TÉCNICO LISBOA

Bases de Dados

T16 - Introdução ao SQL/PSM

Prof. Daniel Faria
Prof. Flávio Martins

Sumário

- Restrições de Integridade
- Introdução ao SQL/PSM
  - Agregação
  - Selects Encadeados
  - Combinações de Conjuntos
  - Determinação de Elemento Distintivo

TÉCNICO LISBOA

Restrições de Integridade em SQL

Restrições de Integridade em SQL

- **Restrições de preenchimento e unicidade:**
  - **Chave primária:** PRIMARY KEY
  - **Chave candidata:** UNIQUE NOT NULL
  - **Unicidade sem obrigatoriedade:** UNIQUE
  - **Obrigatoriedade sem unicidade:** NOT NULL
- **Restrições de integridade referencial:**
  - **Chave estrangeira:** FOREIGN KEY / REFERENCES

Funcionalidade de Chaves Estrangeiras

```
CREATE TABLE superclass(
    id    SERIAL PRIMARY KEY,
    att1  VARCHAR UNIQUE NOT NULL
);
CREATE TABLE subclass(
    id    INTEGER PRIMARY KEY REFERENCES superclass,
    att2  VARCHAR NOT NULL
);
INSERT INTO superclass(att1) VALUES('some value');
INSERT INTO subclass(id,att2) VALUES(1,'another value');
DELETE FROM superclass;
update or delete on table "superclass" violates foreign key constraint
"subclass_id_fkey" on table "subclass"
```

Funcionalidade de Chaves Estrangeiras

- Por omissão impedem a remoção ou atualização de valores que estão a ser referenciados
- Como implementar restrições do tipo “quando uma superclasse / entidade forte é removida, todas as subclasses / entidades fracas também devem ser”?
  - Podemos mudar o comportamento por omissão da chave estrangeira:
    - ON DELETE CASCADE
    - ON UPDATE CASCADE

Funcionalidade de Chaves Estrangeiras

```
ALTER TABLE subclass DROP CONSTRAINT subclass_id_fkey;
ALTER TABLE subclass ADD FOREIGN KEY (id) REFERENCES superclass ON DELETE
CASCADE ON UPDATE CASCADE;

DELETE FROM superclass;
(1 rows affected)
```

Restrições de Integridade em SQL

- Restrições ao valor de atributos
  - Restrições de tipo de dados
    - Criar tipo de dados personalizado (e.g. para enumerações)

```
CREATE TYPE species AS ENUM ('chicken','cow','goat','sheep');

CREATE TYPE gps AS (latitude NUMERIC(8,6),longitude NUMERIC(8,6));
```

  - Verificações ao valor usando CHECK

Funcionalidade do CHECK

- CHECK permite:
  - Expressões algébricas e funções sobre o valor de um atributo
  - Expressões algébricas envolvendo vários atributos

```
CHECK (LENGTH(name) > 3)
CHECK (birthdate > '1920-01-01')
CHECK (EXTRACT(YEAR FROM AGE(birthdate)) > 18)
CHECK (start_date < end_date)
```

- CHECK <u>não permite</u>:
  - Olhar para mais do que uma linha a inserir
  - Olhar para outras tabelas

Outras Restrições de Integridade Comuns

1. "RI-1: Todas as chaves de *entidade* têm de existir em *associação*"
2. "RI-2: Todas as chaves de *superclasse* têm de existir em *subclasse1* ou *subclasse2* mas não em ambas"
3. "RI-3: Todos os pares *att1*,*att2* em *rel1* têm de existir em *rel2*"
4. "RI-4: Todos os pares *att3*,*att4* em *rel3* têm de existir na junção de *rel4* e *rel5*"

São todas restrições de integridade referencial mas geralmente não podem/devem ser implementadas como chaves estrangeiras

Outras Restrições de Integridade Comuns

1. "RI-1: Todas as chaves de *entidade* têm de existir em *associação*"

