7. Uso de bases de datos objeto - relacionales

Class	Bases de datos
Column	Xerach Casanova
Lass Edited time	@Mar 15, 2021 1:06 PM

1. Características de las bases de datos objeto-relacionales

Son aquellas que han evolucionado desde el modelo relacional tradicional a un modelo híbrido que utiliza tecnología orientada a objetos. Clases, objetos y herencia son soportados en los esquemas de la base de datos y en el lenguaje de consulta y manipulación de datos. Además, da soporte a una extensión del modelo de datos con creación personalizada de tipos de datos y métodos.

Oracle implementa el modelo orientado a objetos como una extensión del modelo relacional.

El modelo objeto-relacional ofrece las ventajas de la reutilización de los objetos y a la vez mantiene la capacidad de concurrencia y rendimiento de bd relacionales.

Los tipos de objetos y las características orientadas a objetos permiten organizar los datos y acceder a ellos a alto nivel. Por debajo de la capa de objetos, los datos seguirán siendo almacenados en columnas y tablas, pero se trabaja con ellos de manera más parecida a entidades de la vida real, dando más significado a los datos.

El modelo orientado a objetos es similar a lenguajes como C++ o Java. La reutilización de objetos permite desarrollar aplicaciones de bases de datos más rápidamente y de manera más eficiente. Se permite además a los desarrolladores de aplicaciones acceder a las mismas estructuras de datos creadas en bases de datos.

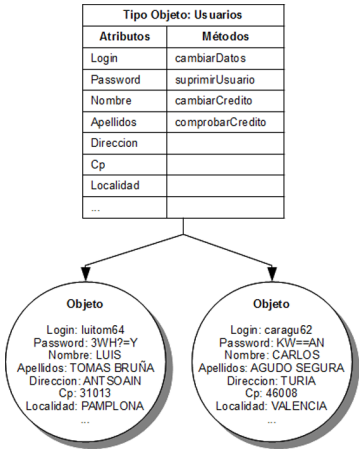
La programación orientada a objetos está enfocada a la construcción de componentes reutilizables. En PL/SQL la programación orientada a objetos está basada en tipos de objetos. Estos tipos permiten modelar objetos de la vida real, separar detalles de interfaces de usuarios e implementación y almacenar los datos de forma permanente en la base de datos. Son especialmente útiles cuando se realizan programas interconectados con Java u otros lenguajes POO.

Las tablas de bases de datos relacionales solo contienen datos, en cambio los objetos pueden incluir acciones sobre ellos. Un objeto compra puede incluir el método para calcular el importe de todos los elementos comprados. Un objeto cliente puede tener métodos para acceder al historial de compras.

2. Tipos de dato objeto

Un tipo de dato objeto es un tipo de dato compuesto definido por el usuario. Representa una estructura de datos, junto con funciones y procedimientos para manipularlos. Las colecciones permiten almacenar varios datos en una misma variable siendo todos del mismo tipo. Los tipos de datos objetos nos permiten almacenar datos de distinto tipo y asociar código a dichos datos.

Las variables que forman la estructura de datos de un tipo de dato objeto reciben el nombre de atributos y se corresponde con sus propiedades. Las funciones o procedimiento del tipo de dato objeto se denominan métodos y son sus acciones.



Al definir un tipo de dato objeto se crea una plantilla abstracta de un objeto real. La plantilla especifica atributos y comportamiento en el entorno de la aplicación. Dependiendo de la aplicación a desarrollar se utilizan unos determinados atributos y comportamientos y se descartan los que no son necesarios.

Los atributos son públicos y visibles desde otros programas cliente, pero los programas deben manipular los datos únicamente a través de los métodos (funciones o procedimientos) en vez de asignar u obtener sus valores directamente para mantener el estado apropiado de los atributos.

Durante la ejecución, una aplicación crea instancias de un tipo de objeto, con valores asignados en sus atributos.

3. Definición de tipos de objeto

La definición o declaración de un tipo de objeto está dividida en una especificación y un cuerpo. La especificación define la interfaz de programación, se declaran los atributos y operaciones o métodos para manipular los datos. El cuerpo se implementa en el código fuente de los métodos.

Toda la información que un programa necesita para usar los métodos se encuentra en la especificación, pudiendo modificar el cuerpo sin que afecte a los programas cliente.

En la especificación se declaran los atributos antes que los métodos. Si el objeto solo tiene atributos no es necesario declarar cuerpo. En el cuerpo no se declaran atributos. Todas las declaraciones realizadas en la especificación del tipo de objeto son públicas y por lo tanto visibles.

Un tipo objeto encapsula datos y operaciones. Se pueden declarar atributos y métodos en la especificación, pero no constantes (constants), excepciones (exceptions), cursores (cursors) o tipos (types). Se debe tener un atributo declarado como mínimo y un máximo de 1000. Los métodos son opcionales.

Para crear objetos en Oracle:

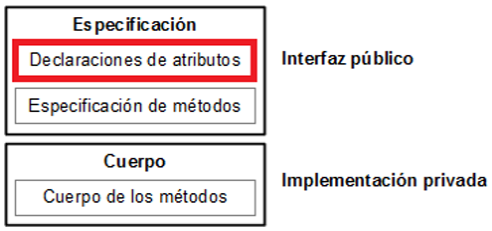
```
CREATE [OR REPLACE] TYPE nombre_tipo AS OBJECT (
Declaración_atributos
Declaración_métodos
);
```

nombre_tipo es el nombre del objeto y debe ser único. Si se quiere reemplazar se utiliza OR REPLACE.

Para eliminar objetos:

```
DROP TYPE nombre_tipo;
```

3.1. Declaración de atributos



Se puede realizar de forma similar a las variables. Se utiliza un nombre y un tipo de dato, el nombre debe ser único dentro del tipo de objeto y puede ser reutilizado en otros tipos de objeto. El tipo de dato en Oracle puede ser cualquiera excepto:

- Long y Longraw
- RowID y URowID
- Los específicils PL/SQL BINARY_INTEGER y sus subtipos, boolean, pls_integer, record, ref cursor, %type y %rowtype.
- Los tipos definidos dentro de un paquete PL/SQL.

