EBD: Database Specification Component

> Plataforma onde toda a comunidade do curso L.EIC pode discutir e apresentar as suas dúvidas relativas às diferentes cadeiras.

A4: Conceptual Data Model

> O objetivo deste artefacto passa por definir conceptualmente o sistema de gestão de base de dados que servirá de suporte principal à plataforma a desenvolver. Aqui são identificadas as entidades, respetivos atributos e suas restrições, assim como as associações que as entidades têm entre si.

1. Class diagram

O seguinte diagrama UML representa o modelo conceptual do sistema.

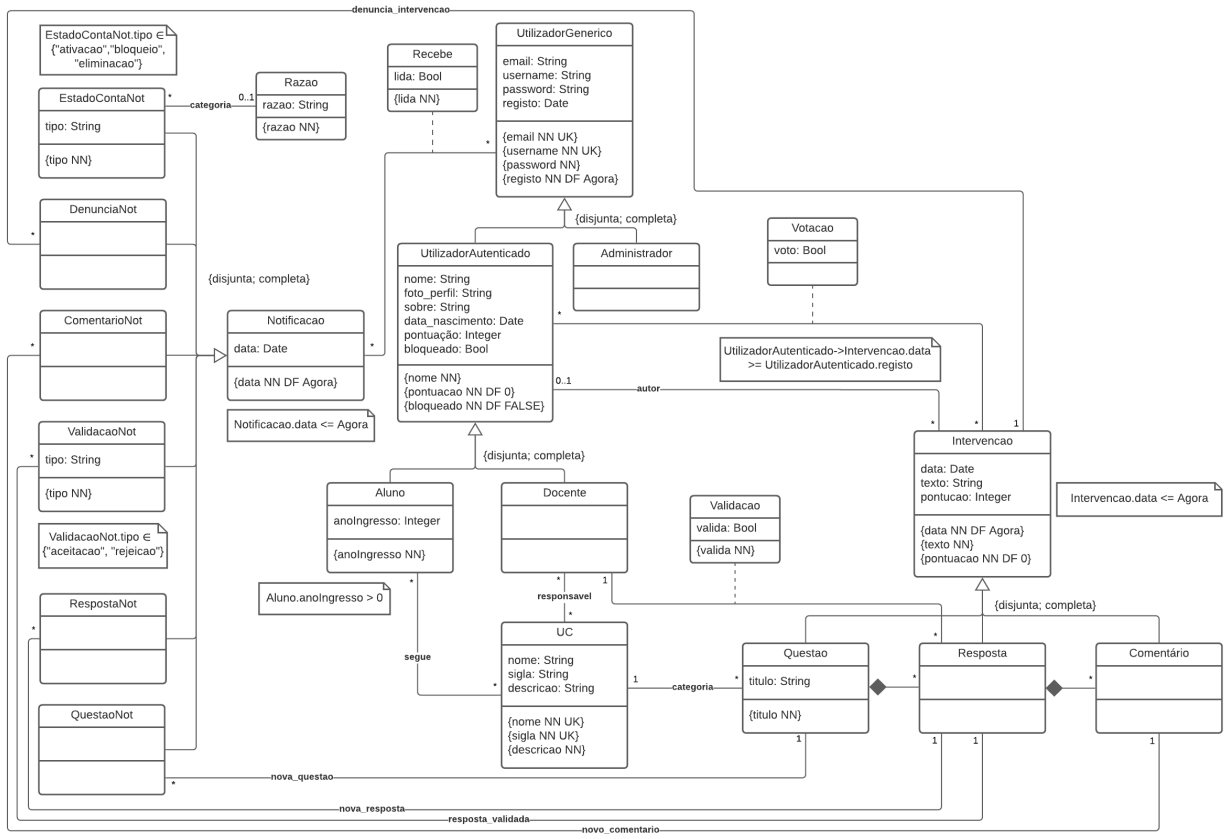

Figure 1: Modelo Conceptual - Diagrama UML

2. Additional Business Rules

Algumas *Business Rules* estão representadas no diagrama UML, enquanto outras encontram-se a seguir indicadas.

- BR01: Um Utilizador nao pode votar nas suas próprias intervenções.
- BR03: A data de uma resposta tem que ser sempre superior à data da respetiva questão.
- BR07: Quando um utilizador é eliminado todas as suas intervenções deixam de estar associadas a um utilizador.

A5: Relational Schema, validation and schema refinement

> Neste artefacto é apresentado o Modelo Relational obtido pela tradução do Modelo Conceptual do nosso sistema de gestão de base de dados, bem como são analisadas as dependências funcionais (FD) de cada relação, verificando se se encontram na BCNF e decompondo-as quando necessário.

1. Relational Schema

O esquema relacional é especificado através da notação compacta e inclui para cada relação os seus atributos, domínios, chaves primárias, chaves estrangeiras e regras de integridade.

Legenda: UK = UNIQUE KEY; NN = NOT NULL; DF = DEFAULT; CK = CHECK

Referência	Relação em Notação Compacta
R01	utilizador(**id**, email UK NN, username UK NN, password NN, data_registo NN DF Agora, tipo CK tipo IN TipoUtilizador, nome CK (tipo<>'Administrador' AND nome IS NN) OR (tipo='Administrador' AND nome IS NULL), foto_perfil CK (tipo='Administrador' AND foto_perfil IS NULL) OR (tipo<>'Administrador'), sobre CK (tipo='Administrador' AND sobre IS NULL) OR (tipo<>'Administrador'), data_nascimento CK (tipo='Administrador' AND data_nascimento IS NULL) OR (tipo<>'Administrador'), pontuacao DF 0 CK (tipo<>'Administrador' AND pontuacao IS NN) OR (tipo='Administrador' AND pontuacao IS NULL), bloqueado DF FALSE CK (tipo<>'Administrador' AND bloqueado IS NN) OR (tipo='Administrador' AND bloqueado IS NULL), ano_ingresso CK (tipo='Aluno' AND ano_ingresso IS NN AND ano_ingresso > 0) OR (tipo<>'Aluno' AND ano_ingresso IS NULL))
R02	uc(**id**, nome UK NN, sigla UK NN, descricao NN)
R03	docente_uc(**id_docente** → utilizador, **id_uc** → uc)
R04	segue_uc(**id_aluno** → utilizador, **id_uc** → uc)
R05	intervencao(**id**, id_autor → utilizador, texto NN, data NN DF Agora CK data <= agora, pontuacao nn df 0, tipo ck in tipointervencao, titulo (tipo="questao" and is nn) or (tipo<>'questao' AND titulo IS NULL), categoria → uc CK (tipo='questao' AND uc IS NN) OR (tipo<>'questao' AND uc IS NULL), id_intervencao → intervencao CK (tipo<>'questao' AND id_intervencao IS NN) OR (tipo='questao' AND id_intervencao IS NULL))
R06	votacao(**id_utilizador** → utilizador, **id_intervencao** → intervencao, voto NN)
R07	validacao(**id_resposta** → intervencao, id_docente → utilizador, valida NN)
R08	razao_bloqueio(**id**, razao UK NN)
R09	notificacao(**id**, data NN DF Agora CK data <= agora, tipo ck in tiponotificacao, id_razao → razao (tipo="estado_conta" and is nn) or (tipo<>'estado_conta' AND id_razao IS NULL), estado CK (tipo='estado_conta' AND estado IS NN AND estado IN TipoEstado) OR (tipo<>'estado_conta' AND estado IS NULL), id_intervencao → intervencao CK (tipo<>'estado_conta' AND id_intervencao IS NN) OR (tipo='estado_conta' AND id_intervencao IS NULL), validacao CK (tipo='validacao' AND validacao IS NN AND validacao IN TipoValidacao) OR (tipo<>'validacao' AND validacao IS NULL)
R10	recebe_not(**id_notificacao** → notificacao, **id_utilizador** → utilizador, lida NN)

Tabela 1: Modelo Relacional em Notação Compacta

Para mapear as generalizações do modelo conceptual para o modelo relacional a estratégia usada foi o estilo *Superclasse* de forma a diminuir a complexidade do número de tabelas e das diferentes entidades que representam e a aumentar a eficiência e performance de leitura do sistema de gestão de base de dados, selecionando quais os campos da tabela que devem ter valor com *checks* de acordo com o tipo desse registo.

