1.1. PL/SQL

- **PL/SQL** es un lenguaje de programación desarrollado por Oracle como extensión de SQL.
- No distingue entre letras mayúsculas o minúsculas.
- Existen 3 bloques en PL/SQL: DECLARE, BEGIN, EXCEPTION y END, que dividen el bloque en tres secciones:

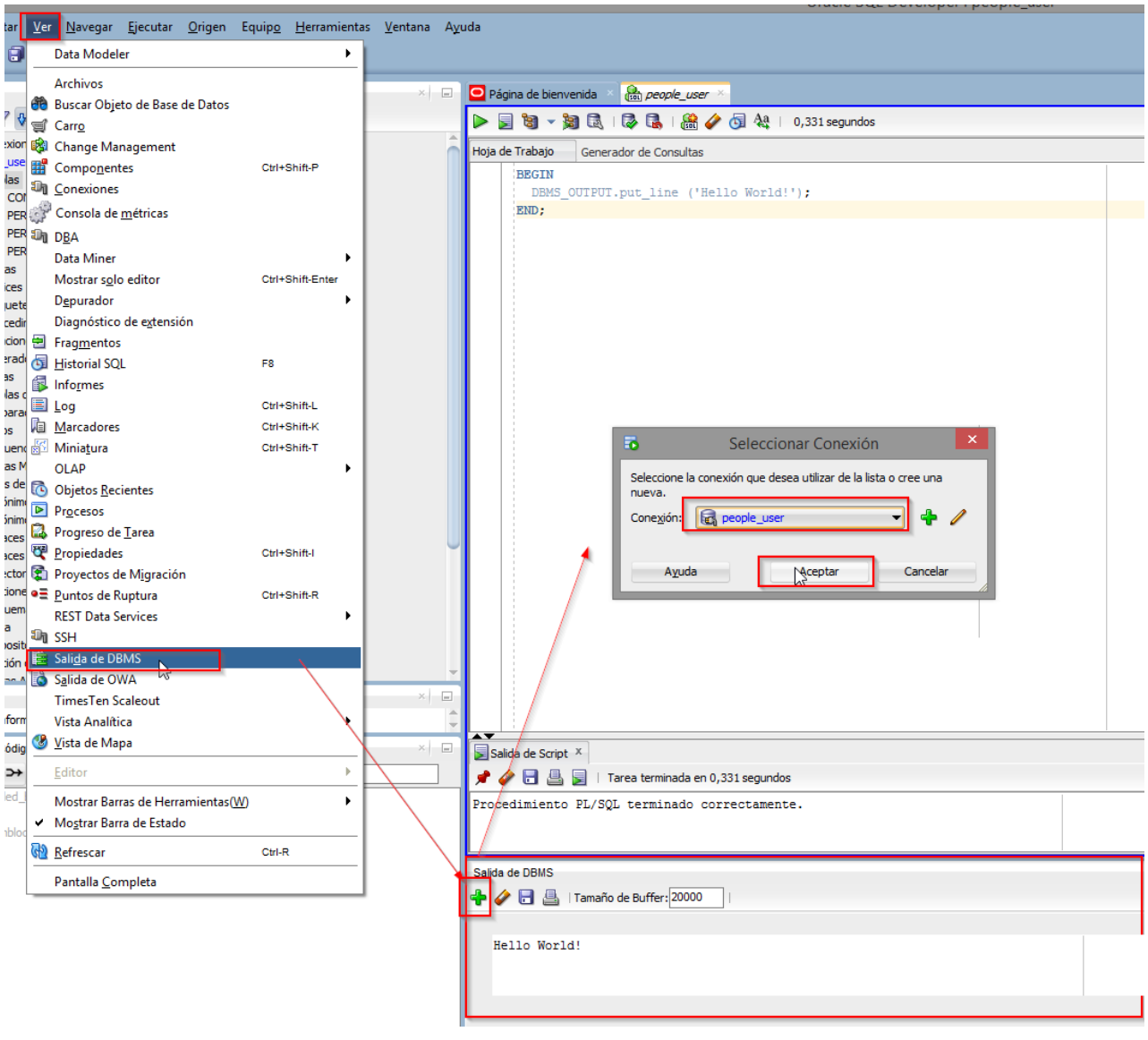
    1. Declarativo: declaraciones que declaran variables, constantes que luego se pueden usar dentro de ese bloque
    2. Ejecutable: declaraciones que se ejecutan cuando se ejecuta el bloque
    3. Manejo de excepciones: una sección especialmente estructurada que puede usar para "atrapar" cualquier excepción que se genere cuando se ejecuta la sección ejecutable

Se puede anidar bloques dentro de otros bloques.

