


5.3. Desarrollo de módulos para OpenERP.

- Cuando instalamos un sistema, nos encontramos con situaciones en la que no es posible cubrir todas las necesidades del cliente con los componentes existentes.
- El sistema es lo suficientemente flexible como para dotarnos de todas las herramientas necesarias para incrementar de forma fácil y eficiente la funcionalidad, adaptándola a cualquier necesidad que nos encontremos.
- Para desarrollar esta actividad nos podemos centrar en dos procesos de desarrollo: el primero es variar el código del servidor modificando su funcionamiento desde el código fuente. El segundo método nos llevará a crear pequeños "paquetes de software" de expansión del código existente, estos paquetes se denominan **módulos**.

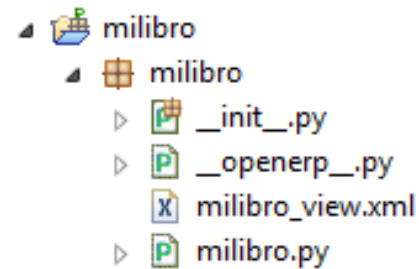
- 
- El uso y funcionamiento de los módulos ya lo hemos tratado en capítulos anteriores, donde hemos visto la gran cantidad de ellos que existen tanto preinstalados en el sistema como descargables desde repositorios.
 - Aquellos módulos que nosotros creemos tendrán un tratamiento similar a cualquier otro, siendo necesario actualizar el sistema desde la **Administración** para detectar el módulo, instalarlo en el sistema de forma general, actualizarlo y borrarlo como los demás y utilizarlo una vez configurado.
 - **Para crear un módulo hay que crear un directorio que contendrá todos los ficheros necesarios y situarlo dentro del directorio addons del servidor** copiándolo o como un fichero comprimido zip. Los ficheros a crear, sus nombres y estructura está determinada por el sistema no pudiendo cambiar la mayoría de ellos.

Pasos para crear un módulo

- La mejor manera de comprender qué es un módulo, por qué elementos está formado, cómo se crea es desarrollando uno.
- 1. Nos desplazamos al directorio **addons** situado dentro de la instalación del servidor (**C:\Archivos de programa\OpenErp \Server** bajo Windows o **/usr/lib/pymodules/python2.7/openerp** bajo Linux) y creamos un nuevo directorio para nuestro módulo: **milibro**.
- 2. Añadimos los ficheros obligatorios: **__init__.py** (dos guiones bajos al inicio y final) y **__openerp__.py**

3. Creamos el fichero de definición del objeto. Por ejemplo: **milibro.py**.

4. Añadimos un fichero XML que contendrá datos para el sistema (las vistas, acciones y menús). Por ejemplo: **milibro_view.xml**.



5. Cuando esté programado el contenido de todos los ficheros tendremos que abrir una sesión con el servidor, desplazarnos hasta **Administración/Módulos** y actualizar la lista de módulos. En el momento en que nuestro módulo sea detectado lo podemos instalar, apareciendo en la pantalla principal.

El fichero `__init__.py`


- Se encarga de cargar en el sistema las definiciones de nuestros objetos, generalmente solo es necesaria la carga del fichero principal, en nuestro caso «milibro.py». Es un fichero **Python**. Su contenido será:


```
# -*- encoding: utf-8 -*-
```

```
import milibro
```

El fichero `__openerp__.py`

- Escrito en formato **Python** y utilizado por el fichero `__init__.py`, define un diccionario anónimo con propiedades predefinidas que utilizará el sistema para determinar los ficheros XML que debe seguir tratando junto con propiedades básicas del módulo y las dependencias con otros módulos del sistema. Los valores a usar son:

- 
- **name.** Describe el nombre del módulo, es obligatorio.
 - **author.** El nombre del autor.
 - **version.** Indica la versión del módulo, teniendo que ser incrementado en uno al menos en el número menor para que se realice una actualización.
 - **description.** Descripción de la función del módulo.
 - **website.** URL con la dirección del programador.
 - **depends.** Es una lista **Python** ([valor, valor,..]) que indica todos los módulos que deben estar instalados obligatoriamente para que este funcione. Esta lista debe incluir obligatoriamente el módulo **base** como mínimo.
 - **init_xml.** Lista de ficheros XML que se utilizarán cuando el servidor se lance con el parámetro **-init=mimodulo**. Las rutas deberán ser relativas a la posición del fichero. En esta lista se incluirán las definiciones de flujos de trabajo, la carga de datos iniciales y de los datos de prueba.

- 
- **update_xml.** Al igual que el anterior representa los ficheros que se utilizan cuando se lance una actualización del módulo mediante el parámetro **-update=mimodulo** o desde el sistema. Estos ficheros XML se utilizarán para definir vistas, informes y asistentes. En el ejemplo veremos que se va a utilizar un fichero XML: `milibro_view.xml`.
 - **category.** Una descripción de la categoría y subcategoría del módulo.
 - **active.** Es un valor booleano True o False determinando si el módulo se debe instalar o no con la creación de la Base de datos. El valor por defecto es False.
 - **installable.** Determina si el módulo se puede o no instalar. Valores posibles: True o False.
 - **demo_xml.** Lista de ficheros XML con los datos de prueba.
 - **license.** Código de la licencia del módulo, para las pruebas deberá ser obligatoriamente GPL-2 que es el valor por defecto.