2. Domains

Especificação de domínios adicionais que complementam a informação de algumas regras de integridade apresentadas na tabela anterior. São também representados em notação compacta:

Nome do Domínio	Especificação do Domínio
Agora	DATE DEFAULT CURRENT_DATE
TipoUtilizador	ENUM ('Administrador', 'Docente', 'Aluno')
TipoIntervencao	ENUM ('questao', 'resposta', 'comentario')
TipoNotificacao	ENUM ('questao', 'resposta', 'comentario', 'validacao', 'denuncia', 'estado_conta')
TipoEstado	ENUM ('ativacao', 'bloqueio', 'eliminacao')
TipoValidacao	ENUM ('aceitacao', 'rejeicao')

Tabela 2: Domínios em Notação Compacta

3. Schema validation

Como forma de validação do Modelo Relacional obtido e representado anteriormente são identificadas todas as dependências funcionais de cada uma das relações, bem como a Forma Normal em que se encontram.

TABELA R01	utilizador
Chaves	{id}, {email}, {username}
Dependências Funcionais:	
FD0101	{id} → {email, username, password, data_registo, bloqueado, tipo}
FD0102	{email} → {id, nome, foto_perfil, sobre, data_nascimento, pontuacao, ano_ingresso}
FD0103	{username} → {id, nome, foto_perfil, sobre, data_nascimento, pontuacao, ano_ingresso}
FORMA NORMAL	BCNF

Tabela 3: Validacao do Esquema da Relação "utilizador"

TABELA R02	uc
Chaves	{id}, {nome}, {sigla}
Dependências Funcionais:	
FD0201	{id} → {nome, sigla, descricao}
FD0202	{nome} → {id, sigla, descricao}
FD0203	{sigla} → {id, nome, descricao}
FORMA NORMAL	BCNF

Tabela 4: Validacao do Esquema da Relação "uc"

TABELA R03	docente_uc
Chaves	{id_docente, id_uc}
Dependências Funcionais:	*none*
FORMA NORMAL	BCNF

Tabela 5: Validacao do Esquema da Relação "docente_uc"

TABELA R04	segue_uc
Chaves	{id_aluno, id_uc}
Dependências Funcionais:	*none*
FORMA NORMAL	BCNF

Tabela 6: Validacao do Esquema da Relação "segue_uc"

TABELA R05	intervencao
Chaves	{id}
Dependências Funcionais:	
FD0501	{id} → {id_autor, texto, data, pontuacao, tipo, titulo, categoria, id_intervencao}
FORMA NORMAL	BCNF

Tabela 7: Validacao do Esquema da Relação "intervencao"

TABELA R06	votacao
Chaves	{id_utilizador, id_intervencao}
Dependências Funcionais:	
FD0601	{id_utilizador, id_intervencao} → {voto}
FORMA NORMAL	BCNF

Tabela 8: Validacao do Esquema da Relação "votacao"

TABELA R07	validacao
Chaves	{id_resposta}
Dependências Funcionais:	
FD0701	{id_resposta} → {id_docente, valida}
FORMA NORMAL	BCNF

Tabela 9: Validacao do Esquema da Relação "validacao"

TABELA R08	razao_bloqueio
Chaves	{id}, {razao}
Dependências Funcionais:	
FD0801	{id} → {razao}
FD0802	{razao} → {id}
FORMA NORMAL	BCNF

Tabela 10: Validacao do Esquema da Relação "razao_bloqueio"

TABELA R09	notificacao
Chaves	{id}
Dependências Funcionais:	
FD0901	{id} → {data, tipo, id_razao, estado, id_intervencao, validacao}
FORMA NORMAL	BCNF

Tabela 11: Validacao do Esquema da Relação "notificacao"

TABELA R10	recebe_not
Chaves	{id_notificacao, id_utilizador}
Dependências Funcionais:	
FD0601	{id_notificacao, id_utilizador} → {lida}
FORMA NORMAL	BCNF

Tabela 12: Validacao do Esquema da Relação "recebe_not"

CONCLUSÃO: Como todas as relações estão na *Boyce-Codd Normal Form* o esquema relacional também está nesta forma normal.
Assim sendo, não é necessário proceder a nenhuma normalização do esquema relacional.

A6: Indexes, triggers, transactions and database population

> Este artefacto tem como objetivo refletir sobre a eficiência de consultas ao sistema de gestão de base de dados bem como apresentar mecanismos que garantem a integridade dos dados.
>
> É composto pelas seguintes secções: *workload*, *indexes* (identificação e caracterização, separados nas categorias de *Performance* e *Full-text Search*), *triggers* e *transactions*.
>
> Este arfetacto incluí também, em anexo, o código necessário para aplicar as regras das secções anteriores e o código para popular a base de dados.

1. Database Workload

Nesta secção, é analisado e estimado o número de tuplos que cada relação virá a ter assim como o seu crescimento a longo prazo.

Nesta reflexão, foi considerado que a nossa aplicação vai ser de grande uso uma vez que será de interesse para todos os estudantes e representante de cadeiras do curso de L.EIC, mantendo desde as questões mais antigas até às mais atuais.

Referência	Nome da Relação	Ordem de grandeza	Crescimento Estimado
R01	utilizador	10 k	200 / ano
R02	uc	50	1 / ano
R03	docente_uc	250	10 / ano
R04	segue_uc	100 k	10 / dia
R05	intervencao	1 M	100 / dia
R06	votacao	5 M	500 / dia
R07	validacao	1 k	5 / dia
R08	razao_bloqueio	10	sem crescimento
R09	notificacao	1.5 M	100 / dia
R10	recebe_not	10 M	10 k / dia

Tabela 13: "Workload" da base de dados

2. Proposed Indices

2.1. Performance Indices

Este tipo de indices são usados de forma a melhorar a performance de *queries* frequentes, principalmente se está adjacente a manipulação de tabelas de grande dimensão.

Índice	IDX01
Relação	intervencao
Atributo	id_intervencao
Tipo	Hash
Cardinalidade	médio
Clustering	Não
Justificação	A tabela *intervenção* é frequentemente acedida com o objetivo de encontrar a intervenção de ordem superior que lhe corresponde (ex: numa resposta é pretendido encontrar a questão correspondente). Este cruzamento é realizado por correspondência exata, pelo que o tipo *hash* torna-se no mais apropriado neste caso.
`Código SQL`	

```
CREATE INDEX intervencao_superior ON intervencao USING hash (id_intervencao);
```

Tabela 14: "Performance Index" da "intervenção" do campo "id_intervencao"

Índice	IDX02
Relação	intervencao
Atributo	id_autor
Tipo	Hash
Cardinalidade	médio
Clustering	Não
Justificação	A tabela *intervenção* é frequentemente acedida com o objetivo de obter as intervenções realizadas por um determinado utilizador (o autor da intervenção). Esta filtragem é realizada por correspondência exata, pelo que o tipo *hash* torna-se no mais apropriado neste caso. Nenhum *clustering* é proposto uma vez que é expectável uma taxa de atualização elevada.
`Código SQL`	

```
CREATE INDEX autor_intervencao ON intervencao USING hash (id_autor);
```

Tabela 15: "Performance Index" da "intervenção" do campo "id_autor"

Índice	IDX03
Relação	notificacao
Atributo	data
Tipo	B-tree
Cardinalidade	médio
Clustering	Não
Justificação	Cada utilizador pretende ter as suas notificações organizadas por data, mostrando primeiro as notificações mais recentes quando procura pelas mesmas. Assim, uma vez que as comparações não são de igualdade o tipo *B-tree* no mais apropriado para este caso.
`Código SQL`	

```
CREATE INDEX data_notificacao ON notificacao USING btree (data);
```

Tabela 16: "Performance Index" da "notificação"

2.2. Full-text Search Indices

Neste sistema, torna-se importante pesquisar as intervenções (em específico, as do tipo 'Questão') também pelo seu conteúdo, podendo ser este o título e/ou texto descritivo, de forma a, mais facilmente, encontrar as questões que nos interessam no sistema.