- El clásico "¡Hola mundo!" El bloque contiene una sección ejecutable que llama al procedimiento DBMS_OUTPUT.PUT_LINE para mostrar texto en la pantalla:

```
BEGIN
  DBMS_OUTPUT.put_line ('Hello World!');
END;
```

Para que podamos ver en SQL Developerel resultado de esa salida, tenemos que ir al menú Ver > Salida DBMS > hacer clic sobre el signo + de la ventana de Salida DBMS y seleccionar la conexión para la que queremos habilitarla



Un ejemplo de bloques anidados y de cómo gestionar excepciones se puede ver en el siguiente bloque:

```
----

DECLARE
    l_message VARCHAR2(100) := 'Hello'; -- se declara una variable que empieza por l para indicar que es local
-- (no es estrictamente necesario que empiecen por l_), de tipo VARCHAR2 y
-- se le asigna la cadena 'Hello'.

BEGIN
    DECLARE
        l_message2 VARCHAR2(5); -- 5 bytes no basta para contener World!, por lo que se lanzará una exception

    BEGIN
        l_message2 := 'World!';
        dbms_output.put_line(l_message || l_message2); -- Se concatenan las cadenas con el operador ||

    EXCEPTION
        WHEN OTHERS THEN
            dbms_output.put_line('exception: ' || dbms_utility.format_error_stack);

-- https://docs.oracle.com/en/database/oracle/oracle-database/18/arpls/DBMS_UTILITY.html#GUID-BF8C0CE6-872A-4CD8-9A78-5FB11C2206EC

    END;

    dbms_output.put_line(l_message);
END;
```

Ejemplo de creación de un procedimiento

```
CREATE OR REPLACE PROCEDURE
hello_world
IS
BEGIN
  DBMS_OUTPUT.put_line
  ('Hello World!');
END hello_world;
```

Ejemplos de cómo llamar a un procedimiento de 3 formas diferentes:

```
BEGIN
   hello_world;
   HELLO_WORLD;
   "HELLO_WORLD";
END;
```

Ejemplo de cómo crear una función

```
create or replace FUNCTION
hello_message
   (place_in IN VARCHAR2)
    RETURN VARCHAR2
IS
BEGIN
   RETURN 'Hello ' || place_in;
END hello_message;
```

Ejemplo de cómo llamar a una función

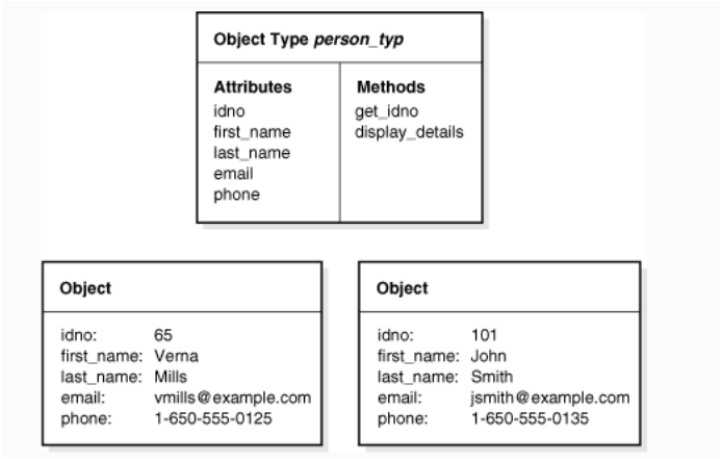
```
DECLARE
  l_message  VARCHAR2 (100);
BEGIN
  l_message := hello_message ('Universe');
  DBMS_OUTPUT.put_line ('l_message contiene: ' || l_message);
END;
```

Más información básica sobre cómo trabajar con PL/SQL: https://www.oracle.com/es/database/technologies/appdev/plsql.html

1.2. Tipos de objetos

Un tipo de objeto es un tipo de tipo de datos como un NUMBER o un VARCHAR2

- Tiene un nombre que lo identifica
- Representa una entidad del mundo real
- Encapsula datos y operaciones
- Al menos un atributo es requerido y los métodos son opcionales
- Cada atributo puede ser de un tipo de datos básico o de un tipo de usuario.
- Sus métodos (procedimientos o funciones) escritos en lenguaje PL/SQL
- Actúan como plantillas para los objetos de cada tipo.