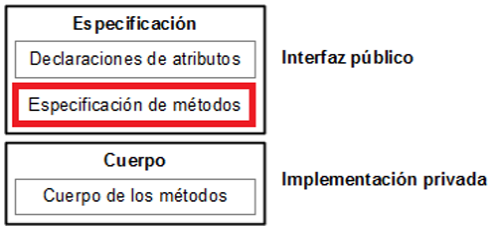
No se deben inicializar atributos usando el operador de asignación, ni cláusula default ni restricción not null. El tipo de dato puede ser otro tipo de objeto.

```
CREATE OR REPLACE TYPE Usuario AS OBJECT (
  login VARCHAR2(10),
  nombre VARCHAR2(30),
  f_ingreso DATE,
  credito NUMBER
);
/
```

Para crear o modificar atributos después de haber sido creado el tipo de objeto se utiliza ALTER TYPE y ADD, MODIFY o DROP ATTRIBUTE

```
ALTER TYPE Usuario DROP ATTRIBUTE f_ingreso;
ALTER TYPE Usuario ADD ATTRIBUTE (apellidos VARCHAR2(40), localidad VARCHAR2(50));
ALTER TYPE Usuario
  ADD ATTRIBUTE cp VARCHAR2(5),
  MODIFY ATTRIBUTE nombre VARCHAR2(35);
```

3.2. Definición de métodos

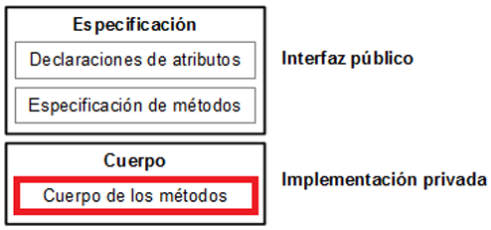


Un método es un subprograma que declaras en la especificación de un tipo de objeto usando las palabras MEMBER o STATIC. El nombre del método no puede ser del mismo que el tipo de objeto ni de algún atributo.

Los métodos tienen dos partes: especificación y cuerpo.

La especificación asigna el nombre al método con sus parámetros opcionales, en el caso de funciones, un tipo de dato de retorno.

```
CREATE OR REPLACE TYPE Usuario AS OBJECT (
  login VARCHAR2(10),
  nombre VARCHAR2(30),
  f_ingreso DATE,
  credito NUMBER,
  MEMBER PROCEDURE incrementoCredito(inc NUMBER)
);
/
```



En el cuerpo se indica el código que debe ejecutar para realizar una determinada tarea cuando es invocado.

```
CREATE OR REPLACE TYPE BODY Usuario AS
  MEMBER PROCEDURE incrementoCredito(inc NUMBER) IS
    BEGIN
      credito := credito + inc;
    END incrementoCredito;
END;
/
```

Por cada especificación que se indique debe existir su correspondiente cuerpo del método, o bien, declararse como NOT INSTANTIABLE, para indicar que el cuerpo del método se encontrará en un subtipo de ese tipo de objeto. El nombre de las cabeceras deben coincidir en especificación y cuerpo.

Los parámetros formales se declaran con nombre y tipo de dato pero sin restricciones de tamaño, el tipo de dato puede ser de los mismos tipos que para los atributos y se aplican las mismas restricciones para los tipos de retorno en las funciones.

El código fuente no solo puede escribirse en PL/SQL, sino también en otros lenguajes de programación.

También se puede usar ALTAR TYPE para añadir, modificar o eliminar métodos de manera similar.

3.3. Parámetro SELF

El parámetro SELF se utiliza en los métodos MEMBER, hace referencia a una instancia del mismo tipo de objeto. Es como el this de java. En funciones member si no se declara SELF, su modo por defecto es IN, en procedimientos MEMBER es IN OUT. Los métodos static no pueden utilizar SELF y tampoco se puede especificar el modo OUT.

```
MEMBER PROCEDURE setNombre(Nombre VARCHAR2) IS
  BEGIN
    /* El primer elemento (SELF.Nombre) hace referencia al atributo del tipo de objeto mientras que el
    segundo (Nombre) hace referencia al parámetro del método */
    SELF.Nombre := Nombre;
  END setNombre;
```

3.4. Sobrecarga

Los métodos se pueden sobrecargar (utilizar el mismo nombre siempre que sus parámetros formales sean distintos en cantidad o tipo de dato).

Al hacer una llamada a método se comparan sus parámetros actuales con los formales de los métodos declarados y se ejecuta aquel que coincida.

no es válida una sobrecarga de dos métodos con parámetros formales que se diferencian solamente en su modo y tampoco funciones que se diferencien solo en el valor de retorno.

```
MEMBER PROCEDURE setNombre(Nombre VARCHAR2)
MEMBER PROCEDURE setNombre(Nombre VARCHAR2, Apellidos VARCHAR2)
```

3.5. Métodos constructores

Todos los objetos tienen un método constructor, que es una función con el mismo nombre que el tipo de objeto e inicializa los atributos, retornando una nueva instancia de ese objeto.

Oracle crea un método constructor por defecto para cada objeto declarado. Sus parámetros formales coinciden en orden, nombres y tipos con los atributos del objeto.

También se pueden declarar constructores propios, reescribiendo el declarado por el sistema o definiendo otro con otros parámetros.

La ventaja de crear un constructor personalizado es el poder verificar que los datos que se van a asignar son correctos.

Si se desea reemplazar el constructor por defecto simplemente se debe utilizar la sentencia CONSTRUCTOR FUNCTION seguida del nombre del tipo de objeto en el que se encuentra. A continuación se indican los parámetros de la manera habitual y por último se indica el valor de retorno, que en este caso es el propio objeto: RETURN SELF AS RESULT.

Se pueden crear varios métodos constructores siguiendo las normas de sobrecarga de métodos.

```
CONSTRUCTOR FUNCTION Usuario(login VARCHAR2, credito NUMBER)
  RETURN SELF AS RESULT

CREATE OR REPLACE TYPE BODY Usuario AS
```

```
  CONSTRUCTOR FUNCTION Usuario(login VARCHAR2, credito NUMBER)
    RETURN SELF AS RESULT
  IS
    BEGIN
      IF (credito >= 0) THEN
        SELF.credito := credito;
      ELSE
        SELF.credito := 0;
      END IF;
      RETURN;
    END;
END;
```

4. Utilización de objetos

4.1. Declaración de objetos

Una vez definido el objeto se puede utilizar para declarar variables de objetos en cualquier bloque PL/SQL, subprograma o paquete.

