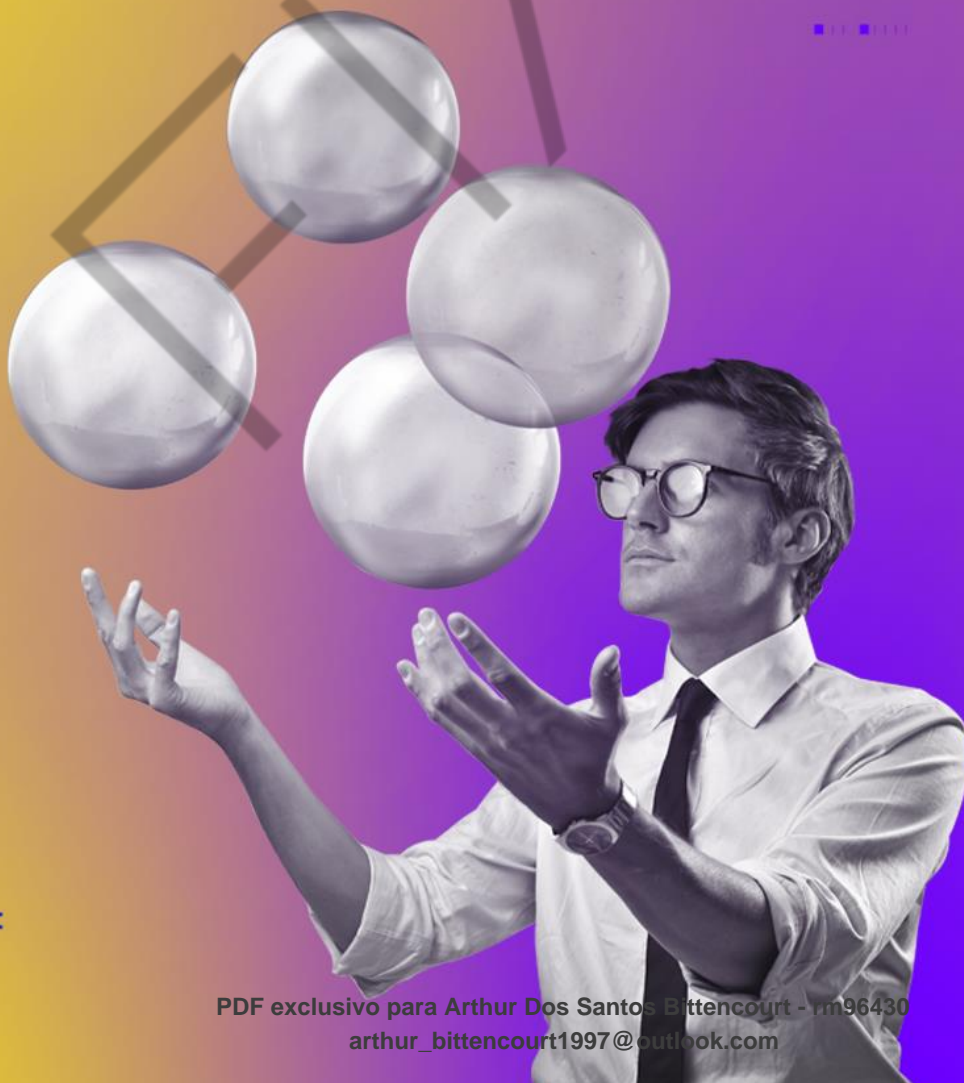


PROGRAMAÇÃO EM BANCO DE DADOS

# CURSORES

MILTON GOYA



3

**LISTA DE CÓDIGOS-FONTE**

Código-fonte 3.1 – Exemplo de consulta a todas as colunas de uma tabela sem %ROWTYPE .....	4
Código-fonte 3.2 – Exemplo de consulta a todas as colunas de uma tabela com %ROWTYPE .....	5
Código-fonte 3.3 – Exemplo de conjunto ativo .....	6
Código-fonte 3.4 – Exemplo de conjunto ativo .....	6
Código-fonte 3.5 – Exemplo de erro gerado pela ausência de um cursor explícito...	7
Código-fonte 3.6 – Exemplo do uso de atributos de cursor .....	8
Código-fonte 3.7 – Sintaxe da declaração de um CURSOR explícito .....	9
Código-fonte 3.8 – Exemplo de declaração de CURSOR explícito .....	9
Código-fonte 3.9 – Sintaxe da instrução OPEN .....	10
Código-fonte 3.10 – Exemplo de OPEN de CURSOR explícito .....	11
Código-fonte 3.11 – Sintaxe da instrução OPEN .....	11
Código-fonte 3.12 – Exemplo de uso da instrução FETCH de CURSOR explícito ...	12
Código-fonte 3.13 – Exemplo de uso da instrução FETCH com LOOP .....	13
Código-fonte 3.14 – Sintaxe da instrução CLOSE .....	13
Código-fonte 3.15 – Exemplo de uso da instrução FETCH com LOOP .....	14
Código-fonte 3.16 – Sintaxe da instrução LOOP de CURSOR FOR.....	14
Código-fonte 3.17 – Exemplo de uso LOOP de CURSOR FOR .....	15
Código-fonte 3.18 – Exemplo de uso LOOP FOR de CURSOR usando subconsulta .....	15
Código-fonte 3.19 – Exemplo de uso da instrução FOR UPDATE .....	16

## SUMÁRIO

3 CURSORES .....	4
3.1 Mais um pouco sobre tipos de dados .....	4
3.2 Tipos de cursores .....	6
3.2.1 Declaração de cursores explícitos .....	9
3.2.2 Abertura do cursor .....	10
3.2.3 Recuperação das linhas do cursor .....	11
