Universidade Federal
de Campina Grande

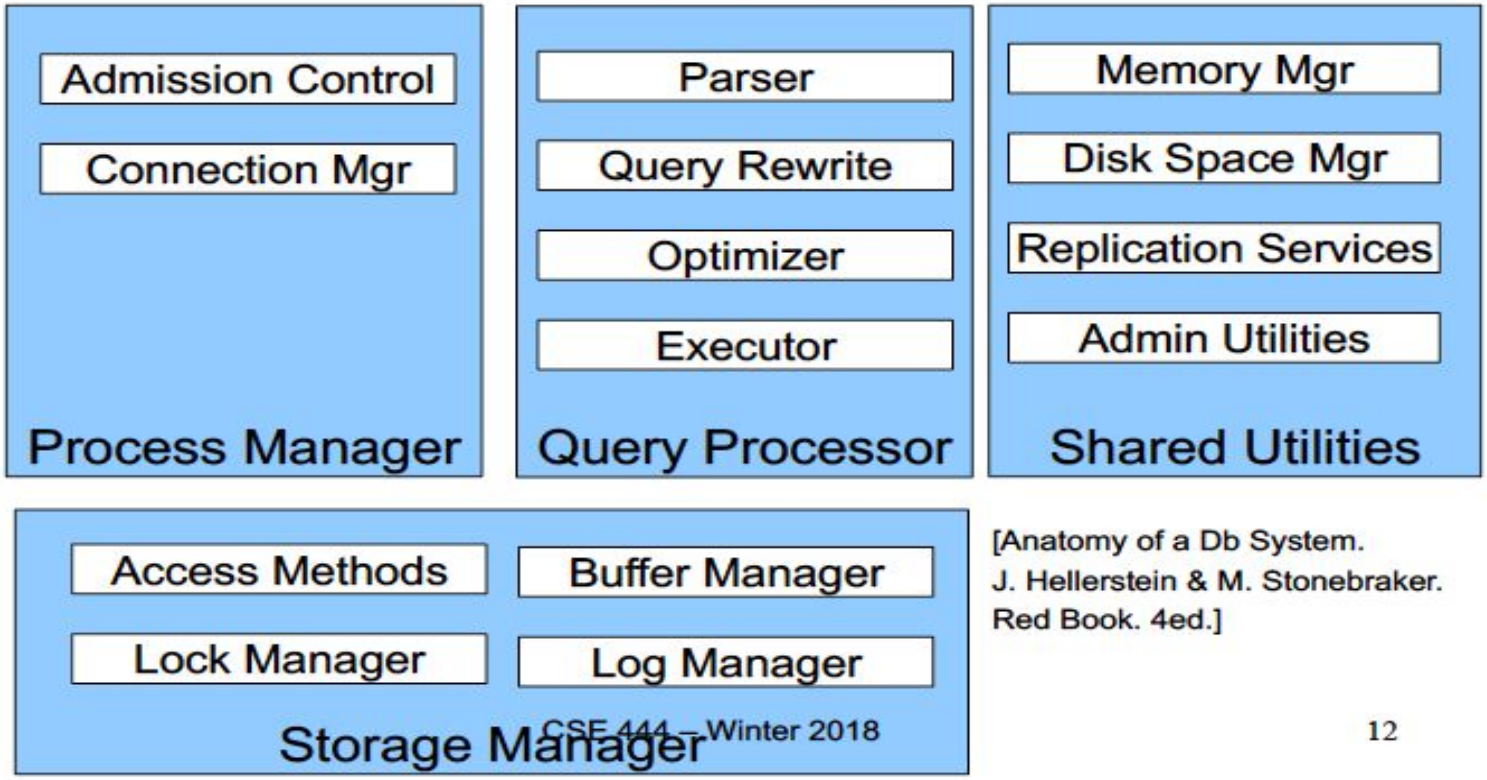
Banco de Dados I
Unidade 7: Arquitetura de SGBD

Prof. Cláudio de Souza Baptista, Ph.D.
Laboratório de Sistemas de Informação – LSI
UFCG