Ese tipo de objeto lo puedes utilizar como tipo de dato para una variable, atributo, elemento de una tabla, parámetro formal o resultado de una función.

Por ejemplo:

```
u1 Usuario;
```

En la declaración de cualquier procedimiento o función, se puede utilizar el tipo de dato objeto definido para pasarlo como parámetro.

```
PROCEDURE setUsuario(u IN Usuario)
```

La llamada a ese método se realiza utilizando el objeto como parámetro:

```
setUsuario(u1);
```

Para que una función retorne objetos:

```
FUNCTION getUsuario(codigo INTEGER) RETURN Usuario
```

Los objetos se crean durante ejecución como instancias del tipo de objeto y cada uno de ellos puede contener valores diferentes en sus atributos.

El ámbito de los objetos sigue las mismas reglas en PL/SQL. Es decir, se crean o instancian en dicho bloque y se destruyen cuando se sale de ellos. En un paquete, los objetos se instancian cuando se hace referencia al paquete y dejan de existir cuando se finaliza sesión en la bd.

4.2. Inicialización de objetos

Para crear o instanciar objetos de un determinado tipo de objeto se hace llamada a su método constructor con la instrucción NEW seguido del nombre del tipo de objeto como llamada a una función, indicando sus parámetros iniciales.

El orden de parámetros debe coincidir con el orden en que están declarados los atributos.

```
variable_objeto := NEW Nombre_Tipo_Objeto (valor_atributo1, valor_atributo2, ...);
```

Por ejemplo:

```
u1 := NEW Usuario('luitom64', 'LUIS ', 'TOMAS BRUÑA', '24/10/07', 100);
```

hasta que no se inicializa un objeto llamando al constructor, el objeto tiene valor null. Es habitual declarar e inicializar a la vez:

```
u1 Usuario := NEW Usuario('luitom64', 'LUIS ', 'TOMAS BRUÑA', '24/10/07', 100);
```

La llamada al constructor se puede realizar en cualquier lugar en el que se puede hacer una llamada a función de manera habitual. Puede ser, por ejemplo parte de una expresión.

Los valores de los parámetros que se pasan al constructor cuando se hace la llamada son asignados a los atributos del objeto que se está creando. Si la llamada es al constructor por defecto de Oracle, se debe indicar un parámetro para cada atributo en el orden declarado, teniendo en cuenta que los atributos no pueden tener valores por defecto asignados en la declaración.

Existe posibilidad de utilizar nombres de los parámetros formales en la llamada al constructor, en lugar de utilizar el modelo posicional de los parámetros y así no hace falta respetar el orden en que se encuentran los parámetros reales respecto a los formales.

```
DECLARE
u1 Usuario;
BEGIN
u1 := NEW Usuario('user1', -10);
/* Se mostrará el crédito como cero, al intentar asignar un crédito negativo */
dbms_output.put_line(u1.credito);
END;
/
```

4.3. Acceso a los atributos de objetos

Para hacer referencia a un atributo de objeto se debe utilizar el nombre del atributo utilizando un punto para acceder al valor que contiene o para modificarlo, precedido del nombre del objeto.

```
nombre_objeto.nombre_atributo
```

por ejemplo:

```
unNombre := usuario1.nombre;
dbms_output.put_line(usuario1.nombre);
```

La modificación del valor puede ser:

```
usuario1.nombre:= 'Nuevo Nombre';
```

Los nombres de los atributos se pueden encadenar, de manera que se permite acceso a atributos de tipos de objetos anidados:

```
sitio1.usuario1.nombre
```

Si se utiliza una expresión con acceso a un atributo de objeto no inicializado, se evalúa NULL, si s intenta asignar valores a un objeto no inicializado se lanza una excepción ACCESS_INTO_NULL. Se puede comprobar sui un objeto es null con IS NULL.

Al intentar una llamada a un método de objeto no inicializado se lanza una excepción NULL_SELF_DISPATCH, si se pasa como parámetro de tipo IN, los atributos del objeto NULL se evalúan como NULL y si el parámetro es de tipo OUT o IN OUT se lanza una excepción al intentar modificar el valor de los atributos.

4.4. Llamada a los métodos de los objetos

Se puede invocar a los métodos utilizando también un punto entre el objeto y el método, los parámetros reales que separan por comas, entre paréntesis.

```
usuario1.setNombreCompleto('Juan', 'García Fernández');
```

Si el método no tiene parámetros se dejan los paréntesis vacíos o se omiten los paréntesis.

```
credito := usuario1.getCredito();
```

La llamada a métodos puede encadenarse, el orden de ejecución de los métodos se hace de derecha a izquierda y se de debe tener en cuenta que el método de la izquierda debe retornar un objeto del tipo correspondiente al de la derecha.

```
sitio1.getUsuario.setNombreCompleto('Juan', 'García Fernández');
```

Los métodos MEMBER son invocados utilizando una instancia del tipo objeto:

```
nombre_objeto.metodo()
```

Los métodos estáticos se invocan usando el tipo de objeto:

```
nombre_tipo_objeto.metodo()
```

4.5. Herencia

PL/SQL admite herencia simple, con lo cual los subtipos o tipos heredados contienen los atributos y métodos del tipo padre, además de los suyos adicionales o incluso sobreescribir los métodos del tipo padre.

Para indicar que es un tipo de objeto heredado se utiliza la palabra reservada UNDER y además tener en cuenta que el tipo de objeto del que hereda debe tener la propiedad NOT FINAL, ya que por defecto los tipos de objeto son FINAL y no se puede heredar de ellos.

Si no se indica lo contrario, se pueden crear objetos de los tipos de objeto declarado, pero usando la opción NOT INSTANTIABLE se puede declarar tipos de objetos que no se pueden instanciar y deben ser padres de otros tipos de objeto.

