**Integração SQL – PL/SQL**

A linguagem SQL (Structured Query Language) é um padrão em bancos de dados relacionais. Ela é uma **linguagem de domínio específico** (DSL – Domains Specific Language), isto é, ela foi criada com o propósito específico de manipular objetos em bancos de dados relacionais. Também é uma **linguagem declarativa**. Isto significa que os “programas” (scripts) em SQL se preocupam em definir o “que” e não o “como”.

              A linguagem SQL é dividida em sublinguagens, de acordo com o tipo de operação executada. A divisão mais granular define 5 sublinguagens:

●        **DDL** (**D**ata ***D***efinition ***L***anguage) – Utilizada na manipulação (criação, alteração, exclusão) de **objetos** do banco de dados (comandos CREATE, ALTER e DROP);

●        **DML** (**D**ata ***M***anipulation ***L***anguage) – Utilizada para manipulação do **conteúdo** de objetos do banco de dados, ou seja, os dados propriamente ditos (comandos INSERT, UPDATE e DELETE);

●        **DQL**(**D**ata ***Q***uery ***L***anguage) – Utilizada na **consulta** dados (comando SELECT);

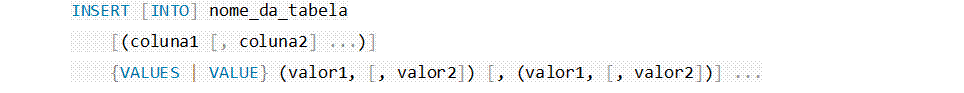
●        **TCL**(**T**ransaction ***C***ontrol ***L***anguage) – Utilizada no controle de **transações** (comandos SET TRANSACTION, START TRANSACTION, COMMIT, ROLLBACK e SAVEPOINT);

●        **DCL** (**D**ata ***C***ontrol ***L***anguage) – Utilizada no controle da segurança dos dados, atribuindo permissões e privilégios de usuários (comandos GRANT, REVOKE e DENY).

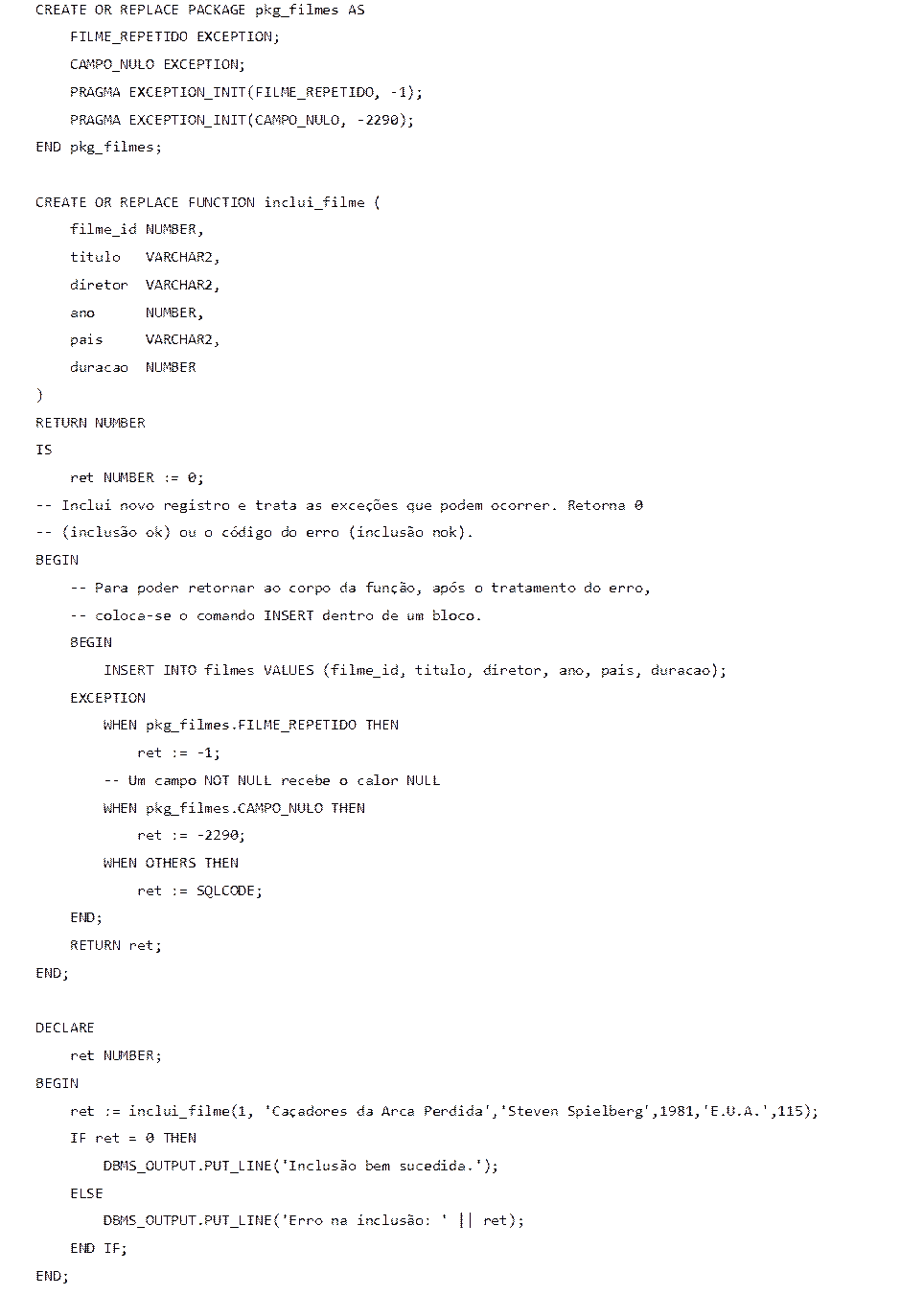
Pelo fato de ter sido desenvolvida especificamente para “rodar” em um SGBDR, a linguagem PL/SQL se integra de forma muito natural com SQL. No entanto, algumas restrições se aplicam. Comandos DDL, utilizados na criação, alteração e exclusão de objetos, não são permitidos em programas PL/SQL.¹ Já os comandos DML e DQL, os mais utilizados no desenvolvimento de aplicações, são permitidos.

