

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

(وما أسألكم عليه من أجر إن اجري إلا علي الله)

اسأل الله العلي العظيم أن يوفقني ويوفقكم الي ما فيه الخير لي ولكم وأرجو من كل من قرأ الكتاب ان لا يبخل عليّ بالثناء لي ولوالدي وان يقدر لي عن اي خطأ موجود في الكتاب .

عندما يبدع القلم وينطق اللسان بقول الحق وتتحرك
اليدان لعمل اجمل ما يتم عمله لينتفع به كل من في العالم
العربي اجمع فنجد **الاستاذ والتلميذ** معا يبدعان كي
ينشران علما رائعا

محاسبة

شهادات دولية

برمجة

لغة الجاوية

صيانة

تعمية بشرية

دبلومات متخصصة

ORACLE FINANCIAL

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

(وما أسألكم عليه من أجر إن اجري إلا علي الله)

اسأل الله العلي العظيم أن يوفقني ويوفقكم الي ما فيه الخير لي ولكم وأرجو من كل من قرأ الكتاب ان لا يبخل عليّ بالثناء لي ولوالدي وان يقدر لي عن اي خطأ موجود في الكتاب .

Mo7amed Reda Abd El-Rahman

ORACLE FINANCIAL CONSULTANT

ACCOUNTANT UNDER ORACLE APPLICATIONS ENVIRONMENT

INSTRUCTOR ORACLE FINANCIAL R12

INSTRUCTOR ORACLE DEVELOPER

Scientific Computing Center – Mansoura University

Mobile : 01066734381



Introduction To PL/SQL

Index

Fundamental (I)

<i>Chapter1:</i>	Introduction to PL/SQL
<i>Chapter2:</i>	Declaring PL/SQL Variables
<i>Chapter3:</i>	Writing Executable Statements
<i>Chapter4:</i>	Interacting with the Oracle Server
<i>Chapter5:</i>	Writing Control Structures
<i>Chapter6:</i>	Working with Composite Data Types
<i>Chapter7:</i>	Using Explicit Cursors
<i>Chapter8:</i>	Handling Exceptions

PL/SQL Program Units Fundamental (II)

<i>Chapter1:</i>	Creating Stored Procedure s
<i>Chapter2:</i>	Creating Stored Functions
<i>Chapter3:</i>	Creating Package

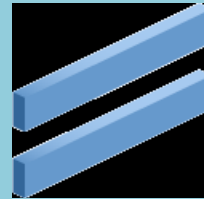
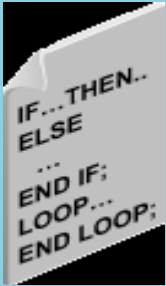
PL/SQL Fundamental (I)

CHAPTER 1

Introduction to PL/SQL

PL/SQL

Procedural Language / Structure Query Language



- ❖ PL \ SQL هي إمتداد للغة الـ SQL بإضافة مميزات لغة البرمجة الإجرائية .
- ❖ جمل التعامل والأستعلام الخاصة بالـ SQL يتم إدراجها داخل الكود الخاص بلغة الـ PL / SQL .
- ❖ سوف نقوم فى الـ PL بعمل كود وتخزينه فى الـ DATABASE ونقوم بالنداء عليه فقط وإستدعائه.
- ❖ سوف نقوم بأستخدام مايسمى بالمتغيرات (Variables) وايضاً الثوابت (Constant) وانواع اخرى كثيرة .
- ❖ سوف نستخدم الجمل الشرطية (If Statement) وجمل التكرار (Loop) وهى تستخدم لتكرار جملة معينة عدد مرات معينة .
- ❖ سوف نقوم بكتابة الكود مرة واحدة ولكننا نستطيع تنفيذة اكثر من مرة .

❖ مميزات الـ PL / SQL :

❖ Modularized تطوير البرنامج :

- ❖ حيث نستطيع وضع اكثر من بلوك داخل بعضها (Nested Blocks)
- ❖ الأستفادة من الخبرات والأكواد السابقة وذلك بجمعها فى شكل مكتبات (Libraries) يمكن الأستفادة منها من بين ادوات اوراقل المختلفة .

❖ Integration التكامل :

- ❖ يتكامل مع منتجات اوراقل مثل الـ Jdeveloper .

❖ Portability المرونة :

- ❖ حيث نستطيع كتابة الكود على اى (Operating System) او (Platform) .

❖ Exception معالجة الأخطاء :

- ❖ حيث نستطيع معالجة الأخطاء وإظهارها بالشكل الذى نريده .

PL/SQL Block Structure

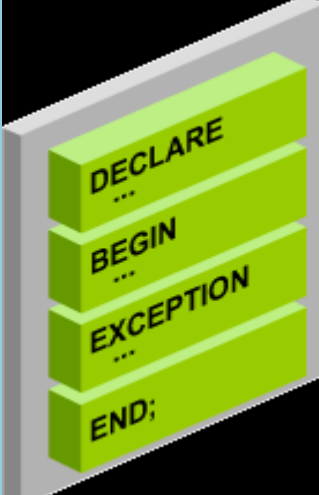
❖ يتكون كود الـ PL من عدة (Blocks) وكل (Block) يتكون من :

PL/SQL Block Structure

DECLARE	– Optional
Variables, cursors, user-defined exceptions	
BEGIN	– Mandatory
– SQL statements	
– PL/SQL statements	
EXCEPTION	– Optional
Actions to perform when errors occur	
END ;	– Mandatory


```

DECLARE
...
BEGIN
...
EXCEPTION
...
END ;
        
```

	Optional	هي منطقة التعريف والأعلان وتحتوى على مايلى : Variables – Cursors – Constants – And Other Types
	Mandatory	المنطقة التنفيذية للـ SQLStatement – Pl / SQL Statement
	Optional	معالجة الأخطاء
	Mandatory	نهاية الـ Block

❖ ملحوظه :

❖ يتكون الـ Block على الأقل من Begin و End .

❖ تنفيذ جمل وعبارات الـ PL / SQL :

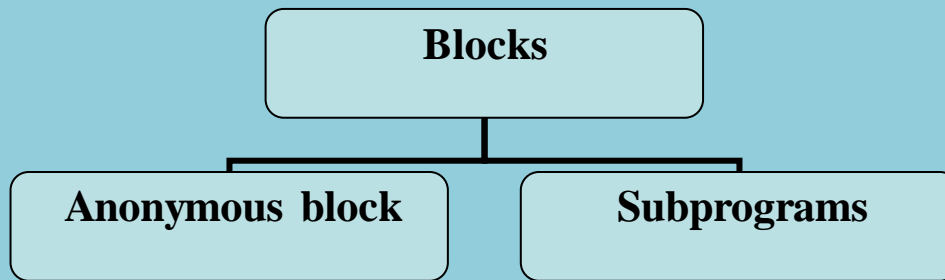
- ❖ يتم وضع علامة (;) فاصلة منقوطة في نهاية كل جملة او عبارة من جمل الـ PL / SQL .
- ❖ عند تنفيذ البلوك بدون أخطاء تظهر الجملة التالية لتوضيح تمام وصحة التنفيذ

PL / SQL procedure successfully completed .

- ❖ لاحظ عدم وجود علامة الفاصلة المنقوطة بعد نهاية كلمة Declare او Begin او Exception
- ❖ لاحظ وجود علامة الفاصلة المنقوطة (;) في نهاية كلمة End .
- ❖ يمكن كتابة أكثر من جملة او عبارة على نفس السطر والفصل بينهم بعلامة الفاصلة المنقوطة (;) ولكن لايجب ذلك لجعل قراءة البرنامج اسهل وكذلك التعديل فيه :

انواع البلوكات Block Types

- لاحظ ان انواع الـ Blocks في لغة الـ PL / SQL يمكن ان تكون منفصلة كلياً او متداخلة مع بعضها البعض (Nested Blocks) وتنقسم هذه الكتل الى قسمين :



(١) " Anonymous block " الكتلة المجهولة :

- ❖ ليس له اسم ويتم تعريفه عند نقطة التطبيق وينفذ ساعتها ويتم ارساله الى معالج اوراق لترجمتها وتنفيذها .

(٢) " Subprograms " وحدات برمجية :

- ❖ هي وحدات او Blocks تقبل معاملات (Parameters) يمكن ان تستند عليها في تنفيذ البرنامج وتنقسم هذه الوحدات الى (Procedure) و (Function) وهذه الوحدات يتم حفظها داخل الـ Database ويتم استدعائها عند الحاجة اليها .

Blocks Types

Anonymous	Procedure	Function
[DECLARE]	PROCEDURE name	FUNCTION name
	IS	RETURN datatype
		IS
BEGIN	BEGIN	BEGIN
--statements	--statements	--statements
		RETURN value;
[EXCEPTION]	[EXCEPTION]	[EXCEPTION]
END ;	END ;	END ;

❖ Differences between Anonymous Blocks and Subprograms

Anonymous Block	Subprogram
ليس له اسم	له اسم
نقوم بعمل Compile له عند كل عملية تنفيذ	نقوم بعمل Compile له مرة واحدة فقط
لايخزن في الـ Database	يخزن في الـ Database
لايستطيع تنفيذه من App اخرى	نستطيع تنفيذه من App اخرى
لايمكن ان يأخذ Parameters	ممكن ان يأخذ Parameters

▪ ما هو الفرق بين الـ Procedure والـ Function ؟

❖ الـ Procedure ممكن تعود بقيمة او اكثر او لا تعود بقيمة .

❖ الـ Function لابد وأن تعود بقيمة .

❖ مثال :

❖ استخراج اسم الموظف رقم ١٠٠

```

Declare
  V_fname varchar2 (20);
Begin
  Select First_name into V_fname
  From employees
  Where employee_id = 100 ;
End ;

```

PL/SQL procedure successfully completed.

❖ ملحوظة :

- ❖ لا بد من تهيئة البرنامج لأستخراج جمل الطباعه عن طريق كتابة الأمر التالي **Set Serveroutput On** وهذه الجملة تكتب مرة واحدة في الـ **Session** وتظل محفوظة حتى نقوم بإغلاق البرنامج .
- ❖ بعد تهيئة البرنامج نقوم بكتابة جملة الطباعه وهى **DBMS_OUTPUT.PUT_LINE** والـ **PUT_LINE** هى **Procedure** محفوظة داخل **Package** تسمى **DBMS_OUTPUT** .
- ❖ توضع جملة الطباعه داخل **Begin** وتأخذ **Parameter** واحد لذلك لانستطيع استخدام الـ (,) بداخلها بل نستخدم (||) فى حالة لو اردنا طباعة اكثر من متغير .
- ❖ فى حالة كتابة الكود بشكل سليم وبدون أخطاء تظهر الجملة التالية : **PL/SQL procedure successfully completed.**

```

Declare
  V_fname varchar2 (20);
Begin
  Select First_name into V_fname
  From employees
  Where employee_id = 100 ;
  DBMS_OUTPUT.PUT_LINE('the first name of the employee is : ' || V_fname );
End ;

```

the first name of the employee is : Steven
 PL/SQL procedure successfully completed.

CHAPTER 2

Declaring PL/SQL Variables

المتغيرات Variables

- الـ **Variable** : هو مكان في الذاكرة يقوم فيه بتخزين بيان معين او قيمه معينه تخزين مؤقت ونستطيع استخدام الـ **Variables** للتعديل والاضافة على القيم الموجودة في الـ **Database** وايضاً نستطيع استخدامها اكثر من مرة .

❖ القواعد العامة لتسمية الـ **Variables** :

- ❖ لابد وأن يبدأ بحرف
- ❖ من الممكن ان يحتوى على حروف وأرقام وعلامات
- ❖ يجب الا يزيد عن ٣٠ حرف
- ❖ الا يكون كلمة من كلمات اوراكل المحجوزة مثل (Select , Insert) او غيرهما

❖ ملحوظة :

- ❖ نقوم بتعريف الـ **Variables** في **Declarative Section** اى الجزء الخاص بـ (Declare) .
- ❖ من الممكن إعطاء الـ **Variables** قيم افتراضية في الـ **Declare** ونستطيع تبديلها في الـ **Executable Section** الجزء الخاص بـ **Begin** .

Example

DECLARE

Variable Name Datatype ;

```
V_hire          Date;
V_deptno        Number  Not Null  := 10 ;          Null  لا يحتوى على
V_location      Varchar2(13)  := 'Mansoura' ;
V_comm          Constant Number    := 1400 ;        هذه القيمة ثابتة لا تتغير
```

Example (I)

DECLARE

V_Name Varchar2(20) ;

BEGIN DBMS_OUTPUT.PUT_LINE ('My name is: '|| V_Name);

في هذه الحالة لن يطبع شيء سوى كلمة : My Name Is فقط

V_Name := 'Mo7amed Reda';

DBMS_OUTPUT.PUT_LINE('My name is: '||V_Name);

في هذه الحالة سيطبع كلمة My Name Is: Mo7amed Reda

END ;

Example (II)

```

DECLARE
V_Name  Varchar2(20) := 'Mo7amed';
BEGIN
V_Name := 'reda ';
DBMS_OUTPUT.PUT_LINE('My name is: '||V_Name);
END ;

```

في هذه الحالة الـ Variable بها قيمة في الـ Declarative Section وبها قيمة أيضاً في الـ Executable Section ولكن عندما نقوم بطباعة القيمة الموجودة بالمتغير ستطبع القيمة الموجودة الـ Executable Section لأنها الأحق في التنفيذ لذلك سيكون ناتج الطباعه My name is : reda .

