SQL BATCH

1. Introdução

Neste roteiro, vamos tratar do conceito de “SQL Batch”. Traduzindo para o nosso idioma, poderíamos dizer algo como “SQL em Lote”, “Lote de comandos SQL”, “Bloco de comandos SQL” ou, ainda, “Script SQL”. Trata-se de uma sequência de comandos SQL que é executado sob uma única submissão. Em muitos casos, dois comandos SQL podem ser executados em duas submissões independentes ou, de forma equivalente, numa única submissão como, por exemplo, os dois comandos abaixo.

```
select * from tabela1
select * from tabela2
```

Podemos enviar os comandos para execução pelo sistema de banco de dados de uma só vez ou separadamente. Neste exemplo simples, não existe diferença aparente, exceto pelo modo como os resultados são apresentados. Submetendo os dois de uma única vez, não haverá iteração adicional com o sistema, e os resultados de ambos serão mostrados de uma única vez. De outra forma, os resultados do primeiro comando serão apresentados; em seguida, enviaremos o segundo comando; e só então, receberemos os resultados do segundo comando. Se você quiser salvar os resultados em arquivos, é possível que, na primeira forma, você tenha de fazê-lo de uma única vez enquanto, de outra forma, tenha de salvá-los em duas. No entanto, se analisarmos além da aparência, existe uma diferença importante, pois com um único “batch” a análise léxica, sintática e semântica é aplicada a toda sequência. Desta forma, se houver um erro num deles, o “batch” inteiro não poderá ser executado. Ao contrário, em batches separados, a análise também é separada.

É importante ressaltar que os dois comandos do exemplo podem ser enviados de uma só vez, mas separados por um símbolo delimitador de “batches”. Neste caso, embora enviados ao sistema numa única vez, não são interpretados como um único “SQL Batch”. Para que tal situação ocorra, é necessário adequar-se às características operacionais de cada produto.

Podemos também entender “SQL batch” como uma sequência de um ou vários comandos e, desta forma, tudo é um “SQL batch”. Levando isto em conta, vamos nos concentrar nesses objetos com dois ou mais comandos porque as de um único comando são um caso particular.

Importante observar que, neste momento, não estamos falando de transação que é outro conceito que trataremos futuramente.

2. Características:

O conceito de “SQL Batch” não é apenas uma modalidade de submissão diferente, mas permite a execução de comandos interligados. Esta dependência entre eles torna inviável a execução separada de apenas parte do bloco. Este assunto antecede ao estudo de “stored procedures”, “functions” e “triggers” porque a estrutura destes objetos tem certa semelhança à do “SQL Batch”, pelo menos, quanto à organização do código, embora existam diferenças importantes.

Vejamos algumas características de “SQL Batch”:

- Sequência de comandos SQL;
- Os comandos da sequência podem ser independentes ou existir uma dependência entre eles;
- Quando houver dependência entre os comandos, eles são agrupados numa estrutura lógica e o objetivo somente pode ser obtido coletivamente. Neste caso, os comandos da sequência, considerados individualmente, não têm significado relevante ou mesmo não podem ser executados, em razão da dependência entre eles;
- Normalmente, neste nível de trabalho com a linguagem SQL, comandos padronizados e proprietários são intercalados;
- São utilizadas construções semelhantes às de outras linguagens de programação;
- Os recursos da linguagem tornam-se menos declarativas e mais imperativas em relação aos comandos SQL já vistos.

3. Tipos de comandos

De forma análoga aos programas em outras linguagens de programação, os "SQL Batch" contemplam declarações e comandos para diferentes fins. Por exemplo:

- Declarações de variáveis;
- Atribuição;
- Controle de fluxo;
- Repetição;
- Apresentação de resultados;
- Manipulação de dados (DML);
- Definição de dados (DDL);
- Tratamento individual das tuplas resultantes de um comando SQL.

4. Exemplo de SQL batch no Oracle®™

Todos os produtos incorporam o conceito de "SQL Batch" e na sua construção contemplam comandos que seguem padrões SQL como também construções proprietárias. A seguir, um exemplo de "SQL Batch" em "PL/SQL" da Oracle:

```
set serveroutput on size 1000000
declare
   v_cmd varchar2(255); -- variável auxiliar para imprimir a linha
                        -- um limite(item) do profile por linha
   v_first boolean;
begin
for pf in (select distinct profile
             from dba_profiles
            where profile <> 'DEFAULT' order by profile)
  loop
    v_first := true;
    for r in (select resource_name,limit from dba_profiles
                where profile = pf.profile
                  and limit <> 'DEFAULT' order by resource_name)
       loop -- imprime o primeiro recurso com o comando create
```

```
         if (v_first) then
             v_cmd:='create profile '|| pf.profile || ' limit ' ||
                 r.resource_name || ' ' || r.limit;
             v_first := false;
         else -- A partir do segundo recurso, sem o comando create
             v_cmd:= r.resource_name || ' ' || r.limit;
         end if;
         dbms_output.put_line(v_cmd); -- cada recurso por linha
       end loop;
    dbms_output.put_line(';');
  end loop;
end;
/
```

Este “SQL Batch” produz comandos para recriar os “profiles” existentes no banco de dados, exceto o “DEFAULT”, que é automaticamente criado. O resultado tem o seguinte aspecto e foi editado com cores alternadas para facilitar a compreensão.

```
create profile MONITORING_PROFILE limit FAILED_LOGIN_ATTEMPTS UNLIMITED
;
create profile PROF1 limit FAILED_LOGIN_ATTEMPTS 20
SESSIONS_PER_USER 5
;
create profile PROF2 limit FAILED_LOGIN_ATTEMPTS 10
IDLE_TIME UNLIMITED
SESSIONS_PER_USER 3
;
create profile PROFILE2 limit CONNECT_TIME 600
FAILED_LOGIN_ATTEMPTS 5
IDLE_TIME 60
LOGICAL_READS_PER_SESSION 10000
PASSWORD_LIFE_TIME 90
PASSWORD_REUSE_MAX UNLIMITED
PASSWORD_REUSE_TIME 180
SESSIONS_PER_USER 50
;

PL/SQL procedure successfully completed.
```

5. “SQL Batch” no Laboratório

Apresentamos abaixo alguns comandos disponíveis na linguagem SQL para a produção de “SQL Batch” no ambiente que utilizamos.

