

# Banco de Dados





# Material Teórico



*Procedures, Functions, Exceptions e Triggers*

**Responsável pelo Conteúdo:**

Prof. Ms. Alexander Gobbato Albuquerque

**Revisão Textual:**

Prof. Ms. Claudio Brites





- Introdução
- EXCEPTION
- Vantagens de Exceções PL / SQL
- Triggers



- O conceito de Stored Procedure é muito importante e grande parte do mundo corporativo utiliza esse recurso. Para que nossa abordagem seja ainda mais profissional, iremos estudar também as exceções que podemos criar em PL/SQL para tratamento de erros. Dedicaremos um tópico, ainda, aos Triggers, com os quais podemos invocar várias chamadas para executar uma determinada tarefa após o acontecimento de um evento.
- Resumindo, nesta unidade trabalharemos os seguintes tópicos de conteúdo: PL/SQL; Procedures; Exceptions; Function; Triggers.

Para obter um bom aproveitamento da unidade, vamos conferir sua estrutura:

Atividade de Sistematização: os exercícios disponibilizados são de autocorreção e tem por objetivo te ajudar a avaliar o que aprendeu na disciplina, identificando se há pontos que precise rever, ou se há necessidade, ainda, de pedir auxílio ao tutor. Além disso, as notas atribuídas aos exercícios serão parte de sua média final da disciplina.

Atividade de Aprofundamento: Ver a atividade que será disponibilizada, ela também será avaliada e a nota irá compor sua média final.

Lembramos a você da importância de realizar todas as atividades propostas dentro do prazo estabelecido para cada unidade, dessa forma, você evitará que o conteúdo se acumule e que você tenha problemas ao final do semestre.

Uma última recomendação, caso tenha problemas para acessar algum item da disciplina, ou dúvidas em relação ao conteúdo, não deixe de entrar em contato com seu professor tutor por meio do link Mensagens.

## Contextualização

Program Language SQL é a linguagem de programação da Oracle que objetiva processar informações do banco de dados.



O Stored Procedure é um subprograma que é compilado e armazenado no servidor. Ele pode ser chamado a partir de um comando SQL qualquer.

A vantagem de usar procedimentos armazenados é que eles podem encapsular rotinas de uso frequente no próprio servidor, e estarão disponíveis para todas as aplicações.

Vamos aprender?

## Introdução



Vamos relembrar que com PL/SQL podemos criar variáveis herdando o tipo e tamanho de outras variáveis ou objetos do banco de dados, tais como: tabelas; criar cursores para tratamento de resultados de queries; criar registros para armazenar resultado de cursores; tratamento de erros; e utilizar comandos de repetição e comparação.

Estudamos também anteriormente que Stored Procedure são subprogramas compilados e armazenados no servidor. Eles podem ser chamados a partir de um comando SQL qualquer.

**Program Language SQL:** é a linguagem de programação da Oracle que objetiva processar informações do banco de dados.

## Stored Procedure

Relembrando que Stored Procedure são Blocos PL/SQL armazenados no banco de dados de forma compilada, podem ou não receber parâmetros. Os parâmetros passados para os subprogramas podem ser de 3 tipos:

- IN (padrão) – passa um valor do ambiente chamador para o subprograma. Esse valor não pode ser alterado dentro do subprograma;
- OUT – passa um valor do subprograma para o ambiente chamador;
- IN OUT – passa um valor do ambiente chamador para o subprograma. Esse valor pode ser alterado dentro do subprograma e retornado com o valor atualizado para o ambiente chamador.

## Functions

São subprogramas que têm por objetivo retornar algum resultado ou retornar algum valor. Estrutura para criação de function:

```
CREATE OR REPLACE FUNCTION Nome_Funcao
  (Argumento1 modo Tipo_de_Dados,
   Argumento2 modo Tipo_de_Dados,
   Argumento3 modo Tipo_de_Dados)
  RETURN Tipo_de_Dado
```

IS ou AS

declarações

Begin

Bloco PI/SQL

Exception

End nome\_da\_função;

A instrução FUNCTION é usada para criar uma função autônoma ou uma função específica. A função armazenada – também chamada de função do usuário ou função definida pelo usuário. A função é um conjunto de instruções PL/SQL chamada pelo nome. Funções armazenadas são muito parecidas com as PROCEDURES, com exceção de que uma função retorna um valor. Funções do usuário podem ser usadas como parte de uma expressão SQL.

