Relatório CBD

Membros do grupo:

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

Felipe Castilho

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, 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 */docs,* 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 Réplicas
- Monitor MariaDB-Monitor: Verifica a saúde dos servidores a cada 2 segundos

4. **Sistema de Auditoria:**

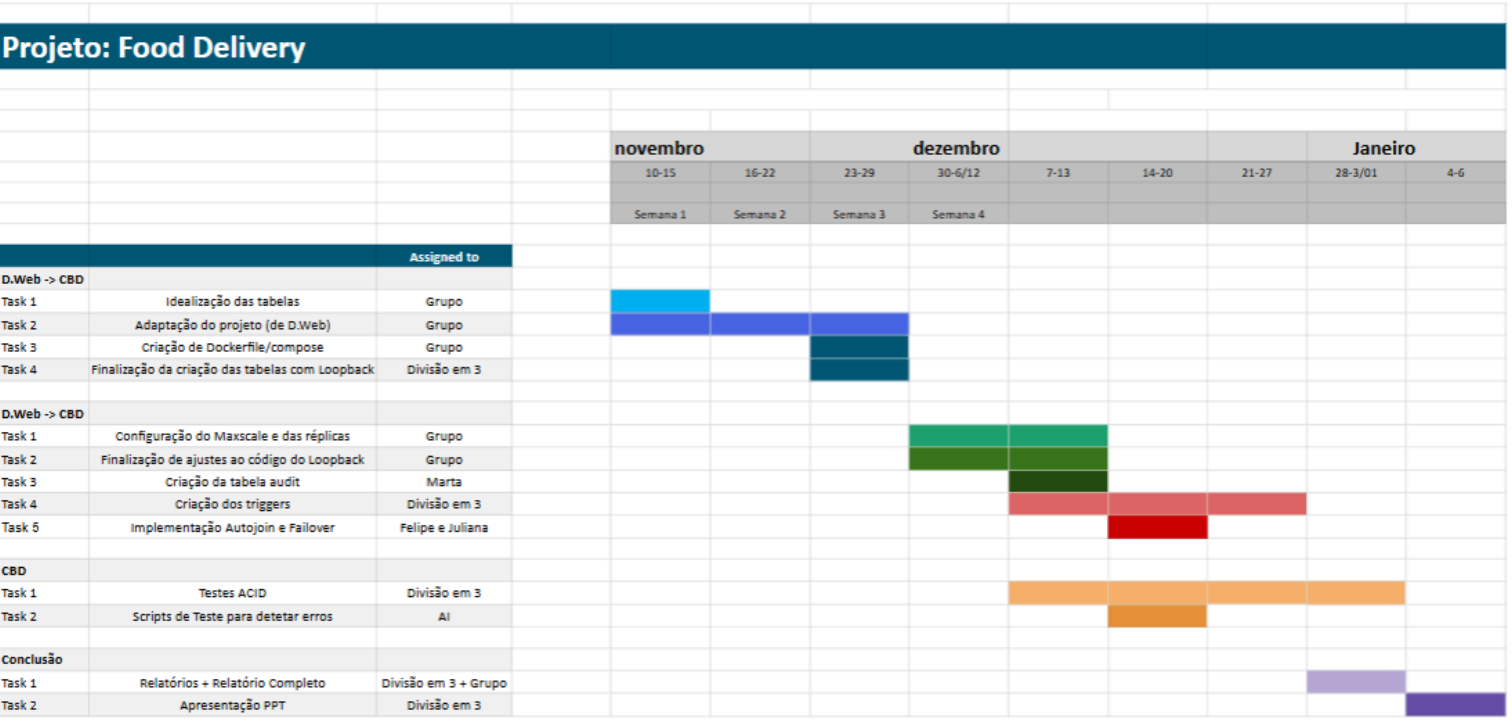
- Triggers Automáticos: 33 triggers (3 por tabela: INSERT, UPDATE e DELETE)
- Armazenamento em JSON: Dados armazenados em formato JSON para maior flexibilidade
- *View* de Consulta: View view_audit_log para facilitar consultas

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



No meu caso:

- Triggers: 12 triggers - restaurantes, pratos, pedidos_pratos e pratos_ingredientes
- Testes ACID: 6 (4 isolamento, 2 consistência)
- Implementação do *AutoJoin* e *FailOver*

Triggers

restaurantes (exemplo do insert)

```
-- TABELA: restaurantes
DROP TRIGGER IF EXISTS audit_restaurantes_insert//
CREATE TRIGGER audit_restaurantes_insert
AFTER INSERT ON restaurantes
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_novos)
    VALUES ('restaurantes','INSERT', NEW.id,
        JSON_OBJECT('nome', NEW.nome, 'morada', NEW.morada, 'codpostal', NEW.codpostal,
                    'email', NEW.email, 'telefone', NEW.telefone, 'especialidade_id', NEW.especialidade_id,
                    'hora_abertura', NEW.hora_abertura, 'hora_fecho', NEW.hora_fecho,
                    'estado', NEW.estado, 'descricao', NEW.descricao, 'taxa_entrega', NEW.taxa_entrega)
    );
END//
```

restaurantes (exemplo do update)

```
DROP TRIGGER IF EXISTS audit_restaurantes_update//
CREATE TRIGGER audit_restaurantes_update
AFTER UPDATE ON restaurantes
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores, dados_novos)
    VALUES ('restaurantes','UPDATE', OLD.id,
        JSON_OBJECT('nome', OLD.nome, 'morada', OLD.morada, 'codpostal', OLD.codpostal,
                    'email', OLD.email, 'telefone', OLD.telefone, 'especialidade_id', OLD.especialidade_id,
                    'hora_abertura', OLD.hora_abertura, 'hora_fecho', OLD.hora_fecho,
                    'estado', OLD.estado, 'descricao', OLD.descricao, 'taxa_entrega', OLD.taxa_entrega),
        JSON_OBJECT('nome', NEW.nome, 'morada', NEW.morada, 'codpostal', NEW.codpostal,
                    'email', NEW.email, 'telefone', NEW.telefone, 'especialidade_id', NEW.especialidade_id,
                    'hora_abertura', NEW.hora_abertura, 'hora_fecho', NEW.hora_fecho,
                    'estado', NEW.estado, 'descricao', NEW.descricao, 'taxa_entrega', NEW.taxa_entrega)
    );
END//
```

restaurantes (exemplo do delete)

```
DROP TRIGGER IF EXISTS audit_restaurantes_delete//
CREATE TRIGGER audit_restaurantes_delete
AFTER DELETE ON restaurantes
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores)
    VALUES ('restaurantes','DELETE', OLD.id,
        JSON_OBJECT('nome', OLD.nome, 'morada', OLD.morada, 'codpostal', OLD.codpostal,
                    'email', OLD.email, 'telefone', OLD.telefone, 'especialidade_id', OLD.especialidade_id,
                    'hora_abertura', OLD.hora_abertura, 'hora_fecho', OLD.hora_fecho,
                    'estado', OLD.estado, 'descricao', OLD.descricao, 'taxa_entrega', OLD.taxa_entrega)
    );
END//
```

pratos (exemplo do insert)

```
-- TABELA: pratos
DROP TRIGGER IF EXISTS audit_pratos_insert//
CREATE TRIGGER audit_pratos_insert
AFTER INSERT ON pratos
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_novos)
    VALUES ('pratos','INSERT', NEW.id,
        JSON_OBJECT('restaurante_id', NEW.restaurante_id, 'categoria_id', NEW.categoria_id,
                    'nome', NEW.nome, 'preco', NEW.preco, 'descricao', NEW.descricao,
                    'disponivel', NEW.disponivel, 'vegetariano', NEW.vegetariano,
                    'vegan', NEW.vegan, 'sem_gluten', NEW.sem_gluten)
    );
END//
```

