ACH2025

Laboratório de Bases de Dados

Aula 10

SQL Oracle

Asserções e gatilhos

Professora:

- **Fátima L. S. Nunes**

✓REGRAS DE INTEGRIDADE (RI)

- garantem que mudanças feitas no BD não resultem em inconsistência de dados;
- protegem o BD de dados acidentais;
- uma RI constitui um predicado arbitrário pertencente ao BD ➔ pode representar altos custos para ser testada;
- normalmente RIs são limitadas às que podem ser verificadas com o mínimo tempo de processamento.

✓RI – Restrições *not null*

- especificadas em SQL DDL;
- aplicadas principalmente em atributos que são chaves primárias, mas podem ser especificadas em qualquer atributo de uma relação;

```
CREATE TABLE EMP (
        empno                   NUMBER(4) NOT NULL,
        job                     VARCHAR2(9) NOT NULL,
        ename                   VARCHAR2(10) NOT NULL,
        hiredate                DATE,
        sal                     NUMBER(7,2) NOT NULL,
        comm                    NUMBER(7,2),
        deptno                  NUMBER(2),
        mgr                     NUMBER(4)
);
```

✓RI – Restrições de domínio

- são as RIs mais elementares;
- especificadas em SQL DDL;
- facilmente verificadas pelo SGBD quando um dado é incorporado ao BD;
- diversos atributos podem ter o mesmo domínio:
  - exemplo: nome e endereço podem ambos ser definidos como string com no máximo 50 posições;
- domínios podem ser definidos com a cláusula ***check*** da DDL.

✓RI – Restrições de domínio

```
CREATE TABLE EMP (
        empno                   NUMBER(4) NOT NULL,
        job                     VARCHAR2(9) NOT NULL
                                CHECK (job IN ('MANAGER', 'CLERK',
'SALESMAN', 'ANALYST')),
        ename                   VARCHAR2(20) NOT NULL,
        hiredate                DATE NULL,
        sal                     NUMBER(7,2) NOT NULL CHECK (sal >= 10),
        comm                    NUMBER(7,2) NULL CHECK (comm BETWEEN 10 AND
400),
        deptno                  NUMBER(2) NULL,
        mgr                     NUMBER(4) NULL
);
```

✓RI – Restrições de domínio

```
CREATE TABLE EMP (
       empno                NUMBER(4) NOT NULL,
       job                  VARCHAR2(9) NOT NULL
                            CHECK (job IN ('MANAGER', 'CLERK', 'SALESMAN',
'ANALYST')),
       ename                VARCHAR2(20) NOT NULL,
       hiredate             DATE NULL,
       sal                  NUMBER(7,2) NOT NULL CHECK (sal >= 10),
       comm                 NUMBER(7,2) NULL CHECK (comm BETWEEN 10 AND 400),
       deptno               NUMBER(2) NULL,
       mgr                  NUMBER(4) NULL
);

SQL> INSERT INTO EMP VALUES (25, 'PROFESSOR','MANOEL', NULL, 500, 5, 10,
1)
*
ERRO na linha 1:
ORA-02290: restrição de checagem (SCOTT.SYS_C00603) violada

SQL> INSERT INTO EMP VALUES (25, 'CLERK','MANOEL', NULL, 500, 20, 10, 1)

1 linha criada.
```

✓RI – Restrições de domínio

- para consultar as restrições definidas em Oracle:
- Tabela user_constraints:

```
SQL> select * from user_constraints;

OWNER                          CONSTRAINT_NAME                C TABLE_NAME
------------------------------ ------------------------------ - ------------------------------
SEARCH_CONDITION
--------------------------------------------------------------------------------
R_OWNER                        R_CONSTRAINT_NAME              DELETE_RU STATUS   DEFERRABLE
------------------------------ ------------------------------ --------- -------- -------------
DEFERRED  VALIDATED     GENERATED      BAD LAST_CHA
--------- ------------- -------------- --- --------
SCOTT                          SYS_C00601                     C EMP
job IN ('MANAGER', 'CLERK', 'SALESMAN', 'ANALYST')
                                                              ENABLED  NOT DEFERRABLE
IMMEDIATE VALIDATED     GENERATED NAME     02/04/11

SCOTT                          SYS_C00602                     C EMP
sal >= 10
                                                              ENABLED  NOT DEFERRABLE
IMMEDIATE VALIDATED     GENERATED NAME     02/04/11

SCOTT                          SYS_C00603                     C EMP
```