Índice	IDX11
Relação	intervencao
Atributo	titulo, texto
Tipo	GIN
Clustering	Não
Justificação	Interessa pesquisar palavras por palavras que possivelmente se encontram no titulo das questões. Com um índice do tipo *GIN* consegue-se ordenar a pesquisa de palavras por ordem de relevância (mais palavras pesadas coincidentes).
`Código SQL`	

```
ALTER TABLE intervencao ADD COLUMN tsvectors TSVECTOR;

CREATE FUNCTION intervencao_procura() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        NEW.tsvectors = (
            setweight(to_tsvector('portuguese', NEW.titulo), 'A') ||
            setweight(to_tsvector('portuguese', NEW.texto), 'B')
        );
    END IF;
    IF TG_OP = 'UPDATE' THEN
        IF (NEW.titulo <> OLD.titulo OR NEW.texto <> OLD.texto) THEN
            NEW.tsvectors = (
```

```
                setweight(to_tsvector('portuguese', NEW.titulo), 'A') ||
                setweight(to_tsvector('portuguese', NEW.texto), 'B')
            );
        END IF;
    END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER intervencao_procura
BEFORE INSERT OR UPDATE ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE intervencao_procura();

CREATE INDEX procura_idx ON intervencao USING GIN (tsvectors);
```

Tabela 17: "FST Index" da "intervenção"

3. Triggers

Triggers são funções e procedimentos que garantem o comprimento de regras de integridade, concretizando, principalmente, as regras de negócio que não conseguem ser definidas nem representadaspelo esquema relacional. Sempre ainda como mecanismos de automização de tarefas de acordo com alterações que acontecem na base de dados (ex.: adição de um novo registo a uma tabela provoca uma alteração do valor de um campo de outra).

Trigger	**TRIGGER01**
Descrição	Um utilizador não pode votar nas suas próprias intervenções (BR01)
`Código SQL`	

```
CREATE FUNCTION proibir_votar_propria_intervencao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF NEW.id_utilizador = (SELECT id_autor FROM intervencao WHERE NEW.id_intervencao=id) THEN
        RAISE EXCEPTION 'Um utilizador não pode votar nas suas próprias intervenções';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER proibir_votar_propria_intervencao
BEFORE INSERT ON votacao
FOR EACH ROW
EXECUTE PROCEDURE proibir_votar_propria_intervencao();
```

Trigger	**TRIGGER02**
Descrição	A data de uma intervençao, no caso de respostas e comentários, tem que ser sempre superior à data da respetiva intervenção que sucede (BR03)
`Código SQL`	

```
CREATE FUNCTION data_maior_intervencao_superior() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF NEW.data <= (SELECT data FROM intervencao WHERE NEW.id_intervencao=id) THEN
        RAISE EXCEPTION 'Uma resposta/comentário não pode ter data inferior à sua respetiva intervenção
                         de ordem superior';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER data_maior_intervencao_superior
BEFORE INSERT OR UPDATE ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE data_maior_intervencao_superior();
```

Trigger	TRIGGER03
Descrição	A pontuação de uma intervenção diferente do tipo 'comentario' é alterada quando ocorre um novo voto, bem como a pontuação do autor da mesma
`Código SQL`	

```
CREATE FUNCTION incr_pontuacao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'comentario'=(SELECT tipo FROM intervencao WHERE id=NEW.id_intervencao) THEN
      RAISE EXCEPTION 'Um comentario não pode ser votado';
    ELSEIF NEW.voto=TRUE THEN
      UPDATE intervencao SET pontuacao=pontuacao+1 WHERE id=NEW.id_intervencao;
      UPDATE utilizador SET pontuacao=pontuacao+1 WHERE id=(SELECT id_autor
                                                            FROM intervencao AS I
                                                            WHERE I.id=NEW.id_intervencao);
    ELSE
      UPDATE intervencao SET pontuacao=pontuacao-1 WHERE id=NEW.id_intervencao;
      UPDATE utilizador SET pontuacao=pontuacao-1 WHERE id=(SELECT id_autor
                                                            FROM intervencao AS I
                                                            WHERE I.id=NEW.id_intervencao);
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER incr_pontuacao
AFTER INSERT ON votacao
FOR EACH ROW
EXECUTE PROCEDURE incr_pontuacao();
```

Trigger	TRIGGER04
Descrição	A pontuação de uma intervenção é altualizada quando o valor de um voto é atualizado
`Código SQL`	

```
CREATE FUNCTION update_pontuacao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF NEW.voto=TRUE AND OLD.voto=FALSE THEN
      UPDATE intervencao SET pontuacao=pontuacao+2 WHERE id=NEW.id_intervencao;
      UPDATE utilizador SET pontuacao=pontuacao+2 WHERE id=(SELECT id_autor
                                                            FROM intervencao AS I
                                                            WHERE I.id=NEW.id_intervencao);
    ELSEIF NEW.voto=FALSE AND OLD.voto=TRUE THEN
      UPDATE intervencao SET pontuacao=pontuacao-2 WHERE id=NEW.id_intervencao;
      UPDATE utilizador SET pontuacao=pontuacao-2 WHERE id=(SELECT id_autor
                                                            FROM intervencao AS I
                                                            WHERE I.id=NEW.id_intervencao);
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER update_pontuacao
AFTER UPDATE OF voto ON votacao
FOR EACH ROW
EXECUTE PROCEDURE update_pontuacao();
```

Trigger	TRIGGER05
Descrição	Só podem ser associados a uma uc utilizadores do tipo docente
`Código SQL`	

```
CREATE FUNCTION verificar_docente_uc() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'Docente' <> (SELECT tipo FROM utilizador WHERE id=NEW.id_docente) THEN
        RAISE EXCEPTION 'Só podem ser associados a uma uc utilizadores do tipo docente';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_docente_uc
BEFORE INSERT ON docente_uc
FOR EACH ROW
EXECUTE PROCEDURE verificar_docente_uc();
```

Trigger	TRIGGER06
Descrição	Só utilizadores do tipo aluno é que podem seguir uma uc
`Código SQL`	

```
CREATE FUNCTION verificar_segue_uc() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'Aluno' <> (SELECT tipo FROM utilizador WHERE id=NEW.id_aluno) THEN
        RAISE EXCEPTION 'Só utilizadores do tipo aluno é que podem seguir uma uc';
    END IF;
```

```
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_segue_uc
BEFORE INSERT ON segue_uc
FOR EACH ROW
EXECUTE PROCEDURE verificar_segue_uc();
```

Trigger	TRIGGER07
Descrição	Administradores não podem ser autores de intervenções
`Código SQL`	

```
CREATE FUNCTION verificar_autor_intervencao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'Administrador' = (SELECT tipo FROM utilizador WHERE id=NEW.id_autor) THEN
        RAISE EXCEPTION 'Administradores não podem ser autores de intervenções';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_autor_intervencao
BEFORE INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE verificar_autor_intervencao();
```

