



**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# ***Oracle Database***

## ***PL/SQL***

Autor:

Eric Gustavo Coronel Castillo

**Febrero - 2019**

## **ORACLE DATABASE – PL/SQL**

Esta obra es de uso exclusivo del Instituto de Investigación de UNI-FIIS

Esta prohibido su uso para otros fines.

Derechos Reservados © 2019 Eric Gustavo Coronel Castillo

**Primera Edición**

## Presentación

Una de las principales características de la Base de Datos Oracle es que permite la creación de bloques de código en la base de datos, que luego pueden ser reutilizados en diferentes aplicaciones. Estos bloques de código pueden ser funciones, procedimientos, paquetes, ó desencadenantes.

Este manual está compuesto por 13 lecciones, donde veremos de una manera práctica el lenguaje PL/SQL, que incluye básicamente:

- Creación de Funciones y Procedimientos
- Manejo de SQL en PL/SQL y Cursores
- Manejo de Excepciones
- Paquetes y Desencadenantes

Este manual lo desarrolle para que sea usado exclusivamente en el curso **ORACLE Database – PL/SQL** en Instituto de Investigación de la UNI- FIIS.

Atentamente,

Eric Gustavo Coronel Castillo  
gcoronelc.blogspot.pe

# Resumen

- Lección 01** En esta lección se detalla los esquemas de ejemplo que tiene la base de datos Oracle para desarrollar nuestros ejercicios, cabe mencionar que muchos textos y ejemplos en diversos artículos están desarrollados con estos esquemas.
- Lección 02** El desarrollador de aplicaciones empresariales debe aprovechar las ventajas de ofrece un motor de base de datos como Oracle 10g, y sin duda alguna el uso de bloques de programa almacenados y ejecutados por el motor de la base de datos es una de estas grandes ventajas. Esta lección lo introduce en la programación con PL/SQL.
- Lección 03** En esta lección se desarrolla el tema de estructuras de control. Estas estructuras de control son la base para implementar los algoritmos de los procesos de negocio en funciones, procedimientos y/o desencadenantes.
- Lección 04** La manipulación de datos es mas efectiva si agrupamos datos relacionados como una sola unidad, y esto se logra aplicando registros. En esta lección desarrollamos este tema con ejemplos muy prácticos y sobre todo el uso de `%ROWTYPE`.
- Lección 05** El uso del lenguaje SQL es sin duda la base para tratar los datos, pero utilizado de manera incrustada en PL/SQL tiene sus restricciones. En esta lección se detalla como utilizar SQL en PL/SQL.
- Lección 06** El proceso de grandes cantidades de registros dentro de un procedimiento es la base para el desarrollo de procesos empresariales, preparar datos para consultas gerenciales, etc. En esta lección veremos como utilizar los cursores para procesar grandes volúmenes de registros.
- Lección 07** Los errores en tiempo de ejecución es todo un problema para los programadores, pero en PL/SQL tenemos un mecanismo basado en excepciones para controlarlos. En esta lección desarrollaremos el uso de excepciones para el control de errores en tiempo de ejecución con el uso de ejemplos prácticos.
- Lección 08** El uso de parámetros de las funciones y procedimientos tiene sus detalles. En esta lección profundizaremos en el uso de parámetros en funciones y procedimientos.
- Lección 09** La mejor manera de organizar nuestras funciones y procedimientos es a través de paquetes. En esta lección se muestra como utilizar los paquetes para agrupar funciones y procedimientos relacionados como una sola unidad dentro de un paquete.
- Lección 10** El uso de desencadenantes permiten programar procesos en cascada, por ejemplo al grabar una factura se genere en background la orden de despacho, esto se logra con el uso de desencadenantes. En este capítulo se ilustra como utilizar los desencadenantes para programar procesos en cascada.

# Contenido

CAPITULO	TEMA	PAGINA
1	Introducción a Oracle Database	1
2	Esquemas Ejemplo	14
3	Crear un Esquema	20
4	Fundamentos de PL/SQL	34
5	Estructuras de Control	40
6	Estructuras de Datos	55
7	Tratamiento de Errores	70
8	Lenguaje SQL	77
9	Lenguaje SQL en PL/SQL	94
10	Cursores	107
11	Funciones y Procedimientos	119
12	Paquetes	128
13	Desencadenantes	133





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 01 Introducción a Oracle Database**

### **Contenido**

Arquitectura de un servidor Oracle

Esquema General

La instancia de Oracle

Procesos de fondo

Area Global del Sistema (SGA)

La base de datos

Estructuras Adicionales

Conexión con una instancia de Oracle

Verificación de los servicios

Esquema General

Conexión local utilizando SQL Plus

Vistas del Sistema

Comandos SQL/Plus

Conexión remota utilizando SQL Plus

Conexión Utilizando iSQL\*Plus

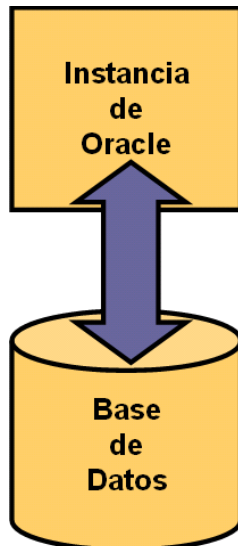
Conceptos generales de almacenamiento

TableSpace

DataFile

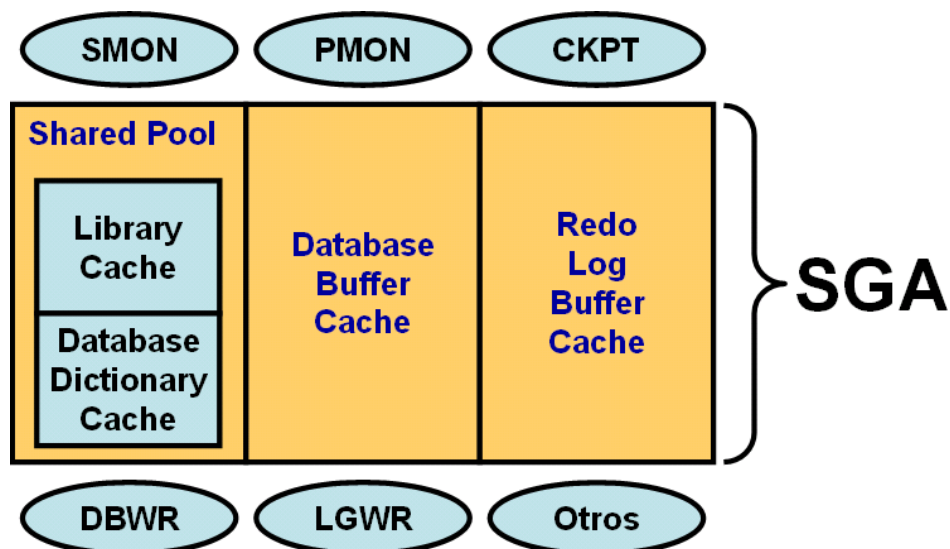
## Arquitectura de un servidor Oracle

### Esquema General



- Por cada instancia de Oracle se tiene una sola base de datos
- En un servidor se pueden crear varias instancias, pero se recomienda solo una, porque cada instancia consume muchos recursos.

### La instancia de Oracle



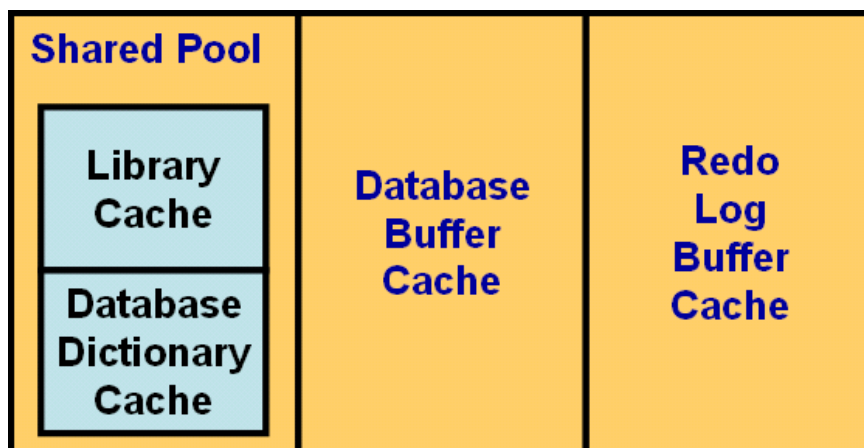
- Está compuesta por procesos de fondo y un área de memoria compartida denominada SYSTEM GLOBAL AREA (SGA).
- El SGA es utilizado para el intercambio de datos entre el servidor y las aplicaciones cliente.
- Una instancia de Oracle solo puede abrir una sola base de datos a la vez.



### Procesos de fondo

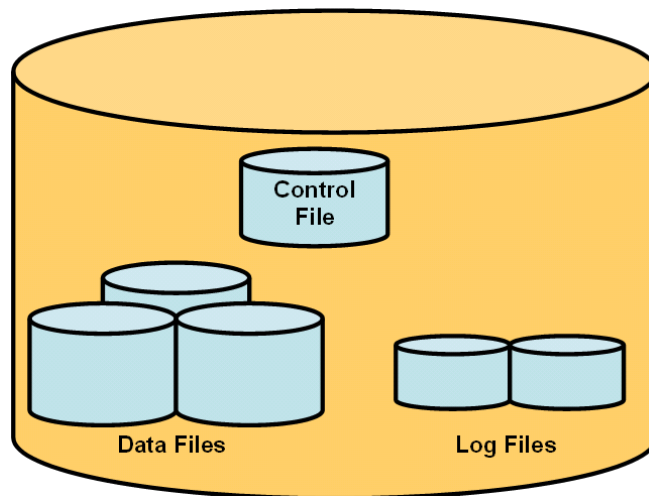
<b>PMON</b>	<i>Process Monitor.</i> Monitorea los procesos de los usuarios, en caso que la conexión falle.
<b>SMON</b>	<i>System Monitor.</i> Este proceso es el encargado de recuperar la instancia y abrir la base de datos, en caso que ocurra alguna falla.
<b>CKPT</b>	<i>CheckPoint Process.</i> Sintoniza las tareas de grabación en la base de datos.
<b>DBWR</b>	<i>Database Writer.</i> Escribe los bloques de datos de la memoria a la base de datos.
<b>LGWR</b>	<i>Log Writer.</i> Graba los bloques del Redo Log del buffer a los archivos Redo Log File.

### Área Global del Sistema (SGA)



<b>Library Cache</b>	Almacena las sentencias SQL más recientes en memoria.
<b>Database Dictionary Cache</b>	Buffer para el diccionario de datos. Tablas, columnas, tipos, índices.
<b>Database Buffer Cache</b>	Buffer de la base de datos, contiene bloques de datos que han sido cargados desde los Data File.
<b>Redo Log Buffer Cache</b>	Bloques de datos que han sido actualizados.

### La base de datos



<b>Control File</b>	Contiene información para mantener y controlar la integridad de la base de datos.
<b>Data Files</b>	Son los archivos donde se almacenan los datos de las aplicaciones.
<b>Redo Log Files</b>	Almacena los cambios hechos en la base de datos con propósito de recuperarlos en caso de falla.

### Estructuras Adicionales

<b>Archivo de Parámetros</b>	Contiene parámetros y valores que definen las características de la instancia y de la base de datos, por ejemplo contiene parámetros que dimensionan el SGA.
<b>Archivo de Password</b>	Se utiliza para validar al usuario que puede bajar y subir la instancia de Oracle.
<b>Archivos Archived Log Files</b>	Los Archived Log Files son copias fuera de línea de los archivos Redo Log Files que son necesarios para el proceso de Recovery en caso de falla del medio de almacenamiento.

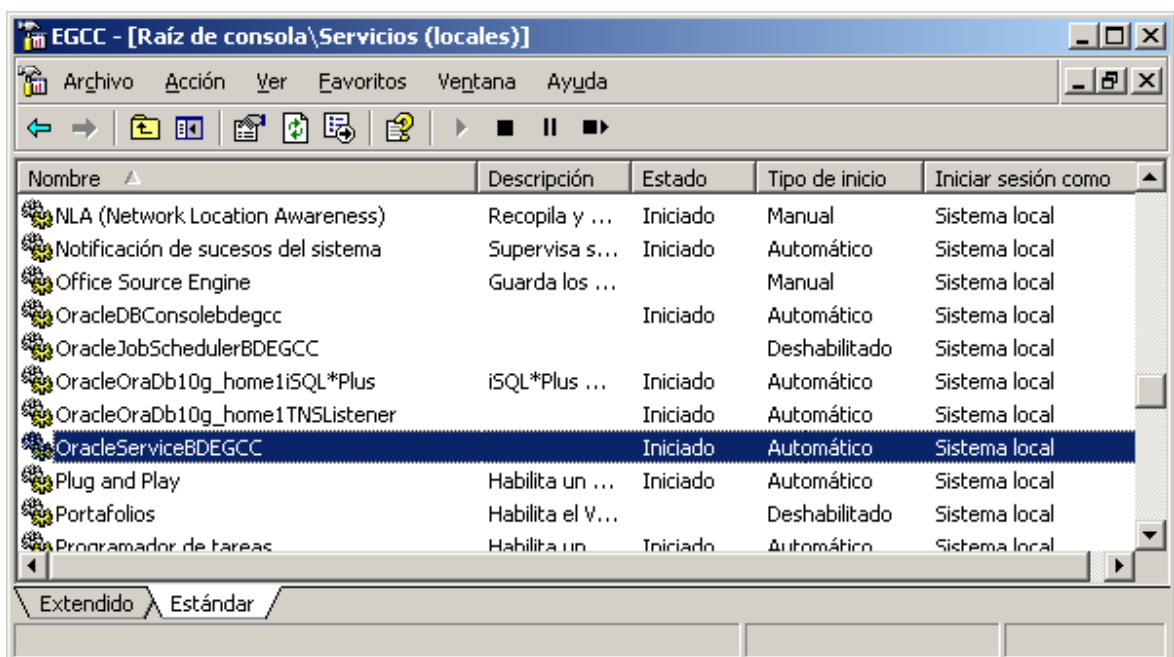
## Conexión con una instancia de Oracle

### Verificación de los servicios

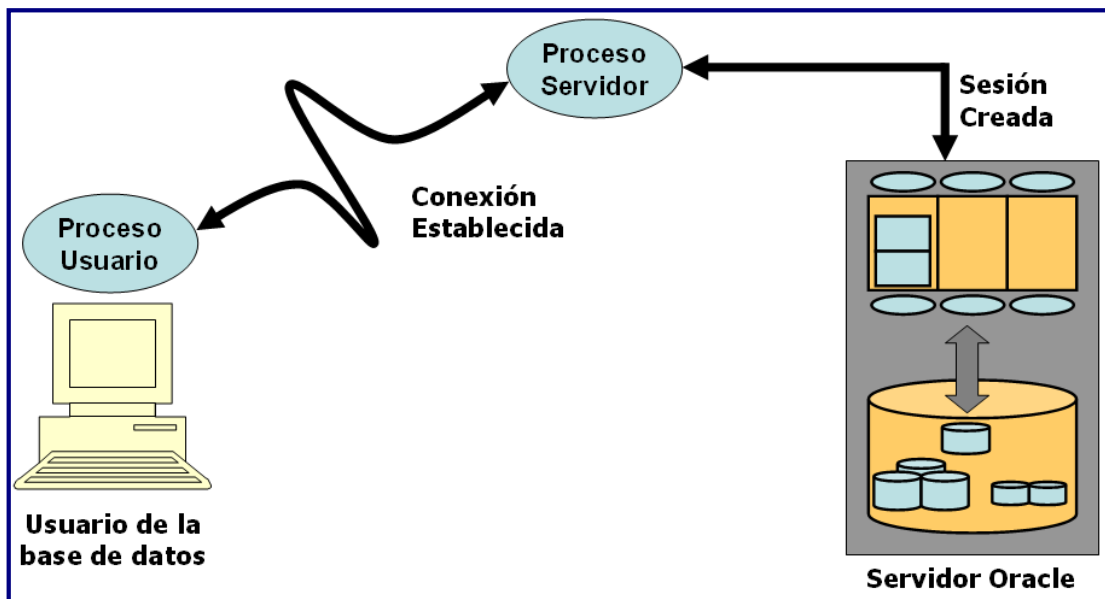
De la relación de servicios creados durante la instalación de Oracle, por ahora nos interesa básicamente tres:

- El servicio relacionado con la instancia y la base de datos, cuyo nombre tiene la siguiente estructura: **OracleServiceXXX**, donde **XXX** representa el nombre de la instancia. Por ejemplo, si la instancia tiene por nombre **BDEGCC**, el servicio sería **OracleServiceBDEGCC**.
- El servicio relacionado con la disponibilidad del servidor para el acceso remoto, el nombre de este servicio es: **OracleOraDb10g\_home1TNSListener**.
- El servicio relacionado con la aplicación **iSQL\*Plus**, este servicio permite ejecutar esta aplicación desde cualquier equipo de la red vía el protocolo HTTP haciendo uso de un navegador Web, el nombre de este servicio es **OracleOraDb10g\_home1iSQL\*Plus**.

Estos tres servicios deben estar ejecutándose, y su verificación se puede realizar en la venta de servicios, a la que accedemos desde el **Panel de control / Herramientas administrativas**.



