Curso Técnico de Odoo Starter-Udemy-HNET

Edgar Josué Benedetto Godoy

0801-1997-23600

ebenedetto@hnetw.com

edgar.benedetto@unah.hn

+504 3330-0171

12/07/2021

Sección 2 - Instalar Odoo con Docker

Comandos Docker

- Ver procesos activos:

```
docker ps
```

Docker

- Son imagenes que ya están compiladas listas para consumir y descargar

Pasos para instalar Odoo con Docker

1. Iniciar el servidor PostgreSQL

```
docker run -d -e POSTGRES_USER=odoo -e POSTGRES_PASSWORD=odoo -e POSTGRES_DB=postgres --name db
postgres:10
```

1. Iniciar una instancia de Odoo
   - El alias del contenedor que ejecuta Postgres debe ser db para que Odoo pueda conectarse al servidor de Postgres.
   - Explicación del siguiente comando:
     A. Crear contenedor igual que en el comando anterior que lo hizo con la bd
     B. El puerto 8069 queda abierto
     C. El nombre del contenedor (puede ser carácteristico de la empresa o versión de Odoo)
     D. Tiene un enlace con el contenedor db
     E. La imagen a descargar sin especificar versión, para que descargue la última versión disponible

```
docker run -p 8069:8069 --name odoo13 --link db:db -t odoo
```

Opcionales

1. Detener y reiniciar la instancia de Odoo

```
$ docker stop odoo
$ docker start -a odoo
```

1. Detener y reiniciar el servidor PostgreSQL

- Cuando se reinicia un servidor PostgreSQL, las instancias de Odoo vinculadas a ese servidor también deben reiniciarse porque la dirección del servidor ha cambiado y, por lo tanto, el enlace está roto.

- Reiniciar el servidor PostgreSQL no afecta las bases de datos creadas

1. Ejecutar Odoo con una configuración personalizada

- El archivo de configuración predeterminado para el servidor (ubicado en `/etc/odoo/odoo.conf`) se puede anular al inicio utilizando volúmenes. Suponga que tiene una configuración personalizada en `/path/to/config/odoo.conf` , luego

```
docker run -v /path/to/config:/etc/odoo -p 8069:8069 --name odoo --link db:db -t odoo
```

1. Montar addons personalizados
   - Puede montar sus propios complementos de Odoo dentro del contenedor de Odoo, en `/mnt/extra-addons`

Sección 4 - Modelos en Odoo

1. **Desarrollar diagramas UML entidad-relación o principalmente entidades con campos**
2. **Crear el modelo y actualizar el modulo en Odoo**
3. **Comprobar que se creo correctamente revisando la tabla en PostgreSQL**

```
psql odoo13

select * from module_name;
```

Ejemplo de la sección

- **Nombre del módulo: book**
- **Nombre del modelo: library_book**
- **Nombre del archivo:**

```
library_book.py
```

- **Contenido del archivo**

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------------

    @author ebenedetto@hnetw.com
    @date 12/07/2021
    @decription Modelo de libro
    @name_file library_book.py
    @version 1.0

-----------------------------------------------------------------------------------------------------------
"""

from odoo import models, fields, api

class LibraryBook(models.Model):
  _name = "library.book"

  name = fields.Char(string="Name")
  active = fields.Boolean("Is Active")
  image = fields.Binary()
  pages = fields.Integer(string="# Pages")
  isbn = fields.Char(string="ISBN", size=13)
  description = fields.Html(string="Description")

  # Campo creado por la sección 4 Campos relacionales
  category_id = fields.Many2one("library.category", string="Category")
```

Curso Técnico de Odoo Starter-Udemy-HNET

Edgar Josué Benedetto Godoy

0801-1997-23600

ebenedetto@hnetw.com

edgar.benedetto@unah.hn

+504 3330-0171

13/07/2021

Sección 5 - Vistas, acciones y Menú en Odoo

Vistas

- **Permite representar registros de la base de datos de un determinado modelo de manera visual. Las vistas más utilizadas son:**

  1. **Vista Formulario - Editar o crear registros**
  2. **Vista Lista**
  3. **Vista de árbol**

Descripción de comandos de una vista

- `<record id="">` **El id del record debe ser único y siempre en minusculas**

Acciones

- **Controlador para llamar a una vista que está definida para un particular modelo**

Menú

- **Le permiten al usuario ejecutar una acción**

Ejemplo de la sección

- **Recordar incluir este archivo en el** **manifest** **del modulo**
- **Nombre del módulo:** **book**
- **Nombre del modelo:** **library_book**
- **Nombre del archivo:**

  `library_book.xml`
- **Contenido del archivo**

