Projeto e operação de Bancos de Dados - Parte 3

José de J. Pérez Alcázar
PhD.
EACH - USP

SQL (Structured Query Language)

Originalmente SQL foi chamado SEQUEL (Structured English Query Language)

Projetado e implementado pela IBM como interface para o sistema R. SQL linguagem dos SGBDs.

Primeiro esforço pela sua padronização ANSI 1986 : SQL1

Expansão do padrão em 1992 → SQL2

SQL99 (SQL3) estendeu SQL com facilidades de OO e outros conceitos.

SQL (LDD e LMD). Além disso permite definir visões, criar e eliminar índices (isto tem sido eliminado em SQL2) e incorporar comandos SQL num LP.

SQL (Structured Query Language)

DEFINIÇÃO DE DADOS EM SQL.

Termos em SQL → Tabela, linha e coluna

CREATE Table

ALTER Table

DROP Table

Versões iniciais de SQL não incluíam o conceito de esquema relacional. Agrupação de tabelas e outras construções que pertencem à mesma aplicação.

Um esquema SQL é identificado por um nome, e inclui um identificador de autorização, bem como descritores de cada elemento do esquema.

SQL (Structured Query Language)

CREATE SCHEMA EMPRESA AUTHORIZATION JPEREZ;

SQL2 também usa o conceito de CATALOG (uma coleção de esquemas num ambiente SQL). Um catálogo contem um esquema especial INFORMATION_SCHEMA (informação de todos os descritores de elementos de todos os esquemas)

Integridade referencial → só entre relações de um mesmo catálogo (compartilhar domínios)

COMANDO CREATE TABLE EM SQL2

- O esquema pode ser implicitamente especificado o explicitamente.
- Exemplo: CREATE TABLE EMPRESA.EMPREGADO

LDD – Criação de Tabelas

- CREATE TABLE - cria uma tabela (tabela base), e define colunas e restrições

```
CREATE TABLE [esquema].tabela (
        atrib1 tipo [<restrições da coluna 1>],
        atrib2 tipo [<restrições da coluna 2>],
        ....
        atribn tipo [<restrições da coluna n>],
        <restrições da tabela>
);
```

LDD – Criação de Tabelas

- Restrições de colunas
  - NOT NULL
  - DEFAULT *valor*
  - CHECK(*condição*)

CREATE TABLE [esquema].*tabela* (
atrib1 tipo [(*tamanho*)] [NOT NULL | DEFAULT *valor*]
[CHECK *(condição)*],
atrib2 tipo [(*tamanho*)] [NOT NULL | DEFAULT *valor*]
[CHECK *(condição)*],
...

LDD – Criação de Tabelas

- Restrições de tabela
  - PRIMARY KEY (*<atributos chave primária>*)
  - UNIQUE (*<atributos chave candidata>*)
  - FOREIGN KEY (*<atributos chave estrangeira>*
    REFERENCES *tabelaRef* [(*<chave primária>*)] [*<ações>*]

- <ações>
  - ON DELETE | ON UPDATE
  - CASCADE | SET NULL | SET DEFAULT
  - CHECK(*condição*)

Restrições de integridade referencial podem ser violadas quando tuplas são incluídas ou eliminadas ou quando atributos que são chaves estrangeiras são modificadas.

SQL – Alguns tipos de dados

- INTEGER | SMALLINT
- DECIMAL [(precision, scale)] - precision é o número total de dígitos total e scale é o número de dígitos depois do ponto
- DOUBLE PRECISION | FLOAT | REAL
- CHAR(n) - tamanho fixo - n caracteres
- VARCHAR(n) - tamanho variável – máximo de n caracteres
- BLOB – Binary Large Object
- DATE | TIME | TIMESTAMP
- ...

LDD – Criação de Domínios

- CREATE DOMAIN – Utilizado para definir domínios de atributos.

  CREATE DOMAIN *nome* AS *tipo [<restrições de coluna>]*

  - Facilita a redefinição de tipos de dados de um domínio utilizados por muitos atributos de um esquema, além de melhorar a legibilidade do esquema.
- Por exemplo:

  CREATE DOMAIN TIPO_NSS AS CHAR(9);

LDD – Criação de domínios

- Pode-se definir um novo domínio com a especificação de uma restrição sobre o tipo de dados.
- Por exemplo:

```
CREATE DOMAIN TIPO_DEPNUM AS INTEGER
        CHECK (TIPO_DEPNUM > 0 AND TIPO_DEPNUM < 21);
```

LDD – Criação de Tabelas

Forma geral:

```
CREATE TABLE [esquema].tabela (
    atrib1 tipo [(tamanho)] [NOT NULL | DEFAULT valor] [CHECK (condição)],
    atrib2 tipo [(tamanho)] [NOT NULL | DEFAULT valor] [CHECK (condição)],
    ...
    [CONSTRAINT nome da restrição]
      PRIMARY KEY (<atributos chave primária>),
    [CONSTRAINT nome da restrição]
      UNIQUE (< atributos chave candidata>),
    [CONSTRAINT nome da restrição]
      FOREIGN KEY (<atributos chave estrangeira>)
          REFERENCES tabelaRef [(<chave primária>)]
              [ON DELETE CASCADE | SET NULL | SET DEFAULT]
              [ON UPDATE CASCADE | SET NULL | SET DEFAULT],
    [CONSTRAINT nome da restrição]
      CHECK (condição)
  );
```

Comandos de definição de dados SQL CREATE TABLE para a definição do esquema EMPRESA

(a)

```
CREATE TABLE EMPREGADO
        ( FNOME             VARCHAR(15)         NOT NULL ,
          MINICIAL          CHAR ,
          LNOME             VARCHAR(15)         NOT NULL ,
          SSN               CHAR(9)             NOT NULL ,
          DATANASC          DATE
          ENDERECO          VARCHAR(30) ,
          SEXO              CHAR ,
          SALARIO           DECIMAL(10,2) ,
          SUPERSSN          CHAR(9) ,
          DNO               INT                 NOT NULL ,
        PRIMARY KEY (SSN) ,
        FOREIGN KEY (SUPERSSN) REFERENCES EMPREGADO(SSN) ,
        FOREIGN KEY (DNO) REFERENCES DEPARTAMENTO(DNUM) ) ;
CREATE TABLE DEPARTAMENTO
        ( DNOME             VARCHAR(15)         NOT NULL ,
          DNUMERO           INT                 NOT NULL ,
          GERSSN            CHAR(9)             NOT NULL ,
          GERDATAINICIO     DATE ,
        PRIMARY KEY (DNUM) ,
        UNIQUE (DNOME) ,
        FOREIGN KEY (MGRSSN) REFERENCES EMPREGADO(SSN) ) ;
CREATE TABLE DEPT_LOCALIZACOES
        ( DNUM              INT                 NOT NULL ,
          DLOCACAO          VARCHAR(15)         NOT NULL ,
        PRIMARY KEY (DNUM, DLOCACAO) ,
        FOREIGN KEY (DNUM) REFERENCES DEPARTAMENTO(DNUM) ) ;
CREATE TABLE PROJETO
        ( PNOME             VARCHAR(15)         NOT NULL ,
          PNUMERO           INT                 NOT NULL ,
          PLOCALIZACAO      VARCHAR(15) ,
          DNUM              INT                 NOT NULL ,
```

