

Relatório CBD

Membros do grupo:

- Felipe Castilho, a047152
- Juliana Moreira, a047188
- Marta Vieira, a046756

Juliana Moreira

Introdução

O nosso projeto, em conjunto com a disciplina de Desenvolvimento Web I, consiste num sistema de replicação com Loopback para resolução do código (controller, model, repository, etc) e React Admin para *front end* com objetivo de simular um sistema de entregas (Food Delivery).

Este trabalho inclui: BD *master*, BD réplica 1 e BD réplica 2 em MariaDB e um Maxscale como *proxy* na distribuição dos pedidos relativamente ao *read-write*. Além disso, tem um audit, um seed (para criar os dados) e um schema.

Além disso, possui um *front end* em React Admin onde demonstra as tabelas e respetivos dados, bem como um *Dashboard* com esses dados.

Sobre o Projeto

Como este trabalho foi realizado em conjunto com Desenvolvimento Web I, foi feito um relatório completo e detalhado sobre o projeto no geral. Este está dividido em 5 capítulos, sendo que o “c5”, em */doc*, no repositório do GitHub, é o que está destinado ao foco em conteúdos de Complementos de Base de Dados.

O projeto encontra-se neste link:

<https://github.com/inf25dw1g13/FoodDeliveryProject/tree/main>

Relatório completo e detalhado, com a descrição técnica, neste:

<https://github.com/inf25dw1g13/FoodDeliveryProject/tree/main/doc>

Capítulo com conteúdos de Complementos de Base de Dados:

<https://github.com/inf25dw1g13/FoodDeliveryProject/blob/main/doc/c5.md>

(sendo que estes se encontram no ZIP que está no moodle - o ZIP do GitHub finalizado)

Descrição Técnica do Projeto

Arquitetura da Base de Dados:

O projeto implementa uma arquitetura de base de dados robusta e escalável, baseada em:

1. Motor de Base de Dados:

- MariaDB 11.2: Sistema de gestão de base de dados relacional, compatível com MySQL
- InnoDB Storage Engine: Motor de armazenamento transacional que garante ACID
- GTID (Global Transaction ID): Sistema de identificação única de transações para replicação confiável

2. Arquitetura de Replicação:

- Topologia Master-Replica: 1 servidor Master (escritas) + 2 servidores Replica (leituras)
- Replicação Baseada em GTID: Permite failover e rejoin automáticos
- Binary Logging em formato ROW: Replicação linha a linha para maior confiabilidade

3. Proxy de Base de Dados:

- MaxScale 23.08: Proxy que implementa separação de leituras e escritas
- Router ReadWriteSplit: Direciona automaticamente escritas para o Master e leituras para as Rélicas
- Monitor MariaDB-Monitor: Verifica a saúde dos servidores a cada 2 segundos

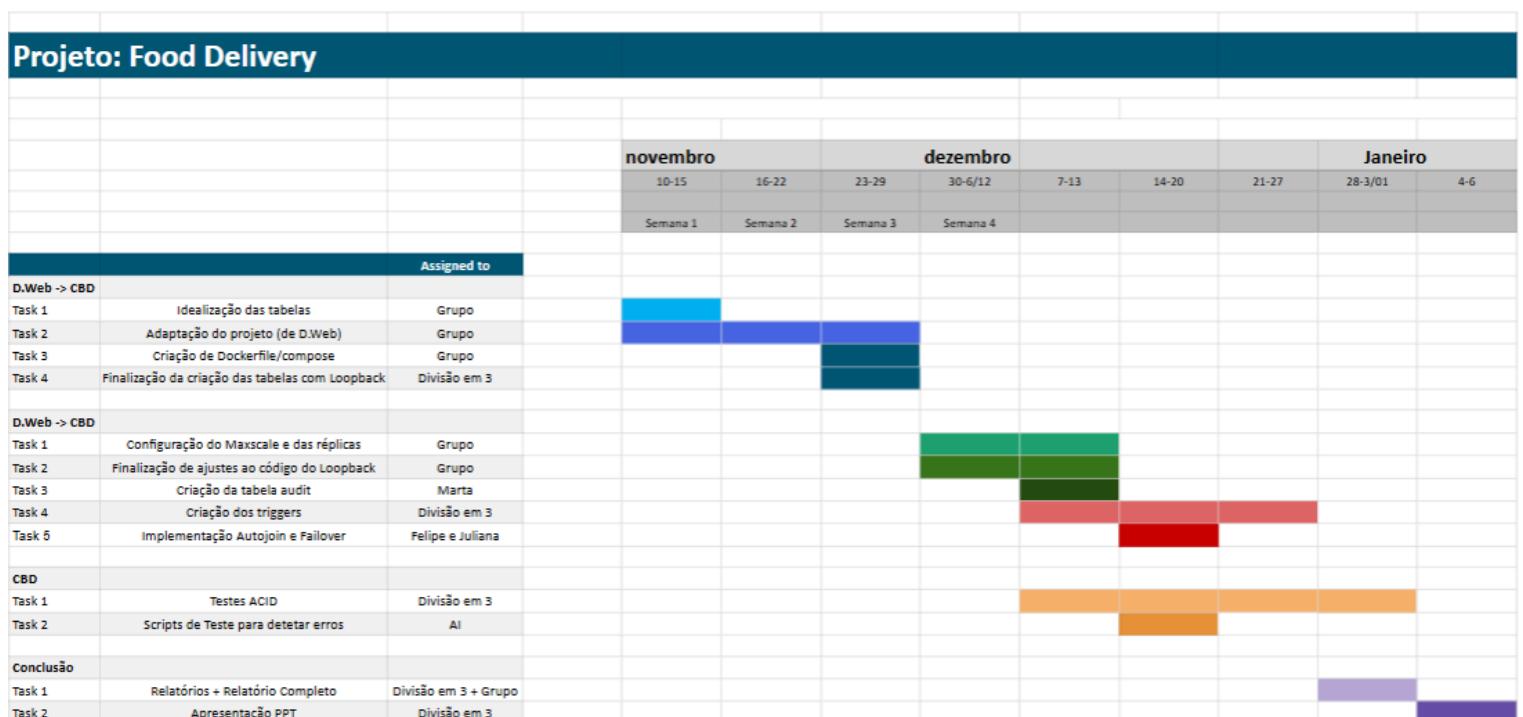
4. Sistema de Auditoria:

- Triggers Automáticos: 33 triggers (3 por tabela: INSERT, UPDATE e DELETE)
- Armazenamento em JSON: Dados armazenados em formato JSON para maior flexibilidade
- *View* de Consulta: View view_audit_log para facilitar consultas

5. Garantias ACID:

- Testes Automatizados: 16 testes em Python que validam todas as propriedades ACID
- 4 testes por propriedade: Cobertura completa de atomicidade, consistência, isolamento e durabilidade

Diagrama de Gantt



Trabalho Individual

Maioritariamente, este projeto foi feito por videochamadas em grupo onde foi possível a realização do mesmo com trabalho de todos. Algo que foi dividido para realização foram as aplicações dos triggers, para audit, e a realização dos testes ACID:

No meu caso:

- triggers: 11 triggers - pedidos, entregadores, entregas e clientes
- testes ACID: 5 (4 atomicidade, 1 durabilidade)
- implementação do *AutoJoin* e *FailOver*

Triggers

pedidos (exemplo do insert)

```
-- TABELA: pedidos
DROP TRIGGER IF EXISTS audit_pedidos_insert//
CREATE TRIGGER audit_pedidos_insert
AFTER INSERT ON pedidos
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_novos)
    VALUES ('pedidos','INSERT', NEW.id,
            JSON_OBJECT('cliente_id', NEW.cliente_id, 'restaurante_id', NEW.restaurante_id,
                        'metodo_pagamento', NEW.metodo_pagamento,
                        'hora_pedido', NEW.hora_pedido, 'total', NEW.total)
            );
END//
```

entregadores (exemplo do update)

```
DROP TRIGGER IF EXISTS audit_entregadores_update//
CREATE TRIGGER audit_entregadores_update
AFTER UPDATE ON entregadores
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores, dados_novos)
    VALUES ('entregadores','UPDATE', OLD.id,
            JSON_OBJECT('nome', OLD.nome, 'email', OLD.email, 'telefone', OLD.telefone,
                        'codpostal', OLD.codpostal, 'estado', OLD.estado),
            JSON_OBJECT('nome', NEW.nome, 'email', NEW.email, 'telefone', NEW.telefone,
                        'codpostal', NEW.codpostal, 'estado', NEW.estado)
            );
END//
```

entregas (exemplo do delete)

```
DROP TRIGGER IF EXISTS audit_entregas_delete//
CREATE TRIGGER audit_entregas_delete
AFTER DELETE ON entregas
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores)
    VALUES ('entregas','DELETE', OLD.id,
            JSON_OBJECT('pedido_id', OLD.pedido_id, 'entregador_id', OLD.entregador_id,
                        'restaurante_id', OLD.restaurante_id, 'tempo_estimado_min', OLD.tempo_estimado_min,
                        'tempo_real_min', OLD.tempo_real_min, 'estado', OLD.estado, 'hora_entrega', OLD.hora_entrega)
            );
END//
```

clientes (exemplo do insert)

```
-- TABELA: clientes
DROP TRIGGER IF EXISTS audit_clientes_insert// 
CREATE TRIGGER audit_clientes_insert
AFTER INSERT ON clientes
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_novos)
    VALUES ('clientes','INSERT', NEW.id,
        JSON_OBJECT('nome', NEW.nome, 'email', NEW.email, 'telefone', NEW.telefone,
        'morada', NEW.morada, 'codpostal', NEW.codpostal)
    );
END//
```