Introdução: Arquitetura de SGBD

- Subsistema de Controle de Concorrência
- Subsistema de Recuperação à Falhas
- Subsistema de Integridade
- Subsistema de Segurança
- Subsistema de Otimização de Consultas

Arquitetura SDBD



[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]

Integridade

Introdução

- Integridade → Precisão, correção, validade
- Refere-se a evitar a perda acidental (não intencional) da consistência do BD
- A perda de integridade pode surgir de:
  - Quedas durante o processamento de uma Ti;
  - Anomalias motivadas pelo acesso concorrente ao BD;
  - Anomalias motivadas pela distribuição dos dados em vários computadores;
  - Erro lógico que viola a suposição de que as Tis devam preservar as restrições de integridade de BD

Introdução

- Integridade → Precisão, correção, validade
- Refere-se a evitar a perda acidental (não intencional) da consistência do BD
- A perda de integridade pode surgir de:
  - Quedas durante o processamento de uma Ti;
  - Anomalias motivadas pelo acesso concorrente ao BD;
  - Anomalias motivadas pela distribuição dos dados em vários computadores;
  - Erro lógico que viola a suposição de que as Tis devam preservar as restrições de integridade de BD

Introdução

- Integridade → Precisão, correção, validade
- Refere-se a evitar a perda acidental (não intencional) da consistência do BD
- A perda de integridade pode surgir de:
  - Quedas durante o processamento de uma Ti;
  - Anomalias motivadas pelo acesso concorrente ao BD;
  - Anomalias motivadas pela distribuição dos dados em vários computadores;
  - Erro lógico que viola a suposição de que as Tis devam preservar as restrições de integridade de BD

Introdução

- Integridade → Precisão, correção, validade
- Refere-se a evitar a perda acidental (não intencional) da consistência do BD
- A perda de integridade pode surgir de:
  - Quedas durante o processamento de uma Ti;
  - Anomalias motivadas pelo acesso concorrente ao BD;
  - Anomalias motivadas pela distribuição dos dados em vários computadores;
  - Erro lógico que viola a suposição de que as Tis devam preservar as restrições de integridade de BD

Introdução

- Integridade → Precisão, correção, validade
- Refere-se a evitar a perda acidental (não intencional) da consistência do BD
- A perda de integridade pode surgir de:
  - Quedas durante o processamento de uma Ti;
  - Anomalias motivadas pelo acesso concorrente ao BD;
  - Anomalias motivadas pela distribuição dos dados em vários computadores;
  - Erro lógico que viola a suposição de que as Tis devam preservar as restrições de integridade de BD

Introdução

Exemplos:

- Informação Incorreta;
- Uma transação de venda ocorre mas o operador insere a data da transação de forma incorreta;
- Um zero é esquecido de ser digitado ao se entrar um salário de um empregado;
- Dados duplicados;
- Um novo departamento é criado, com codDepto = 200, e é inserido na tabela duas vezes;
- Chaves estrangeiras inválidas;
- Departamento codDepto = 300 é fechado, os empregados deste departamento recebem um novo codDepto e um empregado é esquecido de ser atualizado ficando com o extinto código 300.

Introdução

- Para assegurar que os dados do BD estão válidos precisa-se formalizar verificações (**check**) e regras de negócio (**business rules**) que os dados devem aderir.
- Esta forma de assegurar consistência é conhecida como restrições de integridade (**integrity constraints**).
  - **Restrição de Integridade** → Finalidade é fornecer um meio de assegurar que as mudanças feitas no BD, por usuários autorizados, não levam à perda de consistência dos dados.
  - **Questão:** Como garantir Integridade do BD?

Introdução

Solução 1: Inserir as restrições de integridade no código da aplicação

- Problemas:
  - **Sobrecarrega** o programador, pois ele precisa codificar todas as validações;
  - **Susceptível a erros** → pode-se esquecer de fazer uma validação => BD inconsistente
  - **Difícil manutenção** → Se RI mudar, então precisa mudar todas as aplicações que validam àquela RI

Solução 2: Inserir as RI no próprio BD, através do subsistema do SGBD.