### *INCLUSÃO DE REGISTROS*

O comando INSERT é utilizado para a inserção de novos registros em uma tabela. Sua forma geral é:

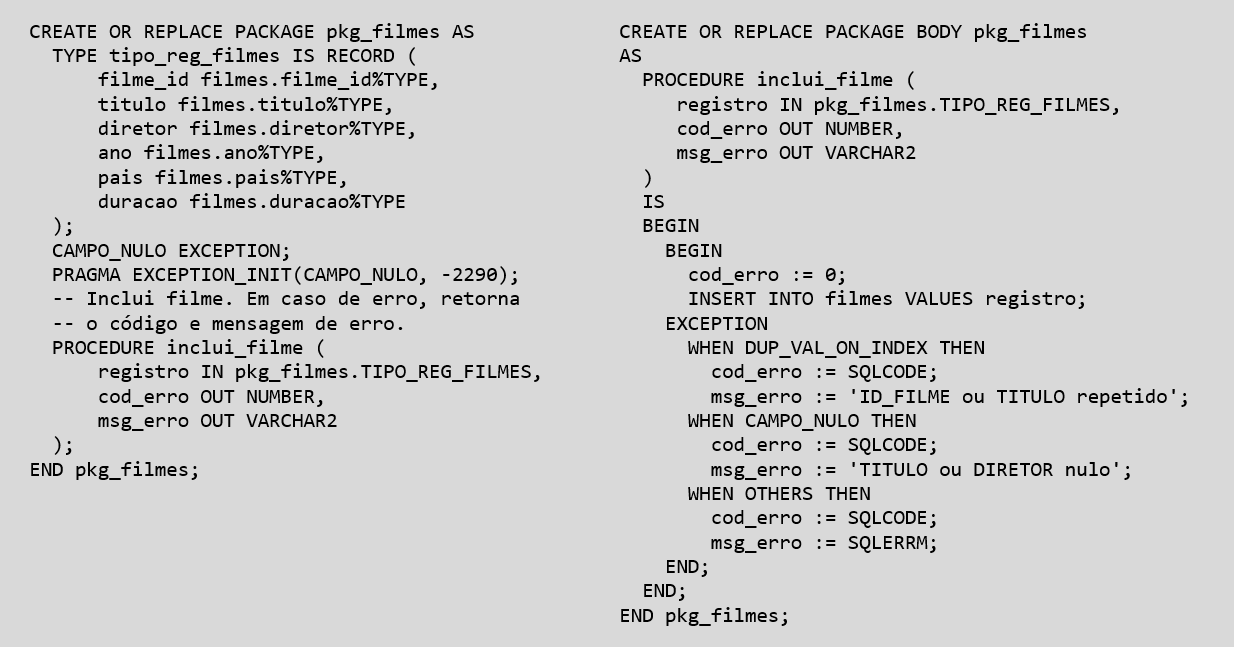


​A relação (coluna1, coluna2, …) é opcional. Quando ela é utilizada, não é necessário colocar os nomes de todas as colunas, mas apenas daquelas que receberão valores (as demais recebem os valores padrão ou NULL). Agora, quando não utilizada, devem ser fornecidos valores para todas as colunas, na ordem em que estão na tabela. O exemplo a seguir ilustra a utilização do comando INSERT em uma procedure. (os scripts SQL para a criação das tabelas encontram-se no material de apoio).​



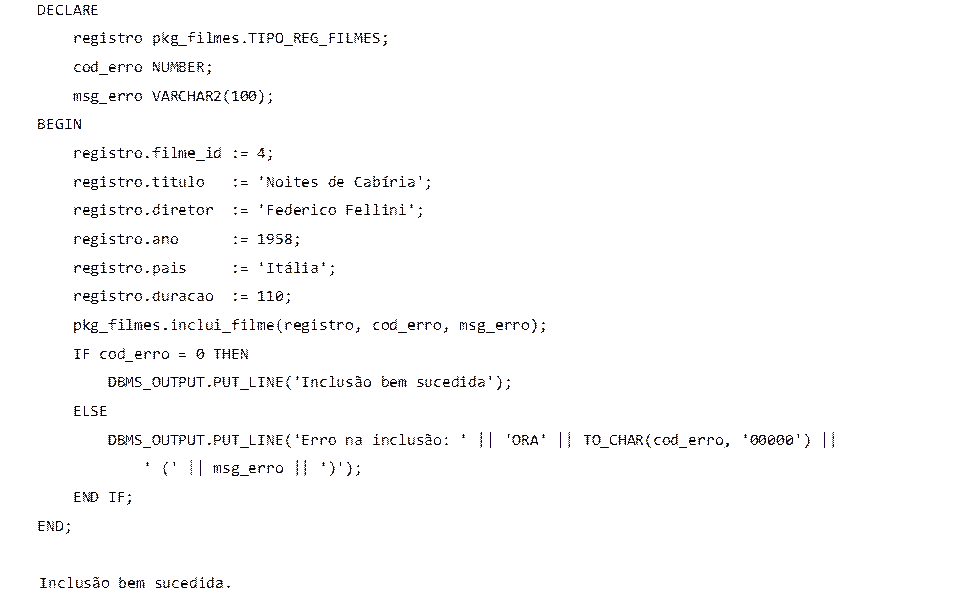
A mensagem ‘Inclusão bem sucedida.’ é exibida após a execução do bloco anônimo.

É possível a utilização do tipo RECORD no lugar dos campos individuais da tabela FILMES tanto na passagem de parâmetros quanto no comando INSERT. A procedure inclui\_filme será colocada dentro da package pkg\_filmes. Além disto, o tratamento de exceções foi alterado a fim de retornar ao programa chamador o código e a mensagem de erro. A seguir, a versão modificada do exemplo anterior.