➤ Para nuestro ejemplo, el contenido será:

```
# -*- encoding: utf-8 -*-
```

```
{
```

```
  "name": "Mi libro",
```

```
  "author": "Pepe",
```

```
  "description": "Ejemplo",
```

```
  "version": "1.4",
```

```
  "depends": [],
```

```
  "data" : ["milibro_view.xml"],
```

```
  "update_xml": [],
```


```
  "category": "Try/Others",
```

```
  "license" : "Other OSI approved licence",
```

```
  "active": False,
```


```
  "installable": True
```

```
}
```


El fichero milibro.py

- Este fichero determinará todos los nuevos objetos y sus propiedades en formato **Python**.
- Podemos usar algún tipo de herramienta tipo **DIA** para crear los diagramas de clases y generar de forma automática el código.
- Para **OpenERP** todo lo almacenado en la base de datos son objetos y utiliza un mecanismo de **ORM** (Mapeo de Objetos Relacionales) para transportar las tablas relacionales a la programación junto con el contenido. De esta manera cuando estamos creando un objeto (una clase) en este fichero se crea automáticamente la tabla asociada con los atributos que definamos.



➤ Dentro de un objeto, una clase (**class**), tendremos que usar obligatoriamente un conjunto de propiedades para determinar los componentes del mismo. Las propiedades válidas son:

- **_name.** El nombre del objeto, la tabla que se va a crear.
- **_columns.** Es un campo obligatorio con la definición de los campos de la tabla.
- **_constrain.** Las restricciones que sufrirán los campos.
- **_defaults.** Valores por defecto a usar en los campos.
- **_inherit.** Objeto del que hereda.
- **_order.** Campos a usar en las sentencias de selección y lectura para ordenar.
- **_rec-name.** Campo a usar en las búsquedas.

- 
- En el ejemplo definimos un objeto milibro (**class**) que heredará de **osv.osv** con cuatro campos: titulo, paginas, autor y editorial.
 - Esta definición creará una tabla: milibro (propiedad **_name**) en la base de datos actual con los tres campos de texto (**fields.char**) de tamaños diferentes (**size**) y uno entero (**fields.integer**).
 - De los campos sólo será imprescindible el título (**required=True**). Para terminar la definición del objeto se inicializa con milibro().
 - El contenido del fichero es:



```
# -*- encoding: utf-8 -*-
```

```
from osv import osv,fields
```

```
class milibro(osv.osv):
```

```
    _name = 'milibro'
```

```
    _columns = {
```

```
        'titulo': fields.char('Title', size=100,required=True),
```


```
        'paginas': fields.integer('Pages', required=False),
```

```
        'autor': fields.char('Author', size=100, required=True),
```

```
        'editorial': fields.char('Editorial', size=100,required=False),
```

```
    }
```

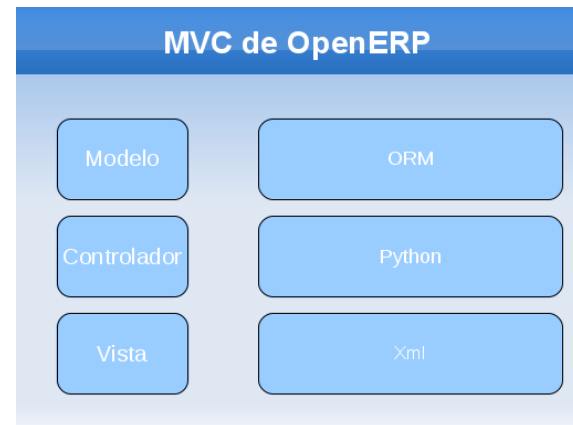
```
milibro()
```

- 
- La estructura que presenta **OpenERP** es jerárquica creando un objeto por cada nivel comenzando la jerarquía generalmente con el nombre del módulo, no un objeto por cada contenido. Es decir, si la tabla milibro tiene cinco líneas, aparecerá un objeto milibro con una propiedad que contendrá los identificadores de cada uno de ellos, no existirán en ningún momento cinco objetos diferentes.
 - Al automatizar el acceso y creación de las tablas mediante **ORM** no es necesario el acceso directo a la base de datos en ningún momento, aunque podemos utilizarlo asumiendo el riesgo que ello presenta.


El fichero milibro_view.xml


- Con este fichero determinaremos los componentes **OpenERP** que se tienen que crear para que nuestro módulo se integre con el sistema.
- Un módulo estará formado por vistas, gráficos, informes, asistentes, flujos de trabajo, menús y acciones. En este fichero definimos las vistas, menús y acciones.


- **OpenERP** utiliza el paradigma **MVC** (Modelo Vista Controlador) para separar los datos de su presentación.



- Con el fichero **Python** hemos definido la estructura de los datos, en este definiremos la estructura de visualización.

- 
- El ejemplo incluirá dos tipos de vistas: una en tipo árbol y otra en tipo formulario. La primera se utiliza para listar los contenidos por filas, la segunda para modificación e inserción de datos.
 - Cuando creemos un módulo serán necesarias al menos la definición de los menús, las acciones de apertura de una vista y, las vistas de árbol y formulario, sin ellas es imposible que nuestro módulo se instale correctamente.
 - **Los menús.** Se define un menú principal y uno secundario dentro de él con las etiquetas **<menuitem>**. Los menús tendrán un identificador (**id**) necesario para las referencias, el nombre a mostrar (**name**) y nombre del menú padre en el que se encuentra (**parent**) excepto para el raíz.