Introdução

Solução 1: Inserir as restrições de integridade no código da aplicação

- Problemas:
  - **Sobrecarrega** o programador, pois ele precisa codificar todas as validações;
  - **Susceptível a erros** → pode-se esquecer de fazer uma validação => BD inconsistente
  - **Difícil manutenção** → Se RI mudar, então precisa mudar todas as aplicações que validam àquela RI

Solução 2: Inserir as RI no próprio BD, através do subsistema do SGBD.

Introdução

Solução 1: Inserir as restrições de integridade no código da aplicação

- Problemas:
  - **Sobrecarrega** o programador, pois ele precisa codificar todas as validações;
  - **Susceptível a erros** pode-se esquecer de fazer uma validação => BD inconsistente
  - **Difícil manutenção** Se RI mudar, então precisa mudar todas as aplicações que validam àquela RI

Solução 2: Inserir as RI no próprio BD, através do subsistema do SGBD.

Introdução

Solução 1: Inserir as restrições de integridade no código da aplicação

- Problemas:
  - **Sobrecarrega** o programador, pois ele precisa codificar todas as validações;
  - **Susceptível a erros** □ pode-se esquecer de fazer uma validação => BD inconsistente
  - **Difícil manutenção** □ Se RI mudar, então precisa mudar todas as aplicações que validam àquela RI

Solução 2: Inserir as RI no próprio BD, através do subsistema do SGBD.

Introdução

Solução 1: Inserir as restrições de integridade no código da aplicação

- Problemas:
  - **Sobrecarrega** o programador, pois ele precisa codificar todas as validações;
  - **Susceptível a erros** □ pode-se esquecer de fazer uma validação => BD inconsistente
  - **Difícil manutenção** □ Se RI mudar, então precisa mudar todas as aplicações que validam àquela RI

Solução 2: Inserir as RI no próprio BD, através do subsistema do SGBD.

Introdução

Solução 1: Inserir as restrições de integridade no código da aplicação

- Problemas:
  - **Sobrecarrega** o programador, pois ele precisa codificar todas as validações;
  - **Susceptível a erros** □ pode-se esquecer de fazer uma validação => BD inconsistente
  - **Difícil manutenção** □ Se RI mudar, então precisa mudar todas as aplicações que validam àquela RI

Solução 2: Inserir as RI no próprio BD, através do subsistema do SGBD.

Regras de Integridade

- Suponhamos uma implementação de um subsistema de integridade com as seguintes funções:
  - Monitorar transações;
  - Se violação → tomar ação apropriada
- Para prover tais funções, o sistema contém **Regras de Integridade**, expressas numa linguagem de alto nível (Ex.: SQL DDL), que detecte uma tentativa de violação e tome a ação apropriada.

Regras de Integridade

Exemplo:

```
R1: After Updating Conta.Saldo:
      Conta.Saldo > 0
      Else
            Do;
                  Set return code to “regra R1 violada”
                  Reject
      End;
```

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD → definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD ⟶ definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD → definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD → definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD → definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD → definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD → definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD → definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's

Regras de Integridade

- **Importante:**
  - As regras são compiladas e armazenadas no dicionário de dados;
  - Uma vez lançadas no sistema, as regras são utilizadas daquele estado do BD → definição deve ser rejeitada se for de encontro ao estado atual do BD
- **Vantagens:**
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo;
  - Deve existir um meio de atualizar regras com o sistema funcionando;
  - Pode-se utilizar linguagem de consulta para as RI's
- **Exemplo:** 'Quais as regras de integridade que se aplicam ao salário dos empregados?'

Regras de Integridade

- No padrão SQL-2 foi introduzido o comando **check()** para validação de integridade.