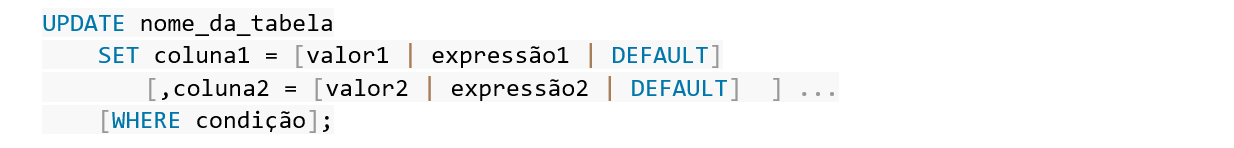
Observe que, na package, os tipos dos campos do RECORD foram definidos de forma diferente. Esta forma é chamada **ancoragem de tipo**e é mais uma boa característica da linguagem PL/SQL, dentro da filosofia orientada a modularidade e de esconder do desenvolvedor detalhes desnecessários. Como cada campo do RECORD corresponde a um campo da tabela, nada mais lógico do que atribuir ao primeiro o mesmo tipo do segundo. A forma geral é tabela.campo%TYPE. Recomenda-se que a ancoragem de tipos sempre que possível.

O bloco anônimo e o resultado exibido após sua execução são mostrados a seguir.

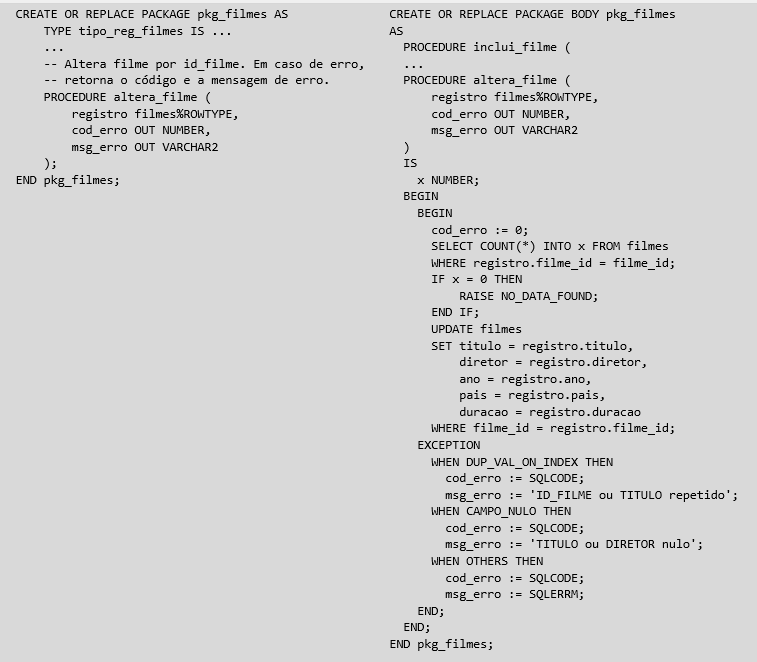


### *ALTERAÇÃO DE REGISTROS*

O comando UPDATE é utilizado para alteração de registros existentes em uma tabela. Sua forma geral é:



​O valor atribuído pode ser uma constante, uma expressão ou o valor padrão para a coluna (DEFAULT). A cláusula SET atribui valores às colunas. Se a cláusula WHERE for incluída, apenas o(s) registro(s) em que condição for verdadeira serão afetados. Caso contrário, todos os registros da tabela serão afetados. O exemplo a seguir ilustra a utilização do comando UPDATE (apenas as alterações na package serão mostradas).​



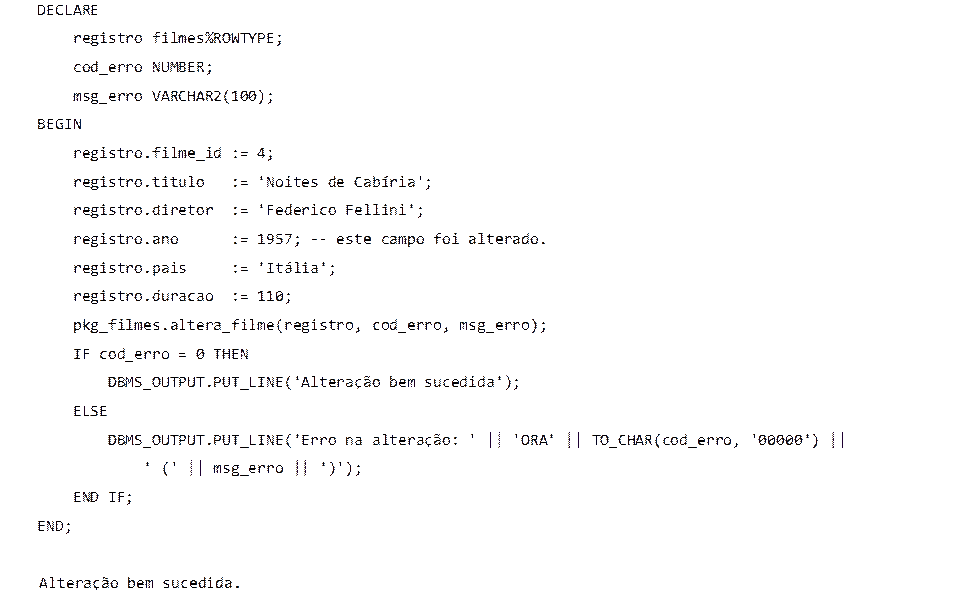
Observe duas coisas. Primeiro, foi utilizado o tipo filmes%ROWTYPE para o parâmetro registro no lugar de TIPO\_REG\_FILMES. A ancoragem de tipo, neste caso, atribuiu a registro um tipo RECORD com a mesma estrutura da tabela filmes. Esta é uma forma muito prática de ancoragem de tipos. Segundo, foi incluído o comando SELECT INTO antes do UPDATE. A função deste comando é simplesmente verificar se o registro a ser alterado existe em FILMES. O comando SELECT INTO atribui à variável x o valor retornado por COUNT(\*). Se este valor for 0 (não existe o registro), a exceção NO\_DATA\_FOUND é gerada.

