A7: Verificação de regras de integridade. Índices, gatilhos e procedimentos

O desenvolvimento deste artefacto consiste num estudo da carga da base de dados, na identificação e caracterização dos índices que são utilizados na tabela, na verificação das regras de integridade com gatilhos (*triggers*) e na especificação de procedimentos criados pelo utilizador (*user-defined functions*).

A7.1 Carga da base de dados:

Para a plataforma AutoLeilões, foi feita uma estimativa para o número de tuplos por tabela. É esperado que, com o crescimento do *site*, os tuplos em cada tabela cresçam de forma proporcional umas às outras (à exceção da tabela Marca, que se mantém inalterada).

Tabela	Número	Notas
Membro	4000	Dos quais, cerca de 10 administradores.
MembroBanido	300	Membros não reais, fraudulentos, com comportamento inadequado, etc.
Leilao	5000	A grande parte de membros registados na plataforma utilizam-na apenas para licitar (e não criar leilões). São cerca de 1000 leilões ativos e por volta de 4000 leilões terminados (500 com licitações, 3500 por tempo).
Mensagem	60000	Assume-se uma média de 10 mensagens por leilão e mais cerca de 10000 mensagens enviadas noutros contextos.
Imagem	15000	Assume-se uma média de 3 imagens por leilão.
Marca	50	Cerca de 50 marcas de todas as partes do mundo.
Licitacao	5000	Assume-se uma média de 1 licitação por leilão (vários leilões não são apelativos e outros são demasiado apelativos).
Notificacao	40000	Assume-se uma média de 10 notificações por membro ativo.
Feedback	800	Para cada leilão terminado com licitações (500), existem até dois *feedbacks* (mas nem sempre é obrigatório deixar *feedback*, o que retira cerca de 200 *feedbacks* que ficaram por trocar).
FeedbackComprador	400	Metade dos *feedbacks* atribuídos.
FeedbackLeiloeiro	400	Metade dos *feedbacks* atribuídos.
Preferencia	8000	Assume-se uma média de duas marcas nas preferências por cliente.
Registo	4000	Assume-se uma média de dois leilões registados por cliente.

Tabela 7.1 - Número de tuplos esperados em cada tabela.

A7.2 Interrogações (SELECT) mais utilizadas:

Serão apresentados as consultas mais relevantes da plataforma, sendo avaliada a frequência com que cada uma é utilizada.

Lista de leilões ordenados por data - são esperados muitas chamadas a este *SELECT*.

```
SELECT *
FROM leilao
ORDER BY leilao.datacolocacao
DESC;
```

Lista de imagens de um leilão - são esperados muitas chamadas a este *SELECT*. A variável *$idleilao* é passada como argumento.

```
SELECT link
FROM imagem, leilao
WHERE imagem.idleilao = $idleilao
AND imagem.idleilao = leilao.idleilao;
```

Lista de licitações de uma leilão ordenadas por data e valor - são esperados algumas chamadas a este *SELECT*. A variável *$idleilao* é passada como argumento.

```
SELECT membro.nomeutilizador, licitacao.valor, licitacao.datalicitacao
FROM licitacao, membro
WHERE licitacao.idcliente = membro.idutilizador
AND licitacao.idleilao = $idleilao
ORDER BY licitacao.valor
DESC;
```

Lista de Administradores ordenados alfabeticamente - não são esperados muitas chamadas a este *SELECT*.

```
SELECT nomeutilizador , nomecivil , email , datanascimento
FROM membro
WHERE membro.tipomembro = 'Admin'
ORDER BY nomeutilizador;
```

Lista de Clientes activos ordenados alfabeticamente - não são esperados muitas chamadas a este *SELECT*.

```
SELECT nomeutilizador , nomecivil , email , datanascimento
FROM membro
WHERE membro.tipomembro = 'Cliente'
AND membro.idutilizador NOT IN
    (
    SELECT idmembrobanido
    FROM membrobanido
    )
ORDER BY nomeutilizador;
```