أنواع المتغيرات Types of Variables

PL/SQL variables:

- المفردة Scalar
- المركبة (CH6) Composite
- المشار بها Reference
- ذات الأحجام الكبيرة (LOB) Large object

Non-PL/SQL variables:

- Host
- Bind

❖ أولاً المتغيرات الخاصة بالـ PL / SQL :

ملحوظة :

- ❖ من الأفضل تسمية الـ Variable باسم يدل على البيان الذي يحمله .
- ❖ من الأفضل عدم تسمية الـ Variable على إسم Column .
- ❖ الـ Variable الذي يكون Constant او Not Null لابد من إعطائه قيم افتراضية في الـ Declarative Section .
- ❖ نستطيع إعطاء الـ Variable قيمة افتراضية عن طريق عمل (=) Assignment Operator او كتابة كلمة Default كما يلي :

❖ V_Name Varchar2(20) := 'Mo7amed Reda' ;

❖ V_Name Varchar2(20) **Default** 'Mo7amed Reda' ;

- ❖ من الأفضل تعريف كل Variable على سطر حتى يسهل علينا عملية القراءة وأيضاً معالجة الأخطاء وسهولة الوصول اليها.
- ❖ سوف نتناول في هذا الـ Chapter النوع الأول وهو الـ **Scalar** الـ **PL/SQL Variables** وسوف نتناول النوع الأخير الخاص بالـ **Non Pl/sql Variables**.

Declaring Scalar Variables

❖ المتغيرات المفردة (Scalar Variables) :

(١) هي متغيرات تحتوي على قيم مفردة ولا يمكن تجزئها الى قيم مفردة أصغر منها فهي انواع لا يمكن ان يحتوى المتغير فيها سوى على قيمة واحد فقط **Number** او **Varchar2** او **Date** او **Boolean** وغيرهم .

DECLARE

```
V_Job          Varchar2 (9) ;
V_Count        Binary_Integer      := 0;
V_Dept         Number (9, 2)       := 0;
V_Orderdate    Date                := Sysdate+ 7;
V_Tax_Rate     Constant Number     := 8.25;
V_Valid        Boolean Not Null     := TRUE;
```

❖ الأنواع المختلفة للـ Data Types :

- **Char (length)** : متغير حرفي ثابت السعة سواء تم ملأها بالبيانات او تركت فارغة وهذا النوع مضر في المساحة لكنه اسرع في التعامل .
- **Varchar2 (length)** : متغير حرفي ذات سعة معينة لكن هذه السعة متغيرة ويتم ملأ المتغير بسعة النص فقط بحد اقصى سعة هذا المتغير فمثلاً متغير حرفي ذات سعة ٣٠ حرف ولم يوضع فيه سوى ٦ حروف يملأ بالحروف ال ٦ ويتم توفير الباقي وهذا النوع مفيد في المساحة لكنه ابطىء من النوع السابق (char) .
- **Long** : هو النوع الأساسى للبيانات النصية ذات سعة بحد اقصى ٣٢٧٦٠ بايت .
- **Long Raw** : هو مثل الـ Long لكنه لا يتم التعامل به ولا يفهمه PL/SQL .
- **number (p, s)** : هو متغير رقمي يأخذ نطاق من خانة واحدة الى ٣٨ خانة وكذلك كسر عشري من ٢٨ الى ١٢٧
 - **p** : عدد خانات الأرقام الصحيحة .
 - **s** : عدد خانات الأرقام العشرية .
- **Binary_integer** : هو النوع الرقمي الصحيح ليأخذ كسور عشرية خلال + ٢١٤٧٤٨٣
- **Pls_integer** : هو مثل النوع السابق لكنه اسرع ويأخذ مساحة اقل .
- **Boolean** : هو نوع يأخذ ثلاث قيم فقط (true , false m null) ويستخدم في حالات الشروط والمقارنات المنطقية فقط .
- **Date** : نوع المتغيرات التاريخية ويحتوى على بيانات تاريخ او وقت او زمن وهو يبدأ من ٤٧١٢ قبل الميلاد الى ٩٩٩٩ بعد الميلاد .

Declare

v_job varchar2 (15); متغير حرفي ذات سعة ١٥ حرف
v_count binary-integer := 0; متغير رقمي صحيح يأخذ قيمة ابتدائية صفر
v_total_sal number (7,2) := 3.17; متغير رقمي من سبع خانات منهم اثنان كسر عشري
v_order_date date:= sysdate +7 ; متغير زمني يبدأ في الأسبوع القادم
C-TAX-RATIO CONSTANT NUMBER (4,2) :=17.25;
 متغير ثابت رقمي مكون من اربع خانات منهم خانتان كسر عشري ويأخذ قيمة افتراضية ١٧.٢٥
V_Flag Boolean Not Null :=True ;
 متغير منطقي "Boolean" لا يأخذ قيمة فارغة Null ويأخذ قيمة افتراضية true .

%TYPE Attribute

❖ هي **Data type** ولكن الميزة بها انها تأخذ نفس الـ **Data type** والـ **Length** الخاص بالـ **Column** .

DECLARE

Variable Name Table Name . Column Name %Type ;
V_Name Employees.Last_Name%Type ;
V_Balance Number(7,2) ;
V_Min_Bal V_Balance%TYPE := 1000;

ملحوظة :

❖ هذا النوع افضل من الـ **Basic Scalar** حيث انه عند عمل اي تغيير او تعديل في الجدول في الـ **Length** مثلاً فإن الـ **Variable** يتغير تلقائياً بتغيير الجدول .

❖ القيم التي تعود بها **Boolean** هي (**true , false , null**) .

❖ **إستخراج إسم ومرتب الموظف الذي يحمل رقم ١٠٠ ؟**

DECLARE

```

V_Name      Employees.Last_Name%Type ;
V_Sal       Number ;
V_Id        Employees.Employee_id%Type := 100 ;
BEGIN
Select Last_Name , Salary into V_Name , V_Sal
From Employees
Where Employee_Id = V_Id ;
DBMS_OUTPUT.PUT_LINE( V_Name || ' and His Salary is : ' ||V_Sal);
END ;

```

Bind Variables

ملحوظة :

- ❖ هذا النوع من المتغيرات يتم إنشاؤه داخل الـ Session وليس داخل الـ Block وتسمى Host Variable .
- ❖ يوضع قبلها كلمة Variable وتستخدم أيضاً في الـ Sql والـ Pl/Sql .
- ❖ عند كتابة Bind Variable لا يوضع بعدة (;) Semi Colon يكون شكلة كالتالى :

Variable Variable Name Datatype

- ❖ عند النداء عليه وإستدعائه لابد ان نضع قبله (:) Colon كما يلى :Variable Name
- ❖ نستطيع إستخدام جملة الطباعة العادية مع الـ Bind Variable .

❖ إستخراج مرتب الموظف رقم ١٧٨ ؟

```
Variable V_Sal Number
BEGIN
  Select Salary into :V_Sal
  From Employees
  Where Employee_Id = 178 ;
END ;
/
Print V_Sal
```

ملحوظة :

❖ يمكن استخدام الـ Bind Variable فى الـ SQL .

مثال :

- ❖ الـ Bind Variable المسمى V_Sal المذكور فى المثال السابق يحتوى على مرتب الموظف رقم ١٧٨ ونفترض اننا نريد الأستعلام عن الموظفين الذين يحصلون على هذا المرتب ؟

```
Select Last_Name
From Employees
Where Salary = :V_Sal ;
```

- ❖ هناك جملة لاتعمل الا مع الـ Bind Variable وهى تستخدم فى الطباعة اتوماتيكياً وتسمى **Set Autoprint On** .

Prompt for Substitution Variables

مثال :

❖ قم بإستخراج مرتب الموظف الذى سأعطيك رقمة ايا كان مستخدماً الـ Bind Variable للمرتب؟

Set Verify Off

Variable V_Sal Number

Accept Empno Prompt ' Please Enter a Valid Employee ID '

Set Autoprint On

DECLARE

V_Id Number(6) := & Empno ;

BEGIN

Select Salary into :V_Sal

From Employees

Where Employee_Id = V_Id ;

END ;

CHAPTER 3

Writing Executable Statements

Lexical Units in a PL/SQL Block

ملحوظة :

- ❖ عند التعامل مع الأحرف والتواريخ لابد من وضعها بين Single Quotation .
- ❖ نستطيع كتابة الكود على أكثر من سطر .
- ❖ نستطيع عمل Comment (تعليق) للكود بطريقتين كما يلي :
- (١) لعمل Comment لسطر واحد نستخدم -- .
- (٢) لعمل Comment لأكثر من سطر نستخدم /* ونقفله بـ */ .
- ❖ من الأفضل ان نقوم بكتابة الـ comment على الأكواد لشرح وظيفة كل كود وحتى اذا جاء مبرمج اخر ليكمل المشروع يستطيع ان يفهم الأكواد بشكل سريع .

SQL Functions in PL/SQL

- ❖ نستطيع استخدام جميع الـ Functions الموجودة بالـ SQL داخل الـ PL
- ماعدا الـ Decode & Group Functions .

DECLARE

V_Lname Varchar2(20) := Initcap(' KING') ;

V_Name Varchar2(20) ;

BEGIN

Select First_Name into V_Name

From Employees

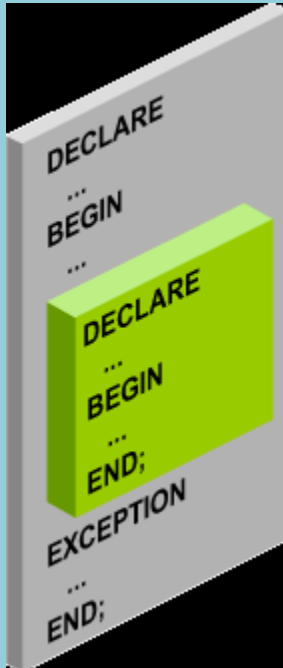
Where Last_Name Like V_Lname And Salary = 24000 ;

END;

Nested Blocks

ملحوظة :

❖ البلوك الداخلى يرى البلوك الخارجى ويؤثر فيه والعكس غير صحيح اى ان البلوك الخارجى لا يرى البلوك الداخلى .



Example

```

DECLARE
V_Father_Name Varchar2(20) := 'reda';
V_Date_of_Birth Date := '04-Apr-1965' ;
BEGIN
DECLARE
V_Child_Name Varchar2(20) := 'mohamed' ;
V_Date_of_Birth Date := '04-Dec-1985';
BEGIN
DBMS_OUTPUT.PUT_LINE( V_Father_Name ); =====>>>  reda
DBMS_OUTPUT.PUT_LINE( V_Date_of_Birth ); =====>>>  04-Dec-1985
DBMS_OUTPUT.PUT_LINE( V_Child_Name ); =====>>>  mohamed
END;
DBMS_OUTPUT.PUT_LINE( V_Date_of_Birth ); =====>>>  04-Apr-1965
END;

```


ملحوظة:

- ❖ نستطيع عمل Qualifier للبلوك وتسميته بأى اسم نريده حتى نتمكن من تحديد البلوك المراد النداء عليه ولكن لابد من وضعه بين << >> .
- ❖ لو هناك متغير فى البلوك الكبير له نفس الاسم فى البلوك الصغير وقمنا بالنداء عليه من داخل البلوك الصغير فالأولوية للمتغير الموجود فى البلوك الصغير .

Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations

Same in SQL

Qualify an Identifier**Example (I)**

<< scc >>

DECLARE**V_Father_Name Varchar2(20) := 'reda';****V_Date_of_Birth Date := '04-Apr-1965';****BEGIN****DECLARE****V_Child_Name Varchar2(20) := 'mohamed' ;****V_Date_of_Birth Date := '04-Dec-1985';****BEGIN****DBMS_OUTPUT.PUT_LINE(V_Father_Name); =====>>> reda****DBMS_OUTPUT.PUT_LINE(scc. V_Date_of_Birth); =====>>> 04-Apr-1965****DBMS_OUTPUT.PUT_LINE(V_Child_Name); =====>>> Mohamed****DBMS_OUTPUT.PUT_LINE(V_Date_of_Birth); =====>>> 04-Dec-1985****END;****END;**

Example (II)

<< scc >>

DECLARE**V_Sal Number(7,2) := 60000 ;****V_Comm Number(7,2) := V_Sal * 0.20;****V_Message Varchar2(255) := ' eligible for commission' ;****BEGIN****DECLARE****V_Sal Number(7,2) := 50000 ;****V_Comm Number(7,2) := 0 ;****V_Total Number(7,2) := V_Sal + V_Comm ;****BEGIN****V_Message := 'CLERK not' || V_Message ;****DBMS_OUTPUT.PUT_LINE (V_Message);** (1)**scc.V_Comm := V_Sal * 0.30;****DBMS_OUTPUT.PUT_LINE (scc.V_Comm);** (2)**END;****V_Message := 'SALESMAN' || v_message;****DBMS_OUTPUT.PUT_LINE (V_Message);** (3)**END;**

- 1) eligible for commission .
- 2) 15000
- 3) SALESMAN eligible for commission .

