A8: Gatilhos da Base de Dados

Ao tentar adicionar um utilizador a um projeto terminado, retorna uma mensagem de erro.

```
CREATE FUNCTION checkClosedProject() RETURNS TRIGGER AS $checkClosedProject$
BEGIN
    IF project.dateClose IS NOT NULL THEN
        RAISE EXCEPTION 'cannot add elements to closed project';
    END IF;
END;
$checkClosedProject$ LANGUAGE plpgsql;
CREATE TRIGGER checkClosedProject BEFORE INSERT ON participation
FOR EACH ROW EXECUTE PROCEDURE checkClosedProject();
```

Ao tentar adicionar um utilizador a uma tarefa fechada, retorna uma mensagem de erro.

```
CREATE FUNCTION checkClosedTask() RETURNS TRIGGER AS $checkClosedTask$
BEGIN
    IF task.dateCompleted IS NOT NULL OR project.dateClose IS NOT NULL THEN
        RAISE EXCEPTION 'cannot add elements to closed task';
    END IF;
END;
$checkClosedTask$ LANGUAGE plpgsql;
CREATE TRIGGER checkClosedTask BEFORE INSERT ON taskParticipation
FOR EACH ROW EXECUTE PROCEDURE checkClosedTask();
```

Se o utilizador se tentar bloquear a si próprio, é retornada uma mensagem de erro.

```
CREATE FUNCTION checkBlock() RETURNS TRIGGER AS $checkBlock$
BEGIN
    IF (blocked.users = blocked.blocked) THEN
        RAISE EXCEPTION 'user cannot block itself';
    END IF;
END;
$checkBlock$ LANGUAGE plpgsql;
CREATE TRIGGER checkBlock BEFORE INSERT ON blocked
EXECUTE PROCEDURE checkBlock();
```

Numa nova inserção na tabela user, pôr, por defeito, o userType como “PORCONFIRMAR” e o perfil como público (boleano a 0)

```
CREATE FUNCTION newUser() RETURNS TRIGGER AS $newUser$
BEGIN
    UPDATE users SET userType = 'PORCONFIRMAR';
    UPDATE users SET privateProfile = 0; --0 - false
END;
```

```
$newUser$ LANGUAGE plpgsql;
CREATE TRIGGER newUser AFTER INSERT ON users
FOR EACH ROW EXECUTE PROCEDURE newUser();
```

Numa nova inserção na tabela task, pôr, por defeito, o progresso a 0 e o status da tarefa como aberta

```
CREATE FUNCTION newTask() RETURNS TRIGGER AS $newTask$
BEGIN
    UPDATE task SET progress = 000;
    UPDATE task SET taskStatus = 'ABERTO';
END;
$newTask$ LANGUAGE plpgsql;
CREATE TRIGGER newTask AFTER INSERT ON task
FOR EACH ROW EXECUTE PROCEDURE newTask();
```

Numa nova inserção na tabela message, pôr, por defeito, como não lida (boleano a 0)

```
CREATE FUNCTION newMessage() RETURNS TRIGGER AS $newMessage$
BEGIN
    UPDATE message SET readStatus = 0; --0 - false
END;
$newMessage$ LANGUAGE plpgsql;
CREATE TRIGGER newMessage AFTER INSERT ON message
FOR EACH ROW EXECUTE PROCEDURE newMessage();
```

Numa nova inserção na tabela admincontact, pôr, por defeito, o status da mensagem como “PENDING”

```
CREATE FUNCTION newAdminContact() RETURNS TRIGGER AS $newAdminContact$
BEGIN
    UPDATE admincontact SET messageStatus = 'PENDENTE';
END;
$newAdminContact$ LANGUAGE plpgsql;
CREATE TRIGGER newAdminContact AFTER INSERT ON admincontact
FOR EACH ROW EXECUTE PROCEDURE newAdminContact();
```

— LBAW1366

- Anterior - A7: Esquema Físico.
- Seguinte - A9: Base de Dados Povoada com Dados.
- Voltar ao projeto.
- Voltar à página inicial.