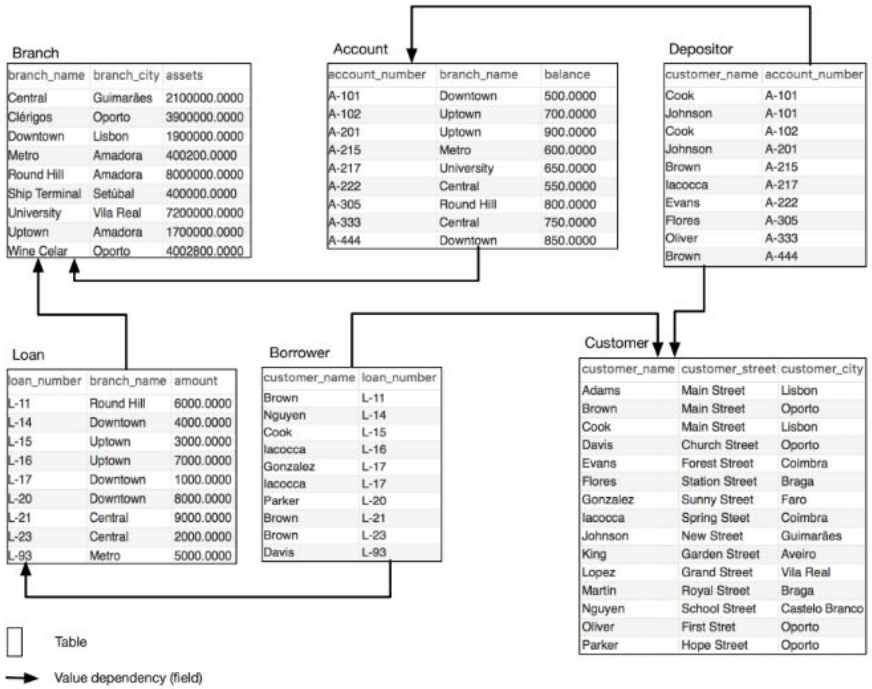
TÉCNICO LISBOA

Funções e Triggers

Slides e Soluções do Laboratório 09

Base de Dados Bank



TÉCNICO LISBOA

1) Funções

a. Escreva uma função em SQL que devolve o saldo absoluto de um cliente, isto é, a diferença entre (1) todo o dinheiro que esse cliente tem em contas e (2) todas as quantias que deve em empréstimos ao banco.

Solução 1

```
CREATE OR REPLACE FUNCTION total_balance (c VARCHAR(80)) RETURNS DECIMAL(16,4) AS $$
  DECLARE credit DECIMAL(16,4);
  DECLARE debt DECIMAL(16,4);
  BEGIN
    SELECT SUM(balance) INTO credit
      FROM account JOIN depositor USING (account_number) WHERE customer_name = c;
    IF credit IS NULL THEN credit:=0; END IF;

    SELECT SUM(amount) INTO debt
      FROM loan JOIN borrower USING (loan_number) WHERE customer_name = c;
    IF debt IS NULL THEN debt:=0; END IF;

    RETURN credit - debt;
  END;
$$ LANGUAGE plpgsql;
```

1) Funções

a. Escreva uma função em SQL que devolve o saldo absoluto de um cliente, isto é, a diferença entre (1) todo o dinheiro que esse cliente tem em contas e (2) todas as quantias que deve em empréstimos ao banco.

Solução 2

```
CREATE OR REPLACE FUNCTION total_balance (IN c VARCHAR(80), OUT diff DECIMAL(16,4)) AS $$
  DECLARE credit DECIMAL(16,4);
  DECLARE debt DECIMAL(16,4);
  BEGIN
    SELECT SUM(balance) INTO credit
      FROM account JOIN depositor USING (account_number) WHERE customer_name = c;
    IF credit IS NULL THEN credit:=0; END IF;

    SELECT SUM(amount) INTO debt
      FROM loan JOIN borrower USING (loan_number) WHERE customer_name = c;
    IF debt IS NULL THEN debt:=0; END IF;

    diff := credit - debt;
  END;
$$ LANGUAGE plpgsql;
```

Idêntica à anterior, mas usando IN e OUT (deixamos de ter return statement)

TÉCNICO LISBOA

1) Funções

a. Escreva uma função em SQL que devolve o saldo absoluto de um cliente, isto é, a diferença entre (1) todo o dinheiro que esse cliente tem em contas e (2) todas as quantias que deve em empréstimos ao banco.

Solução 3

```
CREATE OR REPLACE FUNCTION total_balance (c VARCHAR(80)) RETURNS DECIMAL(16,4) AS
$$
    SELECT COALESCE(SUM(balance),0)-COALESCE(SUM(amount),0)
        FROM account JOIN depositor USING (account_number)
        FULL JOIN borrower USING (customer_name)
        FULL JOIN loan USING (loan_number)
        WHERE customer_name = c;
$$
LANGUAGE sql;
```

Usando o COALESCE para lidar com os casos NULL, podemos implementar a função apenas com SQL

1) Funções

b. Escreva uma função que devolve a diferença entre o saldo médio das contas em duas agências dadas. A função tem como parâmetros as agências a comparar.

Solução 1

```
CREATE OR REPLACE FUNCTION diff (b1 VARCHAR(80),b2 VARCHAR(80)) RETURNS DECIMAL(16,4) AS $$
  DECLARE bal_b1 DECIMAL(16,4);
  DECLARE bal_b2 DECIMAL(16,4);
  BEGIN
    SELECT AVG(balance) INTO bal_b1
      FROM account WHERE branch_name = b1;
    IF bal_b1 IS NULL THEN bal_b1:=0; END IF;

    SELECT AVG(balance) INTO bal_b2
      FROM account WHERE branch_name = b2;
    IF bal_b2 IS NULL THEN bal_b2:=0; END IF;

    RETURN bal_b1 - bal_b2;
  END;
$$ LANGUAGE plpgsql;
```

Análoga à Solução 1 da alínea (a)

TÉCNICO LISBOA

1) Funções

b. Escreva uma função que devolve a diferença entre o saldo médio das contas em duas agências dadas. A função tem como parâmetros as agências a comparar.