   Levaria a FKs circulares (evitar)

2. "RI-2: Todas as chaves de *superclasse* têm de existir em *subclasse1* ou *subclasse2* mas não em ambas"

   "Anti-FK" (não existe)

3. "RI-3: Todos os pares *att1*,*att2* em *rel1* têm de existir em *rel2*"

   Implementável com FKs (atributos podem ser FK de várias tabelas)

4. "RI-4: Todos os pares *att3*,*att4* em *rel3* têm de existir na junção de *rel4* e *rel5*"

   Não implementável com FKs pois requer JOIN

Solução: Triggers & Stored Procedures

TÉCNICO LISBOA

TÉCNICO LISBOA

Introdução ao SQL/PSM

Introdução ao SQL/PSM

- SQL/PSM (Persistent Stored Modules):
  - Extensão ao SQL com linguagem de programação para funções e procedimentos armazenados no SGBD
  - Incorporada no standard SQL em 1999
- Implementada (de forma não 100% compatível) por vários SGBD SQL, e.g.:
  - MySQL stored procedures
  - Oracle PL/SQL (do qual o PSM derivou diretamente)
  - **PostgreSQL PL/pgSQL** (muito semelhante ao PL/SQL)

Introdução ao SQL/PSM

- Define a sintaxe e semântica de:
  - Fluxo de controlo
  - Lidar com excepções
  - Variáveis locais
  - Atribuição de expressões a variáveis e parâmetros
  - Uso processual de cursores
- Define esquema de informação para procedimentos armazenados
- Permite definir métodos para tipos de dados personalizados

Introdução ao SQL/PSM

- Pode ser utilizado em:
  - Funções (CREATE FUNCTION)
  - Procedimentos (CREATE PROCEDURE)
  - Indiretamente em Triggers (CREATE TRIGGER) que executam funções ou procedimentos

BD Exemplo (Bank)

Account (account-number, branch-name, balance

Loan (loan-number, branch-name, amount)

Borrower (customer-name, loan-number)

loan-number: FK(Loan)

Depositor (customer-name, account-number)

account-number: FK(Account)

Credit-info (customer-name, limit, credit-balance)

Employee (employee-name, street, city)

Works (employee-name, branch-name, salary)

employee-name: FK(Employee)