### Esquema General



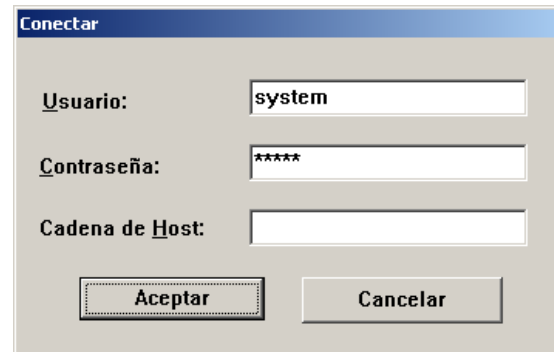
<b>Proceso Usuario</b>	Programa, aplicación ó herramienta que usa el usuario para iniciar un proceso de usuario y establecer una conexión.
<b>Proceso Servidor</b>	<p>Una vez que el proceso de usuario establece la conexión, un proceso servidor es iniciado, el cual manejará las peticiones del proceso usuario.</p> <p>Un proceso servidor puede ser dedicado, es decir solo atiende las peticiones de un solo proceso usuario, o puede ser compartido, con lo cual puede atender múltiples procesos usuarios.</p>
<b>Sesión</b>	<p>Una sesión es una conexión específica de un usuario a un servidor Oracle.</p> <ul style="list-style-type: none"><li>• Se inicia cuando el usuario es validado por el servidor Oracle.</li><li>• Finaliza cuando el usuario termina la sesión en forma normal (logout) o aborta la sesión.</li></ul>

### Conexión local utilizando SQL Plus

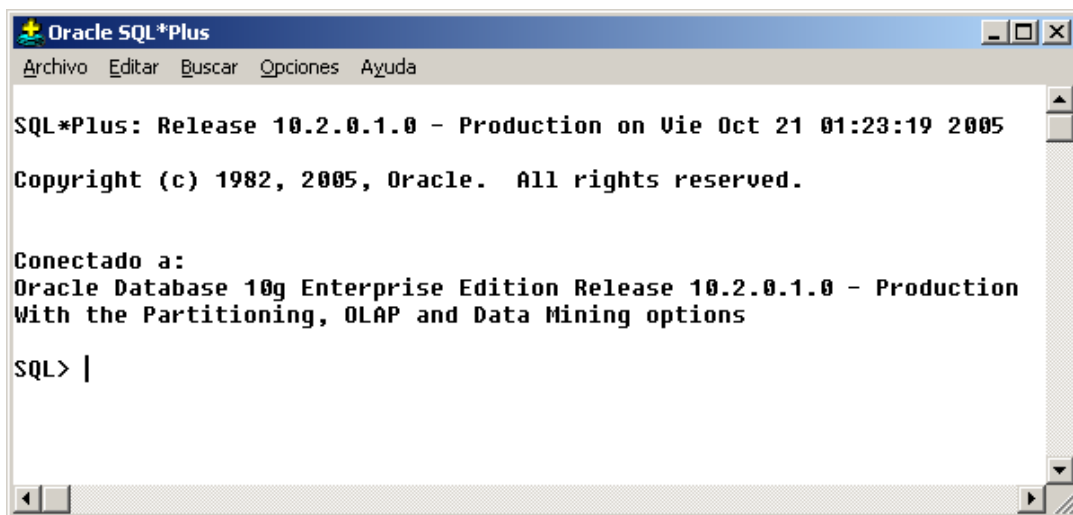
SQL Plus es una herramienta que permite al usuario comunicarse con el servidor, para procesar comandos SQL ó PL/SQL, tiene la flexibilidad de poder realizar inicio y parada (shutdown) de la base de datos.

En la ventana inicial de conexión debemos ingresar el usuario y su contraseña, debe recordar la contraseña que estableció para los usuarios **sys** y **system**.

Usuario	Contraseña
sys	admin.
system	admin



La pantalla de bienvenida de SQL Plus mostrará los siguientes mensajes:



En estos momentos estamos listos para trabajar, por ejemplo, si queremos conectarnos como scout, el comando es el siguiente:

```
SQL> show user
USER es "SYSTEM"

SQL>
```

### Vistas del Sistema

Tenemos algunas vistas que podemos consultar para verificar nuestro servidor: v\$instance, v\$database y v\$sga.

Para realizar las consultas a las vistas, ejecutamos los siguientes comandos:

```
SQL> connect system/manager
Conectado.

SQL> select instance_name from v$instance;

INSTANCE_NAME
-----
sidegcc

SQL> select name from v$database;

NAME
-----
DBEGCC

SQL> select * from v$sga;

NAME                                VALUE
-----
Fixed Size                          282576
Variable Size                       83886080
Database Buffers                    33554432
Redo Buffers                        532480
```

### Comandos SQL/Plus

También contamos con comandos SQL/Plus, algunos de ellos son:

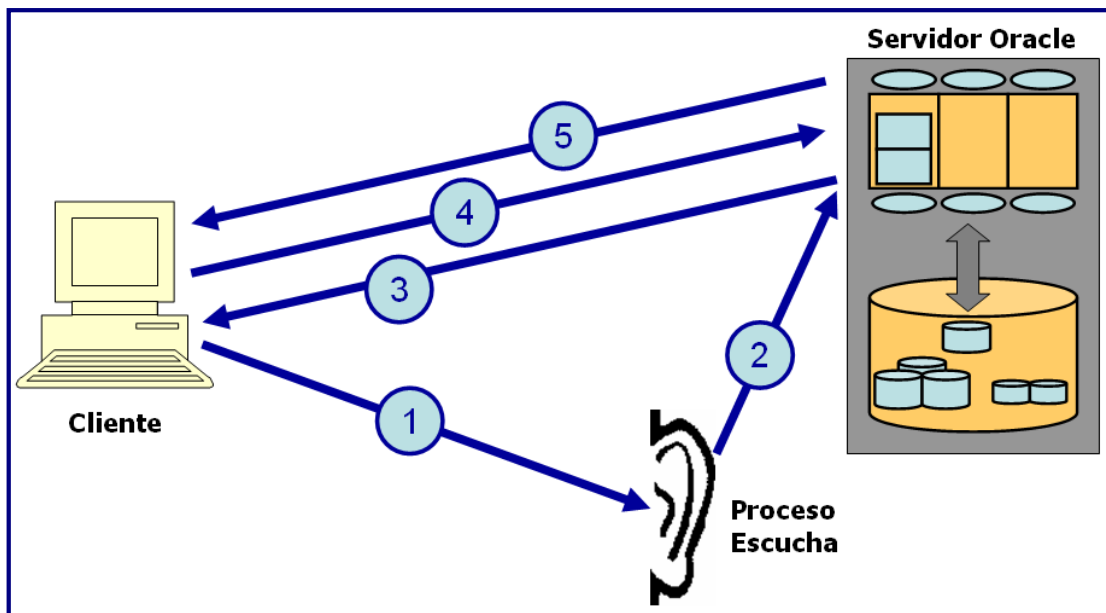
- RUN**            Vuelve a ejecutar la última instrucción ejecutada.
- EDIT**           Edita la última instrucción ejecutada.
- START**          Ejecuta las instrucciones que se encuentran en un archivo.
- SPOOL**          Envía la sesión de trabajo a un archivo.

### Conexión remota utilizando SQL Plus

Oracle tiene su herramienta de red que permite a las aplicaciones en general conectarse a servidores Oracle. El nombre inicial de esta herramienta fue SQL\*Net, luego fue renombrada con el nombre Net8, y hoy día se le conoce como Oracle Net.

Para que una aplicación pueda conectarse remotamente a un servidor Oracle, es necesario que el **Proceso Escucha** se encuentre ejecutándose en el servidor, específicamente el servicio **OracleOraDb10g\_home1TNSListener**.

El esquema general de la conexión remota se puede apreciar en el siguiente gráfico.



El proceso se describe a continuación:

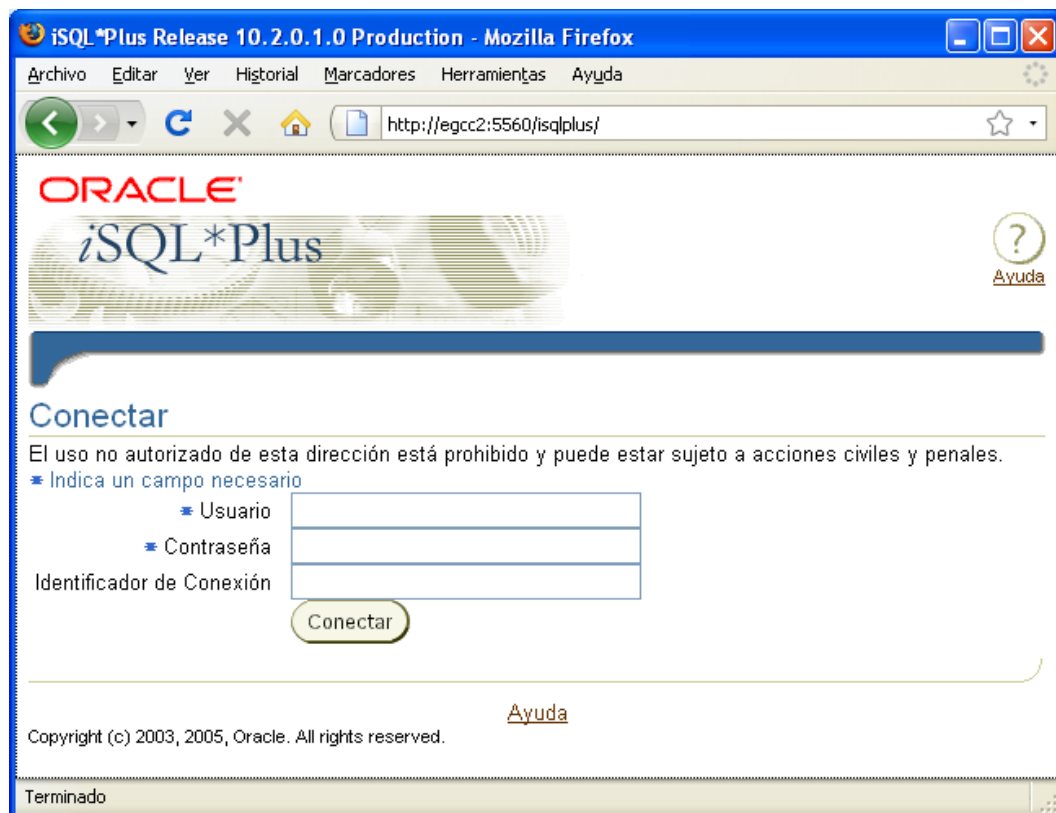
1. El cliente establece una conexión al Proceso Escucha usando el protocolo configurado y envía un paquete CONNECT.
2. El proceso escucha comprueba que el SID esté definido. Si es así, generará un nuevo proceso para ocuparse de la conexión. Una conexión se establece entre el proceso escucha y el nuevo proceso del servidor para pasarle la información del proceso de inicialización. Luego la conexión es cerrada.
3. El proceso del servidor envía un paquete al cliente.
4. Un nuevo paquete CONNECT es enviado al proceso servidor dedicado.
5. El proceso de servidor dedicado acepta la conexión entrante y remite un mensaje de ACEPTADO al nuevo al cliente.

### Conexión Utilizando iSQL\*Plus

Para utilizar **iSQL\*Plus** es necesario que el servicio **OracleOraDb10g\_home1iSQL\*Plus** se encuentre **Iniciado**, y la conexión se puede realizar desde cualquier equipo de la red utilizando un Navegador Web y escribiendo la siguiente **URL** en el campo **Dirección**:

**`http://nombre_servidor:5560/isqlplus`**

La ventana inicial, que se muestra en la siguiente figura, solicita el **Usuario** y **Contraseña** para poder iniciar sesión.





Después de iniciar sesión ingresa a la aplicación, tal como se ilustra en la siguiente figura:



Des aquí usted podrá ejecutar sentencias SQL y bloques de programa PL/SQL.

### Conceptos generales de almacenamiento

#### TableSpace

Unidad lógica en que se divide una base de datos. Es posible consultar los tablespaces utilizando los siguientes comandos:

```
SQL> select * from v$tablespace;
```

TS#	NAME	INC	BIG	FLA	ENC
0	SYSTEM	YES	NO	YES	
1	UNDOTBS1	YES	NO	YES	
2	SYSAUX	YES	NO	YES	
4	USERS	YES	NO	YES	
3	TEMP	NO	NO	YES	
6	EXAMPLE	YES	NO	YES	

6 filas seleccionadas.

```
SQL> select tablespace_name from dba_tablespaces  
2 order by 1;
```

TABLESPACE_NAME
EXAMPLE
SYSAUX
SYSTEM
TEMP
UNDOTBS1
USERS

6 filas seleccionadas.

### DataFile

Es el archivo físico donde se almacenan los datos.

```
SQL> column tablespace_name format a20
SQL> column file_name format a55
SQL> select tablespace_name, file_name from dba_data_files
      2 order by 1;
```

TABLESPACE_NAME	FILE_NAME
EXAMPLE	H:\ORACLE\PRODUCT\10.2.0\ORADATA\BDEGCC\EXAMPLE01.DBF
SYS_AUX	H:\ORACLE\PRODUCT\10.2.0\ORADATA\BDEGCC\SYS_AUX01.DBF
SYSTEM	H:\ORACLE\PRODUCT\10.2.0\ORADATA\BDEGCC\SYSTEM01.DBF
UNDOTBS1	H:\ORACLE\PRODUCT\10.2.0\ORADATA\BDEGCC\UNDOTBS01.DBF
USERS	H:\ORACLE\PRODUCT\10.2.0\ORADATA\BDEGCC\USERS01.DBF

```
SQL>
```





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 02 Esquemas Ejemplo**

### **Contenido**

Esquema de Base de Datos

Esquema SCOTT

Esquema HR

Consultar la Estructura de una Tabla

Consultar el Contenido de una Tabla

## Esquema de Base de Datos

El conjunto de objetos que tiene una cuenta de usuario se denomina *esquema* del usuario, por lo tanto, el nombre del esquema será también el nombre del usuario.

Cuando creamos la base de datos de Oracle, por defecto crea dos esquemas de ejemplo, para poder realizar nuestras pruebas.

Estos esquemas son los siguientes:

**SCOTT** Se trata de un esquema muy básico de recursos humanos, cuenta con tan solo 4 tablas.

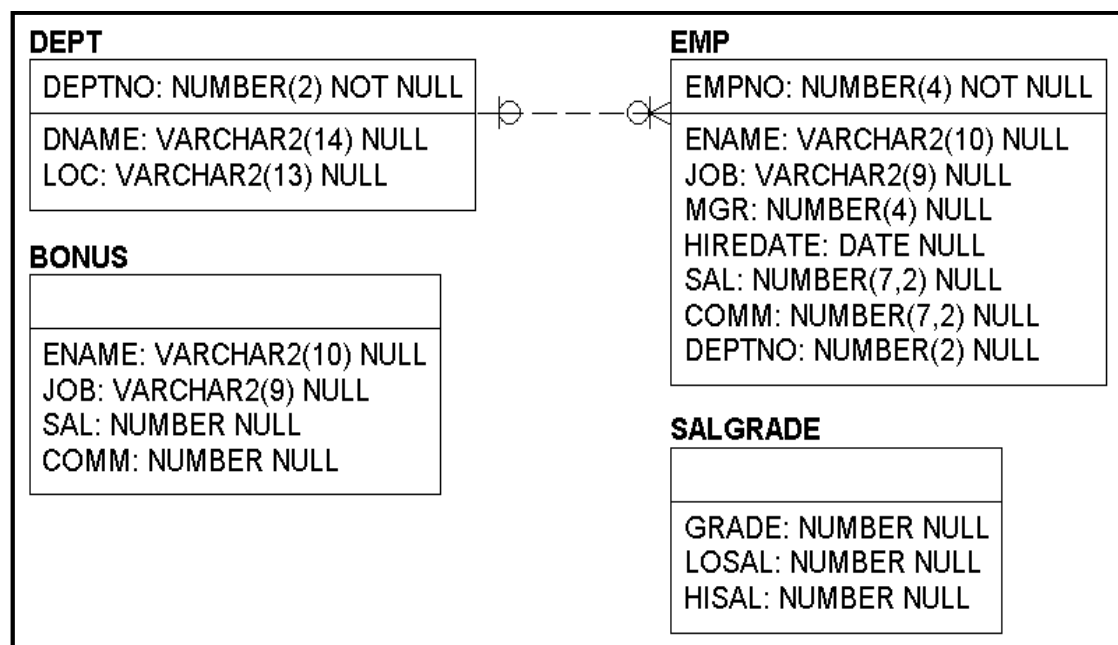
**HR** Se trata también de un esquema de recursos humanos, pero este esquema cuenta con 7 tablas.

## Esquema SCOTT

Para poder iniciar una sesión en el esquema de scott debemos utilizar los siguientes datos:

<b>Usuario</b>	scott
<b>Contraseña</b>	tiger

Su esquema es el siguiente:



El siguiente script permite consultar el catálogo de scott:

### Script 2.1

```
SQL> conn / as sysdba  
Conectado.
```

```
SQL> alter user scott  
2 account unlock;
```

Usuario modificado.