- 
- **La acción.** Con la etiqueta `<record model="ir.actions.act_window" id="action_milibro_form">` definimos una acción para abrir una vista. Esta acción se une a los menús a través de la propiedad **action** del menú que se rellena con el valor de la propiedad **id** definida en la acción. Se define el objeto que necesitamos abrir a través de los atributos **name** y **res_model**.
 - **Las etiquetas <record>** definen las dos **vistas** (árbol y formulario) con los campos a mostrar. Una vista por defecto en formato lista (**<tree>**) y otra para introducción en formato formulario (**<form>**) con los cuatro campos.
 - El contenido del fichero es:



```
<?xml version="1.0" encoding="utf-8"?>
```

```
<openerp>
```

```
<data>
```

```
  <menuitem name="Libros" id="libros_menu"/>
```

```
  <record model="ir.ui.view" id="milibro_tree_view">
```

```
    <field name="name">milibro.tree</field>
```

```
    <field name="model">milibro</field>
```

```
    <field name="type">tree</field>
```

```
    <field name="arch" type="xml">
```

```
      <tree string="Libros">
```

```
        <field name="titulo"/>
```

```
        <field name="paginas"/>
```


```
        <field name="autor"/>
```

```
        <field name="editorial"/>
```


```
      </tree>
```

```
    </field>
```

```
  </record>
```



```
<record model="ir.ui.view" id="milibro_form_view">
  <field name="name">milibro.form</field>
  <field name="model">milibro</field>
  <field name="type">form</field>
  <field name="arch" type="xml">
    <form string="Libros">
      <field name="titulo"/>
      <field name="paginas"/>
      <field name="autor"/>
      <field name="editorial"/>
    </form>
  </field>
</record>
```



```
<record model="ir.actions.act_window" id="action_milibro_form">
    <field name="name">Libros</field>
    <field name="res_model">milibro</field>
    <field name="view_type">form</field>
    <field name="view_mode">tree,form</field>
</record>
<menuitem name="Libros" id="mislibros" parent="milibro_menu"/>
<menuitem name="Mis Libros" id="milibro_form" parent="mislibros" action=
"action_milibro_form"/>
</data>
</openerp>
```

- Una vez creados todos los archivos necesarios, instalar el módulo.


Módulos locales



Mi libro
milibro

Instalar

Mensajería Libros Ventas Contabilidad Más ▾ ✉ Administrator ▾

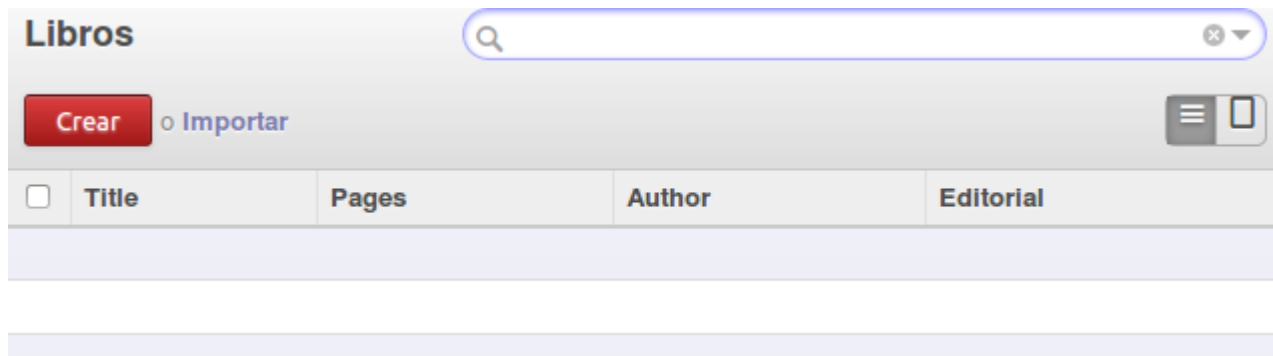


Crear o Importar

<input type="checkbox"/>	Title	Pages	Author	Editorial

Libros
Mis Libros

- Al hacer clic en «Mis Libros», veremos la vista **Tree** (árbol) que creamos:



The screenshot shows a web interface for managing books. At the top, there's a header with the title 'Libros' and a search bar. Below the header, there are two buttons: 'Crear' (red) and 'Importar' (blue). To the right of these buttons is a mobile view toggle icon. Below the buttons is a table with the following columns: 'Title', 'Pages', 'Author', and 'Editorial'. The table is currently empty.

- Y al hacer clic en el botón «Crear», la vista **Form** (formulario) para agregar libros.



The screenshot shows the 'Libros / Nuevo' (Books / New) form. At the top, there's a header with the title 'Libros / Nuevo'. Below the header, there are two buttons: 'Guardar' (red) and 'Descartar' (blue). To the right of these buttons is a mobile view toggle icon. Below the buttons, there are four input fields: 'Title', 'Pages' (with a value of '0'), 'Author', and 'Editorial'.

Actividad 5.40

Crea un **proyecto** Python en Eclipse de nombre **milibro**.