Uma forma mais compacta pode ser utilizada no comando UPDATE. No lugar de se atribuir valores às colunas uma a uma, o comando poderia ser escrito da forma a seguir.

https://paperx-dex-assets.s3.sa-east-1.amazonaws.com/images/1679504913941-JTVvLRfZm7.png

O identificador ROW indica que todos os campos dos registros selecionados pela cláusula WHERE serão alterados, recebendo o conteúdo de registro.

O bloco anônimo e o resultado exibido após sua execução são mostrados a seguir.

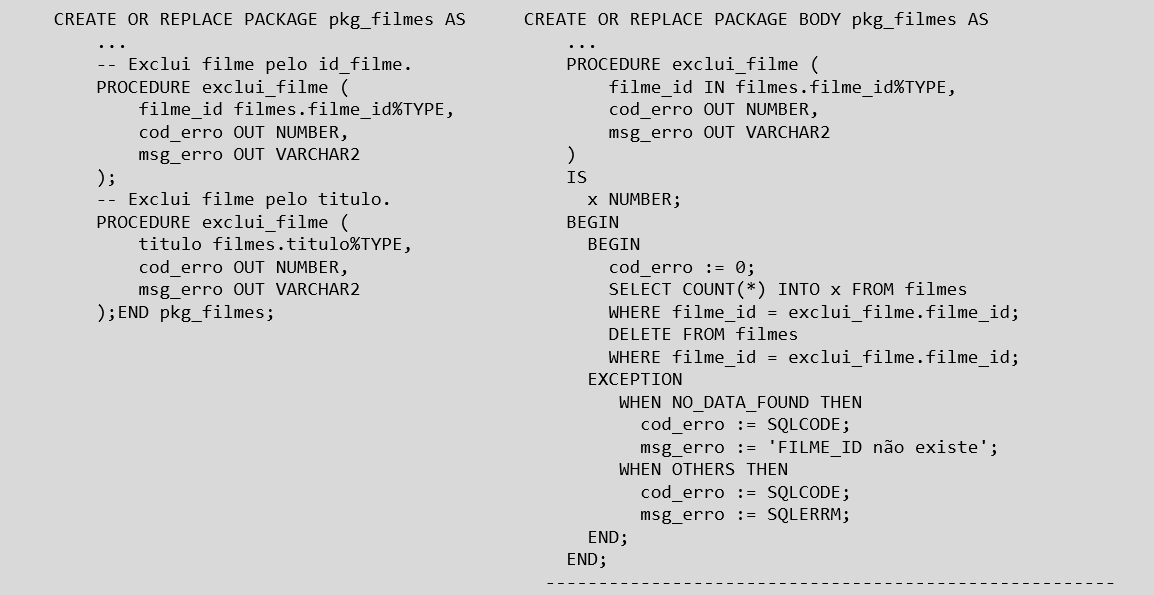


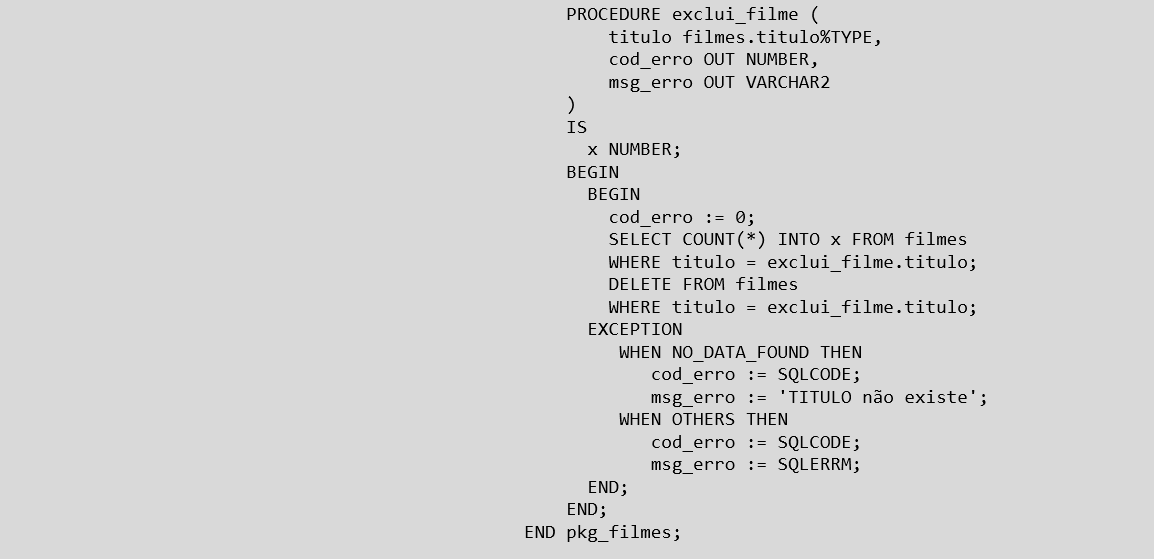
### *EXCLUSÃO DE REGISTROS*

O comando DELETE é utilizado para remover registros. Muito cuidado: a cláusula WHERE **deve** ser utilizada para selecionar os registros a serem excluídos ou **todos** os registros da tabela serão excluídos. A sintaxe mais comum de DELETE é simples:

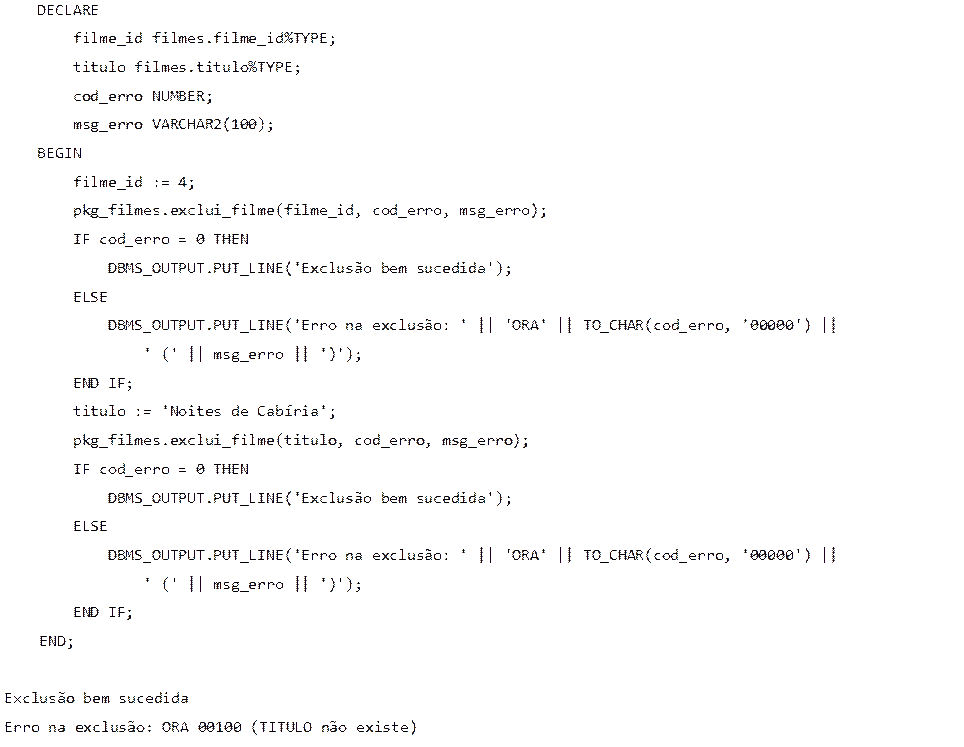
https://paperx-dex-assets.s3.sa-east-1.amazonaws.com/images/1679504956720-3y1UeHUCkm.png

O exemplo a seguir implementa duas versões da função exclui\_registro, utilizando overloading (apenas as alterações na package serão mostradas).





O bloco anônimo e o resultado exibido após sua execução são mostrados a seguir.​

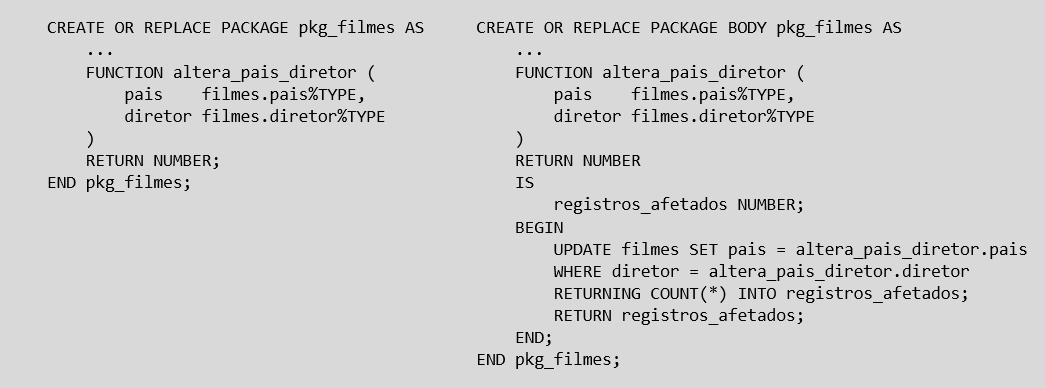


### *A CLÁUSULA*RETURNING

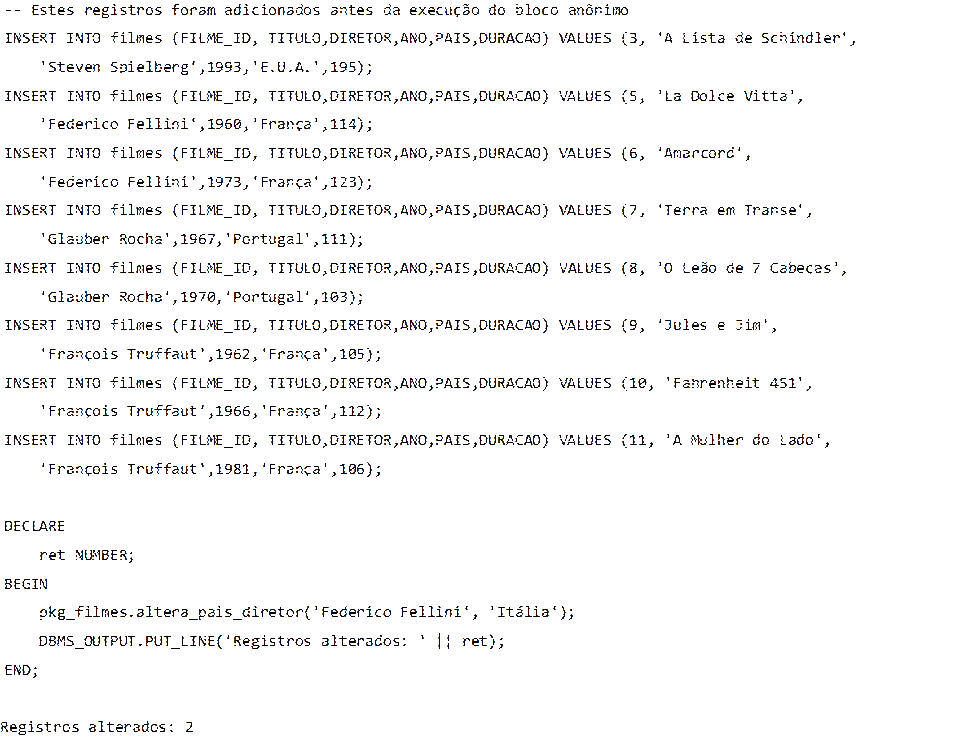
A cláusula RETURNING INTO, quando usada em conjunto com os comandos INSERT, DELETE ou UPDATE permite que informações a respeito da execução do comando DML sejam atribuídos a variáveis. A sua forma geral é mostrada a seguir.