En el siguiente ejemplo puedes ver cómo se crea el tipo de objeto Persona, que se utilizará heredado en el tipo de objeto UsuarioPersona. De esta manera, este último tendrá los atributos de Persona más los atributos declarados en UsuarioPersona. En la creación del objeto puedes observar que se deben asignar los valores para todos los atributos, incluyendo los heredados.

```
CREATE TYPE Persona AS OBJECT (
  nombre VARCHAR2(20),
  apellidos VARCHAR2(30)
) NOT FINAL;
/

CREATE TYPE UsuarioPersona UNDER Persona (
  login VARCHAR(30),
  f_ingreso DATE,
  credito NUMBER
);
/

DECLARE
  u1 UsuarioPersona;
BEGIN
  u1 := NEW UsuarioPersona('nombre1', 'apellidos1', 'user1', '01/01/2001', 100);
  dbms_output.put_line(u1.nombre);
END;
/
```

5. Métodos MAP y ORDER

Las instancias de un tipo de objeto no tienen orden predefinido, para establecer uno, con el fin de hacer ordenación o comparación se crea un método map.

Si se hace una comparación entre dos objetos y deseas saber que uno es mayor que otro hay que establecer con un método MAP cual va a ser el valor que se va a utilizar para comparar (nombre, apellidos...).

Para crear un método MAP se declara un método que retorne el valor que se va a utilizar para hacer la comparación. Este método debe empezar con la palabra MAP.

```
CREATE OR REPLACE TYPE Usuario AS OBJECT (
  login VARCHAR2(30),
  nombre VARCHAR2(30),
  apellidos VARCHAR2(40),
  f_ingreso DATE,
  credito NUMBER,
  MAP MEMBER FUNCTION ordenarUsuario RETURN VARCHAR2
);
/
```

En el cuerpo del método se debe retornar el valor que utilizaremos para realizar comparaciones entre instancias:

```
CREATE OR REPLACE TYPE BODY Usuario AS
  MAP MEMBER FUNCTION ordenarUsuario RETURN VARCHAR2 IS
  BEGIN
    RETURN (apellidos || ' ' || nombre);
  END ordenarUsuario;
END;
/
```

El lenguaje PL/SQL utiliza métodos MAP para evaluar expresiones lógicas que resultan valores booleanos como objeto1 > objeto2 y para hacer las comparaciones implícitas de las cláusulas DISTINCT, GROUP BY y ORDER BY

Cada tipo de objeto solo puede tener un método MAP declarado y solo puede retornar: DATE, NUMBER, VARCHAR2, CHARACTER O REAL.

Ejemplo:

Partimos de un tipo direccion_t definido para guardar datos de una dirección:

```
CREATE or replace TYPE direccion_t  AS OBJECT (
   calle VARCHAR2(200),
   ciudad VARCHAR2(200),
   prov CHAR(2),
   codpos VARCHAR2(20)
 );
```

Sea cliente_t un tipo definido para guardar información de un cliente, que contiene dos métodos, uno para calcular la edad de un cliente y otro sería el que contiene el MAP:

```
create or replace TYPE cliente_t AS OBJECT (
     clinum NUMBER,
     clinomb VARCHAR2(200),
     direccion direccion_t,
     telefono VARCHAR2(20),
     fecha_nac DATE,
     MEMBER FUNCTION edad RETURN NUMBER,
     MAP MEMBER FUNCTION ret_value RETURN NUMBER
  );
```

Escribe el código correspondiente a los métodos edad y ret_value que ordenará por el campo clinum. Es decir cuando se comparen dos objetos del tipo cliente_t se compararán por su atributo clinum.

```
create or replace TYPE BODY cliente_t AS
   MEMBER FUNCTION edad RETURN NUMBER IS
     a integer;
     d DATE;
   BEGIN
       select sysdate into d from dual;
       a:=months_between(sysdate,fecha_nac); -- Obtiene el número de meses que hay entre las dos fechas
       DBMS_OUTPUT.PUT_LINE(a);
       a:=trunc(a/12); -- Trunca el resultado de la división quitando la parte decimal
       RETURN a;
   END;
  MAP MEMBER FUNCTION ret_value RETURN NUMBER
   IS
   BEGIN
       RETURN clinum;
   END;
 END;
```

5.1. Métodos ORDER

De forma similar a MAP, se puede declarar en cualquier tipo de objeto un método ORDER que establece un orden entre dos objetos instanciados de dicho tipo.

Cada tipo de objeto solo puede contener un método ORDER y debe devolver un valor numérico que permite establecer el orden entre los objetos. Si deseas que un objeto sea menor que otro, se puede retornar el -1, si van a ser iguales el 0 y si es mayor el 1.

Para declarar que método va a realizar esta operación debes indicar la palabra ORDER delante de la declaración y debe devolver un INTEGER. Necesita además un parámetro del mismo tipo de objeto, el cual será el que se compare con el objeto que utilice el método.

```
CREATE OR REPLACE TYPE BODY Usuario AS
  ORDER MEMBER FUNCTION ordenUsuario(u Usuario) RETURN INTEGER IS
  BEGIN
    /* La función substr obtiene una subcadena desde la posición indicada hasta el final*/
    IF substr(SELF.login, 7) < substr(u.login, 7) THEN
      RETURN -1;
    ELSIF substr(SELF.login, 7) > substr(u.login, 7) THEN
      RETURN 1;
```

```
    ELSE
      RETURN 0;
    END IF;
  END;
END;
```

El método devolverá un -1, 0 o 1, dependiendo de si el objeto que utiliza el método (SELF) es menor, igual o mayor que el objeto pasado por parámetro.

Se puede declarar un método map o un método order, pero no los dos.

Con un número alto de objetos es preferible usar MAP ya que ORDER es menos eficiente.