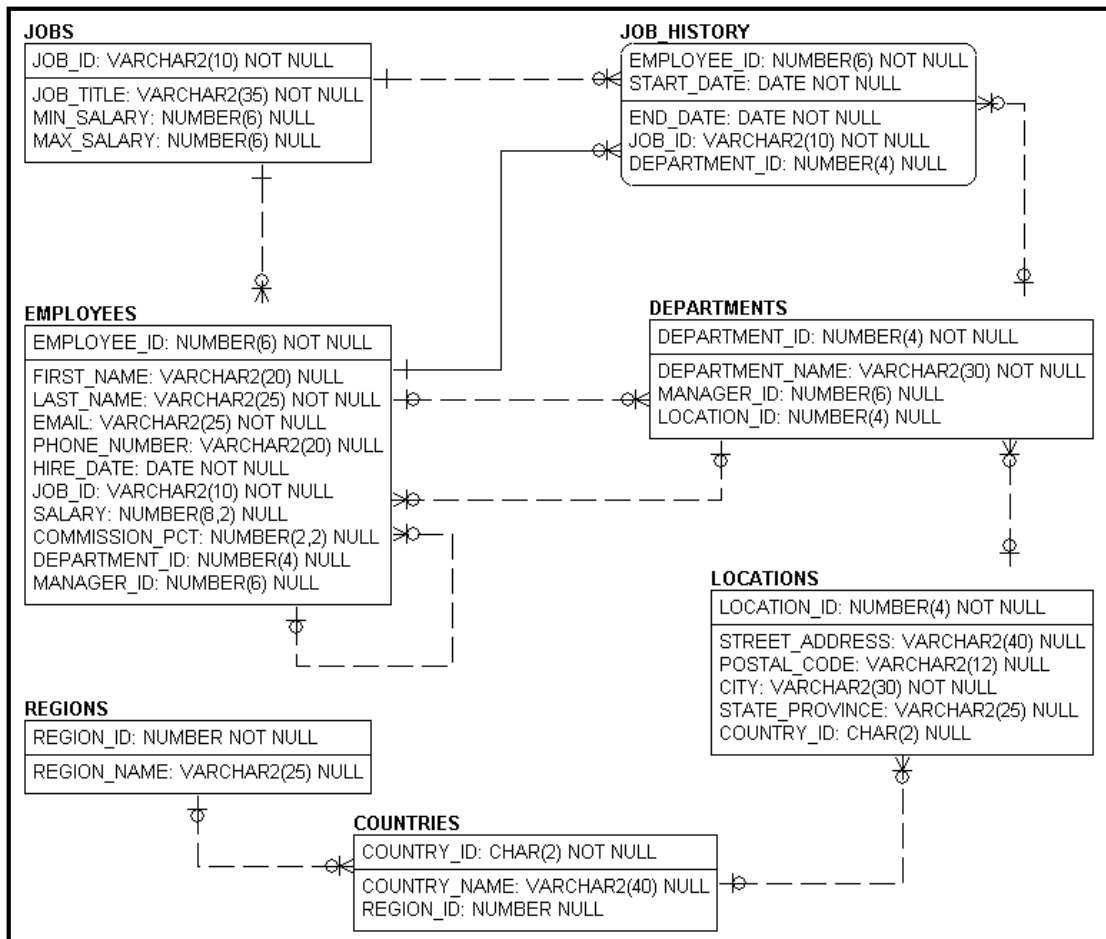
```
SQL> connect scott/tiger  
Connected.
```

```
SQL> select * from cat;
```

TNAME	TABTYPE	CLUSTERID
DEPT	TABLE	
EMP	TABLE	
BONUS	TABLE	
SALGRADE	TABLE	

## Esquema HR

Su esquema es el siguiente:



La cuenta de usuario HR por defecto está bloqueada, así que lo primero que debemos hacer es desbloquearla, el script es el siguiente:

### Script 2.2

```

SQL> connect / as sysdba
Connected.

SQL> alter user hr
  2  identified by hr
  3  account unlock;

User altered.
  
```



Ahora si podemos consultar el catalogo del esquema HR:

### Script 2.3

```
SQL> connect hr/hr
Connected.

SQL> select * from cat;

TABLE_NAME                TABLE_TYPE
-----
COUNTRIES                  TABLE
DEPARTMENTS                TABLE
DEPARTMENTS_SEQ            SEQUENCE
EMPLOYEES                  TABLE
EMPLOYEES_SEQ              SEQUENCE
EMP_DETAILS_VIEW           VIEW
JOBS                       TABLE
JOB_HISTORY                TABLE
LOCATIONS                  TABLE
LOCATIONS_SEQ              SEQUENCE
REGIONS                    TABLE

11 rows selected.
```

También podemos utilizar la siguiente consulta:

### Script 2.4

```
SQL> select * from tab;

TNAME                TABTYPE  CLUSTERID
-----
COUNTRIES            TABLE
DEPARTMENTS          TABLE
EMPLOYEES            TABLE
EMP_DETAILS_VIEW     VIEW
JOBS                 TABLE
JOB_HISTORY          TABLE
LOCATIONS            TABLE
REGIONS              TABLE

8 rows selected.
```

## Consultar la Estructura de una Tabla

### Sintaxis

```
DESCRIBE Nombre_Tabla
```

Como ejemplo ilustrativo consultemos la estructura de la tabla EMP del esquema SCOTT:

### Script 2.5

```
SQL> connect scott/tiger
Connected.

SQL> describe emp
Name                               Null?    Type
-----
EMPNO                              NOT NULL NUMBER(4)
ENAME                              VARCHAR2(10)
JOB                                 VARCHAR2(9)
MGR                                 NUMBER(4)
HIREDATE                           DATE
SAL                                 NUMBER(7,2)
COMM                                NUMBER(7,2)
DEPTNO                             NUMBER(2)
```

## Consultar el Contenido de una Tabla

### Sintaxis

```
SELECT * FROM Nombre_Tabla
```

Como ejemplo ilustrativo consultemos el contenido de la tabla DEPT de SCOTT:

### Script 2.6

```
SQL> select * from dept;

DEPTNO DNAME          LOC
-----
10 ACCOUNTING      NEW YORK
20 RESEARCH        DALLAS
30 SALES            CHICAGO
40 OPERATIONS      BOSTON
```



**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 03 Crear un Esquema**

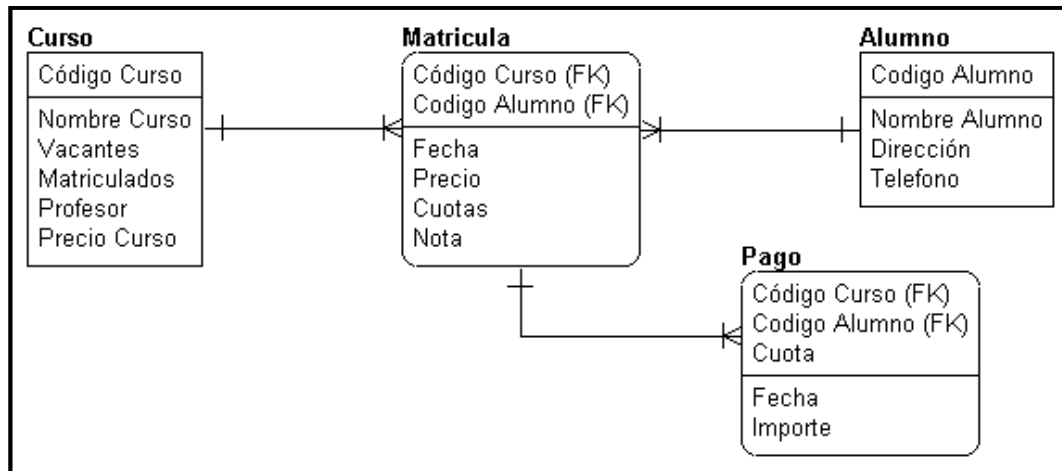
### **Contenido**

- Caso a Desarrollar
- Creación del Usuario para el Esquema
- Creación de Tablas
- Restricción Primary Key (PK)
- Restricción Foreign Key (FK)
- Restricción Default (Valores por Defecto)
- Restricción NOT NULL (Nulidad de una Columna)
- Restricción Unique (Valores Únicos)
- Restricción Check (Reglas de Validación)
- Asignar Privilegios a Usuarios

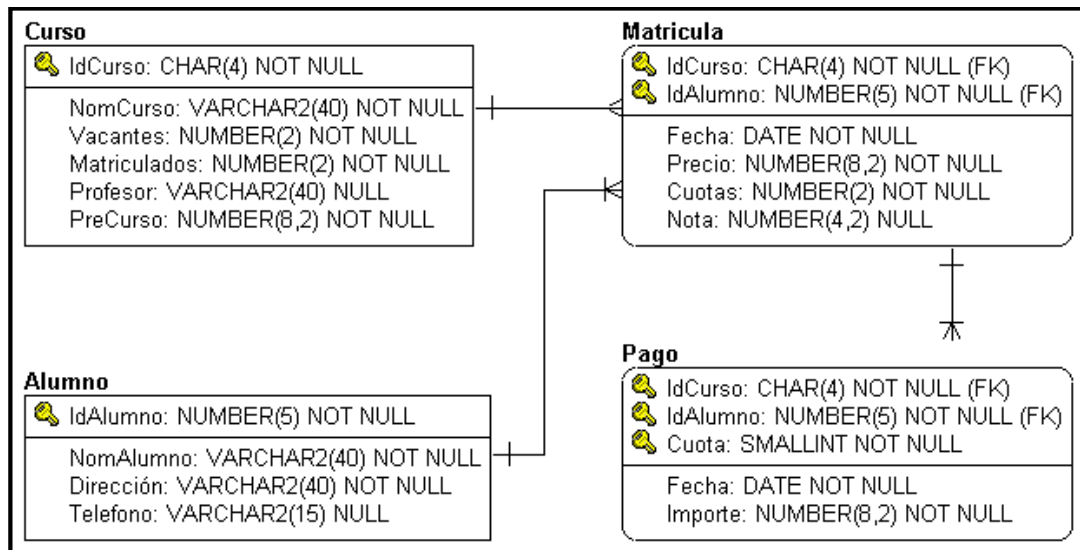
### Caso a Desarrollar

El siguiente modelo trata de una empresa que ofrece cursos de extensión, los participantes tienen la libertad de matricularse sin ninguna restricción, y pueden tener facilidades de pago.

### Modelo Lógico



### Modelo Físico



## Creación del Usuario para el Esquema

---

### Creación del Usuario

#### Script 9.1

```
SQL> conn / as sysdba
Connected.

SQL> create user egcc
      2 identified by admin;

User created.
```

### Asignar Privilegios

Asignaremos privilegios al usuario **egcc** a través de los roles **connect** y **resource**, los cuales le otorgan los privilegios necesarios para que pueda crear sus objetos.

#### Script 9.2

```
SQL> grant connect, resource to egcc;

Grant succeeded.
```

Ahora ya podemos ingresar como usuario **egcc** y crear los objetos que corresponden a su esquema.

## Creación de Tablas

### Sintaxis

```
Create Table NombreTabla(  
  Columna1 Tipo1 [ NULL | NOT NULL ],  
  Columna2 Tipo2 [ NULL | NOT NULL ],  
  Columna2 Tipo2 [ NULL | NOT NULL ],  
  . . .  
  . . .  
);
```

### Tabla Curso

#### Script 9.3

```
SQL> connect egcc/admin  
Connected.  
  
SQL> CREATE TABLE Curso (  
2      IdCurso          CHAR(4) NOT NULL,  
3      NomCurso          VARCHAR2(40) NOT NULL,  
4      Vacantes          NUMBER(2) NOT NULL,  
5      Matriculados      NUMBER(2) NOT NULL,  
6      Profesor          VARCHAR2(40) NULL,  
7      PreCurso          NUMBER(8,2) NOT NULL  
8  );  
  
Table created.
```

### Tabla Alumno

Escriba el script para crear la tabla Alumno.

### Tabla Matricula

Escriba el script para crear la tabla Matricula.

### Tabla Pago

Escriba el script para crear la tabla Pago.

### Restricción Primary Key (PK)

La restricción Primary Key se utiliza para definir la clave primaria de una tabla, en el siguiente cuadro se especifica la(s) columna(s) que conforman la PK de cada tabla.

Tabla	Primary Key
Curso	Incurso
Alumno	IdAlumno
Matricula	IdCurso, IdAlumno
Pago	IdCurso, IdAlumno, Cuota

#### Sintaxis

```
Alter Table NombreTabla  
Add Constraint PK_NombreTabla  
Primary Key ( Columna1, Columna2, . . . );
```

### Tabla Curso

#### Script 9.4

```
SQL> Alter Table Curso  
2      Add Constraint PK_Curso  
3      Primary Key ( IdCurso );  
  
Table altered.
```

### Tabla Alumno

Escriba el script para crear la PK de la tabla Alumno.



### Tabla Matricula

Escriba el script para crear la PK de la tabla Matricula.

### Tabla Pago

Escriba el script para crear la PK de la tabla Pago.

## Restricción Foreign Key (FK)

La restricción Foreign Key se utiliza para definir la relación entre dos tablas, en el siguiente cuadro se especifica la(s) columna(s) que conforman la FK de cada tabla.

Tabla	Foreign Key	Tabla Referenciada
Matricula	IdCurso	Curso
	IdAlumno	Alumno
Pago	IdCurso, IdAlumno	Matricula

### Sintaxis

```
Alter Table NombreTabla  
Add Constraint FK_NombreTabla_TablaReferenciada  
Foreign Key ( Columna1, Columna2, . . . )  
References TablaReferenciada;
```

Es necesario que en la tabla referenciada esté definida la PK, por que la relación se crea entre la PK de la tabla referenciada y las columnas que indicamos en la cláusula Foreign Key.

## Tabla Matricula

### 1ra FK

La primera FK de esta tabla es IdCurso y la tabla referenciada es Curso, el script para crear esta FK es el siguiente:

### Script 9.5

```
SQL> Alter table Matricula  
2 Add Constraint FK_Matricula_Curso  
3 Foreign Key ( IdCurso )  
4 References Curso;  
  
Table altered.
```

### 2da FK

La segunda FK de esta tabla es **IdAlumno** y la tabla referenciada es **Alumno**, escriba usted el script para crear ésta FK.

### Tabla Pago

Esta tabla solo tiene una FK y está compuesta por dos columnas: **IdCurso** e **IdAlumno**, y la tabla referenciada es **Matricula**, escriba usted el script para crear ésta FK.

### Restricción Default (Valores por Defecto)

El *Valor por Defecto* es el que toma una columna cuando no especificamos su valor en una sentencia insert.

#### Sintaxis

```
Alter Table NombreTabla  
Modify ( NombreColumna Default Expresión );
```

### Ejemplo

El número de vacantes por defecto para cualquier curso debe ser 20.

#### Script 9. 6

```
SQL> Alter Table Curso  
2 Modify ( Vacantes default 20 );  
  
Table altered.
```

Para probar el default insertemos un registro en la tabla curso.

#### Script 9. 7

IDCU	NOMCURSO	VACANTES	MATRICULADOS	PROFESOR	PRECURSO
C001	Oracle 9i - Nivel Inicial	20	10	Gustavo Coronel	350

### Restricción NOT NULL (Nulidad de una Columna)

Es muy importante determinar la nulidad de una columna, y es muy importante para el desarrollador tener esta información a la mano cuando crea las aplicaciones.

#### Sintaxis

```
Alter Table NombreTabla  
  Modify ( NombreColumna [NOT] NULL );
```

### Ejemplo

En la tabla alumno, la columna **Telefono** no debe aceptar valores nulos.

#### Script 9. 8

```
SQL> Alter Table Alumno  
  2  Modify ( Telefono NOT NULL );  
  
Table altered.  
  
SQL> describe alumno  
Name                                         Null?    Type  
-----  
IDALUMNO                                    NOT NULL NUMBER(5)  
NOMALUMNO                                   NOT NULL VARCHAR2(40)  
DIRECCIÓN                                  NOT NULL VARCHAR2(40)  
TELEFONO                                    NOT NULL VARCHAR2(15)
```

Si queremos insertar un alumno tendríamos que ingresar datos para todas las columnas.

#### Script 9. 9

```
SQL> insert into alumno  
  2  values(10001, 'Ricardo Marcelo', 'Ingeniería', NULL);  
insert into alumno  
*  
ERROR at line 1:  
ORA-01400: cannot insert NULL into ("EGCC"."ALUMNO"."TELEFONO")
```

El mensaje de error claramente nos indica que no se puede insertar valores nulos en la columna TELEFONO, de la tabla ALUMNO, que se encuentra en el esquema EGCC.

## Restricción Unique (Valores Únicos)

En muchos casos debemos garantizar que los valores de una columna ó conjunto de columnas de una tabla acepten solo valores únicos.

### Sintaxis

```
Alter Constraint NombreTabla  
Add Constraint U_NombreTabla_NombreColumna  
Unique ( Columna1, Columna2, . . . );
```

## Ejemplo

No puede haber dos alumnos con nombres iguales.

### Script 9.10

```
SQL> Alter Table alumno  
2 Add Constraint U_Alumno_NomAlumno  
3 Unique (NomAlumno);  
  
Table altered.
```

Para probar la restricción insertemos datos.

### Script 9.11

```
SQL> Insert Into Alumno  
2 Values( 10001, 'Sergio Matsukawa', 'San Miguel', '456-3456' );  
  
1 row created.  
  
SQL> Insert Into Alumno  
2 Values( 10002, 'Sergio Matsukawa', 'Los Olivos', '521-3456' );  
Insert Into Alumno  
*  
ERROR at line 1:  
ORA-00001: unique constraint (EGCC.U_ALUMNO_NOMALUMNO) violated
```

El mensaje de error del segundo insert nos indica que esta violando el constraint de tipo unique de nombre U\_ALUMNO\_NOMALUMNO en el esquema EGCC.

## Restricción Check (Reglas de Validación)

Las reglas de validación son muy importantes por que permiten establecer una condición a los valores que debe aceptar una columna.

### Sintaxis

```
Alter Table NombreTabla  
Add Constraint CK_NombreTable_NombreColumna  
Check ( Condición );
```

### Ejemplo

El precio de un curso no puede ser cero, ni menor que cero.