5.1. Declaração de variáveis

Nos “SQL Batch”, podemos declarar variáveis. Com exceção de algumas variações sintáticas entre os produtos, a forma de declaração é a seguinte:

```
declare @var1 int,
        @var2 char(10),
        @var3 float,
```

```
        .
        .
        .
```

5.2. Controle de fluxo:

```
if (condition)
   begin
   .
   .
   .
   end
else
   begin
   .
   .
   .
   end
```

A condição utilizada pelo comando pode ser expressa de várias formas, incluindo as tradicionalmente conhecidas envolvendo variáveis, como verificações sobre os dados existentes no sistema. Exemplos:

```
if (@var1 + sin(@var2) > 0.1267) ....
```

```
if (exists(select 1
             from tb_cidades
            where codigo like 'A00001%'
              and habitantes > 1000000)) ...
```

```
if ((select estado
       from tb_cidades
      where codigo = 'A00001') in ('SP','RJ','RS') ...
```

```
if ((select habitantes
       from tb_cidades
      where codigo = @cidade
        and estado = 'PR') = (select max(habitantes)
                                from tb_cidades
                               where estado = 'PR')) ...
```

5.3. Repetição:

```
while (condition)  /*** As mesmas condições do comando IF ***/
   begin
```

```
.
.
.
end
```

5.4. Controle de fluxo numa estrutura de repetição:

As alternativas abaixo, muitas vezes, são desqualificadas por profissionais e educadores. Os exemplos ficam registrados para conhecimento das possibilidades.

a) Break

Este comando interrompe o fluxo de execução e avança para fora da estrutura do "loop" na qual está inserido.

```
while (condition)
  begin
    .
    .
    if (condition1)
       break
    .
    .
  end
```

b) Continue

Este comando reinicia o loop sem executar os comandos seguintes dentro do loop.

```
while (condition)
  begin
    .
    .
    if (condition1)
        continue
    .
    .
  end
```

5.5 Atribuição de valores às variáveis:

Valores podem ser atribuídos às variáveis na forma apresentada abaixo:

```
set @var1 = 10,
    @var2 = 'abcd adb',
    @var3 = 123.87
```

5.6 Atribuição às variáveis de valores resultantes de comandos SQL:

A forma mais interessante de atribuição de valores às variáveis é a partir de dados armazenados no banco de dados, como mostram os exemplos a seguir:

```
select @var1 = habitantes,
       @var2 = nome,
       @var3 = habitantes / 1000.0
  from tb_cidades
 where codigo = 'A99999'
```

5.7. Concatenação de strings:

A operação de concatenação de cadeias é muito utilizada. Em alguns sistemas gerenciadores de bancos de dados, o operador de concatenação é o "||" (duplo símbolo pipe). Outros sistemas utilizam o símbolo “+”.

Exemplos:

```
select 'abc' || '123' from dual; -- Oracle
set @variavel = @variavel_1 || 'xyz'
select @variavel = nome || sobre_nome
  from tb_cidades where codigo = 'A999999'
```

No laboratório, você pode, indistintamente, utilizar o símbolo "||" ou "+". Exemplos:

```
select 'abc' + '123'

set @variavel = @variavel_1 || 'xyz'

select @variavel = nome + sobre_nome
  from tb_cidades where codigo = 'A999999'

select 'abc' + 'cde' || 'fgc' from dual;
```

5.8. Verificação da quantidade de registros selecionados ou afetados

Existem formas de verificação da existência de registros que satisfazem a certa condição, ou mesmo se um comando SQL selecionou ou afetou registros. O modo como esta verificação é realizada é dependente de produto específico.

No laboratório, o SGBD disponibiliza uma variável global que imediatamente após o comando pode ser consultada para saber quantas linhas foram selecionadas, inseridas, alteradas ou removidas. A variável tem o nome abaixo:

```
@@rowcount
```

Por exemplo:

```
declare @tuplas int

select habitante
  from tb_cidades
 where nome = 'São Pedro do Sul'

set @tuplas = @@rowcount

if (@tuplas = 0) -- Poderia utilizar a própria variável @@rowcount,
                 -- mas existe razão justificada para utilizar
                 -- uma variável auxiliar em determinados casos.
   begin
    .
    .
    -- não existe cidade com este nome ...
    .
    .
   end
else if (@tuplas = 1)
   begin
    .
    .
    -- existe uma cidade com este nome ...
    .
    .
   end
else
   begin
    .
    .
    -- mais que uma cidade com este nome ...
    .
    .
   end
```

5.9. SQL opera sobre conjuntos de registros/linhas/tuplas

Os comandos SQL podem afetar um conjunto vazio, unitário ou com vários elementos e os resultados podem não ser os esperados se você não levar em conta a multiplicidade dos registros afetados. Não podemos presumir que os comandos afetam somente um registro.

Por exemplo, no código a seguir:

```
declare @codigo char(20)
select @codigo = codigo from tb_cidades
```

se existem várias linhas na tabela, qual o valor da variável? Portanto, para que exista previsibilidade, é necessária uma condição de seleção que garanta a unicidade.

6. Descrição do problema geral

Para estudarmos o conceito de "SQL Batch" e exercitarmos esta construção, vamos escolher um assunto. Existem situações que descrevemos através de auto relacionamentos. Este assunto foi tratado nas aulas de teoria do semestre anterior. Do ponto de vista da representação relacional, este fato conduz à percepção de certa recursividade entre os dados das tabelas. Por exemplo, considere o seguinte esquema:

```
funcionários=(cod_func, nome,...,
              cod_func_gerente,
              (cod_func_gerente) references funcionários
             )
```

O atributo assinalado em vermelho é uma referência à própria tabela. Esta construção permite que o funcionário "A" tenha como gerente o funcionário "B" que por sua vez tem "C" com seu gerente e assim, sucessivamente, até que no topo da hierarquia não existam mais superiores. Do lado contrário, permite que a partir de um gerente possa se obter todos os seus subordinados e também os subordinados destes últimos até que não existam mais subordinados.

Este conceito é conhecido em diversas áreas como "fechamento transitivo" (transitive closure) que, no caso do exemplo, consiste no conjunto de todos os funcionários que estão na cadeia completa da hierarquia que começa a partir de um funcionário. Genericamente, isto pode ser aplicado a diversas situações. Por exemplo, num grafo orientado "G", dado um vértice "v", podemos estar interessados no conjunto de todos os vértices que podem ser alcançados a partir de "v", incluindo grafos cíclicos e, nesse caso, o processo deve ser capaz de detectar a circularidade e apresentar o resultado correto.

Exemplos de problemas com característica recursiva na definição dos dados:

- Peças compostas de peças;
- Pré-requisitos entre disciplinas;
- Estrutura organizacional de empresas que contempla órgãos que contem outros órgãos;
- Aeroportos interligados por voos aéreos;
- Pessoas conectadas as pessoas.

7. Exemplo de auto relacionamento

Um caso comum e bem comportado de hierarquia do tipo apresentado acima refere-se à organização de uma empresa numa hierarquia de órgãos que pode ser descrita da seguinte forma:

i) A empresa é organizada em áreas, diretorias, divisões, superintendências, departamentos, setores e secretarias, etc.
ii) Existem vários órgãos de cada tipo formando uma estrutura hierárquica.