1.3. Creación de tipos de objetos

Se utiliza **CREATE [OR REPLACE] TYPE** para crear o reemplazar la definición de un tipo de objeto:

```
CREATE [OR REPLACE] TYPE person_typ AS OBJECT (
  idno           NUMBER,
  first_name     VARCHAR2(20),
  last_name      VARCHAR2(25),
  email          VARCHAR2(25),
  phone          VARCHAR2(20),
  MAP MEMBER FUNCTION get_idno RETURN NUMBER,
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ));
/

CREATE [OR REPLACE] TYPE BODY person_typ AS
  MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
  BEGIN
    RETURN idno;
  END;
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ) IS
  BEGIN
    -- use the PUT_LINE procedure of the DBMS_OUTPUT package to display details
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' || first_name || ' ' || last_name);
    DBMS_OUTPUT.PUT_LINE(email || ' '  || phone);
  END;
END;
/
```

Atributos

- Se declara mediante un nombre y un tipo de datos.
- El nombre del atributo debe ser único dentro del tipo de objeto (aunque puede reutilizarse en otros objetos)
- El tipo de los atributos puede ser de cualquier otro tipo de objeto o de cualquier tipo de Oracle excepto: LONG, ROWID, BOOLEAN, LONG RAW, MLSLABE, BINARY_INTEGER (y cualquiera de sus subtipos), BOOLEAN, PLS_INTEGER, RECORD y los tipos definidos en los paquetes PL/SQL. (Puede que la lista no sea exhaustiva)
    - Los tipos de datos de Oracle se pueden consultar en la URL: https://docs.oracle.com/en/database/oracle/oracle-database/18/sqlqr/Data-Types.html#GUID-219C338B-FE60-422A-B196-2F0A01CAD9A4.
    - Podéis ver un resumen en español también aquí: https://codigolite.com/tipos-de-datos-en-la-base-de-datos-oracle/

Normalmente trabajaremos con:

- NUMBER(p, s) para enteros y números reales: La *p* es de precisión y es el número total de dígitos permitidos. La *s* es de escala (scale) y es el número de dígitos permitidos a la derecha del punto decimal
- DATE o `TIMESTAMP` para fecha y fecha y hora
- VARCHAR2(size) para cadenas de longitud variable de hasta size bytes (por defecto) o caracteres si se usa CHAR

*__NOCOPY__: Una sugerencia para que el compilador para que use el paso por referencia, por lo que no se necesita un búfer temporal y no se realizan operaciones de copia a la entrada y a la salida del procedimiento. En su lugar, cualquier modificación de los valores de los parámetros se escribe directamente en la variable del parámetro (parámetro real).

1.4. Métodos

- Aportan funcionalidad al tipo de objeto (de forma similar a la POO)
- Usan la palabra clave MEMBER
- No pueden tener el mismo nombre que el tipo de objeto ni el de ninguno de sus atributos
- Pueden ser funciones (FUNCTION) o procedimientos (PROCEDURE)
  - Un procedimiento no puede devolver un valor, una función sí (aunque ambos puedan devolver datos en parámetros `OUT` e `IN OUT`)
  - Las funciones pueden usarse como parte de una expresión (`campo1 * MIFUNCION(campo2)`), los procedimientos no(deben ser llamados de manera independiente).
- Pueden ser métodos estáticos uno para todos las instancias del tipo de objeto (como en Java). Pueden ser funciones o procedimientos

- Los métodos constan de dos partes:
  - La **especificación** consiste en el nombre del método, una lista opcional de parámetros y en el caso de funciones un tipo de retorno.
  - El **cuerpo** es el código que se ejecuta para llevar a cabo una operación específica.
- Para cada especificación de método en una especificación de tipo debe existir el correspondiente **cuerpo** del método

```
CREATE TYPE BODY person_typ AS
  MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
  BEGIN
    RETURN idno;
  END;
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ) IS
  BEGIN
    -- use the PUT_LINE procedure of the DBMS_OUTPUT package to display details
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' || first_name || ' ' || last_name);
    DBMS_OUTPUT.PUT_LINE(email || ' '  || phone);
  END;
END;
```

/

En general la definición del cuerpo es:

```
CREATE OR REPLACE TYPE BODY person_typ AS
```

<una o varias implementaciones de los métodos>
END;

Donde la <implementación de los métodos> será la siguiente para procedimientos:

[STATIC | MEMBER] PROCEDURE nombreProc [(param1, param2, ...)]
IS
  --declaraciones;
BEGIN
  --instrucciones;
END;

Y la <implementación de los métodos> será la siguiente para funciones:

[STATIC | MEMBER | CONSTRUCTOR] FUNCTION nombreFunction [(param1, param2, ...)] RETURN tipo_valor retorno
IS
  --declaraciones;
BEGIN
  --instrucciones;
END;

El parámetro SELF

Todos los métodos de un tipo de objeto aceptan como primer parámetro una instancia predefinida del mismo tipo denominada **SELF**. (Algo parecido a this en Java)

Independientemente de que se declare implícita o explícitamente, SELF es siempre el primer parámetro pasado a un método. Por ejemplo, el método display_details declara SELF como un parámetro IN OUT (de entrada y salida).

El modo de acceso por omisión de SELF, es decir, cuando no se declara explícitamente es:

- En **funciones** miembro el acceso de SELF es **IN**.
- En **procedimientos**, si **SELF** no se declara, su modo por omisión es **IN OUT.**

Para hacer referencia a sus propios atributos en el cuerpo de un método se puede usar:
SELF.idno o simplemente **idno** (como en Java *this.atributo* o simplemente *atributo*)

1.5. Constructores y declaraciones de objetos

- Todos los tipos de objetos tienen asociado por defecto un método que construye nuevos objetos
- El nombre del método constructor coincide con el nombre del tipo
- Sus parámetros son los atributos del tipo y deberán aportarse en el mismo orden que en su definición
- Pueden crearse constructores adicionales con diferente número de parámetros
- Se puede declarar una variable de tipo person_typ e inicializarla mediante el constructor

```
  DECLARE
  person person_typ := null; -- person is atomically null
BEGIN
-- call the constructor for person_typ
  person :=  person_typ (69, 'Amy', 'Doe', 'adoe@example.com', '1-677-555-0127');
  DBMS_OUTPUT.PUT_LINE(person.first_name || ' ' || person.last_name); -- display details

END;
/
```

Hasta que se inicializa un objeto, invocando al constructor para ese tipo de objeto, el objeto se dice que es atómicamente nulo.

Es una buena práctica inicializar las variables de tipo objeto a null

```
CREATE OR REPLACE TYPE rectangle AS OBJECT
(
  length NUMBER,
  width NUMBER,
  area NUMBER,
-- Define un constructor que solo tiene 2 parámetros
  CONSTRUCTOR FUNCTION rectangle(length NUMBER, width NUMBER)
    RETURN SELF AS RESULT
);
/

CREATE OR REPLACE TYPE BODY rectangle AS
  CONSTRUCTOR FUNCTION rectangle(length NUMBER, width NUMBER)
    RETURN SELF AS RESULT
  AS
  BEGIN
    SELF.length := length;
    SELF.width := width;
    SELF.area := length * width;
    RETURN;
  END;
END;
/
```

1.6. Métodos de comparación

Para comparar los objetos de cierto tipo hay que utilizar entre añadir al tipo de objeto un método MAP u ORDER

- Un método **MAP**:
  - Sirve para indicar cuál de los atributos del tipo de objeto se utilizará para ordenar los objetos del tipo
    El PL/SQL usa esta función para evaluar expesiones booleanas como x > y y para las comparaciones implícitas que requieren las claúsulas DISTINCT, GROUP BY y ORDER BY.
  - Un tipo de objeto puede contener solo una función de MAP
  - MAP debe carecer de parámetros
  - Debe devolver uno de los siguientes tipos escalares: DATE, NUMBER, VARCHAR2 y cualquiera de los tipos ANSI SQL (como CHARACTER o REAL).

- Un método ORDER
  - Utiliza los atributos del objeto sobre el que se ejecuta para realizar un cálculo y compararlo con otro objeto del mismo tipo que toma como argumento de entrada.
  - Un método ORDER devolverá:
    - Un valor negativo si el parámetro de entrada es mayor que el atributo
    - Un valor positivo si ocurre lo contrario
    - Un cero si ambos son iguales.

```
create or replace TYPE person_typ2 AS OBJECT (
  idno           NUMBER,
  first_name     NVARCHAR2(20),
  last_name      VARCHAR2(25),
  email          VARCHAR2(25),
  phone          VARCHAR2(20),
  ORDER MEMBER FUNCTION compara (p2 in person_typ2) RETURN NUMBER,
  MEMBER  PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ2 ));
/

  CREATE or REPLACE TYPE BODY person_typ2 AS
  ORDER MEMBER FUNCTION compara (p2 IN person_typ2) RETURN NUMBER IS
  BEGIN
        RETURN idno-p2.idno;
  END;
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ2 ) IS
  BEGIN
    -- use the PUT_LINE procedure of the DBMS_OUTPUT package to display details
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' || first_name || ' ' || last_name);
    DBMS_OUTPUT.PUT_LINE(email || ' '  || phone);
  END;
END;
/
```

