

PL/SQL: Un lenguaje de Script para BD Oracle y algo más

Manuel Enciso & Enrique Soler

Universidad de Málaga

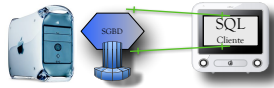
Contents

1	Preliminares	2
2	Entendiendo PL/SQL	2
2.1	Características de PL/SQL	3
2.2	El lenguaje	3
3	PL/SQL básico	4
3.1	Introducción	4
3.2	Declaración y uso de variables y tipos	4
3.3	Instrucciones de control de flujo	5
4	PL/SQL y la Base de Datos	6
4.1	Depuración de código PL/SQL	7
4.2	Manipulación de datos	7
4.3	Cursores	8
4.4	SQL dinámico	10
5	Objetos PL/SQL	10
5.1	Desarrollo de aplicaciones en PL/SQL	10
5.2	Paquetes	12
5.3	Excepciones	13

1 Preliminares

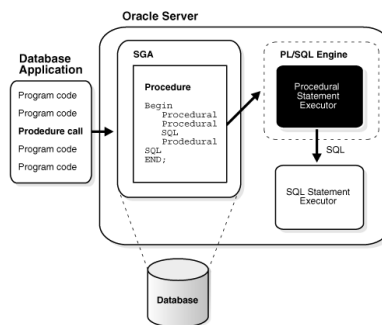
Entornos de ejecución de PL/SQL

- Disponer de servidor ORACLE y una instancia de BD activa.
- Uso de un cliente (*Front End*):

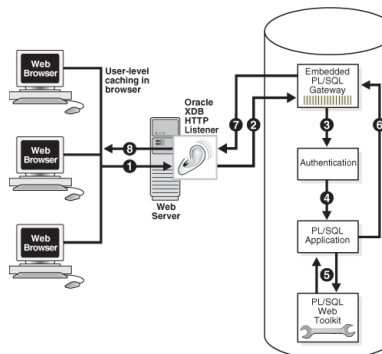


- SQL*PLUS (Oracle 8.1.6).
- SQL*PLUS Worksheet (ORACLE 8i).
- iSQLPLUS (ORACLE 9i).
- Clientes Java-Oracle: JDeveloper, SQLDeveloper, etc.
- Otros clientes de terceras partes (Aqua Data v4.5, TOAD, Net Beans, etc).
- Otras componentes de Oracle: Oracle Forms, SQL*Menu, etc.

Ejecución de PL/SQL en el servidor Oracle



Ejecución de PL/SQL en otros entornos



2 Entendiendo PL/SQL

Entendiendo PL/SQL

Contenido

1. Conocer características de PL/SQL.
2. Conocer la estructura del lenguaje.

2.1 Características de PL/SQL

El entorno

- Lenguaje propietario ORACLE.
- Lenguaje imperativo estructurado.
- Fácil de enlazar con la base de datos.
- Empotrable en la propia base de datos.
- Verdaderamente portable.

2.2 El lenguaje

Estructura código PL/SQL

DECLARE

variable tipo ;
(*Opcional*)

BEGIN

sentencias PL/SQL ;

EXCEPTION

Manejo Errores ;
(*Opcional*)

END;

/ - - Los ficheros terminan con el caracter /

Clases de bloques

- Bloques anónimos.
- Bloques nombrados. Usamos etiquetas (<<etiqueta>>) antes de la sentencia DECLARE y opcionalmente tras el END.
- Procedimientos, paquetes y funciones. Se pueden crear procedimientos almacenados en la base de datos: CREATE OR REPLACE PROCEDURE.
- Disparadores. Bloques asociados a acciones del DML: CREATE OR REPLACE TRIGGER.