Lista de Membros Banidos ordenados alfabeticamente - não são esperados muitas chamadas a este *SELECT*.

```
SELECT nomeutilizador , nomecivil , email , datanascimento , databanido ,
```

```
duracao , motivo
FROM membro , membrobanido
WHERE membro.idutilizador = membrobanido.idmembrobanido
ORDER BY nomeutilizador;
```

Lista de Notificações de um Cliente - são esperados muitas chamadas a este *SELECT*. A variável *$idutilizador* é passada como argumento.

```
SELECT texto, datanotificacao
FROM notificacao
WHERE idutilizador = $idutilizador;
```

Lista de Mensagens recebidas de um determinado Cliente ordenadas por data - são esperados muitas chamadas a este *SELECT*. A variável *$idutilizador* é passada como argumento.

```
SELECT membro.nomeutilizador, mensagem.texto, mensagem.datamensagem
FROM membro , mensagem
WHERE mensagem.idrecetor = $idutilizador
AND membro.idutilizador = $idutilizador
ORDER BY mensagem.datamensagem
DESC;
```

Lista de Mensagens enviadas de um determinado Cliente ordenadas por data - são esperados algumas chamadas a este *SELECT*. A variável *$idutilizador* é passada como argumento.

```
SELECT membro.nomeutilizador, mensagem.texto, mensagem.datamensagem
FROM membro , mensagem
WHERE mensagem.idemissor = $idutilizador
AND membro.idutilizador = $idutilizador
ORDER BY mensagem.datamensagem
DESC;
```

Lista de leilões que um Cliente criou ordenados por data - são esperados muitas chamadas a este *SELECT*. A variável *$idcliente-leiloeiro* é passada como argumento.

```
SELECT leilao.nome, licitacaobase, datacolocacao, duracao, marca.nome
FROM leilao, marca
WHERE leilao.idleiloeiro = $idcliente-leiloeiro
AND leilao.idmarca = marca.idmarca
ORDER BY datacolocacao
DESC;
```

Lista de leilões onde um Cliente licitou ordenados por data - são esperados muitas chamadas a este *SELECT*. A variável *$idcliente* é passada como argumento.

```
SELECT leilao.nome, MAX(licitacao.valor)
FROM licitacao
INNER JOIN membro ON membro.idutilizador = licitacao.idcliente
INNER JOIN leilao ON licitacao.idleilao = leilao.idleilao
WHERE membro.idutilizador = $idcliente
```

```
GROUP BY leilao.nome;
```

Lista de registos em leilões de um Cliente ordenados por data - são esperados algumas chamadas a este *SELECT*. A variável *$idcliente* é passada como argumento.

```
SELECT nome
FROM leilao, registo
WHERE registo.idleilao = leilao.idleilao
AND registo.idcliente = $idcliente
ORDER BY datacolocacao
DESC;
```

Lista dos Feedbacks de um Cliente enquanto comprador ordenados por data - são esperados muitas chamadas a este *SELECT*. A variável *$idcomprador* é passada como argumento.

```
SELECT membro.nomeutilizador, feedback.valor, feedback.texto,
feedback.datafeedback
FROM membro, feedback, feedbackcomprador
WHERE membro.idutilizador = feedbackcomprador.idcomprador
AND membro.idutilizador = $idcomprador
AND feedback.idfeedback = feedbackcomprador.idfeedback
ORDER BY datafeedback
DESC;
```

Lista dos Feedbacks de um Cliente enquanto leiloeiro ordenados por data - são esperados muitas chamadas a este *SELECT*. A variável *$idleiloeiro* é passada como argumento.

```
SELECT membro.nomeutilizador, feedback.valor, feedback.texto,
feedback.datafeedback
FROM membro, feedback, feedbackleiloeiro
WHERE membro.idutilizador = feedbackleiloeiro.idleiloeiro
AND membro.idutilizador = $idleiloeiro
AND feedback.idfeedback = feedbackleiloeiro.idfeedback
ORDER BY datafeedback
DESC;
```

Lista as Marcas de Carros ordenadas alfabeticamente - não são esperados muitas chamadas a este *SELECT*.

```
SELECT nome
FROM marca
ORDER BY nome;
```

A7.3 Alterações (UPDATE/DELETE) mais utilizadas:

De seguida serão apresentados os *UPDATES* e os *DELETES* mais utilizados no contexto do site, sendo avaliada a frequência com que estes são feitos. Não são aqui incluídas as chamadas a *updates* e *deletes* incluídas nos *triggers* especificados mais em baixo.