As funções não podem ser usadas nas seguintes opções:

Em uma cláusula de restrição CHECK de um CREATE TABLE ou ALTER TABLE; em uma cláusula DEFAULT de uma instrução CREATE TABLE ou ALTER TABLE.

Além disso, quando uma função é chamada a partir de uma consulta ou instrução DML, a função não pode ter OUT ou IN OUT como parâmetros de informações.

Exemplo:

```
CREATE or REPLACE FUNCTION balanco (numcont IN NUMBER)
  RETURN NUMBER
  IS contabanco NUMBER(11,2);
  BEGIN
    SELECT saldo_conta
      INTO contabanco
      FROM pedidos
     WHERE pedido_conta = numcont;
    RETURN(contabanco);
  END;
```

A função balanco retorna o saldo de uma conta especificada. Quando a função é executada, o argumento numcont recebe o número da conta e busca seu saldo. A função usa uma instrução SELECT para selecionar a coluna identificada pelo argumento numcont da tabela de pedidos. A função usa uma instrução RETURN para retornar esse valor para o ambiente em que ela foi chamada.

Ao especificar a sintaxe REPLACE, a função será recriada se ela já existir. Utilize essa cláusula para mudar a definição de uma função existente, sem alterar os privilégios. Se você redefinir uma função, então o Oracle Database irá recompilá-lo.

Para testarmos uma função, podemos executar a seguinte instrução:

```
SELECT <nome_função> FROM dual.
```

Utilizando como exemplo a função criada acima a sintaxe seria:

```
SELECT balanco(12345) from dual
```

## EXCEPTION



São todos os erros e imprevistos que podem ocorrer durante a execução de um bloco PL/SQL.

### EXCEPTION

```
WHEN nome_da_exceção THEN
```

```
    Relação_de_comandos
```

```
WHEN nome_da_exceção THEN
```

```
    Relação_de_comandos
```

## Vantagens de Exceções PL / SQL



Ao usarmos exceções para manipulação de erros, temos várias vantagens. Com as exceções é possível tratar os erros de forma confiável com um manipulador de exceção única:

```
BEGIN
```

```
    SELECT ...
```

```
    SELECT ...
```

```
    procedure_that_performs_select();
```

```
    ...
```

```
EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN -- catches all 'no data found' errors
```

Ao invés de verificar se existe um erro em um determinado ponto, adicionaremos um manipulador de exceção para o bloco PL/SQL. Se a exceção é sempre executada nesse bloco (ou qualquer sub-bloco), teremos certeza que será tratada.

Às vezes, o erro não é muito óbvio e não pode ser detectado tão facilmente. Com um único tratamento de exceção podemos capturar cálculos com dados incorretos, erros de divisão por zero e assim por diante. Se houver a necessidade de realizar um tratamento em um ponto específico, podemos colocar uma única instrução ou um grupo de instruções dentro de seu próprio bloco BEGIN-END.

Existem exceções pré-definidas no Oracle como, por exemplo:

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Exception	Raised when ...
ACCESS_INTO_NULL	A program attempts to assign values to the attributes of an uninitialized object.
CASE_NOT_FOUND	None of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	A program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
CURSOR_ALREADY_OPEN	A program attempts to open an already open cursor. A cursor must be closed before it can be reopened. A cursor FOR loop automatically opens the cursor to which it refers, so your program cannot open that cursor inside the loop.
DUP_VAL_ON_INDEX	A program attempts to store duplicate values in a database column that is constrained by a unique index.
INVALID_CURSOR	A program attempts a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	In a SQL statement, the conversion of a character string into a number fails because the string does not represent a valid number. (In procedural statements, VALUE_ERROR is raised.) This exception is also raised when the LIMIT-clause expression in a bulk FETCH statement does not evaluate to a positive number.
LOGIN_DENIED	A program attempts to log on to Oracle with an invalid username or password.