Secuencias

- Elementos productores de valores numéricos. Independientes del resto de objetos del esquema.
- CREATE SEQUENCE Nombre START WITH Primer_Valor INCREMENT BY Numero;
- Funciones:
 - nombre.NEXTVAL: siguiente valor.
 - nombre.CURRVAL: valor actual. No está definido hasta la primera llamada a NEXTVAL
- Pueden usarse en SELECT, VALUES y SET. No se pueden usar en WHERE ni en órdenes de PL/SQL.
- Pueden ser usados por diferentes esquemas.

Funciones de SQL

- Caracteres: CONCAT (||), SUBSTR, LOWER, UPPER, DECODE, LPAD, RPAD, etc.
- Números: aritmética, trigonometría, ROUND, TRUNC, GREATEST, LEAST, etc.
- Fechas: ADD_MONTHS, MONTHS_BETWEEN, NEXT_DAY, diferencia de fechas, etc y SYS-DATE.
- Funciones de conversión: TO_NUMBER, TO_CHAR, TO_DATE.
- Pseudo-columnas: ROWNUM, ROWID.
- Manejo de valores nulos: NVL.

3 PL/SQL básico

PL/SQL básico

Contenido

1. *Conocer rudimentos de PL/SQL.*
2. *Determinar las variables y los tipos.*
3. *Determinar las instrucciones de control de flujo.*

3.1 Introducción

Identificadores

- Son cadenas de caracteres de longitud menor que 30. Comienzan por letra. Pueden incluir los símbolos especiales: `_`, `$` y `#`. No podemos usar palabras reservadas como identificadores.
- Se puede *legalizar* un identificador mediante el uso de las comillas (`"`). Por ejemplo `1_Empleado` es ilegal pero `"1_Empleado"` es legal.

Ejemplo

```
CREATE TABLE Mensajes (Codigo NUMBER(2) PRIMARY KEY,
                        Texto VARCHAR2(50) );

DECLARE
  Var_Cod_Mensaje1 NUMBER(2) DEFAULT 1; -- código del primer mensaje
  Var_Texto_Mensaje1 VARCHAR2(50); -- guarda texto del primer mensaje

  Var_Cod_Mensaje2 NUMBER(2) :=2; -- código del segundo mensaje
  Var_Texto_Mensaje2 VARCHAR2(50); -- guarda texto del segundo mensaje

BEGIN
  Var_Texto_Mensaje1:= 'El vuelo ha sido cancelado';

  INSERT INTO Mensajes VALUES(Var_Cod_Mensaje1,Var_Texto_Mensaje1);
  INSERT INTO Mensajes VALUES(Var_Cod_Mensaje2,'El vuelo tiene demora');

  COMMIT; --fin de transacción
END;
```

3.2 Declaración y uso de variables y tipos

Variables

- Declaración formal: *nombre tipo* [CONSTANT] [NOT NULL] [DEFAULT *valor* | :=*valor*];.
- Si se usa NOT NULL ha de inicializarse y no se le puede cambiar el valor a NULL.
- Las variables no inicializadas toman el valor NULL.
- Usamos CONSTANT para evitar cambios de la variable durante la ejecución del bloque.
- Ámbito de la variable: desde el BEGIN al END tiene accesibilidad.
- Visibilidad: son visibles en su ámbito salvo sobrecarga con otra definición posterior. Uso de bloques nombrados para aumentar visibilidad.

Ámbito de variables

```
<<BLOQUE_1>>
DECLARE
...
Variable_1 NUMBER;
...
BEGIN
...
    DECLARE
    ...
    Variable_1 NUMBER;
    ...
    BEGIN
    ...
        Variable_1 := 1; -- Asigna segunda variable
        BLOQUE_1.Variable_1 := 1; -- Asigna primera variable
    ...
    END;
...
END BLOQUE_1;
```

Tipos

Escalares: los mismos tipos que en la creación de tablas y algunos otros.

- Identificadores de columnas: ROWID. Para manejarlos se puede aplicar ROWIDTOCHAR.
- Booleanos: BOOLEAN. Es trivalente: TRUE, FALSE y NULL.