Dar privilégios de administração a um cliente - não são esperadas muitas chamadas a este *UPDATE*. As variáveis *$tipoMembro* e *$idUtilizador* são passada como argumento.

```
UPDATE membro
SET tipomembro = $tipoMembro
WHERE idutilizador = $idUtilizador;
```

Editar campos de autenticação de um membro - são esperadas muitas chamadas a este *UPDATE*. As variáveis *$nomeUtilizador*, *$e-mail*, *$password* e *$idUtilizador* são passadas como argumento.

```
UPDATE membro
SET nomeutilizador = $nomeUtilizador, email = $e-mail, password = $password
WHERE (idutilizador = $idUtilizador)
AND NOT EXISTS
    (
    SELECT *
    FROM membro
    WHERE (nomeutilizador = $nomeUtilizador
    OR email = $e-mail)
    AND idutilizador != $idUtilizador
    );
```

Editar campo de descrição de um leilão - são esperadas poucas chamadas a este *UPDATE*. As variáveis *$descricao* e *$idLeilao* são passadas como argumento.

```
UPDATE leilao
SET descricao = $descricao
WHERE (idleilao = $idLeilao);
```

Apagar um membro - são esperadas poucas chamadas a este *DELETE*. A variável *$idUtilizador* é passada como argumento.

```
DELETE FROM membro
WHERE idutilizador = $idUtilizador;
```

Apagar um leilão - são esperadas poucas chamadas a este *DELETE*. A variável *$idLeilao* é passada como argumento.

```
DELETE FROM leilao
WHERE idleilao = $idLeilao;
```

Apagar uma mensagem - são esperadas muitas chamadas a este *DELETE*. A variável *$idMensagem* é passada como argumento.

```
DELETE FROM mensagem
WHERE idmensagem = $idMensagem;
```

Apagar uma imagem - são esperadas algumas chamadas a este *DELETE*. A variável *$idImagem* é passada como argumento.

```
DELETE FROM imagem
WHERE idimagem = $idImagem;
```

Apagar uma notificação - são esperadas muitas chamadas a este *DELETE*. A variável *$idNotificacao* é passada como argumento.

```
DELETE FROM notificacao
WHERE idnotificacao = $idNotificacao;
```

A7.4 Índices (INDEXES) e clustering:

Neste ponto serão analisados os possíveis índices que a base de dados usufruirá para um acesso mais rápido a algumas tabelas. É apresentado o código de criação dos índices e os *clusters* usados para cada tabela. Estes *clusters* são criados com o comando *CLUSTER* do *PostgreSQL*, que irá ordenar as colunas de cada tabela com base na informação disponível no índice de forma a acelerar as pesquisas.

Notas: dado que o *PostgreSQL 9.4.5* cria automaticamente índices para todas as *PRIMARY KEYS*, tivemos apenas de criar o seguinte código (as justificações para o *clustering* estão como comentários). Poderiam ser considerados outros índices, mas o seu uso não é recomendado quando:

- As tabelas são pequenas ou não expansíveis com o tempo (por exemplo, a tabela Marca, que tem tamanho fixo).
- As tabelas contêm UPDATEs ou INSERTs com mais frequência do que SELECTs.
- Não devem ser usados em colunas com um grande número de valores NULL.

```
CREATE INDEX nomeUtilizadorIndex ON Membro USING hash(nomeUtilizador);

CREATE INDEX emailUtilizadorIndex ON Membro USING hash(email);

CREATE INDEX linkImagemIndex ON Imagem USING hash(linkImagem);

CREATE INDEX tipoMembroIndex ON Membro USING hash(tipoMembro);

CREATE INDEX dataColocacaoIndex ON Leilao USING btree(dataColocacao);

CREATE INDEX nomeLeilaoIndex ON Leilao USING btree(nome);

CREATE INDEX motivoIndex ON MembroBanido USING btree(motivo);

CREATE INDEX dataFeedbackIndex ON Feedback USING btree(dataFeedback);

 -- encontra leilões rapidamente pelo seu nome.
CLUSTER leilao USING nomeLeilaoIndex;

 -- encontra membros banidos rapidamente pelo motivo pelo quais foram
banidos.
CLUSTER membroBanido USING motivoIndex;
```

```
-- encontra membros rapidamente pelo seu nome.
CLUSTER membro USING nomeUtilizadorIndex;

-- encontra mensagens rapidamente pelo índice das descrições.
CLUSTER mensagem USING msgidx;

-- encontra feedbacks rapidamente pela sua data de colocação.
CLUSTER feedback USING dataFeedbackIndex;
```