CHAPTER 4
Interacting with the Oracle Server

Using PL/SQL to Manipulate Data

INSERT	UPDATE	DELETE	MERGE
---------------	---------------	---------------	--------------

Inserting Data

```
BEGIN
    INSERT into Departments
    Values (280 , 'Reda_Dept', 100 , 1700);
END;
```

Updating Data

```
DECLARE
    Sal_Increase Employees.Salary%Type := 800 ;
BEGIN
    UPDATE Employees
    Set Salary = Salary + Sal_Increase
    Where Job_id = 'ST_CLERK';
END;
```

Deleting Data

```
DECLARE
    Deptno Employees.Department_id%Type: = 280;
BEGIN
    DELETE From Employees
    Where Department_id = Deptno;
END;
```

SQL Cursor

IMPLICIT	EXPLICIT
هو من النوع الضمني اي ان اوراكل هي التي قامت بعمله ويعمل تلقائياً بمجرد النداء عليه	المبرمج هو الذى يقوم بعمله وسوف نتناوله بالتفصيل فى Chapter (7)

ملحوظة:

❖ الـ Cursor يعتبر مساحة فى الـرام تقوم اوراكل بتجهيزها اتوماتيكياً مع كل جملة SQL وتضع بها البيانات القادمة من الـ SQL .

SQL Cursor Attributes for Implicit Cursors

SQL%FOUND	لو وجد بيانات يقوم بإسترجاع True ولو لم يجد يقوم بإسترجاع False
SQL%NOTFOUND	لو لم يجد بيانات يقوم بإسترجاع True ولو وجد بيانات يقوم بإسترجاع False
SQL%ROWCOUNT	يقوم بإسترجاع عدد الصفوف التى تأثرت بالعملية فى الميمورى

❖ قم بمسح الموظف رقم 206 ومن ثم اطبع عدد الصفوف التى تأثرت بالعملية ان وجد هذا الموظف؟

DECLARE

V_Rows_Deleted Varchar2(30) ;

V_Empno Employees.Employee_id%Type := 206;

BEGIN

Delete From Employees

Where Employee_id = V_Empno ;

V_Rows_Deleted := (SQL%Rowcount || ' row deleted.');

DBMS_OUTPUT.PUT_LINE (V_Rows_Deleted);

END;

❖ قم بإعطاء الموظف رقم ١٠٠ انوجد المرتب الأتي ٣٠٠٠٠ ؟

DECLARE

V_Name Varchar2 (20);

BEGIN

Select Last_Name into V_Name

From Employees

Where Employee_Id = 100;

IF SQL%Found Then

Update Employees

Set Salary = ٣0000

Where Employee_Id = 100;

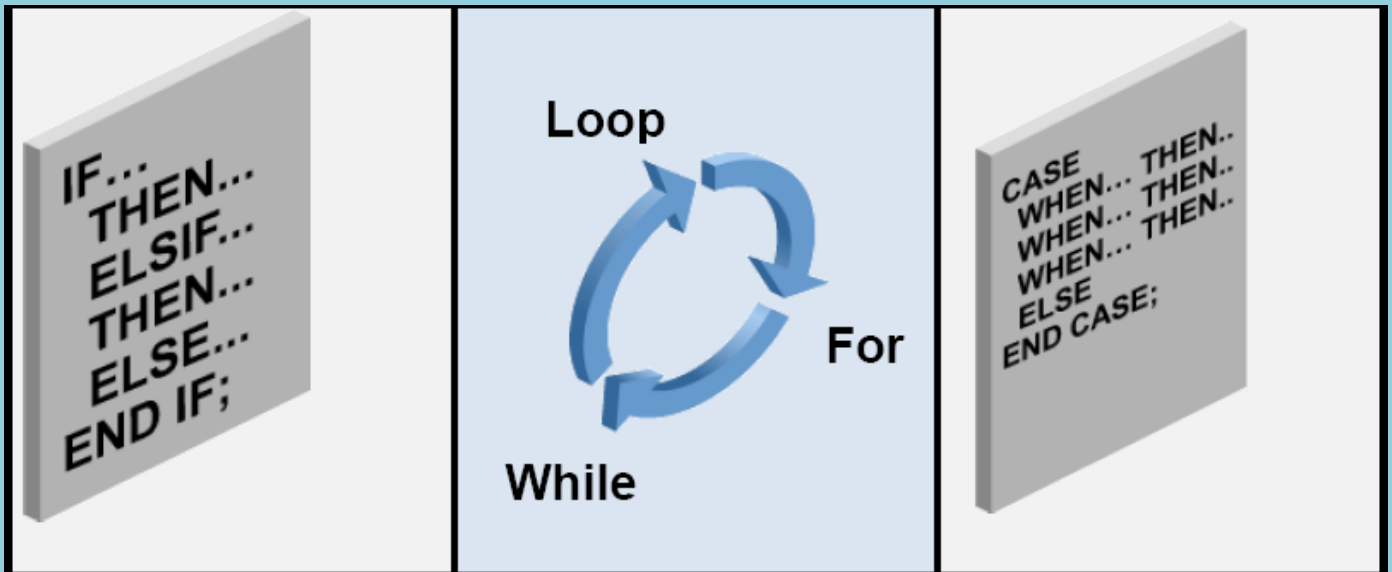
End IF;

END;

ملحوظة:

- ❖ لقد قمنا بعمل جملة Select حتى نعرف هل يوجد موظف بهذا الرقم ام لا فلو وجد سيأتي بإسمه وان لم يجد فلم يأتي بشيء .
- ❖ اما الـ SQL%FOUND فتتنظر الى الـ SQL هل وجدت بيانات ام لا فإن وجدت فسوف تقوم بتنفيذ عملية التعديل وان لم تجد فلن تفعل شيء .

CHAPTER 5
Writing Control Structures



IF Statement

DECLARE

V_Myage Number:=30;

BEGIN

IF V_Myage < 11 Then

DBMS_OUTPUT.PUT_LINE (' I am a child ');

ELSE

DBMS_OUTPUT.PUT_LINE (' I am not a child ');

End IF;

END;

Example

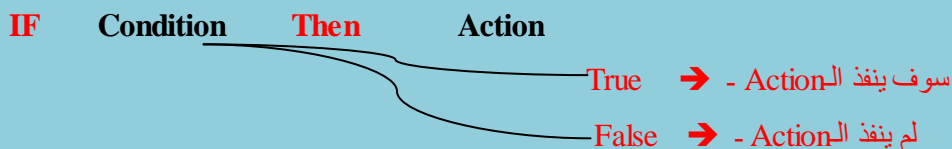
```

DECLARE
    V_Myage Number:=30;
BEGIN
    IF V_Myage < 11 Then
        DBMS_OUTPUT.PUT_LINE (' I am a child ');
    ELSIF V_Myage < 20 Then
        DBMS_OUTPUT.PUT_LINE (' I am young ');
    ELSIF V_Myage < 30 Then
        DBMS_OUTPUT.PUT_LINE (' I am in my twenties');
    ELSIF V_Myage < 40 Then
        DBMS_OUTPUT.PUT_LINE (' I am in my thirties');
    ELSE
        DBMS_OUTPUT.PUT_LINE (' I am always young ');
    End IF;
END;

```

ملحوظة :

(١) لو تحقق الشرط الأول يتم تنفيذ الأمر الذي بعد Then وإذا لم يتحقق الشرط الأول يتم اللجوء الى الشرط الثانى واذ تحقق يتم تنفيذ الأمر الذى بعد then وإذا كانت الشروط كلها null يتم تنفيذ الأمر الذى بعد Else .



(٢) نستطيع وضع أكثر من If فى نفس الجملة .

(٣) كلمة Else تأتى مرة واحدة فى نهاية جملة IF .

(٤) فى نهاية الشروط نضع كلمة ; End if ومنشأش نضع (;) فى نهاية IF .

CASE Expressions

DECLARE

```
V_Grade Char(1) := Upper('&Grade');
```

```
Appraisal Varchar2 (20);
```

BEGIN

```
Appraisal := CASE
```

```
    When V_Grade = 'A' Then 'Excellent'
```

```
    When V_Grade In ('B','C') Then 'Good'
```

```
Else 'No such grade'
```

```
End;
```

```
DBMS_OUTPUT.PUT_LINE ('Grade: ' || V_Grade || ' Appraisal ' || Appraisal);
```

```
END;
```

Handling Nulls

ملحوظة :

- المقارنات البسيطة لو فيها Null الناتج يصبح Null .
- لو وضعنا Not مع Null الناتج يصبح Null .
- مع الأقوى هو الـ False والأضعف هو الـ True .
- مع الأقوى هو الـ True والأضعف هو الـ False .

LOOP Statements

Basic Loop	While Loop	For Loop
------------	------------	----------

Basic Loop

Syntax

Loop

Statement;

Exit [When Condition];

End Loop;

❖ قم بإضافة ٣ مواقع باستخدام الـ **Basic Loop** ؟

DECLARE

V_Country_id Locations.Country_id%Type:= 'CA';

V_Loc_id Locations.Location_id%Type;

V_Counter Number := 1;

V_New_City Locations.City%Type := 'Montreal';

BEGIN

Select Max(Location_id) Into V_Loc_id From Locations

Where Country_id = V_Country_id;

LOOP

Insert Into Locations (Location_id, City, Country_id)

Values ((V_Loc_id + V_Counter), V_New_City, V_Country_id);

V_Counter := V_Counter + 1;

EXIT WHEN V_Counter > 3;

END LOOP;

END;

ملحوظة :

- ❖ هل سنظل الـ Loop تعمل الى مالانهاية؟ بالطبع لا علشان كده هتلاقوني عرفت متغير هو بمثابة عداد الذى يمثل عدد الدورات التى تقوم بها الـ Loop وفى كل دورة نضيف واحد على هذا العداد حتى يصل الى الحد المطلوب ويخرج اذا فالـ Basic Loop لا بد من عمل متغير له يكون بمثابة عداد حتى ننهي عملية الـ Loop عنده .
- ❖ اذا لم نحدد Exit When Condition شرط للخروج من الـ Loop فسوف تظل الـ Loop تعمل الى مالانهاية وسوف تتوقف الـ Database عن العمل بسبب التهنيج ويظهر هذا الـ Numeric or Value Error :Number <= Error . Precision too Large
- ❖ فرضنا اننا قمنا بعمل الشرط التالى <1 Exit When Counter فان الـ Basic Loop سوف تنفذ مرة واحدة على الأقل رغم ان الشرط يخالف الـ Loop وذلك لأن الـ Statement تنفذ اولاً قبل المرور على الـ Condition .

While Loop**Syntax****While Condition Loop****Statement;****End Loop;**❖ **قم بإضافة ٣ مواقع باستخدام الـ WHILE Loop ؟****DECLARE**

```

V_Country_id    Locations.Country_id%Type:= 'CA';
V_Loc_id        Locations.Location_id%Type;
V_Counter       Number := 1;
V_New_City      Locations.City%Type := 'Montreal';

```

BEGIN**Select Max(Location_id) Into V_Loc_id****From Locations****Where Country_id = V_Country_id;****WHILE v_counter <= 3 LOOP****Insert Into Locations (Location_id, City, Country_id)****Values ((V_Loc_id + V_Counter), V_New_City, V_Country_id);****V_Counter := V_Counter + 1;****END LOOP;****END;**

ملحوظة :

❖ الـ While Loop تقوم بعمل Check اولاً على الـ Condition قبل تنفيذ الـ Statement فإذا تحقق تقوم بعمل الـ Loop وإذا لم يتحقق لن تعمل الـ Loop بعكس الـ Basic Loop وكلما تحقق الـ Condition سوف تظل الـ Loop تعمل.

For Loop**Syntax**

For Counter in [Reverse]lower-Bound .. Upper-Bound Loop

Statement;

End Loop;

ملحوظة :

❖ الـ For Loop لا تحتاج الى عمل عداد لها بل تقوم اوراقك بعمله اتوماتيكياً ولكننا نقوم بتسميته فقط .

❖ فلو مثلاً اردنا طباعة الأرقام من ١ ١٠ فاننا نقوم بكتابة الجملة بهذا الشكل

For I in 1 .. 10 Loop DBMS_OUTPUT.PUT_LINE (I);

ولو اردنا طباعة الأرقام بشكل عكسي من الرقم ١٠ حتى الرقم ١ فاننا نقوم بكتابة الجملة بالشكل التالي : For I in 1 .. 10

Reverse Loop DBMS_OUTPUT.PUT_LINE (I);

لانستطيع استخدام Null في بداية العداد واقل قيمه يجب ان نبدأ بها هي (1) .