#### Script 9.12

```
SQL> Alter Table Curso  
2 Add Constraint CK_Curso_PreCurso  
3 Check ( PreCurso > 0 );  
  
Table altered.
```

Probemos el constraint ingresando datos.

#### Script 9.13

```
SQL> Insert Into Curso  
2 Values( 'C002', 'Asp.NET', 20, 7, 'Ricardo Marcelo', -400.00 );  
Insert Into Curso  
*  
ERROR at line 1:  
ORA-02290: check constraint (EGCC.CK_CURSO_PRECURSO) violated
```

Al intentar ingresar un curso con precio negativo, inmediatamente nos muestra el mensaje de error indicándonos que se está violando la regla de validación.

## Asignar Privilegios a Usuarios

Si queremos que otros usuarios puedan operar los objetos de un esquema, debemos darle los privilegios adecuadamente.

### Sintaxis

```
Grant Privilegio On Objeto To Usuario;
```

### Ejemplo

Por ejemplo, el usuario scott necesita consultar la tabla curso.

#### Script 9.14

```
SQL> Grant Select On Curso To Scott;

Grant succeeded.
```

Ahora hagamos la prueba respectiva.

#### Script 9.15

```
SQL> connect scott/tiger
Connected.

SQL> select * from egcc.curso;
```

IDCU	NOMCURSO	VACANTES	MATRICULADOS	PROFESOR	PRECURSO
C001	Oracle 9i - Nivel Inicial	20	10	Gustavo Coronel	350





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 04 Fundamentos de PL/SQL**

### **Contenido**

Introducción a PL/SQL

Tipos de Datos y Variables

## Introducción a PL/SQL

### Bloque

Unidad básica de programación.

### Bloque anónimo

#### Sintaxis

```
Declare
    -----
    -----
Begin
    -----
    -----
End;
```

#### Script 1

```
Declare
    sFecha Varchar2(40);
Begin
    select to_char(sysdate,'dd/mm/yyyy hh24:mm:ss')
        into sFecha from dual;
    dbms_output.put_line( 'Hoy es: ' || sFecha );
End;
```

## Función

### Sintaxis

```
Create Or Replace Function NombreFuncion( Parámetros ) Return  
TipoSalida  
Is  
    -----  
    -----  
Begin  
    -----  
    -----  
End;
```

### Script 2

```
create or replace function fnSuma( a number, b number ) return number  
is  
    c number;  
begin  
    c := a + b;  
    return c;  
end;
```

### Ejecución:

```
SQL> select fnSuma(12,25) from dual;  
  
FNSUMA(12,25)  
-----  
                37
```

## Procedimiento

### Sintaxis

```
Create Or Replace Procedure NombreProcedimiento( Parámetros )  
Is  
    -----  
    -----  
Begin  
    -----  
    -----  
End;
```

### Script 3

```
create or replace procedure prSuma( a number, b number )  
is  
    c number;  
begin  
    c := a + b;  
    dbms_output.put_line( c );  
end;
```

### Ejecución:

```
SQL> begin  
2     prSuma(15,15);  
3 end;  
4 /  
30  
  
PL/SQL procedure successfully completed.
```

## Tipos de Datos y Variables

### Tipos de Datos

Clase	Tipos de Datos
Escalares	Long, Number, Date, Char, Varchar2, Boolean
Compuestos	Record, Table
LOB	BFile, CLob, BLob
Referencia	Ref Cursor

### Declaración de Variables

```
Nombre_Variable Tipo [CONSTANT] [NOT NULL] [:=valor];
```

#### Script 4

```
Declare  
  v_Descripcion varchar2(50);  
  v_NroAsiento  number := 35;  
  v_Contador    binary_integer := 0;
```

### Asignar Valores a Variables

#### Caso 1: Sentencia de asignación

##### Script 5

```
v_NroAsiento := 78;  
v_Descuento := fnSuma(18,16);
```

#### Caso 2: Utilizando la sentencia SELECT

##### Script 6

```
select count(1) into v_Emps from emp;
```

### Script 7

Procedimiento para consultar el nombre de un empleado.

```
create or replace procedure pr101(p_empno number)
is
  v_ename varchar2(10);
begin
  select ename into v_ename from emp where empno = p_empno;
  dbms_output.put_line(v_ename);
end;
```

Ejecución:

```
SQL> execute pr101(7499);
ALLEN

PL/SQL procedure successfully completed.
```

## %Type

Permite declarar variables del mismo tipo de una columna.

### Script 8

Función para consultar el nombre de un departamento.

```
create or replace function fn101(p_deptno dept.deptno%type)
return dept.dname%type
is
  v_dname dept.dname%type;
begin
  select dname into v_dname from dept where deptno = p_deptno;
  return(v_dname);
end;
```

Ejecución:

```
SQL> select fn101(10) from dual;

FN101(10)
-----
ACCOUNTING
```



**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 05 Estructuras de Control**

### **Contenido**

Estructuras Selectivas

Bucles

Otros Elementos de Control

## Estructuras Selectivas

### Estructura IF

#### Caso 1:

```
if (condicion) then
    -----
    -----
    -----
end if
```

#### Script 1

Función para encontrar el mayor de 3 números.

```
create or replace function fn102 (n1 number, n2 number, n3 number) return
number
is
    mayor number := 0;
begin
    if (n1>mayor) then
        mayor := n1;
    end if;
    if (n2>mayor) then
        mayor := n2;
    end if;
    if (n3>mayor) then
        mayor := n3;
    end if;
    return mayor;
end;
```

#### Ejecución:

```
SQL> select fn102(12,45,4) from dual;

FN102(12,45,4)
-----
              45
```



### Caso 2:

```
if (condicion) then
    -----
    -----
else
    -----
    -----
end if
```

### Script 2

Función para encontrar el mayor de 3 números.

```
create or replace function fn103 (n1 number, n2 number, n3 number) return
number
is
    mayor number;
begin
    if (n1>n2) then
        mayor := n1;
    else
        mayor := n2;
    end if;
    if (n3>mayor) then
        mayor := n3;
    end if;
    return mayor;
end;
```

### Ejecución:

```
SQL> select fn103(-4,-8,-23) from dual;

FN103(-4,-8,-23)
-----
                -4
```

### Caso 3:

```
if (condicion1) then
    -----
    -----

elsif (condicion2) then
    -----
    -----

elsif (condicion3) then
    -----
    -----

. . .
[else
    -----
    -----]

end if;
```

### Script 3

Clasificar el salario de un empleado.

```
create or replace function fn104 (p_empno emp.empno%type) return varchar2
is
    v_sal emp.sal%type; v_msg varchar2(30);
begin
    select sal into v_sal from emp where empno = p_empno;
    if (v_sal<2500) then
        v_msg := 'Salario Bajo';
    elsif (v_sal<4000) then
        v_msg := 'Salario Regular';
    else
        v_msg := 'Salario Bueno';
    end if;
    v_msg := to_char(v_sal) || ' - ' || v_msg;
    return v_msg;
end;
```

### Ejecución:

```
SQL> select fn104(7698) from dual;

FN104(7698)
-----
2850 - Salario Regular
```

### Estructura Case

#### Caso 1:

```
case selector
  when valor1 then
    -----
    -----
  when valor2 then
    -----
    -----
  . . .
[else
  -----
  -----]
end case;
```

#### Script 4

Función para evaluar un número.

```
create or replace function fn105(n number) return varchar2
is
  rpta varchar2(30);
begin
  case n
    when 1 then
      rpta := 'Uno';
    when 2 then
      rpta := 'Dos';
    else
      rpta := 'None';
  end case;
  return rpta;
end;
```

Ejecución:

```
SQL> select fn105(1) from dual;

FN105(1)
-----
Uno
```

### Caso 2:

```
case
  when condicion1 then
    -----
    -----
  when condicion2 then
    -----
    -----
  . . .
  [else
    -----
    -----]
end case;
```

### Script 5

Realizar una función para evaluar el salario de un empleado.

```
create or replace function fn106(v_empno emp.empno%type) return varchar2 is
  v_msg varchar2(40);
  v_sal emp.sal%type;
begin
  select sal into v_sal from emp where empno = v_empno;
  case
    when (v_sal > 0 and v_sal <= 2500) then
      v_msg := 'Salario Bajo';
    when (v_sal > 2500 and v_sal <= 4000) then
      v_msg := 'Salario Regular';
    when (v_sal > 4000) then
      v_msg := 'Salario Bueno';
    else
      v_msg := 'Caso Desconocido';
  end case;
  v_msg := to_char(v_sal) || ' - ' || v_msg;
  return v_msg;
end;
```

### Ejecución:

```
SQL> select fn106(7782) from dual;
```

```
FN106(7782)
```

```
-----
2450 - Salario Bajo
```

## Bucles

### Objetos previos

#### Secuencia: sqtest

##### Script 6

```
SQL> create sequence sqtest;  
  
Sequence created.
```

#### Tabla Temporal

##### Script 7

```
SQL> create global temporary table test (  
2   id number primary key,  
3   dato varchar2(30)  
4 ) on commit preserve rows;  
  
Table created.
```

##### Nota

Las tablas temporales almacenan datos temporalmente, la opción **ON COMMIT PRESERVE ROWS** borra las filas de datos al cerrar la sesión, y la opción **ON COMMIT DELETE ROWS** borra los datos al ejecutar **COMMIT**.

### Bucle Simple: LOOP

#### Sintaxis:

```
loop
  -----
  -----
  if (Condición) then
    -----
    -----
    exit;
  end if;
  -----
  -----
  exit when (condición);
  -----
  -----
end loop;
```

#### Script 8

Función para encontrar el factorial de un número.

```
create or replace function fn107 (n number) return number
is
  f number := 1;
  cont number := n;
begin
  loop
    f := f * cont;
    cont := cont - 1;
    exit when (cont=0);
  end loop;
  return f;
end;
```

#### Ejecución:

```
SQL> select fn107(5) from dual;

FN107(5)
-----
      120
```

## Bucle: While

### Sintaxis

```
while (condicion) loop
    -----
    -----
end loop;
```

### Script 9

Insertar datos en una tabla.

```
create or replace procedure pr102 (n number)
is
    k number := 0;
begin
    while (k<n) loop
        insert into test( id,dato )
            values( sqtest.nextval, 'Gustavo Coronel' );
        k := k + 1;
    end loop;
    commit;
    dbms_output.put_line('Proceso Ejecutado');
end;
```

Ejecución:

```
SQL> execute pr102(15);
Proceso Ejecutado

PL/SQL procedure successfully completed.
```

Verificar el contenido de la tabla:

```
SQL> select * from test;

   ID DATO
-----
    1 Gustavo Coronel
    2 Gustavo Coronel
    3 Gustavo Coronel
    4 Gustavo Coronel
    5 Gustavo Coronel
    6 Gustavo Coronel
    7 Gustavo Coronel
    8 Gustavo Coronel
    9 Gustavo Coronel
```

```
10 Gustavo Coronel
11 Gustavo Coronel
12 Gustavo Coronel
13 Gustavo Coronel
14 Gustavo Coronel
15 Gustavo Coronel
```

15 rows selected.

## Bucle: FOR

### Sintaxis

```
for contador in [reverse] limite_inferior .. limite_superior loop
    -----
    -----
end loop;
```

### Script 10

Calcular el factorial de un número.

```
create or replace function fn108 ( n number ) return number
is
    f number := 1;
begin
    for k in 1 .. n loop
        f := f * k;
    end loop;
    return f;
end;
```

Ejecución:

```
SQL> select fn108(5) from dual;

FN108(5)
-----
      120
```



### Script 11

Ilustrar que no es necesario declarar el contador del for.

```
create or replace procedure pr103 ( n number, msg varchar2 )
is
  k number := 1000;
begin
  for k in 1 .. n loop
    dbms_output.put_line( k || ' - ' || msg );
  end loop;
  dbms_output.put_line( 'k = ' || k );
end;
```

Ejecución:

```
SQL> execute pr103(5, 'ALIANZA CAMPEON');
1 - ALIANZA CAMPEON
2 - ALIANZA CAMPEON
3 - ALIANZA CAMPEON
4 - ALIANZA CAMPEON
5 - ALIANZA CAMPEON
k = 1000

PL/SQL procedure successfully completed.
```

### Script 12

Aplicación de reverse. Tabla de multiplicar.

```
create or replace procedure pr104 ( n number )
is
  cad varchar2(30);
begin
  for k in reverse 1 .. 12 loop
    cad := n || ' x ' || k || ' = ' || (n*k);
    dbms_output.put_line( cad );
  end loop;
end;
```

Ejecución:

```
SQL> execute pr104(5);  
5 x 12 = 60  
5 x 11 = 55  
5 x 10 = 50  
5 x 9 = 45  
5 x 8 = 40  
5 x 7 = 35  
5 x 6 = 30  
5 x 5 = 25  
5 x 4 = 20  
5 x 3 = 15  
5 x 2 = 10  
5 x 1 = 5
```

```
PL/SQL procedure successfully completed.
```

## Otros Elementos de Control

### Etiquetas

#### Crear una etiqueta:

```
<<nombre_etiqueta>>
```

#### Saltar a una etiqueta:

```
goto nombre_etiqueta;
```

### Script 13

Elevar un número a una potencia.

```
create or replace function fn109( b number, p number ) return number
is
  r number := 1;
  k number := 0;
begin
  loop
    k := k + 1;
    r := r * b;
    if (k=p) then
      goto fin;
    end if;
  end loop;
  <<fin>>
  return r;
end;
```

#### Ejecución:

```
SQL> select fn109(-5,3) from dual;

FN109(-5,3)
-----
        -125
```

### Restricciones

1. No se puede realizar un salto al interior de un if, bucle, ó bloque interno.
2. No se puede saltar de una cláusula if a otra, en la misma instrucción if.
3. No se puede saltar de un bloque de excepciones al bloque de instrucciones.

### Etiquetando los Bucles

#### Formato

```
<<abc>>
for ...
    -----
    -----
    <<xyz>>
    for ...
        -----
        -----
        if ...
            exit abc;
        end if;
    end loop xyz;
    -----
    -----
end loop abc;
```

### Instrucción NULL

#### Formato

```
if (condición) then
    null;
else
    -----
    -----
end if;
```

#### Script 14

Desarrollar una función para determinar si número es impar.

```
create or replace function fn110( n number )
return varchar2
is
    rtn varchar2(30) := '';
begin
    if ( mod(n,2) = 0 ) then
        null;
    else
        rtn := n || ' es impar';
    end if;
    return rtn;
end;
```

#### Ejecución:

```
SQL> select fn110(15) from dual;

FN110(15)
-----
15 es impar
```





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 06 Estructuras de Datos**

### **Contenido**

- Introducción
- Registros
- Colecciones

## Introducción

---

Oracle provee dos Tipos de Datos Compuestos (Composite Data Types), estos son: Los Registros y Las Colecciones.

Un registro está compuesto por una serie de datos de diferentes, pero relacionados entre sí, muy semejante a una fila de una tabla.

Las Colecciones de PL/SQL son como los arreglos en otros Lenguajes de Programación como: C, C++ y Java. Por lo cual las podemos definir como un conjunto de datos homogéneos almacenados en forma consecutiva en memoria.

Al decir que son un conjunto de datos homogéneos se entiende que todos los valores deben ser del mismo tipo de dato, pero es prudente tener presente que dicho tipo de dato puede ser un tipo de dato compuesto, o sea, una Colección de otra Colección o de un Registro.

En síntesis, usamos las Colecciones de PL/SQL cuando queremos almacenar una lista de valores del mismo tipo de dato.

Por ejemplo, una lista con los nombres (VARCHAR2) de los empleados de la tabla EMPLOYEES. De igual manera, podrías tener una lista con todos los campos de la tabla COUNTRIES, en este caso definirías un Tipo Record equivalente a COUNTRIES%ROWTYPE y posteriormente una Colección de dicho tipo.

Al igual que los Tipos de Datos Escalares (VARCHAR2, NUMBER, etc), las colecciones pueden ser usadas como parámetros de entrada y/o salida de Procedimientos y Funciones, así como también pueden ser el valor de retorno de una función.



## Registros

### Definición

#### Sintaxis:

```
type tipo_registro is record (  
    campo1 tipo1 [not null] [:= valor1],  
    campo2 tipo2 [not null] [:= valor2],  
    -----  
    -----  
);
```

### Declaración de variable

#### Sintaxis:

```
nombre_variable tipo_registro;
```

### Acceso a los campos

#### Sintaxis:

```
nombre_variable.nombre_campo
```

### Script 1

Consultar el nombre y salario de un empleado.

```
create or replace procedure pr105( cod emp.empno%type )
is
  type reg is record (
    nombre emp.ename%type,
    salario emp.sal%type
  );
  r reg;
begin
  select ename, sal into r
  from emp where empno = cod;
  dbms_output.put_line( 'Nombre: ' || r.nombre );
  dbms_output.put_line( 'Salario: ' || r.salario );
end;
```

Ejecución:

```
SQL> execute pr105( 7698 );
Nombre: BLAKE
Salario: 2850

PL/SQL procedure successfully completed.
```

### %RowType

Se utiliza para declarar registros con la misma estructura de una tabla.