```
<? xml version = "1.0" encoding = "utf-8"?>
<odoo>
  <!-- Form View -->
  <record id="library_book_form_view" model="ir.ui.view">
    <field name = "name">library.book.form.view</field>
    <field name = "model">library.book</field>
    <field name = "arch" type="xml">
      <form string="Form Book">
        <sheet>
          <!-- Atributo widget para visualizar mejor la imagen -->
          <field name="image" widget="image"/>
          <group>
            <!-- Campos del Modelo -->
            <field name="name"/>
            <field name="active"/>
            <field name="pages"/>
            <field name="isbn"/>
          </group>
          <group>
            <field name="description"/>
          </group>
        </sheet>
      </form>>
    </field>
  </record>

  <!-- Tree View -->
  <record id="library_book_tree_view" model="ir.ui.view">
    <field name = "name">library.book.tree.view</field>
    <field name = "model">library.book</field>
    <field name = "arch" type="xml">
      <tree string="Tree Book">
        <!-- Campos más relevantes -->
        <field name="name"/>
        <field name="active"/>
        <field name="isbn"/>
        <!-- Campo relacional -->
        <field name="category_id"/>
      </tree>
    </field>
  </record>
```

```
    <!-- Action Book -->
    <record id="action_library_book" model="ir.actions.act_window">
      <!-- Campos más relevantes -->
      <field name="name"/>action.library.book</field>
      <field name="type"/>ir.actions.act_window</field>
      <field name="res_model"/>library.book</field>
      <field name="view_mode"/>tree,form</field>
    </record>

    <!-- Menús -->
    <menuitem id="library_menu_root" name"Library" sequence="1"/>
    <menuitem id="library_book_menu_category" name"Book" sequence="1"
  parent="library_menu_root"/>
    <menuitem id="action_library_book_menu" name"Library" sequence="1"
  parent="library_book_menu_category" action="action_library_book"/>

  </odoo>
```

Sección 4 - Campos Relacionales Many2Many

- **Un autor puede tener uno o varios libros**

Ejemplo de la sección

- **Nombre del módulo: book**
- **Nombre del modelo: library_author**
- **Nombre del archivo:**

```
library_author.py
```

- **Contenido del archivo**
- **Para agregar el campo Many2Many a la vista:**

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

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------

    @author ebenedetto@hnetw.com
    @date 01/07/2021
    @decription Modelo de autor
    @name_file library_author.py
    @version 1.0

-----------------------------------------------------------------------------------------------------
"""

from odoo import models, fields, api

class LibraryAuthor(models.Model):
  _name = "library.author"

  name = fields.Char(string="Name")
  active = fields.Boolean("Is Active?", default=True)

  # Campo Many2one
  country_id = fields.Many2one("res.country")

  # Campo Many2Many
  book_ids = fields.Many2Many("library.book", string="Books")
```

Sección 4 - Herencia de un modelo en Odoo

- **Permite ampliar las funcionalidades del sistema, se puede aplicar herencia sobre modelos y vistas**
- **Modificar modelo existente agregarle funcionalidad, campo, botón, etc**

Formato para crear modulo con scaffold Pycharm

```
./odoo-bin scaffold ../extra_addons/module_name "path_to_your_proyect"
```

Ejemplo de la sección

- **Recordar --> Agregar en el manifest** `'aplication':True`
- **Nombre del módulo: new_book**
- **Nombre del modelo: library_book**
- **Nombre del archivo:**

```
library_book.py
```

- **Contenido del archivo**

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------------

    @author ebenedetto@hnetw.com
    @date 13/07/2021
    @decription Extensión del Modelo de libro
    @name_file library_book.py
    @version 1.0