Registros

- Siempre declarar el tipo y luego una variable sobre él.
- TYPE nombre IS RECORD (

Campo_1 Tipo_1 [NOT NULL] [:= valor_1],
 ... ,
 Campo_k Tipo_k [NOT NULL] [:= valor_2]
);
- Se usa la notación punto para manejarlos.
- Se puede asignar componente a componente o registros enteros (siempre que sean del mismo tipo). También se puede usar una orden SELECT.
- Uso de %ROWTYPE y %TYPE.

Ejemplo

```
CREATE TABLE Mensajes (Codigo NUMBER(2) PRIMARY KEY,
                        Texto VARCHAR2(20) );
```

```
TYPE T_Mensajes IS RECORD (
    Codigo NUMBER(2) ,
    Texto VARCHAR2(20) );
```

```
Var_Mensajes_1 T_Mensajes; Var_Mensajes_2 Mensajes%ROWTYPE;
```

3.3 Instrucciones de control de flujo

Condicionales

- IF-THEN-ELSE.

```
IF Expresión_1 THEN cuerpo_1;
  ELSIF Expresión_2 THEN cuerpo_2;
    ELSIF Expresión_3 THEN cuerpo_3;
      ELSE cuerpo_4;
END IF;
```

- Si la expresión toma valor **TRUE** se ejecuta el **THEN**, pero se ejecuta el **ELSE** tanto en **FALSE** como en **NULL**. Usar predicado **IS NULL** en condiciones.
- Se puede añadir una orden **NULL** si queremos explícitamente no hacer nada.

Bucles

- Hay cuatro tipos: **LOOP**, **WHILE**, **FOR** y **FOR** en cursores.
- Condiciones de parada en **LOOP**:
 - **EXIT [WHEN condición];**
 - **IF condición THEN EXIT; END IF;**
- Los límites pueden ser expresiones y sólo se evalúan una vez.
- Se puede poner **REVERSE** en el contador de un bucle **FOR**.
- Las órdenes **EXIT** pueden ir seguidas de etiquetas de bucles que nos dicen de qué bucle salir.

Ejemplo: bucle **LOOP**

```
BEGIN
  << Etiqueta >> -- opcional, útil en anidamientos
  LOOP
    ... -- Cuerpo del bucle
    EXIT Etiqueta WHEN Condición;
  END LOOP Etiqueta;
END;
```

Ejemplo: bucle **WHILE**

```
BEGIN
  << Etiqueta >> -- opcional, útil en anidamientos
  WHILE Condición LOOP
    ... -- Cuerpo del bucle
  END LOOP Etiqueta;
END;
```

Ejemplo: bucle **FOR**

```
BEGIN
  << Etiqueta >> -- opcional, útil en anidamientos
  FOR Contador IN [REVERSE] ValorInicio..ValorFin LOOP
    ... -- Cuerpo del bucle
  END LOOP Etiqueta;
END;
```

4 PL/SQL y la Base de Datos

PL/SQL y la Base de Datos

Contenido

1. *Uso de la salida estándar para depurar código PL/SQL.*
2. *Manipular la base de datos.*
3. *Aprender el funcionamiento de los cursores.*
4. *SQL Dinámico.*

4.1 Depuración de código PL/SQL

Uso de la salida estándar

- Paquete DBMS_OUTPUT. Activación en SQL*Plus con la orden `SET SERVEROUTPUT ON SIZE longitud.`

<code>PUT(cadena)</code>	Pone la cadena en el buffer.
<code>NEW_LINE</code>	Pone una nueva línea en el buffer.
<code>PUT_LINE(cadena)</code>	Idem que <code>PUT+NEW_LINE</code> .
<code>ENABLE</code>	Activa el buffer con esa longitud.
<code>DISABLE</code>	Desabilita el buffer. Llamadas posteriores a <code>DBMS_OUTPUT</code> no se muestran.