Trigger	TRIGGER08
Descrição	Administradores não podem votar em nenhuma intervenção
`Código SQL`	

```
CREATE FUNCTION verificar_votacao_intervencao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'Administrador' = (SELECT tipo FROM utilizador WHERE id=NEW.id_utilizador) THEN
        RAISE EXCEPTION 'Administradores não podem votar em nenhuma intervenção';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_votacao_intervencao
BEFORE INSERT ON votacao
FOR EACH ROW
EXECUTE PROCEDURE verificar_votacao_intervencao();
```

Trigger	TRIGGER09
Descrição	Só podem ser validadas intervenções do tipo resposta. E por um docente da categoria dessa resposta.
`Código SQL`	

```sql
CREATE FUNCTION verificar_validacao_intervencao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'resposta' <> (SELECT tipo FROM intervencao WHERE id=NEW.id_resposta) THEN
        RAISE EXCEPTION 'Só podem ser validadas intervenções do tipo resposta';
    END IF;
    IF NEW.id_docente NOT IN (SELECT DUC.id_docente
                              FROM (intervencao I1 INNER JOIN intervencao I2 ON I1.id_intervencao = I2.id)
                                                   INNER JOIN docente_uc DUC ON I2.categoria = DUC.id_uc
                              WHERE I1.id = NEW.id_resposta) THEN
        RAISE EXCEPTION 'Apenas docentes da categoria da resposta a podem validar';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_validacao_intervencao
BEFORE INSERT ON validacao
FOR EACH ROW
EXECUTE PROCEDURE verificar_validacao_intervencao();
```

Trigger	TRIGGER10
Descrição	Uma intervenção do tipo 'resposta' ou 'comentario' tem de estar obrigatoriamente associada a uma intervenção do tipo 'questao' ou 'resposta', respetivamente.
`Código SQL`	

```sql
CREATE FUNCTION verificar_associacao_intervencoes() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF NEW.tipo = 'resposta' AND 'questao' <> (SELECT tipo FROM intervencao WHERE id=NEW.id_intervencao) THEN
        RAISE EXCEPTION 'Uma intervenção do tipo "resposta" tem de estar obrigatoriamente associada a uma
                         intervenção do tipo "questao"';
    ELSEIF NEW.tipo = 'comentario' AND 'resposta' <> (SELECT tipo FROM intervencao WHERE id=NEW.id_intervencao) THEN
        RAISE EXCEPTION 'Uma intervenção do tipo "comentario" tem de estar obrigatoriamente associada a uma
                         intervenção do tipo "resposta"';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_associacao_intervencoes
BEFORE INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE verificar_associacao_intervencoes();
```

Trigger	TRIGGER11
Descrição	É gerada uma notificação do tipo 'questao' sempre que é adicionada uma nova questão e esta é redirecionada para todos os utilizadores que seguem ou são docentes da uc que categoriza essa mesma questão.
`Código SQL`	

```
CREATE FUNCTION gerar_notificacao_questao() RETURNS TRIGGER AS
$BODY$
DECLARE
utilizador BIGINT;
notificacaoId BIGINT;
BEGIN
    IF NEW.tipo = 'questao' THEN
        INSERT INTO notificacao(tipo, id_intervencao) VALUES ('questao', NEW.id) RETURNING id INTO notificacaoId;

        FOR utilizador IN (SELECT id_aluno FROM segue_uc WHERE id_uc=NEW.categoria) LOOP
            INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, utilizador, FALSE);
        END LOOP;

        FOR utilizador IN (SELECT id_docente FROM docente_uc WHERE id_uc=NEW.categoria) LOOP
            INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, utilizador, FALSE);
        END LOOP;
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER gerar_notificacao_questao
AFTER INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE gerar_notificacao_questao();
```

Trigger	TRIGGER12
Descrição	É gerada uma notificação do tipo 'resposta' sempre que é adicionada uma nova resposta e esta é redirecionada para o autor da questão a que se está a dar resposta.
`Código SQL`	

```
CREATE FUNCTION gerar_notificacao_resposta() RETURNS TRIGGER AS
$BODY$
DECLARE
autor BIGINT;
notificacaoId BIGINT;
BEGIN
    IF NEW.tipo = 'resposta' THEN
        INSERT INTO notificacao(tipo, id_intervencao) VALUES ('resposta', NEW.id) RETURNING id INTO notificacaoId;

        FOR autor IN (SELECT id_autor FROM intervencao WHERE id=NEW.id_intervencao) LOOP
            INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, autor, FALSE);
        END LOOP;
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER gerar_notificacao_resposta
AFTER INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE gerar_notificacao_resposta();
```

Trigger	TRIGGER13
Descrição	É gerada uma notificação do tipo 'comentario' sempre que é adicionado um novo comentário e esta é redirecionada para o autor da resposta a que se está a dar um comentário.
`Código SQL`	

```
CREATE FUNCTION gerar_notificacao_comentario() RETURNS TRIGGER AS
$BODY$
DECLARE
autor BIGINT;
notificacaoId BIGINT;
BEGIN
    IF NEW.tipo = 'comentario' THEN
        INSERT INTO notificacao(tipo, id_intervencao) VALUES ('comentario', NEW.id) RETURNING id INTO notificacaoId;

        FOR autor IN (SELECT id_autor FROM intervencao WHERE id=NEW.id_intervencao) LOOP
            INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, autor, FALSE);
        END LOOP;
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER gerar_notificacao_comentario
AFTER INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE gerar_notificacao_comentario();
```

Trigger	TRIGGER14
Descrição	É gerada uma notificação do tipo 'validacao' sempre que o uma resposta é validada ou este atributo é modificado e esta é redirecionada para o autor dessa resposta.
`Código SQL`	

```
CREATE FUNCTION gerar_notificacao_validacao() RETURNS TRIGGER AS
$BODY$
DECLARE
autor BIGINT;
notificacaoId BIGINT;
BEGIN
    IF NEW.valida = TRUE THEN
        INSERT INTO notificacao(tipo, id_intervencao, validacao)
            VALUES ('validacao', NEW.id_resposta, 'aceitacao') RETURNING id INTO notificacaoId;
    ELSE
        INSERT INTO notificacao(tipo, id_intervencao, validacao)
            VALUES ('validacao', NEW.id_resposta, 'rejeicao') RETURNING id INTO notificacaoId;
    END IF;
```

```
    FOR autor IN (SELECT id_autor FROM intervencao WHERE id=NEW.id_resposta) LOOP
        INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, autor, FALSE);
    END LOOP;

    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER gerar_notificacao_validacao
AFTER INSERT OR UPDATE OF valida ON validacao
FOR EACH ROW
EXECUTE PROCEDURE gerar_notificacao_validacao();
```

4. Transactions

Transações são necessárias sempre que existem acessos concorrentes que põem em causa a intergidade dos dados do sistema.