Testes ACID

exemplo do Teste 1 de atomicidade:

```
def test_atomicity_1():
    """Teste 1: Transação completa deve ser commitada"""
    print(f"\n{Colors.BLUE}==> Teste Atomicidade 1: Transação Completa
{Colors.RESET}")
    conn = get_connection()
    if not conn:
        return False

    try:
        cursor = conn.cursor()

        #Iniciar transação
        conn.start_transaction()

        #Criar codpostal se não existir
        cursor.execute("""
            INSERT IGNORE INTO codpostal (codpostal, localidade,
cidade)
            VALUES ('1000-001', 'Lisboa', 'Lisboa')
        """)

        #Obter ou criar categoria
        cursor.execute("SELECT id FROM categorias_pratos WHERE nome =
'Teste' LIMIT 1")
        cat_result = cursor.fetchone()
        if cat_result:
```

```

        categoria_id = cat_result[0]
    else:
        cursor.execute("INSERT INTO categorias_pratos (nome) VALUES ('Teste')")
        categoria_id = cursor.lastrowid

    #Criar restaurante
    cursor.execute("""
        INSERT INTO restaurantes (nome, morada, codpostal, email,
        telefone, especialidade_id)
        VALUES ('Teste Atomic 1', 'Rua Teste', '1000-001',
        'teste.atomic1@test.pt', '999999999', %s)
    """, (categoria_id,))
    restaurante_id = cursor.lastrowid

    #Criar prato associado
    cursor.execute("""
        INSERT INTO pratos (restaurante_id, categoria_id, nome,
        preco)
        VALUES (%s, %s, 'Prato Teste', 10.50)
    """, (restaurante_id, categoria_id))

    #Commit
    conn.commit()

    #Verificar se ambos foram criados
    cursor.execute("SELECT COUNT(*) FROM restaurantes WHERE id = %s",
    (restaurante_id,))
    restaurante_exists = cursor.fetchone()[0] > 0

    cursor.execute("SELECT COUNT(*) FROM pratos WHERE restaurante_id = %s",
    (restaurante_id,))
    prato_exists = cursor.fetchone()[0] > 0

    result = restaurante_exists and prato_exists
    print_test("Atomicidade 1", result,
               f"Restaurante e prato criados: {restaurante_exists} e
{prato_exists}")

    #Remoção
    cursor.execute("DELETE FROM pratos WHERE restaurante_id = %s",
    (restaurante_id,))

```

```

        cursor.execute("DELETE FROM restaurantes WHERE id = %s",
(restaurante_id,))
        conn.commit()

        return result
    except Error as e:
        conn.rollback()
        print_test("Atomicidade 1", False, f"Erro: {e}")
        return False
    finally:
        cursor.close()
        conn.close()

```

durabilidade - teste 1:

```

def test_durability_1():
    """Teste 1: Dados commitados persistem após reconexão"""
    print(f"\n{Colors.BLUE}==== Teste Durabilidade 1: Persistência após
Commit ===={Colors.RESET}")

    conn = get_connection()
    if not conn:
        return False

    try:
        cursor = conn.cursor()

        #Criar codpostal se não existir
        cursor.execute("""
            INSERT IGNORE INTO codpostal (codpostal, localidade,
cidade)
            VALUES ('1000-009', 'Lisboa', 'Lisboa')
        """)

        #Obter ou criar categoria
        cursor.execute("SELECT id FROM categorias_pratos WHERE nome =
'Teste' LIMIT 1")
        cat_result = cursor.fetchone()
        if cat_result:
            categoria_id = cat_result[0]
        else:
            cursor.execute("INSERT INTO categorias_pratos (nome) VALUES
('Teste')")
            categoria_id = cursor.lastrowid

```

```
#Criar e commitar
cursor.execute("""
    INSERT INTO restaurantes (nome, morada, codpostal, email,
telefone, especialidade_id)
        VALUES ('Durability Test 1', 'Rua Teste', '1000-009',
'durability.test1@test.pt', '999999999', %s)
    """, (categoria_id,))
restaurante_id = cursor.lastrowid
conn.commit()

#Fechar conexão
cursor.close()
conn.close()

#Reconectar e verificar
conn2 = get_connection()
cursor2 = conn2.cursor()
cursor2.execute("SELECT COUNT(*) FROM restaurantes WHERE id =
%s", (restaurante_id,))
exists = cursor2.fetchone()[0] > 0

result = exists
print_test("Durabilidade 1", result,
           f"Dados persistem após reconexão: {exists}")

#Remoção
cursor2.execute("DELETE FROM restaurantes WHERE id = %s",
(restaurante_id,))
conn2.commit()

cursor2.close()
conn2.close()
return result
except Error as e:
    conn.rollback()
    print_test("Durabilidade 1", False, f"Erro: {e}")
    return False
finally:
    if conn:
        conn.close()
```

Dificuldades

- criação dos testes ACID: demorado, porém com exemplos foi possível a adaptação.
- criação dos triggers devido à demasia, que também tiveram de ser alterados conforme tivéssemos novas alterações em relação aos dados, bem como, por vezes, erros.
Porém, e apesar de demorado, penso que tenham sido bem implementados para o respetivo propósito.

Conclusão

O projeto reflete bem uma simulação de um sistema de entregas, com dados relevantes relativamente às bases de dados e com um sistema funcional de réplica Master-Slave.