- Usualmente se hace uso de la función `put_line`:
`dbms_output.put_line(Mi_Variable); --imprime contenido de variable`

4.2 Manipulación de datos

SQL empotrado

- Sentencias DML se introducen tal cual en código PL/SQL.
- Se pueden usar variables o literales.

```
DECLARE
  vNumero NUMBER(2):=1; --var. de acoplamiento
  vTexto VARCHAR2(20):='An'; --var. de acoplamiento
BEGIN
  INSERT INTO MENSAJES VALUES (vNumero+1,'UNO');
  DELETE FROM MENSAJES WHERE Texto LIKE '%'||vTexto;
END;
/
```

- Variables de acoplamiento sólo para las expresiones. *Nunca un nombre de tabla o de columna.*
- PL/SQL usa acoplamiento temprano por eficiencia.

Consultas

- La salida del `SELECT` se vuelca en una variable:
`SELECT ... INTO Lista_Variables|Registro`
`FROM ...`
`WHERE ...`
- Si la consulta devuelve más de una tupla se produce el error
`ORA1427:Single row query returns more than one row`
- Operaciones: se utilizan igual que en SQL, pero podemos incluir variables o expresiones PL/SQL en el `WHERE` y en el `SELECT` (siempre que no afecte al acoplamiento de los atributos).
- Cuidado con que el nombre de las variables no coincida con el nombre de un atributo de la tabla. Se puede arreglar etiquetando los bloques.

4.3 Cursores

Cursores

- Área de contexto: zona de memoria con información sobre procesamiento de consulta.
- Cursor: puntero a un área de contexto.
- Recuperación de múltiples filas.
- Tipos de cursores:
 - Explícitos (declarados por el programador cuando un SELECT devuelve más de una fila).
 - Implícitos (declarados implícitamente por PL/SQL para sentencias SELECT).

Cursores explícitos

- Declaración simple.

```
CURSOR Nombre_Cursor IS sentencia_select;
```
- Declaración con variables de acoplamiento.

```
DECLARE  Var_Codigo_Pasajero Pasajeros.NIP%TYPE;

CURSOR c_Pasajero IS SELECT Nombre
                        FROM Pasajeros
                        WHERE NIP = Var_Codigo_Pasajero;
```

La variable ha de ser declarada e inicializada antes del cursor.

Manejo del cursor

- Apertura del cursor: `OPEN nombre_cursor;`
- Acciones:
 1. Examina variables de acoplamiento.
 2. Determina el conjunto activo de tuplas.
 3. Apunta el puntero a la primera fila.
- Consistencia de lectura: las variables de acoplamiento se evalúan sólo al abrir el cursor. Su reasignación no cambia el conjunto activo.
- Recuperación de datos:
 - `FETCH nombre_cursor INTO Lista_Variables|Registro;`
 - Se pueden definir variables registro sobre los cursores: `Variable_Cursor Nombre_cursor%ROWTYPE;`
- Cierre del cursor: `CLOSE nombre_cursor;`

Atributos de cursor

- Permiten consultar estado de cada cursor.
- Se consultan sobre la variable de cursor: `nombre_cursor%Atributo`
 - `%FOUND`: Si el `FETCH` ha tenido éxito.
 - `%NOTFOUND`: Si el `FETCH` no ha tenido éxito.
 - `%ROWCOUNT`: Número de filas consultadas hasta ahora.
 - `%ISOPEN`: Se ha abierto y no se ha cerrado.

Recorridos de cursores

- Cuando se extraen varias filas hay que tratarlas una a una en PL/SQL.
- Podemos meter las tuplas en una tabla PL/SQL y obtener un objeto unificado para su tratamiento.
- Es posible hacer el recorrido con LOOP, WHILE y FOR de cursores (el más apropiado).