A linguagem SQL contempla construções que facilitam o manuseio dessas estruturas, por exemplo:

i) with recursive . . .
ii) create recursive view . . .

Alguns produtos fornecem comandos específicos para o tratamento dessa questão (cláusula “connect by”), mas precisam ser validados nestas situações de circularidade e conjuntos de relacionamentos “n:n” que aparecem em cenários genéricos.

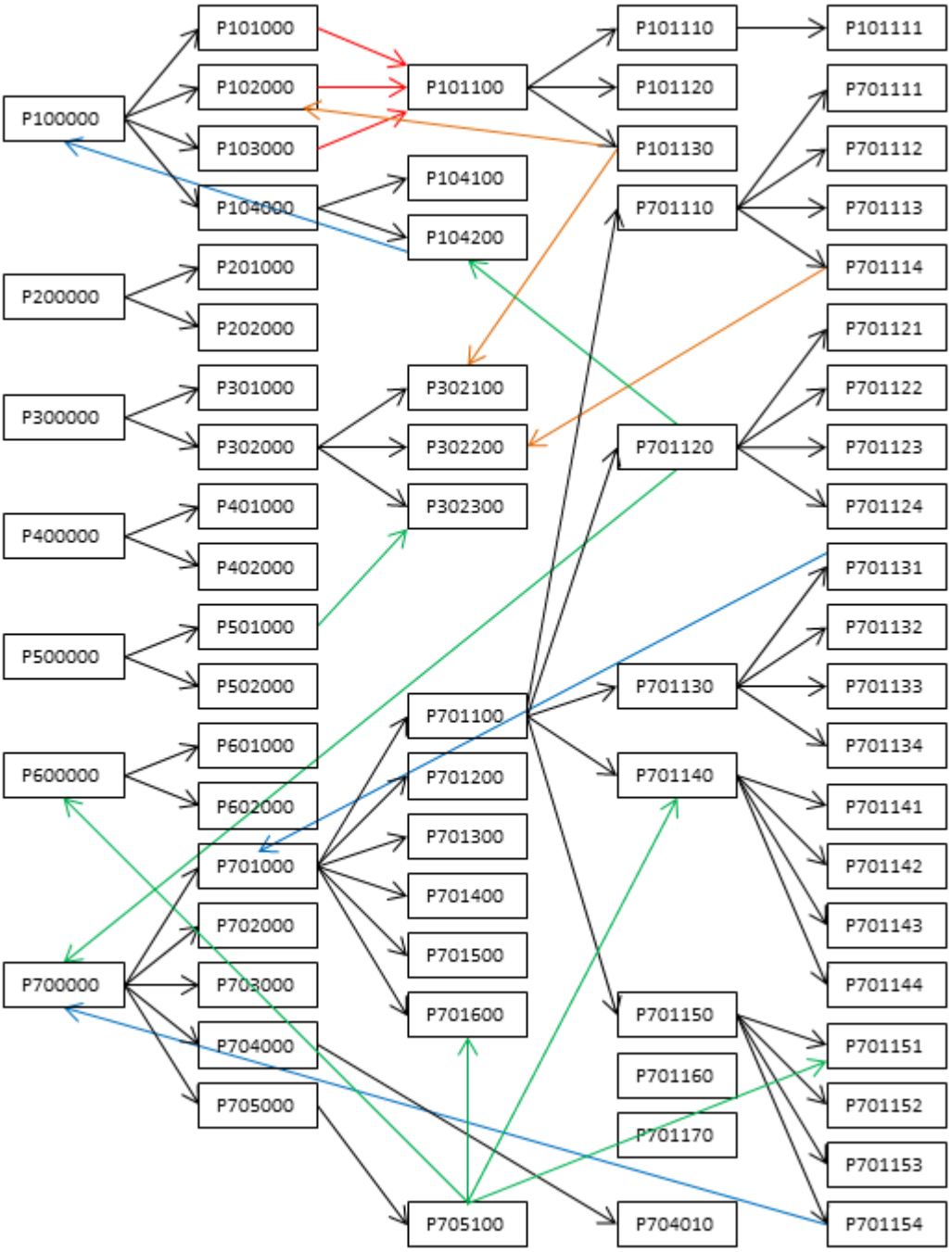
Tomando como exemplo a representação da hierarquia organizacional de uma empresa ou a cadeia de gerência de funcionários na forma citada anteriormente, podemos observar que, colocada na notação de um grafo, ele possui uma forma bastante simples.

Vamos apresentar um problema que, em situações mais genéricas e na notação de grafo, acrescenta duas possibilidades:

i) Um vértice pode ser destino de duas ou mais arestas;
ii) Formação de um ciclo direto ou indireto.

Examinemos a figura 1. A possibilidade de item (i) está contemplada no exemplo e assinalada com arestas vermelhas. O item (ii) aparece em outras situações como a interligação de aeroportos de origem e destino por rotas aéreas, e em diversos temas em grafos. Para ilustrar esta situação frequente em vários cenários, foram incluídas arestas na cor azul na figura. Essas ligações inserem ciclos na representação o que pode trazer uma complicação para algumas estratégias automáticas, mas concedem um caráter genérico à solução. É claro que não é uma única aresta a responsável pelo ciclo; na figura 1 a aresta assinalada em azul poderia ser trocada por outra. Neste exemplo, vamos considerar como aresta responsável pelo ciclo a última aresta adicionada na formação do ciclo, ou seja, antes dela, o ciclo não existia. Estritamente falando, em termos de grafos com ciclos minimais, qualquer aresta é responsável pela circularidade porque removendo-a elimina-se o respectivo ciclo. Esta suposição é para facilitar a visualização de alguns aspectos na figura quando verificaremos a correção da solução do problema que analisaremos.

A figura 1 apresenta as ligações de amizade entre pessoas através de arestas orientadas. A origem da aresta indica a pessoa que requisitou o relacionamento e o destino da aresta representa a pessoa a quem ficou a incumbência de aceitar ou rejeitar a amizade. Além das arestas nas cores vermelha e azul citadas acima, as de cor verde indicam as pendentes de resposta; as de cor marrom indicam as rejeitadas.



Legenda das cores das arestas:

- Solicitação aceita.
- Solicitação aceita – Apenas ressalta a adição de um ciclo no grafo.
- Solicitação pendente de resposta.
- Solicitação recusada.
- Solicitação aceita – Exemplo de acréscimo de conjunto de relacionamentos n:n.

Figura 1 – Representação de relacionamentos entre pessoas

Para estabelecer o vocabulário, vamos convencionar que uma pessoa é amiga “direta” de outra se existe uma aresta entre ambas, independentemente da orientação da aresta. Vamos chamar de amigos indiretos ou transitivamente indiretos àqueles que são ligados por caminhos de mais que uma aresta independentemente da orientação das mesmas. Vamos chamar de requisitante direto a pessoa que está na extremidade origem da aresta. Vamos chamar de requisitantes indiretos de “v” as pessoas que podem atingir o vértice “v” num caminho maior que uma aresta seguindo a orientação delas. Nomenclatura semelhante se aplica aos respondentes diretos e indiretos. Vamos olhar a figura 2 para entendermos melhor estes termos.