TÉCNICO LISBOA

Funções & Procedimentos

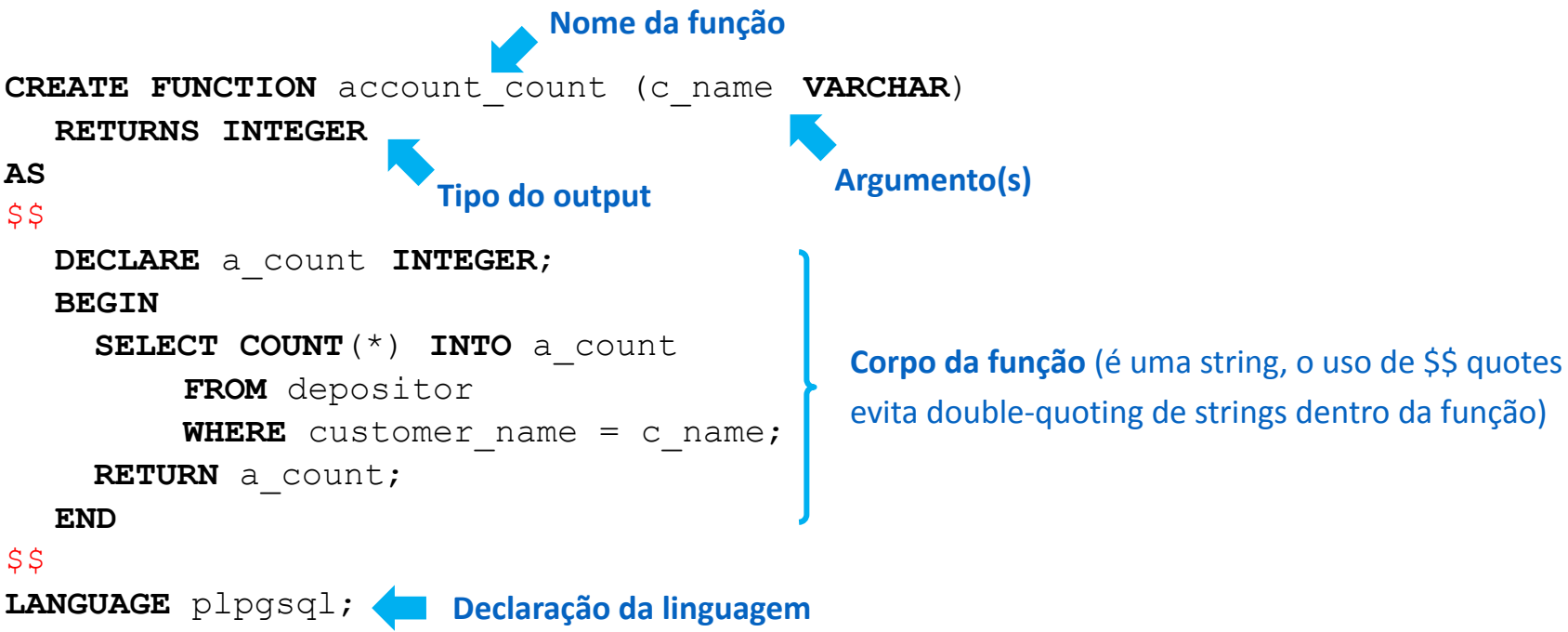
Exemplo de Função

- Função que devolve o número de contas de um cliente dado o seu nome

```
CREATE FUNCTION account_count (c_name VARCHAR)
   RETURNS INTEGER
AS
$$
   DECLARE a_count INTEGER;
   BEGIN
     SELECT COUNT(*) INTO a_count
          FROM depositor
          WHERE customer_name = c_name;
     RETURN a_count;
   END
$$
LANGUAGE plpgsql;
```

TÉCNICO LISBOA

Exemplo de Função



Exemplo de Função

```
CREATE FUNCTION account_count (c_name VARCHAR)
   RETURNS INTEGER
AS
$$
   DECLARE a_count INTEGER;
   BEGIN
     SELECT COUNT(*) INTO a_count
          FROM depositor
          WHERE customer_name = c_name;
     RETURN a_count;
   END
$$
LANGUAGE plpgsql;
```

Declaração de variáveis

Statement de atribuição

Return statement

Função propriamente dita (entre begin e end, composta por statements)

Exemplo de Função

- Alternativa:

```
CREATE FUNCTION account_count (IN c_name VARCHAR, OUT a_count INTEGER)
AS
$$
   BEGIN
     SELECT COUNT(*) INTO a_count
          FROM depositor
          WHERE customer_name = c_name;
   END
$$
LANGUAGE plpgsql;
```

Output(s) definido(s) com OUT

- Permite vários outputs
- RETURNS no cabeçalho é redundante
- Corpo da função não pode declarar as variáveis OUT ou conter return

[Sintaxe frequentemente reservada a PROCEDURES noutras implementações de PSM]

Exemplo de Função

- Utilização:

```
SELECT customer_name,
    customer_street,
    customer_city
FROM customer
WHERE account_count(customer_name) > 1;
```

  - Devolve a informação de todos os clientes que têm pelo menos uma conta (sem precisar de JOIN, ou IN, ou EXISTS para ligar a depositor)
    - Funções raras vezes melhoram o desempenho de queries, mas podem simplificá-las bastante

Funções que Devolvem Tabelas

- Função que devolve todas as contas de um cliente dado o seu nome

```
CREATE FUNCTION accounts_of (VARCHAR)
   RETURNS SETOF account
AS
$$
     SELECT a.account_number, branch_name, balance
     FROM account a, depositor d
     WHERE a.account_number = d.account_number
          AND d.customer_name = $1;
$$
LANGUAGE sql;
```