3.2.4 Fechamento de cursor .....	13
3.2.5 Loops de CURSOR FOR .....	14
3.2.6 loops FOR de CURSOR usando subconsulta .....	15
3.2.7 Cursores de atualização .....	16
CONCLUSÃO .....	18
REFERÊNCIAS .....	19

### 3 CURSORES

#### 3.1 Mais um pouco sobre tipos de dados

Antes de começarmos a falar sobre cursores é necessário que voltemos ao tópico sobre os tipos de dados.

Aprendemos que, ao declarar uma variável, podemos usar o atributo %TYPE para que ela assuma o mesmo tipo e tamanho de uma coluna ou tipo de dados declarado anteriormente. Vimos vários exemplos simples, mas o que acontece se quisermos consultar todas as colunas de uma tabela? Vejamos outro exemplo simples:

```
SET SERVEROUTPUT ON

DECLARE
    v_empno      emp.empno%TYPE;
    v_ename      emp.ename%TYPE;
    v_job        emp.job%TYPE;
    v_mgr        emp.mgr%TYPE;
    v_hiredate    emp.hiredate%TYPE;
    v_sal        emp.sal%TYPE;
    v_comm       emp.comm%TYPE;
    v_deptno     emp.deptno%TYPE;

BEGIN
    SELECT empno, ename, job, mgr,
           hiredate, sal, comm, deptno
       INTO v_empno, v_ename, v_job, v_mgr,
           v_hiredate, v_sal, v_comm, v_deptno
      FROM emp
     WHERE empno = 7839;
    DBMS_OUTPUT.PUT_LINE ('Codigo      = ' || v_empno);
    DBMS_OUTPUT.PUT_LINE ('Nome        = ' || v_ename);
    DBMS_OUTPUT.PUT_LINE ('Cargo       = ' || v_job);
    DBMS_OUTPUT.PUT_LINE ('Gerente     = ' || v_mgr);
    DBMS_OUTPUT.PUT_LINE ('Data        = ' || v_hiredate);
    DBMS_OUTPUT.PUT_LINE ('Sala        = ' || v_sal);
    DBMS_OUTPUT.PUT_LINE ('Comissao    = ' || v_comm);
    DBMS_OUTPUT.PUT_LINE ('Depart.     = ' || v_deptno);
END;
/
```

