ao de PostgreSQL, ajustando o **driver JDBC** e a URL:

```
private static final String DB_URL =
"jdbc:mysql://<HOST>:<PORT>/<DATABASE>?useSSL=false";
```

E a consulta SQL muda para:

```
String query = "SELECT bot, status FROM sua_tabela WHERE status LIKE '%erro%' OR status
LIKE '%warning%'";
```

SQL Server

Para SQL Server, ajuste a URL do banco:

```
private static final String DB_URL =
"jdbc:sqlserver://<HOST>:<PORT>;databaseName=<DATABASE>";
```

A estrutura geral do código permanece a mesma.

Implementação em C#

PostgreSQL

Utiliza o pacote **Npgsql**:

```
using System;
using Npgsql;
using System.IO;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

class PostgreSQLErrorMonitor {
    private static readonly string ConnectionString =
"Host=<HOST>;Port=<PORT>;Database=<DATABASE>;Username=<USER>;Password=<PAS
SWORD>";
    private static readonly string TeamsWebhookUrl = "<URL_DO_WEBHOOK>";

    static async Task Main(string[] args) {
        while (true) {
            await CheckForErrors();
            await Task.Delay(TimeSpan.FromMinutes(10));
        }
    }

    private static async Task CheckForErrors() {
        string query = "SELECT bot, status FROM sua_tabela WHERE status ILIKE '%erro%' OR
status ILIKE '%warning%'";

        using (var conn = new NpgsqlConnection(ConnectionString)) {
            await conn.OpenAsync();
            using (var cmd = new NpgsqlCommand(query, conn))
            using (var reader = await cmd.ExecuteReaderAsync()) {
                while (await reader.ReadAsync()) {
                    string bot = reader.GetString(0);
                    string status = reader.GetString(1);
                    SaveToLog(bot, status);
                    await SendToTeams(bot, status);
                }
            }
        }
    }

    private static void SaveToLog(string bot, string status) {
```

```
        File.AppendAllText("log_postgresql.txt", $"Bot: {bot} - Erro: {status} - Timestamp:
{DateTime.Now}\n");
    }

    private static async Task SendToTeams(string bot, string status) {
        using (var client = new HttpClient()) {
            var payload = new { text = $"Erro detectado no RPA: {bot}\nStatus: {status}" };
            var content = new
StringContent(Newtonsoft.Json.JsonConvert.SerializeObject(payload), Encoding.UTF8,
"application/json");
            var response = await client.PostAsync(TeamsWebhookUrl, content);
            if (!response.IsSuccessStatusCode) {
                Console.WriteLine("Falha ao enviar para o Teams: " + response.StatusCode);
            }
        }
    }
}
```

MySQL

Adapte a string de conexão:

```
private static readonly string ConnectionString =
"Server=<HOST>;Port=<PORT>;Database=<DATABASE>;Uid=<USER>;Pwd=<PASSWORD>;
";
```

SQL Server

Use o **System.Data.SqlClient**:

```
private static readonly string ConnectionString =
"Server=<HOST>,<PORT>;Database=<DATABASE>;User
Id=<USER>;Password=<PASSWORD>;";
```

Enfim

O monitoramento automatizado de erros em bancos de dados é uma solução indispensável para empresas que buscam alta disponibilidade e operação contínua. Esta implementação oferece:

- **Agilidade**: Detecção rápida de falhas com notificações em tempo real.
- **Confiabilidade**: Registro detalhado em logs para auditoria e análise.
- **Flexibilidade**: Compatibilidade com os principais bancos de dados.

Extensões possíveis:

1. **Monitoramento adicional**: Adicionar alertas para métricas como consumo de CPU, espaço em disco ou conexões ativas.
2. **Dashboard centralizado**: Criar um painel de controle para visualização de todos os eventos.
3. **Integração com outras ferramentas**: Ampliar o uso de notificações para sistemas como Slack ou WhatsApp.

Com esta solução, as organizações podem manter um nível superior de resiliência e garantir que os erros sejam tratados antes que comprometam operações críticas.

EducaCiência FastCode para a comunidade