pratos (exemplo do update)

```
DROP TRIGGER IF EXISTS audit_pratos_update//
CREATE TRIGGER audit_pratos_update
AFTER UPDATE ON pratos
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores, dados_novos)
    VALUES ('pratos','UPDATE', OLD.id,
        JSON_OBJECT('restaurante_id', OLD.restaurante_id, 'categoria_id', OLD.categoria_id,
                    'nome', OLD.nome, 'preco', OLD.preco, 'descricao', OLD.descricao,
                    'disponivel', OLD.disponivel, 'vegetariano', OLD.vegetariano,
                    'vegan', OLD.vegan, 'sem_gluten', OLD.sem_gluten),
        JSON_OBJECT('restaurante_id', NEW.restaurante_id, 'categoria_id', NEW.categoria_id,
                    'nome', NEW.nome, 'preco', NEW.preco, 'descricao', NEW.descricao,
                    'disponivel', NEW.disponivel, 'vegetariano', NEW.vegetariano,
                    'vegan', NEW.vegan, 'sem_gluten', NEW.sem_gluten)
    );
END//
```

pratos (exemplo do delete)

```
DROP TRIGGER IF EXISTS audit_pratos_delete//
CREATE TRIGGER audit_pratos_delete
AFTER DELETE ON pratos
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores)
    VALUES ('pratos','DELETE', OLD.id,
        JSON_OBJECT('restaurante_id', OLD.restaurante_id, 'categoria_id', OLD.categoria_id,
                    'nome', OLD.nome, 'preco', OLD.preco, 'descricao', OLD.descricao,
                    'disponivel', OLD.disponivel, 'vegetariano', OLD.vegetariano,
                    'vegan', OLD.vegan, 'sem_gluten', OLD.sem_gluten)
    );
END//
```

pedidos_pratos (exemplo do insert)

```
-- TABELA: pedidos_pratos
DROP TRIGGER IF EXISTS audit_pedidos_pratos_insert//
CREATE TRIGGER audit_pedidos_pratos_insert
AFTER INSERT ON pedidos_pratos
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_novos)
    VALUES ('pedidos_pratos','INSERT', NEW.id,
        JSON_OBJECT('pedido_id', NEW.pedido_id, 'prato_id', NEW.prato_id, 'quantidade', NEW.quantidade)
    );
END//
```

pedidos_pratos (exemplo do update)

```
DROP TRIGGER IF EXISTS audit_pedidos_pratos_update//
CREATE TRIGGER audit_pedidos_pratos_update
AFTER UPDATE ON pedidos_pratos
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores, dados_novos)
    VALUES ('pedidos_pratos','UPDATE', OLD.id,
        JSON_OBJECT('pedido_id', OLD.pedido_id, 'prato_id', OLD.prato_id, 'quantidade', OLD.quantidade),
        JSON_OBJECT('pedido_id', NEW.pedido_id, 'prato_id', NEW.prato_id, 'quantidade', NEW.quantidade)
    );
END//
```

pedidos_pratos (exemplo do delete)

```
DROP TRIGGER IF EXISTS audit_pedidos_pratos_delete//
CREATE TRIGGER audit_pedidos_pratos_delete
AFTER DELETE ON pedidos_pratos
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores)
    VALUES ('pedidos_pratos','DELETE', OLD.id,
        JSON_OBJECT('pedido_id', OLD.pedido_id, 'prato_id', OLD.prato_id, 'quantidade', OLD.quantidade)
    );
END//
```

pratos_ingredientes (exemplo do insert)

```
-- TABELA: pratos_ingredientes
DROP TRIGGER IF EXISTS audit_pratos_ingredientes_insert//
CREATE TRIGGER audit_pratos_ingredientes_insert
AFTER INSERT ON pratos_ingredientes
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_novos)
    VALUES ('pratos_ingredientes','INSERT', NEW.id,
        JSON_OBJECT('prato_id', NEW.prato_id, 'ingrediente_id', NEW.ingrediente_id,
                    'quantidade', NEW.quantidade, 'obrigatorio', NEW.obrigatorio, 'unidade', NEW.unidade)
    );
END//
```