1.7. Tablas que almacenan tipos de objetos

Tablas de objetos

Una tabla de objetos es una clase especial de tabla que almacena un objeto en cada fila y que facilita el acceso a los atributos de esos objetos como si fueran columnas de la tabla.

```
CREATE TABLE person_obj_table OF person_typ;
```

Oracle permite considerar una tabla de objetos desde dos puntos de vista:

• Como una tabla con una sola columna cuyo tipo es el de un tipo de objetos.

• Como una tabla que tiene tantas columnas como atributos los objetos que almacena.

```
INSERT INTO person_obj_table VALUES (
       person_typ(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135') );
INSERT INTO person_obj_table VALUES (
       person_typ(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135') );
-- Permite insertar el mismo objeto con el mismo id varias veces si no se establece una clave primaria


-- Incluso se puede insertar tratando como columnas a los atributos
Insert into PEOPLE_USER.PERSON_OBJ_TABLE
(IDNO,FIRST_NAME,LAST_NAME,EMAIL,PHONE)
values ('101','John','Smith','jsmith@example.com','1-650-555-0135');


SELECT VALUE(p) FROM person_obj_table p
        WHERE p.last_name = 'Smith';
```

La function VALUE toma como argumento un **alias de tabla o vista** (también llamada **variable de correlación)** y devuelve instancias de objetos correspondientes a las filas de la tabla o vista.
*Para ver el resultado con el constructor hay que ejecutarlo como script

```
PERSON_TYP(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135')
PERSON_TYP(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135')
```

También se puede señalar una **clave primaria** como uno de los atributos del objeto

```
CREATE TABLE person_obj_table2 OF person_typ2
(idno PRIMARY KEY);
```

De esta forma no será posible insertar dos objetos con el mismo id:

```
INSERT INTO person_obj_table2 VALUES (
```

```
       person_typ2(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135') );
```

```
INSERT INTO person_obj_table2 VALUES (
```

```
       person_typ2(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135') );
```

```
--El segundo insert no se permite por la restricción de clave primaria sobre idno
SELECT VALUE(p) FROM person_obj_table2 p
```

```
        WHERE p.last_name = 'Smith';
```

Tablas con columnas de tipo objeto

- Se puede crear una tabla con columnas de tipo objeto

```
CREATE TABLE contacts (
  contact         person_typ,
  contact_date    DATE );

INSERT INTO contacts VALUES (
  person_typ (65, 'Verna', 'Mills', 'vmills@example.com',
    '1-650-555-0125'),
  to_date('24 Jun 2003', 'dd Mon YYYY'));
```

Sin embargo, cuando se trata de una columna que almacena objetos, **no podremos insertar cada atributo como si fuera una columna**:

```
INSERT INTO contacts VALUES (
 66, 'Watson', 'Joe', 'jwatson@example.com',
    '1-650-444-4444',
  to_date('24 Jun 2008', 'dd Mon YYYY'));

Error SQL: ORA-00913: demasiados valores
00913. 00000 -  "too many values"
```

1.8. Ejemplos de selección

Tablas de objetos

--Estos dos selects devuelven lo mismo

```
SELECT * FROM person_obj_table p;
```

```
SELECT p.* FROM person_obj_table p;
```

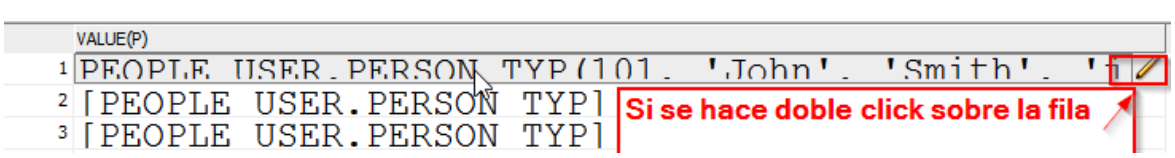
	IDNO	FIRST_NAME	LAST_NAME	EMAIL	PHONE
1	101	John	Smith	jsmith@example.com	1-650-555-0135
2	101	John	Smith	jsmith@example.com	1-650-555-0135