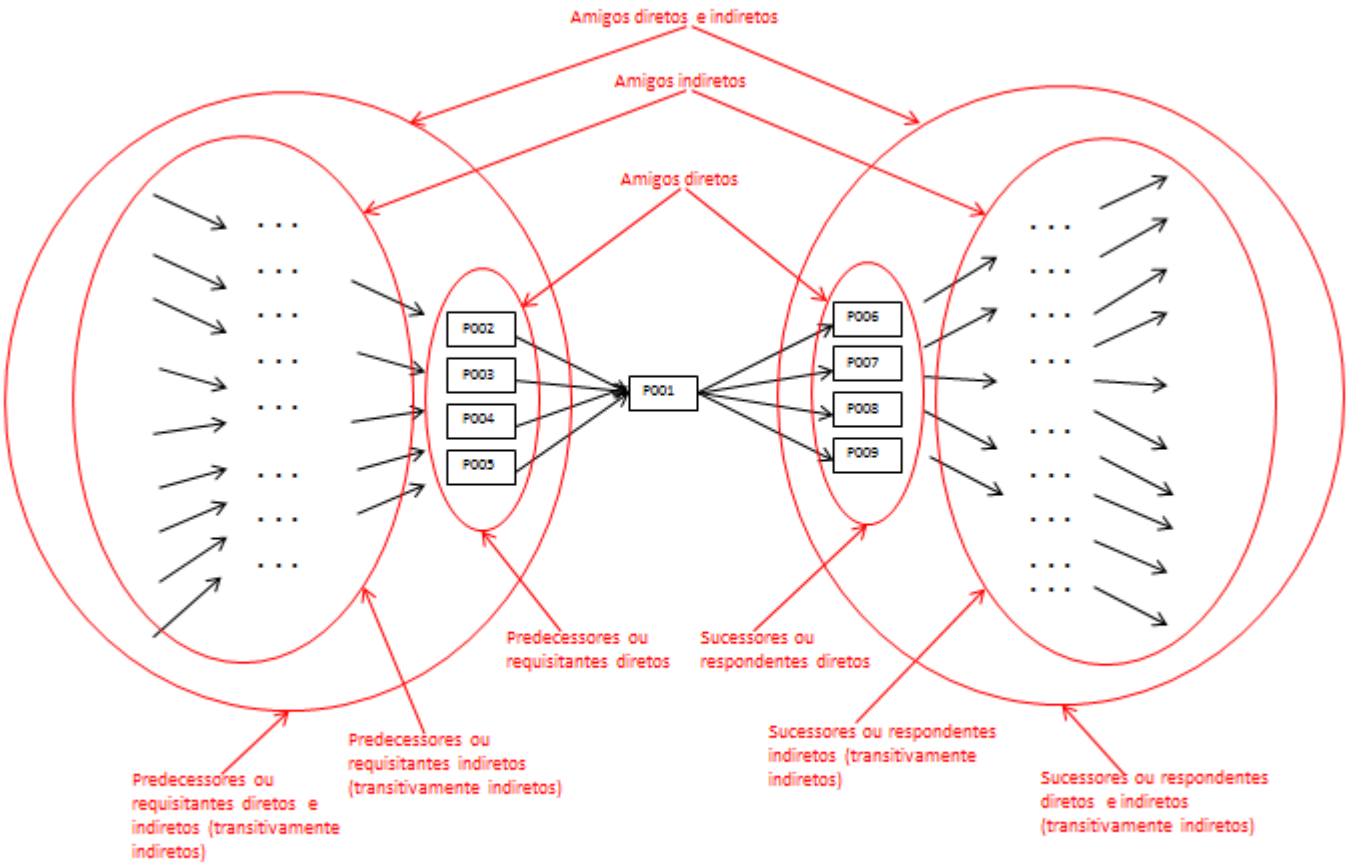


Figura 2 – Nomenclatura que usaremos tendo como referência a pessoa representada pelo vértice central "P001".

8. Representação dos dados no nível Conceitual segundo o MER

Vamos considerar o universo que envolve as pessoas e os relacionamentos de amizade entre elas. Eles são registrados no banco de dados por solicitação de uma pessoa dirigida a outra. Este pode aceitar ou recusar o convite. Muitas informações estão presentes neste universo, incluindo dados pessoais, fotos, preferências, etc. No entanto, vamos nos concentrar, sem perda de generalidade, no conjunto de auto relacionamentos e três atributos para as pessoas. Uma pessoa pode requisitar amizade à mesma pessoa diversas vezes, de modo que a “data do convite” é um atributo discriminador do conjunto de relacionamentos. Isto é razoável nas situações de convites recusados para os quais o requisitante pode reformular novamente a solicitação.

A representação dos dados no nível conceitual é apresentada a seguir, na figura 3, através do diagrama de entidades e relacionamentos. Como já estudamos, tal diagrama é o resultado da aplicação do Modelo de Entidades e Relacionamentos ao universo proposto.

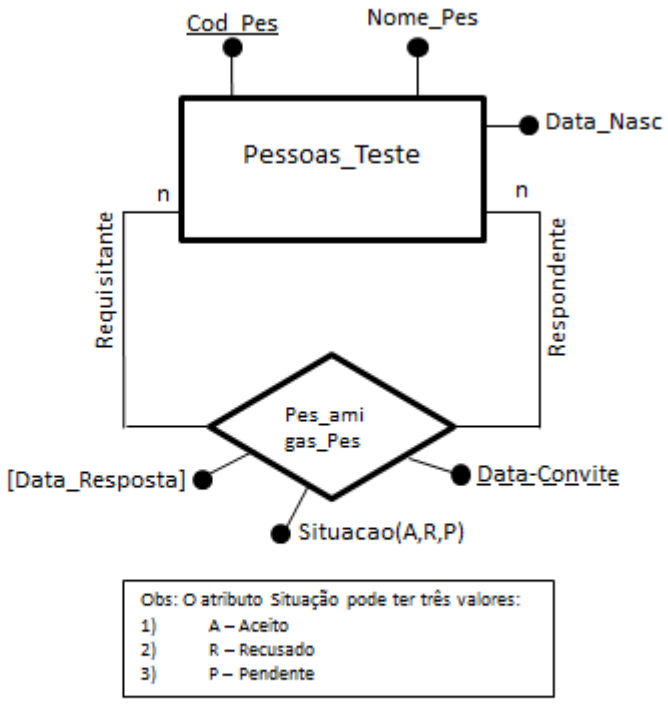
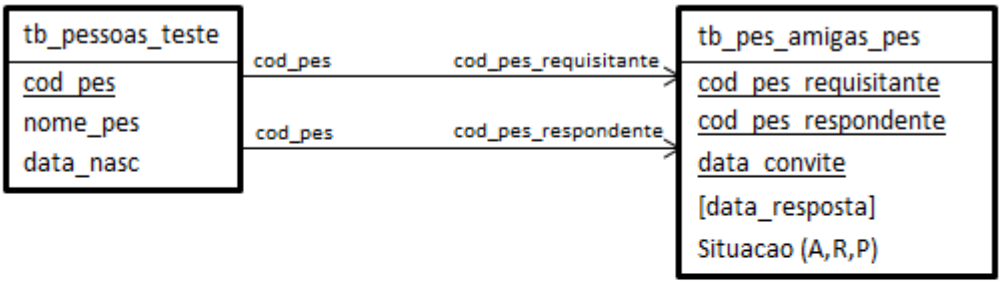