Chave Principal

Chaves alternativas

Chave estrangeira

⇨ Integridade Referencial

Comandos de definição de dados SQL
CREATE TABLE para a definição do esquema EMPRESA (continuação)

```
PRIMARY KEY (PNUM) ,
UNIQUE (PNOME) ,
FOREIGN KEY (DNU) REFERENCES DEPARTAMENTO(DNUM) ) ;
CREATE TABLE TRABALHA_EM
        ( ESSN              CHAR(9)          NOT NULL ,
          PNO               INT              NOT NULL ,
          HORAS             DECIMAL(3,1)     NOT NULL ,
PRIMARY KEY (ESSN, PNO) ,
FOREIGN KEY (ESSN) REFERENCES EMPREGADO(SSN) ,
FOREIGN KEY (PNO) REFERENCES PROJETO(PNUM) ) ;
CREATE TABLE DEPENDENTE
        ( ESSN              CHAR(9)          NOT NULL ,
          DEPENDENT_NAME    VARCHAR(15)      NOT NULL ,
          SEX               CHAR ,
          DATANASC          DATE ,
          PARENTESCO        VARCHAR(8) ,
PRIMARY KEY (ESSN, DEPENDENTE_NOME) ,
FOREIGN KEY (ESSN) REFERENCES EMPREGADO(SSN) ) ;
```

Exemplo ilustrando como os valores do atributo *default* e as ações referenciais engatilhadas são especificados em SQL.

```
CREATE TABLE EMPREGADO
        ( . . . ,
          DNO           INT      NOT NULL      DEFAULT 1,
        CONSTRAINT EMPPK
          PRIMARY KEY (SSN) ,
        CONSTRAINT EMPSUPERFK
          FOREIGN KEY (SUPERSSN) REFERENCES EMPREGADO(SSN)
                        ON DELETE SET NULL ON UPDATE CASCADE ,
        CONSTRAINT EMPDEPTFK
          FOREIGN KEY (DNO) REFERENCES DEPARTAMENTO(DNUMERO)
                        ON DELETE SET DEFAULT ON UPDATE CASCADE );

CREATE TABLE DEPARTAMENTO
        ( . . . ,
          GERSSN CHAR(9) NOT NULL DEFAULT '888665555' ,
          . . . ,
          CONSTRAINT DEPTPK
            PRIMARY KEY (DNUMERO) ,
          CONSTRAINT DEPTSK
            UNIQUE (DNOME),
          CONSTRAINT DEPTMGRFK
            FOREIGN KEY (GERSSN) REFERENCES EMPREGADO(SSN)
                  ON DELETE SET DEFAULT ON UPDATE CASCADE );

CREATE TABLE DEP_LOCALIZACOES
        ( . . . ,
          PRIMARY KEY (DNUMERO, DLOCALIZACAO),
          FOREIGN KEY (DNUMERO) REFERENCES DEPARTAMENTO(DNUMERO)
          ON DELETE CASCADE ON UPDATE CASCADE ) ;
```

Initially deferred.

Set Constraints <constraints names> immediate (deferred)

- Se precisar renomear uma tabela, use o seguinte comando:
- rename nome_tabela_atual to novo_nome_tabela;
- rename cliente to novocliente;

Visualizando a lista de tabelas em Oracle

FIGURE 3-3 Listing names of all tables

LDD – Remoção de Esquema

- DROP SCHEMA - exclui um esquema do banco de dados

  DROP SCHEMA *esquema* [CASCADE | RESTRICT];

  - CASCADE: todos os elementos do esquema são removidos automaticamente
  - RESTRICT: o esquema só será removido se não existir os elementos
- Por exemplo:

  DROP SCHEMA COMPANHIA **CASCADE**;

LDD – Remoção de Tabelas

- DROP TABLE - exclui uma tabela do banco de dados

  DROP TABLE *tabela* [CASCADE | RESTRICT];

  - CASCADE: todas as visões e restrições que referenciam a tabela são removidas automaticamente
  - RESTRICT: a tabela é removida somente se não for referenciada em nenhuma restrição ou visão
- Por exemplo:

  DROP TABLE COMPANHIA.DEPENDENTE **CASCADE**

Em Oracle

- Para remover uma tabela do BD. Exemplo:
- DROP TABLE nome_da_tabela
- Se a tabela não existe ocorrerá um erro.
- Se a tabela eliminada possuir uma chave primária ou única referenciada por FOREIGN KEYS de outras tabelas →
- DROP Table nomedatabela CASCADE Constraints;

- DROP Table em Oracle não remove definitivamente.