Código-fonte 3.1 – Exemplo de consulta a todas as colunas de uma tabela sem %ROWTYPE  
Fonte: Elaborado pelo autor (2020)

Perceberam que foi preciso criar uma variável para cada coluna consultada na tabela? Podemos tornar esse processo mais fácil usando o atributo %ROWTYPE. Veja no exemplo:

```
SET SERVEROUTPUT ON

DECLARE
    emprec emp%ROWTYPE;

BEGIN
    SELECT *
      INTO emprec
    FROM emp
   WHERE empno = 7839;
    DBMS_OUTPUT.PUT_LINE ('Codigo    = ' || emprec.empno);
    DBMS_OUTPUT.PUT_LINE ('Nome      = ' || emprec.ename);
    DBMS_OUTPUT.PUT_LINE ('Cargo    = ' || emprec.job);
    DBMS_OUTPUT.PUT_LINE ('Gerente  = ' || emprec.mgr);
    DBMS_OUTPUT.PUT_LINE ('Data     = ' ||
emprec.hiredate);
    DBMS_OUTPUT.PUT_LINE ('Sala     = ' || emprec.sal);
    DBMS_OUTPUT.PUT_LINE ('Comissao = ' || emprec.comm);
    DBMS_OUTPUT.PUT_LINE ('Depart.  = ' ||
emprec.deptno);
    END;
/
```

Código-fonte 3.2 – Exemplo de consulta a todas as colunas de uma tabela com %ROWTYPE  
Fonte: Elaborado pelo autor (2020)

Notaram que só precisamos da variável EMPREC? Isso só foi possível porque usamos o atributo %ROWTYPE. Segundo a Oracle, 2016, o atributo %ROWTYPE fornece um tipo de registro que representa uma linha de uma tabela em um banco de dados relacional. O registro pode armazenar uma linha inteira de dados, que podem ser selecionados de uma tabela ou obtidos de um CURSOR ou de uma variável de CURSOR.

As variáveis que forem declaradas usando %ROWTYPE terão o mesmo nome e tipo de dados dos tipos de dados referenciados por elas. Em outras palavras, se a tabela EMP possuir uma coluna numérica inteira com duas posições denominada DEPTNO, o registro definido com %ROWTYPE também terá um campo numérico inteiro com duas posições denominado DEPTNO.

Para usar a variável criada no registro, basta usar o nome do campo precedido do nome do registro. Por exemplo, se o seu registro tem o nome de EMPREC e quer referenciar o campo DEPTNO use-o no formato EMPREC.DEPTNO.

É importante salientar que os campos em um registro definido por %ROWTYPE não herdam as restrições (*CONSTRAINTS*) e seus valores padrão.

### 3.2 Tipos de cursores

Para PUGA et al (2015), ao processar uma instrução de SQL, o banco de dados Oracle atribui uma área de trabalho na memória para a execução da instrução SQL. Essa área de memória é denominada área de contexto ou área SQL privada (*PRIVATE SQL AREA*). A área de contexto armazena informações necessárias para executar a instrução SQL.

Na área de contexto podemos encontrar: o número de linhas processadas, um apontador e, no caso de consultas, o conjunto ativo. O conjunto ativo é o conjunto de linhas recuperadas por uma consulta, por exemplo:

```
SELECT *  
FROM emp;
```

Código-fonte 3.3 – Exemplo de conjunto ativo  
Fonte: Elaborado pelo autor (2020)

No exemplo acima, o conjunto ativo será composto por todas as linhas e colunas da tabela EMP. Vejamos outro exemplo:

```
SELECT ename, job  
FROM emp  
WHERE deptno = 20;
```

Código-fonte 3.4 – Exemplo de conjunto ativo  
Fonte: Elaborado pelo autor (2020)

Nesse outro exemplo, o conjunto ativo será composto pelas colunas ENAME e JOB da tabela EMP que atenderem à condição DEPTNO = 20.