Figura 3 – Representação no nível Conceitual através do Diagrama de Entidades e Relacionamentos que é o resultado da aplicação da MER.

9. Representação Relacional (Diagrama Relacional)

O diagrama acima que representa o universo que estamos descrevendo baseia-se no modelo de entidades e relacionamentos que é uma representação no nível conceitual. As representações no nível lógico são obtidas a partir do conceitual e possui características diferentes de acordo com o modelo de dados adotado. Na aula de teoria, obtivemos as representações hierárquica, orientada a grafos e relacional. Esta última pode ser expressa de duas formas: em termos de diagrama que chamamos de "diagrama relacional" e no formato textual denominado esquema relacional. A seguir, na figura 4, está a representação no nível lógico na modalidade relacional na forma de um diagrama relacional e, mais adiante, no formato textual.



Observação: As setas direcionais do diagrama relacional não indicam cardinalidades nem referências . Na notação aqui utilizada, elas indicam o sentido da movimentação da chave primária ao tornar-se uma chave estrangeira.

Figura 4 – Representação no nível Lógico: Diagrama Relacional derivado do Diagrama de Entidades e Relacionamentos.

Como explicado acima, a representação dos dados no nível lógico na modalidade relacional pode ser apresentada em notação textual. A descrição da estrutura de uma relação (tabela) é chamada de esquema relacional. Na figura abaixo, é apresentado um conjunto de esquemas relacionais (no caso, dois esquemas) que é chamado de esquema de banco de dados. Como observação, o texto não são comandos SQL.

```
tb_pessoas_teste  = (cod_pes, nome_pes, data_nasc)

tb_pes_amigas_pes = (cod_pes_requisitante, cod_pes_respondente, data_convite,
                     [data_resposta], situação (A,P,R),
                     (cod_pes_requisitante) references tb_pessoas_teste,
                     (cod_pes_respondente) references tb_pessoas_teste
                    )
```

10. Tabelas do Banco de Dados

Podemos ver no diagrama relacional que é uma representação gráfica do esquema relacional para este banco de dados, que existem duas tabelas que armazenam as informações. A que contem dados das pessoas tem o seguinte nome:

```
tb_pessoas_teste.
```

Neste texto, os dados das pessoas estão reduzidos ao código, nome e data de nascimento. O nome da tabela tem o sufixo “teste” para não ser confundida com outra que armazena informações diferentes e foi ou será utilizada em outras aulas.

Os relacionamentos entre as pessoas ficam registrados na tabela de nome:

```
tb_pes_amigas_pes.
```

No banco de dados ficam assinalados os diferentes papéis das pessoas no convite, ou seja, o de requisitante e o de respondente, mas consideramos que não existe diferença entre eles na amizade.

11. Detalhamento da solução do problema

Falamos anteriormente que vamos tratar dos dados referentes às pessoas e suas amizades. Mas o que exatamente desejamos fazer? Dada uma pessoa, pretendemos obter todas as pessoas que podem ser alcançadas seguindo as arestas que as interligam, sem levar em conta a orientação das setas, ou seja, queremos o conjunto dos amigos diretos e indiretos. Faremos isto para ilustrar a utilização do "SQL Batch". O resultado deve mostrar a lista desses amigos junto com a distância que os separa em termos de número de arestas e as pessoas que respectivamente as antecedem. A pessoa inicial deve aparecer no resultado com a distância zero. Se uma mesma pessoa é alcançável a diferentes distâncias, ela deve aparecer somente na distância mais próxima.

O algoritmo, inicialmente, inclui o vértice do grafo que representa a pessoa inicial no conjunto desejado. Depois, acrescenta, a cada passo, um subconjunto dos vértices adjacentes daqueles que foram incluídos no passo imediatamente anterior. Este processo se repete com incrementos dos novos vértices até que não exista mais nenhum que possa ser incluído no conjunto. Para efeito da questão proposta, os vértices são vizinhos independentemente de ser origem ou destino da aresta. Na figura 5, está uma visão geral do algoritmo.

O fato de não levarmos em conta a origem das arestas faz com que o resultado estabeleça um grafo conexo maximal e, deste modo, qualquer pessoa deste conjunto terá o mesmo conjunto de amigos diretos e indiretos. A diferença estará nas distâncias relativas das pessoas do conjunto. Esta abordagem, neste momento, serve para apresentação do algoritmo, mas, veremos que a consideração da orientação das arestas conduz a um caso particular deste que estamos tratando.

12. Tabelas utilizadas na solução do problema:

Os dados referentes ao universo em estudo estão armazenados nas tabelas acima mencionadas. No entanto, para solucionarmos o problema pelo algoritmo que proporemos, serão necessárias estruturas de dados adicionais que são representadas pelas tabelas enumeradas abaixo. Poderíamos utilizar tabelas que os sistemas gerenciadores denominam "temporárias", mas como existem diferenças entre os produtos nos comandos de criação, na nomenclatura, comportamento e outras propriedades, vamos utilizar estruturas permanentes para armazenamento de dados temporários.

i) tb_pessoas_ligadas. Começa com a pessoa inicial e incorpora a cada passo outras pessoas que podem ser alcançadas direta ou indiretamente. Ao final do processo, conterá as pessoas amigas diretas e transitivamente indiretas tanto na condição de predecessores ou sucessores. Na figura 5, o conteúdo desta tabela corresponde às pessoas que estão nos vértices dentro da elipse maior à esquerda. Esta tabela possui três colunas: a pessoa alcançada, a pessoa anterior no percurso e a distância do indivíduo inicial.

ii) tb_fronteira. Subconjunto das pessoas do conjunto "tb_pessoas_ligada" do item (i) que foram acrescentadas no passo imediatamente anterior. Os vértices constituem a periferia, ou seja, estão na fronteira ou na borda, no sentido de que foram mais recentemente incorporados. Na figura 5, correspondem às pessoas que estão nos vértices internos à elipse maior à esquerda e

fora da menor. A área da figura correspondente tem a cor (mais ou menos) azulada. A estrutura da tabela é a da "tb_pessoas_ligadas" sem o valor da distância.

iii) tb_adjacentes_temp. Novas pessoas que serão incluídas e passarão a constituir a nova fronteira. Estas pessoas estão a uma aresta dos vértices da fronteira atual. Na figura 5, corresponde às pessoas que estão nos vértices dentro da elipse do lado direito da figura. Possui a mesma estrutura da "tb_fronteira".