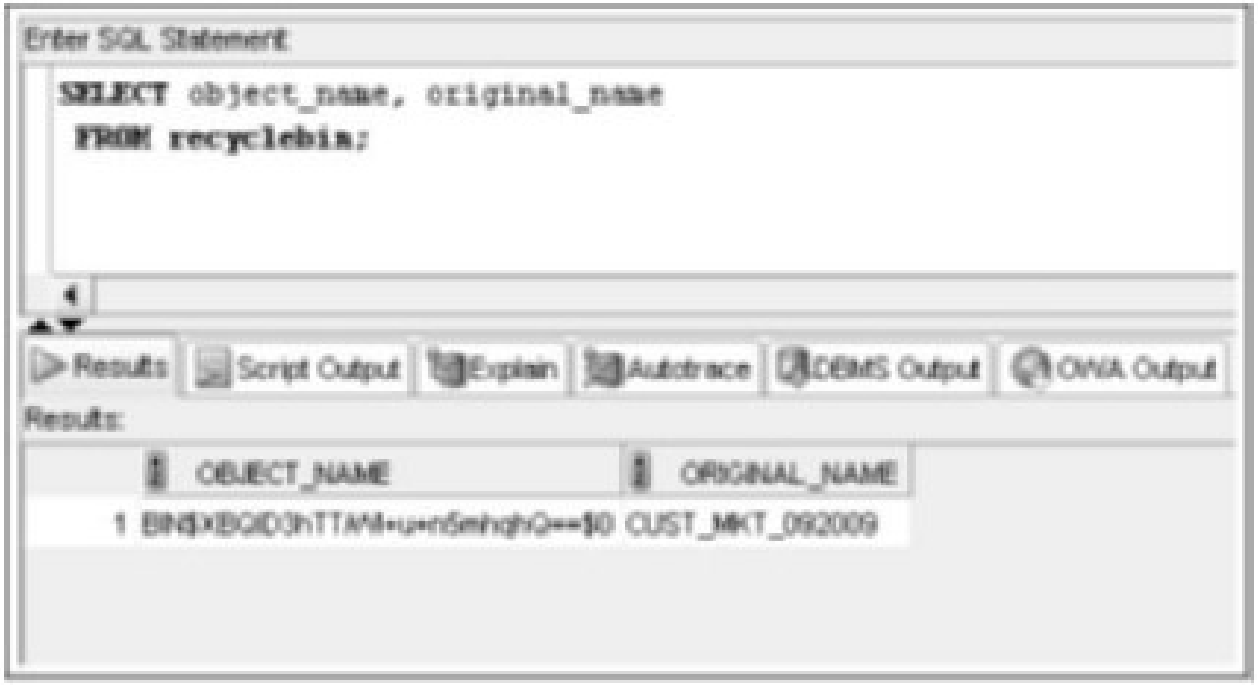


FIGURE 3-36 Checking the recycle bin

FIGURE 3-37 Using FLASHBACK TABLE to restore a dropped table

LDD – Alteração da Tabelas

- ALTER TABLE – incluir/alterar/remover definições de colunas e restrições

  ALTER TABLE *tabela <ação>;*

- *<ação>:*
  - ADD *novoAtrib tipo* [*<restrições de coluna>*]
  - ADD [CONSTRAIN *nome*] *<restrição de tabela>*
  - DROP *atributo* [CASCADE | RESTRICT]
  - DROP CONSTRAINT *nome*

LDD – Alteração da Tabelas

- ADD *novoAtrib tipo* [<*restrições de coluna*>]
  - E o valor do novo atributo nas tuplas já existentes?
    - Se não for especificada nenhuma cláusula default, então o valor será null. Assim, a cláusula NOT NULL não pode ser aplicada.
- Por exemplo:

```
ALTER TABLE COMPANHIA.EMPREGADO
        ADD FUNCAO VARCHAR(12);
```

LDD – Alteração da Tabelas

- DROP *atributo* [CASCADE | RESTRICT]
  - CASCADE – todas as visões e restrições (*constrains*) que referenciam o atributo são removidas automaticamente
  - RESTRICT – o atributo só é removido se não houver nenhuma visão ou restrição que o referencie
- Por exemplo:

```
ALTER TABLE COMPANHIA.EMPREGADO
        DROP FUNCAO CASCADE;
```

LDD – Alteração da Tabelas

- É possível também mudar uma definição de colunas, eliminando uma cláusula DEFAULT ou adicionando uma.
- ALTER TABLE COMPANHIA.DEPARTMENTO ALTER MGRSSN DROP DEFAULT;
- ALTER TABLE COMPANHIA.DEPARTMENTO ALTER MGRSSN SET DEFAULT “333445559”;
- É possível mudar as restrições especificadas sobre uma tabela, adicionando ou eliminando uma restrição (deve ser dado um nome)
- ALTER TABLE COMPANHIA.EMPREGADO DROP CONSTRAINT EMPSUPERFK CASCADE;
- ADD (pode ser nomeada ou não)

Em Oracle

- Exemplo de dessabilitação de Constraints:
- CREATE TABLE dept1

  (deptno number(2) primary key disable,

  dname varchar(10),

  loc varchar2(9));
- Se você inserir duas tuplas com o mesmo deptno o sistema permite.
- Ao ativar a restrição:
- ALTER TABLE dept1 ENABLE PRIMARY KEY;
- O que acontece?

FIGURE 4-36 Disabling and enabling constraints

Sintaxe do Alter table em Oracle

```
ALTER TABLE tablename
ADD (columnname datatype, [DEFAULT] …);
```

FIGURE 3-11 Syntax of the ALTER TABLE . . . ADD command

```
ALTER TABLE tablename
MODIFY (columnname datatype [DEFAULT] ,...);
```

FIGURE 3-13 Syntax of the ALTER TABLE . . . MODIFY command

```
ALTER TABLE tablename
DROP COLUMN columnname;
```

FIGURE 3-19 Syntax of the ALTER TABLE . . . DROP COLUMN command

Em Oracle

- Modificando uma coluna: Usa o opção Modify. Seja a tabela Emp:
  - NomeNull Type
  - ...
  - Sal Number(7,2)
  - ...
- Alter Table Emp Modify (sal number Constraint nn_sal Not Null);

- Adicionando Colunas:
- Por meio da opção ADD podemos acrescentar novas colunas à tabela. Exemplo:
- Alter table dept1 add (state varchar(2));

- Modificando uma coluna:
  - Para alterar uma coluna existente, use a opção MODIFY respeitando as limitações da tabela.
  - Alter table dept1 modify (state varchar(4));
  - Alter table Emp2 modify (sal default 0);

- Para renomear uma coluna:
- alter table cliente rename column nome_cliente to nome;

Regras do Modify

- Uma coluna deve ser no mínimo do tamanho dos campos que ele já contêm.
- Se uma coluna NUMBER já contem dados, você não pode decrementar a precisão ou escala das colunas.
- Mudar o valor default de uma coluna não muda o valor dos dados já existentes.
- Eliminando uma coluna:
- Usando a opção DROP. Exemplo:
- Alter table Emp2 drop (comm).