Bucle FOR de cursores

```
DECLARE
  CURSOR C_Cursor IS SELECT ...;
BEGIN
  FOR Var_Cursor IN C_Cursor LOOP
    ... -- procesar información
  END LOOP;
END;
```

Cursores con parámetros

- Declaración:


```
CURSOR Nombre_Cursor (Parametro Tipo_Datos) IS SELECT ... FROM T WHERE ...(parametro)...;
```
- Llamada:


```
FOR Var_Cursor IN Nombre_Cursor (Valor) LOOP ...
```

Cursores sin definición previa

- Uso: Cuando no vayamos a usarlo en ninguna parte más del código.
- Poco estructurado. Esconde la definición del cursor.
- Llamada:


```
FOR Var_Cursor IN (SELECT ... FROM ....) LOOP ...
```

Cursores para actualización

- Útil cuando se modifica la fila que extrae el cursor.
- Declaración:


```
CURSOR Nombre_Cursor IS SELECT ... FOR UPDATE [OF Columna_1,...,Columna_n] [NOWAIT];
```
- Produce bloqueos mientras dura la orden OPEN. Asegura la no interferencia con otras transacciones en curso.
- Si se produce un bloqueo y lo hemos declarado NOWAIT se produce el error ORA-54: **Resource busy and acquire with NOWAIT specified.**

Cursores para actualización II

- Podemos operar sobre la fila apuntada con FETCH:

```
DECLARE
  CURSOR C_Cursor IS SELECT...
                        FROM Tabla WHERE... FOR UPDATE;
BEGIN
  FOR Var_Cursor IN C_Cursor LOOP
    UPDATE Tabla SET ...
    WHERE CURRENT OF C_Cursor;
  END LOOP;
  COMMIT;
END;
```

La orden COMMIT no puede ser ejecutada dentro del bucle porque elimina los bloqueos e impide seguir procesándolo.

4.4 SQL dinámico

Lanzar sentencias que no están definidas anteriormente

Existen dos opciones:

- Uso de la librería DBMS_SQL
- Uso de la sentencia EXECUTE IMMEDIATE

```
DECLARE
    var_tabla VARCHAR2(100);
BEGIN
    var_tabla:='EMPLEADOS';
    EXECUTE IMMEDIATE 'INSERT INTO '||var_tabla||' VALUES
    (121562,'Manuel Enciso')';
    COMMIT;
END;
```

Podemos por primera vez lanzar una orden que se compila en ejecución.

5 Objetos PL/SQL

PL/SQL

Contenido

1. *Conocer el desarrollo de aplicaciones en PL/SQL.*
2. *Estructurar código en paquetes.*
3. *Realizar un control de excepciones.*

5.1 Desarrollo de aplicaciones en PL/SQL

Procedimientos

```
CREATE [OR REPLACE] PROCEDURE NombreProcedimiento
    [( Argumento_1 [{IN|OUT|IN OUT}] Tipo_1 , ...,
      Argumento_n [{IN|OUT|IN OUT}] Tipo_n )] {IS|AS}
    --declaración de variables opcional
BEGIN
    ... -- tratamiento con PL/SQL
[EXCEPTION ...] -- opcional
END [NombreProcedimiento];
```

Los tipos son predefinidos o creados por el usuario. Si hay muchos parámetros se aconseja usar un registro como entrada para simplificar la comprensión de las llamadas.

Procedimientos (en detalle)

- El modo por defecto de los parámetros formales es IN.
- Las restricciones se comprueban en compilación.
- Modo del parámetro afecta a las asignaciones (: =), a los SELECT...INTO y a los FETCH...INTO.
- Pueden ser llamados desde un bloque de PL/SQL (también desde la sección de excepciones).
- Pueden ser llamados:
 - directamente usando la `Execute Nombre_Procedimiento(Parámetros)`.
 - desde el cuerpo de un Trigger.
 - por otras aplicaciones de la BD (como Oracle Forms).
- Se crean en el servidor.
- Son compilados en la creación.