https://paperx-dex-assets.s3.sa-east-1.amazonaws.com/images/1679505059707-GhXIuYY9mr.png

​Expressão pode envolver literais e/ou colunas da tabela referenciada no comando DML e a quantidade e o tipo das expressões deve ser o mesmo das variáveis. O exemplo a seguir cria a função altera\_pais\_diretor, que altera o país de um determinado diretor, ambos passados como parâmetros, e retorna o número de registros afetados.​

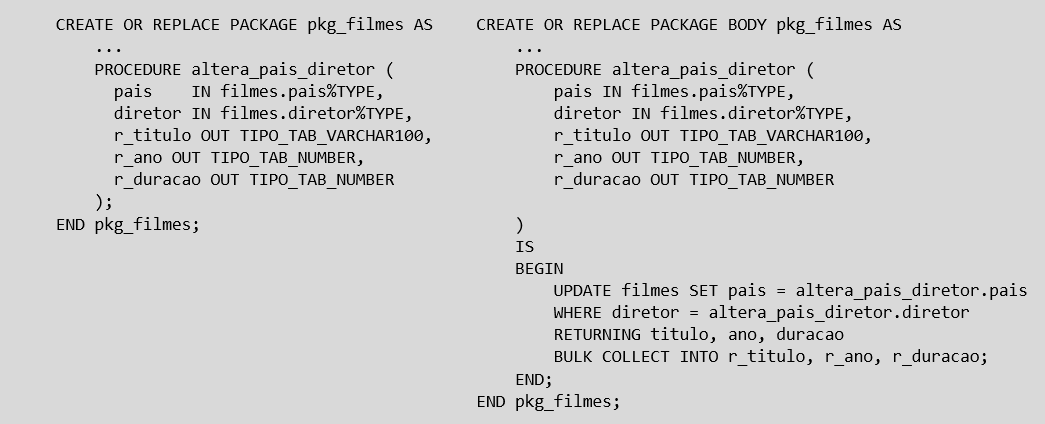


O bloco anônimo que chama esta função e o resultado exibido são mostrados a seguir.



Esta forma da cláusula RETURNING INTO deve ser utilizada quando se sabe que o comando afetará uma linha apenas ou quando são utilizadas funções de agregação para as linhas afetadas (SUM, AVG, MAX etc.). Se o comando associado afetar mais de uma linha da tabela e não forem utilizadas funções de agregação, será gerado o erro ORA-01422: exact fetch returns more than requested number of rows. Se nenhuma linha for afetada, as variáveis receberão valor NULL ou 0, no caso de SUM(\*)

A cláusula RETURNING INTO também permite retornar resultados com mais de uma linha afetada. Nestes casos, deve ser usada a forma RETURNING BULK COLECT INTO. As variáveis que receberão os valores devem ser coleções. No exemplo a seguir, a função altera\_pais\_diretor é transformada em procedure e é modificada a fim de retornar também o título,  ano de lançamento e duração dos registros afetados.

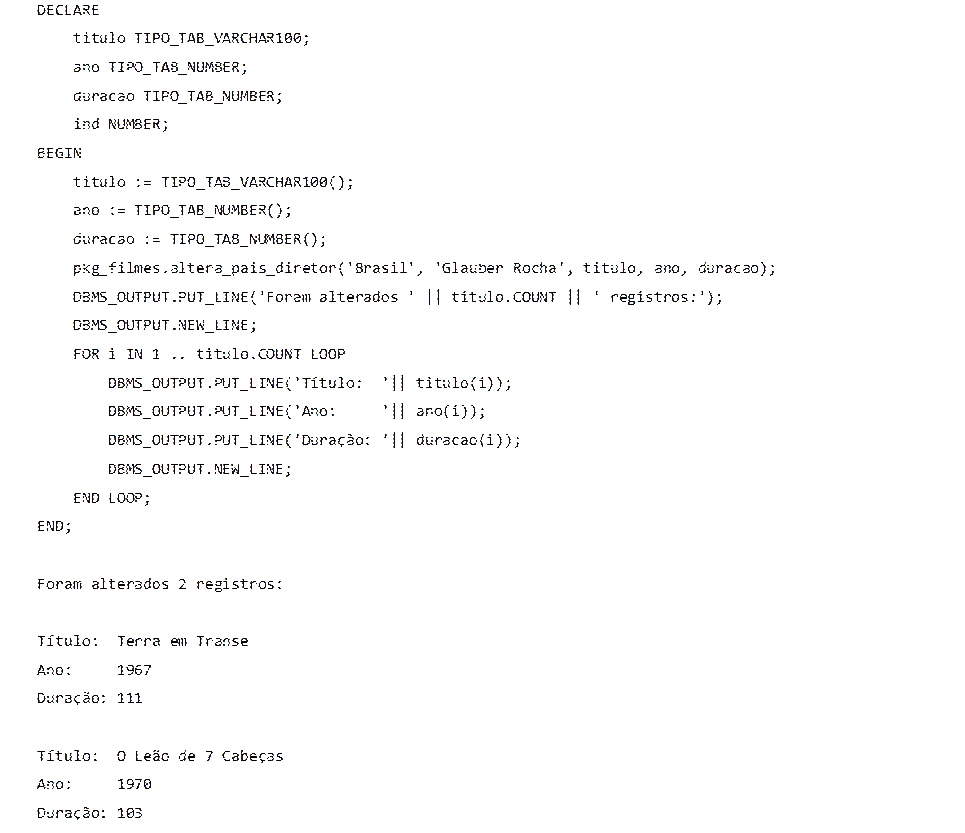


Observe que não são necessários a inicialização e o uso do método EXTEND para as coleções. A própria cláusula RETURNING BULK COLLECT INTO se encarrega de inicializar e estender a coleção para o tamanho necessário.