- a) Añade al fichero `__init__.py` el código del ejemplo.
- b) Añade al proyecto el módulo `__openerp__.py` con el código del ejemplo.
- c) Añade al proyecto el módulo `milibro.py` con el código del ejemplo.
- d) Añade al proyecto el fichero `milibro_view.xml` con el código del ejemplo.
- e) Crea en addons una carpeta de nombre `milibro` y copia dentro `__init__.py`, `__openerp__.py`, `milibro.py` y `milibro_view.xml`. (Recuerda cambiar los permisos a la carpeta)
- f) Actualiza la lista de módulos en la base de datos e instala el módulo.

Añadiendo herencia al ejemplo

- Vamos a extender el primer módulo para añadir más datos al libro y crearemos un nuevo objeto para las categorías. Cada libro tendrá una categoría asociada a través de una relación de muchos a uno.
- El código para la nueva **clase de categorías** sería:

```
class milibro2_categorias(osv.osv):  
    _name = 'milibro2.categorias'  
    _columns = {'name': fields.char('Descripcion', size=150, required=True)}
```

➤ Y para la clase que deriva de milibro (milibro2):

```
class milibro2(osv.osv):
```

```
    _name = 'milibro2'
```

```
    _inherit = 'milibro'
```

```
    _columns = {
```

```
        'isbn': fields.char('ISBN', size=15),
```

```
        'precio': fields.float('Price', digits=(4,2)),
```

```
        'resumen': fields.text('Description'),
```


```
        'fecha': fields.date('Date'),
```

```
        'revisado': fields.boolean('Revisado'),
```

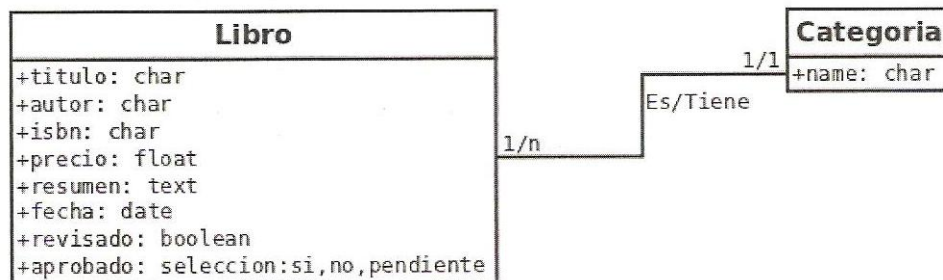
```
        'aprobado': fields.selection((('S','Si'),('N','No'),('P','Pendiente')),'Aprobed'),
```


```
        'categoria': fields.many2one('milibro2.categorias','Category',onDelete='cascade')
```

```
    }
```


- 
- Estas dos clases se incluirán en el **fichero milibro2.py** junto con la sentencia para invocar el módulo **osv** y las llamadas para crear un objeto de cada una de las clases.
 - Para **OpenERP** todo son objetos. Para cada tipo de recurso aparece un objeto diferente con los datos necesarios para acceder a todos los elementos del tipo. En el ejemplo del módulo anterior creamos una clase milibro que contendrá libros, por lo que en el sistema existirá un objeto milibro con el que acceder a todos los libros y sus datos.
 - En esencia a nivel de base de datos se crea un objeto por cada tabla y una propiedad que contendrá todos los identificadores (clave primaria) de todos los registros de la tabla.

- Para transformar las tablas en objetos se utiliza un mecanismo de **ORM** (Mapeado de objetos relacionales) haciendo de puente entre la estructura física de la base de datos y los objetos del sistema.
- Un objeto del sistema se modela como una definición estática de **Python** para el comportamiento y sus datos, junto a una descripción **SQL** del almacenamiento en la base de datos.
- El esquema entidad-relación para el ejemplo sería:



- 
- Al crear un nuevo objeto del sistema, deberemos implementar una clase y heredar del objeto **osv.osv**. De esta manera añadimos los elementos mínimos para que se reconozca el nuevo objeto. Al definir el objeto se utilizarán un conjunto de propiedades predefinidas con nombres ya establecidos que comienzan siempre por un subrayado:

_auto. Por defecto vale **True** y especifica si la tabla se tiene que crear de forma automática.

_columns. Describirá los campos de la tabla a crear.

_constraints. Determina las restricciones de los campos.

_sql_constraints. Especifica las restricciones de los campos a nivel SQL.

_defaults. Valores por defecto para los campos.

_inherit. Objeto del que heredará el que estamos definiendo.



_inherits. Diccionario con los objetos de los que heredará el que estamos definiendo.

No se puede usar conjuntamente con el anterior.

_log-access. Si la establecemos a **True** se añadirán a la tabla cuatro campos para determinar quién, cuando se creó el registro y quién y cuándo se modificó por última vez.


_name. Valor obligatorio que indicará el nombre del objeto, el nombre de la tabla a crear.

_order. Nombres de los campos a usar en las sentencias de lectura y búsqueda.

_rec-name. Nombre del campo a utilizar para las búsquedas, por defecto **name**.


_sequence. Nombre de la secuencia **SQL** que utilizará la tabla para la creación de los identificadores.

_table. Nombre de la tabla a crear. Por defecto utiliza el valor de la propiedad **_name** y cambia los puntos por subrayados bajos.