نستطيع استخدام اكثر من Loop داخل بعضهما في نفس الجملة .

❖ **قم بإضافة ٣ مواقع باستخدام While Loop ؟**

DECLARE

V_Country_id Locations.Country_id%Type:= 'CA';

V_Loc_id Locations.Location_id%Type;

V_New_City Locations.City%Type := 'Montreal';

BEGIN

Select Max(Location_id) Into V_Loc_id From Locations

Where Country_id = V_Country_id;

FOR i in 1..3 LOOP

Insert Into Locations (Location_id, City, Country_id)

Values ((V_Loc_id + i), V_New_City, V_Country_id);

END LOOP;

END;

CHAPTER 6

Working with Composite

Composite Data Types

❖ هناك نوعان وهما :

- 1) PL/SQL Records
- 2) PL/SQL Collections :-
 - a. Index By tables.
 - b. Nested table.
 - c. Varray.

PL/SQL Records

Records

ملحوظة :

- ❖ الـ Record يحمل بداخله اكثر من قيمة على عكس الـ Scalar .
- ❖ يحتوى الـ Record على Fields (3GL) 3 Generation Language ويتم انشاء هذه الخانات بنفس الطريقة التي تستخدمها لغات الـ C والـ C++
- ❖ المبرمج يقوم بتعريف الـ DataType الخاصة بهذه الـ Fields وتسميتها ايضا ثم عمل الـ Variable وإعطاء الـ Record له
- ❖ اى متغير لابد من تحديد نوع بيانات له اى ان الـ Variable لابد من تحديد Datatype له ولكن الأختلاف هذه المرة ان الـ Datatype هى عبارة عن مجموعة من الـ Datatypes وليست واحدة ولذلك يجب علينا انشاء الـ Datatype اولا وبعد ذلك يتم إعطاؤها للمتغير .

Syntax Create Datatype and Variable .

Type Type_Name is Record (Field1 Datatype , Field2 Datatype ,) ;
Variable ;

ملحوظة :

❖ نرى اننا قد قمنا بعمل Datatype وتحتوى على Two Fields وكأننا قمنا بعمل متغيران داخل الـ Datatype هذه وقد قمنا بإعطاء هذه الـ Datatype اسماً.

Field1 (data type)	Field2 (data type)	Field3 (data type)
Employee_id Number(6)	Last_Name Varchar2(25)	Job_Id Varchar2(10)
100	King	AD_PRES

❖ **إستخرج إسم ووظيفة الموظف رقم ١٠٠ باستخدام الـ Record ؟**

DECLARE

Type Emp_Rec is Record (R_Name Varchar2(20) , R_Job Varchar2(20));

V_Rec Emp_Rec ;

BEGIN

Select Last_Name , Job_id into V_Rec

From Employees

Where Employee_id = 100 ;

DBMS_OUTPUT.PUT_LINE (V_Rec.R_Name || ' ' || V_Rec.R_Job) ;

END;

ملحوظة :

- ❖ فى حالة الطباعة لابد من كتابى اسم المتغير دوت اسم الفيلد (Variable.Field name) .
- ❖ عند الأستعلام ووضع القيم داخل المتغير يجب ان يراعى الترتيب الموجود فى الـ Record .

%ROWTYPE Attribute**Syntax**

Variable_Name Table_Name%Rowtype ;

❖ نستخدم الـ %Rowtype لو اردنا استخراج جميع أعمدة الجدول فلو استخدمنا الطريقة العادية وهي الـ Scalar فإننا سنحتاج الى تعريف متغيرات بعدد أعمدة الجدول ولو قمنا بعمل Record فإننا سوف نحتاج الى تعريف Fields بعدد أعمدة الجدول أيضاً وهذا سبب إستخدام خاصية %Rowtype .

❖ **إستخرج جميع بيانات الموظف رقم ١٠٠ وقم بطباعة إسمه ومرتبة؟**

Exempl (I)

DECLARE

V_Rec Employees%Rowtype ;

BEGIN

Select * into V_Rec

From Employees

Where Employee_Id = 100 ;

DBMS_OUTPUT.PUT_LINE (V_Rec.Last_Name || ' ' || V_Rec.Salary);

END;

Exempl (II)

DECLARE

Type T_Rec is Record (R_Sal Number, R_Minsal Number Default 1000,

R_Hire_Date Employees.Hire_Date%Type,

R_Rec1 Employees%Rowtype);

V_Myrec T_Rec ;

BEGIN

V_Myrec.R_Sal := V_Myrec.R_Minsal + 500;

V_Myrec.R_Hire_Date := Sysdate;

Select * into V_Myrec.R_Rec1

From Employees

Where Employee_Id = 100;

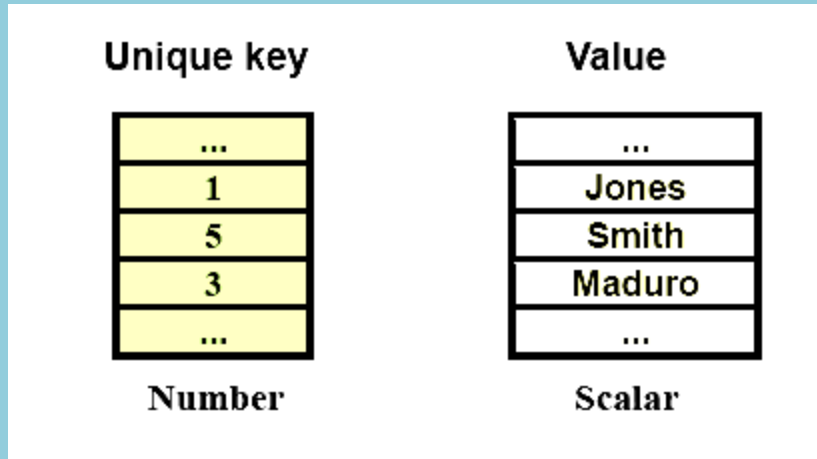
DBMS_OUTPUT.PUT_LINE (V_Myrec.R_rec1.Last_Name || ' ' ||To_Char(V_Myrec.R_Hire_Date)|| ' ' ||To_Char(V_Myrec.R_Sal));

END;

PL/SQL collections

INDEX BY Tables

❖ هي نوع من أنواع الـ Datatype التي نقوم بعملها وهو يتكون من عمودين العمود الأول سيكون Primary Key ويفضل ان تكون نوع البيانات به رقمية والعمود الثاني اما ان يكون Scalar او Composite .



Syntax

TYPE Type_Name IS Table Of (Column_type | Table.Column%Type | Table%Rowtype)

INDEX BY (Datatype);

Variable_Name Type_Name;

ملحوظة :

Using INDEX BY Table Methods

خصائص الـ Index :-

- Exist : - وهي لمعرفة هل هذه الخلية (صف) موجود في الـ index.
 - Count : - ياتي بعدد الخلايا داخل index التي تحتوى على قيم.
 - First : - ياتي برقم اول خلية في الـ Index بها قيمة.
 - Last : - ياتي برقم اخر خلية في الـ index بها قيمة.
 - Prior (n) : - يرجع بعدد الخلايا في الـ Index قبل n.
 - Next (n) : - يرجع بعدد الخلايا في الـ Index بعد n.
 - Trim (n) : - يمسح عدد خلايا (n) من نهاية الـ index
 - Delete (m,n) : - يمسح الخلايا من النطاق m الى n في الـ index
 - Delete (n) : - يمسح الخلية (n) من الـ index
- ويمكن استعمال هذه الـ Methods كالآتي:

Index_NAME.Methods_Name [(parameter)];

Examples(I)**DECLARE**

```

Type T_Name_Table Is Table Of Employees.Last_name%Type

```

```

Index By Number;

```

```

V_Name T_Name_Table;

```

BEGIN

```

V_Name (1) := 'Mo7amed Reda';

```

```

IF V_Name.Exists(1) Then

```

```

    Insert into Employees (LAST_Name , Employee_Id , Hire_Date , Job_Id , Email )

```

```

    Values(V_Name(1) , 800 , '1-JAN-2010' , 'SA_REP' , 'Dev_Habib@Yahoo.com' );

```

END;**Examples(II)****DECLARE**

```

TYPE Emp_Table_Type Is Table Of Employees%Rowtype

```

```

INDEX BY PLS_INTEGER;

```

```

V_Emp Emp_Table_Type;

```

```

V_Max_Count Number := 104;

```

BEGIN

```

FOR i IN 100 .. V_Max_Count LOOP

```

```

    Select * Into V_Emp(i)

```

```

    From Employees

```

```

    Where Employee_Id = i;

```

```

END LOOP;

```

```

FOR i IN V_Emp.FIRST .. V_Emp.LAST LOOP

```

```

    DBMS_OUTPUT.PUT_LINE ( V_Emp(i).Last_Name);

```

```

END LOOP;

```

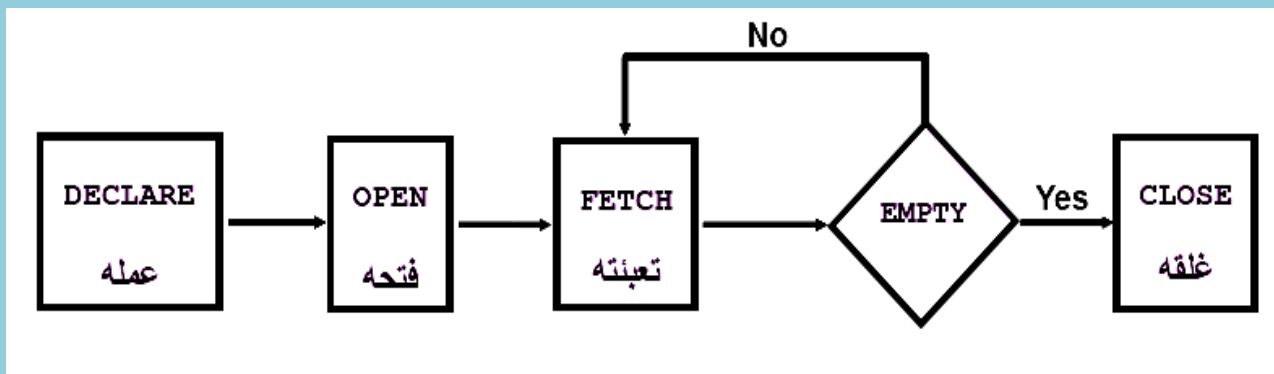
END;

CHAPTER 7

Using Explicit Cursors

❖ ماهو الـ Cursor وماهى فائدته ؟

- الـ Explicit Cursors هو ليس من النوع الضمنى اى ان المبرمج هو الذى يقوم بعمله وفتحه وتعبئته وغلقة وهو عبارة عن مكان فى الميمورى تقوم فيه بعمل عمليه معينه اى اننا نقوم بتخزين كل البيانات التى نحتاجها من الـ Database فى الـ Cursor بدلاً من ان نذهب فى كل مرة الى الـ Database ونستدعى البيانات وهذا طبعاً اسرع بكثير ويوفر وقتاً كبيراً اذا فائدته الكبرى فى السرعة فى الأستعلام عن البيانات فبدلاً من ان نضع القيمه من الـ Database فى المتغير فنأتى بكل البيانات التى نريدها مرة واحده من الـ Cursor الذى بطبيعته مخزن فى الميمورى ثم نقلها الى المتغير .



Syntax

DECLARE

Cursor Cursor_Name is Select_Statement;

BEGIN

Open Cursor_Name ;

Fetch Cursor_Name Into Variables ;

Close Cursor_Name ;

END;

❖ استخراج أرقام وأسماء موظفي الإدارة رقم ٣٠؟

DECLARE

```

Cursor Emp_Cursor is Select Employee_Id , Last_Name
From Employees
  Department_Id = 30 ;
V_Id Employees.Employee_Id%Type ;
V_Name Employees.Last_Name%Type ;

```

BEGIN

```

Open Emp_Cursor ;
  LOOP
Fetch Emp_Cursor into V_Id , V_Name ;
  Exit When Emp_Cursor %Notfound ;
      DBMS_OUTPUT.PUT_LINE ( V_Id || ' and his name is :' || V_Name );
  End Loop ;
Close Emp_Cursor ;

```

END;❖ قم بحل المثال السابق ولكن باستخدام متغيرات من النوع **Composite** وبالتحديد باستخدام الـ **Record** بدلاً من المتغيرات الـ **Scalar** ؟**DECLARE**

```

Cursor Emp_Cursor is Select Employee_Id , Last_Name
From Employees
  Where Department_Id = 30 ;
V_Rec Emp_Cursor %Rowtype ;

```

BEGIN

```

Open Emp_Cursor ;
  LOOP
Fetch Emp_Cursor into V_Rec;
  Exit When Emp_Cursor %Notfound ;
      DBMS_OUTPUT.PUT_LINE ( V_Rec.Employee_Id || ' ' ||V_Rec.Last_Name );
  End Loop ;
Close Emp_Cursor ;

```

END;

ملحوظة:

- لقد قمنا بعمل الـ Cursor ووضعنا به كل البيانات التي نريدها من الـ Database ثم قمنا بإستدعائها من الـ Cursor الى المتغير وبسرعه لأن الأثنان فى الميمورى سواء الـ Variable او الـ Cursor .