Tipo de output é um subset da relação

Argumento anónimo

Neste caso a função é apenas uma query SQL

Funções que Devolvem Tabelas

- Função que devolve todas as contas de um cliente dado o seu nome
- Utilização:

```
SELECT * FROM accounts_of('Smith');

 account_number | branch_name | balance
----------------+-------------+---------
 A-215          | Mianus      |  700.00
 A-444          | North Town  |  625.00
(2 rows)
```

- Funções que devolvem tabelas são semelhantes a “vistas com parâmetros”

Funções que Devolvem Tabelas

- Função que devolve todas as contas de um cliente dado o seu nome (mas sem o branch)

```
CREATE FUNCTION accounts_of (VARCHAR)
   RETURNS TABLE(account_number VARCHAR,
         balance NUMERIC(12,2)

AS
$$
     SELECT a.account_number, balance
     FROM account a, depositor d
     WHERE a.account_number = d.account_number
          AND d.customer_name = $1;
$$
LANGUAGE sql;
```

Tipo de output é uma tabela

Query semelhante à anterior, mas não devolve branch (output não pode ser SETOF account)

Funções com Tipos Personalizados

- Função que devolve todas as contas de um cliente dado o seu nome (mas sem o branch)

```
CREATE TYPE account_data AS (
   account_number VARCHAR,
   balance        NUMERIC(12,2));
```

Subset dos atributos de account (sem branch)

```
CREATE FUNCTION accounts_of (VARCHAR)
   RETURNS account_data
AS
$$
     SELECT a.account_number, balance
     FROM account a, depositor d
     WHERE a.account_number = d.account_number
          AND d.customer_name = $1;
$$
LANGUAGE sql;
```

Tipo de output é um tipo personalizado

Diferença: aqui o output só pode ser uma linha da tabela, nos casos anteriores podiam ser várias!

TÉCNICO LISBOA

Exemplo de Procedimento

- Função que apaga todas as contas que têm saldo zero

```
CREATE PROCEDURE clear_accounts
AS
$$
    DELETE FROM account
    WHERE balance = 0;
$$
LANGUAGE sql;
```

Argumentos e outputs são opcionais

Podemos inserir, atualizar ou remover dados em tabelas

- Utilização:

```
CALL clear_accounts;
```

TÉCNICO LISBOA

Função vs. Procedimento

Categoria	Input(s)	Output(s)	Invocação	Manipulação de Dados	Observações
FUNCTION	Opcional	Mandatório	SELECT	✘ *	Não podem chamar PROCEDUREs
PROCEDURE	Opcional	Opcional	CALL	✔	Podem chamar FUNCTIONs

* Excepto TRIGGER FUNCTIONS

TÉCNICO LISBOA

Sintaxe PSM (PL/pgSQL)

Sintaxe PSM (PL/pgSQL)

- **Blocos BEGIN … END**
  - Podem conter múltiplos comandos SQL
  - Podem conter declarações de variáveis locais
- **Declaração de variáveis**
  - **`DECLARE`** `<variable> <type> [`**`DEFAULT`** `<value>]`
  - O âmbito da variável local está restrito ao bloco onde foi declarada

Sintaxe PSM (PL/pgSQL)

- **Atribuição de valores**
  - `<variable> := <value>;`
  - **`SELECT`** `<columns>` **`INTO`** `<variable>`
    `FROM` …
- **Declaração de Variáveis**
  - `<variable_name>` **`INTEGER;`**
  - `<variable_name> tablename%`**`ROWTYPE;`**
  - `<variable_name> tablename.columnname%`**`TYPE;`**
  - `<variable_name>` **`RECORD;`**
    - Semelhante a ROWTYPE mas sem estrutura pré-definida

Sintaxe PSM (PL/pgSQL)

