Laboratório 2.1: Expressões, restrições e transações SQL com SQLite e MySQL

04/10/2023

1 Uso de expressões `CASE`

A expressão `CASE` pode ser usada para testar um conjunto de condições e retorna o resultado da avaliação sobre tais condições. É possível usar a expressão `CASE` em instruções DDL (e.g `CREATE TABLE`) ou DML (`SELECT, UPDATE ou DELETE` nas cláusulas `WHERE, ORDER BY ou HAVING`).

```
-- Sintaxe
CASE expressão
    WHEN expressao_1 THEN resultado_1
    WHEN expressao_2 THEN resultado_2
    ELSE resultado_else -- o ELSE é opcional e se omitido então o CASE retorna NULL
END
```

1.1 Exemplos

```
CREATE TABLE Entregas (
    idEntrega INTEGER PRIMARY KEY AUTOINCREMENT,
    endereco TEXT NOT NULL,
    cidade TEXT NOT NULL,
    UF TEXT NOT NULL,
    CEP TEXT NOT NULL,
    peso DOUBLE NOT NULL
);

INSERT INTO Entregas (endereco, cidade, UF, CEP, peso)
    VALUES ('Rua ABC, 123', 'Sao Jose', 'SC', '88000-000',10.1),
           ('Rua HIJ, 789', 'Sao Paulo', 'SP', '19000-000',30)
           ('Rua DEF, 456', 'Belo Horizonte', 'MG', '30672-772',200.0);

SELECT idEntrega,
    endereco,
    cidade,
    uf,
    CEP,
    peso,
    CASE UF
        WHEN 'SC'
        THEN 'Estadual'
        ELSE 'Interestadual'
    END Tipo,

    CASE
        WHEN peso >= 200 THEN
        'transporte terrestre'
        WHEN peso < 200 AND UF = 'SC' THEN
        'transporte terrestre'
        ELSE 'transporte aéreo'
    END Transporte
FROM Entregas;
```

2 Restrição `CHECK` ao criar ou alterar tabela

Ao criar uma tabela é possível colocar uma restrição que será avaliada quando expressões `INSERT` ou `UPDATE` forem executadas. Essa abordagem tira a responsabilidade de verificação da aplicação e passa para o Sistema Gerenciador de Banco de Dados (SGBD).

```
-- Restrição no nível de coluna
CREATE TABLE tabela(
    nome TEXT,
    valor INTEGER CHECK(expressão),
    cidade TEXT,

);
```

```
-- Restrição no nível da tabela
CREATE TABLE tabela(
    nome TEXT,
    valor INTEGER,
    cidade TEXT,
    CHECK(expressão)
);
```

2.1 Exemplos

```
CREATE TABLE Pessoa (
    idPessoa INTEGER PRIMARY KEY AUTOINCREMENT,
    nome       TEXT    NOT NULL,
    sobrenome  TEXT    NOT NULL,
    email      TEXT    NOT NULL,
    telefone   TEXT    NOT NULL CHECK (length(telefone) >= 14)
);

-- vai resultar em erro
INSERT INTO Pessoa (nome, sobrenome, email, telefone) VALUES ('Juca', 'Pereira', 'juca@email.com',
     '3381-2100');

-- vai inserir com sucesso
INSERT INTO Pessoa (nome, sobrenome, email, telefone) VALUES ('Juca', 'Pereira', 'juca@email.com',
     '(48) 3381-2100');
```

```
CREATE TABLE Produtos(
    idProduto  INTEGER PRIMARY KEY AUTOINCREMENT,
    nome       TEXT            NOT NULL,
    valor      DECIMAL (10, 2) NOT NULL,
    desconto   DECIMAL (10, 2) NOT NULL DEFAULT 0,
    CHECK (valor >= desconto AND desconto >= 0 AND valor >= 0)
);

-- vai resultar em erro
INSERT INTO Produtos (nome, valor, desconto) VALUES ('Caneta', 5.00, 6.00);
```

```
-- Criando uma tabela com restrições usando a expressão CASE
CREATE TABLE Aluno (idAluno INTEGER PRIMARY KEY AUTOINCREMENT,
    nome TEXT NOT NULL, email TEXT NOT NULL, sexo TEXT NOT NULL, reservista TEXT,
    CONSTRAINT ck_sexo CHECK (sexo in ('M', 'F')),
    CONSTRAINT ck_reservista CHECK (
        CASE
            WHEN sexo = 'M' THEN
                CASE
                    WHEN reservista is not null THEN 1
                    ELSE 0
                END
            ELSE 1
        END
    )
);
```

```sql
-- vai resultar em erro
INSERT INTO Aluno (nome, email, sexo) VALUES ('Juca', 'juca@email.com', 'X');
INSERT INTO Aluno (nome, email, sexo) VALUES ('Juca', 'juca@email.com', 'M');

-- vai inserir com sucesso
INSERT INTO Aluno (nome, email, sexo, reservista) VALUES ('Juca', 'juca@email.com', 'M', '1234');
INSERT INTO Aluno (nome, email, sexo) VALUES ('Ana', 'ana@email.com', 'F');
```

3 Transações

Em um banco de dados relacional transacional, a execução de qualquer instrução SQL irá automaticamente acontecer dentro de uma transação e toda transação é persistida assim que a instrução SQL é finalizada.

