PL/SQL

1. Declaraciones
   - Variables y constantes
   - Cursores
   - Excepciones
2. Instrucciones
   - Asignación
   - Control de flujo
   - Bucles
3. Manejadores de excepciones
4. Anidamiento de bloques
5. Tipos de datos estructurados


```
DECLARE
      /* Parte Declarativa */
BEGIN
      /* Parte de Ejecución */
EXCEPTION
      /* Parte de Excepciones */
END;
```

Declaración de variables y constantes

```
declaración_variables_contantes
      ::= identificador [CONSTANT] tipo [NOT NULL]
          [:= inicializ];
tipo
      ::= tipo_escalar | identificador%TYPE |
          identificador%ROWTYPE
tipo_escalar
      ::= NUMBER [(dig)] | NUMBER(dig,prec) | DATE |
          CHAR [(tam)] | VARCHAR [(tam)] | BOOLEAN
```


- Los identificadores que no se inicializan son nulos (`NULL`).
- La cláusula `CONSTANT` indica la definición de una constante cuyo valor no puede ser modificado. En estos casos, se debe incluir la inicialización de la constante en su declaración.
- La cláusula `NOT NULL` impide que una variable sea nula, y por tanto debe inicializarse a un valor no nulo.

- `NUMBER(dig,prec)` : valor numérico (entero o real)
  - `dig` : número de dígitos
  - `prec` : número de dígitos a la derecha del punto decimal
- `CHAR(tam)` : valor alfanumérico
  - `tam` : tamaño máximo de la cadena de carácteres, su valor por defecto es 1
- `VARCHAR` : sinónimo de `CHAR`
- `BOOLEAN` : puede tomar los valores `TRUE`, `FALSE` o `NULL`


```
numero NUMBER NOT NULL := 8;
pi CONSTANT NUMBER(7,6) := 3.141592;
caracter CHAR := 'F';
cadena CONSTANT CHAR(21) := 'MAGDALENA FESTA PLENA';
cond BOOLEAN;
```

- `%TYPE` define una variable o constante escalar a partir del tipo de otro identificador o columna de una tabla
- `%ROWTYPE` define una variable tupla a partir de otra variable tupla o del nombre de una relación

```
car caracter%TYPE;
descripcion articulos.descrip%TYPE;
tupla_fact facturas%ROWTYPE;
tupla_fact2 tupla_fact%ROWTYPE;
```


Declaración de cursores

- `CURSOR` : asocia el nombre del cursor a una consulta
- `OPEN` : ejecuta la consulta, obteniéndose todas aquellas filas de la base de datos que cumplen la condición de búsqueda (*conjunto activo*)
- `FETCH` : devuelve la fila actual del conjunto activo y mueve el cursor de una fila a la siguiente
- `CLOSE` : cierra el cursor

```
declaración_cursores
      ::= CURSOR nombre_cursor [(lista_parám)] IS
          consulta_SQL;
lista_parám
      ::= nombre_parámetro tipo_parámetro |
          nombre_parámetro tipo_parámetro ,
          lista_parámetros
tipo_parámetro
      ::= tipo_escalar
```



```
CURSOR cur_fact_cliente IS
      SELECT codfac, iva, dto
      FROM facturas
      WHERE codcli = :codcli;
CURSOR cur_lin_factura (cod facturas.codfac%TYPE) IS
      SELECT cant, precio, dto
      FROM lineas_fac
      WHERE codfac = cod;
```

Declaración de excepciones

El programador puede definir excepciones de uso específico, cuyo control es enteramente gestionado por él.

```
declaración_excepción
      ::= nombre_excepción EXCEPTION;
```


Instrucciones de asignación

```
asignación_escalar
      ::= variable_objetivo := expresión_PL/SQL;
variable_objetivo
      ::= identificador | :identificador[:indicador] |
          identificador.atributo

      :codcli := 12345;
      :iva:ind_iva := NULL;
      car := 'D';
      tupla_fact.iva = NULL;
```

- Operadores sobre números: `+`, `-`, `*`, `/`, `**` (exponencial), `MOD` (resto)
- Operadores sobre cadenas: `||` (concatenación)
- Operadores lógicos: `AND`, `OR`, `NOT`
- Operadores sobre cursores: `%ROWCOUNT`, `%NOTFOUND`, `%FOUND`, `%ISOPEN`
- Comparadores clásicos: $<, <=, =, =, ^{=}, <>, >=, >$
- Comparadores SQL: `[NOT] LIKE`, `IS [NOT] NULL`, `[NOT] BETWEEN-AND`, `[NOT] IN`