Create table r ($a_1 d_1$, $a_2 d_2$, … $a_n d_n$,
<regras de integridade 1>
…
<regras de integridade k>)

Regras de Integridade

Ex. 1:

```
create table Conta
(numero char(15),
  nome varchar(30),
  nome_agencia varchar(30),
  saldo float check (saldo >= 0),
  primary key(numero)
  )

create table estudante
       ( nome char(20),
         matricula int,
        nivel char(15) CHECK( nivel IN (‘Bacharelado’, ‘Mestrado’,
‘Doutorado’)),
        primary key(matricula));
```

Regras de Integridade

Ex. 2:

```
check (nome_agencia in (select nome_agencia from agencia))
```

- **Obs.: CUIDADO!** Podemos aqui remover ou alterar um *nome_agencia* em Agencia sem que o **check** seja validado. Ou seja, o **check** é feito na tabela origem (Conta), mas no sentido contrário (Agencia) nenhuma verificação é feita.
- Estes **checks** acima servem só para os atributos nos quais eles estão definidos. Entretanto, podemos ter checks ao nível de tupla (permitem a validação de tupla).

Regras de Integridade

Ex. 2:

```
check (nome_agencia in (select nome_agencia from agencia))
```

- **Obs.: CUIDADO!** Podemos aqui remover ou alterar um *nome_agencia* em Agencia sem que o **check** seja validado. Ou seja, o **check** é feito na tabela origem (Conta), mas no sentido contrário (Agencia) nenhuma verificação é feita.
- Estes **checks** acima servem só para os atributos nos quais eles estão definidos. Entretanto, podemos ter checks ao nível de tupla (permitem a validação de tupla).

Regras de Integridade

Ex. 2:

```
check (nome_agencia in (select nome_agencia from agencia))
```

- **Obs.: CUIDADO!** Podemos aqui remover ou alterar um *nome_agencia* em Agencia sem que o **check** seja validado. Ou seja, o **check** é feito na tabela origem (Conta), mas no sentido contrário (Agencia) nenhuma verificação é feita.
- Estes **checks** acima servem só para os atributos nos quais eles estão definidos. Entretanto, podemos ter checks ao nível de tupla (permitem a validação de tupla).

Regras de Integridade

Ex. 3:

```
CREATE TABLE Conta (
        numero char(15),
        nome varchar(30),
        nome_agencia varchar(30),
        saldo float,
        primary key(numero),
        check (saldo  >= 0)
  )
```

Regras de Integridade

Faixa de valores:

```
Create table teste (
        I int,
        Valor1 float,
        Valor2 float,
        Valor3 int,
        CHECK (
                (valor1 BETWEEN .99 AND 90.0)
        AND
                (valor2 BETWEEN .99 AND 95.2)
        AND (valor3 BETWEEN 0 AND 1000)
);
```

Nomeando Constraints

- É **fortemente recomendado** que os **constraints** sejam nomeados:
  - Facilita a alteração ou remoção;
  - Facilita a vida do programador no entendimento das exceções lançadas pelo SGBD (violação de integridade);

Nomeando Constraints

- É fortemente recomendado que os **constraints** sejam nomeados:
  - Facilita a alteração ou remoção;
  - Facilita a vida do programador no entendimento das exceções lançadas pelo SGBD (violação de integridade);

Ex. 4:

```
CREATE TABLE Conta (
    numero char(15),
    nome varchar(30),
    nome_agencia varchar(30),
    saldo float,
    PRIMARY KEY(numero),
    CONSTRAINT check_saldo_positivo CHECK  (saldo  >= 0)
)
```

Alterando Constraints

```
ALTER TABLE Conta DROP CONSTRAINT check_saldo_positivo;
```

```
ALTER TABLE Conta ADD CONSTRAINT
check_saldo CHECK (saldo > 0);
```

Triggers (Gatilhos)

- Um Gatilho é uma regra do tipo **E_C_A**:
  - **E**: Evento
  - **C**: Condição a ser satisfeita na presença do evento **E**
  - **A**: Ação a ser tomada caso a condição **C** seja satisfeita
- **Exemplo:** Ao invés de restringir contas com saldos negativos, podemos querer ativar uma ação que automaticamente inicia um empréstimo para aquela conta.
  - **Evento**: operação de modificação do saldo
  - **Condição**: se saldo < 0
  - **Ação**: Criar um empréstimo para conta

Triggers (Gatilhos)