✓ Afirmações - Assertions

- predicados que expressam uma condição que o BD deve satisfazer;
- restrições de domínio e restrições de integridade referencial são formas especiais de afirmações:
  - facilmente testadas;
  - se aplicam a uma ampla faixa de situações;
- no entanto, várias restrições não podem ser expressas usando apenas essas formas mais simples.

✓ Afirmações - Assertions

- no entanto, várias restrições não podem ser expressas usando apenas essas formas mais simples:
- Exemplos:
  - soma dos salários dos empregados não pode ultrapassar um determinado valor;
  - ninguém pode ter salário maior que o dono da empresa;
  - um departamento tem pelo menos um empregado.

✓ Afirmações - Assertions

- SQL não fornece construções no formato

  "para todo X, P(X)" onde P é um predicado.

- Solução:

  - implementar restrição por meio de uma construção equivalente:

  "não existe X tal que não P(X)".

- Quando uma afirmação é criada, o SGBD testa sua validade.

  - se afirmação é válida, qualquer modificação futura no BD só será permitida se não violar a afirmação.

✓ Afirmações – Assertions

- **Solução:**
  - **implementar restrição por meio de uma construção equivalente:**

  "não existe X tal que não P(X)"

- **Exemplo:**
  - **soma de todas as quantias de empréstimo para cada agência precisa ser menor que a soma de todos os saldos de conta na agência.**

```
create assertion restricao_soma check
 (not exists (select * from agencia
              where (select sum(quantia) from emprestimo
                     where emprestimo.nome_agencia =
                           agencia.nome_agencia) >=
                    (select sum(saldo) from conta
                     where conta.nome_agencia =
                      agencia.nome_agencia)))
```

✓ Afirmações – Assertions

– Exemplo:

- **cada empréstimo tem pelo menos um cliente que mantém uma conta com um saldo mínimo de $1.000,00.**

```
create assertion restricao_saldo check
 (not exists (select * from emprestimo
              where not exists
                    (select * from tomador, depositante, conta
                    where emprestimo.numemprestimo =
                          tomador.numemprestimo and
                          tomador.nome_cliente =
                          depositante.nome_cliente and
                          depositante.numero_conta =
                          conta.numero_conta and
                          conta.saldo >= 1000)))
```

✓ Afirmações - Assertions

- Os testes podem introduzir quantidade significativa de *overhead* se afirmações complexas forem definidas:

  Por isso, devem ser usadas com critério.

- Muitos SGBDs (Oracle inclusive) não implementam *assertions* de acordo com o padrão SQL-92 exemplificado:

  ➔implementam outros mecanismos para preencher as necessidades de afirmações;

  ➔Oracle: *triggers*, restrições de colunas (*checks*), funções e procedimentos.

✓ Gatilhos - *Triggers*

- Instruções que o sistema executa automaticamente como um efeito colateral de uma modificação no BD.
- Requisitos de um gatilho:
  1. especificar quando um *trigger* deve ser executado, composto de duas partes:
     - *evento* que define quando o gatilho será verificado;
     - *condição* que precisa ser satisfeita para que a execução prossiga;
  2. especificar as ações a serem tomadas quando o gatilho for executado.
- Esse modelo de *trigger* é conhecido como modelo evento-condição-ação para *triggers*.