- Funciones sobre cadenas de caracteres: `ASCII`, `CHR`, `INITCAP`, `INSTR`, `LENGTH`, `LOWER`, `LPAD`, `LTRIM`, `REPLACE`, `RPAD`, `RTRIM`, `SOUNDEX`, `SUBSTR`, `TRANSLATE`, `UPPER`
- Funciones numéricas: `ABS`, `CEIL`, `FLOOR`, `MOD`, `POWER`, `ROUND`, `SIGN`, `SQRT`, `TRUNC`
- Funciones sobre fechas: `ADD_MONTHS`, `LAST_DAY`, `MONTHS_BETWEEN`, `NEW_TIME`, `NEXT_DAY`, `ROUND`, `SYSDATE`, `TRUNC`
- Funciones de conversión: `TO_CHAR`, `TO_DATE`, `TO_NUMBER`
- Funciones de control de errores: `SQLCODE`, `SQLERRM`
- Funciones varias: `UID`, `USER`, `DECODE`, `GREATEST`, `LEAST`, `NVL`, `USERENV`

```
asignación_select
      ::= SELECT lista_select INTO lista_variables
          FROM ...;
asignación_cursor
      ::= FETCH nombre_cursor INTO lista_variables;
lista_variables
      ::= variables_escalares | variable_tupla
variables_escalares
      ::= variable_escalar |
          variable_escalar, variables_escalares
```


Instrucciones de control de flujo

```
condicional
      ::= IF expr_lógica THEN intrucciones_PL/SQL
          [ELSIF expr_lógica THEN intrucciones_PL/SQL]
          [ELSE intrucciones_PL/SQL]
          END IF;
```

Bucles

```
bucle_básico
      ::= LOOP intrucciones_PL/SQL END LOOP;
bucle_mientras
      ::= WHILE expr_lógica LOOP
          intrucciones_PL/SQL END LOOP;
bucle_numérico
      ::= FOR control_numérico LOOP
          intrucciones_PL/SQL END LOOP;
```


Bucles

```
control_numérico
      ::= índice IN [REVERSE] expr_ent .. expr_ent
bucle_cursor
      ::= FOR control_cursor LOOP
          intrucciones_PL/SQL END LOOP;
control_cursor
      ::= variable_tupla IN cursor [(lista_argumentos)] |
          variable_tupla IN sentencia_SQL
```

- Bucle básico: la finalización debe forzarse

```
finalización_bucle
        ::= EXIT [WHEN expr_lógica];
```

- Bucle mientras: se ejecuta mientras se cumpla la expresión lógica
- Bucle numérico: finaliza cuando el índice supera el segundo límite (o es menor que éste, si aparece la cláusula `REVERSE`)
- Bucle sobre cursor: finaliza cuando se han extraído todas las filas del conjunto activo

- El índice asociado a un bucle numérico se declara de modo implícito.
- En un bucle sobre un cursor también se declara de modo implícito la variable tupla asociada. Asimismo, se realiza un `OPEN` del cursor al entrar en el bucle y un `CLOSE` al salir, aunque la salida se produzca mediante `EXIT`, y se realiza un `FETCH` implícito en cada iteración del bucle.

```
fac := 1;
LOOP
      fac := fac * i;
      i := i + 1;
      EXIT WHEN (i <= num);
END LOOP;
fac := 1;
WHILE (i <= num) LOOP
      fac := fac * i;
      i := i + 1;
END LOOP;
```



```
fac := 1;
FOR i IN 1..num
      fac := fac * i;
END LOOP;
imp_fac := 0;
FOR li IN cur_lin_factura (facts.codfac) LOOP
      imp_fac := imp_fac + li.cant *
                 li.precio * (1 - li.dto / 100);
END LOOP
```

Manejadores de excepciones

```
manejador_excepciones
      ::= WHEN excepciones THEN instrucciones_PL/SQL
excepciones
      ::= excep_múltiples | OTHERS
excep_múltiples
      ::= excepción | excepción OR excep_múltiples
```