--El siguiente select devuelve la instancia de tipo person_typ **si se ejecuta como script.**

```
SELECT value(p) FROM person_obj_table p;
```

```
VALUE(P)(IDNO, FIRST_NAME, LAST_NAME, EMAIL, PHONE)
--------------------------------------------------------------------------------
PERSON_TYP(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135')
PERSON_TYP(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135')
```

Si se ejecuta como sentencia, se verá solo el tipo del objeto y si se hace doble clic, aparecerá un icono de lápiz que permitirá ver los valores del objeto.




```
SELECT value(p).get_idno()  FROM person_obj_table p;
```

	VALUE(P).GET_IDNO()
1	101

-- Se puede seleccionar una columna/atributo de una tabla que almacena solo objetos de un tipo sin usar alias

```
select idno
from person_obj_table;
```

	IDNO
1	101
2	101
3	101

Tablas con columnas de tipo objeto

```
select * from contacts;
```

```
CONTACT(IDNO, FIRST_NAME, LAST_NAME, EMAIL, PHONE)
--------------------------------------------------------------------------------
CONTACT_
--------
PERSON_TYP(65, 'Verna', 'Mills', 'vmills@example.com', '1-650-555-0125')
24/06/03
```

```
select contact from contacts;
```

```
CONTACT(IDNO, FIRST_NAME, LAST_NAME, EMAIL, PHONE)
--------------------------------------------------------------------------------
PERSON_TYP(65, 'Verna', 'Mills', 'vmills@example.com', '1-650-555-0125')
```

-- se puede llamar a una función en el select

```
SELECT c.contact.get_idno() FROM contacts c;
```

```
C.CONTACT.GET_IDNO()
--------------------
                  65
```

```
-- No se puede acceder a un atributo de una columna de tipo objeto si no se usa un alias
select contact.idno from contacts;
ORA-00904: "CONTACT"."IDNO": identificador no válido
```

-- Habría que utilizar un alias para la tabla para poder acceder a atributos del objeto almacenado en la columna contact

```
select c.contact.idno from contacts c;
```

	CONTACT.IDNO
1	65

```
SELECT c.contact.display_details() FROM contacts c;
--ORA-06553: PLS-222: en este ámbito no existe ninguna función cuyo nombre sea 'DISPLAY_DETAILS'
```

```
select value(c.contact) from contacts c;
--error: c.contact no es solo un alias
```

1.9. OIDs

- Cada fila de una tabla tipada (OR) tendrá un identificador del objeto filaOID
- Para guardar esos identificadores Oracle utiliza un tipo REF

```
select REF(P) FROM person_obj_table p;

REF(P)
--------------------------------------------------------------------------------
28020909F76238E05D4E8982C5C0320ED9BFF2CB83CE3CAC814256A0B2BE8338781B260300008F0000
280209A575FBF85BE14F5BA3C0D8A9FC51E2A2CB83CE3CAC814256A0B2BE8338781B260300008F0001
280209ED4BA390C33347B687C8CE2F40A7E879CB83CE3CAC814256A0B2BE8338781B260300008F0002
```

Esto no ocurre con las columnas que almacenan objetos:

```
select REF(C.CONTACT) FROM CONTACTS C;

Error SQL: ORA-00904: "C"."CONTACT": identificador no válido
```

1.10. Referencias y derreferencias a otros objetos

- Las referencias permiten que los atributos de los objetos o las columnas de tablas que almacenen objetos puedan hacer referencia a otros objetos
- Oracle permite hacer este tipo de asociación a través del operador REF
- Un atributo de tipo REF almacena una referencia a un objeto del tipo definido

```
CREATE OR REPLACE TYPE empresa_typ AS OBJECT(
nombre varchar2(30),
NIF varchar(20),
director REF person_typ);
/
create table empresa_obj_table OF empresa_typ;

insert into empresa_obj_table
(nombre, nif, director)
values('My company', '1234-1234',
(select ref(p) from
person_obj_table p
where idno=101 and ROWNUM =1) )

-- https://docs.oracle.com/en/database/oracle/oracle-database/18/sqlrf/ROWNUM-Pseudocolumn.html
```

Si seleccionamos los registros, veremos que contiene el OID del objeto person_typ y no todos sus atributos

```
select * from empresa_obj_table;

NOMBRE                         NIF
------------------------------ --------------------
DIRECTOR
--------------------------------------------------------------------------------
My company                     1234-1234
22020809F76238E05D4E8982C5C0320ED9BFF2CB83CE3CAC814256A0B2BE8338781B26
```

Las referencias no se pueden navegar en PL/SQL a través del punto.