#### Sintaxis:

```
NombreVariable NombreTable%RowType;
```

### Script 2

Consultar los datos de un departamento.

```
create or replace procedure pr106( cod dept.deptno%type )
is
  r dept%rowtype;
begin
  select * into r
  from dept where deptno = cod;
  dbms_output.put_line('Codigo: ' || r.deptno);
  dbms_output.put_line('Nombre: ' || r.dname);
  dbms_output.put_line('Localización: ' || r.loc);
end;
```

#### Ejecución:

```
SQL> execute pr106(10);
Codigo: 10
Nombre: ACCOUNTING
Localización: NEW YORK

PL/SQL procedure successfully completed.
```

## Colecciones

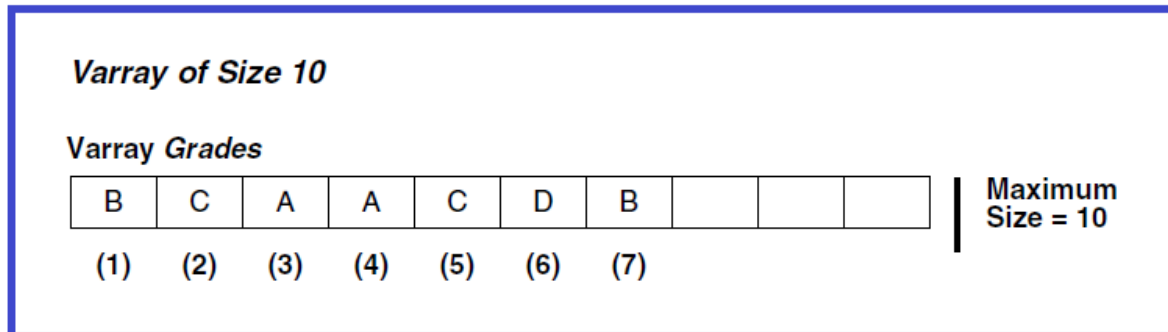
### Métodos para Colecciones

METODO	DESCRIPCIÓN.
<b>COUNT</b>	Devuelve el número de elementos que contiene la Colección. En el caso de los VARRAYs, el valor de COUNT siempre es igual a LAST. Para las Tablas Anidadas y las Matrices Asociativas, COUNT es normalmente igual a LAST. A menos que se eliminen elementos del centro, si esto ocurre, COUNT sería menor que LAST.
<b>DELETE</b>	Solo puede ser usado con Tablas Anidadas y Matrices Asociativas. Tiene Tres variantes: DELETE Elimina todos los elementos de una Colección. DELETE(n) Elimina el elemento <b>n</b> . Si <b>n</b> es nulo, no hace nada. DELETE(m, n) Elimina todos los elementos del rango <b>m..n</b> . Si <b>m</b> es mayor que <b>n</b> , o si <b>m</b> o <b>n</b> es nulo no hace nada.
<b>EXISTS</b>	EXISTS(n) devuelve TRUE si existe el elemento <b>n</b> de una Colección, de lo contrario, devuelve FALSE. Puede utilizar EXISTS para evitar una excepción al hacer referencia a un elemento inexistente. Cuando se le pasa un subíndice fuera de rango, EXISTS devuelve FALSE en lugar de lanzar la excepción SUBSCRIPT_OUTSIDE_LIMIT.
<b>EXTEND</b>	Es un método exclusivo de las Tablas Anidadas y Los VARRAYs. Este procedimiento tiene tres formas: EXTEND Agrega un elemento nulo. EXTEND(n) Agrega <b>n</b> elementos nulos. EXTEND(n, i) Agrega <b>n</b> copias del elemento <b>i</b> . EXTEND opera sobre el tamaño interno de una Colección. Si EXTEND encuentra elementos eliminados, los incluye en su recuento.
<b>FIRST</b>	FIRST retorna el primer valor (más pequeño) de subíndice en una colección. Los valores de subíndice suelen ser enteros, pero en el caso de las Matrices Asociativas también pueden ser cadenas. Si la Colección está vacía, FIRST retorna NULL. Para Los VARRAYs, FIRST siempre retorna 1.
<b>LAST</b>	LAST retorna el último valor (más grande) de subíndice en una Colección. Al igual que FIRST, si la Colección está vacía, LAST retorna NULL. Para Los VARRAYs, LAST siempre es igual a COUNT. Para las Tablas Anidadas y las Matrices Asociativas, LAST es normalmente igual a COUNT. A menos que se eliminen elementos del centro, si esto ocurre, LAST sería mayor que COUNT.
<b>PRIOR(n)</b>	Retorna el número de índice que antecede al índice <b>n</b> .
<b>NEXT(n)</b>	Retorna el número de índice que sigue (sucede) al índice <b>n</b> .

<b>LIMIT</b>	<p>Para las Tablas Anidadas y las Matrices Asociativas (No tienen un tamaño máximo), LIMIT retorna NULL.</p> <p>Para los VARRAYs, LIMIT retorna el número máximo de elementos que un VARRAY puede contener (Especificado en la definición de tipo).</p>				
<b>TRIM</b>	<p>Es un método exclusivo de las Tablas Anidadas y los VARRAYs.</p> <p>Este procedimiento tiene dos formas:</p> <table> <tr> <td>TRIM</td><td>Elimina un elemento del final de una Colección.</td></tr> <tr> <td>TRIM(n)</td><td>Elimina <b>n</b> elementos del final de una Colección. Si <b>n</b> es mayor que COUNT, TRIM(n) lanza la excepción SUBSCRIPT_BEYOND_COUNT.</td></tr> </table> <p>TRIM opera en el tamaño interno de una Colección.</p> <p>Si TRIM encuentra elementos eliminados, los incluye en su recuento.</p>	TRIM	Elimina un elemento del final de una Colección.	TRIM(n)	Elimina <b>n</b> elementos del final de una Colección. Si <b>n</b> es mayor que COUNT, TRIM(n) lanza la excepción SUBSCRIPT_BEYOND_COUNT.
TRIM	Elimina un elemento del final de una Colección.				
TRIM(n)	Elimina <b>n</b> elementos del final de una Colección. Si <b>n</b> es mayor que COUNT, TRIM(n) lanza la excepción SUBSCRIPT_BEYOND_COUNT.				

## Los VARRAYs

Entendiendo los ARRAYs:



### Sintaxis:

```
TYPE NuevoTipoDeDato IS VARRAY(Tamaño) OF TipoDeDato;
```

### Script 3

Ejemplo ilustrativo de cómo usar VARRAYs.

```
DECLARE
    -- Definimos los tipos de datos
    TYPE AlumnosArray IS VARRAY(5) OF VARCHAR2(100);
    TYPE NotasArray IS VARRAY(5) OF NUMBER(4);
    -- Definiendo las variables
    alumnos AlumnosArray;
    notas NotasArray;
BEGIN
    -- Creando los arreglos
    alumnos := AlumnosArray('Gustavo','Lucero','Ricardo','Andrea','Laura');
    notas := NotasArray(20,18,16,10,15);
    -- Mostrando los arreglos
    FOR i IN 1 .. alumnos.count LOOP
        dbms_output.PUT_LINE( alumnos(i) || ' - ' || notas(i) );
    END LOOP;
END;
```

Ejecución:

```
Gustavo - 20
Lucero - 18
Ricardo - 16
Andrea - 10
Laura - 15
```

### Script 4

Segundo ejemplo de cómo usar VARRAYs:

```
DECLARE
    -- Definimos los tipos de datos
    TYPE VARRAY_EMPLEADOS IS VARRAY(5000) OF HR.EMPLOYEES%ROWTYPE;
    -- Definiendo las variables
    V_EMPLEADOS VARRAY_EMPLEADOS;
    V_CONT NUMBER(8);
BEGIN
    V_EMPLEADOS := VARRAY_EMPLEADOS();
    DBMS_OUTPUT.PUT_LINE('TAMAÑO INICIAL: ' || V_EMPLEADOS.COUNT);
    FOR REC IN (SELECT * FROM HR.EMPLOYEES) LOOP
        V_EMPLEADOS.EXTEND;
        V_CONT := V_EMPLEADOS.COUNT;
        V_EMPLEADOS(V_CONT) := REC;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('TAMAÑO FINAL: ' || V_EMPLEADOS.COUNT);
    FOR I IN V_EMPLEADOS.FIRST..V_EMPLEADOS.LAST LOOP
        DBMS_OUTPUT.PUT_LINE('I || '.- ' || V_EMPLEADOS(I).FIRST_NAME);
    END LOOP;
END;
```

Ejecución:

```
TAMAÑO INICIAL: 0
TAMAÑO FINAL: 107
1.- Steven
2.- Neena
. . .
106.- Shelley
107.- William
```

### Matrices Asociativas (Associative Array)

#### Características:

- Son tablas exclusivas de PL/SQL, esto significa que pueden existir en estructuras de memoria de PL/SQL (Paquetes, Funciones, Procedimientos etc.) pero no pueden ser creadas como objetos / columnas de Base de Datos.
- Están compuestas de dos columnas a las cuales no es posible asignarles nombres:
  - ✓ La primera columna es el índice (index).
  - ✓ La segunda columna contiene el dato almacenado (value).
  - ✓ El índice es usado para localizar el dato almacenado en la segunda columna.
  - ✓ Los valores del índice pueden ser tanto negativos como positivos y no necesariamente tienen que ser insertados de forma secuencial o consecutiva, esto es, puede agregar el índice 4 antes que el 3.
- No son inicializadas al momento de su declaración.
- Tienen un tamaño dinámico, por lo cual pueden crecer tanto como sea necesario.

#### Sintaxis:

```
TYPE nuevo_tipo_dato IS TABLE OF
{
    column_type
    | variable%TYPE
    | table.column%TYPE} [NOT NULL]
    | table%ROWTYPE
}
INDEX BY {PLS_INTEGER|BINARY_INTEGER|VARCHAR2 (<tamaño>) ;
```



### Script 5

En el presente ejemplo se ilustra de una manera sencilla el uso de matrices asociativas:

```
DECLARE
  TYPE ARRAY_NOTAS IS TABLE OF NUMBER
  INDEX BY BINARY_INTEGER;
  NOTAS ARRAY_NOTAS;
BEGIN
  -- CARGAR NOTAS
  NOTAS(1) := 20;
  NOTAS(2) := 18;
  NOTAS(3) := 15;
  NOTAS(4) := 17;
  -- MOSTRAR NOTAS
  FOR I IN 1..NOTAS.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('NOTA ' || I || ': ' || NOTAS(I));
  END LOOP;
END;
```

### Ejecución:

```
NOTA 1: 20
NOTA 2: 18
NOTA 3: 15
NOTA 4: 17
```

### Script 6

Segundo ejemplo de arreglo asociativo.

```
DECLARE
  -- Definimos el tipo de dato
  TYPE ARRAY_EMPLEADOS IS TABLE OF HR.EMPLOYEES%ROWTYPE
  INDEX BY BINARY_INTEGER;
  -- Definiendo las variables
  V_EMPLEADOS ARRAY_EMPLEADOS;
BEGIN
  DBMS_OUTPUT.PUT_LINE('TAMAÑO INICIAL: ' || V_EMPLEADOS.COUNT);
  FOR REC IN (SELECT * FROM HR.EMPLOYEES) LOOP
    V_EMPLEADOS(V_EMPLEADOS.COUNT + 1) := REC;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('TAMAÑO FINAL: ' || V_EMPLEADOS.COUNT);
  FOR I IN V_EMPLEADOS.FIRST..V_EMPLEADOS.LAST LOOP
    DBMS_OUTPUT.PUT_LINE( I || '.- ' || V_EMPLEADOS(I).FIRST_NAME);
  END LOOP;
END;
```

Ejecución:

```
TAMAÑO INICIAL: 0
TAMAÑO FINAL: 107
1.- Steven
2.- Neena
. . .
106.- Shelley
107.- William
```

### Tablas Anidadas (Nested Table)

#### Características:

- Pueden ser declaradas en bloques de PL/SQL, así como también en la Base de Datos.
- Al igual que las Matrices Asociativas, las Tablas Anidadas tienen un tamaño dinámico y puede contener elementos vacíos, o sea, sus Índices no tiene que ser consecutivos.
- Deben ser Inicializadas antes de ser usadas. Una variable de Tabla Anidada no inicializada es una colección nula.
- Para ser inicializadas es necesario hacer uso de su constructor. Dicho Constructor es una función con el mismo nombre que el **Tipo Colección**, el cual devuelve una colección de ese tipo.
- A diferencia de las Matrices Asociativas, las Tablas Anidadas no pueden contener índices negativos.
- Es importante que tengas en cuenta que, aunque se hace referencia a la primera columna como **INDICE**, las Tablas Anidadas no tienen índices, más bien es una columna con números.

#### Sintaxis:

```
TYPE nuevo_tipo_dato IS TABLE OF
{
    column_type
    | variable%TYPE
    | table.column%TYPE} [NOT NULL]
    | table%ROWTYPE
}
```

### Script 7

Ejemplo simple de una tabla anidada de tipo VARCHAR2.

```
DECLARE
  TYPE tabla_varchar2 IS TABLE OF VARCHAR2(100);
  empleados  tabla_varchar2 := tabla_varchar2();
BEGIN
  -- Tamaño Inicial
  DBMS_OUTPUT.PUT_LINE('Tamaño Inicial: ' || empleados.COUNT);
  -- Se añaden 4 elementos
  empleados.EXTEND (4);
  empleados (1) := 'Pepe';
  empleados (2) := 'Elena';
  empleados (3) := 'Carmen';
  empleados (4) := 'Juan';
  -- Se añade un elemento mas
  empleados.EXTEND;
  empleados (empleados.LAST) := 'Gustavo';
  -- Tamaño Final
  DBMS_OUTPUT.PUT_LINE('Tamaño Final: ' || empleados.COUNT);
  -- Mostrar lista
  FOR I IN 1 .. empleados.COUNT
  LOOP
    DBMS_OUTPUT.put_line ( empleados(I) );
  END LOOP;
END;
```

### Ejecución:

```
Tamaño Inicial: 0
Tamaño Final: 5
Pepe
Elena
Carmen
Juan
Gustavo
```

### Script 8

Segundo ejemplo de tablas anidadas.

```
DECLARE
  -- Definimos los tipo de datos
  TYPE TABLA_EMPLEADOS IS TABLE OF HR.EMPLOYEES%ROWTYPE;
  -- Definiendo las variables
  V_EMPLEADOS TABLA_EMPLEADOS;
BEGIN
  V_EMPLEADOS := TABLA_EMPLEADOS();
  DBMS_OUTPUT.PUT_LINE('TAMAÑO INICIAL: ' || V_EMPLEADOS.COUNT);
  FOR REC IN (SELECT * FROM HR.EMPLOYEES) LOOP
    V_EMPLEADOS.EXTEND;
    V_EMPLEADOS(V_EMPLEADOS.LAST) := REC;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('TAMAÑO FINAL: ' || V_EMPLEADOS.COUNT);
  FOR I IN V_EMPLEADOS.FIRST..V_EMPLEADOS.LAST LOOP
    DBMS_OUTPUT.PUT_LINE( I || '.- ' || V_EMPLEADOS(I).FIRST_NAME);
  END LOOP;
END;
```

Ejecución:

```
TAMAÑO INICIAL: 0
TAMAÑO FINAL: 107
1.- Steven
2.- Neena
. . .
106.- Shelley
107.- William
```





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 07 Tratamiento de Errores**

### **Contenido**

- Tipos de Errores
- Excepciones
- Esquema General
- Excepciones Predefinidas
- Instrucción RAISE
- Excepciones de Usuario
- Generación de Mensajes de Error

## Tipos de Errores

Tipo de Error	Quién Informa	Cómo es tratado
De Compilación	Compilador PL/SQL	Interactivamente: el compilador informa de los errores y el programador debe corregirlos.
De Ejecución	Motor de Ejecución PL/SQL	Programáticamente: las excepciones son generadas e interceptadas por las rutinas de tratamiento de excepciones.

## Excepciones

Es el mecanismo de tratamiento de errores en tiempo de ejecución. Tenemos dos tipos de excepciones:

- Definidas por el usuario
- Predefinidas

## Esquema General

**Formato:**

```
BEGIN

    < Bloque de instrucciones a controlar >

EXCEPTION
    WHEN nombre_excepción THEN
        secuencia_de_instrucciones;
    WHEN nombre_excepción THEN
        secuencia_de_instrucciones;
    [ WHEN OTHERS THEN
        secuencia_de_instrucciones; ]
END;
```



## Excepciones Predefinidas

Oracle ha definido diversas excepciones que corresponden con los errores Oracle más comunes.

Excepción	Descripción
INVALID_CURSOR	Ocurre cuando se hace referencia a un cursor que esta cerrado.
CURSOR_ALREADY_OPEN	Ocurre cuando se trata de abrir un cursor que ya esta abierto.
NO_DATA_FOUND	Ocurre cuando una sentencia SELECT no retorna ninguna fila.
TOO_MANY_ROWS	Ocurre cuando una sentencia SELECT retorna mas de una fila.
VALUE_ERROR	Ocurre cuando hay conflicto de tipos de datos.

### Script 1

Desarrollar un procedimiento para consultar el salario de un empleado.

```
create or replace procedure FindEmp( Cod Emp.EmpNo%Type )
is
    Salario Emp.Sal%Type;
Begin
    Select Sal Into Salario
    From Emp
    Where EmpNo = Cod;
    DBMS_Output.Put_Line( 'Salario: ' || Salario );
Exception
    When No_Data_Found Then
        DBMS_Output.Put_Line( 'Código no existe.' );
End;
```

### Ejecución:

```
SQL> exec FindEmp( 9999 );
Código no existe.

PL/SQL procedure successfully completed.
```

## Instrucción RAISE

Permite generar una excepción.