La activación de una excepción la realiza el sistema, cuando se produce un error interno, o el usuario, mediante utilización de la sentencia `RAISE`

```
activar_excepción
      ::= RAISE excepción_válida;
```

Cuando una excepción se activa, la ejecución continúa en la parte de manejadores. Si no la hay, se finaliza la ejecución del bloque y se mantiene la excepción. En la parte de manejadores:

- Se busca, de modo secuencial, un manejador que corresponda con la excepción activada, si no se encuentra se finaliza el bloque, manteniendo activa la excepción
- Una vez encontrado el manejador, se desactiva la excepción y se ejecutan las instrucciones asociadas, finalizándose después el bloque
- Si se desea mantener activa la excepción, o se desea activar cualquier otra, es posible incluir la sentencia `RAISE` (finaliza el bloque y activa la excepción asociada)

- La cláusula `OTHERS` corresponde a todas las excepciones, conviene poner su manejador el último para que los anteriores se puedan ejecutar
- Excepciones predefinidas:
  - `NO_DATA_FOUND, TOO_MANY_ROWS`
  - `INVALID_NUMBER, VALUE_ERROR, ZERO_DIVIDE, DUP_VAL_ON_INDEX`
  - `CURSOR_ALREADY_OPEN, INVALID_CURSOR`
  - `LOGIN_DENIED, NOT_LOGGED_ON`
  - `PROGRAM_ERROR, STORAGE_ERROR, TIMEOUT_ON_RESOURCE`

Anidamiento de bloques

En ocasiones puede resultar interesante realizar un anidamiento de bloques PL/SQL incluyendo un bloque dentro de la parte de instrucciones de otro bloque. En estos casos:

- Todas las constantes, variables y excepciones definidas en el bloque externo son visibles en el bloque interno, pero no al revés
- Si existiera conflicto de nombres entre las declaraciones de los distintos bloques anidados, un bloque toma la declaración más cercana

Tipos de datos estructurados

```
declaración_registro
      ::= TYPE nombre_tipo IS
          RECORD (campo [, campo] ...);
declaración_vector
      ::= TYPE nombre_tipo IS VARRAY (tamaño_maximo)
          OF tipo_datos [NOT NULL];
declaración_tabla
      ::= TYPE nombre_tipo IS TABLE
          OF tipo_datos [NOT NULL];
```

```
TYPE tipo_reg_cliente IS RECORD
        ( nombre VARCHAR(50),
          direccion VARCHAR(50),
          codpostal VARCHAR(5) );

reg_cliente tipo_reg_cliente;
reg_factura factura%ROWTYPE;
```



```
DECLARE
   TYPE tipo_varray IS
      VARRAY(50) OF cliente.nombre%TYPE;
   v_varray1 tipo_varray;
   v_varray2 tipo_varray;
BEGIN
   v_varray1 := tipo_varray('Ana', 'Lola');
      -- se crea con dos elementos
   v_varray1.EXTEND;
   v_varray1(3) := 'Luis';
   -- v_varray1(4) := 'Juan'; ERROR
   v_varray2 := tipo_varray(); -- se crea vacío
   IF v_varray2 IS NULL -- TRUE
   THEN v_varray2 := v_varray1;
      -- asignación de vectores
   END IF;
   ...
```

```
DECLARE
   TYPE tipo_tabla IS
      TABLE OF cliente.nombre%TYPE;
   v_tabla1 tipo_tabla;
   v_tabla2 tipo_tabla;
BEGIN
   v_tabla1 := tipo_tabla('Ana', 'Lola');
   v_tabla1(2) := NULL; -- posición vacía
   v_tabla1.DELETE(1); -- borra posición
   v_tabla1.EXTEND;
   v_tabla1(3) := 'Luis';
   -- v_tabla1(4) := 'Juan'; ERROR
   v_tabla2 := tipo_tabla(); -- se crea vacía
   IF v_tabla1(1).EXISTS -- FALSE
   THEN ...;
   ELSE v_tabla1(1) := 'Pepe'; -- se vuelve a crear
   END IF;
   ...
```


Métodos predefinidos para manejar vectores:

- EXISTS
- COUNT
- FIRST
- LAST
- PRIOR
- NEXT
- EXTEND
- TRIM
- DELETE