1. Instala odoo en tu ubuntu 20.04

Instala en tu forma preferida la versión de ubuntu 20.04 (o si tienes la 16.04 también funciona).

NOTA: Si intentas usar otra distribución u otra versión a la indicada, es probable que tengas problemas con el script de instalación.

Por lo que te recomiendo ampliamente, usar Ubuntu-20.04

Conéctate como root en tu consola, usando:

**sudo su - root**

(Usa la contraseña de tu usuario actual)

O

**su - root**

(Usa la constraseña del usuario root)

Instala wget para poder descargar el archivo de instalación, corriendo los siguientes comandos:

**apt-get update && apt-get install wget**

Descarga y corre el siguiente script

**wget https://raw.githubusercontent.com/Vauxoo/docker-odoo-training/master/install.sh -O odoo\_install.sh**  
**chmod +x ./odoo\_install.sh**  
**./odoo\_install.sh odoousr**

Al terminar, tendrás un usuario del sistema operativo llamado "odoousr" con el password "odoousrpwd" con el cuál podrás acceder con:

**sudo su - odoousr**

**Advertencia**

De ahora en adelante, ya no hagas nada con el usuario "root", asegúrate de que siempre tengas el usuario recién creado logueado en tu consola.

Dentro de este usuario, tendrás una carpeta llamada odoo-repo donde podrás encontrar odoo-bin

Crea tus llaves ssh

Crea tus llaves ssh para el usuario odoousr, corre el siguiente comando:

**ssh-keygen -t rsa -b 4096 -C "micorreo@vauxoo.com"**

(A todo dale enter, enter, enter sin contraseña)

Crea un usuario en [github.com](http://github.com/)

Copia tu llave pública ejecutando **cat ~/.ssh/id\_rsa.pub**

Agrega tu llave pública en <https://github.com/settings/ssh/new>

Prueba que tu consola puede conectarse a github, corre el siguiente comando:

**ssh -T git@github.com**

**Advertencia**

Deberías de ver un mensaje que dice: "Hi TU NOMBRE!"

Configura tu usuario y correo a tu git local, corre los siguientes comandos:

**git config --global user.name "Tu Nombre"**  
**git config --global user.email "tucorreo@tudominio"**

NOTA: Cambia "Tu Nombre" por tu nombre real, y "tucorreo@tudominio" por tu cuenta de correo real.

Comprueba tu configuración con:

**git config -l**

**Advertencia**

Deberías de ver  tu user.name y tu user.email recientemente configurados.

Crea un repositorio en github, llamado "openacademy-repo":

<https://github.com/new>

Inicia/Detén el servidor Odoo

Odoo utiliza una arquitectura cliente/servidor en la que los clientes son los navegadores que acceden al servidor Odoo vía RPC.

La lógica de negocio y sus extensiones se realizan generalmente del lado del servidor, aunque se apoya en algunas características del cliente (por ejemplo, una representación de datos como mapas interactivos) que pueden ser añadidas al cliente.

Para iniciar el servidor, simplemente ejecuta el comando [odoo-bin](http://vauxoo.github.io/odoo/reference/cmdline.html#reference-cmdline) en la consola, agregando la ruta completa al archivo:

**./odoo-bin**

El servidor se detiene pulsando **Ctrl-C** dos veces desde la terminal, o matando el proceso correspondiente.

SEE VIDEO nro 1.

**Construir un módulo para Odoo**

Las extensiones de servidor y cliente se empaquetan como módulos que opcionalmente se cargan en una base de datos.

Los módulos en Odoo pueden añadir nueva lógica de negocio a un sistema Odoo, o modificar y ampliar la lógica de negocios existente: puede crear un módulo para agregar las reglas de contabilidad de su país adicionándolo al soporte de contabilidad genérico de Odoo, mientras que otro módulo puede añadir soporte para la visualización en tiempo real de una flota de autobuses.

Todo en Odoo comienza y termina con módulos.

Composición de un módulo

Un módulo de Odoo puede contener una serie de elementos:

Objetos de negocios

Declarado como clases de Python, estos recursos son automáticamente persistentes en Odoo basados en su configuración

Archivos de datos

Los archivos XML o CSV declaran metadatos (vistas o reportes), datos de configuración (parametrizan los módulos), datos de demostración y más

Controladores Web

Manejo de peticiones de los navegadores web

Datos estáticos Web

Archivos de imágenes, CSS o javascript utilizados por la interfaz web o sitio web

Estructura de un módulo

Cada módulo es un directorio dentro de un directorio de módulos. Los directorios de módulos se especifican mediante la opción --addons-path.

