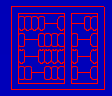
Restrições de integridade

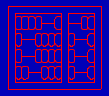


Vantagens de restrições no esquema da BD:

- impede inserções ou atualizações que violem *regras de negócio* específicas da BD;
- verificação de *regras de negócio* pelas aplicações é ineficiente, pois:
  - o código pode ser complexo e incorretamente programado;
  - deve ser repetido em cada aplicação que acessa a BD;
  - novas *regras de negócio* implicam em alteração das aplicações;

UNICAMP

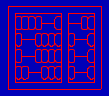
Restrições de integridade



- a centralização das *regras de negócio* no projeto do esquema da BD simplifica o projeto e a administração da BD, garantindo que as regras serão cumpridas;

- as restrições devem fazer parte do projeto da BD desde o início, e incorporadas ao sistema *antes* da BD ser preenchida com dados.

Restrições de integridade



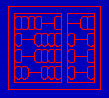
Três tipos de restrições de integridade:

1. restrições sobre atributos de uma tabela base;

2. restrições sobre domínios;

3. *asserções*: restrições de integridade genéricas que podem envolver diversas tabelas e são verificadas quando qualquer uma delas é modificada.

Restrições sobre uma tabela base



1. proibido receber o valor nulo (*not null*);
2. chave primária, chaves estrangeiras e chaves alternativas;
3. valor *default* para atributos.

Chaves alternativas: criadas via qualificador *unique*;
recomendável preceder *unique* com *not null*.
Exemplo:

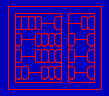
not null unique (nome, pnome)

natp numeric(4) unique,

create [unique] index nome_índice on nome_tabela(nome_coluna(s))

UNICAMP

Restrições do tipo *check (condition)*

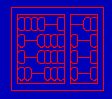


Expressão condicional do tipo “atributo $<$ constante” ou pode conter uma ou mais consultas aninhadas fazendo referências a outras tabelas.

Problema: atualizações em outras tabelas referenciadas pela condição não são verificadas.

Esta é a diferença básica entre *check (condition)* e “asserções”

Exemplos de *check (condition)*



1. “nenhum salário deve ser inferior a $1000”:

   check(salário >= 1000)

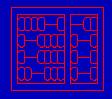
2. “o sexo do funcionário deve ter um dos valores ‘M’ ou ‘F’”:

   check(sex in (‘M’, ‘F’))

3. “nenhum salário deve ser inferior ao mínimo dos salários”:

   check (salário >= (select(min(salário) from Funcionários))

Exemplos de *check (condition)*



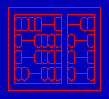
4. “nenhum funcionário deve ter salário superior ao do seu chefe”:

check (salário <= (select f1.salário
from Funcionários f1
where f1.numf = Funcionários.num_chefe))

5. “a média dos salários deve ser superior a $1000”:

check ((select avg(salário) from Funcionários)
> 1000)

Exemplos de *check (condition)*



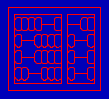
6. “nenhum funcionário pode ter seu salário diminuído.”

Esta restrição não pode ser expressa via *check (condition)* ou via “asserções”:

o salário atual do funcionário precisa ser comparado com o novo salário;

pode, no entanto, ser expressa através de *triggers*.

Exemplos de *check (condition)*



Um ponto importante na verificação da condição:

tudo se passa como se a modificação requerida (por *insert, update ou delete*) fosse feita,

então a condição é avaliada e,

caso falsa (ou *unknown*), a modificação é desfeita.

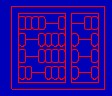
Restrições podem receber um nome:

Constraint nome_restrição check(condição)

Útil e recomendável, pois permite remover ou alterar a restrição.

UNICAMP

Asserções



Mais poderosas que condições;

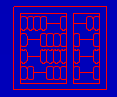
definidas independentemente de quaisquer das tabelas que referenciam;

verificadas quando qualquer uma delas é modificada;

sintaxe:

create assertion nome_asserção check (condition)

Asserções: exemplos



1. *create assertion af1 check (not exists (select * from Funcionários where salário < 1000))*

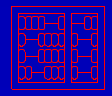
2. *create assertion af2 check (not exists (select * from Funcionários where sex not in ('M', 'F'))*

3. *create assertion af3 check (not exists (select * from Funcionários where (salário < (select(min(salário) from Funcionários)))))*

4. *create assertion af4 check (not exists (select * from Funcionários where (salário > (select f1.salário from Funcionários f1 where f1.numf = Funcionários.num_chefe))))*

UNICAMP

Asserções: exemplos



5. *create assertion af5 check ((select avg(salário) from Funcionários) > 1000)*

Através do comando *alter table* é possível adicionar ou remover restrições:

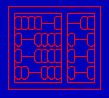
alter table Funcionários add constraint chk_salário check (salário > 1000),

alter table Funcionários drop constraint chk_salário

O comando *drop assertion nome-asserção* permite remover uma asserção.

UNICAMP

Violação de integridade referencial

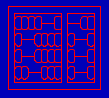


SQL/92: chave estrangeira pode referenciar *qualquer* chave da tabela referenciada.