- 
- En este módulo creamos dos objetos, siempre empezando por aquel que no tiene relaciones: las categorías, ya que para hacer uso de un objeto este tiene que estar definido antes.
 - El primer objeto se llamará `milibro2.categorías` y tendrá únicamente un campo **name**.
 - El segundo se nombrará como `milibro2` y heredará del objeto `milibro` (**`_inherit`**) definido anteriormente.
 - En la herencia recoge todos los campos anteriormente establecidos y añade `isbn`, `precio`, `resumen`, `fecha`, `revisado` y `aprobado`. Este objeto tiene una relación de muchos a uno con las categorías a través del campo final categorías (**`manyzone`**).

Tipos de herencia en OpenERP

- Vamos a tratar la herencia desde el punto de vista de **OpenERP** y de los objetos del sistema.
- Podemos utilizar tres tipos de herencia que se va a determinar en función de los valores en las propiedades **_name**, **_inherit** e **_inherits**.
- **Herencia por extensión.** Este tipo de herencia añade atributos a un objeto existente sin crear uno nuevo. Se produce cuando las propiedades **_name** e **_inherit** son iguales y se fijan a un objeto ya existente en el sistema. Para que los nuevos campos se vean en las vistas hay que extender las vistas del objeto del que heredamos y añadirlos de forma similar.


- 
- **Herencia por prototipo.** En este caso se crea un nuevo objeto que hereda todos los componentes del padre, pero totalmente independiente. Esta herencia se crea cuando establecemos en la propiedad **_name** un valor no registrado y en la propiedad **_inherit** un valor de un objeto registrado. De esta manera se añaden al nuevo objeto todos los campos definidos en el heredado.
 - **Herencia por delegación.** Esta herencia se usa cuando necesitamos crear un nuevo objeto a partir de varios ya existentes, es decir, cuando vamos a utilizar una herencia múltiple. Para que se pueda dar tendremos que establecer un nombre de objeto no registrado en la propiedad **_name** y crear un diccionario con todos los objetos de los que vayamos a heredar con la variable **_inherits**. El nuevo objeto será creado con todos los campos de todos los objetos definidos más los que añadamos nosotros.





Los campos de los objetos

- Se definen a través de **_columns**, que es un **diccionario** en el que definiremos los valores necesarios.
- La clave del diccionario será el nombre del campo a crear y el valor la definición. En el ejemplo anterior se crea un campo llamado **precio** en la tabla del tipo **float** con una precisión de cuatro dígitos más dos decimales y una descripción **Price**.
- La definición del campo comienza con la estructura **fields.tipo('descripción',[opciones])** en donde tipo es uno de los tipos básicos siguientes:

- **boolean.** Campo verdadero o falso sin más opciones.
- **integer.** Valor entero.
- **float.** Valor decimal en el que podemos especificar la precisión con la opción **digits=(A, B)**. **A** indicará el número de cifras enteras y **B** el número de dígitos decimales.
- **char.** Cadena de texto de tamaño fijo especificada en el valor **size=valor**.
- **text.** Cadena de texto sin longitud.
- **date.** Fecha.
- **datetime.** Fecha y hora.
- **binary.** Campo binario.
- **selection.** Este campo crea una lista con los posibles valores que contendrá. En el ejemplo el campo contendrá los valores S, N o P pero se mostrarán al usuario Si, No o Pendiente. Hay que darse cuenta de que es una **tupla** de **tuplas**.

- 
- Tras la elección del tipo, lo general es que se establezca una descripción del campo, excepto para **selection** que la descripción va después de la definición de los valores. A continuación, algunos tipos añaden opciones propias (**char** usa **size**, **float** usa **digits**) pudiendo añadir al final la combinación que queramos de las siguientes opciones separadas por comas:
- **required=True**. Si el campo es obligatorio.
 - **select=True**. Si necesitamos crear un índice sobre el campo.
 - **translate=True**. Para decir al sistema que intente mostrar traducido el campo.
 - **ondelete="set null|cascade|set default|restrict|no action"** solo para las restricciones indicando cómo se tiene que comportar la restricción de integridad cuando se borre el padre.
 - **readonly=True**. Para establecer que el campo es de solo lectura.

- 
- Aunque los tipos básicos construyen lo imprescindible para almacenar la información, no representan fielmente un esquema relacional, siendo necesario algo más. Para poder representar correctamente el mundo necesitamos relaciones entre los objetos y en algunos casos campos calculados. El uso de los campos calculados se queda fuera de este estudio, pero no las relaciones entre los objetos (tablas).
 - **OpenERP define cuatro tipos de relaciones diferentes**, pero de ellas una está obsoleta. Se permite crear relaciones uno a uno, uno a muchos, muchos a uno y muchos a muchos. La primera (uno a uno) está obsoleta, recomendando que se sustituyan por relaciones muchos a uno. La última es una relación muy poco frecuente ya que en los esquemas entidad - relación se sustituyen por relaciones uno a muchos. Por todo ello nos quedan dos relaciones: **muchos a uno** y **uno a muchos**.

- 
- Una relación muchos a uno se crea en la parte n de la relación haciendo referencia a la parte uno. En el ejemplo un libro tiene una categoría pero una categoría tiene muchos libros, la relación la definiremos dentro de libro, no dentro de categoría. Esta situación implica que tengamos que tener definido primero los objetos de la parte uno para que no falle el sistema. Esta es la razón por la que en el ejemplo se define primero milibro2.categorías y a continuación milibro2 con la relación **many2one**.
 - La definición de la relación se crea utilizando como tipo **many2one**, en el primer parámetro estableceremos el objeto con el que queremos crear la relación, en el segundo la descripción del campo y a continuación alguno de los valores optativos si lo creemos necesario. En el ejemplo anterior se crea un campo categoría relacionado mediante **muchos a uno** con el objeto milibro2.categorías y se utilizará el criterio de borrar en **cascada** en caso que en la tabla principal se borre un registro

Actividad 5.41

Crea un proyecto Python en Eclipse de nombre milibro2.

