GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

UD06. DESARROLLO DE MÓDULOS. DATOS

Sistemas de Gestión Empresarial
2 Curso // CFGS DAM // Informática y Comunicaciones
Alfredo Oltra

Cicles Formatius

ÍNDEX

Versión: 240122.2333

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

☑ **Atención**. Importante prestar atención a esta información.

Interesante. Ofrece información sobre algun detalle a tener en cuenta.

1 INTRODUCCIÓN

Como ya hemos visto, parte de la codificación de un módulo consiste en la inserción de datos en ciertas tablas de la base de datos, como por ejemplo las vistas o las acciones. Pero esta inserción de datos puede ir más allá. Por ejemplo, es posible que se desee que ciertos modelos tengan datos por defecto o que se quieran incorporar datos desde un ERP previo en el proceso de migración (las referencias de los productos, clientes, empleados…). Para todo ello existen dos opciones, el uso de ficheros XML o el uso de ficheros CSV.

Los ficheros de datos, para que sean cargados por Odoo, deben ser incluidos en el apartado data del fichero *__manifest__.py* teniendo en cuenta que el orden en el que se incluyen es muy importante para el correcto funcionamiento de la carga.

2 FICHEROS *XML*

Si tomamos como ejemplo la definición de las vistas, observaremos que es un XML con una etiqueta `<odoo>`, otra `<data>` y dentro los `<record>` de cada vista. Así es como le decimos a *Odoo* lo que se debe guardar en la base de datos.

```
<odoo>
      <data>
        <record model="{model name}" id="{record identifier}">
            <field name="{a field name}">{a value}</field>
        </record>
      </data>
<odoo>
```

En este ejemplo, en el modelo ponemos el nombre del modelo en el que guardará y en *id* el *External ID*. Tras ello, indicamos cada uno de los *fields* a los que queremos darle valor.

¿Qué es un *External ID* en *Odoo*?

La primera consideración a tener en cuenta es que estamos utilizando un ORM que transforma las declaraciones de clases que heredan de *models.Model* en tablas de *PostgreSQL* y los *records* declarados en XML en registros de esas tablas. Todos los registros del ORM tienen una columna *id* que los identifica de forma unívoca en su tabla. Esto permite que, durante la ejecución del programa, funcionen las claves ajenas entre modelos. Esto no tiene ninguna diferencia respecto al modelo tradicional sin ORM.

El problema al que se enfrentan los programadores de Odoo es que hay que crear ficheros de datos XML en los que se definen relaciones entre modelos antes de instalar el módulo. Estas relaciones no se pueden referir al *id* porque es un código auto-numérico que no es predecible en el momento de programar.

Para solucionarlo se inventó el *External ID*. Este identificador está escrito en lenguaje humano y ha de ser distinto a cualquier identificador del programa. Para garantizar eso se recomienda poner el nombre del módulo, un punto y un nombre que identifique la utilidad y significado del registro.

Hay que tener en cuenta que todos los elementos de *Odoo* pueden tener un identificador externo: módulos, modelos, vistas, acciones, menús, registros, fields, etc. Por eso hay que establecer unas reglas. Por ejemplo: *school.teacher_view_form* serviría para el formulario que muestra a los profesores del módulo *school*.

☑ Podemos buscar los identificadores externos directamente en el modo desarrollador de *Odoo* en el apartado de *Ajustes* > *Técnico* > *Identificadores externos*.

Cuando se hacen los ficheros de datos, los *fields* simples son muy sencillos de rellenar. Los *Binary* e *Image* tienen que estar en formato *Base64*, pero eso se puede conseguir fácilmente con el comando *base64* de GNU/Linux o sitios web como base64decode .

Lo más complicado son los *field* relacionales. Para conseguirlo hay que utilizar los identificadores externos, ya que no es recomendado en ningún caso usar el campo *id*. En realidad, al crear la relación se guardará el *id* en la base de datos, pero se hará después de evaluar el *External ID*.

Para rellenar un *Many2one* hay que usar `ref()`:

```
<field name="product_id" ref="product.product1"/>
```

En ocasiones queremos que el valor sea calculado con Python durante el momento de la instalación del módulo. Para ello usamos `eval()`:

```
<field name="date" eval="(datetime.now()+timedelta(-1)).strftime('%Y-%m-%d')"/>
<field name="product_id" eval="ref('product.product1')"/> # Equivalente al ejemplo anterior
<field name="price" eval="ref('product.product1').price"/>
```

Para los *x2many* se deben usar `eval()` con `ref()` y una tripleta que indica lo que hay que hacer:

```
<field name="tag_ids" eval="[(6,0,
[ref('fleet.vehicle_tag_leasing'),ref('fleet.vehicle_tag_compact'),
ref('fleet.vehicle_tag_senior')] )]" />
```

☑ Para que la inserción del registro se realice correctamente en la tabla correspondiente, es necesario que el registro al que se hace referencia con *ref* exista previamente en la base de datos, por lo que debe ser incluido previamente a la inclusión del registro desde el que se hace referencia. Dicho en otras palabras, es necesario que el elemento con el *external ID* al que hacemos referencia se ubique antes del elemento a cargar.

Esas tripletas tienen los siguientes significados:

1. *(0, _ , {'field': value})*: crea un nuevo registro y lo vincula.
2. *(1, id, {'field': value})*: actualiza los valores de un registro ya vinculado.
3. *(2, id, _)*: desvincula y elimina el registro

4. *(3, id, _)*: desvincula, pero no elimina el registro de la relación.
5. *(4, id, _)*: vincula un registro que ya existe.
6. *(5, _, _)*: desvincula, pero no elimina todos los registros.
7. *(6, _, [ids])*: reemplaza la lista de registros.