-----------------------------------------------------------------------------------------------------------
"""

from odoo import models, fields, api

# models.Model --> Es una herencia de una tabla persistente SQL en PostgreSQL
class LibraryBook(models.Model):
  _inherit = "library.book"

  # Campo de tipo fecha
  date = fields.Date(string="Release Date")

  # Campo necesario para la sección 5 - Herencia de Views XML en Odoo
  author_id = fields.Many2one("library.author")

  # Campo calculado usado en la sección 6 - Campos calculados
  category_count = fields.Integer(
      string="Nbr Categories",
      compute="_count_categ"
  )

  def _count_categ(self):
    """
    self --> Es un record set, y el valor depende de donde se invoque, puede
      valer más de un registro, como se invoca desde una vista lista, se
      cargan todos los registros, por lo tanto se debe recorrer con un ciclo
    """
    for book in self:
      book.categ_count = len(book.category_ids)
```

Ejemplo de la sección

- Recordar incluir este archivo en el**manifest** del modulo
- Nombre del módulo: **new_book**
- Nombre del modelo: **library_book**
- Nombre del archivo:

```
library_book.xml
```

- Contenido del archivo

```
<? xml version = "1.0" encoding = "utf-8"?>
<odoo>
  <!-- Form View -->
  <record id="library_book_form_view_inherit" model="ir.ui.view">
    <field name = "name">library.book.form.view</field>
    <field name = "model">library.book</field>
    <field name = "inherit_id" ref="library.library_book_form_view"
    <field name = "arch" type="xml">
      <!-- Agregar el campo nuevo después del campo name -->
      <field name="name" position="after">
        <field name="date"/>
        <field name="categ_count"/>
      </field>
    </field>
  </record>
</odoo>
```

Sección 4 - Campos relacionados en Odoo

- Permite crear relaciones entre modelos, en SQL significaría bases de datos relacionales

1. Many2one N:1
2. One2Many 1:N
3. Many2many N:M

- Por ejemplo un libro puede tener muchas categorias, y una categoria puede estar

Ejemplo de la sección

- Nombre del módulo: **book**
- Nombre del modelo: **library_category**
- Nombre del archivo:

```
library_category.py
```

- Contenido del archivo

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------------

    @author ebenedetto@hnetw.com
    @date 13/07/2021
    @decription Modelo de categoria de libro
    @name_file library_book.py
    @version 1.0

-----------------------------------------------------------------------------------------------------------
"""

from odoo import models, fields, api

class LibraryCategory(models.Model):
  _name = "library.category"

  name = fields.Char(string="Name")
  active = fields.Boolean("Is Active?")
```

Agregar un en una vista heredada usando xpath

1. **Heredar la vista**
2. **Especificar el campo de referencia para determinar si el nuevo campo se agregará antes o después o reemplazará a este campo de referencia**
3. **Agregar el campo nuevo**

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
    <!-- Vista heredada y XPATH -->
    <record id="res_partner_inherit" model="ir.ui.view">
        <field name="name">Res Partner Inherit</field>
        <field name="model">res.partner</field>
        <field name="inherit_id" ref="base.view_partner_form">
        <field name="arch" type="xml">
            <xpath expr="//field[@name='vat']" position = "after">
                <field name="message"/>
            </xpath>
            <xpath expr="//field[@name='vat']" position = "before">
                <field name="other_information"/>
            </xpath>
            <xpath expr="//field[@name='vat']" position = "replace">
                <field name="new_message"/>
            </xpath>
        </field>
    </record>
</odoo>
```

Sección 5 - Herencia de Views XML en Odoo

- **En Odoo en modo desarrollador entrando en el Edit View Form de un formulario, el campo de External ID es el que contiene el identificador de la vista, el cual se debe colocar en ref**
- **Para heredar una vista se usa el campo inherit_id y en ref colocando el nombre del modulo y se coloca el el id del xml de la vista a heredar**

```
<field name="inherit_id" ref="module_name.parent_xml_id"

<field name="inherit_id" ref="library.library_book_form_view"
```

Ejemplo de la sección

- **Recordar incluir este archivo en el manifest del modulo**
- **Nombre del módulo: book**
- **Nombre del modelo: library_author**
- **Nombre del archivo:**