Solução 2

```
CREATE OR REPLACE FUNCTION diff (IN b1 VARCHAR(80), IN b2 VARCHAR(80), OUT d DECIMAL(16,4) AS $$
  DECLARE bal_b1 DECIMAL(16,4);
  DECLARE bal_b2 DECIMAL(16,4);
  BEGIN
    SELECT AVG(balance) INTO bal_b1
      FROM account WHERE branch_name = b1;
    IF bal_b1 IS NULL THEN bal_b1:=0; END IF;

    SELECT AVG(balance) INTO bal_b2
      FROM account WHERE branch_name = b2;
    IF bal_b2 IS NULL THEN bal_b2:=0; END IF;

    d := bal_b1 - bal_b2;
  END;
$$ LANGUAGE plpgsql;
```

Análoga à Solução 2 da alínea (a)

TÉCNICO LISBOA

1) Funções

b. Escreva uma função que devolve a diferença entre o saldo médio das contas em duas agências dadas. A função tem como parâmetros as agências a comparar.

Solução 3

```
CREATE OR REPLACE FUNCTION diff (b1 VARCHAR(80),b2 VARCHAR(80)) RETURNS DECIMAL(16,4) AS $$
  SELECT bal_b1-bal_b2 FROM
    (SELECT COALESCE(AVG(balance),0) AS bal_b1 FROM account WHERE branch_name = b1) AS r1,
    (SELECT COALESCE(AVG(balance),0) AS bal_b2 FROM account WHERE branch_name = b2) AS r2;
$$ LANGUAGE sql;
```

Análoga à Solução 3 da alínea (a)

1) Funções

c. Usando a função desenvolvida na alínea anterior, escreva uma consulta que permita determinar qual é a agência cujo saldo médio é o maior.

```
SELECT b1.branch_name
FROM branch b1, branch b2
GROUP BY b1.branch_name
HAVING MIN(diff(b1.branch_name, b2.branch_name)) = 0;
```

A(s) agência(s) cujo saldo médio é maior vai/vão ter sempre resultados >= 0 para qualquer caso possível da função diff. No pior caso tem 0 quando comparada com ela mesma. Portanto, é aquela cujo mínimo de todos os casos possíveis de diff (i.e., todos os pares de branches) é 0. Todas as outras agências terão pelo menos 1 valor negativo para diff (quando comparadas com a maior).
Qualquer outra implementação exigiria selects encadeados, e seria desnecessariamente complexa.

1) Funções

d. A tabela depositor associa clientes a contas. Escreva em SQL os comandos necessários de modo a que se uma conta for apagada da tabela account, sejam também apagados os registos na tabela depositor correspondentes.

Não é preciso função ou trigger. Basta atualizar a FOREIGN KEY para ON DELETE CASCADE.

```
-- verificar nome da FK
\d depositor
-- remover FK
ALTER TABLE depositor DROP CONSTRAINT fk_depositor_account;
-- adicionar FK com funcionalidade desejada
ALTER TABLE depositor ADD CONSTRAINT fk_depositor_account FOREIGN KEY(account_number)
  REFERENCES account(account_number) ON DELETE CASCADE;
```

1) Funções

e. Repita a alínea anterior para os empréstimos (i.e., com as tabelas borrower e loan).

A mesma coisa que na anterior:

```
-- verificar nome da FK
\d borrower
-- remover FK
ALTER TABLE borrower DROP CONSTRAINT fk_borrower_loan;
-- adicionar FK com funcionalidade desejada
ALTER TABLE borrower ADD CONSTRAINT fk_borrower_loan FOREIGN KEY(loan_number)
  REFERENCES loan(loan_number) ON DELETE CASCADE;
```

2) Triggers

a. Escreva um trigger que elimina um empréstimo de um cliente quando a respetiva quantia (amount) em dívida for inferior ou igual a zero. As amortizações de um empréstimo são feitas através de um update do campo amount na tabela loan. O trigger deverá assegurar que quando o amount seja negativo, o valor (negativo) seja adicionado como asset do branch respectivo, e o respectivo empréstimo seja eliminado das tabelas borrower e loan.

```
CREATE OR REPLACE FUNCTION update_loan() RETURNS TRIGGER AS $$
  BEGIN
    IF NEW.amount <= 0 THEN
      UPDATE branch b SET assets = assets - NEW.amount
        WHERE b.branch_name = NEW.branch_name;
      DELETE FROM borrower b WHERE b.loan_number = NEW.loan_number;
      DELETE FROM loan l WHERE l.loan_number = NEW.loan_number;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER update_loan_trigger AFTER UPDATE ON loan
  FOR EACH ROW EXECUTE FUNCTION update_loan();
```

2) Triggers

b. Uma nova política do banco obriga a não conceder a partir de agora novos empréstimos a clientes que não estejam associados a contas no banco. Escreva um trigger para impedir a associação desses clientes a um empréstimo.

```
CREATE OR REPLACE FUNCTION check_depositor_loan() RETURNS TRIGGER AS $$
  BEGIN
    IF NEW.customer_name NOT IN (SELECT customer_name FROM depositor) THEN
      RAISE EXCEPTION 'Customer % has no account in this bank.',NEW.customer_name;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER check_depositor_loan_trigger BEFORE INSERT ON borrower
  FOR EACH ROW EXECUTE FUNCTION check_depositor_loan();
```