- **Condições**
  - ```
    IF <condition>
        THEN <statements1>
        ELSE <statements2>
    END IF;
    ```
- **Ciclos**
  - ```
    WHILE <condition>
        LOOP <statements>
    END LOOP;
    ```
  - ```
    FOR <variable> IN <select_clause>
        LOOP <statements>
    END LOOP;
    ```

TÉCNICO LISBOA
    ```

Cursores

- Mecanismo para ler uma tabela linha a linha, análogo a iteradores noutra linguagens de programação
- Declaração: **`DECLARE`** `cursor_name` **`CURSOR FOR SELECT`** … **`FROM`** …
- Abertura: **`OPEN`** `cursor_name`
- Iteração: **`FETCH`** `cursor_name` **`INTO`** …
- Fecho: **`CLOSE`** `cursor_name`

Cursores

- Exemplo: uso de cursor para calcular a média

```
CREATE OR REPLACE FUNCTION average_balance(OUT avg_balance REAL) AS
$$
DECLARE
  cursor_account CURSOR FOR SELECT balance FROM account;
  balance REAL; sum_balance REAL := 0.0; count_balance INTEGER := 0;
BEGIN
  OPEN cursor_account;
  LOOP
     FETCH cursor_account INTO balance;
     IF NOT FOUND THEN EXIT;
     END IF;
     sum_balance = sum_balance + balance;
     count_balance = count_balance + 1;
  END LOOP;
  CLOSE cursor_account;
  avg_balance = sum_balance / count_balance;
END
$$ LANGUAGE plpgsql;
```

TÉCNICO LISBOA

Triggers

Triggers

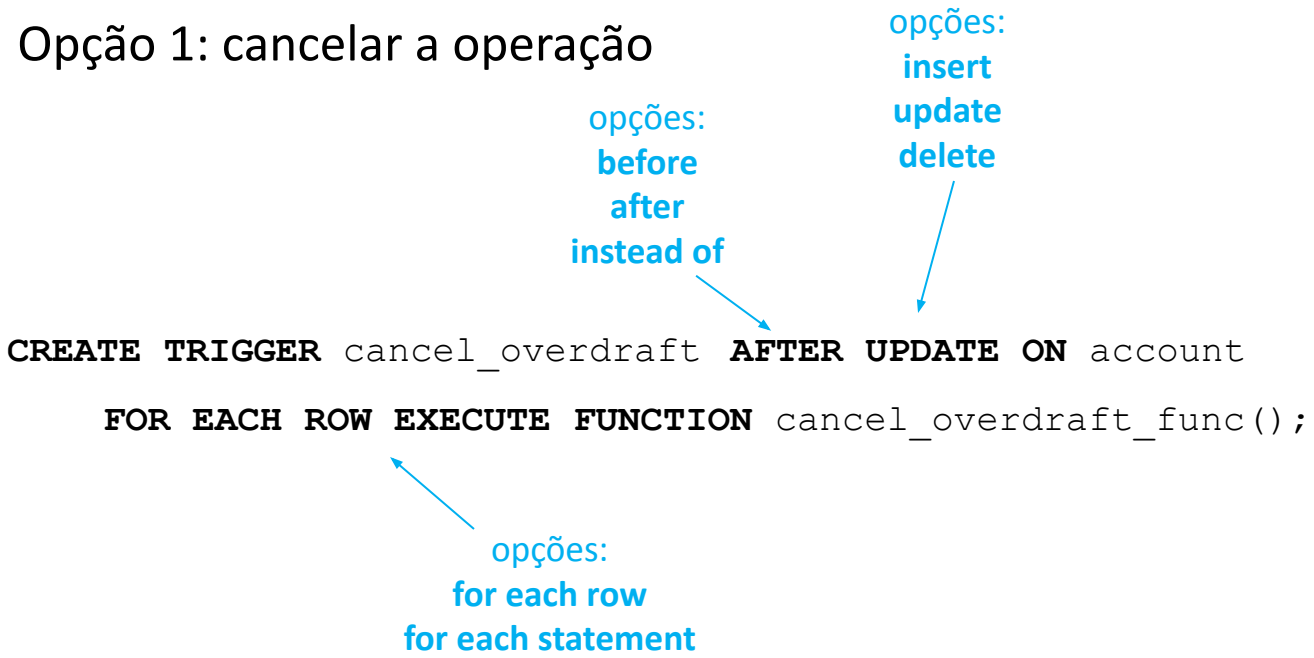
- Um trigger é uma instrução executada em reacção a uma modificação na BD
  - Permite implementar restrições de integridade mais sofisticadas bem como procedimentos que “resolvem” problemas no estado da BD
- Para especificar um trigger é necessário definir:
  - As condições em que o trigger é disparado
  - As acções a fazer quando o trigger é executado
- Além de declarar o trigger em si, normalmente é necessário declarar uma trigger function (i.e. uma função com return type TRIGGER)

Triggers

- Cenário: um cliente tenta levantar uma quantia superior ao seu saldo
  - Opção 1: cancelar a operação
  - Opção 2:
    - Criar um empréstimo igual à quantia em falta com o mesmo número que a conta
    - Colocar o saldo da conta a zero

Triggers

- Cenário: um cliente tenta levantar uma quantia superior ao seu saldo
  - Opção 1: cancelar a operação



Triggers

- Cenário: um cliente tenta levantar uma quantia superior ao seu saldo
  - Opção 1: cancelar a operação