NO_DATA_FOUND	A SELECT INTO statement returns no rows, or your program references a deleted element in a nested table or an uninitialized element in an index-by table.
NOT_LOGGED_ON	Because this exception is used internally by some SQL functions to signal that they are finished, you should not rely on this exception being propagated if you raise it within a function that is called as part of a query.
PROGRAM_ERROR	PL/SQL has an internal problem.
ROWTYPE_MISMATCH	The host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types. For example, when an open host cursor variable is passed to a stored subprogram, the return types of the actual and formal parameters must be compatible.
SELF_IS_NULL	A program attempts to call a MEMBER method, but the instance of the object type has not been initialized. The built-in parameter SELF points to the object, and is always the first parameter passed to a MEMBER method.
STORAGE_ERROR	PL/SQL runs out of memory or memory has been corrupted.
SUBSCRIPT_BEYOND_COUNT	A program references a nested table or varray element using an index number larger than the number of elements in the collection.
SUBSCRIPT_OUTSIDE_LIMIT	A program references a nested table or varray element using an index number (-1 for example) that is outside the legal range.
SYS_INVALID_ROWID	The conversion of a character string into a universal rowid fails because the character string does not represent a valid rowid.
TIMEOUT_ON_RESOURCE	A time-out occurs while Oracle is waiting for a resource.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row.
VALUE_ERROR	An arithmetic, conversion, truncation, or size-constraint error occurs. For example, when your program selects a column value into a character variable, if the value is longer than the declared length of the variable, PL/SQL aborts the assignment and raises VALUE_ERROR. In procedural statements, VALUE_ERROR is raised if the conversion of a character string into a number fails. (In SQL statements, INVALID_NUMBER is raised).
ZERO_DIVIDE	A program attempts to divide a number by zero.

No exemplo a seguir, usaremos uma exceção já criada no Oracle:

```
set serveroutput on
Begin
  Insert into Pais values(&codigo,'&nome');
  Commit;
  Dbms_output.put_line('Comando executado com sucesso');
```

### Exception

**When Dup\_Val\_On\_Index Then**

**Dbms\_output.put\_line ('ERRO!!! Código de país já existente');**

**When Others then Dbms\_output.put\_line ('Erro ao cadastrar o país');**

end;

Também é possível criar exceções diferentes das que já foram definidas e criadas pelo sistema de banco de dados. A exceção que foi criada pelo usuário deverá ser invocada pelo comando RAISE, exemplo:

**Declare**

**Nome\_da\_exceção EXCEPTION;**

**Begin**

**Relação\_de\_comandos**

**IF ... Then**

**RAISE Nome\_da\_exceção;**

**End If;**

**Relação\_de\_comandos**

**EXCEPTION**

**WHEN nome\_da\_exceção THEN**

**Relação\_de\_comandos**

**End;**

Vejamos um exemplo prático de um tratamento de exceção criado pelo usuário:

SET SERVEROUTPUT ON

SET VERIFY OFF

declare

```
codigo cliente.cd_cliente%type;
nome cliente.nm_cliente%type;
fisica cliente.ie_fisica_juridica%type;
sexo cliente.ie_sexo%type;
est_civ cliente.est_civ%type;
Valida_campos exception;
```

Begin

```
codigo:=&codigo;
nome:='&nome';
fisica:='&fisica';
sexo:='&sexo';
```

```

est_civ:='&est_civ';

If (fisica='F' and (sexo is null or est_civ is null)) then
  Raise Valida_Campos;

Elseif (fisica= 'J' and (sexo is not null or est_civ is not null)) then
  Raise Valida_campos;

Else
  Insert into cliente (cd_cliente, nm_cliente, ie_fisica_juridica, ie_sexo, est_civ)
  values (codigo, nome, fisica, sexo, est_civ);
  dbms_output.put_line ('Cliente cadastrado com sucesso');
  COMMIT;
end if;

Exception
  When Valida_campos then
    dbms_output.put_line ('Verificar campos sexo ou estado civil preenchido incorretamente');
  when Dup_Val_On_Index then
    dbms_output.put_line (' ERRO!!! Código já cadastrado, favor cadastrar novo código
diferente de '|| codigo);
  end;

```

## Triggers



São blocos PL/SQL disparados automaticamente e implicitamente sempre que ocorrer um evento associado a uma determinada tabela (INSERT, UPDATE ou DELETE).

Um Trigger pode ser utilizado para:

- Manutenção de tabelas de auditoria;
- Manutenção de tabelas duplicadas;
- Implementação de níveis de segurança mais complexos;
- Geração de valores de colunas – por exemplo, gerar o valor total da nota fiscal a cada inclusão, exclusão ou alteração de itens da nota.