- a) Añade el fichero `__init__.py` y escribe el código que necesita.
- b) Añade al proyecto el módulo `__openerp__.py` con el código que consideres oportuno.
- c) Añade al proyecto el módulo `milibro2.py` con el código del ejemplo y escribe las líneas que le faltan.

- A la hora de escribir el fichero **XML** para las vistas, pensemos en cómo queremos que se muestren los objetos.
- Al hacer clic sobre categorías, queremos que se muestre una vista en árbol con las categorías que vayamos agregando.



- La vista formulario para agregar categorías nuevas tendría la forma:

milibro2_cat / Nuevo

Guardar o Descartar

Descripcion

- Al hacer clic sobre Mis Libros 2, se abrirá la vista de tipo árbol con los libros agregados.


milibro2		
Crear o Importar		
<input type="checkbox"/>	Title	Author
<input type="checkbox"/>	Prueba2	Teresa


- Y la vista formulario donde agregaremos los nuevos libros:


The screenshot shows a web-based form titled 'milibro2 / Nuevo'. At the top, there are two buttons: 'Guardar' (red) and 'Descartar' (blue). Below the buttons, the form is organized into two main sections. The left section contains fields for 'Title', 'Author', 'Price' (with a value of '0,00'), 'Revisado' (a checkbox), and 'Category' (a dropdown menu). The right section contains fields for 'Pages' (with a value of '0'), 'ISBN', 'Date', 'Aprobado' (a checkbox), and 'Description' (a large text area). The form is styled with a clean, modern look, using light blue and white colors.

Vistas y eventos

- El diseño de **OpenERP** utiliza el paradigma de **Modelo-Vista-Controlador** para gestionar los datos.
- Al separar los datos de la presentación conseguimos independencia, pudiéndose implementar diferentes entornos de acceso al servidor.


- 
- Las interfaces que se presentan a los clientes son dinámicas describiéndose a través de ficheros **XML** que se pueden editar en cualquier momento, incluso durante la ejecución del servidor. Si un fichero cambia, simplemente recargando la pestaña correspondiente se reflejarán las diferencias.
 - Cuando diseñamos los modelos de presentación tenemos que determinar cómo se va a realizar la visualización en pantalla. Para este proceso disponemos de dos tipos de vistas principales y algunas de apoyo.
 - **Vista formulario.** Se utiliza para la edición de datos, los campos se distribuyen a lo largo de la visualización utilizando un conjunto de reglas:


- 
- ✓ A cada campo se le añade su etiqueta con el nombre. Los campos se sitúan de izquierda a derecha y de arriba abajo según el orden de definición en el fichero XML.
 - ✓ Cada pantalla está definida por cuatro columnas y un número indeterminado de filas. Cada campo utiliza por defecto dos columnas, una para la etiqueta y otra para el campo. Podemos fijar mediante atributos que un campo se extienda a lo largo de varias columnas ampliando de esta manera su tamaño.
 - ✓ Se puede utilizar una etiqueta especial para dividir una columna en tantas columnas como necesitemos.
 - **Vista en árbol.** Las vistas en árbol se usan en los listados de datos, son vistas muy simples y con un número menor de opciones.

- 
- **Vista de búsqueda.** Es un complemento a la vista en árbol que añade un panel de búsqueda y filtrado avanzado en la parte superior de la misma.
 - **Vista de gráfico.** Es un nuevo modo de vista para los formularios mostrando un gráfico formado a partir de los datos.

Definición de las vistas

- Las vistas se definen en el fichero XML que establezcamos en el fichero **__openerp__.py**.
- Este fichero tiene una estructura especial y necesita de al menos tres elementos para definirla correctamente.

- 
- Los elementos que hay que crear son: Una **acción** para la vista, un **menú** para ejecutar la acción y el conjunto de **vistas** asociadas a la acción.
 - Para la acción utilizaremos una etiqueta `<record model="ir.actions.act_windows">` para definirla. A continuación, definiremos el menú asociado a través de la etiqueta `<menuitem>` y para terminar crearemos tantas vistas como hayamos definido en la acción a través de diferentes etiquetas `<record model="ir.ui.view">`.
 - Si la acción no determina las vistas a usar se deberán implementar una tipo **formulario** y una tipo **árbol** por defecto.
 - Dentro de la etiqueta de definición de vista (**tree**, **form**, **search** o **graph**) tendremos que crear la estructura de visualización adecuada al usuario.


- 
- Para implementarla tenemos dos tipos de elementos, elementos **grupales** y elementos de **datos**. Los elementos **grupales** no muestran datos, simplemente los organizan.
 - **<separator string="Nombre a mostrar" colspan="Número">**. Crea una división entre campos a través de una línea y un texto. El texto a mostrar aparecerá en la propiedad **string** y el número de columnas que se utilizarán para este elemento se definen a través de **colspan**.
 - **<notebook colspan="Número">**. Este elemento crea un control de pestañas, simplemente se utiliza para agrupar pestañas no para definir datos. Obligatoriamente tendrá al menos una pestaña dentro definida con **<page**.