✓ Gatilhos - *Triggers*

- **Mecanismos úteis para alertar usuários ou iniciar tarefas automaticamente quando certas condições são atendidas.**
- **Exemplo:**
  - **Em vez de permitir saldos negativos em contas, o banco trata saldos devedores como empréstimos, da seguinte forma:**
    - **saldo da conta passa para zero;**
    - **é criada uma conta de empréstimo com o valor do saldo devedor – o número do empréstimo é idêntico ao número da conta.**
  - **evento: atualização na relação *conta*;**
  - **condição para executar: saldo negativo;**
  - **ação: passar saldo da tupla em *conta* para zero e criar tupla na relação *empréstimo*.**

✓ *Triggers* em SQL

- disponíveis a partir do SQL-1999.
- cada SGBD implementa de uma forma ➔ problemas de compatibilidade.

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
      insert into tomador
            (select nome_cliente, numero_conta
                  from depositante
                  where nrow.numero_conta =
                  depositante.numero_conta);
      insert into emprestimo values
                  (nrow.numero_conta, nrow.nome_agencia,
                   -nrow.saldo);
      update conta set saldo = 0
```

✓ *Triggers* em SQL

- disponíveis a partir do SQL-1999.
- cada SGBD implementa de uma forma ➔ problemas de compatibilidade.

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nome_cliente, numero_conta
                        from depositante
                        where nrow.numro_conta=depositante.numero_conta);
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Trigger é iniciado após qualquer atualização da relação *conta*

✓ *Triggers* em SQL

- disponíveis a partir do SQL-1999.
- cada SGBD implementa de uma forma ➔ problemas de compatibilidade.

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nome_cliente, numero_conta
                        from depositante
                        where nrow.numro_conta=depositante.numero_conta);
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Percorre explicitamente cada linha atualizada

✓ *Triggers* em SQL

- disponíveis a partir do SQL-1999.
- cada SGBD implementa de uma forma ➔ problemas de compatibilidade.

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nome_cl                    _
                        from depositante
                        where nrow.numro_conta=depositante.numero_conta);
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Cria uma variável *nrow* que armazena o valor de uma linha atualizada após a atualização

(chamada *variável de transição)*

✓ *Triggers* em SQL

- disponíveis a partir do SQL-1999.
- cada SGBD implementa de uma forma ➔ problemas de compatibilidade.

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nome_cliente, numero_conta
                        from depositante
                        where nrow.numro_conta=depositante.numero_conta);
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Especifica a condição para que o *trigger* (restante do código) seja executado

✓ *Triggers* em SQL

- disponíveis a partir do SQL-1999.
- cada SGBD implementa de uma forma ➔ problemas de compatibilidade.

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into
                (sele
                                nrow.numro_conta=depositante.numero_conta);
        insert into      stimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        upd e conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Define que as instruções seguintes devem ser executadas como uma única instrução composta

✓ *Triggers* em SQL

- disponíveis a partir do SQL-1999.
- cada SGBD implementa de uma forma ➔ problemas de compatibilidade.

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into toma
                (sele  nome_cliente, numero_conta
                        from depositante
                        where nrow.numro_conta=depositante.numero_conta);
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Criam novas tuplas nas respectivas relações para representar novo empréstimo

✓ *Triggers* em SQL

- disponíveis a partir do SQL-1999.
- cada SGBD implementa de uma forma ➔ problemas de compatibilidade.

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nome_cliente, numero_conta
                        from depositante
                        where nrow.numro_conta=depositante.numero_conta);
        insert in
                                                    e_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Atualiza o saldo negativo para zero

✓ *Triggers* em SQL

– **opções disponíveis:**

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nome_cliente, numero_conta
                        from depositante
                        where nrow.numro_conta=depositante.numero_conta);
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Pode ser *insert* ou *delete*

✓ *Triggers* em SQL

– **opções disponíveis:**

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nome_cliente, num
                        from depositante
                        where nrow.numro_c                    );
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Pode especificar colunas.

Exemplo:

```
create trigger
trigger_saldo_devedor
after update of saldo
on conta
```

✓ *Triggers* em SQL

– **opções disponíveis:**

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nom
                        from
                        wher                                    nta);
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Pode especificar uma linha antiga.

Exemplo:

referencing old row as oldrow

(armazena o valor antigo de uma linha atualizada ou excluída)

✓ *Triggers* em SQL

– **opções disponíveis:**

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo < 0
begin atomic
        insert into tomador
                (select nome_client
                        from deposi
                        where nrow.
        insert into emprestimo value
                        (nrow.numero_conta, nrow.nome_agencia, -
nrow.saldo);
        update conta set saldo = 0
                where conta.numero_conta= nrow.numero_conta
end
```

Pode ser ativado antes (*before*) do evento

Exemplo:

```
create trigger
trigger_saldo_devedor before
update on conta
```

✓ *Triggers* em SQL

– **opções disponíveis:**

```
create trigger trigger_saldo_devedor after update on conta
referencing new row as nrow
for each row
when nrow.saldo
begin atomic
        insert into
                (se
        insert into
nrow.saldo);
        update cont
                whe
end
```

Em vez de executar uma ação para cada linha afetada, é possível executar uma única ação para a instrução SQL inteira que causou o insert/delete/update.

Para isso, usa-se a cláusula **`for each statement`**

Neste caso, a cláusula **`referencing`** pode ser usada como **`referencing old table as`** ou **`referencing new table as`** para referenciar tabelas temporárias (chamadas *tabelas de transição*). Essas tabelas só podem ser usadas com triggers **after.**

✓ *Triggers* em SQL

- **Para desativar trigger:**

```
alter trigger nome_trigger disable
```

- **Alguns SGBDs usam:**

```
disable trigger nome_trigger
```

- **Para remover trigger:**

```
drop trigger nome_trigger
```

✓ *Triggers* em Oracle

– **Sintaxe geral:**

CREATE OR REPLACE TRIGGER *trigger_name*
triggering_event [*trigger_restriction*]
BEGIN
triggered_action;
END;

Exemplo:

```
CREATE OR REPLACE TRIGGER log_salary_increase
AFTER UPDATE OF salary ON EMP
FOR EACH ROW
BEGIN
        INSERT INTO Emp_log (Emp_id, Log_date, New_salary, Action)
        VALUES (:NEW.employee_id, SYSDATE, :NEW.salary, 'New
Salary');
END;
```

Exercícios

Considerando o modelo de dados definido anteriormente (BD EMPRESA):

1. Revise as tabelas e acrescente afirmações (restrições do tipo CHECK) adequadas para os atributos.
2. Crie um atributo na tabela DEPT que armazene a quantidade total de funcionários do departamento.
3. Construa um gatilho (*trigger*) que some 1 no atributo criado toda vez que for inserido um funcionário na tabela EMP (observe que deverá ser somado 1 no respectivo departamento do funcionário inserido).
4. Construa um gatilho (*trigger*) que subtraia 1 no atributo criado toda vez que for removido um funcionário na tabela EMP (observe que deverá ser subtraído1 no respectivo departamento do funcionário inserido).
5. Construa um gatilho (*trigger*) que acrescente um bônus para o funcionário (tabela BONUS) toda vez que o cargo do funcionário for alterado para MANAGER.
6. Acrescente um atributo na tabela de funcionário que armazene a sua grade salarial.
7. Construa um gatilho (*trigger*) que atualize o atributo criado (grade salarial), considerando a tabela SALGRADE toda vez que o salário do funcionário for alterado.
8. Construa um gatilho (*trigger*) que exclua o registro do departamento toda vez que seu número de funcionários for atualizado para zero.

ACH2025

Laboratório de Bases de Dados

Aula 10

SQL Oracle

Asserções e gatilhos

Professora:

- **Fátima L. S. Nunes**