B-trees foram utilizadas para todos os campos que façam comparações (operadores '<' ou '>') ou possíveis pesquisas em intervalos (tal como o nome de um leilão, por exemplo). *Hashs*, por sua vez, foram utilizadas em comparações que utilizam apenas o operador '='. Apresenta-se de seguida a justificação para a existência de cada índice:

Nome	Tipo	Justificação
nomeUtilizadorIndex	Hash	Atributo UNIQUE NOT NULL da tabela Membro.
emailUtilizadorIndex	Hash	Atributo UNIQUE NOT NULL da tabela Membro.
linkImagemIndex	Hash	Atributo UNIQUE NOT NULL da tabela Imagem.
tipoMembroIndex	Hash	Utilizado para comparar (operador '=') o valor tipoMembro com os seus dois valores possíveis ('Admin' ou 'Cliente') em cláusulas do tipo WHERE. Justifica-se ainda a sua existência pelo tamanho esperado da tabela Membro.
dataColocacaoIndex	B-tree	Utilizado muito frequentemente para ordenar a lista de leilões pela data de colocação (ORDER BY Leilao.dataColocacao). Justifica-se ainda a sua existência pelo tamanho esperado da tabela Leilão.
nomeLeilaoIndex	B-tree	Utilizado muito frequentemente para ordenar a lista de leilões utilizando o seu nome (ORDER BY Leilao.nomeLeilao). Justifica-se ainda a sua existência pelo tamanho esperado da tabela Leilão.
motivoIndex	B-tree	Utilizado para encontrar texto no motivo de expulsão de um membro (utilizando a cláusula LIKE). Como o texto é pequeno, não se justifica a implementação de um 'full text search'.
dataFeedbackIndex	B-tree	Utilizado muito frequentemente para ordenar a lista de feedbacks pela data de colocação dos mesmos (ORDER BY Feedback.dataColocacao). Justifica-se ainda a sua existência pelo tamanho esperado da tabela Feedback.

A7.5 Full-Text Search:

A capacidade do PostgreSQL de usar "*full-text search*" será evidenciada nos seguintes campos:

Tabela	Atributo	Tipo
Leilao	descricao	VARCHAR(2500)
Mensagem	texto	VARCHAR(5000)

Como dá para perceber, face à quantidade de texto que cada um destes atributos pode receber, e dada a quantidade de tuplos esperados na base de dados, facilmente *queries* demoriam muito tempo a completarem. Para usar o *full-text search*, serão criados os seguintes índices (do tipo gin, uma vez que os atributos são do tipo *static data*):

```
CREATE INDEX leilaoidx ON Leilao USING gin(to_tsvector('portuguese',
```

```
descricao));
CREATE INDEX msgidx ON Mensagem USING gin(to_tsvector('portuguese', texto));
```

Após criados estes índices, temos as seguintes consultas, que são feitas de forma eficiente em poucos milisegundos:

```
SELECT * FROM Leilao WHERE to_tsvector('portuguese', descricao) @@
to_tsquery('portuguese', 'quilometragem');
```

Neste caso em particular, é feita uma *full text search* na tabela Leilao utilizando o índice leilaoidx pela palavra 'quilometragem'. Para procurarmos em mensagens, podemos fazer, por exemplo:

```
SELECT * FROM Mensagem WHERE to_tsvector('portuguese', texto) @@
to_tsquery('portuguese', 'conteudo da mensagem');
```