Truco

la mayoría de las opciones de línea de comandos también se pueden establecer usando un archivo de configuración

Un módulo de Odoo es declarado por su manifiesto. Ver la Documentación del manifiesto aquí.

Un módulo es también un paquete de Python con un archivo \_\_init\_\_.py, que contiene instrucciones de importación de varios archivos de Python en dicho módulo.

Por ejemplo, si un módulo tiene un solo archivo mymodule.py, el archivo \_\_init\_\_.py podría contener:

from . import mymodule

Odoo ofrece un mecanismo para ayudar a comenzar un nuevo módulo, el comando odoo-bin tiene un subcomando scaffold para crear un módulo vacío:

$ odoo-bin scaffold <module name> <where to put it>

El comando crea un subdirectorio para un módulo y automáticamente crea un montón de archivos estándar para dicho módulo. La mayoría de ellos contienen sólo código comentado o XML. A lo largo de este tutorial se explicará el uso de la mayoría de estos archivos.

Ejercicio

Creación de un módulo

Utilizar la línea de comandos anterior para crear un módulo vacío Open Academy e instalarlo en Odoo.

Ejecuta el comando odoo-bin scaffold openacademy addons.

Adapta el archivo de manifiesto al módulo.

No te preocupes por los demás archivos.

openacademy/\_\_manifest\_\_.py

# -\*- coding: utf-8 -\*-

{

'name': "Open Academy",

'summary': """Manage trainings""",

'description': """

Open Academy module for managing trainings:

- training courses

- training sessions

- attendees registration

""",

'author': "My Company",

'website': "http://www.yourcompany.com",

# Categories can be used to filter modules in modules listing

# Check https://github.com/odoo/odoo/blob/master/odoo/addons/base/module/module\_data.xml

# for the full list

'category': 'Test',

'version': '0.1',

# any module necessary for this one to work correctly

'depends': ['base'],

# always loaded

'data': [

# 'security/ir.model.access.csv',

'templates.xml',

],

# only loaded in demonstration mode

'demo': [

'demo.xml',

],

}

openacademy/\_\_init\_\_.py

# -\*- coding: utf-8 -\*-

from . import controllers

from . import models

openacademy/controllers.py

# -\*- coding: utf-8 -\*-

from odoo import http

# class Openacademy(http.Controller):

# @http.route('/openacademy/openacademy/', auth='public')

# def index(self, \*\*kw):

# return "Hello, world"

# @http.route('/openacademy/openacademy/objects/', auth='public')

# def list(self, \*\*kw):

# return http.request.render('openacademy.listing', {

# 'root': '/openacademy/openacademy',

# 'objects': http.request.env['openacademy.openacademy'].search([]),

# })

# @http.route('/openacademy/openacademy/objects/<model("openacademy.openacademy"):obj>/', auth='public')

# def object(self, obj, \*\*kw):

# return http.request.render('openacademy.object', {

# 'object': obj

# })

openacademy/demo.xml

<odoo>

<!-- -->

<!-- <record id="object0" model="openacademy.openacademy"> -->

<!-- <field name="name">Object 0</field> -->

<!-- </record> -->

<!-- -->

<!-- <record id="object1" model="openacademy.openacademy"> -->

<!-- <field name="name">Object 1</field> -->

<!-- </record> -->

<!-- -->

<!-- <record id="object2" model="openacademy.openacademy"> -->

<!-- <field name="name">Object 2</field> -->

<!-- </record> -->

<!-- -->

<!-- <record id="object3" model="openacademy.openacademy"> -->

<!-- <field name="name">Object 3</field> -->

<!-- </record> -->

<!-- -->

<!-- <record id="object4" model="openacademy.openacademy"> -->

<!-- <field name="name">Object 4</field> -->

<!-- </record> -->

<!-- -->

</odoo>

openacademy/models.py

# -\*- coding: utf-8 -\*-

from odoo import models, fields, api

# class openacademy(models.Model):

# \_name = 'openacademy.openacademy'

# name = fields.Char()

openacademy/security/ir.model.access.csv

id,name,model\_id/id,group\_id/id,perm\_read,perm\_write,perm\_create,perm\_unlink

access\_openacademy\_openacademy,openacademy.openacademy,model\_openacademy\_openacademy,,1,0,0,0