- Principais características:
  1. A ação pode ser executada antes ou depois do evento;
  2. A ação pode referenciar ambos antigo (old) e novo (new) valores das tuplas que foram inseridas, deletadas ou atualizadas no evento que causou o disparo da ação;
  3. Eventos de update podem ser limitados a um atributo particular ou a um conjunto de atributos;
  4. Uma condição pode ser especificada por uma cláusula WHEN, a ação é executada somente se a regra for disparada e a condição seja verdadeira;
  5. Triggers são definidos como nível de linha/tupla (row level) ou nível de comando (statement level)

Triggers (Gatilhos)

- Principais características:
  1. A ação pode ser executada antes ou depois do evento;
  2. A ação pode referenciar ambos antigo (old) e novo (new) valores das tuplas que foram inseridas, deletadas ou atualizadas no evento que causou o disparo da ação;
  3. Eventos de update podem ser limitados a um atributo particular ou a um conjunto de atributos;
  4. Uma condição pode ser especificada por uma cláusula WHEN, a ação é executada somente se a regra for disparada e a condição seja verdadeira;
  5. Triggers são definidos como nível de linha/tupla (row level) ou nível de comando (statement level)

Triggers (Gatilhos)

- Principais características:
  1. A ação pode ser executada antes ou depois do evento;
  2. A ação pode referenciar ambos antigo (old) e novo (new) valores das tuplas que foram inseridas, deletadas ou atualizadas no evento que causou o disparo da ação;
  3. Eventos de update podem ser limitados a um atributo particular ou a um conjunto de atributos;
  4. Uma condição pode ser especificada por uma cláusula WHEN, a ação é executada somente se a regra for disparada e a condição seja verdadeira;
  5. Triggers são definidos como nível de linha/tupla (row level) ou nível de comando (statement level)

Triggers (Gatilhos)

- Principais características:
  1. A ação pode ser executada antes ou depois do evento;
  2. A ação pode referenciar ambos antigo (old) e novo (new) valores das tuplas que foram inseridas, deletadas ou atualizadas no evento que causou o disparo da ação;
  3. Eventos de update podem ser limitados a um atributo particular ou a um conjunto de atributos;
  4. Uma condição pode ser especificada por uma cláusula WHEN, a ação é executada somente se a regra for disparada e a condição seja verdadeira;
  5. Triggers são definidos como nível de linha/tupla (row level) ou nível de comando (statement level)

Triggers (Gatilhos)

- Principais características:
  1. A ação pode ser executada antes ou depois do evento;
  2. A ação pode referenciar ambos antigo (old) e novo (new) valores das tuplas que foram inseridas, deletadas ou atualizadas no evento que causou o disparo da ação;
  3. Eventos de update podem ser limitados a um atributo particular ou a um conjunto de atributos;
  4. Uma condição pode ser especificada por uma cláusula WHEN, a ação é executada somente se a regra for disparada e a condição seja verdadeira;
  5. Triggers são definidos como nível de linha/tupla (row level) ou nível de comando (statement level)

Triggers (Gatilhos)

- Principais características:
  1. A ação pode ser executada antes ou depois do evento;
  2. A ação pode referenciar ambos antigo (old) e novo (new) valores das tuplas que foram inseridas, deletadas ou atualizadas no evento que causou o disparo da ação;
  3. Eventos de update podem ser limitados a um atributo particular ou a um conjunto de atributos;
  4. Uma condição pode ser especificada por uma cláusula WHEN, a ação é executada somente se a regra for disparada e a condição seja verdadeira;
  5. Triggers são definidos como nível de linha/tupla (row level) ou nível de comando (statement level)

Triggers (Gatilhos)

- Pode-se criar Triggers que são ativados por comandos (eventos):
  - **INSERT:** o trigger é invocado quando uma nova linha é inserida na tabela associada ao trigger;
  - **DELETE**: o trigger é invocado quando uma linha da tabela associada ao trigger é removida;
  - **UPDATE:** o trigger é invocado quando uma linha da tabela associada é atualizada;
  - **UPDATE OF column-list:** o triger é invocado quando uma linha da tabela associada é atualizada de tal forma que a column-list tem sido modificada.