Ações permitidas quando ocorre violação:
D = Dependentes, F = Funcionários:

- nenhuma ação especificada: impede as operações;
- *on delete set null* e *on update set null*: tuplas órfãs em D têm o valor da chave estrangeira mudado para *Null*;
- *on delete set default*, *on update set default*: chave estrangeira em D deve ter valor *default* predefinido: tuplas órfãs de D têm o seu valor mudado para o valor *default*.

Violação de integridade referencial



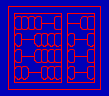
- *on delete cascade*: se *delete* em F gera tuplas órfãs em D, elas são removidas;

  propaga-se recursivamente para tabelas que referenciem via chave estrangeira a chave removida de D.

- *on update cascade*: as tuplas que se tornaram órfãs em D são alteradas para que a chave estrangeira fique igual à da nova chave primária de F;

- inserção de uma tupla órfã em D: única ação possível é rejeitar a inserção.

Violação de integridade referencial



create table Dependentes

(numf int,

nomed char(15) not null,

parentesco char(12) not null,

primary key (numf, nomed, parentesco),

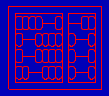
numf references Funcionários(numf)

on update cascade

on delete set null)

UNICAMP

Verificação postergada de restrições

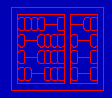


Exemplo ilustrativo: duas tabelas T1 e T2, cada uma delas contendo uma chave estrangeira referenciando a chave primária da outra tabela;

qualquer tentativa de inserir uma linha em T1 (ou em T2), violará a restrição de integridade referencial e falhará:

não há na outra tabela a linha com a chave referenciada pela primeira.

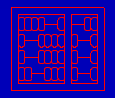
Stored procedures - exemplo



```
create procedure transfer(in:from char(7), in:to
char(7), in:amount decimal(7,2),
out:from_balance decimal(9,2), out :to_balance
decimal(9,2))
begin
update accounts
set balance = balance - :amount
where account_id = :from;
update accounts
set balance = balance + :amount
where account_id = :to;
select f.balance, t.balance
into :from_balance, :to_balance
from accounts f, accounts t
where f.accounts_id = :from and
t.accounts_id=:to;
end
```


UNICAMP

Stored procedures



1. *in* e *out* denotam os parâmetros de entrada e de saída, (se simultaneamente de entrada e de saída: *inout*);

2. comandos SQL separados por ‘;’ funcionam como um único comando SQL;

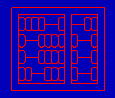
3. cláusula *into*: coloca os resultados de um comando *select* em parâmetros de saída.

transfer poderia ser invocado pela aplicação através da chamada:

exec sql call transfer(’A123456’, ’B987654’, 1000.00, :X, :Y),

UNICAMP

Gatilhos (*triggers*)



São procedimentos remotos especiais:

invocados automaticamente pelo SGBD quando feitas operações (*update, insert* ou *delete*) sobre tabela associada ao gatilho;

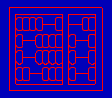
semelhantes a *rotinas de interrupção por software*,

também denominados de *recursos de bases de dados ativas*, pois SGBD participa de forma ativa em certas atualizações da BD;

eles aumentam enormemente o poder da linguagem SQL para definir e impor *regras de negócio* complexas.

UNICAMP

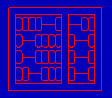
Gatilhos (*triggers*)



Vantagens adicionais:

- manutenção/atualização centralizada de regras de negócio independentemente do número de aplicações da BD; permite inclusive atualizar tabelas não diretamente relacionadas com a tabela associada ao *trigger*;
- permite abortar transações que violem regras de negócio, eliminando seus efeitos sobre a BD;
- auditar atualizações críticas na BD para detectar possíveis violações de segurança;

Gatilhos (*triggers*)

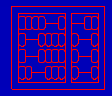


- disparar ações não relacionadas com operações sobre a BD; exemplo:

  enviar um e_mail quando o estado de atributo de alguma tabela chega a um valor crítico

- triggers foram especificados no padrão SQL:1999.

Gatilhos (*triggers*): especificação



Uma única tabela associada: *tabela alvo*,

e uma operação dentre *insert, update, delete*:

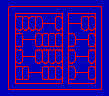
se operação executada por um comando SQL (*triggering statement*), "dispara" (invoca) a execução do gatilho;

para cada operação *insert, update delete*, podemos ter um gatilho diferente;

não impede imediatamente a operação de modificação: antes testa por uma condição opcional especificada na cláusula *when*;

se não for verdadeira, tudo se passa como se o gatilho não tivesse sido invocado, senão a "ação" especificada é executada.

Gatilhos (*triggers*): especificação



A condição pode ser tão complexa quanto desejado: qualquer expressão na cláusula *where* de um comando *select*.