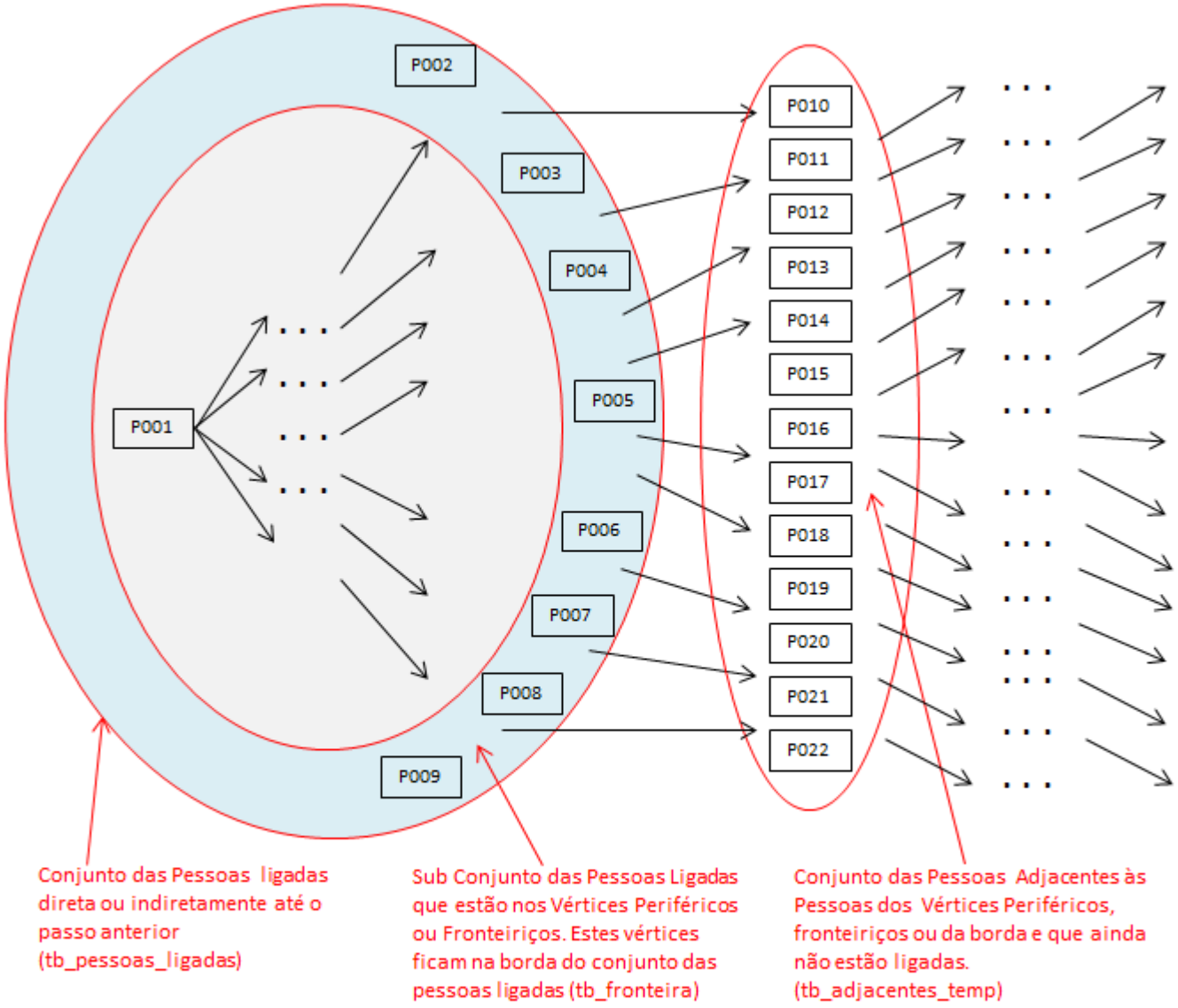


Figura 5 – Um Passo do Algoritmo

13. Solução por “SQL Batch”

O script abaixo resolve o problema e apresenta a informação da distância da ligação, ou seja, o número do passo em que a pessoa foi alcançada e também o vizinho imediatamente anterior. A solução do problema proposto seria melhor na forma de “stored procedure” que será assunto de próximas aulas. Por enquanto, vamos resolver na forma de “SQL Batch” para compreensão deste assunto e também porque este é uma prévia do que veremos sobre “stored procedure”. O código fonte está dividido em três partes:

i) Parte I: Antecede a parte repetitiva, contemplando a declaração de variáveis, inicialização e a definição do primeiro conjunto dos vértices periféricos;
ii) Parte II: O bloco repetitivo, ou seja, a parte iterativa controlada pelo comando “while”;
iii) Parte III: Apresentação dos resultados.

Estas três partes apresentadas separadamente devem ser executadas de uma só vez para que tenha o efeito de “SQL Batch”.

A figura 5 (Um passo do algoritmo) apresenta as ações que devem ser realizadas em cada passo, considerando que, da iteração anterior, recebe o conjunto total das pessoas ligadas até o momento e também a fronteira. As ações, resumidamente são:

i) Determinar as pessoas que estão nos vértices adjacentes à fronteira e que ainda não estão entre os amigos já encontrados até o passo anterior. Atendendo ao enunciado da questão, não importa se as pessoas são diretamente ou indiretamente ligadas e, ainda, se a ligação ocorre como requisitante ou respondente.

ii) As pessoas encontradas constituirão a nova fronteira.

iii) As pessoas da nova fronteira são incluídas no conjunto dos amigos diretos e indiretos.

iv) A parte iterativa se repete até que não tenha mais amigos a serem incluídos, ou seja, a iteração anterior detectou que o conjunto dos vértices da fronteira é vazio.