Explicit Cursor Attributes

<u>Attribute</u>	<u>Type</u>	<u>Description</u>
%ISOPEN	Boolean	تعود بقيمه True لو مفتوح والعكس.
%NOTFOUND	Boolean	تعود بقيمه True لو الـ Cursor فاضى والعكس.
%FOUND	Boolean	تعود بقيمه True لو الـ Cursor به بيانات والعكس.
%ROWCOUNT	Number	تعود بعدد الصفوف التي تم سحبها من الـ Cursor.

Cursor FOR Loops

- استخدام For Loop مع الـ Cursor من الأشياء الشائعة الأستخدام حيث اننا لانحتاج الى تعريف متغير بالنسبة للـ For Loop وأيضاً لانحتاج لفتح ولا لتعبئة ولا لغلاق الـ Cursor اذا فلن نحتاج الالعمل الـ Cursor فقط وتقوم اوراكل اوتوماتيكياً بفتحة وتعبئته وغلقة .

❖ استخراج ارقام واسماء موظفى الإدارة رقن ٣٠ بإستخدام الـ For Loop ؟

DECLARE

Cursor Emp_Cursor is Select Employee_Id , Last_Name

From Employees

Where Department_Id = 30 ;

BEGIN

For Rec in Emp_Cursor Loop

DBMS_OUTPUT.PUT_LINE (Rec.Employee_Id || ' ' ||Rec.Last_Name);

End Loop ;

END;

Syntax %ISOPEN Attribute

```

BEGIN
IF NOT c_emp_cursor%ISOPEN THEN
OPEN c_emp_cursor;
END IF;
LOOP.....

```

❖ مثال على استخدام الخاصية **%ROWCOUNT** و **%NOTFOUND**.

```

ECLARE

```

```

    Cursor C_Emp Is Select Employee_Id, last_name
    From Employees;
    V_Rec C_Emp%Rowtype;

```

```

BEGIN

```

```

    Open C_Emp ;
        LOOP
    Fetch C_Emp INTO V_Rec ;
    Exit When C_Emp %Rowcount > 10 or C_Emp %Notfound;
    DBMS_OUTPUT.PUT_LINE ( V_Rec.Employee_Id ||' '||V_Rec.Last_Name);
        End Loop;
    Close C_Emp ;

```

```

END ;

```

ملحوظة :

- في هذه الحالة اي شرط يتحقق اولاً تخرج الـ Loop سواء الـ Cursor اصبح فارغاً او عدد الصفوف التي تم سحبها اكبر من ١٠ .

❖ مثال على استخدام الـ Cursor FOR Loops Using Subqueries .

```

BEGIN
  For R in ( Select Employee_Id, Last_Name From Employees
            Where Department_Id =30 ) LOOP
    DBMS_OUTPUT.PUT_LINE ( R.Employee_Id ||' '||R.Last_Name);
  End Loop;
END;
```

ملحوظة :

- كل ما فعلناه اننا بدلاً من عمل الـ Cursor ووضع جملة الاستعلام داخله قمنا بعمل Subquery مكان الـ Cursor في الـ for Loop فقط وهي تقوم بنفس الوظيفة ولكن ليس بنفس السرعة .

❖ مثال على استخدام الـ Cursors with Subqueries .

```

DECLARE
  Cursor Sub is Select D.Department_Id , D.Department_Name,E.Staff
                From Departments D, (Select Department_Id,Count (*) as Staff
                From Employees
                Group by Department_Id) E
                Where D.Department_Id = E.Department_Id And E.Staff >= 3;
BEGIN .....
```

ملحوظة :

- ناتج هذا الـ Cursor هو ارقام واسماء الإدارات التي بها موظفين اكبر من او يساوى 3 موظفين .

Cursors with Parameters

- الفائدة منه اننا نستطيع تغيير القيم الناتجة عن الـ Cursor عن طريق إدخال Parameters لتغيير النواتج .

Example

DECLARE

```
Cursor C_Emp (P_Id Number) is Select Employee_Id, Last_Name
From Employees
Where Department_Id = P_Id ;
```

```
V_Id Number ;
```

```
V_Name Varchar2(20) ;
```

BEGIN

```
Open C_Emp( 10 ) ;
```

```
Loop
```

```
Fetch C_Emp Into V_Id , V_Name ;
```

```
Exit When C_Emp%Notfound ;
```

```
DBMS_OUTPUT.PUT_LINE (V_Id || ' ' || V_Name);
```

```
End Loop ;
```

```
Close C_Emp ;
```

```
DBMS_OUTPUT.PUT_LINE ('-----');
```

```
Open C_Emp( 30 ) ;
```

```
Loop
```

```
Fetch C_Emp Into V_Id , V_Name ;
```

```
Exit When C_Emp%Notfound ;
```

```
DBMS_OUTPUT.PUT_LINE (V_Id || ' ' || V_Name);
```

```
End Loop ;
```

```
Close C_Emp ;
```

END;

ملحوظة :

- بهذا الشكل نكون قد قمنا بتغيير البيانات اكثر من مرة عن طريق فتح الـ Cursor مرة اخرى ببيانات اخرى ولكن ذلك يتطلب منا غلق الـ Cursor القديم والا سيظهر لنا Error انه مفتوح بالفعل وللتغلب على هذه المشكلة نستخدم الخاصية Cursor_Name%Isopen .

CHAPTER 8 Handling Exceptions

- كما ذكرنا من قبل ان هذه المرحلة هي مرحلة إختيارية نستطيع الاستغناء عنها ولكنها مرحلة هامة وفيها يتم التعامل مع الأخطاء التي يمكن ان تظهر للمستخدم النهائي ومعالجتها وإظهارها له بصور يستطيع فهمها وقرأتها حيث ان الأخطاء التي تظهر من اوراكل لا يستطيع المستخدم النهائي فهمها لذلك نظهرها له بصورة بسيطة.

Exception Types

Predefined Oracle Server	Implicitly Raised	متعارف عليه وضمني
Non-predefined Oracle Server	Implicitly Raised	غير متعارف عليه وضمني
User-defined	Explicitly Raised	المبرمج هو الذي يقوم بعمله والتحكم به

Predefined Oracle Server

- هناك حوالي ٢١ Error متعارف عليهم من قبل اوراكل وهم المشهورين ويكون لديهم إسم لل Error وايضاً رقم ونص للرسالة .

Example

DECLARE

V_Name Varchar2(20);

BEGIN

Select First_Name into V_Name

From Employees

Where Last_Name Like 'King' ;

DBMS_OUTPUT.PUT_LINE ('V_Name');

EXCEPTION

When Too_Many_Rows Then

DBMS_OUTPUT.PUT_LINE ('Query Retrieved Multiple Rows');

END;

ملحوظة:

Syntax Exception

EXCEPTION

When Exception1 [Or Exception2] Then Statement ;

[When Then Statement ;]

[When Others

- لو وجد اكثر من موظف يسمى King فإنه سوف يظهر Error ان هناك اكثر من شخص ولكن مع كتابة ال- Exception فستظهر الرسالة التلى كتبناها وهذا ال- Error من المشاهير من ال- ٢١
- لو قمنا بكتابة هذه الجملة

Select Last_Name From employees Where salary = 2542184518 ;

فإنه سوف يظهر لنا هذا ال- Error <== No Data Found لأنه لا يوجد موظف يأخذ هذا المرتب

DECLARE

V_Name Varchar2(20);

V_Sal Number;

BEGIN

Select Last_Name into V_Name

From employees

Where salary = 2542184518;

DBMS_OUTPUT.PUT_LINE ('V_Name');

DBMS_OUTPUT.PUT_LINE ('V_Sal');

EXCEPTION

When No_Data_Found Then

DBMS_OUTPUT.PUT_LINE ('لا يوجد موظف يأخذ هذا المرتب');

When Others Then

DBMS_OUTPUT.PUT_LINE (SQLCODE || " || SQLERRM);

END;

ملحوظة:

- بهذه الصورة سوف يظهر انه لا يوجد موظف بهذا المرتب ولن يقوم بطباعة ال- V_sal ولا ال- V_name لأنهما جاءا بعد جملة Select التى تحتوى على ال- Error اما لو وضعنا جملة الطباعه المرتب والاسم اعلى ال- Select فإنه سوف تقوم بطباعة Null .

SQLCODE	تأتى برقم ال- Error
SQLERRM	تأتى بنص رسالة ال- Error

وسوف نقوم الآن بعرض الـ **Errors** المتعارف عليها والمشهورة وكما قلنا من قبل انها ٢١ **Error** متعارف عليهم وهم كالتالى :

Oracle Exception Name	Oracle Error	Explanation
DUP_VAL_ON_INDEX	ORA-00001	You attempted to create a duplicate value in a field restricted by a unique index.
TIMEOUT_ON_RESOURCE	ORA-00051	A resource timed out, took too long.
TRANSACTION_BACKED_OUT	ORA-00061	The remote portion of a transaction has rolled back.
INVALID_CURSOR	ORA-01001	The cursor does not yet exist. The cursor must be OPENed before any FETCH cursor or CLOSE cursor operation.
NOT_LOGGED_ON	ORA-01012	You are not logged on.
LOGIN_DENIED	ORA-01017	Invalid username/password.
NO_DATA_FOUND	ORA-01403	No data was returned
TOO_MANY_ROWS	ORA-01422	You tried to execute a SELECT INTO statement and more than one row was returned.
ZERO_DIVIDE	ORA-01476	Divide by zero error.
INVALID_NUMBER	ORA-01722	Converting a string to a number was unsuccessful.
STORAGE_ERROR	ORA-06500	Out of memory.
PROGRAM_ERROR	ORA-06501	Generic "Contact Oracle support" message.
VALUE_ERROR	ORA-06502	You tried to perform an operation and there was a error on a conversion, truncation, or invalid constraining of numeric or character data.
ROWTYPE_MISMATCH	ORA-06504	
CURSOR_ALREADY_OPEN	ORA-06511	The cursor is already open.
ACCESS_INTO_NULL	ORA-06530	
COLLECTION_IS_NULL	ORA-06531	
SELF_IS_NULL		Your program attempts to call a MEMBER method on a null instance. That is, the built-in parameter SELF (which is always the first parameter passed to a MEMBER method) is null.
SUBSCRIPT_BEYOND_COUNT		Your program references a nested table or varray element using an index number larger than the number of elements in the collection

Non predefined Oracle Server

- فى هذه الحالة الـ Error لا يمتلك هوية اى لا يمتلك اسم ولديه بالطبع رقم ولديه رساله ولكن المشكله اننا لانستطيع النداء عليه لأن ليس له اسم لذلك نقوم بعمل متغير من النوع Exception اى انه يحمل بداخله Error ونضع الـ Error بداخله عن طريق مايسمى بالـ Paragma وهى التى تربط وتضع الـ Error داخل الـ Variable
- مالذى يحدث اذا حاولنا اضافة قيمة Null فى عمود Not Null ؟
- بالطبع سيظهر Error وهو رقمة -01400 ورسالته تقول اننا لانستطيع وضع قيمة Null داخل عمود Not Null لكن المستخدم النهائى لو ظهرت له هذه الرساله وهذا الرقم لن يفهم شيئاً لذلك سوف نظهر له رساله يستطيع فهمها .

Example

DECLARE

V_Error Exception;

Pragma Exception_init (V_Error , -01400);

BEGIN

Insert into Departments (Department_Id , Department_Name)

Values (Null , 'Mo7amed Reda');

EXCEPTION

When V_Error Then

DBMS_OUTPUT.PUT_LINE ('الابد من وضع قيمة داخل العمود');

END;

User Defined Exceptions

وهما نوعان :-

Raise	Raise_Application_Error
--------------	--------------------------------

- هذه المرة لاتوجد مشكلة لأنه يظهر لنا **PL/SQL Procedural Successfully** ولكن الكود لم يطبق فعلياً ولم ينجح فى تنفيذ المطلوب ولكنه بدلاً من ان يظهر لنا Error يوضح ان العملية لم تتم يظهر انها نجحت فعن طريق استخدام Raise نستطيع اذا لم ينفذ الكود المطلوب منه فإننا نجبره على اظهار رساله توضح ان العملية لم تتم .