Transação	TRAN01
Justificação	Ao obter o número de intervenções do tipo questão bem como as informações sobre cada uma pode ocorrer a inserção de novas intervenções ou a eliminação de outras, pelo que a informação que seria retornada por ambos os *selects* possivelmente diferiria. É *READ-ONLY* uma vez que só usa *selects*
Nível de Isolamento	*SERIALIZABLE READ ONLY*
`Código SQL Completo`	

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY;

SELECT COUNT(*)
FROM intervencao
WHERE tipo='questao';

SELECT titulo, categoria, pontuacao, data, (SELECT COUNT(*) FROM validacao
                                            WHERE id_resposta IN
                                                (SELECT id FROM intervencao WHERE id_intervencao=I.id)
                                                AND valida = TRUE) AS n_respostas_validadas
FROM intervencao AS I
WHERE tipo='questao';

END TRANSACTION;
```

Annex A. SQL Code

Os *scripts* de criação e população da base de dados (*SQL Code*) são incluídos como anexos nesta secção.

A.1. Database schema

O *script* de criação da base de dados incluí o código necessário à construção das relações que constituem o esquema relacional, assegurando, igualmente, as regras de integridade indicadas para cada relação do esquema.

Link para o ficheiro no repositário: Script Criacao

```
DROP SCHEMA IF EXISTS lbaw2185  CASCADE;
CREATE SCHEMA lbaw2185;

SET search_path TO lbaw2185;

-----------------------------------------
-- Types
-----------------------------------------

CREATE TYPE "tipo_utilizador" AS ENUM ('Administrador', 'Docente', 'Aluno');

CREATE TYPE "tipo_intervencao" AS ENUM ('questao', 'resposta', 'comentario');

CREATE TYPE "tipo_notificacao" AS ENUM ('questao', 'resposta', 'comentario', 'validacao', 'denuncia', 'estado_conta');

CREATE TYPE "tipo_estado" AS ENUM ('ativacao', 'bloqueio', 'eliminacao');

CREATE TYPE "tipo_validacao" AS ENUM ('aceitacao', 'rejeicao');

-----------------------------------------
-- Tables
-----------------------------------------

CREATE TABLE "utilizador" (
    id            SERIAL PRIMARY KEY,
    email         TEXT NOT NULL CONSTRAINT email_uk UNIQUE ,
    username      TEXT NOT NULL CONSTRAINT username_uk UNIQUE,
    password      TEXT NOT NULL,
    data_registo  TIMESTAMP  NOT NULL DEFAULT now(),
    nome          TEXT,
    foto_perfil   TEXT,
    sobre         TEXT,
    data_nascimento TIMESTAMP ,
    pontuacao     INTEGER DEFAULT 0,
    bloqueado     BOOLEAN DEFAULT FALSE,
    ano_ingresso  INTEGER,
    tipo tipo_utilizador NOT NULL,

    CONSTRAINT nome_NN         CHECK ((tipo='Administrador' AND nome IS NULL)
                                  OR (tipo<>'Administrador' AND nome IS NOT NULL)),
    CONSTRAINT foto_perfil_NN  CHECK ((tipo='Administrador' AND foto_perfil IS NULL)
                                  OR (tipo<>'Administrador')),
    CONSTRAINT sobre_NN        CHECK ((tipo='Administrador' AND sobre IS NULL)
                                  OR (tipo<>'Administrador')),
    CONSTRAINT data_nasc_NN    CHECK ((tipo='Administrador' AND data_nascimento IS NULL)
                                  OR (tipo<>'Administrador')),
    CONSTRAINT bloqueado_NN    CHECK ((tipo='Administrador' AND bloqueado IS NULL)
                                  OR (tipo<>'Administrador' AND bloqueado IS NOT NULL)),
    CONSTRAINT pontuacao_NN    CHECK ((tipo='Administrador' AND pontuacao IS NULL)
                                  OR (tipo<>'Administrador' AND pontuacao IS NOT NULL)),
    CONSTRAINT ano_ingresso_NN CHECK ((tipo<>'Aluno' AND ano_ingresso IS NULL)
                                  OR (tipo='Aluno' AND ano_ingresso IS NOT NULL AND ano_ingresso > 0))
);

CREATE TABLE "uc" (
    id        SERIAL PRIMARY KEY,
```

```
    nome      TEXT NOT NULL CONSTRAINT nome_uk UNIQUE,
    sigla     TEXT NOT NULL CONSTRAINT sigla_uk UNIQUE,
    descricao TEXT NOT NULL
);

CREATE TABLE "docente_uc" (
    id_docente  INTEGER REFERENCES utilizador ON DELETE CASCADE ON UPDATE CASCADE,
    id_uc       INTEGER REFERENCES uc ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (id_docente, id_uc)
);

CREATE TABLE "segue_uc" (
    id_aluno    INTEGER REFERENCES utilizador ON DELETE CASCADE ON UPDATE CASCADE,
    id_uc       INTEGER REFERENCES uc ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (id_aluno, id_uc)
);

CREATE TABLE "intervencao" (
    id              SERIAL PRIMARY KEY,
    id_autor        INTEGER REFERENCES utilizador ON DELETE SET NULL ON UPDATE CASCADE,
    texto           TEXT NOT NULL,
    data            TIMESTAMP  NOT NULL DEFAULT now(),
    pontuacao       INTEGER NOT NULL DEFAULT 0,
    titulo          TEXT,
    categoria       INTEGER REFERENCES uc ON DELETE CASCADE ON UPDATE CASCADE,
    id_intervencao  INTEGER REFERENCES intervencao ON DELETE CASCADE ON UPDATE CASCADE,
    tipo tipo_intervencao NOT NULL,

    CONSTRAINT data_menor_agora   CHECK (data <= now()),
    CONSTRAINT titulo_categ_NN    CHECK ((tipo ='questao' AND titulo IS NOT NULL AND categoria IS NOT NULL)
                                      OR (tipo<>'questao' AND titulo IS NULL AND categoria IS NULL)),
    CONSTRAINT id_intervencao_NN  CHECK ((tipo<>'questao' AND id_intervencao IS NOT NULL)
                                      OR (tipo='questao' AND id_intervencao IS NULL))
);

CREATE TABLE "votacao" (
    id_utilizador   INTEGER REFERENCES utilizador ON DELETE SET NULL ON UPDATE CASCADE,
    id_intervencao  INTEGER REFERENCES intervencao ON DELETE CASCADE ON UPDATE CASCADE,
    voto            BOOLEAN NOT NULL,
    PRIMARY KEY (id_utilizador, id_intervencao)
);

CREATE TABLE "validacao" (
    id_resposta INTEGER PRIMARY KEY REFERENCES intervencao ON DELETE CASCADE ON UPDATE CASCADE,
    id_docente  INTEGER REFERENCES utilizador ON DELETE SET NULL ON UPDATE CASCADE,
    valida      BOOLEAN NOT NULL
);


CREATE TABLE "razao_bloqueio" (
    id      SERIAL PRIMARY KEY,
    razao   TEXT NOT NULL CONSTRAINT razao_uk UNIQUE
);

CREATE TABLE "notificacao" (
    id              SERIAL PRIMARY KEY,
    data            TIMESTAMP  NOT NULL DEFAULT now(),
    id_razao        INTEGER REFERENCES razao_bloqueio ON DELETE RESTRICT ON UPDATE CASCADE,
    id_intervencao  INTEGER REFERENCES intervencao ON DELETE CASCADE ON UPDATE CASCADE,
    estado tipo_estado,
    validacao tipo_validacao,
    tipo tipo_notificacao NOT NULL,

    CONSTRAINT data_menor_agora CHECK (data <= now()),
    CONSTRAINT razao_estado_NN  CHECK ((tipo='estado_conta' AND id_razao IS NOT NULL AND estado IS NOT NULL)
                                    OR (tipo<>'estado_conta' AND id_razao IS NULL AND estado IS NULL)),
    CONSTRAINT intervencao_NN   CHECK ((tipo<>'estado_conta' AND id_intervencao IS NOT NULL)
                                    OR (tipo='estado_conta' AND id_intervencao IS NULL)),
    CONSTRAINT validacao_NN     CHECK ((tipo='validacao' AND validacao IS NOT NULL)
                                    OR (tipo<>'validacao' AND validacao IS NULL))
);

CREATE TABLE "recebe_not" (
    id_notificacao  INTEGER REFERENCES notificacao ON DELETE CASCADE ON UPDATE CASCADE,
```

```
    id_utilizador   INTEGER REFERENCES utilizador ON DELETE CASCADE ON UPDATE CASCADE,
    lida            BOOLEAN NOT NULL,
    PRIMARY KEY (id_notificacao, id_utilizador)
);