Observações:

i) Um olhar atento indicará que o algoritmo mostrado abaixo, na parte inicial, pode incluir uma linha vazia ao conjunto final e, dentro da parte iterativa, obrigatoriamente executa um último comando “insert” sem dados. Isto não interfere nos resultados, visto que tratam-se de conjunto vazio, mas estamos realizando uma operação desnecessária. Uma alteração simples resolve esta questão, basta retirar o comando que inclui os dados na tabela de amigos da parte inicial e movimentar o mesmo comando do final do “loop” para o início. Preferi a forma apresentada para ficar aderente à representação da figura 5, caso contrário, ela precisaria ser modificada porque no início da iteração o conjunto total dos amigos e da fronteira seriam disjuntos.

ii) O problema aqui proposto pode ser generalizado para que as arestas tenham um custo ou peso. Do ponto de vista do armazenamento da informação, precisamos incluir na representação conceitual (Diagrama de Entidades e Relacionamentos) um atributo de relacionamento chamado “custo”. Esta alteração repercute nos passos posteriores chegando ao banco de dados.

Do ponto de vista do algoritmo, há necessidade de revisão porque consideramos todas as arestas com o mesmo peso de valor unitário.

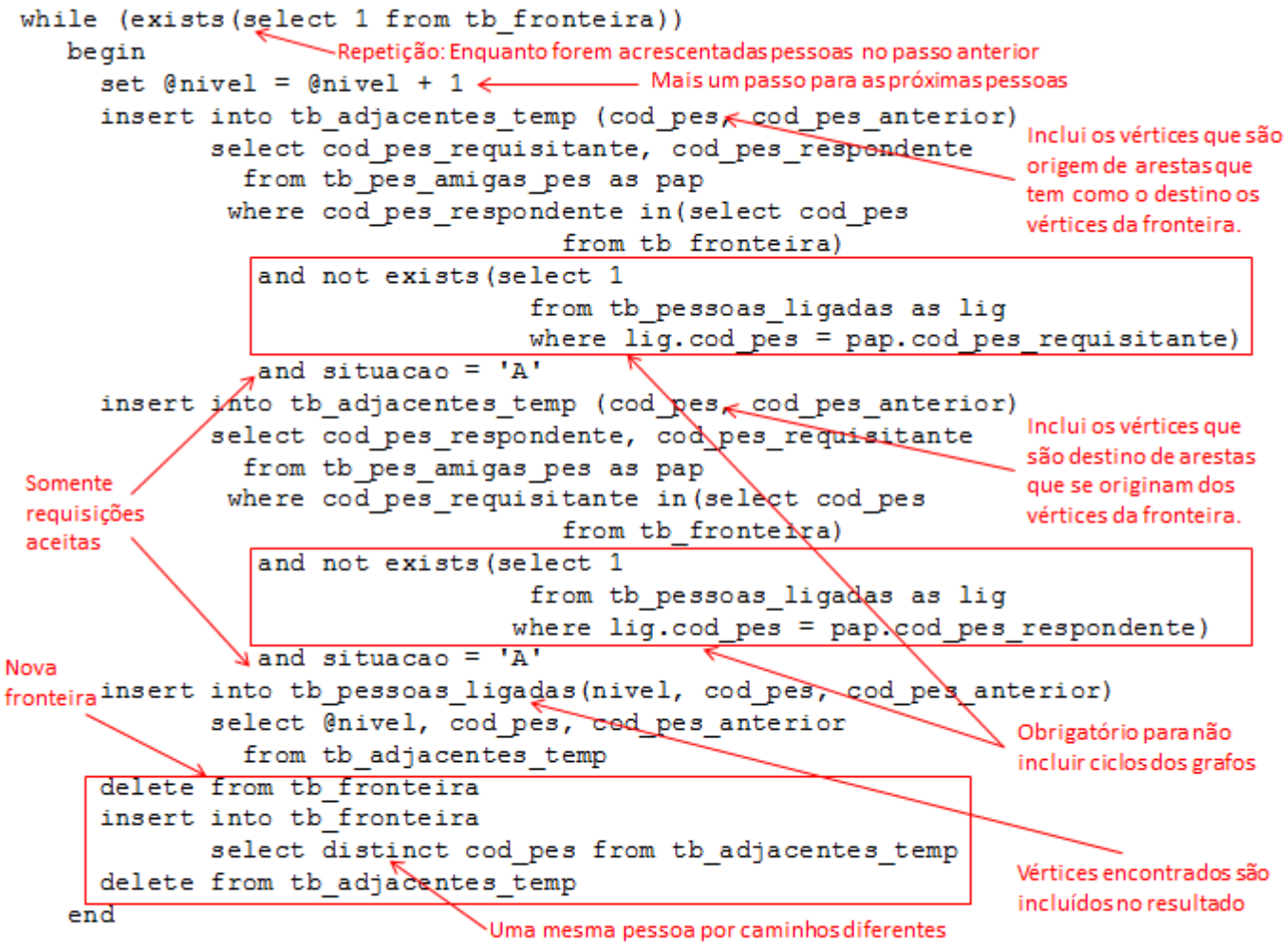
iii) Em termos de grafos, estamos procurando os vértices de subgrafos conexos maximais dentro de um grafo genérico. Neste sentido, a execução deste "SQL Batch" resulta no mesmo conjunto de vértices para qualquer vértice que pertence ao mesmo subgrafo; as diferenças são as distâncias que os separam.

13.1. SQL Batch: Parte I

```
declare
  @cod_pes char(7),        <- Código da pessoa informada
  @nivel int               <- Número do passo
begin
  set @cod_pes = 'P100000',<- Código da pessoa informada
      @nivel = 0           <- Número do passo inicial

  delete from tb_pessoas_ligadas
  delete from tb_fronteira
  delete from tb_adjacentes_temp
                           <- Limpeza das tabelas temporárias e com dados procurados

  insert into tb_pessoas_ligadas(nivel, cod_pes,
                                 cod_pes_anterior)
         values(@nivel,@cod_pes, '0000000')
                           <- Inserção da pessoa inicial

  insert into tb_fronteira (cod_pes)
         values(@cod_pes)
```