También se puede usar para borrar registros:

```
<delete model="cine.session" id="session_cine1_1"></delete>
```

2.1 *noupdate*

Hay que tener en cuenta que cada vez que instalemos o, sobre todo, acutualicemos el módulo, los fichero de datos serán cargados. El atributo *noupdate* indica si los registros incluidos en el fichero para un modelo determinado deben actualizarse o no. Por defecto, su valor es 0 (*False*) por lo que, si los registros definidos en el fichero son modificados a mano por los usuarios del módulo, al actualizar el módulo las modificaciones se perderán.

```
<odoo>
  <data noupdate="1">
    <record id="factory.product1" model="factory.product"/>
      <field name="name"/>Mesa<field>
      <field name="price"/>35<field>
    </record>

    <record id="factory.product2" model="factory.product"/>
      <field name="name"/>Sofa<field>
      <field name="price"/>350<field>
    </record>
  </data>
</odoo>
```

Como se puede, la declaración del atributo afecta a todos los registros definidos dentro del apartado *data*. Si queremos tener registros con el valor por defecto de *noupdate,* es posible crear otro bloque *data* para de modelo sin el atributo *noupdate.* Además, en caso de estar en modo desarrollador, es posible acceder al menu *Ajustes / Técnicos / External ID* y desde ahí a uno de los registros *existentes* y modificar el atributo *noupdate.*

2.2 forcecreate

De manera similar al *noupdate*, existe otro atributo llamado *forcecreate*. Este atributo se define para cada registro (el *noupdate* era a nivel de bloque *data*) e indica que el registro sólo se debe crear la primera vez, pero en el caso de que el usuario lo borre de manera manual, no debe ser insertado de nuevo. Su valor por defecto es 1 (*True*), es decir, el registro se crea siempre.

```
<odoo>
  <data noupdate="1">
    <record id="factory.product1" model="factory.product" forcecreate="0"/>
      <field name="name"/>Mesa<field>
      <field name="price"/>35<field>
    </record>
  </data>
</odoo>
```

2.3 Funciones

Una de las posibilidades más interesantes de la inserción mediante ficheros XML es la posibilidad de poder llamar a funciones que se ejecuten por cada uno de los registros a insertar

Por ejemplo, siguiendo con el caso anterior, queremos incluir un campo en el modelo producto que sea el tipo de producto, tipo que se define con las tres primeras letras del nombre en mayúsculas. Evidentemente se puede incluir a mano, pero es costoso y además propenso a errores y mucho más en el caso de que realicemos modificaciones de esos nombres.

Para ello definimos una función `create_type` que será la encarga de calcular el tipo. Dicha función podría ser algo como:

```
@api.model
def _create_type(self):
    for prod in self.search(['type','=',False]):
      prod.type = prod[:3].upper()
```

Una vez creada, en el documento XML la llamamos

```
<odoo>
  <data noupdate="1">
    ...
    <function model="factory.product" name="_create_type"/>
  </data>
</odoo>
```

☑ Es importante tener en cuenta que la ejecución de la función se realizará cada vez que se instale o se actualice el módulo, por lo que la propia función debe de incluir la lógica necesaria para evitar sobreescrituras. Por ejemplo, en este caso filtramos por aquellos productos que aún no tengan asignado el type `self.search(['type','=',False])`.

3 FICHEROS *CSV*

Los ficheros CSV permiten también la incorporación de datos pero de una manera menos detallada y con menos funcionalidades que los XML. De hecho el propio *scaffolding* genera el fichero de permisos mediante este formato.

La creación de los ficheros requiere cumplir una serie de condiciones:

- Tiene que haber un fichero por modelo
- El nombre del fichero tiene que ser el nombre del modelo (ojo, no confundir con el nombre en la base de datos), por ejemplo: *ir.model.access*
- En la primera línea del fichero tiene que aparecer los nombre de los campos del modelo que vamos a rellenar, además de uno que se llamará *id*. El *id*, que es el extrenal ID del registro, es obligatorio. El resto lo será si son *required*.

```
# Fichero  factory.product.csv

Id,name,price
factory.product1,Mesa,35
factory.product2,Sofa,350
```

Los ficheros CSV no permiten ni el uso de funciones ni de los atributos *noupdate* y *forcecreate.*

3.1 Relaciones

La creación de registros que tengan relaciones *Many2One*, *One2many o Many2many* es posible utilizando cierta notación.

En el caso de relaciones *Many2One* es necesario inidcar como nombre del campo el nombre de del campo de la relación seguido de *:id,* por ejemplo *wood_id:id.* En cada uno de los registros se indica el identificador externo del registro vinculado.

```
# Fichero  factory.collection.csv

Id,name
factory.collection_1,Eden
factory.collection_2,Joil


# Fichero  factory.product.csv

Id,name,price,collection_id:id
factory.product1,Mesa,35,factory.collection_1
factory.product2,Silla,46,factory.collection_2
```

☑ Las relaciones *One2many* se generan automáticamente a partir de las *Many2one*.

Las relaciones *Many2many* funcionan de manera similar a las *Many2one* pero en este caso los identificadores se incluyen como una cadena de *externals Id* separados por comas.

```
# Fichero  factory.wood.csv

Id,name
factory.wood_1,Pino
factory.wood_2,Nogal


# Fichero  factory.product.csv

Id,name,price,wood_id:id
factory.product1,Mesa,35,"factory.wood_1,factory.wood_2"
factory.product2,Silla,46,"factory.wood_2"
```

☑ En cualquiera de los dos casos, para que los registros sean encontrados por *Odoo* es necesario que la carga de los ficheros se realice en el orden correcto.

4 BIBLIOGRAFIA

1. Documentación de Odoo
2. odoo-master

5 AUTORES

- Alfredo Oltra