pratos_ingredientes (exemplo do update)

```
DROP TRIGGER IF EXISTS audit_pratos_ingredientes_update//
CREATE TRIGGER audit_pratos_ingredientes_update
AFTER UPDATE ON pratos_ingredientes
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores, dados_novos)
    VALUES ('pratos_ingredientes','UPDATE', OLD.id,
        JSON_OBJECT('prato_id', OLD.prato_id, 'ingrediente_id', OLD.ingrediente_id,
                    'quantidade', OLD.quantidade, 'obrigatorio', OLD.obrigatorio, 'unidade', OLD.unidade),
        JSON_OBJECT('prato_id', NEW.prato_id, 'ingrediente_id', NEW.ingrediente_id,
                    'quantidade', NEW.quantidade, 'obrigatorio', NEW.obrigatorio, 'unidade', NEW.unidade)
    );
END//
```

pratos_ingredientes (exemplo do delete)

```
DROP TRIGGER IF EXISTS audit_pratos_ingredientes_delete//
CREATE TRIGGER audit_pratos_ingredientes_delete
AFTER DELETE ON pratos_ingredientes
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anteriores)
    VALUES ('pratos_ingredientes','DELETE', OLD.id,
        JSON_OBJECT('prato_id', OLD.prato_id, 'ingrediente_id', OLD.ingrediente_id,
                    'quantidade', OLD.quantidade, 'obrigatorio', OLD.obrigatorio, 'unidade', OLD.unidade)
    );
END//
```

Testes ACID

Exemplo do Teste 1 de Isolamento:

```
def test_isolation_1():
    """Teste 1: Transações não commitadas não são visíveis"""
    print(f"\n{Colors.BLUE}=== Teste Isolamento 1: Dirty Read
==={Colors.RESET}")
    conn1 = get_connection()
    conn2 = get_connection()

    if not conn1 or not conn2:
        return False

    try:
        cursor1 = conn1.cursor()
        cursor2 = conn2.cursor()

        #Criar codpostal se não existir (fora da transação)
        cursor1.execute("""
            INSERT IGNORE INTO codpostal (codpostal, localidade, cidade)
            VALUES ('1000-006', 'Lisboa', 'Lisboa')
        """)
        conn1.commit()

        #Obter ou criar categoria (fora da transação)
        cursor1.execute("SELECT id FROM categorias_pratos WHERE nome =
'Teste' LIMIT 1")
        cat_result = cursor1.fetchone()
        if cat_result:
            categoria_id = cat_result[0]
        else:
            cursor1.execute("INSERT INTO categorias_pratos (nome) VALUES
('Teste')")
            conn1.commit()
            categoria_id = cursor1.lastrowid

        #Transação 1: Criar restaurante mas não commitar
        #(autocommit=False já inicia transação implícita)
        cursor1.execute("""
            INSERT INTO restaurantes (nome, morada, codpostal, email,
telefone, especialidade_id)
            VALUES ('Isolation Test 1', 'Rua Teste', '1000-006',
'isolation.test1@test.pt', '666666666', %s)
        """, (categoria_id,))
```

```
        restaurante_id = cursor1.lastrowid

        #Transação 2: Tentar ler (não deve ver)
        cursor2.execute("SELECT COUNT(*) FROM restaurantes WHERE id = %s",
                        (restaurante_id,))
        count = cursor2.fetchone()[0]

        #Rollback transação 1
        conn1.rollback()

        result = count == 0
        print_test("Isolamento 1", result,
                   f"Transação não commitada não é visível: {count == 0}")

        return result
    except Error as e:
        conn1.rollback()
        print_test("Isolamento 1", False, f"Erro: {e}")
        return False
    finally:
        cursor1.close()
        cursor2.close()
        conn1.close()
        conn2.close()
```

Os Restantes testes ACID podem ser encontrado no path do projeto:

Food_Delivery_Final\Food_Delivery_Project\extra\test_acid.py

Dificuldades

- Testes ACID: demorado, porém com exemplos foi possível a adaptação.

- Triggers: pelo fato de serem executados no “background”, então rastrear erros ou efeitos colaterais, como dados alterados sem perceber, é complicado. Além de impactar na performance, porque ao ter muitos triggers em operações frequentes podem deixar o sistema mais lento.

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.