Para poder transformar la referencia a un objeto a un objeto hay que usar **DEREF:**

```
DECLARE
l_person_typ person_typ := null;
BEGIN
    SELECT  DEREF(e.director) INTO l_person_typ from empresa_obj_table e where rownum = 1;
    l_person_typ.display_details();
END;
```

Aunque en SQL, se puede acceder a un atributo de la referencia porque se realiza un DEREF implícito:

```
select e.director.first_name from empresa_obj_table e;
DIRECTOR.FIRST_NAME
--------------------

John
```

1.11. Acotar referencias

Es posible acotar las referencias con SCOPE a los objetos de un determinado tipo que deban estar en una determinada tabla:

```
CREATE TYPE cust_address_typ_new AS OBJECT
    ( street_address     VARCHAR2(40)
    , postal_code        VARCHAR2(10)
    , city               VARCHAR2(30)
    , state_province     VARCHAR2(10)
    , country_id         CHAR(2)
    );
/
CREATE TABLE address_table OF cust_address_typ_new;

CREATE TABLE customer_addresses (
   add_id NUMBER,
   address REF cust_address_typ_new
   SCOPE IS address_table);
```

Probamos a insertar datos en ambas tablas:

```
insert into address_table
   values( cust_address_typ_new('Gran Vía 23', '36208', 'Vigo', 'Pontevedra', 'ES'));
```

```
insert into customer_addresses values (1, (select ref(addt) from address_table addt where rownum =1 ));
```

Consultamos los datos insertados:

```
SELECT * FROM ADDRESS_TABLE;
SELECT * FROM customer_addresses;
```

```
STREET_ADDRESS                         POSTAL_COD CITY                         STATE_PROV CO
-------------------------------------- ---------- ------------------------------ ---------- --
Gran Vía 23                            36208      Vigo                           Pontevedra ES


    ADD_ID
----------
ADDRESS
--------------------------------------------------------------------------------------------------------------
         1
220208D0561A88A8FA42669505E2BC93AE40F1EC4A647F9FC648F482DBFBC500492E67
```

Si se intenta insertar una referencia a un objeto cust_address_typ_new que no esté en la tabla address_table, no se pemitirá el insert:

```
CREATE TABLE address_table2 OF cust_address_typ_new;
```

```
```

```
   insert into address_table2 values(  cust_address_typ_new('Príncipe 12', '36201', 'Vigo', 'Pontevedra', 'ES'));
```

```
   insert into customer_addresses values (1, (select ref(addt) from address_table2 addt where rownum =1 ));
```

Error SQL: ORA-22889: el valor REF no apunta a la tabla de ámbito
22889. 00000 - "REF value does not point to scoped table"

1.12. Referencias circulares

Si existen referencias circulares entre dos tipos A y B , habrá que definir inicialmente un tipo incompleto.

Para resolverlo se utilizan las Forward Type Definitions:

- Declarar A (tipo incompleto), crear B y crear A
- Recompilar A

```
CREATE or replace TYPE Empleado; -- tipo incompleto
/
CREATE or replace TYPE Departamento AS OBJECT (
numero NUMBER,
jefe REF Empleado );
/
CREATE or replace TYPE Empleado AS OBJECT (
nombre VARCHAR2(20),
dept REF Departamento
);
/
```

1.13. Tipos colección: VARRAY

El tipo VARRAY

- VARRAY es un tipo de datos que se usa para un conjunto ordenado de elementos del mismo tipo.
- Cada elemento tiene asociado un índice que indica su posición dentro del array.
- El constructor por defecto tiene el nombre del tipo. Los argumentos de entrada de estas funciones son el conjunto de elementos que forman la colección separados por comas y entre paréntesis, y el resultado es un valor del tipo colección.
- Oracle permite que los VARRAY sean de longitud variable, aunque es necesario especificar un tamaño máximo cuando se declara el tipo VARRAY.
- Cuando se declara un tipo VARRAY no se produce ninguna reserva de espacio. Se almacena junto con el resto de columnas de su tabla, pero si es demasiado largo (más de 4000 bytes) se almacena aparte de la tabla como un BLOB.
- La principal limitación del tipo VARRAY es que en las consultas es imposible poner condiciones sobre los elementos almacenados dentro. Desde una consulta SQL, los valores de un VARRAY solamente pueden ser accedidos y recuperados como un bloque. Sin embargo, desde un programa PL/SQL si que es posible definir un bucle que itere sobre los elementos de un VARRAY.