-----------------------------------------
-- Indexes
-----------------------------------------

CREATE INDEX intervencao_superior ON intervencao USING hash (id_intervencao);

CREATE INDEX autor_intervencao ON intervencao USING hash (id_autor);

CREATE INDEX data_notificacao ON notificacao USING btree (data);

-- FTS Index

ALTER TABLE intervencao ADD COLUMN tsvectors TSVECTOR;

CREATE FUNCTION intervencao_procura() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        NEW.tsvectors = (
            setweight(to_tsvector('portuguese', NEW.titulo), 'A') ||
            setweight(to_tsvector('portuguese', NEW.texto), 'B')
        );
    END IF;
    IF TG_OP = 'UPDATE' THEN
        IF (NEW.titulo <> OLD.titulo OR NEW.texto <> OLD.texto) THEN
            NEW.tsvectors = (
                setweight(to_tsvector('portuguese', NEW.titulo), 'A') ||
                setweight(to_tsvector('portuguese', NEW.texto), 'B')
            );
        END IF;
    END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER intervencao_procura
BEFORE INSERT OR UPDATE ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE intervencao_procura();

CREATE INDEX procura_idx ON intervencao USING GIN (tsvectors);

-----------------------------------------
-- TRIGGERS and UDFs
-----------------------------------------

-- TRIGGER01
CREATE FUNCTION proibir_votar_propria_intervencao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF NEW.id_utilizador = (SELECT id_autor FROM intervencao WHERE NEW.id_intervencao=id) THEN
        RAISE EXCEPTION 'Um utilizador não pode votar nas suas próprias intervenções';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER proibir_votar_propria_intervencao
BEFORE INSERT ON votacao
FOR EACH ROW
EXECUTE PROCEDURE proibir_votar_propria_intervencao();

-- TRIGGER02
CREATE FUNCTION data_maior_intervencao_superior() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF NEW.data <= (SELECT data FROM intervencao WHERE NEW.id_intervencao=id) THEN
        RAISE EXCEPTION 'Uma resposta/comentário não pode ter data inferior à sua respetiva
                         intervenção de ordem superior';
```

```
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER data_maior_intervencao_superior
BEFORE INSERT OR UPDATE ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE data_maior_intervencao_superior();

-- TRIGGER03
CREATE FUNCTION incr_pontuacao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'comentario'=(SELECT tipo FROM intervencao WHERE id=NEW.id_intervencao) THEN
      RAISE EXCEPTION 'Um comentario não pode ser votado';
    ELSEIF NEW.voto=TRUE THEN
      UPDATE intervencao SET pontuacao=pontuacao+1 WHERE id=NEW.id_intervencao;
      UPDATE utilizador SET pontuacao=pontuacao+1 WHERE id=(SELECT id_autor
                                                            FROM intervencao AS I
                                                            WHERE I.id=NEW.id_intervencao);
    ELSE
      UPDATE intervencao SET pontuacao=pontuacao-1 WHERE id=NEW.id_intervencao;
      UPDATE utilizador SET pontuacao=pontuacao-1 WHERE id=(SELECT id_autor
                                                            FROM intervencao AS I
                                                            WHERE I.id=NEW.id_intervencao);
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER incr_pontuacao
AFTER INSERT ON votacao
FOR EACH ROW
EXECUTE PROCEDURE incr_pontuacao();

-- TRIGGER04
CREATE FUNCTION update_pontuacao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF NEW.voto=TRUE AND OLD.voto=FALSE THEN
      UPDATE intervencao SET pontuacao=pontuacao+2 WHERE id=NEW.id_intervencao;
      UPDATE utilizador SET pontuacao=pontuacao+2 WHERE id=(SELECT id_autor
                                                            FROM intervencao AS I
                                                            WHERE I.id=NEW.id_intervencao);
    ELSEIF NEW.voto=FALSE AND OLD.voto=TRUE THEN
      UPDATE intervencao SET pontuacao=pontuacao-2 WHERE id=NEW.id_intervencao;
      UPDATE utilizador SET pontuacao=pontuacao-2 WHERE id=(SELECT id_autor
                                                            FROM intervencao AS I
                                                            WHERE I.id=NEW.id_intervencao);
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER update_pontuacao
AFTER UPDATE OF voto ON votacao
FOR EACH ROW
EXECUTE PROCEDURE update_pontuacao();

-- TRIGGER05
CREATE FUNCTION verificar_docente_uc() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'Docente' <> (SELECT tipo FROM utilizador WHERE id=NEW.id_docente) THEN
        RAISE EXCEPTION 'Só podem ser associados a uma uc utilizadores do tipo docente';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER verificar_docente_uc
BEFORE INSERT ON docente_uc
FOR EACH ROW
EXECUTE PROCEDURE verificar_docente_uc();

-- TRIGGER06
CREATE FUNCTION verificar_segue_uc() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'Aluno' <> (SELECT tipo FROM utilizador WHERE id=NEW.id_aluno) THEN
        RAISE EXCEPTION 'Só utilizadores do tipo aluno é que podem seguir uma uc';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_segue_uc
BEFORE INSERT ON segue_uc
FOR EACH ROW
EXECUTE PROCEDURE verificar_segue_uc();

-- TRIGGER07
CREATE FUNCTION verificar_autor_intervencao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'Administrador' = (SELECT tipo FROM utilizador WHERE id=NEW.id_autor) THEN
        RAISE EXCEPTION 'Administradores não podem ser autores de intervenções';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_autor_intervencao
BEFORE INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE verificar_autor_intervencao();

-- TRIGGER08
CREATE FUNCTION verificar_votacao_intervencao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'Administrador' = (SELECT tipo FROM utilizador WHERE id=NEW.id_utilizador) THEN
        RAISE EXCEPTION 'Administradores não podem votar em nenhuma intervenção';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_votacao_intervencao
BEFORE INSERT ON votacao
FOR EACH ROW
EXECUTE PROCEDURE verificar_votacao_intervencao();

-- TRIGGER09
CREATE FUNCTION verificar_validacao_intervencao() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF 'resposta' <> (SELECT tipo FROM intervencao WHERE id=NEW.id_resposta) THEN
        RAISE EXCEPTION 'Só podem ser validadas intervenções do tipo resposta';
    END IF;
    IF NEW.id_docente NOT IN (SELECT DUC.id_docente
                              FROM (intervencao I1 INNER JOIN intervencao I2 ON I1.id_intervencao = I2.id)
                                                   INNER JOIN docente_uc DUC ON I2.categoria = DUC.id_uc
                              WHERE I1.id = NEW.id_resposta) THEN
        RAISE EXCEPTION 'Apenas docentes da categoria da resposta a podem validar';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;
```

```sql
CREATE TRIGGER verificar_validacao_intervencao
BEFORE INSERT ON validacao
FOR EACH ROW
EXECUTE PROCEDURE verificar_validacao_intervencao();

-- TRIGGER10
CREATE FUNCTION verificar_associacao_intervencoes() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF NEW.tipo = 'resposta' AND 'questao' <> (SELECT tipo FROM intervencao WHERE id=NEW.id_intervencao) THEN
        RAISE EXCEPTION 'Uma intervenção do tipo "resposta" tem de estar obrigatoriamente associada a uma
                         intervenção do tipo "questao"';
    ELSEIF NEW.tipo = 'comentario' AND 'resposta' <> (SELECT tipo FROM intervencao WHERE id=NEW.id_intervencao) THEN
        RAISE EXCEPTION 'Uma intervenção do tipo "comentario" tem de estar obrigatoriamente associada a uma
                         intervenção do tipo "resposta"';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER verificar_associacao_intervencoes
BEFORE INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE verificar_associacao_intervencoes();