A7.6 Gatilhos (triggers):

Nesta secção são apresentados os *triggers* mais complexos que foram criados para responder a cenários que não podem ser alcançados de forma simples. São descritos os principais *triggers*, quando ocorrem e o código usado para os criar. Todos os *triggers* utilizam procedimentos auxiliares que são definidas na secção seguinte (A7.7).

```
CREATE TRIGGER maiorLicitacao
BEFORE INSERT
ON Licitacao
FOR EACH ROW EXECUTE PROCEDURE atualizarLeilao();
```

O *trigger* maiorLicitacao vai atualizar o valor da licitação atual de um leilão para o valor de uma nova licitação (necessariamente mais alto), adicionando uma notificação no cliente que tinha a licitação atual a informar que essa sua licitação foi ultrapassada e uma notificação no leiloeiro a informar que existe uma nova licitação nesse leilão. Este *trigger* utiliza o procedimento *atualizarLeilao()*.

```
CREATE TRIGGER leilaoApagado
BEFORE DELETE
ON Leilao
FOR EACH ROW EXECUTE PROCEDURE notificaLicitacoesRemovidas ();
```

O *trigger* leilaoApagado é responsável por, sempre que um leilão é apagado (porque o leiloeiro apagou o leilão ou porque foi banido), notificar os clientes que licitaram neste leilão de que o leilão já não existe e de que as suas licitações foram apagadas. Este *trigger* utiliza o procedimento *notificaLicitacoesRemovidas()*.

```
CREATE TRIGGER membroBanido
AFTER INSERT
ON MembroBanido
FOR EACH ROW EXECUTE PROCEDURE membroBanido ();
```

O *trigger* membroBanido é responsável por notificar o utilizador do motivo e da duração da sua

expulsão (bem como informar que todos os seus leilões e licitações foram apagados). Para isto, o *trigger* utiliza o procedimento membroBanido(). Nos casos em que o membro banido detinha a maior licitação de um leilão, este *trigger* utiliza ainda o procedimento atualizaMaiorLicitacao() para reverter o valor da licitação mais alta desse leilão para o valor anterior.

```
CREATE TRIGGER mensagem
AFTER INSERT
ON Mensagem
FOR EACH ROW EXECUTE PROCEDURE notificaMensagem ();
```

O *triger* mensagem adiciona uma notificação sempre que um utilizador receber uma mensagem nova de outro utilizador (mencionando o seu nome). Este *trigger* utiliza o procedimento notificaMensagem().

A7.7 Procedimentos para os gatilhos:

Os procedimentos a utilizar são, na sua maioria, do tipo *trigger procedure*, estando todos relacionadas com os *triggers* em cima referidos. Estes procedimentos utilizam *PL/pgSQL* para declarar variáveis e usar estruturas de controlo (*ifs* e *loops*) para resolver cenários complexos do problema.

```
CREATE OR REPLACE FUNCTION atualizarLeilao ()
  RETURNS TRIGGER AS
$$
    BEGIN
        INSERT INTO Notificacao (idUtilizador, texto) VALUES
        ((SELECT idLeiloeiro FROM Leilao WHERE idLeilao = NEW.idLeilao),
            'O cliente ' || (SELECT nomeUtilizador FROM Membro WHERE
idUtilizador = NEW.idCliente) || ' licitou no seu leilao ' || (SELECT nome
FROM Leilao WHERE idLeilao = NEW.idLeilao) || '.');
        IF (NEW.valor > (SELECT valor FROM Licitacao WHERE idLeilao =
NEW.idLeilao ORDER BY valor LIMIT 1))
        THEN
            INSERT INTO Notificacao (idUtilizador, texto) VALUES
            ((SELECT idCliente FROM Licitacao WHERE idLeilao = NEW.idLeilao
ORDER BY valor LIMIT 1),
                'A sua licitacao no leilao ' || (SELECT nome FROM Leilao
WHERE idLeilao = NEW.idLeilao) || ' foi ultrapassada.');
        END IF;
        UPDATE Leilao SET licitacaoAtual = NEW.valor WHERE idLeilao =
NEW.idLeilao AND
            (NEW.valor > licitacaoAtual OR licitacaoAtual IS NULL);
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
```