Visualização de Restrições

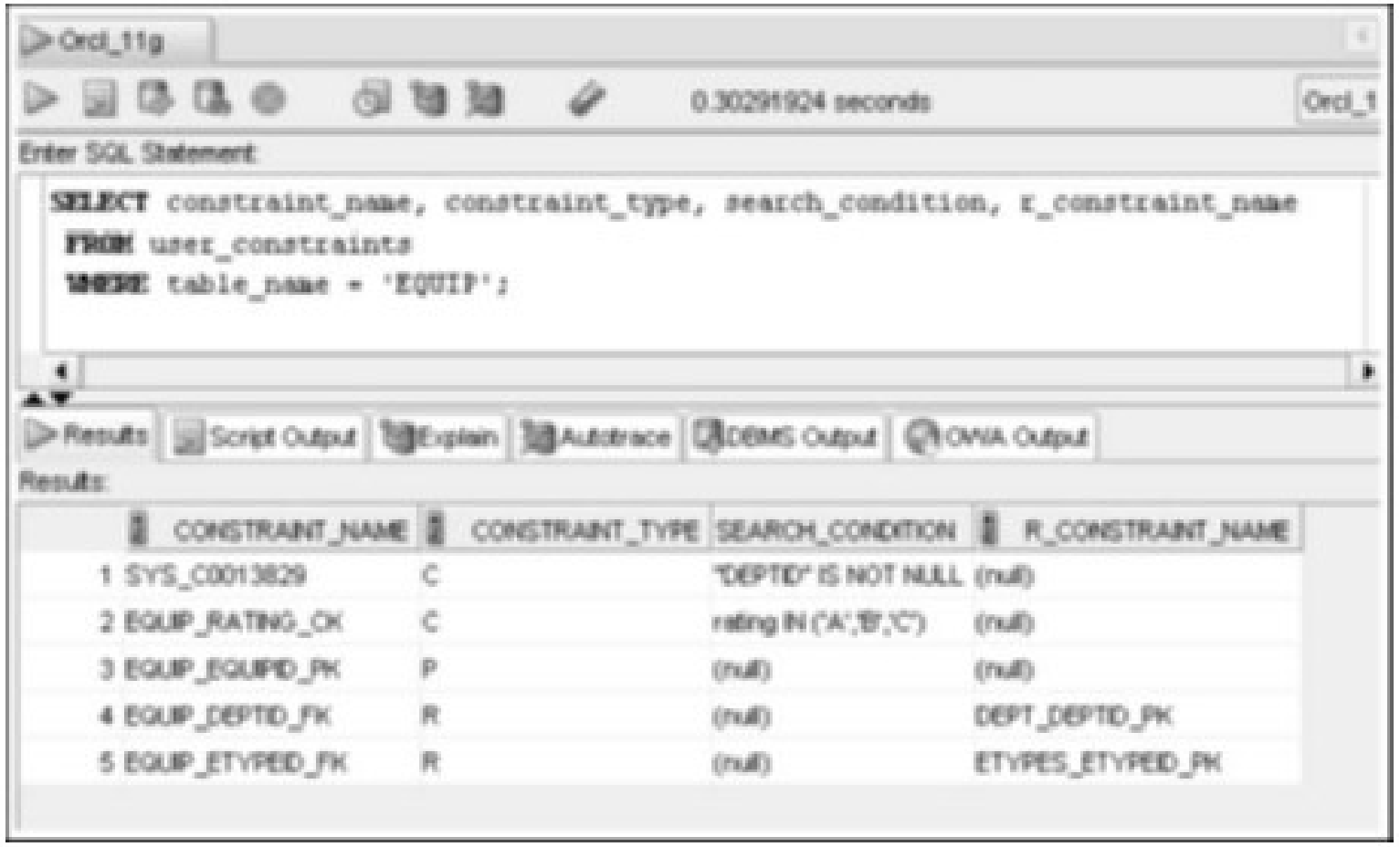


FIGURE 4-33 SELECT statement to view data about existing constraints

LMD - Consulta

- SELECT – Comando de consulta.
  - Forma geral:

SELECT [**DISTINCT** | **ALL**] <*lista de atributos*>
FROM <*lista de tabelas*>
[**WHERE** <*condições*>]
[**GROUP BY** *atributo*]
[**HAVING** <*condições*>]
[**ORDER BY** *atributo* [**ASC** | **DESC**]];

LMD - Consulta

- SELECT – seleciona O QUE se deseja na tabela resultado:
  - **<lista de atributos>** ou * (para todos os atributos)
  - **ALL** – inclui tuplas duplicadas (é o default)
  - **DISTINCT** – elimina tuplas duplicadas
  - **FROM** – DE ONDE retirar os dados necessários
  - **WHERE** – CONDIÇÕES de seleção dos resultados.

SQL (Structured Query Language)

Consiste de três cláusulas:

Select: Corresponde à operação de projeção da álgebra.

From: Lista as relações que vão ser a examinadas na avaliação da expressão. Corresponde ao produto cartesiano da álgebra.

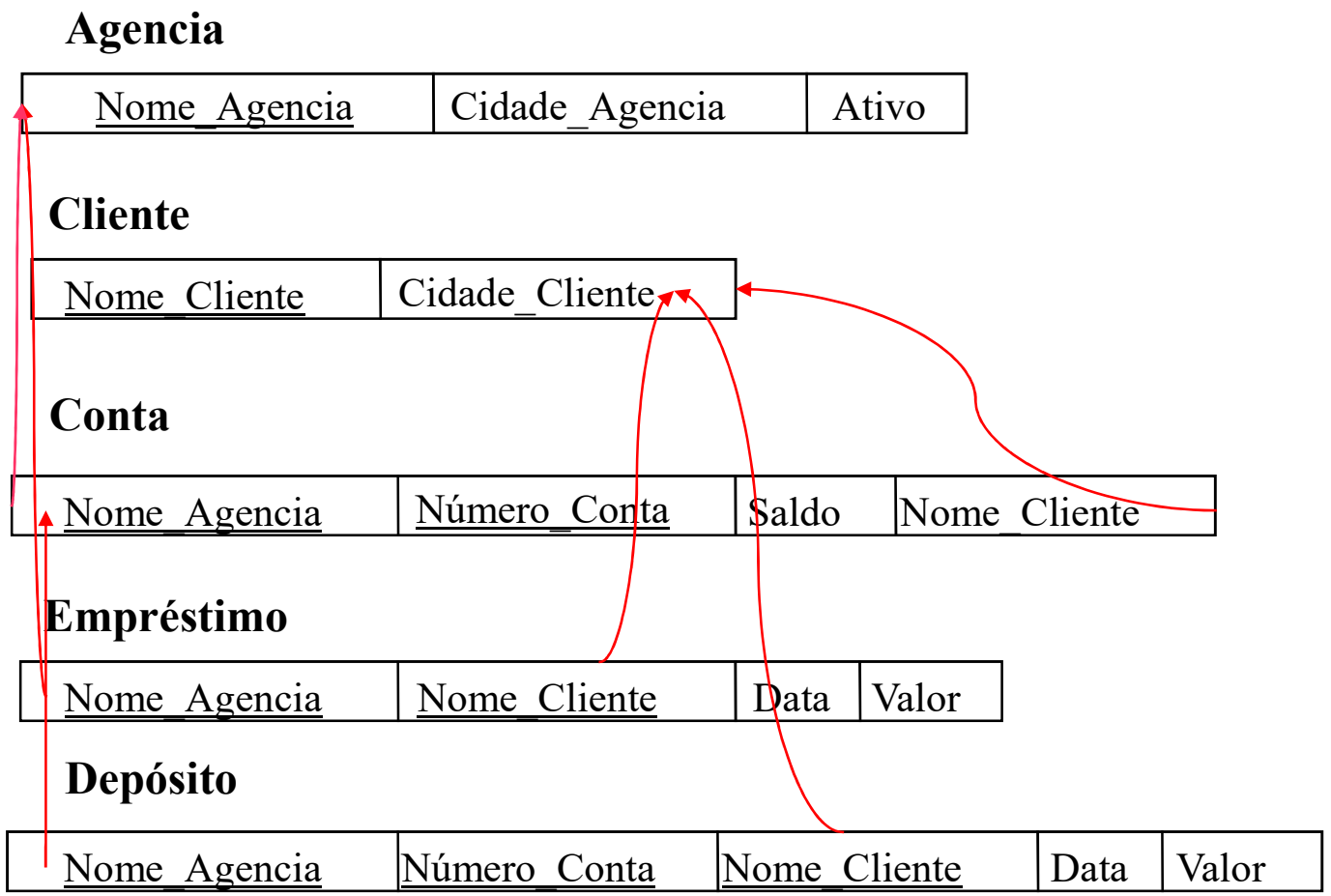
Where: Corresponde ao predicado de seleção da álgebra. É composto de um predicado que referencia atributos das relações que aparecem na cláusula From.