Ejemplo:

Siguiendo el ejemplo del apartado anterior, tipo cliente_t quedaría:

```
CREATE or replace TYPE cliente_t AS OBJECT (
    clinum NUMBER,
    clinomb VARCHAR2(200),
    direccion direccion_t,
    telefono VARCHAR2(20),
    fecha_nac DATE,
    ORDER MEMBER FUNCTION cli_ordenados (x cliente_t) RETURN INTEGER,
    MEMBER FUNCTION edad RETURN NUMBER);
```

Implementa el método cli_ordenados.

```
ORDER MEMBER FUNCTION cli_ordenados (x IN cliente_t)
RETURN INTEGER IS
BEGIN
      RETURN clinum - x.clinum;  /*la resta de los dos números clinum*/
END;
END;
```

6. Tipos de datos colección

Son como vectores o matrices, pero en este caso solo pueden ser de una dimensión y los elementos se indexan mediante un valor de tipo numérico o cadenas de caracteres.

Oracle proporciona VARRAY y NESTED TABLE (tabla anidada) como tipos de datos colección.

- **VARRAY** es una colección de elementos a la que se le establece un máximo número de elementos al declararse. La longitud es fija y la eliminación de elementos no ahorra espacio en memoria.
- **NESTED TABLE** almacena cualquier número de elementos y tiene un tamaño dinámico, no teniendo por qué existir valores para todas las posiciones de la colección.
- Una variación son los arrays asociativos, que utilizan valores arbitrarios para sus índices y no son necesariamente consecutivos.

Para almacenar un número fijo de elementos o hacer un recorrido entre elementos de forma ordenada, o necesitas obtener y manipular toda la colección como un valor se usa VARRAY

Si se necesita ejecutar consultas sobre una colección de manera eficiente o manipular elementos de manera arbitraria, hacer operaciones de inserción, actualización o borrado masivo se usa NESTED TABLE.

Se pueden declarar como instrucción SQL o en el bloque de declaraciones de un programa PL/SQL. El tipo de dato en PL/SQL puede ser cualquiera excepto REF CURSOR. En SQL no pueden ser: BINARY_INTEGER, PLS_INTEGER, BOOLEAN, LONG, LONG RAW, NATURAL, NATURALN, POSITIVE, POSITIVEN, REF CURSOR, SIGNTYPE Y STRING.

Una tabla de una base de datos puede contener columnas que sean colecciones. Sobre una tabla que contiene colecciones se pueden realizar operaciones de consulta y manipulación de datos de la misma

manera que con tablas habituales.

6.2. Declaración y uso de colecciones

```
TYPE nombre_tipo IS VARRAY (tamaño_max) OF tipo_elemento;
TYPE nombre_tipo IS TABLE OF tipo_elemento;
TYPE nombre_tipo IS TABLE OF tipo_elemento INDEX BY tipo_índice;
```

nompre_tipo es el nombre de la colección, tamaño_max es el número máximo en caso de VARRAY y tipo_elemento es el tipo de elementos que forman la colección, que también pueden ser objetos.

En caso de que la declaración sea en SQL se declara así:

```
CREATE [OR REPLACE] TYPE.
CREATE TYPE nombre_tipo IS ...
```

tipo_índice representa el tipo de dato para el índice. Puede ser PLS_INTEGER, BINARY_IINTEGER o VARCHAR2. Este último lleva entre paréntesis el tamaño.

Hasta que no sea iniciada, la colección es NULL, para inicializar una colección se usa el constructor y se le pasan como parámetros los valores iniciales de la colección:

```
DECLARE
  TYPE Colores IS TABLE OF VARCHAR(10);
  misColores Colores;
BEGIN
  misColores := Colores('Rojo', 'Naranja', 'Amarillo', 'Verde', 'Azul');
END;
```

La inicialización se puede realizar en el bloque de código del programa o directamente en el bloque de declaraciones:

```
DECLARE
  TYPE Colores IS TABLE OF VARCHAR(10);
  misColores Colores := Colores('Rojo', 'Naranja', 'Amarillo', 'Verde', 'Azul');
```

Para obtener uno de los elementos de la colección o modificar su contenido se debe indicar el nombre de la colección y entre paréntesis el índice que ocupa. Tanto en VARRAY como en NESTED TABLE el primer elemento es 1.

```
dbms_output.put_line(misColores(2));
```

```
misColores(3) := 'Gris';
```

En el siguiente ejemplo puedes comprobar cómo pueden utilizarse las colecciones para almacenar sus datos en una tabla de la base de datos, así como la utilización con sentencias de consulta y manipulación de los datos de la colección que se encuentra en la tabla.

```
CREATE TYPE ListaColores AS TABLE OF VARCHAR2(20);
/
CREATE TABLE flores (nombre VARCHAR2(20), coloresFlor ListaColores)
  NESTED TABLE coloresFlor STORE AS colores_tab;
```

```
DECLARE
  colores ListaColores;
BEGIN
  INSERT INTO flores VALUES('Rosa', ListaColores('Rojo','Amarillo','Blanco'));
  colores := ListaColores('Rojo','Amarillo','Blanco','Rosa Claro');
  UPDATE flores SET coloresFlor = colores WHERE nombre = 'Rosa';
  SELECT coloresFlor INTO colores FROM flores WHERE nombre = 'Rosa';
END;/
```

Al definir una tabla que contiene un atributo que es de tipo tabla, es necesario darle un nombre de almacenamiento a NESTED TABLE mediante la clausula STORE AS que es un nombre interno y no se puede utilizar para acceder directamente a la tabla anidada.

7. Tablas de objetos

También se pueden almacenar objetos en tablas de igual manera que los tipos de datos habituales.

Los tipos de dato objeto se pueden usar para formar una tabla exclusivamente formado por elementos de ese tipo, o también para usarse como un tipo de columna más entre otras columnas de otros tipos de datos.

Para crear una tabla formada por un determinado tipo de dato objeto o tabla de objetos:

```
CREATE TABLE NombreTabla OF TipoObjeto;
```

Siendo nombreTabla el nombre de la tabla que almacenará los objetos de tipoObjeto:

```
CREATE TABLE UsuariosObj OF Usuario;
```

Si una tabla hace uso de un tipo objeto, no se puede modificar la estructura ni eliminar dicho tipo de objeto, Desde que un tipo objeto es utilizado en una tabla no se puede volver a definir.