O CURSOR é o apontador da área de contexto. Um CURSOR permite que você nomeie uma instrução SQL, acesse a informação em sua área de contexto e, até certo ponto, controle seu processamento.

A Oracle, 2016, informa que existem dois tipos de cursores. Os cursores implícitos e os cursores explícitos.

Um **CURSOR implícito** não é declarado. O **CURSOR implícito** é criado sem a intervenção do usuário para todas as instruções de definição de dados (DDL), manipulação de dados (DML) e instruções **SELECT ... INTO** que retornam apenas uma linha. Se a sua consulta retornar mais de uma linha dentro do bloco PL/SQL um erro será gerado. Para corrigir esse erro, você declarar um **CURSOR**. Veja o erro acontecer no exemplo abaixo:

```
DECLARE
    emprec emp%ROWTYPE;
BEGIN
    SELECT SUM(sal)
        INTO emprec.sal
        FROM emp
    GROUP BY deptno;
    DBMS_OUTPUT.PUT_LINE ('Salario = ' || emprec.sal);
END;
/
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of
rows
ORA-06512: at line 4
```

Código-fonte 3.5 – Exemplo de erro gerado pela ausência de um cursor explícito  
Fonte: Elaborado pelo autor (2020)

Note que o programa retorna o erro ORA-01422. Esse erro foi gerado porque a execução do programa retornou mais de uma linha. Esse erro pode ser corrigido pela adoção do uso de um **CURSOR explícito**.

Segundo a Oracle (2016), um **CURSOR explícito** deve ser declarado explicitamente na área declarativa (**DECLARE**) do bloco PL/SQL. Ao usarmos um **CURSOR explícito**, podemos processar várias linhas, controlar a área de contexto e os processos que nela ocorrem.

Segundo a Oracle (2016), o conjunto retornado em um **CURSOR explícito** é denominado de **conjunto ativo**. O tamanho de um conjunto ativo depende da quantidade de linhas em uma tabela que atendam as condições aplicadas em uma consulta. O **CURSOR explícito** identifica a linha que está sendo processada no momento que é chamada de **linha atual**.

Para PUGA et al. (2015), tanto o **CURSOR implícito** quanto o **CURSOR explícito** possuem quatro atributos em comum: **%FOUND**, **%ISOPEN**, **%NOTFOUND**

e %ROWCOUNT. Esses atributos retornam informações úteis sobre a execução dos comandos.

**%FOUND** retorna verdadeiro (TRUE), caso alguma linha (tupla) tenha sido afetada.

**%ISOPEN** retorna verdadeiro (TRUE), caso o CURSOR esteja aberto.

**%NOTFOUND** retorna verdadeiro (TRUE), caso não tenha encontrado nenhuma tupla. Caso tenha encontrado, retornará falso (FALSE) até a última tupla.

**%ROWCOUNT** retorna o número de tuplas do CURSOR.

```
BEGIN
  DELETE
    FROM emp
  WHERE deptno = 10;
  DBMS_OUTPUT.PUT_LINE ('Linhas apagadas = ' ||
SQL%ROWCOUNT);
  ROLLBACK;
END;
/
```

Código-fonte 3.6 – Exemplo do uso de atributos de cursor  
Fonte: Elaborado pelo autor (2020)

No exemplo acima, o banco de dados Oracle cria um CURSOR implícito, apaga os registros do departamento 10 da tabela EMP, exibe a quantidade de linhas deletadas e desfaz a operação.

Para FEUERSTEIN et al (2014), o CURSOR implícito possui dois atributos adicionais: %BULK\_ROWCOUNT e %BULK\_EXCEPTION.

Um programa PL/SQL abre um cursor, processa linhas retornadas por uma consulta e, em seguida, fecha o cursor. O cursor marca a posição atual no conjunto ativo.