Select $A_1, A_2,\ldots,A_n$	→	Ai's	atributos
From $r_1, r_2,\ldots,r_m$	→	ri's	relações
Where P	→	P	predicado

$((r_1 \times r_2 \ldots \times r_m)$ WHERE p) $[A_1, A_2,\ldots A_n]$

Se a cláusula Where for omitida, o predicado P é verdadeiro

Esquema Exemplo



SQL (Structured Query Language)

"*" no lugar dos atributos, representa "todos"

Resultado → Relação

Exemplo:

"Encontrar os nomes de todas as agências na relação depósito"

Select Nome_Agência Deposito [nome-agência]
From Depósito

SQL permite duplicados nas relações.

Select distinct Nome-Agência
From Depósito

SQL (Structured Query Language)

union, intersect, e except

(***Select distinct*** Nome_Cliente
From Depósito
Where Nome_Agência = “Downtown”)

(Deposito WHERE Nome_agência=“DT”) [nome-cliente]

union, intersect, except

(***Select distinct*** Nome_Cliente
From Empréstimo
Where Nome_Agência = “Downtown”)

SQL não tinha um operador explícito de JUNÇÂO. ¿O que era feito?

SQL (Structured Query Language)

EXEMPLO

Achar todos os clientes (e a cidade deles) que têm um empréstimo em alguma agência.

```
Select  Cliente.Nome_Cliente, Cidade_Cliente
From    Empréstimo, Cliente
Where Empréstimo.Nome_Cliente =  Cliente.Nome_Cliente
```

Encontrar os nomes de todos os clientes (e sua cidade) que têm um empréstimo na agência 'Perryridge' .

```
Select  Cliente.Nome_Cliente, Cidade_Cliente
From    Empréstimo, Cliente
Where Empréstimo.Nome_Cliente =  Cliente.Nome_Cliente
and Nome_Agência = 'Perryridge'
```

SQL usa and, or e not

SQL (Structured Query Language)

Predicado e conectores +, -, *, /, and, or,

between

Simplificar cláusulas Where

```
Select   Numero_Conta
From     Conta
Where    saldo between 90000 and 100000
```

not between

Operadores para comparações de cadeias de caracteres

% igual a qualquer sub-cadeia

- igual a qualquer caractere

Perry % '---'

%idge%

SQL (Structured Query Language)

Exemplo

```
Select   Nome_Cliente                    '\'  'ab\%cd%'
From     cliente                              'ab cd%'
Where    Cidade_Cliente like '% Main%'
```

Encontrar todos os clientes que têm um empréstimo e fizeram um depósito na agência “Perryridge”

in

```
Select   Nome_Cliente
From     Empréstimo
Where    Nome_Agência = 'Perryridge'  and
         Nome_Cliente  in (Select   Nome_Cliente
                           From   Deposito
                           Where  Nome_Agência = 'Perryridge')
```

(a1, a2,...., an) **a1, a2, ..., an**

SQL (Structured Query Language)

not in

Encontrar todos os clientes que fizeram um depósito na agência 'Perryridge' mas que não têm um empréstimo nesta agência.

```
Select  Nome_Cliente
From   Depósito
Where Nome_Agência = 'Perryridge' and
                Nome_Cliente  not in
      (Select   Nome_Cliente
       From    Empréstimo
            Where Nome_Agência = 'Perryridge')
```

SQL (Structured Query Language)

Utilização de variáveis

Achar todos os clientes que fizeram um depósito em alguma agência na qual "Jones" fez um depósito.

```
Select   T.Nome_Cliente
From     Deposito as S, Deposito as T
Where    S.Nome_Cliente = 'Jones' and
                  S.Nome_Agência = T.Nome_Agência
```

Outra forma?

Achar todas as agências que têm um ativo maior que o de alguma agência localizada em Brooklyn.

```
Select   T.Nome_Agência
From     Agência as T, Agência as S
Where    T.Ativo  > S.Ativo and
                  S.Cidade_Agência = 'Brooklyn
```

Outra forma?

SQL (Structured Query Language)

Outra forma ?

> some ⟶ < >= =

Mudando algúm por todos

Select Nome_Agência
From *Agência*
Where Ativo > all (Select Ativo
From Agência
Where Cidade_Agência = 'Brooklyn')

Comparação de conjuntos para determinar se um conjunto está contido em outro

contains - not contains

SQL (Structured Query Language)

Exemplo: Achar todos os clientes que têm uma conta em todas as agências localizadas em Brooklyn

```
Select Nome_Cliente
From Conta as S
Where ( Select Nome_Agência
            From Conta as T
            Where S.Nome_Cliente = T.Nome_Cliente )
            contains
          ( Select Nome_Agência
             From Agência
             Where Cidade_Agência = 'Brooklyn' )
```

Esta operação foi omitida no padrão ANSI : Processamento muito custoso

SQL (Structured Query Language)

Testes para relações vazías

¿Cómo testar se uma subconsulta tem alguma tupla no seu resultado?

Exists

Achar todos os clientes que tem uma conta e um empréstimo na agência 'Perryridge'.

```
Select Nome_Cliente
From Cliente
where exists ( Select *
      From Conta
      Where Conta.Nome_Cliente = Cliente.Nome_Cliente
           and Nome_Agência = 'Perryridge' )
   and exists ( Select *
      From Emprestimo
      Where Emprestimo.Nome_Cliente = Cliente.Nome_Cliente
          and Nome_Agência = 'Perryridge' )
```

SQL (Structured Query Language)

not exists

- Achar a todos os clientes da agência Perryridge que têm uma conta alí mas nâo um emprestimo

- Considerando de novo a consulta:

- “Achar todos os clientes que têm uma conta em todas as agências localizadas em Brooklyn”

SQL (Structured Query Language

```
Select          S.Nome_Cliente
From  Cliente as S
Where not exists ( Select Nome_Agência
                     From    Agência
                     Where   Cidade_Agência = 'Brooklyn' )
                   except
                  ( Select   T.Nome_Agência
                     From    Conta as T
                     Where   S.Nome_Cliente =
                                        T.Nome_Cliente) )
```

SQL (Structured Query Language)

Ordenação da apresentação de tuplas
order by

"Listar em ordem alfabético todos os clientes que têm um empréstimo na agência Perryridge"

```
Select      distinct  Nome_Cliente
From        Empréstimo
Where       Nome_Agência = 'Perryridge'
Order by     Nome_Cliente
```

SQL (Structured Query Language)

Por definição SQL apresenta todos os elementos em ordem ascendente (cláusulas desc e asc).