-- TRIGGER11
CREATE FUNCTION gerar_notificacao_questao() RETURNS TRIGGER AS
$BODY$
DECLARE
utilizador BIGINT;
notificacaoId BIGINT;
BEGIN
    IF NEW.tipo = 'questao' THEN
        INSERT INTO notificacao(tipo, id_intervencao) VALUES ('questao', NEW.id) RETURNING id INTO notificacaoId;

        FOR utilizador IN (SELECT id_aluno FROM segue_uc WHERE id_uc=NEW.categoria) LOOP
            INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, utilizador, FALSE);
        END LOOP;

        FOR utilizador IN (SELECT id_docente FROM docente_uc WHERE id_uc=NEW.categoria) LOOP
            INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, utilizador, FALSE);
        END LOOP;
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER gerar_notificacao_questao
AFTER INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE gerar_notificacao_questao();

-- TRIGGER12
CREATE FUNCTION gerar_notificacao_resposta() RETURNS TRIGGER AS
$BODY$
DECLARE
autor BIGINT;
notificacaoId BIGINT;
BEGIN
    IF NEW.tipo = 'resposta' THEN
        INSERT INTO notificacao(tipo, id_intervencao) VALUES ('resposta', NEW.id) RETURNING id INTO notificacaoId;

        FOR autor IN (SELECT id_autor FROM intervencao WHERE id=NEW.id_intervencao) LOOP
            INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, autor, FALSE);
        END LOOP;
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER gerar_notificacao_resposta
AFTER INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE gerar_notificacao_resposta();

-- TRIGGER13
CREATE FUNCTION gerar_notificacao_comentario() RETURNS TRIGGER AS
$BODY$
DECLARE
autor BIGINT;
notificacaoId BIGINT;
BEGIN
    IF NEW.tipo = 'comentario' THEN
        INSERT INTO notificacao(tipo, id_intervencao) VALUES ('comentario', NEW.id) RETURNING id INTO notificacaoId;

        FOR autor IN (SELECT id_autor FROM intervencao WHERE id=NEW.id_intervencao) LOOP
            INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, autor, FALSE);
        END LOOP;
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER gerar_notificacao_comentario
AFTER INSERT ON intervencao
FOR EACH ROW
EXECUTE PROCEDURE gerar_notificacao_comentario();

-- TRIGGER14
CREATE FUNCTION gerar_notificacao_validacao() RETURNS TRIGGER AS
$BODY$
DECLARE
autor BIGINT;
notificacaoId BIGINT;
BEGIN
    IF NEW.valida = TRUE THEN
        INSERT INTO notificacao(tipo, id_intervencao, validacao)
            VALUES ('validacao', NEW.id_resposta, 'aceitacao') RETURNING id INTO notificacaoId;
    ELSE
        INSERT INTO notificacao(tipo, id_intervencao, validacao)
            VALUES ('validacao', NEW.id_resposta, 'rejeicao') RETURNING id INTO notificacaoId;
    END IF;

    FOR autor IN (SELECT id_autor FROM intervencao WHERE id=NEW.id_resposta) LOOP
        INSERT INTO recebe_not(id_notificacao, id_utilizador, lida) VALUES (notificacaoId, autor, FALSE);
    END LOOP;

    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER gerar_notificacao_validacao
AFTER INSERT OR UPDATE OF valida ON validacao
FOR EACH ROW
EXECUTE PROCEDURE gerar_notificacao_validacao();
```

A.2. Database population

O *script* de povoamento da base de dados incluí o código necessário para povoar inicialmente da base de dados.

Link para o ficheiro no repositário: Script Povoamento

```
-----------------------------------------
-- utilizador
-----------------------------------------

-- Administrador
INSERT INTO utilizador (id, email, username, password, tipo, pontuacao, bloqueado, data_registo)
```

```
    VALUES (0, 'jfcunha@fe.up.pt', 'admin', 'admin', 'Administrador', NULL, NULL, '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, pontuacao, bloqueado, data_registo)
    VALUES (1, 'percurso.academico@fe.up.pt', 'secretaria', 'p-academico', 'Administrador', NULL, NULL, '2021-11-01');

-- Docente
INSERT INTO utilizador (id, email, username, password, tipo, nome, sobre, data_registo)
    VALUES (2, 'mbb@fc.up.pt', 'mbb', 'mbb', 'Docente', 'Manuel Bernardo Martins Barbosa', 'My research interests
            lie in Cryptography and Information Security and Formal Verification.', '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, sobre, data_registo)
    VALUES (3, 'jpleal@fc.up.pt', 'jpleal', 'jpleal', 'Docente', 'José Paulo de Vilhena Geraldes Leal', 'Para além
        de professor, interesso-me por escrever livros pedagógicos.', '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, sobre, data_registo)
    VALUES (4, 'ssn@fe.up.pt', 'ssn', 'ssn', 'Docente', 'Sérgio Sobral Nunes', 'I am an Assistant
            Professor at the Department of Informatics Engineering at the Faculty of Engineering of the
            University of Porto (FEUP), and a Senior Researcher at the Centre for Information Systems
            and Computer Graphics at INESC TEC.', '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, sobre, data_registo)
    VALUES (5, 'tbs@fe.up.pt', 'tbs', 'tbs', 'Docente', 'Tiago Boldt Pereira de Sousa', 'Conclui o Mestrado
            em Mestrado Integrado em Engenharia Informática e Computação em 2011 pela Universidade do Porto
            Faculdade de Engenharia. Publiquei 5 artigos em revistas especializadas.', '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, sobre, data_registo)
    VALUES (6, 'amflorid@fc.up.pt', 'amflorid', 'amflorid', 'Docente', 'António Mário da Silva Marcos Florido',
            'Sou investigador e membro da direção do Laboratório de Inteligência Artificial e Ciência de
            Computadores (LIACC) da FCUP.', '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, sobre, data_registo)
    VALUES (7, 'mricardo@fe.up.pt', 'mricardo', 'mricardo', 'Docente', 'Manuel Alberto Pereira Ricardo',
            'Licenciado, Mestre e Doutor (2000) em Engenharia Eletrotécnica e de Computadores, ramo de
            Telecomunicações, pela Faculdade de Engenharia da Universidade do Porto (FEUP).', '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, sobre, data_registo)
    VALUES (8, 'pabranda@fc.up.pt', 'pabranda', 'pabranda', 'Docente', 'Pedro Miguel Alves Brandão',
            'Fiz o meu doutoramento no Computer Laboratory da Univ. de Cambridge sobre o tema de Body Sensor Networks.
            Obtive uma bolsa da Fundação para a Ciência e Tecnologia para suporte ao doutoramento.', '2021-11-01');

-- Aluno
INSERT INTO utilizador (id, email, username, password, tipo, nome, data_nascimento, ano_ingresso, data_registo)
    VALUES ( 9, 'up201805455@fc.up.pt', 'up201805455', 'up201805455', 'Aluno', 'Alexandre Afonso',
            '2000-07-23 11:00:00', 2018, '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, data_nascimento, ano_ingresso, data_registo)
    VALUES (10, 'up201906852@fe.up.pt', 'up201906852', 'up201906852', 'Aluno', 'Henrique Nunes',
            '2001-02-08 13:00:00', 2019, '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, data_nascimento, ano_ingresso, data_registo)
    VALUES (11, 'up201905427@fe.up.pt', 'up201905427', 'up201905427', 'Aluno', 'Patrícia Oliveira',
            '2001-03-19 17:00:00', 2019, '2021-11-01');
INSERT INTO utilizador (id, email, username, password, tipo, nome, data_nascimento, ano_ingresso, data_registo)
    VALUES (12, 'up201805327@fc.up.pt', 'up201805327', 'up201805327', 'Aluno', 'Tiago Antunes',
            '2000-06-10 11:00:00', 2018, '2021-11-01');

