Capítulo 5: Integridade e Segurança

- Restrições ao Domínio
- Integridade Referencial
- Asserções
- Triggers
- Segurança e Autorizações

Restrições ao Domínio

- As restrições de integridade impõem-se para garantir que os dados ficam protegidos contra “estragos” acidentais. Devem assegurar que da actualização dos dados não resulta a perda da consistência.
- Restrições ao domínio são a forma mais elementar de restrição de integridade.
- Testam condições sobre valores a introduzir em atributos, restringindo o domínio do atributo em causa.
- Em SQL, as restrições ao domínio são definidas aquando da definição das tabelas
- A forma mais comum de restrição ao domínio é a proibição do valor **null** (em SQL, colocando **not null** depois do atributo).

Restrições *check*

- Em SQL, podem-se impor outras restrições usando, depois dum nome de atributo A,

  check(*condição(A)***)**

  onde *condição(A)* denota uma condição imposta sobre o atributo A.

- Ou então, após a declaração de todos os atributos, podem-se impor restrições que envolvam mais do que um atributo.

```
create table products (
        product_no integer,
        name varchar2(50),
        price number check (price > 0),
        discounted_price number check (discounted_price > 0),
        check (price > discounted_price));
```

Restrições *check*

```
create table alunos(
        num_aluno number(6) not null,
        nome varchar2(30) not null,
        local varchar2(25),
        data_nsc date not null,
        sexo char(1) not null check ( sexo in ( 'F' , 'M' )  ),
        cod_curso number(3) not null);
```

- As restrições do tipo check:
  - não podem ser definidas sobre uma view.
  - apenas podem referir colunas na tabela onde são definidas i.e. não podem referir colunas noutras tabelas.
  - não podem incluir sub-consultas.

Integridade Referencial

- Garante que um valor que ocorre numa relação para um certo conjunto de atributos também ocorre num outro conjunto de atributos de outra relação.
  - Exemplo: Se "Perryridge" é o nome de uma agência que ocorre num dos tuplos da relação *loan*, então existe um tuplo na relação *branch* para o balcão "Perryridge".
- Definição Formal
  - Sejam $r_1(R_1)$ e $r_2(R_2)$ duas relações com (super-)chaves K_1 e K_2 respectivamente.
  - O subconjunto α de R_2 é uma ***chave externa*** referindo K_1 na relação r_1, se para todo t_2 em r_2 existe um tuplo t_1 em r_1 tal que $t_1[K_1] = t_2[\alpha]$.
  - Uma restrição de integridade referencial é um tipo de dependência de inclusão porque pode ser formalizada através de
    $\prod_{\alpha} (r_2) \subseteq \prod_{K1} (r_1)$
    - e.g. $\prod_{branch\_name}(loan) \subseteq \prod_{branch\_name} (branch)$

Integridade de referência e tuplos soltos

- **Tuplos soltos** – Tuplos que "desaparecem" numa junção.
  - Considere as relações $r_1(R_1)$ e $r_2(R_2)$
  - Um tuplo t de r_i (i = 1 ou i=2) é um tuplo solto se t não pertence a:

    $$\prod_{Ri} (r_1 \bowtie r_2)$$

- Dizer que *K* de r_2 é uma chave externa referindo *K* em r_1 (assumindo que *K* contém todos os atributos comuns a R_1 e R_2) é o mesmo que dizer que:
  - r_2 não pode ter tuplos soltos na junção com r_1, ou que
  - $r_1 \bowtie r_2 = r_1 \ltimes r_2$

Tuplos soltos - Exemplo

empréstimos

número	nome_bacão	valor
A-102	Sete Rios	400
A-216	Almada	750

balcões

nome_balcão	cidade
Sete Rios	Lisboa
Benfica	Lisboa

Tuplo solto

empréstimos ⋈ balcões

número	nome_bacão	valor	cidade
A-102	Sete Rios	400	Lisboa

empréstimos ⟕ balcões

número	nome_bacão	valor	cidade
A-102	Sete Rios	400	Lisboa
NULL	Benfica	NULL	Lisboa

Não causa qualquer problema

Porque a chave extrena é de **nome_bacão** em **empréstimos**, referindo **balcões**, e não o contrário