- VARRAY se pueden usar para:
  - Definir el tipo de dato de una columna de una tabla relacional.
  - Definir el tipo de dato de un atributo de un tipo de objeto.
  - Para definir una variable PL/SQL, un parámetro o el tipo que devuelve una función

Más sobre VARRAYS: https://docs.oracle.com/en/database/oracle/oracle-database/18/sqlrf/Data-Types.html#GUID-EAA3885B-06AA-4F0D-85E7-C43352E5E2AC

```
CREATE TYPE lista_tel_typ AS VARRAY(10) OF VARCHAR2(20) ;
/
create or replace TYPE person_typ_telefonos AS OBJECT (
  idno           NUMBER,
  first_name     NVARCHAR2(20),
  last_name      VARCHAR2(25),
  email          VARCHAR2(25),
  phones     lista_tel_typ,

 -- MAP MEMBER FUNCTION get_idno RETURN NUMBER,
  ORDER MEMBER FUNCTION compara (p2 in person_typ_telefonos) RETURN INTEGER,
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ_telefonos ));
  /

 create or replace TYPE BODY person_typ_telefonos AS
   ORDER MEMBER FUNCTION compara (p2 in person_typ_telefonos) RETURN INTEGER IS
  BEGIN
        RETURN idno-p2.idno;
  END;
  MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ_telefonos ) IS
  BEGIN
    -- use the PUT_LINE procedure of the DBMS_OUTPUT package to display details
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' || first_name || ' ' || last_name);
    DBMS_OUTPUT.PUT_LINE(email);
     FOR J IN 1.. phones.COUNT LOOP
     DBMS_OUTPUT.PUT_LINE('Teléfono ' || to_char(J) || ': '|| phones(J));
     END LOOP;
  END;
END;
/

CREATE TABLE AGENDA OF  person_typ_telefonos (idno PRIMARY KEY);

INSERT INTO agenda values ( person_typ_telefonos(101, 'John', 'Smith', 'jsmith@example.com',
lista_tel_typ('986 123 123', '982 12 34 56', '981 13 57 98')));

INSERT INTO agenda values ( person_typ_telefonos( 65, 'Verna', 'Mills', 'vmills@example.com',
lista_tel_typ('91 123 123', '92 12 34 56', '902 13 57 98')));
```

Si recorremos uno a uno los registros de SELECT * FROM agenda con un CURSOR

```
DECLARE
    CURSOR cursor_agenda IS
    SELECT
        *
    FROM
        agenda;

    persona   person_typ_telefonos := NULL;
    telefonos lista_tel_typ := NULL;
BEGIN
    FOR cursor_registro IN cursor_agenda LOOP
       -- creamos un objeto de tipo lista_tel_typ
       telefonos := lista_tel_typ();
       -- recorremos cada telefono guardado en phones mostrando el nº de teléfono
        FOR j IN 1..cursor_registro.phones.count LOOP
            dbms_output.put_line('Person con idno: ' || To_char(cursor_registro.idno) || ' tiene el teléfono: ' || cursor_registro.phones(j));
       -- extend añade un elemento null a la colección
       --https://docs.oracle.com/cd/B12037_01/appdev.101/b10807/13_elems006.htm
            telefonos.extend();
       -- last devuelve el índice numérico del último elemento. Si está vacío devuelve null
       -- en varrays first es 1 y last es count
       -- Guardamos en la variable telefonos uno a uno los telefonos recuperados del registro de la tabla agenda
            telefonos(telefonos.last) := cursor_registro.phones(j);
        END LOOP;

       -- cursor_registro no es un objeto de tipo person_typ_telefonos,
       -- sino una estructura de datos con los mismos atributos (pero no los mimos métodos)
       -- Para invocar un método, hay que tener una instancia del tipo person_typ_telefono
        persona := person_typ_telefonos(
                                    cursor_registro.idno,
                                    cursor_registro.first_name,
                                    cursor_registro.last_name,
                                    cursor_registro.email,
                                    telefonos
                );

        persona.display_details();
    END LOOP;
END;
/
```