"Achar os empréstimos em ordem descendente de valor e se vários empréstimos têm a mesma quantidad, os ordenamos em ordem ascendente pelo nome do cliente"

```
Select   *
From     Empréstimo
Order by valor desc, Nome_Cliente  asc
```

Funções de Agregação

oferece a capacide de calcular funções de grupos de tuplas usando a cláusula group by → FORMA GRUPOS

Tuplas com o mesmo valor num atributo são colocadas num grupo

- média avg
- mínimo mín
- máximo máx
- total sum
- contador count

} FUNCÕES DE AGREGAÇÃO

Exemplos

Qual é o saldo médio das contas em todas as agências?

```
Select      Nome_Agência, avg (Saldo)
From        Conta
Group by    Nome_Agência
```

Funções de Agregação

- Exemplo: (Elmasri y Navathe)

```
SELECT DNO, COUNT(*), AVG(SALARIO)
FROM EMPREGADO
GROUP BY DNO;
```

Veja Fig.

Resultado do GROUP BY

(a)

PNOME	MINICIAL	LNOME	SSN	. . .	SALARIO	SUPERSSN	DNO
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	. . .	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	null	1

Agrupamento das tuplas EMPREGADO por meio do valor de DNO

DNO	COUNT (*)	AVG (SALARIO)
5	4	33250
4	3	31000
1	1	55000

Resultado da Q24

Funções de Agregação

Cómo expressar condições que aplicam-se a grupos no lugar de tuplas?

Exemplo

"Encontrar as agências com saldo médio das contas maior de 1200 US$."

having aplicado depois da formação dos grupos

```
Select    Nome_Agência, avg(saldo)
From      Conta
Group by  Nome_Agência
      having avg (saldo)>1200
```

Funções de Agregação

- Outro exemplo (Elmasri e Navathe):

```
SELECT PNUMERO, PNOME, COUNT(*)
FROM PROJETO, TRABALHA_EM
WHERE PNUMERO=PNO
GROUP BY PNUMERO, PNOME
HAVING COUNT(*) > 2;
```

- Veja Fig.

Exemplo de Having

PNOME	PNUMERO		ESSN	PNO	HORAS
ProdutoX	1		123456789	1	32,5
ProdutoX	1		453453453	1	20,0
ProdutoY	2		123456789	2	7,5
ProdutoY	2		453453453	2	20,0
ProdutoY	2		333445555	2	10,0
ProdutoZ	3		666884444	3	40,0
ProdutoZ	3		333445555	3	10,0
Automacao	10	. . .	333445555	10	10,0
Automacao	10		999887777	10	10,0
Automacao	10		987987987	10	35,0
Reorganizacao	20		333445555	20	10,0
Reorganizacao	20		987654321	20	15,0
Reorganizacao	20		888665555	20	null
NovosBeneficios	30		987987987	30	5,0
NovosBeneficios	30		987654321	30	20,0
NovosBeneficios	30		999887777	30	30,0

Esses grupos não são selecionados por HAVING, condição da Q26

Depois da aplicação da cláusula WHERE, mas antes da aplicação da cláusula HAVING

PNOME	PNUMERO		ESSN	PNO	HORAS
ProdutoY	2		123456789	2	7,5
ProdutoY	2		453453453	2	20,0
ProdutoY	2		333445555	2	10,0
Automacao	10	. . .	333445555	10	10,0
Automacao	10		999887777	10	10,0
Automacao	10		987987987	10	35,0
Reoganizacao	20		333445555	20	10,0
Reoganizacao	20		987654321	20	15,0
Reoganizacao	20		888665555	20	null
NovosBeneficios	30		987987987	30	5,0
NovosBeneficios	30		987654321	30	20,0
NovosBeneficios	30		999887777	30	30,0

Depois da aplicação da condição da cláusula HAVING

PNOME	COUNT (*)
ProdutoY	3
Automacao	3
Reorganizacao	3
NovosBeneficios	3

Resultado da Q26
(PNUMERO não apresentado)

Operador Count

casos nos quais os duplicados devem ser eliminados antes de se calcular uma função de agregação.

Exemplo

"Achar o número de clientes com depósitos para cada agência"

Select Nome_Agência, count (distinct Nome_Cliente)

From Depósito

Group by Nome_Agência

"Encontrar o número de tuplas na relação Cliente"

Select count (*)

From Cliente

Operador Count

Se na mesma consulta aparecem uma cláusula *Where* e uma cláusula *having*, primeiro aplica-se o predicado da cláusula *Where*.

Exemplo

"Encontrar o valor médio de todos os clientes com depósitos que moram em Harrison e têm pelo menos 3 contas"

```
Select   avg(valor)
From     Depósito, Cliente
Where    Depósito.Nome_Cliente = Cliente. Nome_Cliente and
                 Cidade_Cliente = 'Harrison'
Group by Depósito. Nome_Cliente
having count(distinct  Nome_Agencia, Numero_Conta) >= 3
```

Modificando o Banco de Dados

Linguagens formais de consulta não incluem facilidades para mudar o BD.

SQL e linguagens comerciais permitem

ELIMINAÇÃO

Só tuplas completas podem ser eliminadas

delete r p → predicado qualquer

where p r → relação

As tuplas *t* em *r* tal que P(t) é verdadeiro, são eliminadas

delete Empréstimo

Elimina todas as tuplas

Modificando o Banco de Dados

Exemplos

Eliminar todas as contas de Smith

```
delete Conta
where Nome_Cliente = 'Smith'
```

Remover todas as contas das agências localizadas em Needham

```
delete Conta
where Nome_Agência in (Select Nome_Agência
                       from  Agência
                       where Cidade_Agência = 'Needham')
```

Modificando o Banco de Dados

Comando delete que contem um Select incluído que referencia a relação que vai ser atualizada → Anomalias potenciais.

```
Delete Depósito
where valor < ( select avg(valor)
                from Depósito )
```

SOLUÇÃO: Marcar as tuplas removidas. SQL padrão trata isto simplesmente não permitindo consultas deste tipo

Modificando o Banco de Dados

- INSERT – insere uma ou mais tuplas em uma tabela
  - Inserção de 1 tupla:

    INSERT INTO *tabela* [(*atrib1*,*atrib2*,...)]
    VALUES (*valor1*, *valor2*,...)
  - Inserção de múltiplas tuplas:

    INSERT INTO *tabela* [(*atrib1*,*atrib2*,...)]
    <*comando* **SELECT**>

- Exemplo 1 – Inserção de uma única tupla:
  - Inserir 3 as tuplas na relação PROJETO:


```
INSERT INTO PROJETO VALUES ('ProductX', '1', 'Bellaire', '5')
INSERT INTO PROJETO VALUES ('ProductY', '2', 'Sugarland', '5')
INSERT INTO PROJETO VALUES ('ProductZ', '3', 'Houston', '5')
```

  - O terceiro valor ‘5’ corresponde ao departamento 5. Logo, o departamento 5 deve existir na relação DEPARTAMENTO para que as inserções tenham sucesso; pois caso contrário, violaria a restrição de integridade referencial.