Tuplos soltos - Exemplo

empréstimos

número	nome_bacão	valor
A-102	Sete Rios	400
A-216	Almada	750

balcões

nome_balcão	cidade
Sete Rios	Lisboa
Benfica	Lisboa

Tuplo solto indesejável

empréstimos ⋈ balcões

número	nome_bacão	valor	cidade
A-102	Sete Rios	400	Lisboa

empréstimos ⟗ balcões

número	nome_bacão	valor	cidade
A-102	Sete Rios	400	Lisboa
A-216	Almada	750	NULL

Não queremos permitir

- Temos que garantir que
  empréstimos ⋈ balcões = empréstimos ⟗ balcões
- ou seja que **nome_bacão** em **empréstimos** é chave externa referindo **balcões**

Verificação da Integridade Referencial e Modificação da Base de Dados

- Aquando das modificações da Base de Dados, é necessário efectuar um conjunto de testes de modo a garantir a preservação da seguinte restrição de integridade referencial:

$$\prod_{\alpha}(r_2) \subseteq \prod_{K}(r_1)$$

- **Inserção.** Se um tuplo t_2 é inserido em r_2, o sistema tem de garantir que existe um tuplo t_1 em r_1 tal que $t_1[K] = t_2[\alpha]$. Ou seja

$$t_2[\alpha] \in \prod_{K}(r_1)$$

- **Remoção.** Se um tuplo t_1 é removido de r_1, o sistema deve calcular o conjunto de tuplos em r_2 que referenciam t_1:

$$\sigma_{\alpha = t1[K]}(r_2)$$

Se este conjunto não é vazio, o comando de remoção é rejeitado, ou os tuplos que referenciam t_1 devem ser eles próprios removidos (remoções em cascata são possíveis).

Modificação da Base de Dados (Cont.)

- **Actualização.** Existem duas situações:
  - Se um tuplo t_2 é actualizado na relação r_2 em que é modificado o valor da chave externa α, então é efectuado um teste similar ao da inserção. Seja t_2' o novo valor do tuplo t_2. O sistema deve garantir que

$$t_2'[\alpha] \in \prod_K(r_1)$$

  - Se o tuplo t_1 é actualizado em r_1, e a operação altera o valor da chave primária (K), então é efectuado um teste semelhante ao da remoção. O sistema deve calcular

$$\sigma_{\alpha = t1[K]}(r_2)$$

    usando o valor anterior de t_1 (o valor antes da actualização). Se o conjunto é não vazio, a actualização:
    - pode ser rejeitada,
    - pode ser propagada em cascata,
    - os tuplos podem ser removidos.

Integridade Referencial em SQL

- As chaves primárias, candidatas e externas podem ser especificadas na instrução SQL **create table**:
  - A cláusula **primary key** da instrução **create table** inclui a lista de dos atributos que formam a chave primária.
  - Uma cláusula **unique key** da instrução **create table** inclui uma lista de atributos que formam uma chave candidata.
  - Uma cláusula **foreign key** da instrução **create table** inclui quer uma lista de atributos que constituem uma chave externa quer o nome da relação (e eventualmente nomes de atributos) referenciada por essa chave externa.

Integridade Referêncial em SQL – Exemplo

create table cursos(cod_curso number(3) **not null**,
nome varchar(35) **not null**
primary key (cod_curso));

create table cadeiras(cod_cadeira number(3) **not null**, …,
primary key (cod_cadeira));

create table curso_cadeira(
cod_curso number(3) **not null**,
cod_cadeira number(3) **not null**,
semestre number(2) **not null**,
primary key (cod_curso, cod_cadeira),
foreign key (cod_curso) **references** cursos,
foreign key (cod_cadeira) **references** cadeiras);