As transações podem ser iniciadas manualmente por meio do comando `BEGIN` e neste caso só serão persistidas por meio do comando `COMMIT` ou desfeitas por meio do comando `ROLLBACK`. O comando `END TRANSACTION` é um apelido para o comando `COMMIT`. Por fim, as transações podem ser aninhadas (uma dentro de outra) por meio dos comandos `SAVEPOINT` e `RELEASE`.

O SQLite provê suporte a múltiplas transações de leitura (e.g. `SELECT`), porém somente uma única transação de escrita (e.g. `CREATE, DELETE, DROP, INSERT ou UPDATE`).

3.1 Exemplo

```sql
CREATE TABLE Contas (
    numeroConta INTEGER NOT NULL PRIMARY KEY,
    saldo DECIMAL NOT NULL DEFAULT 0,
    CHECK(saldo >= 0)
);

CREATE TABLE historico_operacoes (
    numeroOperacao INTEGER PRIMARY KEY AUTOINCREMENT,
    numeroConta INTEGER NOT NULL,
    operacao TEXT NOT NULL,
    total DECIMAL NOT NULL,
    data TEXT NOT NULL,
    CHECK(operacao in ('+', '-')),
    FOREIGN KEY (numeroConta) REFERENCES Contas(numeroConta)
);

INSERT INTO Contas VALUES (123,1000),(456,50);
```

```sql
-- Transferindo valores entre contas
BEGIN TRANSACTION;
-- Debitando R$ 200,00 da conta 123
UPDATE Contas SET saldo = saldo - 200 WHERE numeroConta = 123;
-- Creditando R$ 200,00 na conta 456
UPDATE Contas SET saldo = saldo + 200 WHERE numeroConta = 456;

-- Registrando a transferência no histórico
INSERT INTO historico_operacoes (numeroConta, operacao, total, data)
            VALUES (123,'-',200,datetime('now'))
INSERT INTO historico_operacoes (numeroConta, operacao, total, data)
            VALUES (456,'+',200,datetime('now'))

COMMIT;
```

Agora, tente executar individualmente cada instrução da listagem a seguir.

```
SELECT * FROM historico_operacoes;

BEGIN TRANSACTION;

-- Resultará em erro, pois não tem saldo suficiente
UPDATE Contas SET saldo = saldo - 4000 WHERE numeroConta = 123;

-- Será registrado normalmente
INSERT INTO historico_operacoes (numeroConta, operacao, total, data)
            VALUES (123,'-',4000,datetime('now'));

SELECT * FROM historico_operacoes;

-- Irá desfazer tudo que foi alterado nesta transação
ROLLBACK;

SELECT * FROM historico_operacoes;
```

O SQLite implementa apenas um subconjunto limitado da instrução `ALTER TABLE`, pois só permite renomear uma tabela, renomear uma coluna, adicionar ou remover uma coluna. Caso necessite fazer outro tipo de alteração, como adicionar uma restrição, então terá que recorrer as transações.

Assumindo que tenha a tabela `Pessoa`, bem como os registros adicionados a ela, da Subseção 2.1, iremos adicionar mais uma coluna `sexo` e adicionaremos uma restrição (`CHECK`) para indicar os valores permitidos para essa nova coluna (`M ou F`).

```
-- desabilitando a restrição de chave estrangeira. Para o exemplo em questão isso não é relevante
PRAGMA foreign_keys = OFF;
BEGIN TRANSACTION;

ALTER TABLE Pessoa RENAME TO _pessoa_antiga;

CREATE TABLE Pessoa (
    idPessoa INTEGER PRIMARY KEY AUTOINCREMENT,
    nome       TEXT    NOT NULL,
    sobrenome  TEXT    NOT NULL,
    email      TEXT    NOT NULL,
    sexo       TEXT    DEFAULT ' ' CHECK (sexo in ('M','F', ' ')),
    telefone   TEXT    NOT NULL CHECK (length(telefone) >= 14)
);

INSERT INTO Pessoa (nome, sobrenome, email,telefone)
        SELECT nome, sobrenome, email, telefone FROM _pessoa_antiga;

DROP TABLE  _pessoa_antiga;

END TRANSACTION;
-- habilitando a restrição de chave estrangeira. Para o exemplo em questão isso não é relevante
PRAGMA foreign_keys = ON;
```

Referências

- https://www.sqlite.org/lang_transaction.html
- https://www.sqlitetutorial.net/sqlite-transaction/
- https://dev.mysql.com/doc/refman/8.0/en/sql-transactional-statements.html
- https://www.sqlite.org/lang_altertable.html