Triggers (Gatilhos)

- Pode-se criar Triggers que são ativados por comandos (eventos):
  - **INSERT:** o trigger é invocado quando uma nova linha é inserida na tabela associada ao trigger;
  - **DELETE**: o trigger é invocado quando uma linha da tabela associada ao trigger é removida;
  - **UPDATE:** o trigger é invocado quando uma linha da tabela associada é atualizada;
  - **UPDATE OF column-list:** o triger é invocado quando uma linha da tabela associada é atualizada de tal forma que a column-list tem sido modificada.

Triggers (Gatilhos)

- Pode-se criar Triggers que são ativados por comandos (eventos):
  - **INSERT:** o trigger é invocado quando uma nova linha é inserida na tabela associada ao trigger;
  - **DELETE**: o trigger é invocado quando uma linha da tabela associada ao trigger é removida;
  - **UPDATE:** o trigger é invocado quando uma linha da tabela associada é atualizada;
  - **UPDATE OF column-list:** o triger é invocado quando uma linha da tabela associada é atualizada de tal forma que a column-list tem sido modificada.

Triggers (Gatilhos)

- Pode-se criar Triggers que são ativados por comandos (eventos):
  - **INSERT:** o trigger é invocado quando uma nova linha é inserida na tabela associada ao trigger;
  - **DELETE**: o trigger é invocado quando uma linha da tabela associada ao trigger é removida;
  - **UPDATE:** o trigger é invocado quando uma linha da tabela associada é atualizada;
  - **UPDATE OF column-list:** o triger é invocado quando uma linha da tabela associada é atualizada de tal forma que a column-list tem sido modificada.

Triggers (Gatilhos)

- Pode-se criar Triggers que são ativados por comandos (eventos):
  - **INSERT:** o trigger é invocado quando uma nova linha é inserida na tabela associada ao trigger;
  - **DELETE**: o trigger é invocado quando uma linha da tabela associada ao trigger é removida;
  - **UPDATE:** o trigger é invocado quando uma linha da tabela associada é atualizada;
  - **UPDATE OF column-list:** o triger é invocado quando uma linha da tabela associada é atualizada de tal forma que a column-list tem sido modificada.

Triggers (Gatilhos)

- Pode-se criar Triggers que são ativados por comandos (eventos):
  - **INSERT:** o trigger é invocado quando uma nova linha é inserida na tabela associada ao trigger;
  - **DELETE**: o trigger é invocado quando uma linha da tabela associada ao trigger é removida;
  - **UPDATE:** o trigger é invocado quando uma linha da tabela associada é atualizada;
  - **UPDATE OF column-list:** o trigger é invocado quando uma linha da tabela associada é atualizada de tal forma que a column-list tem sido modificada.

Triggers (Gatilhos)

- Triggers a nível de linha podem executar **BEFORE** (antes) ou **AFTER** (depois) cada linha seja modificada por um evento do tipo: **INSERT**, **DELETE**, **UPDATE**.

Triggers (Gatilhos)

Ex.1: Cada vez que o valor do saldo for dobrado, incremente o status.

```
CREATE TRIGGER VerificaSaldo
        AFTER UPDATE OF saldo ON Conta
        REFERENCING
                OLD ROW AS antigo,
                NEW ROW AS novo
        FOR EACH ROW
        WHEN (novo.saldo > 2*antigo.saldo)
                UPDATE Conta
                SET Status = Status + 1;
                WHERE numero = novo.numero
```

Definição de Triggers no SQL:1999:

```
<trigger definition> ::=
CREATE TRIGGER <trigger name> <trigger action time> <trigger event>
ON <table name>
[REFERENCING <old or new values alias list>]
<triggered action>
<trigger name> ::= <schema qualified name>
<trigger action time> := BEFORE | AFTER
<trigger event> ::= INSERT| DELETE| UPDATE
                    [ OF <trigger column list> ]
<trigger column list> ::= <column name list>
 <old or new values alias list> ::= <old or new values alias>...
 <old or new values alias> ::= OLD [ ROW ] [ AS ] <old values correlation name>
                             | NEW [ ROW ] [ AS ] <new values correlation name>
                             | OLD TABLE [ AS ] <old values table alias>
                             | NEW TABLE [ AS ] <new values table alias>
<old values correlation name> ::= <correlation name>
 <new values correlation name> ::= <correlation name>
 <old values table alias> ::= <identifier>
 <new values table alias> ::= <identifier>
 <triggered action> ::= [ FOR EACH { ROW | STATEMENT } ]
                    [ WHEN (<trigger condition>) ]
                   <triggered SQL statement>
 <triggered SQL statement> ::= <SQL procedure statement> | BEGIN ATOMIC
                             { <SQL procedure statement> <semicolon> }...
END
```