- **<page string="">**. El contenido de un notebook se organiza en pestañas a través de esta etiqueta. Crearemos tantas etiquetas page como pestañas necesitemos y dentro se definirá la visualización usando otras etiquetas de agrupación excepto notebook y page.
- **<group colspan="Número" rowspan="Número" expand="yes" col="Número" string="Cadena a mostrar">**. Este elemento agrupará varios controles de datos en columnas de la siguiente manera. Primero el control se extenderá a través del número de columnas especificadas en **colspan** y de filas con **rowspan**. El espacio que se le asigne se dividirá en columnas, tantas como las definidas en la propiedad **col**. El espacio no utilizado se distribuirá entre el resto siempre que la propiedad **expand** se encuentre a yes.



➤ Los elementos de datos a usar son los siguientes:


- **<newline/>**. No es un elemento de datos, crea un salto de línea en la visualización obligando al siguiente control a estar en la siguiente fila.
- **<label string="Texto"/>**. Añade una etiqueta con el texto.
- **<field**. Define un campo que obligatoriamente tendrá que estar definido en el modelo de datos. Es decir en una clase dentro de un fichero. Las propiedades que podemos usar son:
 - **name= " "** es el nombre del campo definido en el fichero **Python** en una clase del objeto utilizado en la definición de la vista.
 - **select="1"** si queremos que el campo actual sea un campo utilizado en búsquedas, con lo que se creará un índice.

- 
- `colspan="Número"`. Cantidad de columnas a utilizar por parte del campo, por defecto dos, incluyendo la etiqueta.
 - `readonly="1"`. Si queremos que el campo sea de solo lectura.
 - `required="1"`. El campo es obligatorio incluso si no se ha definido así en el modelo de datos (El objeto o clase).
 - `nolabel="1"`. No se imprimirá la etiqueta asociada al nombre del campo.
 - `invisible="True"`. Para ocultar el campo y su etiqueta.
 - `string="Texto"`. Valor a poner en la etiqueta, si no se utilizará el nombre del campo.
 - `onchange="función"`. Nombre y parámetros de la función a llamar cuando cambie el valor del campo. Esta función estará definida en el modelo de datos.
 - `widget="tipo"`. Cambia el control por defecto que se visualiza por otro. Se puede utilizar: `one2many_list`, `many2one_list`, `many2many`, `url`, `email`, `image`, `float_time` y `reference`.



Menús y acciones

- Un menú es un elemento visual que se encarga de enlazar una acción con una vista a respuesta del usuario.
- El menú se presentará dentro del árbol en una posición concreta especificando el elemento padre y de forma recursiva usando esta propiedad (**parent**) para crear todo el árbol.
- El único menú que carecerá de propiedad **parent** será el menú raíz.
- La definición se realiza a través de la etiqueta **<menuitem** que puede tener las siguientes propiedades:

- 
- **name="valor"**. El nombre del menú y a mostrar en el árbol.
 - **action="id_action"**. El identificador de la acción que se llevará a cabo.
 - **icon, web_icon, web_icon_hover**. Definirán un fichero para utilizar en los interfaces cliente. La ruta debe ser relativa a la raíz de la instalación del módulo.
 - **groups**. Es un campo que puede contener un conjunto de grupos de seguridad del sistema separados por comas indicando aquellos que tienen acceso.
 - **sequence="Nº"**. es utilizado para organizar los elementos dentro del menú. Si no existe este valor se crean según se definen de arriba abajo.


Actividad 5.42


- a) Añade al proyecto el fichero milibro2_view.xml con las vistas que se necesiten.
- b) Crea en addons una carpeta de nombre milibro2 y copia dentro __init__.py, __openerp__.py, milibro2.py y milibro2_view.xml.
- c) Actualiza la lista de módulos en la base de datos e instala el módulo.
- d) ¿Cómo podría modificar el menú para que se incluyera dentro del menú “Libros” como se muestra a continuación?




Herencia en las vistas

- Cuando definimos vistas de objetos heredados (la propiedad **_inherit=_name** en la definición del objeto) se nos plantean dos posibilidades.
- La primera es crear una vista nueva completa desde cero haciendo referencia al nuevo modelo (propiedad **res_model** de la acción) o utilizar la vista que ha creado el padre y redefinirla.
- La primera opción la hemos visto a través del ejemplo de milibro2, abordaremos ahora la redefinición de vistas.
- Al redefinir una vista tendremos en cuenta que el objeto que acabamos de crear ha extendido la tabla del padre para añadir campos, no se ha creado una nueva.