Al crear una tabla de esta manera podemos almacenar objetos de tipo usuario en una tabla, quedando sus datos persistentes mientras no sean eliminados de la tabla. Hasta ahora, al guardar los objetos en variables, al terminar la ejecución, la información que contienen desaparece.

Cuando se instancia un objeto con el fin de almacenarlo en una tabla, este objeto no tiene identidad fuera de la tabla de la bd, pero el tipo de objeto existe independiente de cualquier tabla, para crear objetos en cualquier modo.

Las tablas que contienen solo filas con objetos se llaman tablas de objetos. Los atributos se muestran como columnas de la tabla.

```
SQL> select * from usuariosobj;
LOGIN      NOMBRE     APELLIDOS            F_INGRES    CREDITO
---------- ---------- -------------------- -------- ----------
luitom64   LUIS       TOMAS BRUNA          24/10/07         50
caragu72   CARLOS     AGUDO SEGURA         06/07/07        100
```

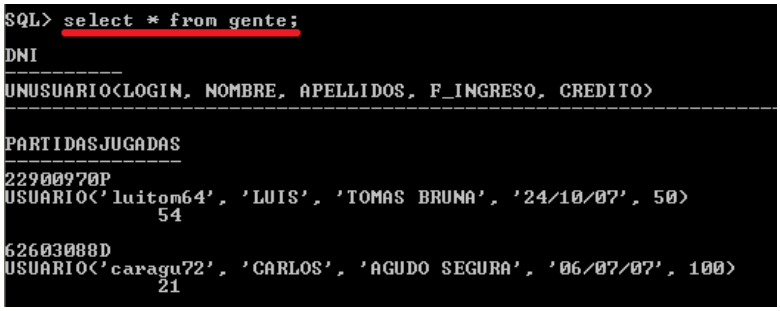
7.1. Tablas con columnas tipo objeto

Se puede usar cualquier tipo de objeto declarado previamente para usarlo como tipo de dato de una columna de una tabla de la bd. Una vez creada la tabla se puede utilizar cualquier sentencia SQL para insertar un objeto, seleccionar sus atributos y actualizar sus datos.

Para crear una tabla en la que alguna de sus columnas sea un tipo de objeto, simplemente se especifica que el tipo de dato de esa columna es del tipo de objeto.

```
CREATE TABLE Gente (
  dni VARCHAR2(10),
  unUsuario Usuario,
  partidasJugadas SMALLINT
);
```

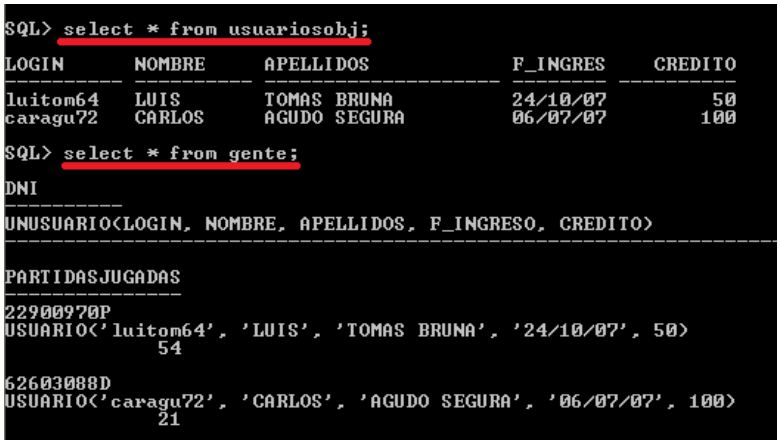
Los datos del campo unUsuario se muestran como integrantes de cada objeto Usuario, a diferencia de la tabla de objetos que has visto en el apartado anterior. Ahora todos los atributos del tipo de objeto Usuario no se muestran como si fueran varias columnas de la tabla, sino que forman parte de una única columna.



7.2. Uso de la sentencia SELECT

Se puede utilizar la sentencia SELECT para obtener datos de filas almacenadas en tablas de objetos o tablas con columnas de tipos de objetos.

```
SELECT * FROM NombreTabla;
```



La tabla que forma parte de la consulta puede ser una tabla de objetos o una tablas que contiene columnas de tipos objetos.

En las sentencias SELECT con objetos se pueden incluir cláusulas y funciones de agrupamiento de las sentencias SELECT: SUM, MAX, WHERE, ORDER, JOIN...

Se usan alias para hacer referencias al nombre de la tabla. por ejemplo, en la siguiente consulta se obtiene el nombre y los apellidos de los usuarios con algo de crédito.

```
SELECT u.nombre, u.apellidos FROM UsuariosObj u WHERE u.credito > 0
```

Si se trata de una tabla con columnas de tipo objeto, el acceso a los atributos del objeto se debe realizar indicando previamente el nombre asignado a la columna que contiene los objetos.

```
SELECT g.unUsuario.nombre, g.unUsuario.apellidos FROM Gente g;
```

7.3. Inserción de objetos

Se usa la misma que para introducir datos de manera habitual. Usando insert de sql.

Para hacer un INSERT de un determinado tipo de objetos o si posee un campo de un determinado tipo de objeto, se suministra a la sentencia INSERT un objeto instanciado de su tipo de objeto correspondiente.

```
DECLARE
  u1 Usuario;
  u2 Usuario;
BEGIN
  u1 := NEW Usuario('luitom64', 'LUIS', 'TOMAS BRUNA', '24/10/2007', 50);
  u2 := NEW Usuario('caragu72', 'CARLOS', 'AGUDO SEGURA', '06/07/2007', 100);
  INSERT INTO UsuariosObj VALUES (u1);
  INSERT INTO UsuariosObj VALUES (u2);
END;
```

También se puede crear el objeto dentro de la misma sentencia insert, sin necesidad de guardarla en una variable.

```
INSERT INTO UsuariosObj VALUES (Usuario('luitom64', 'LUIS', 'TOMAS BRUNA', '24/10/2007', 50));
```

y luego comprobar los resultados haciendo un select habitual

```
SELECT * FROM UsuariosObj;
```

De la misma manera, podemos realizar una inserción de filas en tablas con columnas de tipo objeto.

```
INSERT INTO Gente VALUES ('22900970P', Usuario('luitom64', 'LUIS', 'TOMAS BRUNA', '24/10/2007', 50), 54);
INSERT INTO Gente VALUES ('62603088D', u2, 21);
```

7.4. Modificación de objetos

La diferencia entre los UPDATE habituales es la forma de especificar los nombres de los campos afectados, según sea una tabla de objetos o un atabla con alguna columna de tipo objeto.