A sequência para execução do controle dos cursores consiste em:

1. Declaração do CURSOR - Declare o cursor nomeando-o e definindo a estrutura da consulta a ser executada dentro dele.
2. Abertura do CURSOR - Abra o cursor. A instrução OPEN executa a consulta e vincula as variáveis que estiverem referenciadas. As linhas identificadas pela consulta são chamadas conjunto ativo e estão agora disponíveis para extração.



3. Recuperação das linhas - Extraia dados do cursor.
4. Verificação do término das tuplas - Após cada extração, você testa o cursor para qualquer linha existente. Se não existirem mais linhas para serem processadas, você precisará fechar o cursor.
5. Fechamento do CURSOR - Feche o cursor. A instrução CLOSE libera o conjunto ativo de linhas. Agora é possível reabrir o cursor e estabelecer um novo conjunto ativo, se necessário.

### 3.2.1 Declaração de cursores explícitos

Para a Oracle (2016), os cursores explícitos devem ser declarados no bloco PL/SQL e a manipulação do seu conjunto ativo deve ser realizada na área de processamento do bloco.

```
CURSOR nome_cursor IS  
    consulta;
```

Código-fonte 3.7 – Sintaxe da declaração de um CURSOR explícito  
Fonte: ORACLE (2016)

**onde,**

**CURSOR** é a instrução utilizada para declarar um cursor explícito.

**nome\_cursor** é um identificador para o cursor.

**consulta** é uma instrução SELECT.

Não se deve incluir a cláusula INTO na declaração de cursor, porque ela aparecerá posteriormente na instrução FETCH.

Pode-se fazer referência a variáveis dentro da consulta, mas elas devem ser declaradas antes da instrução CURSOR.

Vejamos um exemplo do uso da declaração de um CURSOR:

DECLARE

```
CURSOR cursor_emp IS  
    SELECT deptno, SUM(sal)  
    FROM emp  
    GROUP BY deptno;
```

...

Código-fonte 3.8 – Exemplo de declaração de CURSOR explícito  
Fonte: Elaborado pelo autor (2020)

No exemplo estamos criando um CURSOR de nome CURSOR\_EMP que fará a soma dos salários de todos os funcionários e os agrupará pelo código do departamento em que trabalham. Após a declaração do CURSOR, devemos abri-lo.

### 3.2.2 Abertura do cursor

Para a ORACLE, 2016, a instrução OPEN abre o cursor para executar a consulta e identificar o conjunto ativo, que consiste em todas as linhas que atendem aos critérios de pesquisa da consulta. O cursor aponta agora para a primeira linha do conjunto ativo.

```
OPEN nome_cursor;
```

Código-fonte 3.9 – Sintaxe da instrução OPEN  
Fonte: ORACLE (2016)

**onde,**

**OPEN** é a instrução utilizada para abrir um cursor explícito.

**nome\_cursor** é um identificador para o cursor

OPEN é uma instrução executável que realiza as seguintes operações:

- Aloca memória dinamicamente para uma área de contexto que finalmente conterá informações cruciais de processamento;
- Analisa a instrução SELECT;
- Vincula as variáveis de entrada — isto é, define o valor das variáveis de entrada obtendo seus endereços de memória;
- Identifica o conjunto ativo — isto é, o conjunto de linhas que satisfaz os critérios de pesquisa. As linhas no conjunto ativo não são recuperadas para variáveis quando a instrução OPEN é executada. Em vez disso, a instrução FETCH recupera as linhas;
- Posiciona o indicador imediatamente antes da primeira linha no conjunto ativo.

É importante notar que, se a consulta não retornar qualquer linha quando o cursor for aberto, o PL/SQL não criará uma exceção, é possível testar o status do cursor após a extração.

Abaixo, um exemplo do uso da instrução OPEN.

DECLARE

```
CURSOR cursor_emp IS
    SELECT deptno, SUM(sal)
    FROM emp
    GROUP BY deptno;
BEGIN
    OPEN cursor_emp;
END;
/
```

Código-fonte 3.10 – Exemplo de OPEN de CURSOR explícito  
Fonte: Elaborado pelo autor (2020)

O exemplo acima é a continuação do exemplo anterior. Após definirmos o CURSOR denominado de CURSOR\_EMP, o abrimos com o comando OPEN.