```
  author_view.xml
```

- **Contenido del archivo**

```
<? xml version = "1.0" encoding = "utf-8"?>
<odoo>
  <!-- Inherit Form View book, add author_id -->
  <record id="library_book_form_view_author" model="ir.ui.view">
    <field name = "name">library.book.form.author</field>
    <field name = "model">library.book</field>
    <field name = "inherit_id" ref="library.library_book_form_view"
    <field name = "arch" type="xml">
      <!-- Agregar el campo nuevo después del campo name -->
      <field name="isbn" position="after">
        <field name="author_id"/>
      </field>
    </field>
  </record>
</odoo>
```

Sección 6 - ORM - Default Get (Bonus)

ORM

- Object relational mapping - Traduce tablas SQL a modelos en Python
- Solo se traducen carácteristicas importantes como campos, restricciones, etc.

Métodos comunes en el framework de Odoo

- Métodos por defecto al construir un modelo:
  1. default_get() --> Se ejecuta antes del método create(), Sirve para cargar valores por defecto del registro nuevo a crear, recibe una lista de parámetros, lleva un decorador @api.model decorator (uso obligatorio)
  2. create() --> Se ejecuta cuando se crean registros
  3. write() --> Modificar registros
  4. update() --> Actualizar registros
  5. unlink() --> Elimiar registros (similar a DELETE)
  6. search() --> Buscar entre los registros
  7. browse() --> Buscar y manipular registros

Ejemplo de la sección

- Recordar --> Agregarlo en el**manifest**
- Nombre del módulo: **sale_date_order**
- Nombre del modelo: **sale_order**
- Nombre del archivo:

  `sale_order.py`
- Contenido del archivo

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------------

    @author ebenedetto@hnetw.com
    @date 13/07/2021
    @decription Modelo de fecha de orden de venta
    @name_file sale_order.py
    @version 1.0

-----------------------------------------------------------------------------------------------------------
"""

from odoo import models, fields, api

class SaleOrder(models.Model):
  # Heredar orden de venta
  _name = "sale.order"

  @api.model
  def default_get(self, fields_list):
    """
    Colocar un valor por default en el campo de fecha de expiración

    Se llama a super, se le envía como parámetro el nombre de la clase
      fields_list --> Lista de parámetros

    Recibe una lista de parámetros y devuelve un dict
    """
    res = super(SaleOrder, self).default_get(fields_list)

    # MALA PRÁCTICA
    # Obtener la fecha actual haciendo una asignación directa
    # res['validity_date'] = fields.Date.today()

    # BUENAS PRÁCTICAS
    # Utilizar update en vez de la asignación directa de un campo
    res.update({'validity_date':fields.Date.today()})
    return res
```

Sección 6 - ORM - Campos Calculados

- Se agrega **compute** al atributo que se quiera ejecutar `code = fields.Char(compute="_get_code")`
- Siempre se ejecutan cuando son imbocados, no es un valor constante que se almacena en base de datos, pero si se puede almacenar
- Los valores son dinámicos, pero se debe tener cuidado a nivel de rendimiento, por ejemplo cuando se quiere **sumar el total de distintas facturas en una vista tree**

Declaración e invocación de campos calculados

1. Si se declara en una vista **List** lo va a invocar al método
2. Si se declara en la vista **Form** se invoca solo al entrar al Form
3. Los campos calculados se invocan dependiendo donde se declaren y solo se invocan entrando en la vista

Sección 6 - Browse, Search y Domain en Odoo

Domain o Dominio - Filtrar registros

- Definición: Un **dominio** está compuesto por listas y tuplas en pyton, cada tupla representa una clausula **WHERE**
- Cada vez que se usa el buscador, se hace uso de filtros

In [ ]:

```
# SQL
# SELECT id, name from library_book WHERE id = 1;

# Python
domain = [('id', '=', 1)]
self.env['library.book'].search(domain)

# SQL
# SELECT id, name from library_book WHERE id = 2;

# Python
res = self.env['library.book'].search([('id', '=', 2)])
res.id # 2
res.name # Nombre del libro
```