Ventajas

- Aumenta la flexibilidad en el ámbito de la seguridad.
- Introduce la herencia de permisos en el contexto del esquema.
- Mejora el rendimiento
 - Disminuye el tráfico de la red.
 - No se requiere compilación en la ejecución.
 - No requiere lectura a disco (si ya está cargado en la Shared Pool o en la SGA).
- Reducción de uso de memoria (copia compartida por diferentes usuarios).
- Productividad en desarrollo (desarrollo de rutinas de mantenimiento de tablas).
- Refuerzo de la integridad.

Parámetros

- La declaración de tipos es no restringida.
- Hay que tener en cuenta la restricción de variables que se invocan con el procedimiento para evitar el error `ORA-6502: numeric or value error`.
- Se puede restringir la declaración de tipos usando `%type`:

```
CREATE OR REPLACE PROCEDURE Procedimiento (
    ...,
    Argumento IN Tabla.Atributo%TYPE,
    ...) AS
BEGIN
    ... -- tratamiento con PL/SQL
END Procedimiento;
```

Llamadas

- Se pueden considerar llamadas estándar especificando valores predefinidos y obviando los parámetros en la llamada.

```
CREATE OR REPLACE PROCEDURE Nombre_Procedimiento
( Argumento_1 IN Tipo_1 DEFAULT Valor_Predif_1,
  ...,
  Argumento_n IN OUT Tipo_n :=Valor_Predif_n ) IS
BEGIN
    ... --tratamiento con PL/SQL
END Nombre_Procedimiento;

EXEC Nombre_Procedimiento(); --llamada a procedimiento
```

Llamadas (más detalles)

- Para asignar sólo un subconjunto de los parámetros se usa la notación nominal: `Nombre_Proc(Arg_1=>Valor_1, ..., Arg_n=>Valor_n);`
- Sólo obliga a mantenimiento ante grandes cambios de la especificación.
- Permite usar valores predeterminados en cualquier parámetro, no sólo en los primeros.

Funciones

- Las llamadas a funciones son parte de una expresión PL/SQL.
- Declaración

```
CREATE [OR REPLACE] FUNCTION NombreFunción
    [( Sección de parámetros )]
    RETURN Tipo_Devuelto {IS|AS}
    --declaración de variables opcional
BEGIN
    --Cuerpo_Función
    RETURN Valor_Devuelto;
END NombreFunción;
```

- RETURN devuelve el control al lugar donde se hizo la llamada.
- Las funciones pueden devolver más de un valor usando para ello parámetros en modo OUT. No aconsejable.

Subrutinas

- Declaración

```
DECLARE --Bloque externo
    Var_Numero NUMBER(3);
    FUNCTION Cambiar_Numero(P_Number NUMBER)
        RETURN NUMBER AS
    BEGIN
        ... -- Cuerpo función
    END Cambiar_Numero;
BEGIN -- Cuerpo Bloque
    Var_Numero:= Cambiar_Numero(Var_Numero);
    ...
END;
```

- Los subprogramas locales siempre al final del DECLARE.

Uso de rutinas

- Para borrar objetos: DROP PROCEDURE Nombre.Procedimiento o DROP FUNCTION Nombre.Función.
- Se siguen las normas de propagación de errores habituales.
- Si se propaga una excepción más allá del bloque por falta de tratamiento, los parámetros no se devuelven.
- Privilegios sobre procedimientos y funciones: EXECUTE. El permiso de ejecución sobre una rutina permite usar todos los objetos referenciados explícitamente en el mismo.
- La rutina se ejecuta sobre los objetos del esquema dueño de la misma.

5.2 Paquetes

Paquetes

- Crea una asociación de un conjunto de objetos: procedimientos, funciones, variables, excepciones, tipos y cursores. Posibilitan el uso de variables globales en PL/SQL (salvo en aplicaciones Web).
- Nunca son locales.
- Se pueden agrupar los objetos según varios criterios: perfil de usuario, objeto del paquete, etc.
- Consta de *Especificación* (o cabecera) y *Cuerpo*.