Por exemplo, podemos criar um Trigger – ou gatilho, como também é conhecido – para executar a seguinte tarefa: ao se alterar um valor da tabela produto, automaticamente grava-se o valor antigo e o novo valor em uma tabela chamada de Tmp\_Preco\_Prod.

Tabela produto:

cd_produto	vl_custo_medio
1	1.5
2	2.3
3	.65
4	3.12
5	1.5

Atualizar o vl\_custo\_medio do produto de código 4 para 2.93

cd_produto	vl_anterior	vl_novo
4	3.12	2.93

Ao executar o comando *Update produto set vl\_custo\_medio =2.93 where cd\_produto=4*, a tabela *tmp\_preco\_prod* é alimentada automaticamente.

Para se criar um trigger a sintaxe é a seguinte:

CREATE OR REPLACE TRIGGER BASICO (1)

BEFORE (2) UPDATE (3) ON AUDITORIA (4)

FOR EACH ROW

[ BLOCO PL/SQL ]

Onde (1) é o nome de seu *Trigger*, (2) é o tempo em que será disparado o evento – BEFORE OU AFTER – (3) será o comando DML que será executado no sistema – pode ser UPDATE, INSERT ou DELETE – e (4) é o nome da tabela que você quer que um trigger seja executado.

Create or Replace Trigger Verifica\_Produto

Before Update of vl\_custo\_medio on produto

For each row

Begin

Insert into Tmp\_Preco\_Prod

Values(:old.cd\_produto, :old.vl\_custo\_medio, :new.vl\_custo\_medio);

End;

## Entendendo os pseudo registros :Old e :New

Ao trabalharmos com Triggers de nível de linha, utilizamos estes dois pseudo registros, eles servem para fazer as comparações das colunas velhas (:old) com as novas (:new), são muito utilizados para fazer Update nas colunas.

Os registros especiais :NEW e :OLD armazenam temporariamente os valores do último registro manipulado pelo SGBD.

Ao inserir, é criado o registro :NEW

Ao apagar, é criado o registro :OLD

Ao atualizar, são criados os 2 registros :NEW e :OLD

DML	:OLD	:NEW
INSERT	NULO	VALORES NOVOS
DELETE	VALORES ANTIGOS	NULO
UPDATE	VALORES ANTIGOS	VALORES NOVOS

Desabilitando e Habilitando um Trigger Específico:

ALTER TRIGGER nome\_da\_trigger DISABLE;

ALTER TRIGGER nome\_da\_trigger ENABLE;

Desabilitando e Habilitando todos os Triggers de uma tabela:

ALTER TABLE nome\_tabela DISABLE ALL TRIGGERS;

ALTER TABLE nome\_tabela ENABLE ALL TRIGGERS;

Eliminando Triggers e Procedimentos:

DROP TRIGGER nome\_trigger;

DROP PROCEDURE nome\_procedure;

Inspecionando o dicionário de dados:

SELECT object\_name, object\_type FROM user\_objects WHERE object\_type = 'TRIGGER';

Este comando é executado para ver todas as triggers do Usuário.

Obtendo detalhes dos Triggers:

DESC USER\_TRIGGERS

Este comando lhe dará todas as informações objetivas dos triggers, como: tabela associada, corpo, Variável utilizada, etc.

## Material Complementar

O conteúdo sobre Function e Trigger pode ser muito mais aprofundado, vale a pena dar uma lida na documentação da Oracle sobre essas duas técnicas. Disponível em:

- [http://docs.oracle.com/cd/B19306\\_01/appdev.102/b14251/adfns\\_triggers.htm](http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/adfns_triggers.htm)

E também:

- <http://www.techonthenet.com/oracle/functions.php>

## Referências

Fanderuff, Damaris. **Dominando o Oracle 9i: modelagem e desenvolvimento.** São Paulo: Pearson Education do Brasil, 2003.

Material oficial da Oracle no curso: Oracle Database 10 G – SQL Fundamentals I.

Sobre Function: [http://docs.oracle.com/cd/B19306\\_01/server.102/b14200/statements\\_5009.htm#i2092700](http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_5009.htm#i2092700)

Sobre Triggers: [http://docs.oracle.com/cd/B19306\\_01/appdev.102/b14251/adfns\\_triggers](http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/adfns_triggers).



**Cruzeiro do Sul**  
Educacional