Trigger

- Observações:

1) Por default, um trigger é statement level
2) Se o trigger for statement-level não se pode usar new e old
3) Dentro da cláusula WHEN não pode invocar uma subquery
4) Ao executar o trigger, verificar se houve erros de compilação

Triggers (Gatilhos)

- **Uso de Instead-of triggers**:
  - Permite que ao invés de realizar o evento (BEFORE|AFTER) use-se o corpo do **trigger** para realizar aquele evento;
  - Embora não seja parte do padrão SQL-1999, é implementado por alguns SGBDs;
  - Muito útil em atualizações de visões.

Triggers (Gatilhos)

- **Exemplo:**

```
CREATE VIEW EmpDSC AS
Select mat, nome, sal
From Empregado
Where Depto = “DSC”;

CREATE TRIGGER InsereDSC
INSTEAD OF INSERT ON EmpDSC
FOR EACH ROW
BEGIN
    INSERT INTO Empregado(mat, nome, sal, depto)
    VALUES (:new.mat, :new.nome, :new.sal, ‘DSC’);
END;
```

Triggers no Oracle

- **Sintaxe básica:** Criação de trigger no Oracle

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
   {BEFORE|AFTER} {INSERT|DELETE|UPDATE|UPDATE OF
<column_name>} ON <table_name>
   [REFERENCING [NEW AS <new_row_name>] [OLD AS
<old_row_name>]]

   [FOR EACH ROW [WHEN (<trigger_condition>)]]

   <trigger_body>

<trigger_body>:
      DECLARE
       declaração de variaveis
      BEGIN
            comandos SQL ou PL/SQL
      END;
```

Triggers no Oracle

- **Obs.:** Pode-se criar apenas **BEFORE** e **AFTER** triggers para tabelas. (**INSTEAD OF** triggers são apenas para views (tipicamente usados em view updates.)
- Pode-se especificar até 3 triggering events usando a palavra-chave **OR**. Além disso, **UPDATE** pode ser opcionalmente seguido pela palavra-chave **OF** e uma lista de atributos na <table_name>.
- **Exemplo:**

... INSERT ON R ...

... INSERT OR DELETE OR UPDATE ON R ...

... UPDATE OF A, B OR INSERT ON R ...

Triggers no Oracle

- Se a opção **FOR EACH ROW** for especificada, o trigger é row-level (válido para cada linha); caso contrário, o trigger é statement-level (válido para o comando).
- Default é Statement-level
- Apenas para triggers row-level:
  - **<trigger_body>** é um bloco PL/SQL, ao invés de uma sequência de comandos SQL
- Restrições no **<trigger_body>** (para evitar, por exemplo, triggers em cascata):
  - Não se pode modificar a mesma relação que está sendo modificada pelo evento do trigger

Exemplo de Trigger:

```
CREATE TABLE T4 (a INTEGER, b CHAR(10));
CREATE TABLE T5 (c CHAR(10), d INTEGER);

/* Inserir uma tupla em T5  quando uma tupla for inserida em  T4: */
CREATE TRIGGER trig1
    AFTER INSERT ON T4
    REFERENCING NEW AS newRow
    FOR EACH ROW
    WHEN (newRow.a <= 10)
    BEGIN
        INSERT INTO T5 VALUES(:newRow.b, :newRow.a);
    END;
```

Triggers no Oracle:

- **Mostrando erros na definição do Trigger:**
  - Como em procedimentos PL/SQL, obtém-se o erro:

```
Warning: Trigger created with compilation errors
```

  - Podemos ver a mensagem de erro:

```
show errors trigger <trigger_name>;
```

ou

```
SELECT *
FROM user_errors
WHERE type = 'TRIGGER' AND
name = 'NOME_DO_TRIGGER'
```

Triggers no Oracle:

- **Para ver os Triggers definidos:**

```
select trigger_name from user_triggers;
```

- **Para maiores detalhes num trigger particular:**

```
select trigger_type, triggering_event, table_name,
referencing_names, trigger_body
from user_triggers
where trigger_name = '<trigger_name>';
```

Triggers no Oracle:

- **Removendo Triggers:**

```
drop trigger <trigger_name>;
```

- **Desabilitando Triggers:**

```
alter trigger <trigger_name> {disable|enable};
```

Triggers no Oracle:

- **Abortando Triggers com erro:**

```
create table Person (age int);

CREATE TRIGGER PersonCheckAge
AFTER INSERT OR UPDATE OF age ON Person
FOR EACH ROW
BEGIN
    IF (:new.age < 0) THEN
        RAISE_APPLICATION_ERROR(-20000, 'Idade negativa!!');
    END IF;
END;
```

Triggers no Oracle:

- Se tentarmos executar:

```
insert into Person values (-3);
```

- Obteríamos a mensagem de erro (e a inserção não seria efetuada):

```
ERROR at line 1:
ORA-20000: Idade negativa!!
ORA-06512: at
"MYNAME.PERSONCHECKAGE", line 3
ORA-04088: error during execution of trigger
'MYNAME.PERSONCHECKAGE'
```

Exemplos Triggers no Oracle:

```
CREATE OR REPLACE TRIGGER Print_salary_changes
  BEFORE  UPDATE ON Emp_tab
  FOR EACH ROW
WHEN (new.Empno > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff  := :new.sal  - :old.sal;
    dbms_output.put('Old salary: ' || :old.sal);
    dbms_output.put('  New salary: ' || :new.sal);
    dbms_output.put_line('  Difference ' || sal_diff);
END;
```

Exemplos Triggers no Oracle:

```
CREATE OR REPLACE TRIGGER trg_pedidos_valor
BEFORE INSERT
ON pedidos
FOR EACH ROW
BEGIN
      IF :NEW.quantidade  >= 10000 THEN
          raise_application_error(-20001,"O valor do pedido
          deve ser inferior a 10.000,00");
      END IF;
END;
```

Exemplos Triggers no Oracle:

```
CREATE OR REPLACE TRIGGER Log_salary_increase
   AFTER UPDATE ON empregado
   FOR EACH ROW
   WHEN (NEW.Sal < 1500) -- menor do que salário mínimo
   BEGIN
       INSERT INTO Emp_log (Emp_id, Log_date,   Alerta)
       VALUES (:NEW.EmpId, SYSDATE, :NEW.sal, 'ABAIXO MIN.');
   END;
```

Exemplos Triggers no Oracle:

```
CREATE OR REPLACE TRIGGER trg_validacao_row_level
    BEFORE INSERT OR UPDATE ON produtos
    FOR EACH ROW
BEGIN
    -- Converte descrição para maiúsculo
    :NEW.descricao := UPPER(:NEW.descricao);

    -- Se estoque for negativo, zera
    IF :NEW.estoque < 0 THEN
        :NEW.estoque := 0;
    END IF;

    -- Se preço for negativo, define como 0.01
    IF :NEW.preco <= 0 THEN
        :NEW.preco := 0.01;
    END IF;
END;
```

Exemplos Triggers no Oracle:

```
CREATE OR REPLACE TRIGGER trg_log_statement_level
    AFTER INSERT ON produtos
BEGIN
  -- Registra apenas uma vez, independente de quantas linhas
  -- foram afetadas
  DBMS OUTPUT.PUT LINE('=======================================');
  DBMS OUTPUT.PUT LINE('OPERAÇÃO DE INSERÇÃO REALIZADA EM: ' ||
TO CHAR(SYSDATE, 'DD/MM/YYYY HH24:MI:SS'));
  DBMS OUTPUT.PUT LINE('USUÁRIO: ' || USER);
  DBMS OUTPUT.PUT LINE('TOTAL DE LINHAS AFETADAS: ' || SQL%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE('=======================================');
END;
```