### Script 2

```
create or replace procedure UpdateSalEmp
(Codigo Emp.EmpNo%Type, Salario Emp.Sal%Type )
is
    Cont Number;
Begin
    Select Count(*) Into Cont
    From Emp
    Where EmpNo = Codigo;
    If (Cont=0) Then
        Raise No_Data_Found;
    End If;
    Update Emp
    Set Sal = Salario
    Where EmpNo = Codigo;
    Commit;
    DBMS_Output.Put_Line( 'Proceso OK' );
Exception
    When No_Data_Found Then
        DBMS_Output.Put_Line( 'Código no existe.' );
End;
```

### Ejecución:

```
SQL> exec UpdateSalEmp( 9999, 5000 );
Código no existe.

PL/SQL procedure successfully completed.
```

## Excepciones de Usuario

### Script 3

Desarrollar una segunda versión del procedimiento UpdateSalEmp, pero con una excepción de usuario.

```
create or replace procedure UpdateSalEmp2
( Codigo Emp.EmpNo%Type, Salario Emp.Sal%Type )
is
    Cont Number;
    Excepl Exception;
Begin
    Select Count(*) Into Cont
    From Emp
    Where EmpNo = Codigo;
    If (Cont=0) Then
        Raise Excepl;
    End If;
    Update Emp
    Set Sal = Salario
    Where EmpNo = Codigo;
    Commit;
    DBMS_Output.Put_Line( 'Proceso OK' );
Exception
    When Excepl Then
        DBMS_Output.Put_Line( 'Código no existe.' );
End;
```

### Ejecución:

```
SQL> exec UpdateSalEmp2( 9999, 5000 );
Código no existe.

PL/SQL procedure successfully completed.
```

## Generación de Mensajes de Error

Los mensajes de error que se generan con `RAISE_APPLICATION_ERROR` deben estar entre -20,999 y -20,000.

### Script 4

Desarrollar una tercera versión del procedimiento `UpdateSalEmp`, pero esta vez genere un mensaje de error.

```
create or replace procedure UpdateSalEmp3
( Codigo Emp.EmpNo%Type, Salario Emp.Sal%Type )
is
    Cont Number;
Begin
    Select Count(*) Into Cont
    From Emp
    Where EmpNo = Codigo;
    If (Cont=0) Then
        Raise_Application_Error( -20000, 'No existe empleado.' );
    End If;
    Update Emp
    Set Sal = Salario
    Where EmpNo = Codigo;
    Commit;
    DBMS_Output.Put_Line( 'Proceso OK' );
End;
```

### Ejecución:

```
SQL> exec UpdateSalEmp3( 9999, 5000 );
BEGIN UpdateSalEmp3( 9999, 5000 ); END;

*
ERROR at line 1:
ORA-20000: No existe empleado.
ORA-06512: at "SCOTT.UPDATESALEMP3", line 10
ORA-06512: at line 1
```

### Script 5

Otra versión del mismo procedimiento.

```
create or replace procedure UpdateSalEmp4
( Codigo Emp.EmpNo%Type, Salario Emp.Sal%Type )
is
    Cont Number;
Begin
    Select Count(*) Into Cont
    From Emp
    Where EmpNo = Codigo;
    If (Cont=0) Then
        Raise_Application_Error( -20000, 'No existe empleado.' );
    End If;
    Update Emp
    Set Sal = Salario
    Where EmpNo = Codigo;
    Commit;
    DBMS_Output.Put_Line( 'Proceso OK' );
Exception
    When Others Then
        dbms_output.put_line ( 'Error Nro. ORA' || to_char(sqlcode) );
        dbms_output.put_line ( sqlerrm );
End;
```

Ejecución:

```
SQL> exec UpdateSalEmp4( 9999, 5000 );
Error Nro. ORA-20000
ORA-20000: No existe empleado.

PL/SQL procedure successfully completed.
```





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 08 LENGUAJE SQL**

### **Contenido**

- Insertando Filas
- Modificando Datos
- Eliminando Filas
- Transacciones
- Propiedades de una Transacción
- Operación de Transacciones

## Insertando Filas

### Inserciones una Sola Fila

#### Script 1

```
SQL> connect hr/hr
Connected.

SQL> insert into
  2 departments(department_id, department_name, manager_id, location_id)
  3 values(300, 'Departamento 300', 100, 1800);

1 row created.

SQL> commit;
Commit complete.
```

### Insertando Filas con Valores Nulos

#### Script 2

**Método Implícito:** Se omiten las columnas que aceptan valores nulos.

```
SQL> insert into
  2 departments(department_id, department_name)
  3 values(301, 'Departamento 301');

1 row created.

SQL> commit;
Commit complete.
```



### Script 3

**Método Explicito:** Especificamos la palabra clave NULL en las columnas donde queremos insertar un valor nulo.

```
SQL> insert into departments
  2  values(302, 'Departamento 302', NULL, NULL);

1 row created.

SQL> commit;
Commit complete.
```

### Insertando Valores Especiales

#### Script 4

```
SQL> insert into employees (employee_id,
  2      first_name, last_name,
  3      email, phone_number,
  4      hire_date, job_id, salary,
  5      commission_pct, manager_id,
  6      department_id)
  7  values(250,
  8      'Gustavo', 'Coronel',
  9      'gcoronel@miempresa.com', '511.481.1070',
 10      sysdate, 'FI_MGR', 14000,
 11      NULL, 102, 100);

1 row created.

SQL> commit;
Commit complete.
```

## Insertando Valores Específicos de Fecha

### Script 5

```
SQL> insert into employees
  2 values(251, 'Ricardo', 'Marcelo',
  3       'rmarcelo@techsoft.com', '511.555.4567',
  4       to_date('FEB 4, 2005', 'MON DD, YYYY'),
  5       'AC_ACCOUNT', 11000, NULL, 100, 30);

1 row created.

SQL> commit;
Commit complete.
```

## Usando & Sustitución para el Ingreso de Valores

### Script 6

```
SQL> insert into
  2 departments (department_id, department_name, location_id)
  3 values (&department_id, '&department_name', &location_id);
Enter value for department_id: 3003
Enter value for department_name: Departamento 303
Enter value for location_id: 2800
old   3: values (&department_id, '&department_name', &location_id)
new   3: values (3003, 'Departamento 303', 2800)

1 row created.

SQL> commit;
Commit complete.
```

## Copiando Filas Desde Otra Tabla

### Script 7

```
SQL> create table test
2  (
3  id number(6) primary key,
4  name varchar2(20),
5  salary number(8,2)
6  );

Table created.

SQL> insert into test (id, name, salary)
2  select employee_id, first_name, salary
3  from employees
4  where department_id = 30;

7 rows created.

SQL> commit;
Commit complete.
```

## Insertando en Múltiples Tablas

### Script 8

Primero creamos las siguientes tablas: test50 y test80.

```
SQL> create table test50
2  (
3      id number(6) primary key,
4      name varchar2(20),
5      salary number(8,2)
6  );
```

Table created.

```
SQL> create table test80
2  (
3      id number(6) primary key,
4      name varchar2(20),
5      salary number(8,2)
6  );
```

Table created.

Luego limpiamos la tabla **test**.

```
SQL> delete from test;
7 rows deleted.
```

```
SQL> commit;
Commit complete.
```

Ahora procedemos a insertar datos en las tres tablas a partir de la tabla **employees**.

```
SQL> insert all
  2  when department_id = 50 then
  3      into test50 (id, name, salary)
  4      values(employee_id, first_name, salary)
  5  when department_id = 80 then
  6      into test80 (id, name, salary)
  7      values (employee_id, first_name, salary)
  8  else
  9      into test(id, name, salary)
 10      values(employee_id, first_name, salary)
 11  select department_id, employee_id, first_name, salary
 12  from employees;
```

109 rows created.

```
SQL> commit;
Commit complete.
```

## Modificando Datos

### Actualizando una Columna de una Tabla

#### Script 9

Incrementar el salario de todos los empleados en 10%.

```
SQL> update employees
  2  set salary = salary * 1.10;
```

109 rows updated.

```
SQL> Commit;
Commit complete.
```

### Seleccionando las Filas a Actualizar

#### Script 10

Ricardo Marcelo (Employee\_id=251) ha sido trasladado del departamento de Compras (Department\_id = 30) al departamento de Ventas (Department\_id = 80).

```
SQL> select employee_id, first_name, department_id, salary
  2  from employees
  3  where employee_id = 251;
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY
251	Ricardo	30	12100

```
SQL> update employees
  2  set department_id = 80
  3  where employee_id = 251;
```

1 row updated.

```
SQL> select employee_id, first_name, department_id, salary
  2  from employees
  3  where employee_id = 251;
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY
251	Ricardo	80	12100

```
SQL> commit;
Commit complete.
```

### Actualizando Columnas con Subconsultas

#### Script 11

Gustavo Coronel (Employee\_id = 250) ha sido trasladado al mismo departamento del empleado 203, y su salario tiene que ser el máximo permitido en su puesto de trabajo.

```
SQL> select employee_id, first_name, last_name,
  2  department_id, job_id, salary
  3  from employees
  4  where employee_id = 250;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID	JOB_ID	SALARY
250	Gustavo	Coronel	100	FI_MGR	15400

```
SQL> update employees
  2  set department_id = (select department_id from employees
  3                        where employee_id = 203),
  4  salary = (select max_salary from jobs
  5             where jobs.job_id = employees.job_id)
  6  where employee_id = 250;
```

1 row updated.

```
SQL> commit;
Commit complete.
```

```
SQL> select employee_id, first_name, last_name, department_id, job_id,
salary
  2  from employees
  3  where employee_id = 250;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID	JOB_ID	SALARY
250	Gustavo	Coronel	40	FI_MGR	16000

### Actualizando Varias Columnas con una Subconsulta

Asumiremos que tenemos la tabla **resumen\_dept**, con la siguiente estructura:

Columna	Tipo de Dato	Nulos	Descripción
Department_id	Number(4)	No	Código de Departamento.
Emps	Number(4)	Si	Cantidad de Empleados en el departamento.
Planilla	Number(10,2)	Si	Importe de la planilla en el departamento.

Esta tabla guarda la cantidad de empleados y el importe de la planilla por departamento.

#### Script 12

Este script crea la tabla **resumen\_det** e inserta los departamentos.

```
SQL> create table resumen_dept
2  (
3      department_id number(4) primary key,
4      emps number(4),
5      planilla number(10,2)
6  );

Table created.

SQL> insert into resumen_dept (department_id)
2  select department_id from departments;

31 rows created.

SQL> commit;
Commit complete.
```



### Script 13

Este script actualiza la tabla **resumen\_dept**.

```
SQL> update resumen_dept
  2  set (emps, planilla) = (select count(*), sum(salary)
  3      from employees
  4      where employees.department_id = resumen_dept.department_id);

31 rows updated.

SQL> commit;
Commit complete.
```

### Error de Integridad Referencial

#### Script 14

```
SQL> update employees
  2  set department_id = 55
  3  where department_id = 110;
update employees
*
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK) violated - parent key not found
```

## Eliminando Filas

### Eliminar Todas la Filas de una Tabla

#### Script 15

```
SQL> select count(*) from test;
```

```
  COUNT (*)  
-----  
          30
```

```
SQL> delete from test;
```

```
30 rows deleted.
```

```
SQL> commit;
```

```
Commit complete.
```

```
SQL> select count(*) from test;
```

```
  COUNT (*)  
-----  
          0
```

### Seleccionando las Filas a Eliminar

#### Creando una tabla de prueba

#### Script 16

```
SQL> create table copia_emp  
  2 as select * from employees;
```

```
Table created.
```

### Eliminando una sola fila

#### Script 17

```
SQL> delete from copia_emp
      2  where employee_id = 190;

1 row deleted.

SQL> commit;
Commit complete.
```

### Eliminando un grupo de filas

#### Script 18

```
SQL> delete from copia_emp
      2  where department_id = 50;

44 rows deleted.

SQL> commit;
Commit complete.
```

## Uso de Subconsultas

#### Script 19

Eliminar los empleados que tienen el salario máximo en cada puesto de trabajo.

```
SQL> delete from copia_emp
      2  where salary = (select max_salary from jobs
      3                    where jobs.job_id = copia_emp.job_id);

1 row deleted.

SQL> commit;

Commit complete.
```

## Error de Integridad Referencial

### Script 20

```
SQL> delete from departments
      2  where department_id = 50;
delete from departments
*
ERROR at line 1:
ORA-02292: integrity constraint (HR.EMP_DEPT_FK) violated - child record
found
```

## Truncando una Tabla

### Script 21

```
SQL> select count(*) from copia_emp;

COUNT(*)
-----
        64

SQL> truncate table copia_emp;

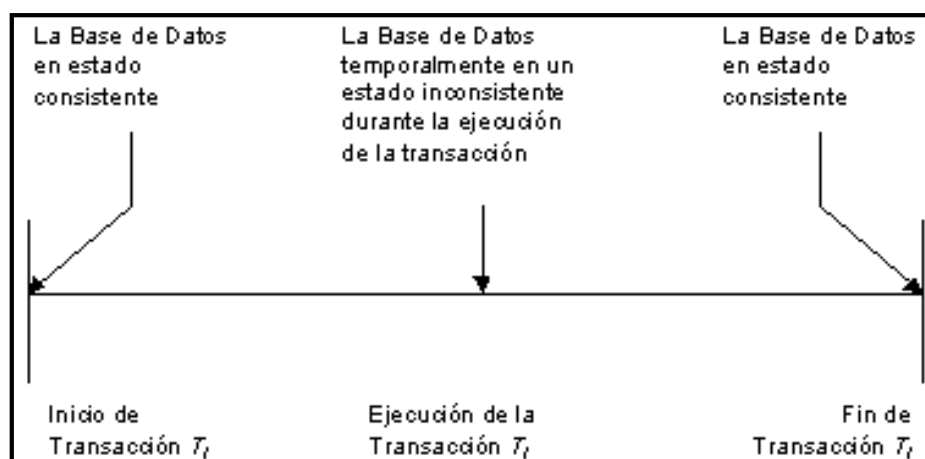
Table truncated.

SQL> select count(*) from copia_emp;

COUNT(*)
-----
         0
```

## Transacciones

Una **transacción** es un grupo de acciones que hacen transformaciones consistentes en las tablas preservando la consistencia de la base de datos. Una base de datos está en un estado *consistente* si obedece todas las restricciones de integridad definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones, y eliminaciones de información. Por supuesto, se quiere asegurar que la base de datos nunca entre en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al fin de la ejecución de una transacción.



Lo que se persigue con el manejo de transacciones es por un lado tener una transparencia adecuada de las acciones concurrentes a una base de datos y por otro lado tener una transparencia adecuada en el manejo de las fallas que se pueden presentar en una base de datos.

### Propiedades de una Transacción

Una transacción debe tener las propiedades ACID, que son las iniciales en inglés de las siguientes características: Atomicity, Consistency, Isolation, Durability.

#### Atomicidad

Una transacción constituye una unidad atómica de ejecución y se ejecuta exactamente una vez; o se realiza todo el trabajo o nada de él en absoluto.

#### Coherencia

Una transacción mantiene la coherencia de los datos, transformando un estado coherente de datos en otro estado coherente de datos. Los datos enlazados por una transacción deben conservarse semánticamente.

#### Aislamiento

Una transacción es una unidad de aislamiento y cada una se produce aislada e

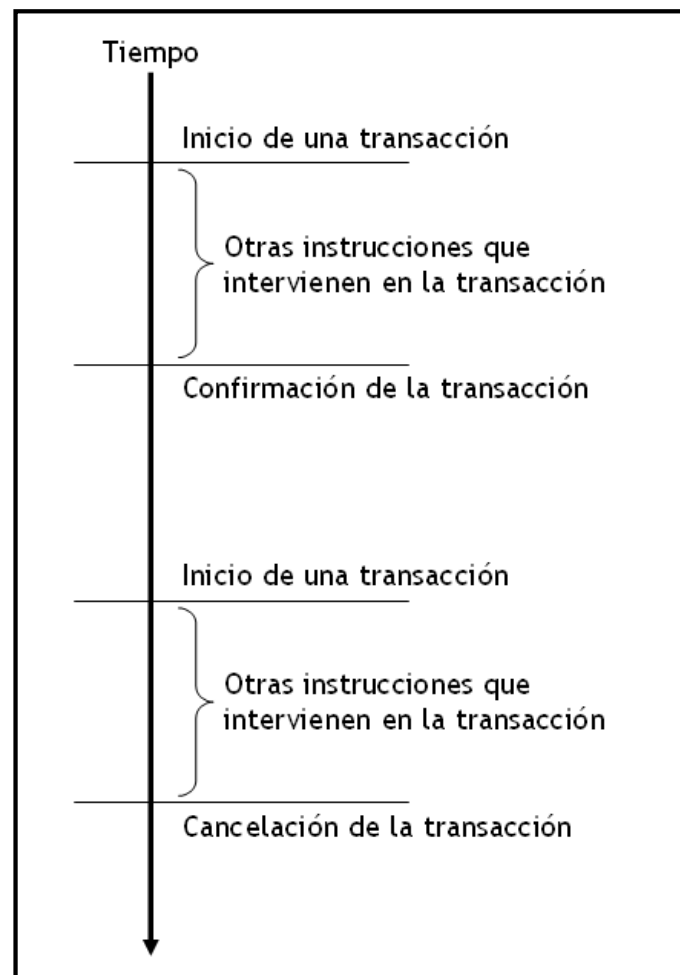
independientemente de las transacciones concurrentes. Una transacción nunca debe ver las fases intermedias de otra transacción.

## Durabilidad

Una transacción es una unidad de recuperación. Si una transacción tiene éxito, sus actualizaciones persisten, aun cuando falle el equipo o se apague. Si una transacción no tiene éxito, el sistema permanece en el estado anterior antes de la transacción.