Raise (١)**Example**

```

DECLARE
  V_Error Exception ;
BEGIN
  Update Employees
    Set Salary = 12000
    Where Employee_Id = 123456 ;
  IF SQL%Notfound Then Raise V_Error ;
  End IF;
  DBMS_OUTPUT.PUT_LINE (SQL%Rowcount);
  Commit;
EXCEPTION
  When V_Error Then
  DBMS_OUTPUT.PUT_LINE ('لا يوجد موظف بهذا الرقم');
END;

```

- لو لم نستخدم Raise لكان ظهر لنا رساله انه تمت عملية التحديث بنجاح وهذا طبعاً لم يحدث لعدم وجود موظف بهذا الرقم .
- `SQL%ROWCOUNT <=` تقوم بالقراءة عدد الصفوف التي تأثرت بالعملية من خلال الميمورى فإذا كنا كتبنا `Commit` قبل `DBMS` فإنه لن يظهر ولن يطبع شيئاً وذلك لأن `Commit` تقوم بعمل `Save` للعمليات من الميمورى الى الـ `Database` وتمحى ما فى الميمورى .

Raise_Application_Error (٢)

- في هذه المرة هناك Error او لا لايهم لكن المبرمج من خلاله يستطيع ظهور Error صريح مثل Error اوراكل له رقم ورسالة ومن الممكن ان نقوم بأستخدام ارقام خاصة باوراكل ولقد قامت اوراكل بوض مجموعه من الأرقام نستطيع استخدامها وهي تبدأ من ٢٠٠٠٠ حتى ٢٠٩٩٩ ونستخدم Raise_Application_Error حتى نقوم بهذه العملية ونستطيع ان نفعل ذلك بمكانين وهما اما منطقة ال- Exception Section او منطقة ال- Executable Section .

- Executable Section

```

BEGIN
    Delete From Employees
        Where Employee_Id = 123456;
    IF SQL%Found Then
        DBMS_OUTPUT.PUT_LINE (SQL%Rowcount );
    Elsif SQL%Notfound Then
        Raise_Application_Error (-20001, 'الرقم بهذا موظف يوجد لا');
    End IF;
END;
```

- Exception Section

```

DECLARE
    V_Name Varchar2(20);
BEGIN
    Select Last_Name into V_Name
        From Employees
        Where Employee_Id = 123456;
EXCEPTION
    When No_Data_Found Then
        Raise_Application_Error (-20002, 'الرقم بهذا موظف يوجد لا');
END;
```



PLSQL
Program Units

CHAPTER 1

Creating Stored Procedures

- هو نوع من انواع البلوكات ويعتبار احد افراد عائلة الـ Subprogram وتكوينه شبيه بالبلوك الـ Anonymous مع اختلاف بسيط مع الـ Procedure لانستخدم الـ Declare ونستخدم بدلاً منه Is او As حتى لو لم نضطر الى تعريف Variables والباقي كما هو وميزة الـ Procedure اننا نقوم بكتابة الكود مرة واحدة ونستطيع تنفيذة والنداء عليه اكثر من مرة لأنه مخزن في الـ Database ولكنه ليس من الضروري ان يعود هذا الأجراء بقيمة معينه ... اى انه ممكن ان يعود بقيمة او لا ...

❖ **مالفرق بين الـ (Subprogram (Procedure , Function) والـ Anonymous Block ؟**

<u>Anonymous</u>	<u>Subprogram</u>
ليس له اسم	لها اسم
نقوم بعمل Compile كل مره يُنفذ .	نقوم بعمل Compile له مره واحده فقط .
لا يُخزن في الـ Database	يُخزن في الـ Database
لا نستطيع تنفيذة من App اخري	نستطيع تنفيذة من App اخري
لا يأخذ Parameters	ممكن ان يأخذ Parameters

Syntax

Create [or Replace] Procedure Procedure_Name

[(Parameter1 [Mode1] Datatype1, Parameter2 [Mode2] Datatype2, . . .)]

Is | As

Variables;

BEGIN

END [Procedure_Name];

ملحوظة :

- نقوم باستخدام كلمة للتعديل على الـ Procedure .
- الـ Procedure تستطيع ان تأخذ Parameters او لا .
- عند عمل الـ Procedure او الـ Function فإن الـ Compiler يقوم بعمل Check عليهم فإذا كان الـ Syntax ليس به أخطاء فإنه تظهر رسالة تقول Procedure Created وإذا كان هناك Error فسوف تظهر هذه الرسالة
Warning: Function Created with Compilation Error .
- لمعرفة الخطأ الموجود نقوم بكتابة Show Errors .

❖ **قم بعمل Procedure تقوم بإضافة ادارة جديدة؟**

```

CREATE PROCEDURE Add_Dept
IS
    V_Dept_Id Number := 280 ;
    V_Dept_Name Varchar2(20) := 'Outbox';
BEGIN
    Insert into Departments ( Department_Id,Department_Name)
    Values (V_Dept_Id , V_Dept_Name ); DBMS_OUTPUT.PUT_LINE ('Inserted '||
SQL%Rowcount ||' row ');
END;
```

بهذا الشكل قد منا بعمل Procedure ولكنها لم تنفذ بعد وان اردنا ان ننفذها فإن ذلك يتم بطريقتين وهما :

- Execute Add_Dept ;
- BEGIN
Add_Dept ;
END;

ملحوظة :

- يفضل استخدام الطريقة الثانية لأن الطريقة الأولى لا تعمل في الفورمز مستقبلاً .

Parameters

- هي شبيهه بالمتغيرات ولكنه تمرر القيمة فقط داخلها اما المتغير فيقوم بحمل القيمة ونقوم بتعريف الـ Parameter بعد اسم الـ Procedure ولا يأخذ Length .

P_Name Varchar2(20)	x	P_Name Varchar2	√
---------------------	---	-----------------	---

➤ والـ Parameter له انواع وهي :-

IN	وهو الـ Default ونقوم بتمرير قيم من خارج الـ Procedure الي داخلها حتي تقوم بتنفيذ العمليه اي ان الـ Pro تنتظر مني قيمه حتي تنفذ العمليه .
OUT	تقوم الـ Procedure بتنفيذ العمليه المطلوبه وتممر الناتج من خلال الـ Parameter الي المتغير اي ان الـ Pro هي التي ستعطيني قيمه ولذلك لا بد من عمل متغير في هذه الحاله
IN OUT	نقوم بإعطاء الـ Procedure قيمه ثم تقوم هي بعمل عمليه علي هذه القيمه ومن ثم تخرجها لنا بشكل اخر.

ملحوظة :

- **Parameters Formal** : - هو الـ Parameter فى حالة الأنشاء مثل :

```
Create Procedure Add_Name (P_Name Varchar2 )
```

- **Actual Parameters** : - هو الـ Parameter فى حالة إعطاء قيمة له مثل :

```
BEGIN
  Add_Name (' Mo7amed Reda');
END;
```

Using IN Parameters: Example

❖ قوم بعمل Procedure نقوم بإعطائها رقم الموظف ونسبة الحوافز وتقوم هى بالتعديل فى مرتبه بهذه النسبة ؟

```
CREATE OR REPLACE PROCEDURE Raise_Salary
  ( P_Id IN Employees.Employee_Id%Type, P_Percent IN Number)
IS
BEGIN
  Update Employees
  Set Salary = Salary + (Salary * P_Percent)
  Where Employee_Id = P_Id;
END Raise_Salary;
```

ولتفيذها:

```
BEGIN
  Raise_Salary ( 100 , .4 );
END;
```

Using OUT Parameters: Example

❖ قم بعمل Procedure نقوم بإعطائها رقم الموظف وهى تعطينا اسمه ومرتبته ؟

```
CREATE OR REPLACE PROCEDURE Query_Emp
  (P_Id IN Number ,P_Name OUT Varchar2, P_Sal OUT Number)
IS
BEGIN
  Select Last_Name, Salary Into P_Name, P_Sal
  From Employees
  Where Employee_Id = P_Id;
END Query_Emp;
```



```

DECLARE
    V_Name Varchar2(20);
    V_Sal Number;
BEGIN
    Query_Emp ( 171 , V_Name , V_Sal );
    DBMS_OUTPUT.PUT_LINE ( V_Name || ' ' || V_Sal );
END;

```

طريقة أخرى للتنفيذ :-

```

VARIABLE V_Name Varchar2(25)
VARIABLE V_Sal Number
EXECUTE Query_Emp (171, :V_Name , :V_Sal) ;
/
PRINT V_Name , V_Sal

```

ملحوظة:

- لا بد من عمل Variable عندما نقوم بالنداء على ال Procedure بها Parameter من النوع Out .

Using IN OUT Parameters: Example

❖ سوف أعطيك رقم موبايل واريد استخراج بهذا الشكل 010)0984-455 ؟

```

CREATE OR REPLACE PROCEDURE Format_Phone
    (P_Phone_No IN OUT Varchar2)
IS
BEGIN
    P_Phone_No := '(' || Substr (P_Phone_No,1,3) ||
    ')' || Substr (P_Phone_No,4,3) ||
    '-' || Substr (P_Phone_No,7);
END ;

```

```

DECLARE
    V_Phone Varchar2(20) := '01009844556';
BEGIN
    Format_Phone (V_Phone );
    DBMS_OUTPUT.PUT_LINE (V_Phone );
END;
```

ملحوظة :

- لابد من عمل Variable عندما نقوم بالنداء على الـ Procedure بها Parameter من النوع In Out

Syntax for Passing Parameters

- نستطيع النداء على الـ Parameters بأكثر من شكل :-

Positiona	(الترتيب) نقوم بوضع قيم الـ Parameter بنفس الترتيب الموجود في الـ Procedure
Named	(الاسم) نقوم بكتابة إم الـ Parameter وبجواراة القيم الخاصة به : P_Name => 'Mo7amed Reda'
Combination	(الإم والترتيب) نقوم بكتابة القيم بالطريقتين السابقتين معاً

❖ قم بإنشاء Sequence تبدأ من ١٠٠٠ وتزيد بمعدل ١ ؟

```
Create Sequence Reda_Seq Start With 1000 Increment By 1 ;
```

❖ قم بعمل Procedure نقوم بإعطائها رقم الموظف وهي تعطينا إسمه ومرتبه ثم قوم بالنداء عليها بالثلاث طرق

السابقة ؟

```

CREATE OR REPLACE PROCEDURE Add_Dept
    ( P_Name Varchar2, P_Loc IN Number )
IS
BEGIN
    Insert Into Departments ( Department_Id, Department_Name,
    Location_Id)
    Values (Reda_Seq.Nextval , P_Name, P_Loc);
END Add_Dept;
```

• Passing by Positional Notation

Execute Add_Dept ('Education' , 2300);

• Passing by Named Notation

Execute Add_Dept (P_Loc => 2400, P_Name => 'Training');

• Passing by Combination Notation

Execute Add_Dept ('Advertising' , P_Loc => 2500);

Invoking Procedures

- نستطيع النداء على الـ Parameters من خلال anonymous blocks وتعرضنا لأمتلة كثيرة في هذا الصدد وايضاً من خلال Procedure اخرى .

CREATE OR REPLACE PROCEDURE Process_Employees

IS

Cursor Emp_Cursor is Select Employee_Id

From Employees;

BEGIN

For Rec in Emp_Cursor Loop

Raise_Salary Rec.Employee_Id , .3);

End Loop;

Commit;

END Process_Employees;

Handled Exceptions

- على إفتراض اننا قمنا بعمل Procedure تسمى Add_Dept وقمنا بعمل Exception لها من الأخطاء التي يمكن ان تقع بها ثم قمنا بعمل Procedure اخرى ووظيفتها هي النداء على Add_Dept .

❖ **ماذا يحدث لو حدث خطأ في الـ Add_Dept ولم نكن قد قمنا بمعالجته ؟**

- اذا كنا قد قمنا بمعالجته فلن يحدث شيئاً اما اذا كان هناك Error ولم يعالج فسوف يظهر لنا Error ويوقف كل شيء .

Removing Procedures**Syntax**

- **DROP Procedure Procedure_Name ;**

Example

- **DROP Procedure Raise_Salary;**

Viewing Procedures in the Data Dictionary

- لرؤية ومعرفة الكود الذى قد قمت بكتابته داخل الـ Procedure التى قمت بعملها

Select *

From User_Source

Where Name = ' Add_Dept '

And Type = 'PROCEDURE'

Order by Line;

- لرؤية ومعرفة اسماء الـ Procedure التى قمت بعملها

Select Object_Name

From User_Objects

Where Object_Type = 'PROCEDURE';

CHAPTER 2

Creating Stored Functions

- هو نوع من انواع البلوكات وهو يعتبر احد افراد عائلة الـ Subprogram وتكوينه يكون شبيه بالبلوك العادى الـ Anonymous مع اختلاف بسيط لاتستخدم Declare مع الـ Function وتستخدم بدلاً منها Is او As ولا بد من كتابة Is او As حتى لو لم نضطر الـ تعريف Variables اما الباقي كما هو وميزة الـ Function اننا نقوم بكتابة الكود مرة واحدة والنداء عليه وتنفيذة اكثر من مرة لأنها مخزنه في الـ Database ولا بد ان تعود الـ Function بقيمة .