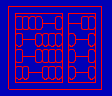
Principais opções de triggers:

- "ação" pode ser executada "antes" ou "depois" das modificações na tabela alvo feitas pelo *triggering statement*;

- "ação" pode referenciar os valores antigos e novos das linhas atualizadas pelo comando SQL, através de duas *tabelas transientes*, *old* e *new*

UNICAMP

Gatilhos (*triggers*): especificação

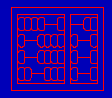


- um gatilho associado a comando *update*, pode especificar uma ou mais colunas: apenas se elas forem atualizadas o gatilho será invocado

- a “ação” pode ser executada de duas formas:

  1. para cada tupla modificada (*for each row*),
  2. uma única vez para todas as tuplas modificadas (um comando SQL pode atualizar várias tuplas).

Gatilhos (*triggers*): um exemplo



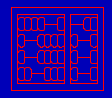
"Criar um gatilho a ser invocado toda vez que a coluna *salário* da tabela Funcionários for modificada por um comando *update*":

várias linhas podem ter o salário atualizado num único comando *update*;

a "ação" do gatilho deve ser executada para cada linha onde foi feita a atualização;

a "ação" deve impedir que funcionários tenham o seu salário diminuído.

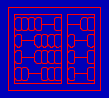
Gatilhos (*triggers*): um exemplo



1. *create trigger verifica_salários*
2. *before update of salário on Funcionários*
3. *referencing old row as old_tuple new row as new_tuple*
4. *for each row*
5. *when (new_tuple.salário < old_tuple.salário)*
6. [*begin atomic*]
7. *set new_tuple.salário = old_tuple.salário*
8. [*end*]

UNICAMP

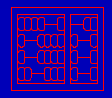
Gatilhos (*triggers*): um exemplo



- linha 1: cria um gatilho com nome *verifica_salários*;
- linha 2: o gatilho será disparado se coluna salário for atualizada por um *update*;
- linha 2: *before* significa que a ação do gatilho deve ser executada “antes” da atualização gerada pelo evento (isto é, pelo comando *update*);
- linha 3: cria um sinônimo para a linha alterada *(old_tuple)* e outro para a linha após a alteração *(new_tuple)*;

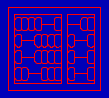
UNICAMP

Gatilhos (*triggers*): um exemplo



- linha 4: a “ação” do gatilho deve ser invocada para cada linha atualizada;
- linha 5: a condição da cláusula *when* só é verdadeira se o salário atualizado é menor que o salário original; *old_tuple* e *new_tuple* contêm exatamente uma linha com os atributos antigos e novos de um funcionário cujo salário foi atualizado;
- linha 6: opcional, usada quando a “ação” requer vários comandos SQL;

Gatilhos (*triggers*): um exemplo

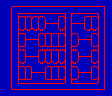


- linha 7: o valor de *new_tuple.salário* é atualizado para refletir o valor anterior do salário; como o trigger é disparado com a opção *before each row* é este valor que será utilizado para atualizar o salário, efetivamente desfazendo a atualização do *triggering statement*.

SQL:1999 permite atualizar a tabela alvo como parte da ação do *trigger* ;

Oracle proíbe tais atualizações e a tabela alvo é eufemisticamente denominada *tabela mutante*; Oracle não permite sequer que seja diretamente consultada como parte da ação do *trigger*.

Gatilhos (*triggers*): outras opções



1. linha 2: em vez de *before* especificar *after*: a ação só será executada “após” a execução do *triggering statement*;

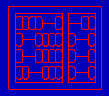
2. linha 2: além de *update* especificar um dentre *insert* ou *delete*: no caso de *update* a cláusula *of* para uma ou mais colunas é opcional;

3. linha 3: se o evento for *insert* apenas a tabela *new* é criada e *old as* é inválido;

   se o evento for *delete*, então *old* conterá as linhas removidas e *new as* é inválido;

UNICAMP

Gatilhos (*triggers*): outras opções

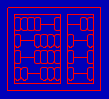


4. linha 4: se *for each row* é especificada o *trigger* é denominado *row trigger*;

   se for omitida o gatilho terá o escopo de todo o comando que o invocou, (sua ação será executada uma única vez): denominado *statement trigger*;

   nesse caso as tabelas *old* e *new* conterão todas as linhas antigas e todas as linhas novas, respectivamente.

Before triggers versus *after triggers*



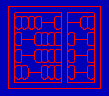
- *before trigger* :

  apropriado para estender o mecanismo de restrições impondo regras de negócio mais complexas, para validar e eventualmente corrigir operações de atualização na BD (como no exemplo de atualização dos salários dos funcionários).

- *after trigger* :

  útil para ações de auditoria, de monitoramento, para emissão de relatórios estatísticos de atualizações da BD, e para ações externas à BD como envio de e_mails ou escrita de arquivos não gerenciados pelo SGBD.

Gatilhos (*triggers*): exercícios



1. É possível tratar de uma só vez todas as modificações causadas pelo evento que invoca o gatilho, omitindo a cláusula *for each row* : modifique o código de *verifica_salários* para este fim;

   (neste caso é mais apropriado usar os sinônimos *old_table* e *new_table* em vez de *old_tuple* e *new_tuple*).

2. É possível executar a ação do gatilho em vez das modificações causadas pelo evento; isto sugere uma forma ainda mais simples de codificar o gatilho *verifica_salários*. Faça isto.