- 
- Recordamos que el nombre de la tabla se determina en la propiedad **_name** del objeto y se usa en la propiedad **res_model** de la acción.
 - Ahora que hemos determinado los campos a utilizar hay que decirle a la vista que use la definida por el padre. Esto se hace al añadir en la definición de la vista el campo `<field name="inherit_id" ref="id_vista_padre" />`. El sistema interpretará la vista que hagamos referencia y la incorporará a nuestra definición de forma automática.
 - Con la incorporación de la vista padre logramos que todos los campos que existen se añadan a la nueva, a continuación hay que realizar las modificaciones necesarias para agregar los nuevos campos. Añadir nuevos campos a la vista heredada tiene una sintaxis especial, en la que cambiaremos la típica etiqueta **<form** por una etiqueta **<data** y especificaremos los campos de la siguiente manera:

- 
- Si hay que eliminar un campo añadiremos una etiqueta **<field name="nombre campo a eliminar de la vista padre" position="replace" />**.
 - Para remplazar una campo por otro procederemos de la misma manera que el punto anterior, pero en vez de terminar la definición en una línea, dentro añadiremos otra etiqueta **<field name="nuevo campo">**.
 - Para añadir campos seleccionaremos un campo base que exista a través de una etiqueta **<field** y dentro añadiremos tantas etiquetas **<field** como campos nuevos queramos añadir. Lo único que tendremos en cuenta es que la primera etiqueta debe especificar cómo se van a añadir los campos si antes (before) o después (after) en el atributo **position**.

- 
- Habrá casos en los que determinar un campo será difícil porque aparece varias veces, por la estructura, etc. En estos casos podremos usar un elemento **<xpath** del XML para hacer una búsqueda sobre el árbol creado y dentro realizar las modificaciones. En el siguiente ejemplo se insertará el nuevo campo **age** después (**position="after"**) del campo **email** que estará dentro de un **form**, que estará dentro de un campo llamado **address** en cualquier posición del árbol. Ejemplo:

```
<xpath  
  expr="//field[@name='address']/form/field[@name='email']"  
  position="after">  
  <field name="age"/>  
</xpath>
```

Actividad 5.43

Copia al servidor addons el contenido de la carpeta milibro3.

- a) Actualiza la lista de módulos e instálalo.
- b) Accede a «LIBROS» y desde allí al menú «Mis Libros 3». Intenta agregar un libro.

Nota: tiene que producirse un error.

En este ejemplo eliminamos el campo título, con lo que el formulario queda inservible ya que es requerido, además tiene un efecto colateral, la vista del padre (milibro) también se modifica pasando a no poderse utilizar para añadir nuevos datos.

Actividad 5.44

Crea un módulo para gestionar revistas que se integrará con milibro2.

- El menú aparecerá dentro de los menús de milibro2.
- El módulo utilizará un campo categoría que haga referencia al objeto `milibro2.categorias`.
 - Crea los ficheros `__init__.py`, `__openerp__.py`, `mirevista.py` y `mirevista_view.xml`.
- Copia los ficheros al directorio addons, actualiza la lista de módulos e instálalo.

Internacionalización

- Para terminar la programación de nuestro módulo, deberíamos proporcionar las traducciones necesarias. Cuando creamos un módulo las normas dictan que el nombre y etiquetas de los campos se utilicen en inglés creando los ficheros de idioma necesarios. La traducción se hará de la siguiente manera.
- 1. Una vez terminado el módulo crearemos los ficheros **mimodulo.pot** y **es.po** exportando desde un sistema **OpenERP** con el módulo instalado. El primero sin seleccionar ningún idioma, el segundo marcando el idioma necesario, en este caso español. En ambos casos se selecciona «**PO**» como formato del archivo.
- 2. Dentro del directorio **addons**, abriremos la carpeta de nuestro módulo, crearemos una carpeta de nombre **i18n** y copiaremos dentro el fichero **mimodulo.pot** y el **es.po**.

3. Abriremos el fichero **es.po** y añadiremos las traducciones. Se trata de escribir la traducción de los campos **msgid** en los campos **msgstr**.

```
# Translation of OpenERP Server.
# This file contains the translation of the following modules:
#   * milibro
#
msgid ""
msgstr ""
"Project-Id-Version: OpenERP Server 6.0.1\n"
"Report-Msgid-Bugs-To: support@openerp.com\n"
"POT-Creation-Date: 2014-02-15 20:03+0000\n"
"PO-Revision-Date: 2014-02-15 20:03+0000\n"
"Last-Translator: <>\n"
"Language-Team: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: \n"
"Plural-Forms: \n"

#. module: milibro
#: field:milibro,titulo:0
msgid "Title"
msgstr "Titulo"

#. module: milibro
#: model:ir.model,name:milibro.model_milibro
msgid "milibro"
msgstr "milibro"

#. module: milibro
#: model:ir.ui.menu,name:milibro.milibro_form
msgid "Mis Libros"
msgstr "Mis Libros"
```

Usuarios

Grupos

Usuarios

Traducciones

Idiomas

Cargar una traducción

Importar / Exportar

Importar traducción

Exportar traducción

Términos de la aplicación

Exportar traducción

Preferencias de exportación

Idioma

Nuevo idioma (Plantilla de traducción vacía)

Formato del archivo

Archivo PO

Módulos a exportar

Mi libro

Exportar

o Cancelar

Importar traducción

Nombre del idioma

Español

Código

es_ES

Archivo

es.po (1.

Seleccionar

Guardar como

Limpiar

Sobrescribir términos existentes

☒

Importar

o Cancelar

Importar traducción

Exportar traducción

Términos de la aplicación

Términos traducidos

Sincronizar términos

Técnico

Email

Acciones

Interfaz de usuario

Sincronizar términos

Sincronizar traducción

Idioma

Actualizar

o Cancelar



Distribución

- Con el módulo ya terminado, creamos un fichero .zip que será el que distribuiremos



Actividad 5.45

Traduce los módulos milibro y milibro2.

Actividad 5.46

Distribuye los módulos creados como ficheros zip e instálalos en otro servidor.