```
CREATE OR REPLACE cancel_overdraft_func() RETURNS TRIGGER AS
$$
BEGIN
     IF NEW.balance < 0 THEN
          RAISE EXCEPTION 'A conta % tem saldo insuficiente.',
               NEW.account_number;
  END IF;
  RETURN NEW;
END
$$ LANGUAGE plpgsql;
```

Retorna o tuplo que vai ser inserido (num trigger AFTER não faz diferença)

Triggers

- Cenário: um cliente tenta levantar uma quantia superior ao seu saldo
  - Opção 1: cancelar a operação

```
UPDATE account SET balance = balance-500 WHERE account_number = 'A-102';

ERROR:  A conta A-102 tem saldo insuficiente.
CONTEXT:  PL/pgSQL function cancel_overdraft_func() line 4 at RAISE
```

Triggers

- Cenário: um cliente tenta levantar uma quantia superior ao seu saldo
  - Opção 2: Criar um empréstimo igual à quantia em falta com o mesmo número que a conta; colocar o saldo da conta a zero

```
CREATE TRIGGER overdraft_loan AFTER UPDATE ON account
    FOR EACH ROW EXECUTE FUNCTION overdraft_loan_func();
```

Triggers

- Opção 2

```
CREATE OR REPLACE overdraft_loan_func() RETURNS TRIGGER AS
$$
BEGIN
     IF NEW.balance < 0 THEN
          INSERT INTO loan VALUES (NEW.account_number, NEW.branch_name, (-1)*NEW.balance);
          INSERT INTO borrower
               (SELECT customer_name, account_number FROM depositor
                    WHERE depositor.account_number = new.account_number);
          UPDATE account SET balance = 0
               WHERE account.account_number = NEW.account_number;
     END IF;
     RETURN NEW;
END
$$ LANGUAGE plpgsql;
```

Opções de Triggers

- **Timing:** BEFORE / AFTER / INSTEAD OF
  - AFTER: só é afectada a inserção/atualização se produzir uma excepção
  - BEFORE & INSTEAD OF: permitem alterar a inserção/atualização
- **Execução:** FOR EACH ROW / FOR EACH STATEMENT
  - FOR EACH ROW: o trigger despoleta uma vez por cada linha alterada
  - FOR EACH STATEMENT: o trigger despoleta uma vez por cada statement SQL (opção defeito por omissão)

TÉCNICO LISBOA