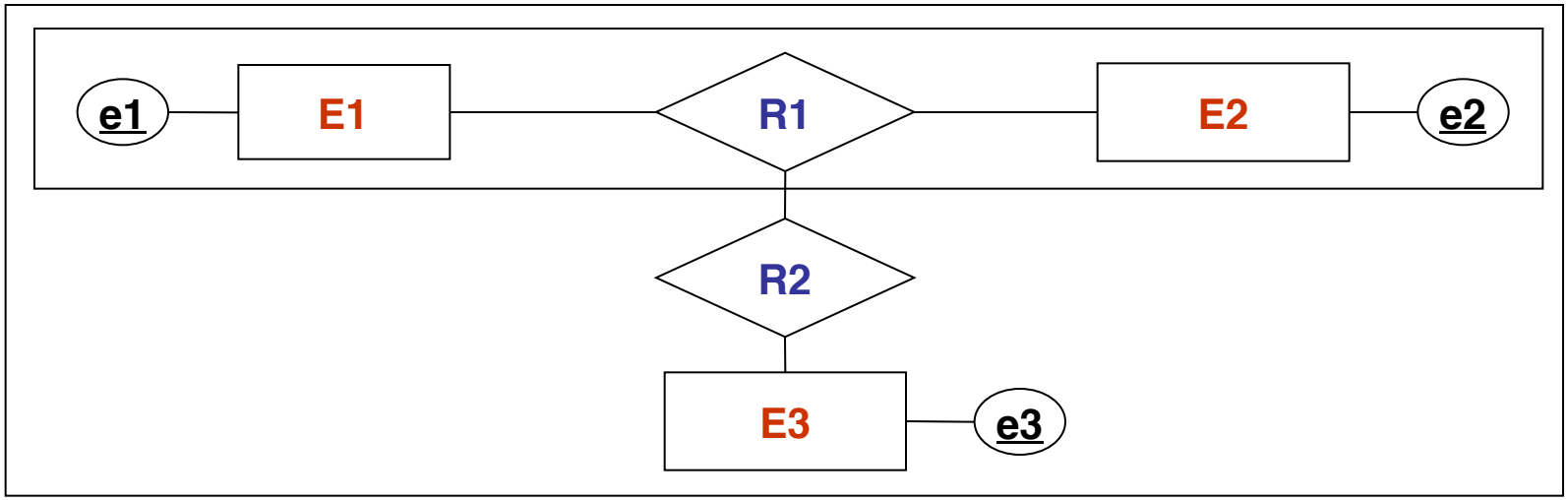
Integridade de referência em SQL

- Em SQL tuplos com valor **null** são ignorados no teste de integridade referencial:
  - Em SQL, se α de R_2 é uma chave externa referindo K_1 na relação r_1 então para todo t_2 em r_2 tal que nenhum dos atributos de α em t_2 tem valor **null**, tem que existir um tuplo t_1 em r_1 tal que $t_1[K_1] = t_2[\alpha]$.
  - Facilita em certos casos: E.g. relação chefe_de em empregados no que se refere ao topo da hierarquia.
  - Quando não se pretende isto, pode-se sempre especificar os atributos em α como sendo **not null**.

Integridade de referência em SQL

- A integridade de referência é verificada apenas no final duma transacção
  - Passos intermédios podem violar a integridade referencial desde que passos posteriores a reponham
  - Caso contrário seria impossível criar alguns estados da base de dados, e.g. inserir dois tuplos cujas chaves externas apontam um para o outro (e.g. atributo *cônjuge* da relação *casado)*

Integridade Referêncial em SQL – Exemplo



create table E1(e1 number(3) **not null primary key**);

create table E2(e2 number(3) **not null primary key**);

create table E3(e3 number(3) **not null primary key**);

create table R1(e1 number(3) **not null**,
e2 number(3) **not null**,
primary key (e1, e2),
foreign key (e1) **references** E1,
foreign key (e2) **references** E2);

create table R2(e1 number(3) **not null**,
e2 number(3) **not null**,
e3 number(3) **not null**,
primary key (e1, e2, e3),
foreign key (e1,e2) **references** R1,
foreign key (e3) **references** E3);

Acções em Cascata em SQL

create table *account*

. . .
foreign key*(branch_name)* **references** *branch*
on delete cascade
on update cascade
. . .)

- Com as cláusulas **on delete cascade**, se a remoção de um tuplo na relação *branch* resulta na violação da restrição da integridade referencial, a remoção propaga-se em "cascata" para a relação *account*, removendo o tuplo que referia a agência que tinha sido eliminada.
- Actualizações em cascata são semelhantes. Não estão implementadas pelo Oracle!

Acções em cascata em SQL (cont.)

