Introdução às Bases de Dados

Restrições de Integridade Complexas

FCUL, Departamento de Informática

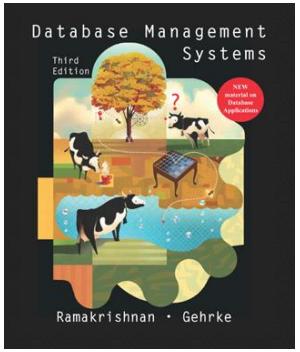
Ano Letivo 2021/2022

Ana Paula Afonso

Sumário e Referências

- Sumário
  - Restrições complexas em SQL (CHECK…)
    - Criação de domínios
    - Criação de tipos
    - Restrições numa tabela
    - Asserções - restrições sobre várias tabelas
  - *Triggers* e bases de dados ativas
    - Definição, exemplos e problemas com *triggers*
  - CHECK vs *triggers*

- Referências
  - R. Ramakrishnan (**capítulo 5, secção 5.7, 5.8 e 5.9**)



Regras de Integridade Complexas

- Modelo relacional fornece restrições de integridade
  - Domínio, Coluna, Entidade (ou chave), Referencial, *User-Defined*

- **Suporte para restrições complexas**
  - Domínio: definição de domínios para além dos fornecidos pelo SGBD
  - Coluna: uso de SELECTs em CHECKs de tabelas
  - *User-defined*: definição de asserções e *triggers*
- Asserções
  - CHECKs independentes de tabelas
- *Triggers*
  - Procedimentos invocados aquando de escritas em tabelas

Definição de Novos Domínios

- Definição de novos domínios
  - Para manter coerência nas colunas
  - Ex. novo domínio com nºs inteiros entre 1 e 10

```
CREATE DOMAIN ratingval INTEGER
       DEFAULT 1
       CHECK ( VALUE >= 1 AND VALUE <= 10 )
```

- Aplicação do novo domínio em coluna de tabela
  - Ex. `CREATE TABLE Sailors (….,rating` **`ratingval,`** `…)`
  - Se INSERT em Sailors omitir o valor de rating, este é preenchido com 1

- Valores de `rating` podem ser comparados com os de colunas do tipo INTEGER
  - Limitação conceptual, pois os domínios são diferentes

Definição de Tipos

- Comando CREATE TYPE define um novo tipo de dados abstrato
  - Necessita de métodos próprios para suportar comparações, adições, … mesmo que baseado em domínios simples, como INTEGER
  - Para evitar comparações entre tipos diferentes
  - Exemplo:

```
CREATE TYPE ratingtype AS INTEGER
```

- Domain vs Type
  - Colunas `ratingtype` **não** podem ser comparadas (ou operadas) com colunas do tipo `INTEGER`
  - Enquanto, colunas `ratingval` podem ser comparadas (ou operadas) com colunas do tipo `INTEGER`

Restrições Complexas numa Tabela

- Definição de tabelas pode incluir cláusulas CHECK

```
CREATE TABLE Sailors ( sid INTEGER,…,rating INTEGER,
        CHECK (rating >= 1 AND rating <= 10 ))
```

- Definição de tabelas pode incluir restrições mais complexas
  - Com a consulta a outras tabelas
  - Exemplo: reservas de barcos**, exceto para barcos com nome Interlake**

```
CREATE TABLE Reserves (sid INTEGER, bid INTEGER, day DATE,
 PRIMARY KEY (sid, bid),
 FOREIGN KEY (sid) REFERENCES Sailors (sid),
 FOREIGN KEY (bid) REFERENCES Boats (bid),
 CONSTRAINT noInterlakeRes
        CHECK ('Interlake' <> ( SELECT B.bname
                                FROM Boats B
                                WHERE B.bid = Reserves.bid )))
```

  - Condição verificada para cada INSERT ou UPDATE na tabela `Reserves`

Restrições Complexas numa Tabela

- Outro exemplo com restrição complexa
  - O número de barcos e de marinheiros não pode ultrapassar os 100

```
CREATE TABLE Sailors (
        …
CONSTRAINT smallClub
CHECK ((SELECT COUNT(S.sid) FROM Sailors S)
        +
        (SELECT COUNT(B.bid) FROM Boats B) < 100))
```

- Problemas?

Restrições Complexas numa Tabela

- Outro exemplo com restrição complexa
  - O número de barcos e de marinheiros não pode ultrapassar os 100

```
CREATE TABLE Sailors (
        …
CONSTRAINT smallClub
CHECK ((SELECT COUNT(S.sid) FROM Sailors S)
        + (SELECT COUNT(B.bid) FROM Boats B) < 100))
```

- Problemas
  - Restrição associada a **Sailors**, apesar de envolver também **Boats**

    Decisão de colocar restrição em Sailors é arbitrária
  - Enquanto Sailors estiver vazia, condição não precisa de ser verificada

    Número de barcos pode crescer indefinidamente (e ficar superior a 100)
  - Solução: criação de condições não associadas a uma qualquer tabela - asserções

Asserções

- Restrições que não estão associadas a uma qualquer tabela
  - Definidas ao mesmo nível das tabelas no esquema de dados
  - Apropriadas para restrições que abrangem múltiplas tabelas