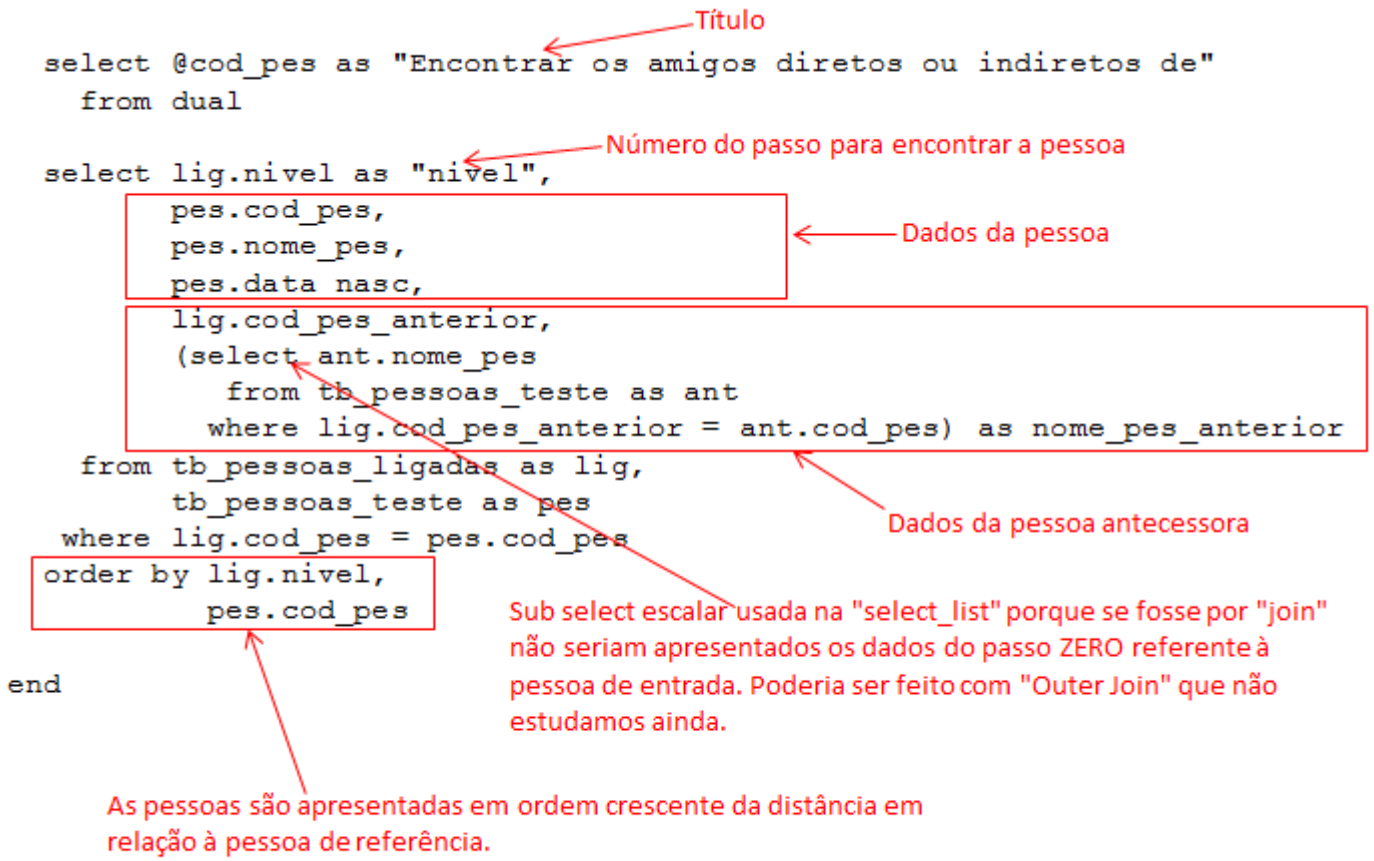
Parte I do "SQL Batch": Antes da fase repetitiva (Inicialização)

13.2. SQL Batch: Parte II



Parte II do "SQL Batch": Bloco repetitivo ("Loop")

13.3. SQL Batch: Parte III




```
select @cod_pes as "Encontrar os amigos diretos ou indiretos de"
  from dual

select lig.nivel as "nivel",
       pes.cod_pes,
       pes.nome_pes,
       pes.data_nasc,
       lig.cod_pes_anterior,
       (select ant.nome_pes
          from tb_pessoas_teste as ant
         where lig.cod_pes_anterior = ant.cod_pes) as nome_pes_anterior
  from tb_pessoas_ligadas as lig,
       tb_pessoas_teste as pes
 where lig.cod_pes = pes.cod_pes
order by lig.nivel,
         pes.cod_pes

end
```

Parte III do "SQL Batch": Apresentação dos resultados

14. Resultado da execução do "SQL Batch"

Segue o resultado da execução do "SQL Batch" para a pessoa de código "P100000".

```
Encontrar os amigos diretos ou indiretos de
-------------------------------------------
P100000

nivel       cod_pes nome           data_nasc        cod_pes_anterior nome_pes_anterior
----------- ------- -------------- ---------------- ---------------- ----------------
          0 P100000 Sandro              Jan  2 2000 0000000          NULL
          1 P101000 Ludmila             Apr 18 2002 P100000          Sandro
          1 P102000 Adriana             Jul 18 2001 P100000          Sandro
          1 P103000 Santoro             Dec  9 2004 P100000          Sandro
          1 P104000 Tobias              Oct 23 1999 P100000          Sandro
          1 P104200 Marco               Feb 18 2002 P100000          Sandro
          2 P101100 Jamil               Apr 20 2004 P101000          Ludmila
          2 P101100 Jamil               Apr 20 2004 P102000          Adriana
          2 P101100 Jamil               Apr 20 2004 P103000          Santoro
          2 P104100 Andressa            Nov  9 2001 P104000          Tobias
          3 P101110 Gian                Mar 25 2003 P101100          Jamil
          3 P101120 Amanda              Oct 15 2000 P101100          Jamil
          3 P101130 Fabiola             Nov 27 2000 P101100          Jamil
          4 P101111 Cassia              Apr 24 2002 P101110          Gian
```

Observações:

a) No resultado acima, a pessoa de código "P101100" aparece mais de uma vez como podemos ver nas linhas vermelhas. Isto ocorre porque estamos apresentando o vértice imediatamente anterior nos vários percursos que, neste caso, formam três caminhos diferentes. A leitura é a seguinte: O Sandro é amigo indireto de Jamil e está a duas arestas de distância através de Ludmila, Adriana e Santoro, os três amigos diretos. Se quisermos apresentar somente os vértices sem o seus antecessores, ele apareceria apenas uma vez no resultado. Pode-se alternativamente, escolher por apresentar apenas um dos predecessores por um critério qualquer, por exemplo, o menor código segundo a classificação lexicográfica.

b) A sequência de vértices do grafo percorrida da pessoa inicial a outra qualquer, que é alcançável, pode ser recomposta fazendo o percurso inverso, uma vez que, em cada vértice atingido temos o vértice predecessor informado.

c) A interpretação dos resultados em termos de nível e amigos é a seguinte: Os indivíduos do nível 1 são os amigos diretos da pessoa de código P100000 (Sandro); os do nível 2 são os amigos dos amigos; os do nível 3 são os amigos dos amigos dos amigos, e assim sucessivamente.

15. Legibilidade dos comandos e scripts

De forma análoga aos programas em linguagem de programação, é importante escrever o código fonte de forma legível com:

i) Alinhamento vertical conveniente que mostre a pertinência dos comandos aos seus respectivos blocos;
ii) Alinhamento vertical adequado para indicar as cláusulas que fazem parte de um comando;
iii) Comentários pertinentes (Seja razoável: tudo que seja essencial; nada mais que o essencial).

Até a próxima aula!
Prof. Satoshi Nagayama.