- Exemplo 2 – Inserção de múltiplas tuplas:
  - Popular uma tabela temporária DEPTS_INFO:

```
CREATE TABLE DEPTS_INFO (
                    DEPT_NAME       VARCHAR(10),
                    NO_OF_EMPS      INTEGER,
                    TOTAL_SAL       INTEGER
  );
INSERT INTO DEPTS_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
                    SELECT DNAME, COUNT (*), SUM (SALARY)
                    FROM DEPARTMENT, EMPLOYEE
                    WHERE DNUMBER=DNO
                    GROUP BY DNAME;
```

Comando de inserção de linhas nas tabelas em Oracle

```
create table newagain (
   first number(4) constraint pk_newa
   primary key,
   second date default sysdate,
   third varchar(20) not null );
```

Comando de inserção de linhas nas tabelas em Oracle

insert into newagain values (1111, ’10/10/2004’, ‘aaaa’);

- Agora vamos omitir o conteúdo da segunda coluna
- insert into newagain values (2222, , ‘bbbb’);

Modificando o Banco de Dados SQL Padrão

Linguagens formais de consulta não incluem facilidades para mudar o BD.

SQL e linguagens comerciais permitem

ELIMINAÇÃO

Só tuplas completas podem ser eliminadas

delete r p → predicado qualquer

where p r → relação

As tuplas *t* em *r* tal que P(t) é verdadeiro, são eliminadas

delete Empréstimo

→ Elimina todas as tuplas

Modificando o Banco de Dados

Exemplos

Eliminar todas as contas de Smith

```
delete Conta
where Nome_Cliente = 'Smith'
```

Remover todas as contas das agências localizadas em Needham

```
delete Conta
where Nome_Agência in (Select Nome_Agência
                       from  Agência
                       where Cidade_Agência = 'Needham')
```

Modificando o Banco de Dados

Comando delete que contem um Select incluído que referencia a relação que vai ser atualizada → Anomalias potenciais.

```
Delete Depósito
where valor < ( select avg(valor)
                from Depósito )
```

SOLUÇÃO: Marcar as tuplas removidas. SQL padrão trata isto simplesmente não permitindo consultas deste tipo

Remoção de registros em Oracle

```
DELETE FROM tablename
[WHERE condition];
```

Modificando o Banco de Dados

ACTUALIZAÇÕES

Mudar os valores de uma tupla sem que ela seja atualizada totalmente.

Ejemplo:

Pagamento de juros e todos os saldos devem ser aumentados em 5%

```
update Conta
set saldo = saldo * 1.05
```

Modificando o Banco de Dados

Suponha contas com saldos maiores que US$ 10000 recebem o 6% de juros, enquanto que as outras recebem o 5%.

```
update Conta
set saldo = saldo * 1.06
where saldo > 10000
```

```
update Conta
set saldo = saldo * 1.05
where saldo ≤ 10000
```

Comandos Case para Actualizações Condicionais

A mesma consulta de antes: Incremente todas as contas com saldos sobre $10,000 em 6%, todas as outras contas recebem 5%.

(En SQL 99)

```
update Conta
        set saldo = case
            when saldo <= 10000 then saldo *1.05
            else  saldo * 1.06
          end
```

Update em Oracle

```
UPDATE tablename
SET columnname = new_datavalue, …
[WHERE condition];
```

FIGURE 5-21 Syntax of the UPDATE command

FIGURE 5-23 UPDATE command to reassign regions

Especificando Restrições

Em SQL2, os usuários podem especificar restrições gerais. Exemplo:

Restrição de verificação sobre atributos:

Seja a tabela:

EstrelaCinema(Nome, Endereço, sexo, data_nasc)

Possível declaração de sexo:

Sexo CHAR(1) CHECK(sexo IN (´ F ´ , ´ M ´))

Restrição de verificação sobre tuplas:

```
CREATE TABLE EstrelaCinema (
    nome CHAR(30) UNIQUE,
    endereço VARCHAR(255),
    sexo CHAR(1),
    data_nasc DATE,
    CHECK (sexo = ´ F ´ OR nome NOT LIKE ´ Ms.% ´ )
```

Restrição sobre domínios:

```
CREATE DOMAIN D_NUM AS INTEGER
CHECK (D_NUM>0 AND D_NUM<21);
```

Outro exemplo de CHECK

```
CONSTRAINT AtivoRestr
CHECK ( Ativo >
        (SELECT Sum(Saldo)
         FROM Conta
        WHERE Conta.Nome_Agencia =
                    Nome_Agencia) )
```

Aquí a cláusula CHECK refere-se a outras tabelas. Estas restrições vigoram somente se a tabela associada é não vazia

Especificando Restrições

- **Asserções:**
- CREATE ASSERTION SALARY_CONSTRAINT
  CHECK (NOT EXISTS (SELECT *
  FROM EMPLOYEE E, EMPLOYEE M,
  DEPARTMENT D
  WHERE E.SALARY > M.SALARY AND
  E.DNO=D.DNUMBER AND
  D.MGRSSN=M.SSN));

Restrições em PostgreSQL

```
CREATE [ OR REPLACE ] RULE nome AS ON evento
    TO tabela [ WHERE condição ]
    DO [ ALSO | INSTEAD ] { NOTHING | comando | ( comando ; comando ... ) }
```

```
CREATE RULE "_RETURN" AS
    ON SELECT TO t1
    DO INSTEAD
        SELECT * FROM t2;

CREATE RULE "_RETURN" AS
    ON SELECT TO t2
    DO INSTEAD
        SELECT * FROM t1;

SELECT * FROM t1;
```