❖ **مالفرق بين الـ Subprogram (Procedure , Function) والـ Anonymous Block ؟**

<u>Anonymous</u>	<u>Subprogram</u>
ليس له اسم	لها اسم
نقوم بعمل Compile كل مره يُنفذ .	نقوم بعمل Compile له مره واحده فقط .
لا يُخزن في الـ Database	يُخزن في الـ Database
لا نستطيع تنفيذة من App اخري	نستطيع تنفيذة من App اخري
لا يأخذ Parameters	ممكن ان يأخذ Parameters

Syntax

```
CREATE [ OR REPLACE ] FUNCTION Function_Name [(Parameter1 [mode1] Datatype1, .....)]
RETURN Datatype
IS|AS
    [Variables ; ...]
BEGIN
    RETURN expression;
END [function_name];
```

ملحوظة :

- الفرق بين الـ Procedure والـ Function في الـ Return Datatype اى اننا نقوم بتهيئة الـ Function ان الناتج المراد إستخراجة سيكون من الذى قمت بتحديدة ونستقبله من خلال Return Expression الذى فيه نحدد المتغير الذى نريد ان نضع فيه القيمة .
- لانستطيع مع الـ Function استخدام Parameter الا من النوع In فقط .
- الـ Parameter كما عرفناها سابقاً وعند تعريفها فإننا لانعطيها Length وايضاً الـ Return عند تحديد الـ Datatype لا نحدد لها Length ايضاً .

❖ قم بعمل Function نعطيهها رقم الموظف وهي تعطينا مرتب ؟

```
CREATE OR REPLACE FUNCTION Get_Sal ( P_Id Number )
RETURN Number
IS
V_Sal Number:= 0;
BEGIN
Select Salary into V_Sal
From Employees
Where Employee_id = P_Id;
RETURN V_Sal;
END Get_Sal;
```

هناك اربع طرق لتنفيذ هذه الـ Function ؟

1	Execute DBMS_OUTPUT.PUT_LINE (Get_Sal(100));
2	Variable V_Sal Execute :V_Sal := Get_Sal(100);
3	DECLARE V_Sal Number ; BEGIN V_Sal := Get_Sal(100) ; DBMS_OUTPUT.PUT_LINE (v_Sal); END;
4	Select Job_Id , Get_Sal (100) From Employees;

ملحوظة :

- لا بد من عمل متغير عند النداء على الـ Function لأسترجاع القيمة فيه .

❖ قم بعمل Function نعطيهها المرتب وهي تعطينا صافي الضريبة الخاص به ؟

```
CREATE OR REPLACE FUNCTION Tax (P_Value IN Number)
RETURN Number
IS
BEGIN
RETURN (P_Value * 0.08);
END Tax;
```

```
Select Employee_Id, Last_Name, Salary, Tax(Salary)
```

```
From Employees
```

```
Where Department_Id = 100;
```

❖ قم بعمل Function نعطيها رقم الموظف وتعود ب True لو مرتبه اكبر من متوسط مرتبات ادارته و False لو العكس ؟

```
CREATE FUNCTION Check_Sal ( P_Id Number)
```

```
RETURN Boolean
```

```
IS
```

```
V_Dept_Id Employees.Department_Id%Type;
```

```
V_Empno Employees.Employee_Id%Type ;
```

```
V_Sal Employees.Salary%Type;
```

```
V_Avg_Sal Employees.Salary%Type;
```

```
BEGIN
```

```
    Select Salary,Department_Id into V_Sal,V_Dept_Id
```

```
    From Employees
```

```
    Where Employee_Id = P_Id;
```

```
    Select Avg(Salary) into V_Avg_Sal
```

```
    From Employees
```

```
    Where Department_Id=V_Dept_Id;
```

```
IF V_Sal > V_Avg_Sal THEN
```

```
    RETURN True;
```

```
ELSE
```

```
    RETURN False;
```

```
End IF;
```

```
EXCEPTION
```

```
When NO_DATA_FOUND Then
```

```
    RETURN Null;
```

```
END;
```

```

DECLARE
    V_Check Boolean ;
BEGIN
    V_Check := Check_Sal ( 205) ;
    IF V_Check = True Then
        DBMS_OUTPUT.PUT_LINE ('T');
    ELSIF V_Check = False Then
        DBMS_OUTPUT.PUT_LINE ('F');
    End IF ;
END;

```

ملحوظة :

- لرؤية الخطأ نكتب Show Errors .

Removing Functions

Syntax

```

DROP Function Function_Name ;

```

Example

```

DROP Function Get_Sal ;

```

Viewing Functions in the Data Dictionary

❖ لرؤية زمعرفة الكود الذى قمت بكتابته داخل الـ **Function** التى قمت بعملها .

Select Text

```

From User_Source
Where Name = ' Get_Sal '
And Type = 'FUNCTION' ;

```

❖ لرؤية ومعرفة أسماء الـ **Functions** التى قمت بعملها .

Select Object_Name

```

From User_Objects
Where Object_Type ='FUNCTION';

```


CHAPTER ٣ CREATING PACKAGES

الأهداف:

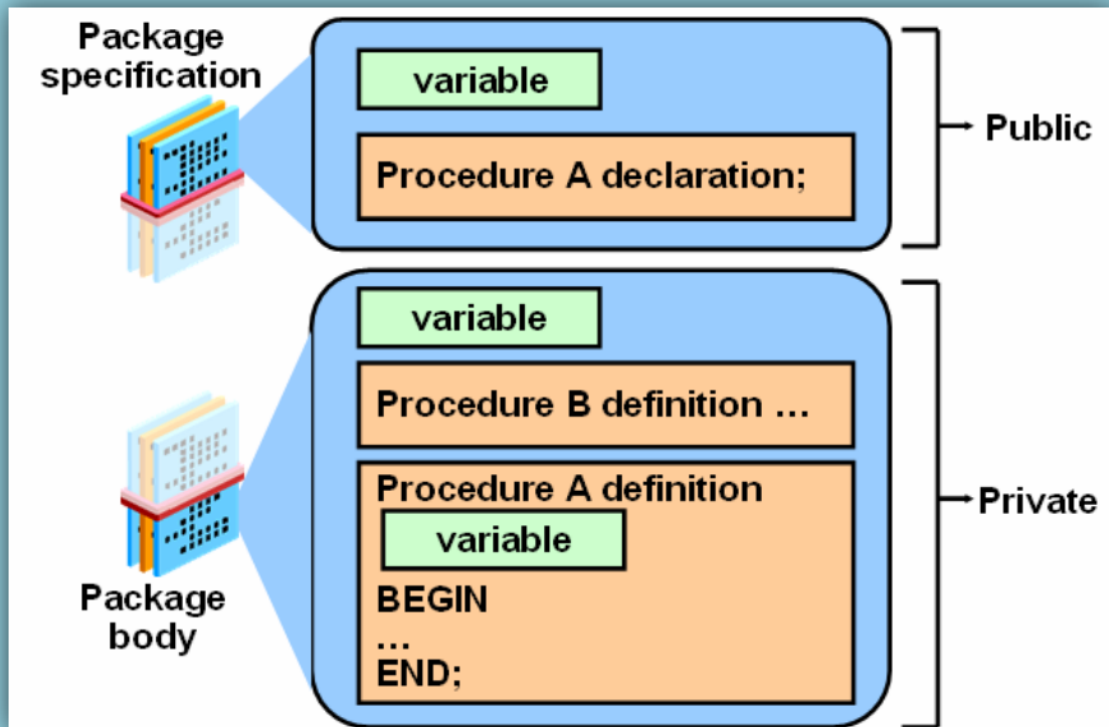
- التعرف على الـ Package ومكوناتها .
- انشاء Package تتضمن مجموعة من Variables – Cursors – Procedures – Functions – Exceptions .
- التعرف على كيفية استدعاء وتشغيل الـ Package .

تتكون الـ Package من جزئين وهما :

Body (٢)

Specification (١)

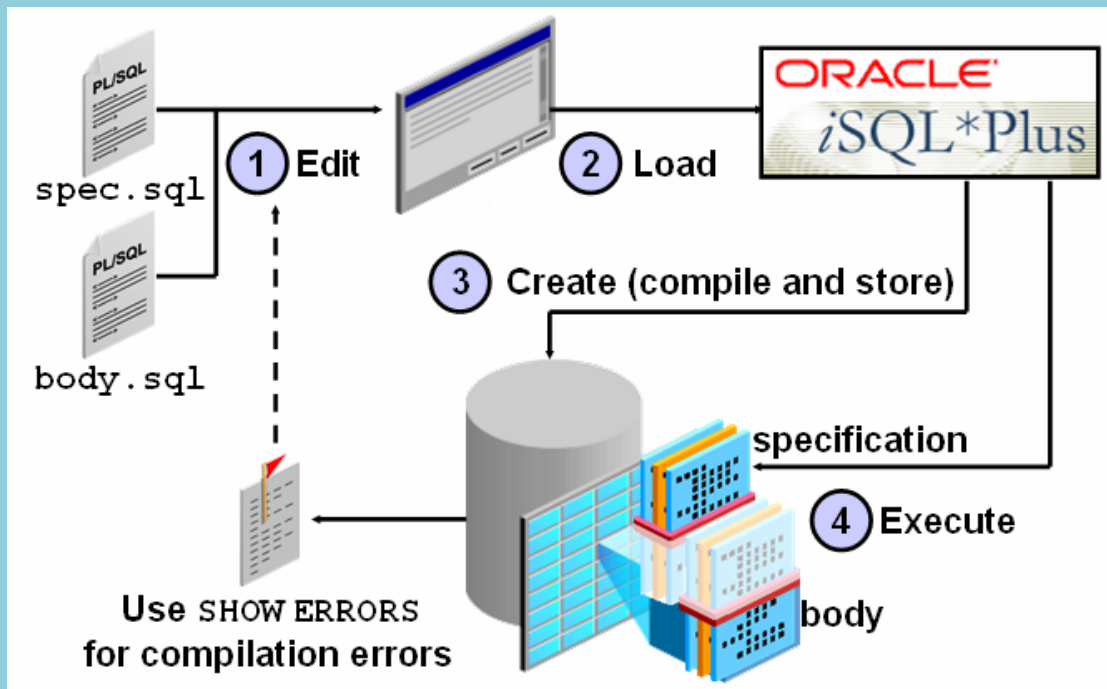
- لاحظ انه لايد من اشاء الـ Specification او لا ثم نقوم بإنشاء الـ Body حيث انه لايمكن اشاء الـ Body او لا .
- يمكنك إستدعاء او عمل (Call) لأى Procedure او Function بشرط وجودهم فى الـ Specification (الجزء الـ Public) ولكن لايمكن أستدعاء Procedure او Function الموجودين فى الـ Package Body فقط (الجزء الـ Private فقط) .



- فى الرسم السابق:

- يمكننا إستخدام الـ Package عن طريق الـ (Procedure A) حيث تم تعريف هذه الـ Procedure فى الـ Specification .
- ولايمكننا إستخدام الـ Package عن طريق الـ (Procedure B) حيث لم تعرف هذه الـ Procedure فى الـ Specification .
- لاحظ انه يتم حفظ الـ Specification والـ Body الخاصين بالـ Package فى جزئين منفصلين داخل الأوراكل .

- كيف يقوم اوراقل بتنفيذ الكود الخاص بالـ Package :-



- (١) يتم كتابة الكود اولاً في Text File .
- (٢) يتم عمل Load للكود الذى تم كتابته الى الـ Isql*plus Environment .
- (٣) يتم عمل Compile And Store للـ Package اذا كان الكود سليم يتم تخزين الـ Package فى الـ Database واذا كان فى الكود اى خطأ سنرى ان الـ Package يوجد بها اخطاء ولرؤية الخطأ نقوم بكتابة الأمر Show Errors وذلك لمعرفة الـ Compilation Errors .
- (٤) يتم عمل Execute لكل من الـ Specification والـ Body .

إنشاء الـ Package Specification

Creating the Package Specification

```
CREATE (OR REPLACE) PACKAGE package_name
IS | AS
    public type and item declarations
    subprogram specifications
END package_name;
```

- **Or Replace** : هى إختيارية ونقوم بكتابتها حتى نتمكن من عمل اى تعديل فى الـ Package Specification بدلاً من ان نقوم بإزالتها ثم التعديل فيها .
- **Subprogramme** : الـ Procedure والـ Function حيث يتم تعريفهم فى الـ Specification ويمكن إستخدامهم بعد ذلك عن طريق الـ Package .

Parameters	Description
Package_name	تعبر عن الاسم الخاص بالـ Package
Public Type and item Declaration	Declare Variable, Cursor, Constant , Exception , or Types
Subprogram Specification	تعريف كلاً من الـ Procedure و الـ Function

مثال:

- الهدف من المثال التالي هو انشاء Package تقوم بالتأكد من أن أى موظف جديد يتم تعيينه فى الشركة لابد أن يكون الـ Comm الخاص به اقل من اكبر Comm فى الشركة .

```
CREATE OR REPLACE PACKAGE comm_package IS
  g_comm NUMBER := 0.10;  --initialized to 0.10
  PROCEDURE reset_comm
    (p_comm IN NUMBER);
END comm_package;
```

- تم تعريف Variable بإسم G_comm ويأخذ قيمة إفتراضية 0.10
- وأيضاً تم تعريف Procedure بإسم Reset_comm لها Parameter واحد نوعه Number .
- وبالتالي يكون الـ G_comm والـ Reset_comm الأثنان من النوع public وذلك لأنه تم تعريفهم فى الـ Specification
- إنشاء الـ Package Body .