## Operación de Transacciones

El siguiente gráfico ilustra el funcionamiento de una transacción, cuando es confirmada y cuando es cancelada.



## Inicio de una transacción

El inicio de una transacción es de manera automática cuando ejecutamos una sentencia **INSERT**, **UPDATE**, **DELETE** o **SELECT – FOR UPDATE**. La ejecución de cualquiera de estas

sentencias da inicio a una transacción. Las instrucciones que se ejecuten a continuación formaran parte de la misma transacción.

### Confirmación de una transacción

Para confirmar los cambios realizados durante una transacción utilizamos la sentencia **COMMIT**.

### Cancelar una transacción

Para cancelar los cambios realizados durante una transacción utilizamos la sentencia **ROLLBACK**.

### Script 22

Incrementar el salario al empleado Ricardo Marcelo (employee\_id = 251) en 15%.

```
SQL> select employee_id, salary
2   from employees
3  where employee_id = 251;
```

EMPLOYEE_ID	SALARY
251	12100

```
SQL> update employees
2   set salary = salary * 1.15
3  where employee_id = 251;
```

1 row updated.

```
SQL> select employee_id, salary
2   from employees
3  where employee_id = 251;
```

EMPLOYEE_ID	SALARY
251	13915

```
SQL> commit;
```

Commit complete.







**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 09 LENGUAJE SQL EN PL/SQL**

### Contenido

- Categorías
- SQL Dinámico
- Sentencia: SELECT
- Sentencia: INSERT
- Sentencia: UPDATE
- Sentencia: DELETE
- Nombres de Variables
- Cláusula Returning
- Referencias de Tablas
- Enlaces de Base de Datos
- Sinónimos

### Categorías

---

1. DML Lenguaje de Manipulación de Datos: Select, Insert, Update, Delete.
2. DDL Lenguaje de Definición de Datos: Create, Alter, Drop, Grant.
3. Control de Transacciones: Commit, Rollback.
4. Control de Sesiones: Alter Session.
5. Control del Sistema: Alter System.

### Uso de SQL en PL/SQL

Las categorías permitidas de SQL en PL/SQL son solamente la 1 y 3: DML y Control de Transacciones.

## SQL Dinámico

Permite ejecutar cualquier tipo de instrucción SQL desde PL/SQL.

### Script 1

```
create or replace procedure pr107( cmd varchar2)
is
begin
    execute immediate cmd;
end;
```

#### Ejecución 1:

```
SQL> exec pr107('create table t1 ( id number, dato varchar2(30) )');

PL/SQL procedure successfully completed.
```

#### Ejecución 2:

```
SQL> exec pr107('insert into t1 values( 1, ''Oracle is Powerful'' )');

PL/SQL procedure successfully completed.
```

#### Verificando la tabla t1:

```
SQL> select * from t1;

      ID DATO
-----
      1 Oracle is Powerful
```

## Sentencia: SELECT

### Sintaxis

```
Select columnas into variables/registro  
From NombreTabla  
Where condición;
```

### Script 2

Consultar la cantidad de empleados y el importe de la planilla de un departamento.

```
create or replace procedure pr108(cod dept.deptno%type)  
is  
    emps number;  
    planilla number;  
begin  
    select count(*), sum(sal) into emps, planilla  
        from emp  
        where deptno = cod;  
    dbms_output.put_line('Empleados: ' || emps);  
    dbms_output.put_line('Planilla: ' || planilla);  
end;
```

### Ejecución:

```
SQL> exec pr108( 20 );  
Empleados: 5  
Planilla: 10875  
  
PL/SQL procedure successfully completed.
```

## Sentencia: INSERT

### Sintaxis 1

```
Insert into NombreTabla[(columnas)] values(datos);
```

### Sintaxis 2

```
Insert into NombreTabla[(columns)] select ... ;
```

### Script 3

Procedimiento para registrar un nuevo departamento.

```
create or replace procedure pr109( cod number, nom varchar2, loc varchar2)
is
begin
    insert into dept values(cod, nom, loc);
    commit;
    dbms_output.put_line('Proceso OK');
end;
```

Ejecución:

```
SQL> exec pr109( 50, 'Deportes', 'Los Olivos' );
Proceso OK

PL/SQL procedure successfully completed.
```

Verificando tabla **dept**:

```
SQL> select * from dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	Deportes	Los Olivos

## Sentencia: UPDATE

### Sintaxis

```
Update Nombretabla
Set  columna1 = valor,
    (columna2, columna3, ...) = (sentencia_select),
    ...
where (condición);
```

### Script 4

Para este ejemplo crearemos la tabla **RESUMEN**, donde almacenaremos el número de empleados por departamento y el importe de su planilla.

Creación de la tabla resumen:

```
SQL> create table resumen(
2     deptno number(2) primary key,
3     emps number(2),
4     planilla number(10,2)
5 );
```

Table created.

Insertando los códigos de los departamentos:

```
SQL> insert into resumen(deptno)
2     select deptno from dept;
```

5 rows created.

Procedimiento para actualizar las columnas **emps** y **planilla**:

```
create or replace procedure pr110
is
begin
    update resumen
        set (emps,planilla) = (select count(*), sum(sal) from emp
                                where emp.deptno = resumen.deptno);

    commit;

    dbms_output.put_line('Proceso Ok');
end;
```

### Ejecución:

```
SQL> exec pr110;  
Proceso Ok  
  
PL/SQL procedure successfully completed.
```

### Verificar la ejecución:

```
SQL> select * from resumen;
```

DEPTNO	EMPS	PLANILLA
10	3	8750
20	5	10875
30	6	9400
40	0	
50	0	

## Sentencia: DELETE

### Sintaxis:

```
Delete from NombreTabla  
Where condición;
```

### Script 5

Desarrollar un procedimiento para eliminar un departamento, primero debe verificar que no tenga registros relacionados en la tabla EMP.

```
create or replace procedure pr111(cod number)  
is  
    cont number;  
begin  
    select count(*) into cont from dept where deptno = cod;  
    if cont = 0 then  
        dbms_output.put_line('No existe');  
        return;  
    end if;  
    select count(*) into cont from emp where deptno = cod;  
    if cont > 0 then  
        dbms_output.put_line('No puede se eliminado');  
        return;  
    end if;  
    delete from dept where deptno = cod;  
    commit;  
    dbms_output.put_line('Proceso Ok');  
end;
```

### Ejecución:

```
SQL> exec pr111(10);  
No puede se eliminado  
  
PL/SQL procedure successfully completed.
```



## Nombres de Variables

Se recomienda no utilizar nombres de variables y/o parámetros iguales a los nombres de las columnas, sobre todo si van a ser utilizadas en las instrucciones SQL.

### Script 6

Desarrollar un procedimiento para eliminar un empleado. El siguiente procedimiento tiene un resultado inesperado.

```
create or replace procedure pr112(empno number)
is
begin
    delete from emp where empno = empno;
end;
```

Si intentamos eliminar un empleado, se eliminarán todos los registros de la tabla **EMP**:

```
SQL> exec pr112(7654);

PL/SQL procedure successfully completed.
```

Verifiquemos el resultado:

```
SQL> select * from emp;

no rows selected
```

Esto sucede incluso si el código del empleado no existe. Ahora ejecutemos la sentencia **ROLLBACK** para recuperar los empleados.

```
SQL> rollback;

Rollback complete.
```

Si consultamos nuevamente la tabla **EMP** tendremos los registros recuperados.

## Cláusula Returning

Sirve para obtener información de la última fila modificada, puede ser utilizado con las sentencias insert, update, y delete.

### Sintaxis:

```
returning expresión, ... into variable, ... ;
```

### Script 7

Ejemplo ilustrativo de cómo usar RETURNING. La tabla test y la secuencia SQTEST se crear en una lección anterior.

```
create or replace procedure pr113(msg varchar2)
is
    v_rowid rowid;
    v_id    number;
begin
    insert into test values(sqtest.nextval,msg)
        returning rowid, id into v_rowid, v_id;
    commit;
    dbms_output.put_line('RowId: ' || v_rowid);
    dbms_output.put_line('Id:    ' || v_id);
end;
```

### Ejecución:

```
SQL> exec pr113('El deporte es salud.');
```

RowId:	AAQAEJAABAAAAEKAAA
Id:	21

## Referencias de Tablas

### Sintaxis:

```
[esquema.]tabla[@enlace]
```

### Script 8

En el siguiente script iniciamos sesión con privilegio SYSDBA y luego consultamos la tabla DEPT de SCOTT.

```
SQL> conn / as sysdba  
Conectado.
```

```
SQL> select * from scott.dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

## Enlaces de Base de Datos

### Sintaxis

```
create [shared] [public] database link nombre_enlace
connect to nombre_usuario identified by contraseña
[using cadena_sqlnet];
```

### Script 9

Crearemos un enlace remoto público para conectarnos como usuario HR, este enlace debe ser creado como usuario SYS:

```
SQL> conn / as sysdba
Connected.

SQL> create public database link lnk_demo
2   connect to hr identified by hr
3   using 'dbegcc';

Database link created.
```

Ahora haremos una consulta al esquema HR:

```
SQL> conn scott/tiger
Connected.

SQL> select employee_id, first_name
2   from employees@lnk_demo
3   where department_id = 30;

EMPLOYEE_ID FIRST_NAME
-----
114 Den
115 Alexander
116 Shelli
117 Sigal
118 Guy
119 Karen

6 rows selected.
```

## Sinónimos

Facilitan la referencia a tablas de otros esquemas, incluso de otros servidores.

### Sintaxis

```
create [or replace] [public] synonym [esquema.] sinonimo
for [esquema.] objeto [@@blink]
```

### Script 10

Crearemos un sinónimo público para acceder a la tabla EMPLOYEES del esquema HR. Debemos crearlo como SYS.

```
SQL> conn / as sysdba
Connected.
SQL> create or replace public synonym hr_emp
2      for employees@lnk_demo;

Synonym created.
```

Ahora como **scott** accederemos a la tabla **employees** mediante el sinónimo:

```
SQL> conn scott/tiger
Connected.

SQL> select employee_id, first_name
2      from hr_emp
3      where rownum = 1;

EMPLOYEE_ID FIRST_NAME
-----
100 Steven
```





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 10 Cursores**

### Contenido

- Procesamiento de Cursores
- Atributos de los Cursores
- Bucles de Extracción
- Bucles For Implícitos
- Cursores Select For Update
- Cursores Implícitos

## Procesamiento de Cursores

Los cursores permiten realizar recorridos a través de las filas de una consulta.

### Pasos a Seguir

1. Declarar el cursor
2. Apertura del cursor
3. Extracción de los resultados
4. Cerrar el cursor

### Declarar un Cursor

#### Sintaxis

```
CURSOR nombre_cursor [ ( Parámetros ) ]  
IS sentencia_select;
```

#### Script 1

```
declare  
    cursor c_demo is select * from dept;
```

### Apertura de un Cursor

#### Sintaxis

```
OPEN nombre_cursor [(argumentos)];
```

#### Script 2

```
open c_demo;
```



### Extracción de datos desde un cursor

#### Sintaxis 1

```
Fetch nombre_cursor into lista_variables;
```

#### Sintaxis 2

```
Fetch nombre_cursor into registro;
```

#### Script 3

```
fetch c_demo into cod, nom, loc;
```

### Cerrar un cursor

#### Sintaxis

```
Close nombre_cursor;
```

#### Script 4

```
create or replace procedure pr114
is
  cursor c_demo is select * from dept;
  r dept%rowtype;
begin
  open c_demo;
  fetch c_demo into r;
  close c_demo;
  dbms_output.put_line('deptno: ' || r.deptno);
  dbms_output.put_line('dname: ' || r.dname);
  dbms_output.put_line('loc: ' || r.loc);
end;
```

### Ejecución:

```
SQL> exec pr114;  
deptno: 10  
dname: ACCOUNTING  
loc: NEW YORK  
  
PL/SQL procedure successfully completed.
```

## Atributos de los Cursores

ATRIBUTO	TIPO	DESCRIPCIÓN
%Found	BOOLEAN	Devuelve TRUE si la última sentencia FETCH fue exitosa.
%NotFound	BOOLEAN	Devuelve TRUE si la última sentencia FETCH no fue exitosa.
%IsOpen	BOOLEAN	Este atributo se utiliza para averiguar si un cursor está abierto o no.
%RowCount	NUMBER	Este atributo se utiliza para averiguar la cantidad de filas que se van extrayendo del cursor.

## Bucles de Extracción

### Bucles Simples

#### Formato

```
open cursor_name ;  
loop  
    fetch cursor_name into ... ;  
    exit when cursor_name%notfound;  
    -----  
    -----  
    -----  
end loop;  
close cursor_name;
```

### Script 5

Procedimiento para listar los códigos y nombres de los empleados.

```
create or replace procedure pr115
is
  cursor c_emp is select * from emp;
  r emp%rowtype;
begin
  open c_emp;
  loop
    fetch c_emp into r;
    exit when c_emp%notfound;
    dbms_output.put_line(r.empno || ' - ' || r.ename);
  end loop;
  close c_emp;
end;
```

Ejecución:

```
SQL> exec pr115;
7369 - SMITH
7499 - ALLEN
7521 - WARD
7566 - JONES
7654 - MARTIN
7698 - BLAKE
7782 - CLARK
7788 - SCOTT
7839 - KING
7844 - TURNER
7876 - ADAMS
7900 - JAMES
7902 - FORD
7934 - MILLER

PL/SQL procedure successfully completed.
```

## Bucles While

### Formato

```
open cursor_name;
fetch cursor_name into ... ;
while cursor_name%found loop
    -----
    -----
    -----
    fetch cursor_name into ... ;
end loop;
close cursor_name;
```

### Script 6

Tenemos una tabla de nombre PLANILLAMES para guardar la planilla por mes por departamento, la estructura es la siguiente:

```
create table PLANILLAMES (
    anio number(4),
    mes number(2),
    deptno number(2),
    emps number(2) not null,
    planilla number(10,2) not null,
    constraint pk_planillames primary key(anio,mes, deptno)
);
```

Ahora desarrollaremos un procedimiento para generar la planilla de un determinado mes. Este procedimiento debe verificar si la planilla ya fue generada.

```
create or replace procedure pr116(p_anio number, p_mes number)
is
  cursor c_dept is select deptno from dept;
  v_deptno dept.deptno%type;
  cont number;
  v_emps number;
  v_planilla number;
begin
  select count(*) into cont
    from planillames
   where anio = p_anio and mes = p_mes;
  if (cont > 0) then
    dbms_output.put_line('Ya esta procesado');
    return;
  end if;
  open c_dept;
  fetch c_dept into v_deptno;
  while c_dept%found loop
    select count(*), sum(sal) into v_emps, v_planilla
      from emp
     where deptno = v_deptno;
    insert into planillames
      values(p_anio, p_mes, v_deptno, v_emps, nvl(v_planilla,0));
    fetch c_dept into v_deptno;
  end loop;
  close c_dept;
  commit;
  dbms_output.put_line('Proceso ok.');
```

end;

**Ejecución:**

```
SQL> exec pr116(2005,2);
Proceso ok.

PL/SQL procedure successfully completed.
```

Consultemos el resultado:

```
SQL> select * from planillames;
```

ANIO	MES	DEPTNO	EMPS	PLANILLA
2005	2	10	3	8750
2005	2	20	5	10875
2005	2	30	6	9400
2005	2	40	0	0
2005	2	50	0	0

## Bucle For

### Formato

```
for variable in cursor_name loop
    -----
    -----
end loop
```

### Script 7

Procedimiento para determinar el número de empleados y el importe de la planilla, por departamento.

```
create or replace procedure pr117
is
    cursor c_dept is select * from dept;
    emps number;
    planilla number;
    cad varchar2(100);
begin
    for r in c_dept loop
        select count(*), sum(nvl(sal,0)) into emps, planilla
            from emp where deptno = r.deptno;
        cad := r.deptno || ' - ' || emps || ' - ' || nvl(planilla,0);
        dbms_output.put_line(cad);
    end loop;
end;
```

Ejecución:

```
SQL> exec pr117;  
10 - 3 - 8750  
20 - 5 - 10875  
30 - 6 - 9400  
40 - 0 - 0  
50 - 0 - 0  
  
PL/SQL procedure successfully completed.
```

### Ejercicio 1

Desarrollar un procedimiento que determine el empleado con mayor sueldo por departamento.

## Bucles For Implícitos

### Formato

```
for variable in (sentencia_select) loop
    -----
    -----
end loop
```

### Script 8

Determinar el sueldo promedio por departamento.

```
create or replace procedure pr118
is
    prom number;
begin
    for r in (select deptno from dept) loop
        select avg(nvl(sal,0)) into prom
            from emp where deptno = r.deptno;
        dbms_output.put_line(r.deptno || '-' ||
to_char(nvl(prom,0), '999,990.00'));
    end loop;
end;
```

### Ejecución:

```
SQL> exec pr118;
10-   2,916.67
20-   2,175.00
30-   1,566.67
40-         0.00
50-         0.00

PL/SQL procedure successfully completed.
```



## Cursores Select For Update

### Sintaxis

```
for update [of lista_columnas] [nowait | wait n]
```

### Script 6 . 1

Listado de empleados.

```
create or replace procedure pr119
is
  cursor c_demo is select * from emp for update wait 2;
begin
  for r in c_demo loop
    dbms_output.put_line(r.empno || '-' || r.ename);
  end loop;
end;
```

Antes de ejecutar el procedimiento, en otra ventana inicie una transacción sobre la tabla **EMP**.

```
SQL> exec pr119;
BEGIN pr119; END;

*
ERROR at line 1:
ORA-30006: resource busy; acquire with WAIT timeout expired
ORA-06512: at "SCOTT.PR119", line 3
ORA-06512: at "SCOTT.PR119", line 5
ORA-06512: at line 1;
```

## Cursores Implícitos

Se puede utilizar **SQL%Atributo** para verificar la ejecución de una sentencia SQL.