O próximo passo é usarmos a instrução FETCH para recuperarmos os dados do conjunto ativo.

### 3.2.3 Recuperação das linhas do cursor

Para a ORACLE, 2016, a instrução FETCH recupera as linhas no conjunto ativo uma de cada vez. Após cada extração, o cursor avança para a próxima linha no conjunto ativo.

```
FETCH cursor_name
INTO [variável1, variável2, ...|record_name];
```

Código-fonte 3.11 – Sintaxe da instrução OPEN  
Fonte: ORACLE (2016)

**onde,**

**FETCH** é a instrução utilizada para extrair dados de um conjunto ativo.

**nome\_cursor** é um identificador para o cursor.

**variável** é uma variável de saída para armazenar os resultados, o número de variáveis deve ser compatível com o número de colunas da consulta.

**record\_name** é o nome do registro em que os dados recuperados são armazenados. A variável de registro pode ser declarada usando o atributo %ROWTYPE.

A instrução FETCH realiza as seguintes operações:

- Avança o indicador para a próxima linha no conjunto ativo;

- Lê os dados da linha atual para as variáveis PL/SQL de saída.

Para PUGA, et al (2015), deve-se incluir o mesmo número de variáveis na cláusula INTO da instrução FETCH do que as colunas na instrução SELECT, as variáveis devem ser do mesmo tipo de dado da coluna correspondente. Como alternativa, pode ser definido um registro para o cursor que faça referência do registro na cláusula FETCH INTO.

Também é necessário verificar se o cursor possui linhas. Se uma extração não obtiver valores, não existem linhas remanescentes para serem processadas no conjunto ativo e nenhum erro será registrado. No caso de não existirem mais linhas é chegado o momento de fechar o CURSOR.

```
DECLARE
    emprec emp%ROWTYPE;
    CURSOR cursor_emp IS
        SELECT deptno, SUM(sal)
           FROM emp
          GROUP BY deptno;
BEGIN
    OPEN cursor_emp;
    FETCH cursor_emp INTO emprec.deptno, emprec.sal;
    DBMS_OUTPUT.PUT_LINE ('Departamento: ' || emprec.deptno);
    DBMS_OUTPUT.PUT_LINE ('Salario      : ' || emprec.sal);
END;
/
```

Código-fonte 3.12 – Exemplo de uso da instrução FETCH de CURSOR explícito  
Fonte: Elaborado pelo autor (2020)

O exemplo acima cria uma variável do tipo ROWTYPE de nome EMPREC, define um CURSOR de nome CURSOR\_EMP, o abre com o comando OPEN, extrai os dados com a instrução FETCH e exibe os códigos do departamento e a soma dos salários desse departamento. Existe um problema grave nesse código: ele só executa a instrução FETCH uma vez. Para que o programa funcione corretamente, é necessário que a instrução FETCH seja executada até que todos os dados sejam extraídos. Será necessário acrescentar um laço de repetição, ou LOOP, para que extraíamos todos os dados do conjunto ativo. Veja o exemplo a seguir:

```
DECLARE
    emprec emp%ROWTYPE;
    CURSOR cursor_emp IS
        SELECT deptno, SUM(sal)
           FROM emp
          GROUP BY deptno;
```

```
BEGIN
  OPEN cursor_emp;
  LOOP
    FETCH cursor_emp INTO emprec.deptno, emprec.sal;
    EXIT WHEN cursor_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE ('Departamento:      ' ||
emprec.deptno);
    DBMS_OUTPUT.PUT_LINE ('Salario          : ' || emprec.sal);
  END LOOP;
END;
/
```

Código-fonte 3.13 – Exemplo de uso da instrução FETCH com LOOP  
Fonte: Elaborado pelo autor (2020)

No exemplo acima, incluímos a instrução LOOP. O laço será encerrado quando o atributo %NOTFOUND retornar o valor TRUE. Esse atributo só retorna TRUE quando não localiza mais nenhuma tupla no conjunto ativo.