Observe também que os tipos TIPO\_TAB\_VARCHAR100 e TIPO\_TAB\_NUMBER não foram declarados na package. Eles devem ser declarados globalmente ao esquema para que variáveis com estes tipos possam ser utilizadas em comandos SQL. Para declará-los no esquema, deve-se utilizar o comando CREATE TYPE, conforme mostrado a seguir.

https://paperx-dex-assets.s3.sa-east-1.amazonaws.com/images/1679505260293-W16v0Z13lA.png

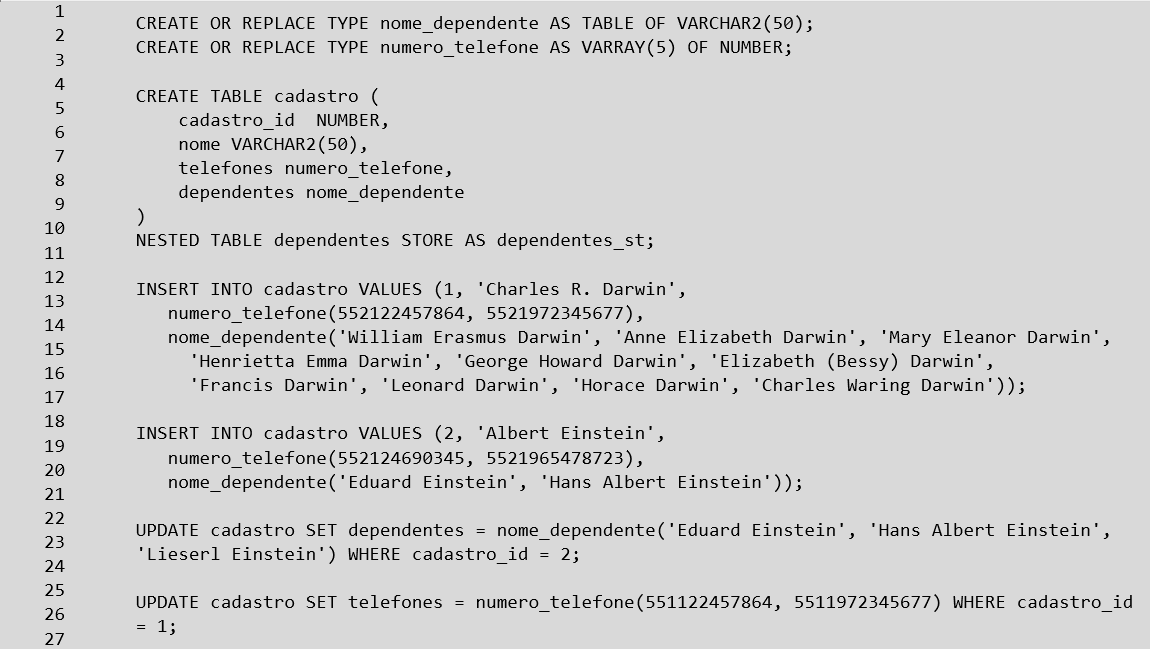
O bloco anônimo que chama a procedure e o resultado exibido após a execução são mostrados a seguir.​

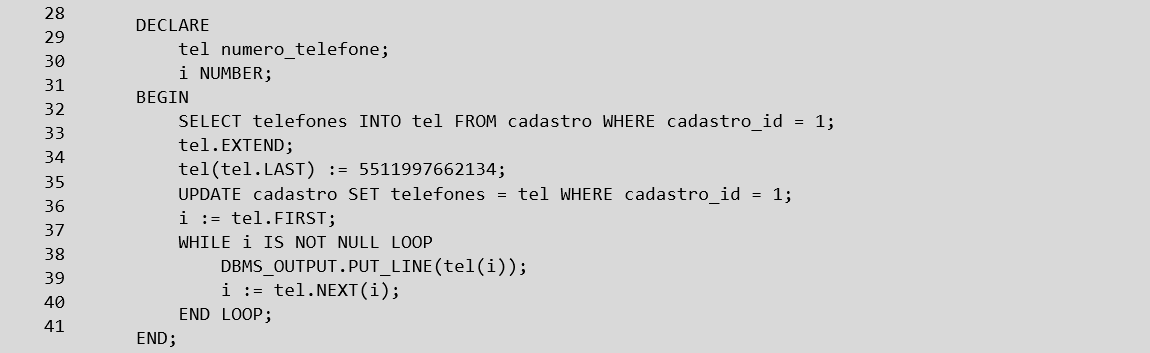


* **TABELAS E TIPOS ESTRUTURADOS**

Colunas de uma tabela podem ser definidas como tabelas aninhadas ou VARRAY. Desta forma, o SGBDR Oracle oferece suporte nativo a atributos multivalorados. A atribuição do tipo VARRAY a colunas de uma tabela é mais simples, já que este tipo tem limite em seu tamanho. Primeiro, deve-se criar um tipo através do comando CREATE TYPE. Tipos criados ficam armazenados e são globais ao esquema onde foram criados. Em seguida, atribui-se à coluna o tipo criado.

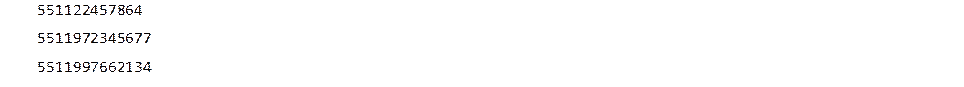
Tabelas aninhadas exigem algumas definições a mais, já que são não limitadas. Os exemplos a seguir ilustram a criação de tabelas com colunas de tipos estruturados.





Nas linhas 1 e 2 são declarados dois tipos globais. Nas linhas 5 a 10 a tabela cadastro é definida, com duas colunas (telefones e dependentes) de tipos estruturados. A linha 10 é necessária quando há colunas do tipo tabelas aninhada. Varrays são armazenados juntamente com as demais colunas da tabela, enquanto tabelas aninhadas são armazenadas separadamente. A cláusula NESTED TABLE STORE AS nomeia a estrutura de armazenamento da tabela aninhada. São necessárias tantas cláusulas quantas forem as tabelas aninhadas.