### Script 9

Actualizar el salario de un empleado.

```
create or replace procedure pr120(cod number, delta number)
is
begin
    update emp
        set sal = sal + delta
        where empno = cod;
    if sql%notfound then
        dbms_output.put_line('no existe');
    else
        commit;
        dbms_output.put_line('proceso ok');
    end if;
end;
```

Ejecución:

```
SQL> exec pr120(7369,200);
proceso ok

PL/SQL procedure successfully completed.
```



**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 11 FUNCIONES Y PROCEDIMIENTOS**

### **Contenido**

- Funciones
- Procedimientos
- Parámetros

## Funciones

---

### Sintaxis

```
CREATE OR REPLACE FUNCTION nombre_función (  
    Arg1 [ IN | OUT | IN OUT ] tipo,  
    Arg2 [ IN | OUT | IN OUT ] tipo,  
    . . . ) RETURN tipo  
AS  
  
    Declaraciones_Locales  
  
BEGIN  
  
    Cuerpo_Función  
  
EXCEPTION  
  
    Tratamiento_Excepciones  
  
END nombre_procedimiento;
```

### Script 1

```
create or replace function cantemp( p_deptno dept.deptno%type) return
number
as
    v_cont number;
begin
    select count(*) into v_cont from emp where deptno = p_deptno;
    return v_cont;
end cantemp;
```

### Ejecución 1

```
SQL> declare
2     v_cant number;
3     begin
4         v_cant := cantemp( 10 );
5         dbms_output.put_line( v_cant || ' empleados' );
6     end;
7     /
3 empleados

PL/SQL procedure successfully completed.
```

### Ejecución 2

```
declare
    cursor c_dept is select deptno from dept;
    v_cant number;
begin
    for v_dept in c_dept loop
        v_cant := cantemp(v_dept.deptno);
        dbms_output.put_line( 'dept=' || v_dept.deptno || ' empleados: ' ||
v_cant );
    end loop;
end;
```

## Procedimientos

### Sintaxis

```
CREATE OR REPLACE PROCEDURE nombre_procedimiento (  
    Arg1 [ IN | OUT | IN OUT ] tipo,  
    Arg2 [ IN | OUT | IN OUT ] tipo,  
    . . . )  
AS  
  
    Declaraciones_Locales  
  
BEGIN  
  
    Cuerpo_Procedimiento  
  
EXCEPTION  
  
    Tratamiento_Excepciones  
  
END nombre_procedimiento;
```

### Script 2

```
create or replace procedure adddept(  
    p_deptno dept.deptno%type,  
    p_dname dept.dname%type,  
    p_loc dept.loc%type )  
as  
begin  
    insert into dept(deptno, dname, loc) values(p_deptno, p_dname, p_loc);  
end adddept;
```

### Ejecución

```
SQL> exec adddept( 15, 'demo', 'lima');  
  
PL/SQL procedure successfully completed.
```

## Parámetros

MODO	DESCRIPCIÓN
IN	El valor del parámetro real se pasa al procedimiento cuando se produce la llamada al mismo. Dentro del procedimiento, el parámetro formal se comporta como una constante PL/SQL, se considera de solo lectura y no puede ser modificado. Cuando el procedimiento finaliza y devuelve el control al entorno desde donde se produjo la llamada, el parámetro real no se modifica.
OUT	Se ignora cual valor que el parámetro real pueda tener cuando se produce la llamada al procedimiento. Dentro del procedimiento, el parámetro formal se comporta como una variable sin inicializar, por lo que su valor inicial es NULL. Puede leerse y escribir en dicha variable. Cuando el procedimiento finaliza y devuelve el control al entorno desde donde se produjo la llamada, se asigna el valor del parámetro formal al parámetro real.
IN OUT	Este modo es una combinación de los modos IN y OUT.

### Script 3

```
Create or Replace Procedure prTestOUT1
( p_Raise IN Boolean, p_Dato Out Varchar2 )
Is
    Excepl Exception;
Begin
    p_Dato := 'Alianza Campeon';
    If p_Raise Then
        Raise Excepl;
    Else
        Return;
    End If;
End;
```

### Ejecución 1

```
SQL> Declare
2   Rpta Varchar2(20) := 'Shakira';
3   Begin
4   DBMS_Output.Put_Line( 'Valor Inicial: ' || Rpta );
5   prTestOUT1( False, Rpta );
6   DBMS_Output.Put_Line( 'Valor Final: ' || Rpta );
7   End;
8   /
Valor Inicial: Shakira
Valor Final: Alianza Campeon

PL/SQL procedure successfully completed.
```



### Ejecución 2

```
SQL> Declare
2   Rpta Varchar2(20) := 'Shakira';
3   Begin
4   DBMS_Output.Put_Line( 'Valor Inicial: ' || Rpta );
5   prTestOUT1( True, Rpta );
6   DBMS_Output.Put_Line( 'Valor Final: ' || Rpta );
7   Exception
8   When Others Then
9   DBMS_Output.Put_Line( 'Valor Después del error: ' || Rpta );
10  End;
11  /
Valor Inicial: Shakira
Valor Después del error: Shakira

PL/SQL procedure successfully completed.
```

### Uso de NOCOPY

#### Script 4

```
Create or Replace Procedure prTestOUT2
( p_Raise IN Boolean, p_Dato Out NOCOPY Varchar2 )
Is
    Excepl Exception;
Begin
    p_Dato := 'Alianza Campeon';
    If p_Raise Then
        Raise Excepl;
    Else
        Return;
    End If;
End;
```

#### Ejecución 1

```
SQL> Declare
2   Rpta Varchar2(20) := 'Shakira';
3   Begin
4       DBMS_Output.Put_Line( 'Valor Inicial: ' || Rpta );
5       prTestOUT2( False, Rpta );
6       DBMS_Output.Put_Line( 'Valor Final: ' || Rpta );
7   End;
8   /
Valor Inicial: Shakira
Valor Final: Alianza Campeon

PL/SQL procedure successfully completed.
```

### Ejecución 2

```
SQL> Declare
2   Rpta Varchar2(20) := 'Shakira';
3   Begin
4   DBMS_Output.Put_Line( 'Valor Inicial: ' || Rpta );
5   prTestOUT2( True, Rpta );
6   DBMS_Output.Put_Line( 'Valor Final: ' || Rpta );
7   Exception
8   When Others Then
9   DBMS_Output.Put_Line( 'Valor Después del error: ' || Rpta );
10  End;
11  /
Valor Inicial: Shakira
Valor Después del error: Alianza Campeon

PL/SQL procedure successfully completed.
```





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 12 PAQUETES**

### **Contenido**

- Especificación del Paquete
- Cuerpo del Paquete
- Paquetes Predefinidos

---

### Especificación del Paquete

---

Es una estructura PL/SQL que permite almacenar definiciones, funciones y procedimientos relacionados como una sola unidad.

#### Sintaxis

```
CREATE OR REPLACE PACKAGE nombre_paquete AS  
  
    Definiciones  
  
END nombre_paquete;
```

#### Script 1

```
CREATE OR REPLACE PACKAGE testpackage as  
  
    function suma( n1 in number, n2 in number ) return number;  
  
END testpackage;
```

### Cuerpo del Paquete

#### Sintaxis

```
CREATE OR REPLACE PACKAGE BODY nombre_paquete AS  
  
    Implementación  
  
END nombre_paquete;
```

#### Script 2

```
CREATE OR REPLACE PACKAGE BODY testpackage as  
  
    function suma( n1 in number, n2 in number ) return number  
    as  
        rtn number;  
    begin  
        rtn := n1 + n2;  
        return rtn;  
    end;  
  
END testpackage;
```

#### Ejecución 1

```
SQL> select testpackage.suma( 12,13) from dual;  
  
TESTPACKAGE.SUMA(12,13)  
-----  
                        25
```

### Ejecución 2

```
SQL> declare
  2   v_suma number;
  3   begin
  4   v_suma := testpackage.suma( 12,13);
  5   dbms_output.put_line ( 'Suma: ' || v_suma );
  6   end;
  7   /
```

Suma: 25

PL/SQL procedure successfully completed.



### Paquetes Predefinidos

Oracle Database provee más de 700 paquetes predefinidos, a continuación, tenemos un listado de los más conocidos:

PAQUETE	DESCRIPCIÓN
DBMS_OUTPUT	Este paquete sirve para mostrar mensajes por pantalla. Para que esta información se vea por pantalla tienes que activarlo.  SQL> SET SERVEROUTPUT ON
DBMS_STATS	Este paquete se utiliza para modificar, ver, exportar, importar y borrar estadísticas de la base de datos. Oracle recomienda la utilización de este paquete a partir de la versión 9i para el cálculo de estadísticas en lugar de usar ANALYZE debido a que es más exacto y más eficiente. Aunque ANALYZE ya no sea la opción adecuada para calcular estadísticas eso no quiere decir que ya se pueda utilizar, sigue disponible y se puede utilizar para validar la estructura de una tabla o buscar por CHAINED ROWS y algunas otras cosas.
DBMS_SQL	Este paquete se utiliza para acceder a la base de datos con SQL dinámico.
DBMS_DDL	Este paquete provee procedimiento que permite el acceso a un número determinado de sentencias DDL dentro de los programas PL/SQL.
DBMS_UTILITY	Este paquete provee procedimientos y funciones para el análisis de objetos y la administración de la base de datos Oracle. Por ejemplo se pueden compilar todos los objetos que se encuentran en un determinado esquema de base de datos.
UTL_FILE	Este paquete añade capacidades para entrada/salida de archivos.
DBMS_ALERT	Este paquete provee procedimientos y funciones que permiten a las aplicaciones nombrar y emitir señales de alerta.
DBMS_LOB	Este paquete provee procedimientos y funciones que manipulan datos de tipo BLOB, CLOB, NCLOB y BFILE.
DBMS_SESSION	Este paquete provee procedimientos y funciones para controlar una sesión de usuario de una aplicación.
DBMS_TRANSACTION	Este paquete provee procedimientos y funciones para el control de transacciones.





**INSTITUTO DE INVESTIGACIÓN FIIS-UNI**  
Facultad de Ingeniería Industrial y de Sistemas

# **TALLER DE PROGRAMACION CON ORACLE PL/SQL**

---

## **Capítulo 13 DESENCADENANTES**

### **Contenido**

- ¿Qué es un Desencadenantes?
- Tipos de Desencadenantes
- Sintaxis General
- Disparadores DML
- Funciones Lógicas: inserting, deleting, updating.
- Uso de :old y :new
- Disparadores de Sustitución
- Disparadores del Sistema

## ¿Qué es un Desencadenantes?

---

Los desencadenantes son similares a los procedimientos y funciones, en el sentido de que son bloques nominados de PL/SQL con secciones declarativa, ejecutable y de tratamiento de excepciones, se almacenan en la base de datos como objetos independientes, y no pueden ser locales a un bloque o paquete.

Los desencadenantes difieren de los procedimientos y funciones en la forma en que se ejecutan, mientras que el desencadenante se ejecuta de manera implícita, los procedimientos y las funciones se ejecutan de manera explícita.

Los desencadenantes se activan cuando ocurre un suceso. El suceso que lo activa puede ser una operación DML (INSERT, UPDATE o DELETE) sobre una tabla o ciertos tipos de vista.

También es posible asociar desencadenantes con sucesos del sistema, como el inicio y finalización de una sesión, y ciertos tipos de operaciones del lenguaje de definición de datos (DDL).

## Tipos de Desencadenantes

---

Los desencadenantes se agrupan en las siguientes categorías:

- Desencadenantes DML
- Desencadenantes de Sustitución: Se aplica a vistas.
- Desencadenantes del Sistema

### Sintaxis General

```
CREATE OR REPLACE TRIGGER nombre_desencadenante
{BEFORE | AFTER | INSTEAD OF} suceso_disparo
[cláusula_referencia]
[WHEN condición_dispro]
[FOR EACH ROW]
BEGIN
    -----
    -----
    -----
END;
```

### Desencadenantes DML

CATEGORÍA	VALORES	COMENTARIOS
Instrucción	INSERT, DELETE, UPDATE	Define qué tipo de instrucción DML causa la activación del desencadenante.
Temporización	Antes o Después	Define si el desencadenante se ejecuta antes o después de la instrucción que lo activa.
Nivel	Fila o Instrucción	Define si el desencadenante se ejecuta por instrucción o por cada fila.

### Funciones Lógicas: inserting, deleting, updating.

Estas funciones se utilizan cuando el evento de un desencadenante es compuesto, es decir, queremos que el desencadenante se active ante diferentes operaciones DML pero no queremos que haga lo mismo para cualquiera de los eventos activadores. Con lo visto hasta ahora, la única solución sería diseñar un desencadenante para cada una de las acciones DML de activación. Sin embargo, con las funciones inserting, deleting y updating, podremos hacer todo en un único desencadenante.

#### Script 1

```
CREATE OR REPLACE TRIGGER tr_test_emp
AFTER INSERT OR DELETE OR UPDATE ON emp
BEGIN
    if inserting then
        dbms_output.put_line( 'nuevo empleado se ha insertado' );
    Elsif updating then
        dbms_output.put_line( 'un empleado se ha modificado' );
    Elsif deleting then
        dbms_output.put_line( 'un empleado se ha eliminado' );
    end if;
END tr_test_emp;
```

### Uso de :old y :new

INSTRUCCIÓN DE DISPARO	:OLD	:NEW
INSERT	NULL	Valores que se insertan en la tabla.
UPDATE	Valores originales.	Nuevos valores.
DELETE	Valores originales.	NULL

#### Script 2

Realizar un desencadenante que registre los cambios de salario de un empleado, el usuario que lo realiza y la fecha.

Tabla para registrar los cambios de salario:

```
Create Table Sal_History(
  EmpNo Number(4) not null,
  SalOld Number(7,2) null,
  SalNew Number(7,2) null,
  StartDate Date not null,
  SetUser Varchar2(30) not null
);
```

El desencadenante para registrar los cambios de salario:

```
Create or Replace Trigger tr_UpdateEmpSal
After Insert OR Update ON Emp
For Each Row
Begin
  Insert Into Sal_History(EmpNo, SalOld, SalNew, StartDate, SetUser)
  Values( :New.EmpNo, :Old.Sal, :New.Sal, sysdate, USER );
End tr_UpdateEmpSal;
```

### Prueba 1:

```
SQL> update emp
  2  set sal = 1200
  3  where empno = 7369;
```

1 row updated.

```
SQL> select * from Sal_History;
```

EMPNO	SALOLD	SALNEW	STARTDAT	SETUSER
7369	1000	1200	19/02/05	SCOTT

```
SQL> rollback;
```

Rollback complete.

### Prueba 2:

```
SQL> update emp
  2  set sal = 5000
  3  where empno in (7788, 7902);
```

2 rows updated.

```
SQL> select * from Sal_History;
```

EMPNO	SALOLD	SALNEW	STARTDAT	SETUSER
7902	3000	5000	19/02/05	SCOTT
7788	3000	5000	19/02/05	SCOTT

```
SQL> rollback;
```

Rollback complete.



### Prueba 3:

```
SQL> conn / as sysdba
Connected.
```

```
SQL> update scott.emp
  2  set sal = 20000
  3  where empno = 7844;
```

```
1 row updated.
```

```
SQL> select * from scott.sal_history;
```

EMPNO	SALOLD	SALNEW	STARTDAT	SETUSER
7844	1500	20000	19/02/05	SYS

### Script 3

En este script se ilustra como deshabilitar un desencadenante.

```
SQL> alter trigger tr_UpdateEmpSal disable;
```

```
Trigger altered.
```

```
SQL> update emp
  2  set sal = 5000
  3  where empno = 7788 ;
```

```
1 row updated.
```

```
SQL> select * from Sal_History;
```

```
no rows selected
```

```
SQL> rollback;
```

```
Rollback complete.
```

### Desencadenantes de Sustitución

#### Script 4

Vista para consultar los empleados:

```
Create Or Replace View v_empleados As
Select e.empno, e.ename, d.deptno, d.dname
From emp e Inner Join dept d
On e.deptno = d.deptno;
```

Desencadenante para insertar nuevos registros:

```
Create Or Replace Trigger tr_vista
  Instead Of Insert Or Delete On v_empleados
  For Each Row
Declare
  cuenta Number;
Begin
  If Inserting Then
    Select Count(*) Into cuenta From dept Where deptno = :new.deptno;
    If cuenta = 0 Then
      Insert Into dept(deptno,dname)
      Values(:New.deptno, :New.dname);
    End If;

    Select Count(*) Into cuenta From emp Where empno = :new.empno;
    If cuenta = 0 Then
      Insert Into emp(empno,ename,deptno)
      Values(:New.empno, :New.ename, :New.deptno);
    End If;
  Elsif Deleting Then
    Delete From emp Where empno = :old.empno;
  End If;
End tr_vista;
```

### Desencadenantes del Sistema

#### Script 5

Desencadenante que impide borrar cualquier objeto del esquema actual.

```
Create Or Replace Trigger tr_deny_drop
before Drop ON Schema
Begin
    raise_application_error( -20000, 'No es posible borrar objetos. ');
End tr_deny_drop;
```