- Se existe uma cadeia de dependências de chaves externas através de várias relações, com um **on delete cascade** especificado em cada dependência, uma remoção ou actualização num dos extremos pode-se propagar através de toda a cadeia.
- Se uma remoção ou actualização em cascata origina uma violação de uma restrição que não pode ser tratada por uma outra operação em cascata, o sistema aborta a transacção. Como resultado, todas as alterações provocadas pela transacção e respectivas acções em cascata serão anuladas.
- Alternativas às operações em cascata:
  - **on delete set null**
  - **on delete set default**

Asserções

- Uma *asserção* é um predicado que exprime uma condição que gostaríamos de ver sempre satisfeita na base de dados.
- Em SQL as asserções têm a forma:

  create assertion <nome> **check** <predicado>

- Quando se define uma asserção, o sistema testa-a, e volta a testá-la, sempre que há modificações na base de dados (que a possam violar)
  - Estes testes podem introduzir um overhead significativo; logo as asserções são para usar com cuidado e de forma comedida.
- Impor uma condição da forma "para todo o X, P(X)", há que o fazer usando uma dupla negação: "não existe X tal que not P(X)"
- O Oracle não permite definir asserções.

Exemplo de Asserção

- Em cada balcão, a soma dos montantes de todos os seus empréstimos tem que ser sempre inferior à soma de todos os seus depósitos.

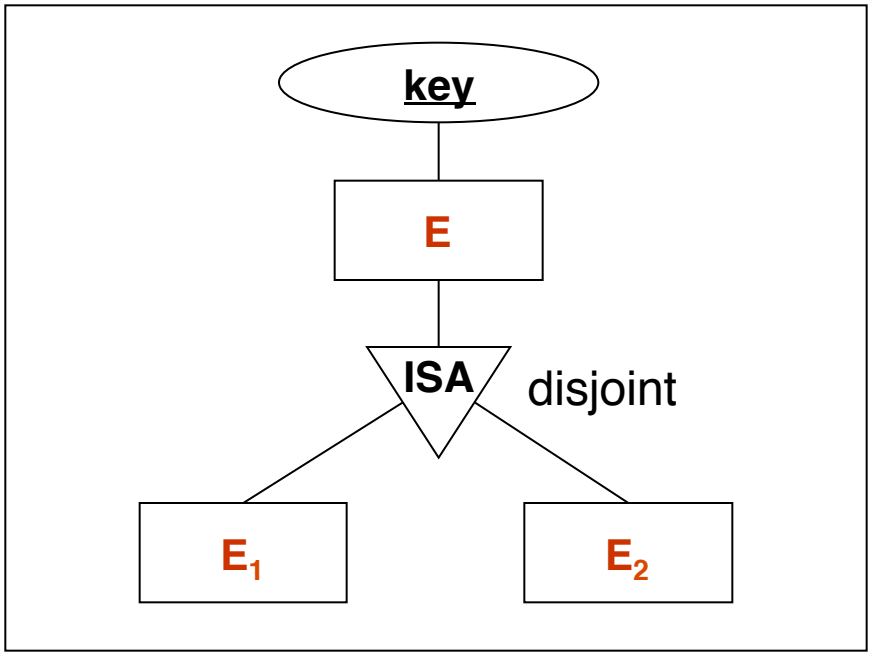
```
create assertion sum_constraint check
    (not exists (select * from branch
                 where
                     (select sum(amount) from loan
                        where loan.branch_name =
                              branch.branch_name
                     )
                    >= some
                     (select sum(balance) from account
                        where account.branch_name =
                              branch.branch_name
                     )
                 )
    )
```

Outro Exemplo

- Todo o empréstimo tem que estar sempre ligado a pelo menos um cliente de uma conta (de depósito) cujo saldo é não inferior a metade do valor do empréstimo

```
create assertion balance_constraint check
   (not exists (
       select * from loan
       where not exists (
           select *
           from borrower natural inner join depositor
                                natural inner join account
           where loan.loan_number = borrower.loan_number
                 and account.balance >= 0,5 * loan.ammount
                                )
                  )
    )
```

Exemplo de Asserção



- Uma especialização duma entidade geral E (com chave key) em E_1 e E_2 é disjunta.

create assertion *disjE1E2* **check**
(not exists ((select *key* **from** E_1**) intersect (select** *key* **from** E_2**)))**

Triggers

- Um **trigger** é um "comando" que é executado automaticamente pelo sistema, como side-effect duma modificação à base de dados dum determinado tipo pré-definido.
- Para definir um trigger, há que:
  - Especificar que *evento* faz disparar trigger
  - Especificar em que *condições* o trigger deve ser executado.
  - Especificar que *acção* fazer quando o trigger é executado.
- São conhecidos como **event-condition-action rules**
- Os triggers são armazenados na base de dados, e executados para todos as interacções com esta.
- O Oracle suporta triggers, embora com uma sintaxe ligeiramente diferente da do SQL.