### 3.2.4 Fechamento de cursor

Para a ORACLE, 2016, a instrução CLOSE desativa o cursor e o conjunto ativo se torna indefinido. É uma boa prática fechar o cursor após completar o processamento da instrução SELECT. Se necessário o cursor poderá ser reaberto. O conjunto ativo pode ser estabelecido diversas vezes durante a execução do programa. Ao fecharmos um CURSOR, a área de contexto é liberada.

```
CLOSE nome_cursor;
```

Código-fonte 3.14 – Sintaxe da instrução CLOSE  
Fonte: ORACLE (2016)

**onde,**

**CLOSE** é a instrução utilizada para fechar um cursor.

**nome\_cursor** é um identificador para o cursor.

Embora seja possível terminar o bloco PL/SQL sem fechar cursores, é boa prática fechar qualquer cursor declarado explicitamente para liberar recursos.

Vejamos nosso programa com a instrução CLOSE.

```
DECLARE
  emprec emp%ROWTYPE;
  CURSOR cursor_emp IS
    SELECT deptno, SUM(sal)
    FROM emp
```

```
        GROUP BY deptno;
BEGIN
    OPEN cursor_emp;
    LOOP
        FETCH cursor_emp INTO emprec.deptno, emprec.sal;
        EXIT WHEN cursor_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE      ('Departamento:      ' ||
emprec.deptno);
        DBMS_OUTPUT.PUT_LINE ('Salario      : ' || emprec.sal);
    END LOOP;
    CLOSE cursor_emp;
END;
/
```

Código-fonte 3.15 – Exemplo de uso da instrução FETCH com LOOP  
Fonte: Elaborado pelo autor (2020)

O exemplo acima é continuação do exemplo que estamos desenvolvendo. Dessa vez incluímos a instrução CLOSE para encerrar nosso CURSOR.

A forma tradicional de trabalharmos com o CURSOR envolve a declaração do CURSOR, abrir o CURSOR, extrair o conjunto ativo dentro de um laço e, finalmente, encerrar o CURSOR. Com a evolução da linguagem PL/SQL, foram criadas outras formas de trabalhar com CURSOR. Vamos examiná-las.

### 3.2.5 Loops de CURSOR FOR

Para ORACLE, 2016, o LOOP de CURSOR FOR é uma forma mais simplificada de usar um CURSOR. O CURSOR é aberto implicitamente, os dados do conjunto ativo são extraídos a cada iteração do LOOP. O LOOP é encerrado assim que a última tupla for processada e o CURSOR é encerrado implicitamente.

```
FOR nome_registro IN nome_cursor LOOP
    Instruções;
END LOOP;
```

Código-fonte 3.16 – Sintaxe da instrução LOOP de CURSOR FOR  
Fonte: ORACLE (2016)

**onde,**

**FOR** é a instrução utilizada para iniciar o laço.

**nome\_registro** é um identificador para o registro.

**nome\_cursor** é um identificador para o cursor.

**END LOOP;** é a instrução utilizada para encerrar o laço.

Vamos alterar nosso exemplo para usar o LOOP de CURSOR FOR

```
DECLARE
  CURSOR cursor_emp IS
    SELECT deptno, SUM(sal) soma
    FROM emp
    GROUP BY deptno;
BEGIN
  FOR emprec IN cursor_emp LOOP
    DBMS_OUTPUT.PUT_LINE ('Departamento: ' || emprec.deptno);
    DBMS_OUTPUT.PUT_LINE ('Salario      : ' || emprec.soma);
  END LOOP;
END;
```

```
/
```

Código-fonte 3.17 – Exemplo de uso LOOP de CURSOR FOR  
Fonte: Elaborado pelo autor (2020)

No exemplo acima, o registro foi definido no início do laço FOR. O registro terá a estrutura necessária para receber o conjunto ativo definido na declaração do CURSOR. As operações de OPEN, FETCH e CLOSE foram executadas implicitamente dentro do laço FOR. É uma forma simplificada de uso do CURSOR.