-----------------------------------------
-- uc
-----------------------------------------

INSERT INTO uc (id, nome, sigla, descricao)
    VALUES (0, 'Fundamentos de Segurança Informática', 'FSI', 'Visa dotar os estudantes de uma visão abrangente
            dos aspetos de segurança inerentes ao desenvolvimento e operação de sistemas informáticos.');
INSERT INTO uc (id, nome, sigla, descricao)
    VALUES (1, 'Linguagens e Tecnologias Web', 'LTW', 'Desenvolve competências nas linguagens e tecnologias WEB,
            no contexto tecnológico atual, ou que foram determinantes no processo evolutivo da WEB.');
INSERT INTO uc (id, nome, sigla, descricao)
    VALUES (2, 'Laboratório de Bases de Dados e Aplicações Web', 'LBAW', 'Oferece uma perspetiva prática sobre duas
            áreas centrais da engenharia informática: bases de dados e linguagens e tecnologias web.');
INSERT INTO uc (id, nome, sigla, descricao)
    VALUES (3, 'Programação Funcional e em Lógica', 'PFL', 'Os paradigmas de Programação Funcional e de Programação
            em Lógica apresentam abordagens declarativas e baseadas em processos formais de raciocínio à
            programação.');
INSERT INTO uc (id, nome, sigla, descricao)
    VALUES (4, 'Redes de Computadores', 'RC', 'Introduz os estudantes no domínio de conhecimento das redes de
            comunicações: canais de comunicação e controlo da ligação de dados, modelos de erro e atraso...');

-----------------------------------------
-- docente_uc
-----------------------------------------

INSERT INTO docente_uc (id_docente, id_uc) VALUES (2, 0);
INSERT INTO docente_uc (id_docente, id_uc) VALUES (3, 1);
```

```sql
INSERT INTO docente_uc (id_docente, id_uc) VALUES (4, 2);
INSERT INTO docente_uc (id_docente, id_uc) VALUES (5, 2);
INSERT INTO docente_uc (id_docente, id_uc) VALUES (6, 3);
INSERT INTO docente_uc (id_docente, id_uc) VALUES (7, 4);
INSERT INTO docente_uc (id_docente, id_uc) VALUES (8, 4);
INSERT INTO docente_uc (id_docente, id_uc) VALUES (8, 2);

----------------------------------------
-- segue_uc
----------------------------------------

INSERT INTO segue_uc (id_aluno, id_uc) VALUES (9, 2);
INSERT INTO segue_uc (id_aluno, id_uc) VALUES (12, 2);
INSERT INTO segue_uc (id_aluno, id_uc) VALUES (10, 0);
INSERT INTO segue_uc (id_aluno, id_uc) VALUES (10, 3);
INSERT INTO segue_uc (id_aluno, id_uc) VALUES (11, 1);

----------------------------------------
-- intervencao
----------------------------------------

-- questao
INSERT INTO intervencao (id, id_autor, titulo, texto, categoria, data, tipo)
    VALUES (0, 10, 'Como fazer um Wireframe?', 'Não sei que aplicação usar. Qual é a melhor e mais fácil de usar?',
            2, '2021-11-04 13:00:00', 'questao');
INSERT INTO intervencao (id, id_autor, titulo, texto, categoria, data, tipo)
    VALUES (1, 10, 'Sitemap:pode existir ligação componente-página?', 'É correto ligar uma página diretamente a uma
            componente de outra página?', 2, '2021-11-03 13:00:00', 'questao');
INSERT INTO intervencao (id, id_autor, titulo, texto, categoria, data, tipo)
    VALUES (2, 10, 'Qual o nome do professor de LTW?', 'Não consigo entrar no sigarra e preciso mesmo de saber o
            nome do professor...', 1, '2021-11-06 13:00:00', 'questao');

-- resposta
INSERT INTO intervencao (id, id_autor, texto, id_intervencao, data, tipo)
    VALUES (3, 9, 'O invision é o melhor embora não permita por textos por cima de caixas brancas. Há também o
            figma, mas é bastante mais complexo.', 0, '2021-11-05 13:00:00', 'resposta');
INSERT INTO intervencao (id, id_autor, texto, id_intervencao, data, tipo)
    VALUES (4, 12, 'Faz no papel e digitaliza, não há nada melhor.', 0, '2021-11-05 13:00:00', 'resposta');
INSERT INTO intervencao (id, id_autor, texto, id_intervencao, data, tipo)
    VALUES (5, 12, 'José Paulo Leal...', 2, '2021-11-06 13:23:00', 'resposta');
INSERT INTO intervencao (id, id_autor, texto, id_intervencao, data, tipo)
    VALUES (6, 9, 'Nome: José Paulo de Vilhena Geraldes Leal, Email: jpleal@fc.up.pt', 2, '2021-11-06 13:59:00',
            'resposta');

-- comentario
INSERT INTO intervencao (id, id_autor, texto, id_intervencao, data, tipo)
    VALUES (7, 11, 'Outra ferramenta que podes usar é o draw.io', 3, '2021-11-05 14:00:00', 'comentario');
INSERT INTO intervencao (id, id_autor, texto, id_intervencao, data, tipo)
    VALUES (8,  5, 'O figma é o melhor!', 3, '2021-11-05 14:21:00', 'comentario');

----------------------------------------
-- votacao
----------------------------------------

INSERT INTO votacao (id_utilizador, id_intervencao, voto) VALUES (11, 3, TRUE);
INSERT INTO votacao (id_utilizador, id_intervencao, voto) VALUES ( 5, 3, TRUE);
INSERT INTO votacao (id_utilizador, id_intervencao, voto) VALUES ( 9, 2, FALSE);
INSERT INTO votacao (id_utilizador, id_intervencao, voto) VALUES (11, 2, FALSE);
INSERT INTO votacao (id_utilizador, id_intervencao, voto) VALUES (12, 2, FALSE);

----------------------------------------
-- validacao
----------------------------------------

INSERT INTO validacao (id_resposta, id_docente, valida) VALUES (3, 5, TRUE);
INSERT INTO validacao (id_resposta, id_docente, valida) VALUES (5, 3, TRUE);
INSERT INTO validacao (id_resposta, id_docente, valida) VALUES (6, 3, TRUE);

----------------------------------------
-- razao_bloqueio
----------------------------------------

INSERT INTO razao_bloqueio (id, razao) VALUES (0, 'Conteúdo inapropriado');
INSERT INTO razao_bloqueio (id, razao) VALUES (1, 'Conta foi roubada');
```

```
INSERT INTO razao_bloqueio (id, razao) VALUES (2, 'Uso irresponsável do sistema');
INSERT INTO razao_bloqueio (id, razao) VALUES (3, 'Conteúdo ofensivo');
INSERT INTO razao_bloqueio (id, razao) VALUES (4, 'Outro');

-----------------------------------------
-- notificacao
-----------------------------------------

INSERT INTO notificacao (id, data, tipo, id_razao, estado)
    VALUES (0, '2021-11-10 15:00:00', 'estado_conta', 4, 'ativacao');

-----------------------------------------
-- recebe_not
-----------------------------------------

INSERT INTO recebe_not (id_notificacao, id_utilizador, lida) VALUES (0, 8, FALSE);
```

Revision history

Sendo esta a primeira submissão, nenhuma alteração digna de histórico foi realizada.

GROUP2185, 29/11/2021

- Alexandre Afonso, up201805455@fc.up.pt
- Henrique Nunes, up201906852@fe.up.pt
- Patrícia Oliveira, up201905427@fe.up.pt (editor)
- Tiago Antunes, up201805327@fc.up.pt