Creating the Package Body

```
CREATE [OR REPLACE] PACKAGE BODY package_name
IS|AS
  private type and item declarations
  subprogram bodies
END package name;
```

- **Or Replace** : هى إختيارية ونقوم بكتابتها حتى نتمكن من عمل أى تعديل فى الـ Package Body بدلاً من ان نقوم بإزالتها ثم التعديل فيها .
- **Subprogramme** : الـ Procedure والـ Function حيث يتم تعريفهم فى الـ Specification او لا يتم تعريفهم ونقوم هنا بكتابة الكود الخاص بهم فى الـ Package Body .

```
CREATE OR REPLACE PACKAGE BODY comm_package
IS
    FUNCTION validate_comm (p_comm IN NUMBER)
        RETURN BOOLEAN
    IS
        v_max_comm    NUMBER;
    BEGIN
        SELECT      MAX(commission_pct)
        INTO        v_max_comm
        FROM        employees;
        IF    p_comm > v_max_comm THEN RETURN(FALSE);
        ELSE    RETURN(TRUE);
        END IF;
    END validate_comm;
```

```
PROCEDURE reset_comm (p_comm IN NUMBER)
IS
BEGIN
    IF validate_comm(p_comm)
    THEN    g_comm:=p_comm; --reset global variable
    ELSE
        RAISE_APPLICATION_ERROR(-20210,'Invalid commission');
    END IF;
END reset_comm;
END comm_package;
```

- نقوم داخل الـ Package Body بإنشاء أى Subprogramme تم تعريفها فى الـ Specification او لم يتم تعريفها كما فى المثال السابق .

ملحوظة :

- يجب ان يكون اسم الـ Package فى الجزء الأول (Specification) هو نفس الأسم فى الجزء الثانى (Body) .

Invoking Package Constructs

- كيف يمكن إستدعاء وتشغيل الـ Package .

```
EXECUTE comm_package.reset_comm(0.15)
```

- فى المثال السابق تم تنفيذ الـ Package عن طريق نفس المستخدم او الـ USER الذى قام بإنشائها .

- اي اسم الحزمة او لا ثم نقطة ثم اسم الاجراء او الوظيفة .

```
EXECUTE scott.comm_package.reset_comm(0.15)
```

- فى هذا المثال تم تنفيذ الـ Package عن طريق مستخدم اخر مثل SCOTT وذلك عن طريق منح هذا المستخدم الـ Privilage او

الصلاحيه المناسبه له لى يتمكن من إستخدام هذه الـ Package .

Declaring a Bodiless Package

- تعريف Package بدون Package Body .

مثال :

```
CREATE OR REPLACE PACKAGE global_consts IS
  mile_2_kilo      CONSTANT  NUMBER  :=  1.6093;
  kilo_2_mile     CONSTANT  NUMBER  :=  0.6214;
  yard_2_meter    CONSTANT  NUMBER  :=  0.9144;
  meter_2_yard    CONSTANT  NUMBER  :=  1.0936;
END global_consts;
```

- في هذا المثال قمنا بتعريف Variables داخل الـ Package Specification وبالتالي تكون هذه الـ Variable من النوع الـ Global وقد قمنا بإنشاء الـ Package السابقه وهي تحتوى على مجموعة من الـ Variables وهذه الـ Variables تحتوى على قيم افتراضية .
- ثم نقوم بإنشاء الـ Procedure التالية وهي توضح إمكانية إستخدام الـ Package داخل الـ Procedure .

```
CREATE OR REPLACE PROCEDURE meter_to_yard
  (p_meter IN NUMBER, p_yard OUT NUMBER)
IS
BEGIN
  p_yard := p_meter * global_consts.meter_2_yard;
END meter_to_yard;
/
VARIABLE yard NUMBER
EXECUTE meter_to_yard (1, :yard)
```

Removing Package

To remove the package specification and the body, use the following syntax:

```
DROP PACKAGE package_name;
```

To remove the package body, use the following syntax :

```
DROP PACKAGE BODY package_name;
```

حذف الـ Package :

- عرفنا سابقاً ان الـ Package مقسمه الى جزئين (رأس Specification) و(جسم Body)
- لذلك يمكن إزالة الجسم Body وبقاء الرأس Specification تعمل بدون الجسم وذلك للكائنات التي ليس لها علاقة او مرتبطة بالـ Body مثل المتغيرات التي لها قيم ابتدائية والـ Exceptions والـ Cursors اما الإجراءات و الدوال فأنها لاتعمل وتنتج خطأ .

ملحوظة:

عند حذف الـ Package Specification يتم حذف الـ Body بطريقة أوتوماتيكية .

مزايا استخدام الـ Package:

- (١) يمكنك تجميع بعض الـ Subprogramme المرتبطة ببعضها في Package واحدة .
- (٢) يمكن عمل إخفاء للكود (Hiding Information) من خلال الـ Package وذلك في الجزء الـ Private .
- (٣) اداء افضل .
- (٤) اهم شيء هو عملية الصيانه حيث تسهل عملية الصيانة باستخدام الـ Package او الحزم .

CHAPTER 4 More Package Concepts

❖ عملية الـ Overloading :-

- تتيح هذه العملية إستخدام نفس الأسم لعدة Procedures او Functions داخل نفس الـ Package وهذه العملية تطلب ان يكون هناك إختلاف بين هذه الـ Subprogramme فى عدد الـ Parameters او ترتيبهم او انواع بياناتهم كما سوف يتم شرحه ونتيح هذه العملية كتابة وتطوير البرامج بمرونة اكثر .
- لاحظ ان هذه العملية لايمكن ان تحدث لـ Procedures او Functions خارج الـ Package .
- هذه العملية تمكن المبرمج من تعريف عدة برمجيات مختلفة بنفس الأسم داخل نفس الـ Package . ويتم التفريق والتعرف على الـ Subprogramme المراد إستخدامه بواسطة الأختلاف فى عدد الـ Parameters او نوع البيانات لهذه الـ Parameters او ترتيبهم .

❖ ملحوظة:

- لاتتم عملية الـ Overloading اذا كان الأختلاف فقط بين الـ Parameters فى نوع البيانات وهذه البيانات من نفس العائلة مثل (Number , Decimal) هما من نفس عائلة النوع الرقمية و (Varchar2, char) هم من نفس عائلة النوع الحرفى .
- ان يكون الأختلاف فقط فى القيمة المرجعة (Return) وذلك فى الدوال حتى لو كان انواع بيانات الرجوع من عائلات مختلفة .
- اذا حدث ماسبق من ملاحظات يحدث خطأ ولايحدث عملية Overloading .

❖ مثال على الـ Overloading :

```
CREATE OR REPLACE PACKAGE over_pack
IS
  PROCEDURE add_dept
    (p_deptno IN departments.department_id%TYPE,
     p_name IN departments.department_name%TYPE
                                     DEFAULT 'unknown',
     p_loc IN departments.location_id%TYPE DEFAULT 0);
  PROCEDURE add_dept
    (p_name IN departments.department_name%TYPE
                                     DEFAULT 'unknown',
     p_loc IN departments.location_id%TYPE DEFAULT 0);
END over_pack;
```

- بهذا نكون قد انشأنا Package Specification .
- فى المثال السابق قد انشأنا Package بداخلها اثنان Procure بنفس الأسم لتطبيق الـ Overloading .
- الأختلاف بين الـ Procedures هو ان :
 - الـ Procedure الأولى تأخذ ثلاثة Parameters .
 - الـ Procedure الثانية تأخذ اثنان Parameter فقط لاغير .

- بعد ذلك نقوم بإنشاء الـ Package Body كما يلي :

```
CREATE OR REPLACE PACKAGE BODY over_pack IS
  PROCEDURE add_dept
    (p_deptno IN departments.department_id%TYPE,
     p_name IN departments.department_name%TYPE DEFAULT 'unknown',
     p_loc IN departments.location_id%TYPE DEFAULT 0)
  IS
  BEGIN
    INSERT INTO departments (department_id,
                             department_name, location_id)
    VALUES (p_deptno, p_name, p_loc);
  END add_dept;
  PROCEDURE add_dept
    (p_name IN departments.department_name%TYPE DEFAULT 'unknown',
     p_loc IN departments.location_id%TYPE DEFAULT 0)
  IS
  BEGIN
    INSERT INTO departments (department_id,
                             department_name, location_id)
    VALUES (departments_seq.NEXTVAL, p_name, p_loc);
  END add_dept;
END over_pack;
```

الشكل التالي: يبين كيفية استدعاء وتنفيذ كلا الـ Procedures السابق تعريفهم في الـ Package Specification ونتيجة تنفيذهما :

```
EXECUTE over_pack.add_dept (980, 'Education', 2500)
EXECUTE over_pack.add_dept ('Training', 2400)
SELECT * FROM departments
WHERE department_id = 980;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
980	Education		2500

```
SELECT * FROM departments
WHERE department_name = 'Training';
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Training		2400

مثال اخر على الـ Overloading :

```

FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, P2 VARCHAR2) RETURN VARCHAR2;
FUNCTION TO CHAR (p1 NUMBER, P2 VARCHAR2) RETURN VARCHAR2;

```

ملحوظة:

- معظم الدوال الداخلية للأوراكل مطبق بها مفهوم الـ Overloading .
- مثال على ذلك دالة الـ To_Char كما فى المثال السابق ونرى فى المثال السابق اربع تعريفات مختلفة لنفس الدالة وبنفس الأسموذلك فى Package بإسم Standard .
- الأولى بها Parameter واحد نوعه Date وترجع Vchar2 .
- الثانية بها Parameter واحد نوعه Number وترجع Vchar2 .
- الثالثة بها إثنان Parameters الأول نوعه DATE والثانى Vchar2 وترجع VARCHAR2 .
- الرابعه بها إثنان Parameters الأول نوعه Number والثانى Vchar2 وترجع VARCHAR2 .

ملحوظة:

- اذا تم عمل دالة اخرى بنفس الأسم لدوال داخلية يتم إستبدال الدوال القديمة ووضع الدوال الجديدة التى عملها المستخدم وذلك فى الدوال الخاصة بالـ Standard Package .

إستخدام الـ Forward Declartionsملحوظة:

- عند إستخدام الـ Function داخل الـ Procedure ان تكون الـ Function قد تم إنشاؤها وتعريفها قبل الـ Procedure فيجب تعريف الكائن سواء متغير او Subprogramme قبل الأستخدام.

```

CREATE OR REPLACE PACKAGE BODY forward_pack
IS
  PROCEDURE award_bonus (. . .)
  IS
  BEGIN
    calc_rating (. . .);          --illegal reference
  END;

  PROCEDURE calc_rating (. . .)
  IS
  BEGIN
    . . .
  END;

END forward_pack;
/

```

- فى المثال السابق نلاحظ عدم إمكانية استخدام الـ Procedure المسمى Calc_rating حيث قمنا باستخدام الـ Procedure قبل تعريفه وهذا ملايكن استخدامه فى الـ PL\SQL فلكى نتمكن من استخدام تلك الـ Procedure لابد من تعريفه اولاً فى الجزء الخاص بالـ Declartion ثم نقوم بعد ذلك باستخدام هذه الـ Procedure فى الجزء الخاص بالـ Body كما فى المثال التالى :

```

CREATE OR REPLACE PACKAGE BODY forward_pack
IS
  PROCEDURE calc_rating(. . .);      -- forward declaration
  PROCEDURE award_bonus(. . .)
  IS                                  -- subprograms defined
  BEGIN                               -- in alphabetical order
    calc_rating(. . .);
    . . .
  END;

  PROCEDURE calc_rating(. . .)
  IS
  BEGIN
    . . .
  END;

END forward_pack;
/

```

- فى المثال السابق تم معالجة تلك المشكلة وذلك عن طريق تعريف (Calc_Rating) Procedure فى الجزء الخاص بالـ Declartion .

وهذه العملية تتيح تجميع اكثر من Subprogramme مرتبطة ببعضها فى Package واحدة بحيث يتم تعريفهم فى الـ Specification الخاص بالـ Package وكتابة الكود داخل الـ Body وذلك يفيد فى إخفاء تفاصيل الكود اى زيادة من السرية . (security)

Mo7amed Reda Abd El-Rahman

ORACLE FINANCIAL CONSULTANT

INSTRUCTOR ORACLE FINANCIAL R12

INSTRUCTOR ORACLE DEVELOPER

Scientific Computing Center – Mansoura University

E-mail: mreda47@hotmail.com

<https://www.facebook.com/Eng.mohamedreda.scc>

[Mobile: 01066734381](tel:01066734381)