atualizarLeilao() → Utilizado pelo *trigger* maiorLicitacao.

```
CREATE OR REPLACE FUNCTION notificaLicitacoesRemovidas ()
    RETURNS TRIGGER AS
```

```
$$
  DECLARE
    _leiloeiro INTEGER;
    _licitacao INTEGER;
    _nomeLeilao VARCHAR(50);
  BEGIN
        FOR _licitacao IN (SELECT idLicitacao FROM Licitacao WHERE idLeilao
= OLD.idLeilao)
        LOOP
            SELECT nome FROM Leilao WHERE idLeilao = OLD.idLeilao INTO
_nomeLeilao;
            SELECT idCliente FROM Licitacao WHERE idLicitacao = _licitacao
INTO _leiloeiro;
            IF _leiloeiro IS NOT NULL THEN
                INSERT INTO Notificacao (idUtilizador, texto) VALUES
(_leiloeiro, 'O leilao ' || _nomeLeilao || ' foi eliminado, como tal as tuas
licitacoes nesse leilao tambem foram eliminadas.');
            END IF;
        END LOOP;
        RETURN OLD;
  END;
$$ LANGUAGE plpgsql;
```

notificaLicitacoesRemovidas() → Utilizado pelo *trigger* leilaoApagado.

```
CREATE OR REPLACE FUNCTION atualizaMaiorLicitacao ()
    RETURNS void AS
$$
    DECLARE
        _leilao INTEGER;
        _cliente INTEGER;
        _nomeLeilao VARCHAR(50);
    BEGIN
        FOR _leilao IN (SELECT idLeilao FROM Leilao)
        LOOP
            UPDATE Leilao SET licitacaoAtual = (SELECT valor FROM Licitacao
WHERE idLeilao = _leilao ORDER BY valor DESC LIMIT 1)
            WHERE idLeilao = _leilao RETURNING nome INTO _nomeLeilao;
            SELECT idCliente INTO _cliente FROM Licitacao WHERE idLeilao =
_leilao ORDER BY valor DESC LIMIT 1;
            IF (_cliente) IS NOT NULL THEN
                INSERT INTO Notificacao (idUtilizador, texto) VALUES (
_cliente, 'A sua licitacao no leilao ' || _nomeLeilao || ' voltou a ser a
mais alta.');
            END IF;
        END LOOP;
    END;
$$ LANGUAGE plpgsql;
```

atualizaMaiorLicitacao() → Utilizado pelo *trigger* membroBanido. Este procedimento não devolve nenhum valor.

```
CREATE OR REPLACE FUNCTION membroBanido ()
    RETURNS TRIGGER AS
$$
    BEGIN
        -- Leiloeiro banido
        DELETE FROM Leilao
            WHERE idLeiloeiro = NEW.idMembroBanido;
        -- Cliente banido
        DELETE FROM Licitacao
            WHERE idCliente = NEW.idMembroBanido;
        EXECUTE atualizaMaiorLicitacao ();
        INSERT INTO Notificacao (idUtilizador, texto) VALUES (
NEW.idMembroBanido, 'Foi banido durante ' ||  NEW.duracao ||
         ' dias, como tal todos os seu leiloes/licitacoes foram apagadas.
Mensagem do administrador: ' || NEW.motivo );
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
```

membroBanido() → Utilizado pelo *trigger* membroBanido.

```
CREATE OR REPLACE FUNCTION notificaMensagem ()
    RETURNS TRIGGER AS
$$
    BEGIN
        INSERT INTO Notificacao (idUtilizador, texto) VALUES (
NEW.idRecetor, 'Recebeu uma nova mensagem de ' || (SELECT nomeUtilizador
FROM Membro WHERE idUtilizador = NEW.idEmissor) || '.' );
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
```

notificaMensagem() → Utilizado pelo *trigger* mensagem.

— *AutoLeilões, Lda.*

[Voltar à página principal]