Exemplo de Trigger

- Imagine uma situação em que o banco aceita que haja saldos negativos e, nesses casos:
  - coloca o saldo a 0
  - cria um empréstimo com o valor em dívida
  - Atribui a este empréstimo um número idêntico ao da conta de depósito
- O trigger deve ser executado sempre que há uma actualização na relação *account* que faz com que o saldo passe a negativo.

Codificação do Exemplo em SQL:1999

```
create trigger overdraft_trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
        insert into borrower
           (select customer_name, account_number
            from depositor
            where nrow.account_number =
                        depositor.account_number);
        insert into loan values
           (nrow.account_number, nrow.branch_name,
                                                – nrow.balance);
        update account set balance = 0
        where account.account_number = nrow.account_number
end
```

Eventos e Acções de Triggers em SQL

- Os eventos que podem fazer disparar um trigger são **insert**, **delete** ou **update**
- No Oracle, também podem disparar triggers eventos de **servererror**, **logon**, **logoff**, **startup** e **shutdown**.
- Triggers sobre **update** podem-se restringuir só a alguns atributos
  - **E.g. create trigger** *overdraft_trigger* **after update of** *balance* **on** *account*
- Pode-se referenciar o valor dos atributos antes e depois da modificação
  - **referencing old row as** : para **delete**s e **update**s
  - **referencing new row as** **:** para **insert**s e **update**s
- Pode-se fazer disparar um trigger antes do evento, para codificar restrições. E.g. converter espaços em **null**.

```
create trigger setnull_trigger before update on r
referencing new row as nrow
for each row
    when nrow.phone_number = ' '
    set nrow.phone_number = null
```

- Para além do **before** e do **after** no Oracle existe também o **instead of.**

Acções Externas

- Por vezes podemos querer que um dado evento faça disparar uma acção para o exterior.
  - Por exemplo, numa base de dados de uma armazém, sempre que a quantidade de um produto desce abaixo (devido a um **update)** de um determinado valor podemos querer encomendar esse produto, ou disparar algum alarme.
- Os triggers não podem ser usados para implementar acções sobre o exterior, mas...
  - podem ser usados para guardar numa tabela separada acções-a-levar-a-cabo. Podem depois haver procedimentos que, periodicamente verificam essa tabela separada.
- E.g. Uma base de um armazém com as tabelas
  - *inventario(item, quant):* Que quantidade há de cada produto
  - *quantMin(item, quant) :* Qual a quantidade mínima de cada produto
  - *reposicoes(item, quant):* Quanto encomendar sempre que está em falta
  - *aencomendar(item, quant) :* Coisas a encomendar (lido por procedimento)

Exemplo de Acções Externas

```
create trigger aenc_trigger after update of quant on inventario
referencing old row as orow, new row as nrow
for each row
        when nrow.quant <= some (select quant
                                 from quantMin
                                 where quantMin.item = orow.item)
                and orow.quant > some (select quant
                                       from quantMin
                                       where quantMin.item = orow.item)
  begin
          insert into aencomendar
                (select item, quant
                 from reposicoes
                 where reposicoes.item = orow.item)
  end
```

Sintaxe de Triggers em Oracle

create [**or replace**] **trigger** *<nome_trigger>*
{**before** | **after** [**instead of**} *<evento>*
[**referencing old as** *<nome_antes>*]
[**referencing new as** *<nome_depois>*]
for each row
when *<condição>*
begin
<Sequencia de comandos, terminados por ;>
end;
/

- *Evento* pode ser:
  - **delete on** *<tabela ou view>*
  - **insert on** *<tabela ou view>*
  - **update on** *<tabela ou view>*
  - **update of** *<atributos separados por ,>* **on** *<tabela ou view>*
  - **servererror**, **logon**, **logoff**, **startup** ou **shutdown**
- Os comandos são PL/SQL o que inclui os comandos SQL, mais WHILEs, IFs, etc (ver manuais)
- Dentro da condição os *nome_antes* e *nome_depois* podem ser usados sem mais. Mas nos comandos têm que ter o símbolo ‘:’ antes!!!