Para tabla de objetos se hace referencia a los atributos de los objetos justo después del nombre asignado a la tabla.

```
UPDATE NombreTabla
  SET NombreTabla.atributoModificado = nuevoValor
  WHERE NombreTabla.atributoBusqueda = valorBusqueda;
```

```
UPDATE UsuariosObj
  SET UsuariosObj.credito = 0
  WHERE UsuariosObj.login = 'luitom64';
```

Es habitual abreviar con alias:

```
UPDATE UsuariosObj u
  SET u.credito = 0
  WHERE u.login = 'luitom64';
```

También se puede cambiar un objeto por otro:

```
UPDATE UsuariosObj u SET u = Usuario('juaesc82', 'JUAN', 'ESCUDERO LARRASA', '10/04/2011', 0) WHERE u.login = 'caragu72';
```

Si se trata de una tabla con columna de tipo objeto se hace referencia al nombre de la columna que contiene los objetos.

```
UPDATE NombreTabla
  SET NombreTabla.colObjeto.atributoModificado = nuevoValor
  WHERE NombreTabla.colObjeto.atributoBusqueda = valorBusqueda;
```

```
UPDATE Gente g
  SET g.unUsuario.credito = 0
  WHERE g.unUsuario.login = 'luitom64';
```

O bien, puedes cambiar todo un objeto por otro, manteniendo el resto de los datos de la fila sin modificar

```
UPDATE Gente g
  SET g.unUsuario = Usuario('juaesc82', 'JUAN', 'ESCUDERO LARRASA', '10/04/2011', 0)
  WHERE g.unUsuario.login = 'caragu72';
```

7.5. Borrado de objetos

El uso del DELETE sobre objetos almacenados en tablas es:

```
DELETE FROM NombreTablaObjetos WHERE condición;
```

Recordando siempre que si no se utiliza WHERE, se eliminan todos los objetos de la tabla.

```
DELETE FROM UsuariosObj u WHERE u.credito = 0;
```

De manera similar se usa el borrado de filas en tablas en la que alguna de sus columnas son objetos.

```
DELETE FROM Gente g WHERE g.unUsuario.credito = 0;
```

También se pueden combinar con otras consultas SELECT, de manera que en vez de realizar el borrado sobre una determinada tabla, se haga sobre el resultado de una consulta, o bien que la condición que determina las filas que deben ser eliminadas sea también el resultado de una consulta.

7.6. Consultas con función VALUE

Cuando se tiene la necesidad de hacer referencia a un objeto en lugar de alguno de sus atributos, se utiliza la función VALUE junto al nombre de la tabla de objetos o sus alias, dentro de un SELECT.

```
INSERT INTO Favoritos SELECT VALUE(u) FROM UsuariosObj u WHERE u.credito >= 100;
```

Esa misma función se puede usar para hacer comparaciones de igualdad entre objetos.

```
SELECT u.login FROM UsuariosObj u JOIN Favoritos f ON VALUE(u)=VALUE(f);
```

Si llevamos la comparación a una columna de tipo objetos, en el caso de la referencia a la columna, se obtiene directamente el objeto sin VALUE:

```
SELECT g.dni FROM Gente g JOIN Favoritos f ON g.unUsuario=VALUE(f);
```

Con la cláusula INTO se puede guardar en variables el objeto obtenido en las consultas usando la función VALUE.

Una vez asignado el objeto a la variable se puede hacer uso de ella de la manera habitual en manipulación de objetos:

```
DECLARE
  u1 Usuario;
  u2 Usuario;
BEGIN
  SELECT VALUE(u) INTO u1 FROM UsuariosObj u WHERE u.login = 'luitom64';
  dbms_output.put_line(u1.nombre);
  u2 := u1;
  dbms_output.put_line(u2.nombre);
END;
```

7.7. Referencias a objetos

El paso de objetos a un método es ineficiente si el objeto es grande, lo mejor es pasar un puntero a dicho objeto, lo que permite que el método que lo recibe pueda hacer referencia a dicho objeto sin que sea necesario pasarlo por completo. Ese puntero se llama REF.

Al compartir un objeto mediante referencia, los datos no se duplican y si se hacen cambios en un atributo se producen en un único lugar.

Cada objeto almacenado en una tabla tiene un identificador de objetos y sirve como referencia a dicho objeto.

Las referencias se crean con el modificador REF delante del tipo objeto y se pueden usar con variables, parámetros, campos, atributos e incluso como variables de entrada o salida para sentencias de manipulación de datos en SQL.

```
CREATE OR REPLACE TYPE Partida AS OBJECT (
  codigo INTEGER,
  nombre VARCHAR2(20),
  usuarioCreador REF Usuario
);
/

DECLARE
  u_ref REF Usuario;
  p1 Partida;
BEGIN
  SELECT REF(u) INTO u_ref FROM UsuariosObj u WHERE u.login = 'luitom64';
  p1 := NEW Partida(1, 'partida1', u_ref);
END;
/
```

Las referencias a tipos de objetos solo se pueden usar si han si han sido declarados previamente. En el ejemplo anterior, no se puede declarar el tipo partida antes que el tipo usuario.

Si surgen dos tipos que utilizan referencias mutuas, se soluciona haciendo una declaración de tipo anticipada. Se realiza indicando únicamente el nombre del tipo que se detallará más adelante.

```
CREATE OR REPLACE TYPE tipo2;
/
CREATE OR REPLACE TYPE tipo1 AS OBJECT (
  tipo2_ref REF tipo2
  /*Declaración del resto de atributos del tipo1*/
);
/
CREATE OR REPLACE TYPE tipo2 AS OBJECT (
  tipo1_ref REF tipo1
  /*Declaración del resto de atributos del tipo2*/
);
/
```

7.8. Navegación a través de referencias.

No se puede acceder directamente a los atributos de un objeto referenciado dentro de una tabla, para ellos se usa la función DEREF.