### 3.2.6 loops FOR de CURSOR usando subconsulta

Para a Oracle (2016), a evolução da linguagem PL/SQL acabou levando a não ser mais necessário declarar um CURSOR explicitamente na sessão de DECLARE.

O CURSOR pode ser substituído por uma subconsulta dentro do LAÇO FOR. Observe que não será possível fazer referência aos atributos de cursores explícitos se você usar uma subconsulta em um loop FOR de cursor, porque você não poderá dar um nome explícito ao cursor. Veja no exemplo abaixo.

```
BEGIN
  FOR emprec IN (SELECT deptno, SUM(sal) soma
    FROM emp GROUP BY deptno)
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Departamento: ' || emprec.deptno);
    DBMS_OUTPUT.PUT_LINE ('Salario      : ' || emprec.soma);
  END LOOP;
END;
/
```

Código-fonte 3.18 – Exemplo de uso LOOP FOR de CURSOR usando subconsulta  
Fonte: Elaborado pelo autor (2020)

No exemplo acima, a sessão declarativa foi eliminada. Tanto o registro como o CURSOR são definidos dentro do laço FOR.

### 3.2.7 Cursores de atualização

Para a Oracle (2016), o comando SELECT FOR UPDATE permite que você bloqueie (LOCK) linhas específicas de uma tabela para garantir que nenhuma alteração será efetuada nessas linhas após você lê-las e serem alocadas no conjunto ativo.

Você deve usar a cláusula FOR UPDATE para adquirir bloqueios exclusivos ao declarar um CURSOR que será referenciado na cláusula CURRENT OF de uma instrução UPDATE ou DELETE.

A instrução SELECT ... FOR UPDATE identifica as linhas que serão atualizadas ou excluídas, e bloqueará cada linha no conjunto de resultados.

A palavra-chave opcional NOWAIT diz a Oracle que não aguarde se as linhas solicitadas foram bloqueadas por outro usuário. O controle é imediatamente retornado ao seu programa para que ele possa fazer outros trabalhos antes de tentar novamente adquirir o bloqueio. Se você omitir a palavra-chave NOWAIT, a Oracle aguarda até que as linhas estejam disponíveis.

Vejamos um exemplo:

```
DECLARE
  emprec emp%ROWTYPE;
  CURSOR cursor_emp IS
    SELECT empno, sal
      FROM emp
     FOR UPDATE;
BEGIN
  OPEN cursor_emp;
  LOOP
    FETCH cursor_emp INTO emprec.empno, emprec.sal;
    EXIT WHEN cursor_emp%NOTFOUND;
    UPDATE emp SET sal = sal * 1.05 WHERE CURRENT OF
cursor_emp;
  END LOOP;
  CLOSE cursor_emp;
END;
/
```

Código-fonte 3.19 – Exemplo de uso da instrução FOR UPDATE  
Fonte: Elaborado pelo autor (2020)



O exemplo acima bloqueia as linhas extraídas para o conjunto ativo. A cada iteração do laço, atualiza o salário do funcionário em 5%.



## CONCLUSÃO

Conforme visto neste capítulo, cursores são estruturas de dados essenciais para se trabalhar com uma massa grande de dados, e os vários tipos de estruturas do tipo cursor permitem flexibilidade ao utilizá-las nas várias situações em que são necessários.



## REFERÊNCIAS

DILLON, Sean; BECK, Christopher; KYTE, Thomas; KALLMAN, Joel; ROGERS, Howard. **Beginning Oracle Programming**. Apress, 2013.

FEUERSTEIN, Steven; PRIBYL, Bill. **Oracle PL/Sql Programming**. O'Reilly Media, 2014.

ORACLE. **Oracle Database: PL/SQL Language Reference 12c Release 2 (12.2)** B28370-05. Oracle Press, 2016.

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **Banco de dados**. São Paulo: Pearson, 2015.