- Exemplo para a restrição complexa do slide anterior

```
CREATE ASSERTION smallClub
        CHECK ((SELECT COUNT (S.sid) FROM Sailors S)
                +(SELECT COUNT (B. bid) FROM Boats B)<100)
```

  - Não está associada a nenhuma tabela
  - Condição verificada para cada INSERT ou UPDATE em Sailors ou Boats

Triggers

- Procedimento que é automaticamente despoletado quando se realizam escritas específicas
  - **Evento** de escrita ativa **condição** que permite, ou não, execução de **ação**

- **Evento**
  - Tipo de escrita na base de dados que faz ativar o *trigger*
  - Tipos de escrita: qualquer combinação de INSERT, UPDATE, e DELETE
  - Escritas podem ser numa tabela inteira ou em colunas específicas
  - Opções de ativação do *trigger*
    - Antes ou depois da escrita se concretizar
    - Uma só vez para um bloco inteiro de escritas ou para cada linha escrita
- **Condição** (opcional)
  - Uma interrogação ou um teste verificado aquando da ativação do *trigger*
- **Ação**
  - Código do procedimento executado quando o *trigger* é ativado e a condição anterior satisfeita

Exemplo de *Trigger*

```
CREATE TRIGGER init_count
   BEFORE INSERT ON Students
```
← **Evento**

```
DECLARE
   count INTEGER
BEGIN
   count := 0;
END
```
← **Ação**

Nota: trigger sem condição

Exemplo de *Trigger*

```
CREATE TRIGGER incr_count
   AFTER INSERT ON Students                    ← Evento
    WHEN (new.age < 18)                        ← Condição
         -- 'new' is just-inserted tuple
    FOR EACH ROW                               ← Ação (Oracle PL/SQL Syntax)
    BEGIN                                          -- Row-level trigger
         count := count + 1;
    END
```

Observações

1. Também existe old e new para referir o valor antes e depois de um UPDATE

Triggers: MySQL syntax

```
CREATE
    [DEFINER = user]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

MySQL: exemplo de Trigger

```
CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));

CREATE TRIGGER ins_sum
      BEFORE INSERT ON account
      FOR EACH ROW SET @sum = @sum + NEW.amount;

SET @sum = 0;
INSERT INTO account VALUES
      (137,14.98),
      (141,1937.50),
      (97,-100.00);

SELECT @sum AS 'Total amount inserted';
Total amount inserted
1852.48
```

MySQL: exemplo de trigger

```
CREATE TRIGGER upd_check
      BEFORE UPDATE ON account
       FOR EACH ROW
       BEGIN
           IF NEW.amount < 0 THEN
               SET NEW.amount = 0;
           ELSEIF NEW.amount > 100 THEN
               SET NEW.amount = 100;
           END IF;
       END;
```

Oracle: Exemplo de Trigger

```
CREATE TRIGGER verifica_aumento_vencimento
  BEFORE UPDATE OF vencimento ON empregado
  FOR EACH ROW
  WHEN (new.vencimento > old.vencimento)
DECLARE
  vencimento_do_chefe NUMBER := NULL;
BEGIN     -- Supõe-se que só existe um chefe.
  SELECT vencimento INTO vencimento_do_chefe
    FROM empregado WHERE (categoria = 'chefe');
  IF (:new.vencimento > vencimento_do_chefe) THEN
    RAISE_APPLICATION_ERROR(-20001, 'Não pode ganhar mais que o chefe!')
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'A empresa não tem chefe!');
  WHEN OTHERS THEN RAISE;
END;
```

Evento que ativa o *trigger*

Condição após ativação (opcional)

Ação do *trigger* (procedimento)

Variável new guarda a linha que resultaria do UPDATE, e old refere a linha antes do UPDATE

Exceções, se lançadas, cancelam o UPDATE

(fonte: António Ferreira, SIBD 2016)

Bases de Dados Ativas e *Triggers*

- Base de dados ativa
  - Base de dados com *triggers* associados
- Usos típicos de *triggers*
  - Restrições de integridade complexas
  - Autorizações de acesso e auditoria de escritas em tabelas
    Ex. que utilizadores escreveram em certa tabela e a que horas
  - Réplicas síncronas de tabelas
- Definição de *triggers* – responsável e problemas
  - Tipicamente definidos (ou autorizados) pelo DBA = *DataBase Administrator*
  - Razão: consequências do uso de *triggers* podem ser difíceis de entender
    Vários *triggers* podem ser ativados em simultâneo, por ordem arbitrária
    Ação de um *trigger* pode ativar outros *triggers* (*triggers* recursivos)
  - Um uso criterioso de **restrições de integridade pode** frequentemente **substituir/evitar uso de *triggers***

CHECKS vs. Triggers

- **CHECKs**
  - Declarativos
  - Mais fáceis de entender
  - Mais eficientes pois podem ser otimizados pelos SGBDs
  - Restrições de integridade verificadas em permanência
    Para quaisquer operações de escrita na base de dados
- **Triggers**
  - Procedimentais (requer saber programar)
  - Desempenho depende da qualidade do programador
    Sempre que possível, devem ser usadas restrições de integridade declarativas
  - Podem ser usados para outros fins (além de manter integridade)
    Autorizações de acesso, auditoria, estatísticas, replicação de dados…
  - Restrições de integridade verificadas para escritas específicas em tabelas
    Necessário cuidado para cobrir todos os cenários possíveis de escrita de dados