Esta función toma una referencia a un objeto y retorna el valor de ese objeto.

```
u_ref REF Usuario;
u1 Usuario;
```

Si u_ref hacer referencia a un objeto de tipo usuario de la tabla UsuariosObj, para obtener info de sus atributos usamos la función DEREF. Se usa como parte de una consulta SELEECT, utilizando una tabla tras la cláusula FROM.

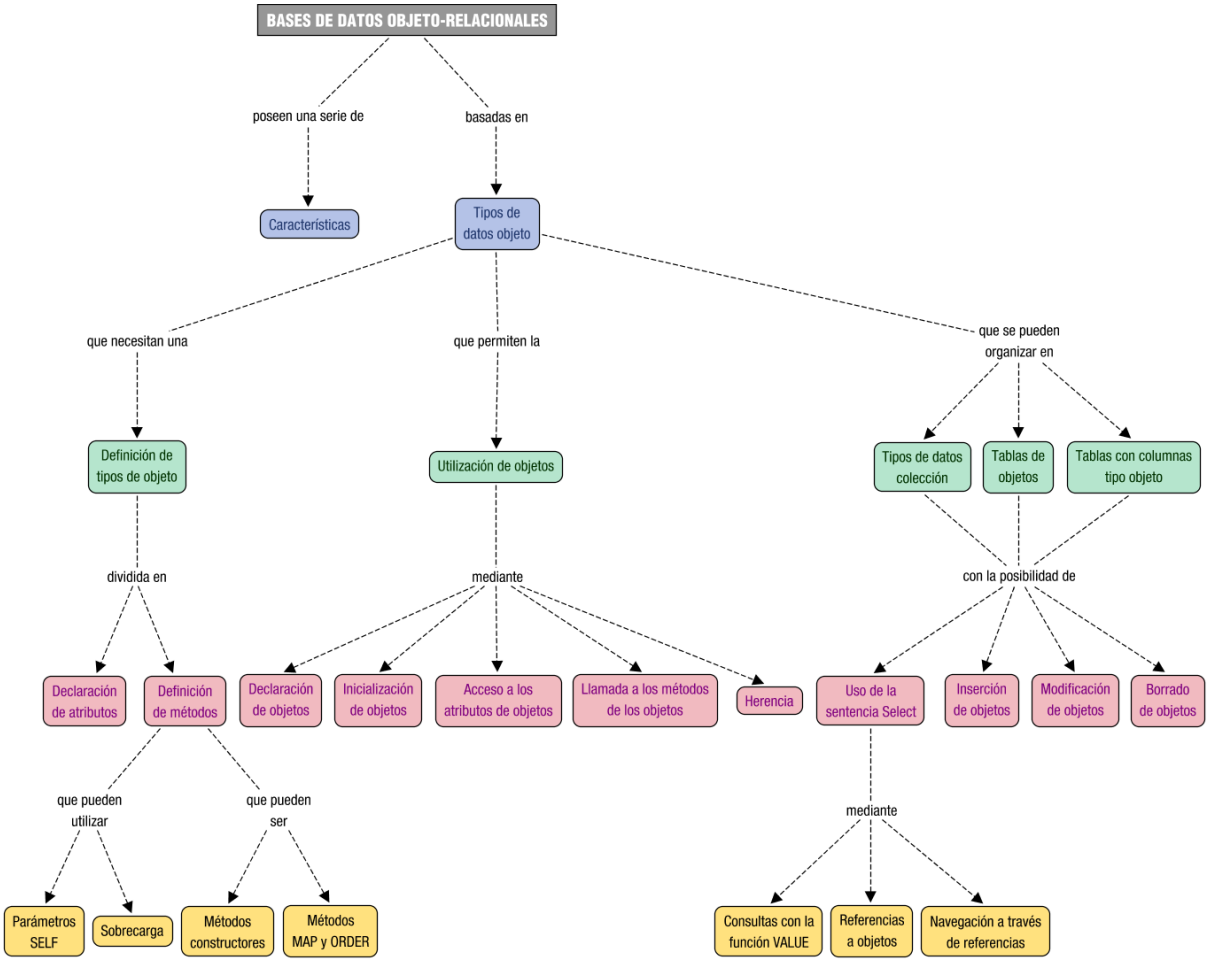
La BD de oracle ofrece una tabla DUAL para estas operaciones, es creada automáticamente por la bd y es accesible por todos los usuarios, solo tiene un campo y un registro, es una tabla comodín.

```
SELECT DEREF(u_ref) INTO u1 FROM Dual;
dbms_output.put_line(u1.nombre);
```

Para obtener el objeto referenciado por una variable REF, se utiliza una consulta sobre cualquier tabla, independientemente de la tabla en la que se encuentre el objeto referenciado, solo existe la condición de que se obtenga una sola fila como resultado. El resultado será un objeto almacenado en la tabla UsuariosObj

```
DECLARE
US USUARIO;
U_REF REF USUARIO;
BEGIN
SELECT REF(U) INTO U_REF FROM UsuObj u WHERE u.login = 'luitom64';
SELECT DEREF(u_ref) INTO us FROM dual;
dbms_output.put_line(us.nombre);
END;
```

Mapa conceptual


BASES DE DATOS OBJETO-RELACIONALES
poseen una serie de
basadas en
Características
Tipos de datos objeto
que necesitan una
que permiten la
que se pueden organizar en
Definición de tipos de objeto
Utilización de objetos
Tipos de datos colección
Tablas de objetos
Tablas con columnas tipo objeto
dividida en
mediante
con la posibilidad de
Declaración de atributos
Definición de métodos
Declaración de objetos
Inicialización de objetos
Acceso a los atributos de objetos
Llamada a los métodos de los objetos
Herencia
Uso de la sentencia Select
Inserción de objetos
Modificación de objetos
Borrado de objetos
que pueden utilizar
que pueden ser
mediante
Parámetros SELF
Sobrecarga
Métodos constructores
Métodos MAP y ORDER
Consultas con la función VALUE
Referencias a objetos
Navegación a través de referencias