Definições aceitas pelo PostgreSQL, mas que geram erro.
Regras circulares

Restrições em PostgreSQL

```
-- Restriccao de integridade que nao permite a existencia de marcas que fabriquem pcs e portateis (ambos).
      -- NOTA: nao funciona em POSTGRESQL

    CREATE ASSERTION pcportatil CHECK ( NOT EXISTS

                                    (SELECT marca FROM produto WHERE tipo='pc'
                                            INTERSECT
                                            SELECT marca FROM produto WHERE tipo='portatil')
                                                    );

-- Alternativa (aceite em POSTGRESQL)

    CREATE RULE pcportatil_i AS
     ON INSERT TO produto WHERE NEW.tipo='pc' AND
     NEW.marca IN (SELECT marca
                                    FROM produto
                                    WHERE tipo='portatil')
     OR
     NEW.tipo='portatil' AND
     NEW.marca IN (SELECT marca
                                    FROM produto
                                    WHERE tipo='pc')
     DO INSTEAD NOTHING;
```

```
 CREATE RULE pcportatil_u AS
ON UPDATE TO produto WHERE NEW.tipo='pc' AND
NEW.marca IN (SELECT marca
                    FROM produto
                    WHERE tipo='portatil')
OR
NEW.tipo='portatil' AND
NEW.marca IN (SELECT marca
                    FROM produt

                    WHERE tipo='pc')
DO INSTEAD NOTHING;
```

Disparadores e Bancos de Dados Ativos

- Um disparador (“trigger”) é um procedimento que é invocado automaticamente pelo SGBD em resposta a mudanças especificadas do BD. UM BD que tem um conjunto associado de *triggers* é chamado um BD ativo.
- A descrição de um *trigger* contem três partes:
  - Evento: uma mudança no BD que ativa o *trigger*.
  - Condição: uma consulta ou teste que é executada quando o *trigger* é ativado.
  - Ação: um procedimento que é executado quando o *trigger* é ativado e sua condição é verdadeira.
- *Trigger* é como um `deamon’ que monitora um BD.

Disparadores

- Um evento pode ser um operação de insert, delete or update.
- Um condição pode ser uma sentença falsa ou verdadeira (ex: todos os salários dos empregados são menores que 100000) ou uma consulta (verdadeiro se vazia; falso em caso contrário.
- Ação faz referência aos valores antes e depois da atualização.

Exemplos -BD simplificado da Empresa

EMPREGADO

NOME	SSN	SALARIO	DNO	SUPERVISOR_SSN

DEPARTAMENTO

DNOME	DNO	TOTAL_SAL	GERENTE_SSN

Especificando regras ativas como gatilhos em notação Oracle.

(a) Gatilhos para manter automaticamente a consistência de TOTAL_SAL **de** DEPARTAMENTO.

(b) Gatilho para comparar o salário de um empregado com o de seu supervisor.

(a) **R1:**
```
CREATE TRIGGER TOTALSAL1
AFTER INSERT ON EMPREGADO
FOR EACH ROW
WHEN (NEW.DNO IS NOT NULL)
    UPDATE DEPARTMENTO
    SET TOTAL_SAL=TOTAL_SAL + NEW.SALARIO
    WHERE DNO=NEW.DNO;
```

NEW

R2:
```
CREATE TRIGGER TOTALSAL2
AFTER UPDATE OF EMPREGADO ON SALARIO
FOR EACH ROW
WHEN (NEW.DNO IS NOT NULL)
    UPDATE DEPARTAMENTO
    SET TOTAL_SAL=TOTAL_SAL + NEW.SALARIO – OLD.SALARIO
    WHERE DNO=NEW.DNO;
```

Para cada tupla modificada. Contrário a For each statement

R3:
```
CREATE TRIGGER TOTALSAL3
AFTER UPDATE OF DNO ON EMPREGADO
FOR EACH ROW
    BEGIN
    UPDATE DAPARTAMENTO
    SET TOTAL_SAL=TOTAL_SAL + NEW.SALARIO
    WHERE DNO=NEW.DNO;
    UPDATE DEPARTAMENTO
    SET TOTAL_SAL=TOTAL_SAL– OLD.SALARIO
    WHERE DNO=OLD.DNO;
    END;
```

R4:
```
CREATE TRIGGER TOTALSAL4
AFTER DELETE ON EMPREGADO
FOR EACH ROW
WHEN (OLD.DNO IS NOT NULL)
    UPDATE DEPARTAMENTO
    SET TOTAL_SAL=TOTAL_SAL – OLD.SALARIO
    WHERE DNO=OLD.DNO;
```

(b) **R5:**
```
CREATE TRIGGER INFORM_SUPERVISOR1
BEFORE INSERT OR UPDATE OF SALARIO, SUPERVISOR_SSN ON EMPREGADO
FOR EACH ROW
WHEN
(NEW.SALARY> (SELECT SALARIO FROM EMPREGADO
                WHERE SSN=NEW.SUPERVISOR_SSN))
        INFORM_SUPERVISOR(NEW.SUPERVISOR_SSN, NEW.SSN);
```

Sintaxe de Oracle

<trigger> ::= **CREATE TRIGGER** <nome gatilho>
(**AFTER** | **BEFORE**) <eventos ativadores> **ON** <nome da tabela>
[**FOR EACH ROW**]
[**WHEN** <condicao>]
<acoes disparadas> ;

<eventos ativadores>::=<evento ativador> {**OR** <evento ativador>}

<acao disparada>::=**INSERT** | **DELETE** | **UPDATE** [**OF** <nome coluna> {, <nome coluna>}]

<acao disparada> ::=<PL/SQL block>

Sintaxe de PostgreSQL

```
CREATE TRIGGER nome { BEFORE | AFTER } { evento [ OR ... ] }
    ON tabela [ FOR [ EACH ] { ROW | STATEMENT } ]
    EXECUTE PROCEDURE nome_da_função ( argumentos )
```

```
CREATE TABLE emp (
    nome_emp        text,
    salario         integer,
    ultima_data     timestamp,
    ultimo_usuario text
);
```

Sintaxe de PostgreSQL

```
CREATE FUNCTION emp_gatilho() RETURNS trigger AS $emp_gatilho$
    BEGIN
        -- Verificar se foi fornecido o nome e o salário do empregado
        IF NEW.nome_emp IS NULL THEN
            RAISE EXCEPTION 'O nome do empregado não pode ser nulo';
        END IF;
        IF NEW.salario IS NULL THEN
            RAISE EXCEPTION '% não pode ter um salário nulo', NEW.nome_emp;
        END IF;

        -- Quem paga para trabalhar?
        IF NEW.salario < 0 THEN
            RAISE EXCEPTION '% não pode ter um salário negativo', NEW.nome_emp;
        END IF;

        -- Registrar quem alterou a folha de pagamento e quando
        NEW.ultima_data := 'now';
        NEW.ultimo_usuario := current_user;
        RETURN NEW;
    END;
$emp_gatilho$ LANGUAGE plpgsql;

CREATE TRIGGER emp_gatilho BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_gatilho();
```