Statement Triggers

- São executados após (antes, ou em vez de) uma instrução completa vs. os anteriores que são executadas após alterações em cada linha
- Sintaxe:

  create [**or replace**] **trigger** *<nome_trigger>*
  {**before** | **after** | **instead of**} *<evento>*
  begin
  <Sequencia de comandos, terminados por ;>
  end;

- Para ser usado quando as condições são para testar globalmente e não linha a linha.

Uso de triggers

- Podem usar-se para implementar assertions, fazendo raise_application_error quando as condições não se verificam.
- Não usar triggers:
  - Quando as restrições podem ser impostas doutra forma (com a excepção das asserções)!!
    - Os triggers são mais difíceis de manter e são menos eficientes.
  - Quando se querem manter sumários
    - Para tal usem-se views e se eficiência for importante usem-se materialized views
- Os triggers permitem uma grande generalidade na imposição de restrições e, também por isso mesmo, devem ser usados com grande cuidado.

Exemplo

```
create table alunos( num_aluno number(6) not null, …,
                     cod_curso number(3) not null,
                     primary key (num_aluno),
                     unique (num_aluno, cod_curso)
                     foreign key cod_curso references curso);

create table curso_cadeira(
          cod_curso number(3) not null,
          cod_cadeira number(3) not null, …,
          primary key (cod_curso, cod_cadeira), …);

create table inscricoes( num_aluno number(6) not null,
                         cod_curso number(3) not null,
                         cod_cadeira number(5) not null,
                         data_inscricao date not null, …,
          primary key (num_aluno, cod_curso, cod_cadeira,
    data_inscricao),
          foreign key (num_aluno, cod_curso)
                         references alunos(num_aluno, cod_curso),
          foreign key (cod_curso, cod_cadeira) references curso_cadeira);
```

Triggers para actualização de vistas

- Podemos utilizar triggers para efectuar modificações através de vistas.
- Para tal, criamos triggers para todas as operações permitidas, como por exemplo:
  - para a inserção (do tipo instead of insert on),
  - para a remoção (do tipo instead of delete on)
  - para a actualização (do tipo instead of update on).
- Consideremos a vista:

create view *info_empréstimos* **as**
select *loan_number, customer_name, amount*
from *borrower* **natural inner join** *loan*

Triggers para actualização de vistas

- Se quisermos permitir a remoção de empréstimos através da vista, criamos o trigger:

```
create trigger remove_empréstimos
        instead of delete on info_empréstimos
        referencing old row as orow
        for each row
begin
        delete from loan
                where loan_number = orow. loan_number ;
        delete from borrower
                where loan_number = orow. loan_number ;
end
```

Triggers para actualização de vistas

- Se quisermos permitir a inserção de empréstimos através da vista, criamos o trigger:

```
create trigger insere_empréstimos
        instead of insert on info_empréstimos
        referencing new row as nrow
        for each row
begin
        insert into loan
                values (nrow.loan_number, NULL, amount);
        insert into borrower
                values (nrow.customer_name, nrow.loan_number)
end
```

Triggers para actualização de vistas

- Se quisermos permitir a actualização do valor do empréstimo através da vista, criamos o trigger:

```
create trigger actualiza_empréstimos
        instead of update of amount on info_empréstimos
        referencing new row as nrow
        referencing old row as orow
        for each row
begin
        update loan
        set amount = nrow. amount
                where loan_number = orow. loan_number ;
end
```

Triggers para inserção de chaves

- Se quisermos preencher automaticamente a chave de um tuplo, aquando da sua inserção, recorrendo a uma sequência:

```
create trigger chave_aluno
before insert on alunos
for each row
declare
        aluno_id number;
begin
        select seq_aluno.nextval
        into aluno_id
        from dual;
        :new.num_aluno := aluno_id;
end
```