Paquetes: declaración de especificación

```
CREATE OR REPLACE PACKAGE Nombre_Paquete AS
    ... --especificación de funciones
    FUNCTION Una_Rutina(arg1 Tipo) RETURN Tipo_Devuelto;

    ... --especificación de procedimientos
    PROCEDURE Un_Procedimiento(parametro Tipo);
END Nombre_Paquete;
```

Paquetes: declaración de cuerpo

```
CREATE OR REPLACE PACKAGE BODY Nombre_Paquete AS
    ... --cuerpo de funciones
    FUNCTION Una_Rutina(arg1 Tipo) RETURN Tipo_Devuelto IS
    BEGIN
        --cuerpo de la función
    END;

    ... --cuerpo de procedimientos
    PROCEDURE Un_Procedimiento(parametro Tipo) IS
    BEGIN
        --cuerpo del procedimiento
    END;
END Nombre_Paquete;
```

5.3 Excepciones**Control de Excepciones**

```
DECLARE
    Declaración excepciones
BEGIN
    Producción de excepciones
EXCEPTION
    Manejo Errores
END;
```

- Legibilidad.
- Evita olvidar tratamiento de algún error.
- Tratamiento separado, uniforme y completo de errores.

Ejemplo

```
DECLARE
    Error EXCEPTION;
BEGIN
    RAISE Error;
EXCEPTION
    WHEN Error [OR Error_Predef] THEN ... ;
END;
```

Excepciones predefinidas: cursores

- Cursores explícitos e implícitos:
 - `INVALID_CURSOR` (ORA-1001): Se intenta efectuar una operación ilegal (ej. cerrar o intentar extraer datos de un cursor no abierto)
 - `CURSOR_ALREADY_OPEN`: Se Intenta abrir un cursor ya abierto.
- Cursores implícitos:
 - `NO_DATA_FOUND` (ORA-1403): Una orden `SELECT..INTO` no devuelve ninguna fila o Se referencia un elemento de una tabla PL/SQL al que no se le ha asignado ningún valor previamente.
 - `TOO_MANY_ROWS` (ORA-1422): Una orden `SELECT..INTO` devuelve más de una fila.

Excepciones predefinidas: tipos

- Errores de tipos:
 - `INVALID_NUMBER` (ORA-1722): Falla la conversión a un tipo `NUMBER` o usamos un dato no numérico en lugar de un dato numérico.
 - `VALUE_ERROR` (ORA-6504): Se produjo un error aritmético, de conversión, de truncamiento o de restricciones en una orden procedimental.
Si es una orden SQL se produce la excepción `INVALID_NUMBER`.
 - `ZERO_DIVIDE` (ORA-1476): División por cero.

Excepciones predefinidas: restricciones y entorno

- Restricciones:
 - `DUP_VAL_ON_INDEX` (ORA-0001): Se intentó violar la restricción `UNIQUE`.
- Errores de entorno:
 - `TRANSACTION_BACKED_OUT` (ORA-0061): Transacción cancelada por bloqueos.
 - `LOGIN_DENIED` (ORA-1017): Falla nombre de usuario o contraseña.
 - `STORAGE_ERROR` (ORA-6500): Error interno: PL/SQL se queda sin memoria.
 - `PROGRAM_ERROR` (ORA-6501): Error interno (el motor PL/SQL de Oracle falló).
 - `TIMEOUT_ON_RESOURCE` (ORA-0051): Fin de intervalo cuando se esperaba un recurso.

Gestión de Excepciones

- Gestor de excepciones `OTHERS`: Siempre incluir uno en el bloque superior.
- Usar funciones `SQLCODE` y `SQLERRM(Código)`
`SQLERRM` son 512 bytes. Usar `SUBSTR(SQLERRM,1,long)`.