A inclusão de linhas na tabela é feita através do comando INSERT (linhas 12 a 20). Observe que os valores para as colunas telefones e dependentes devem ser inseridos utilizando-se seus respectivos constructors. A alteração de colunas segue a mesma lógica. Não é possível se alterar um elemento individual de colunas de tipos estruturados. Se isto for necessário, deve-se primeiro trazer todos os elementos para uma variável através do comando SELECT INTO. Depois de feitas as alterações, a coluna inteira deve ser atualizada. Este procedimento é mostrado no bloco anônimo (linhas 28 a 41). Após a execução do bloco anônimo, o resultado é:



### *TRANSAÇÕES E PL/SQL*

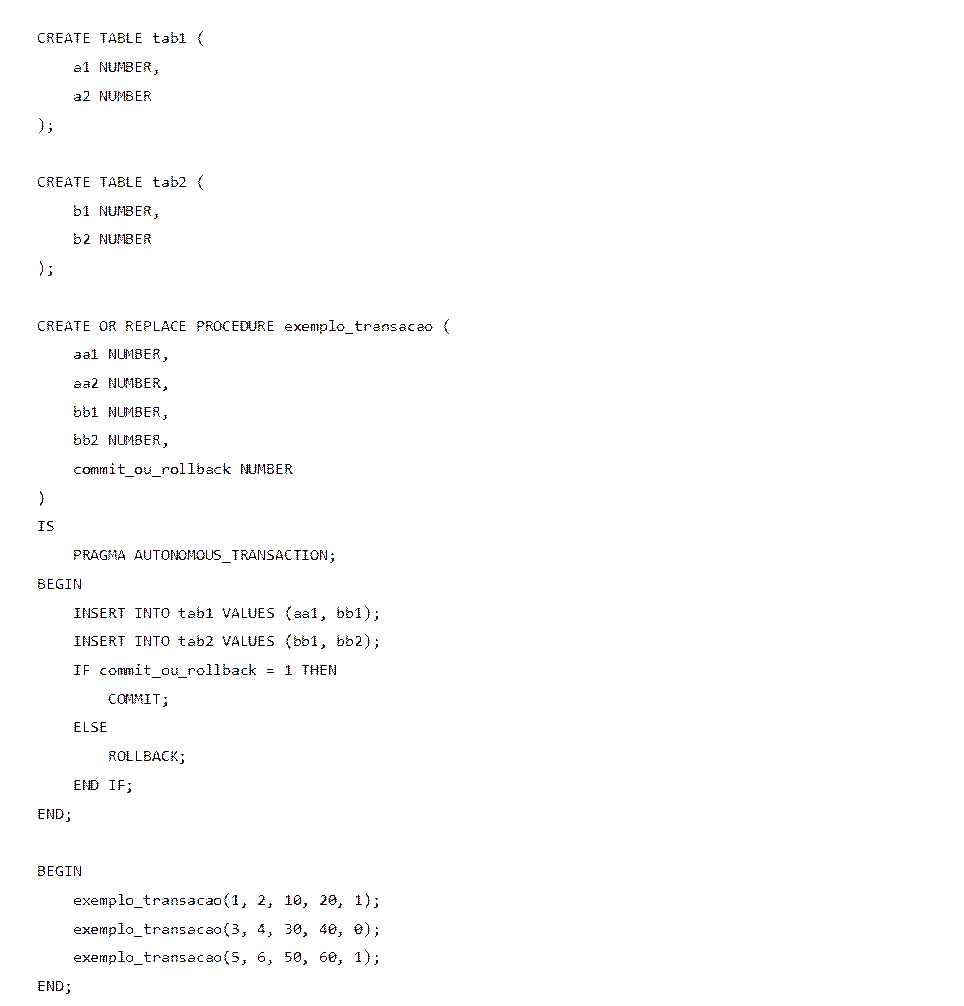
Uma transação é um conjunto de comandos SQL que formam uma unidade indivisível. Cada comando realiza parte da tarefa e é necessário que todos sejam executados para que a tarefa seja concluída com sucesso. O agrupamento de comandos em uma transação informa ao SGBDR que todos os comandos devem ser executados com sucesso ou o estado do banco de dados não é alterado. É mais ou menos como um “tudo ou nada”.

No jargão de banco de dados, uma transação deve obedecer aos critérios ACID (Atomic, Consistent, Isolated, Durable ou Atômico, Consistente, Isolado e Persistente).

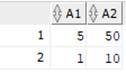
Blocos de programa PL/SQL podem ser configurados como transações autônomas. Isto significa que, mesmo que tenham sido chamados de dentro de outra transação, os comandos de controle da transação considerarão o estado do banco de dados no momento em que o bloco foi iniciado. Para se definir um bloco PL/SQL como uma transação autônoma, deve-se utilizar a diretiva PRAGMA AUTONOMOUS\_TRANSACTION. Os blocos PL/SQL que podem ser configurados como transações autônomas são blocos anônimos (apenas o bloco mais externo), funções e procedures, tanto individuais quanto em packages e triggers.

Transações autônomas devem ser utilizadas em situações onde o bloco contém diversos comandos SQL que alteram o estado do banco de dados e se deseja desfazer estas alterações em determinadas situações (em caso de erro, por exemplo).

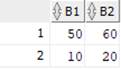
Em uma transação, todas as alterações feitas em tabelas de um banco de dados são tornadas persistentes, ou seja, passam a representar o novo estado do banco de dados para todos os usuários quando um COMMIT é executado. Enquanto um COMMIT não é executado, todas as alterações feitas podem ser revertidas através do comando ROLLBACK. O exemplo a seguir ilustra o uso de transações autônomas.



​Com as tabelas TAB1 e TAB2 inicialmente vazias, a primeira chamada à procedure insere um registro em cada e executa um COMMIT. Na segunda chamada, os registros são inseridos, mas um ROLLBACK é executado, fazendo com que o estado das tabelas retorne ao que era antes das duas inserções. A terceira chamada insere mais um registro em cada tabela. O conteúdo das duas tabelas, após a execução do bloco anônimo, é mostrado a seguir.​



TAB1



TAB2