openacademy/templates.xml

<odoo>

<!-- <template id="listing"> -->

<!-- <ul> -->

<!-- <li t-foreach="objects" t-as="object"> -->

<!-- <a t-attf-href="{{ root }}/objects/{{ object.id }}"> -->

<!-- <t t-esc="object.display\_name"/> -->

<!-- </a> -->

<!-- </li> -->

<!-- </ul> -->

<!-- </template> -->

<!-- <template id="object"> -->

<!-- <h1><t t-esc="object.display\_name"/></h1> -->

<!-- <dl> -->

<!-- <t t-foreach="object.\_fields" t-as="field"> -->

<!-- <dt><t t-esc="field"/></dt> -->

<!-- <dd><t t-esc="object[field]"/></dd> -->

<!-- </t> -->

<!-- </dl> -->

<!-- </template> -->

</odoo>

Mapeo Objeto-Relacional

Un componente clave de Odoo es el ORM. Esta capa evita escribir la mayoría del SQL manualmente y proporciona extensibilidad y servicios de seguridad 2.

Los objetos de negocio se declaran como clases Python que extienden Model la cual los integra en el sistema automatizado de persistencia.

Módulos pueden ser configurados para establecer un número de atributos en su definición. El atributo más importante es \_name el cuál es requerido y define el nombre para el modelo en el sistema de Odoo. Aquí es mínimamente completada la definición de un modelo:

from odoo import models

class MinimalModel(models.Model):

\_name = 'test.model'

Campos en el Modelo

Los campos se utilizan para definir lo que puede almacenar el modelo y dónde. Los campos se definen como atributos en el modelo:

from odoo import models, fields

class LessMinimalModel(models.Model):

\_name = 'test.model2'

name = fields.Char()

Atributos comunes

Así como en el modelo, los campos se pueden configurar pasando los atributos como parámetros normales:

name = field.Char(required=True)

Algunos atributos están disponibles en todos los campos, aquí están los más comunes:

string (unicode, default: field's name)

La etiqueta del campo en la interfaz del usuario (lo que el usuario leerá).

required (bool, default: False)

Si es Verdadero, el campo no puede estar vacío, deberá tener un valor predeterminado o siempre se tendrá que asignar un valor al crear un registro.

help (unicode, default: '')

Formato largo, proporciona una información de ayuda en la interfaz de usuario.

index (bool, default: False)

Solicita a Odoo la creación de un índice database index en la columna.

Campos Simples

En general hay dos categorías de campos: Los campos ”Simples" los cuales son valores atómicos almacenados directamente en la tabla correspondiente al modelo, y los campos “Relacionales” los cuales enlazan registros (registros del mismo modelo o de diferentes modelos).

Ejemplo de campos simples son los Boolean, Date, Char.

Campos reservados

Odoo crea algunos campos en todos los modelos1. Estos campos son manejados por el sistema y no deberían ser escritos. Ellos pueden ser leídos si representan alguna utilidad o son necesarios:

id (Id)

El identificador único para un registro en su modelo.

create\_date (Datetime)

Fecha de creación del registro.

create\_uid (Many2one)

Usuario quien creó el registro.

write\_date (Datetime)

Fecha de la última modificación del registro.

write\_uid (Many2one)

usuario quien hizo la última modificación del registro.

Campos Especiales

Por defecto, Odoo también requiere un campo name en todos los modelos para varios comportamientos de búsqueda y visualización. El campo utilizado para estos fines se puede reemplazar estableciendo \_rec\_name.

Ejercicio

Define un modelo

Define un nuevo modelo de datos Curso en el módulo openacademy. Un curso tiene un título y una descripción. Los cursos deben tener un título.

Edita el archivo openacademy/models/models.py incluyendo una clase Course (intenta nombrar todo en inglés).

openacademy/models.py

from odoo import models, fields, api

class Course(models.Model):

\_name = 'openacademy.course'

name = fields.Char(string="Title", required=True)

description = fields.Text()

Archivos de datos

Odoo es un sistema altamente dirigido por datos. Aunque el comportamiento personalizado utiliza una parte de código Python el valor de un módulo está en los datos que éste establece cuando se carga.

Truco

algunos módulos existen únicamente para agregar datos a Odoo

Los datos del módulo son declarados vía Archivos de Datos, archivos XML con elementos <record>. Cada elemento <record> crea o actualiza registros en la base de datos.

<odoo>

<record model="{model name}" id="{record identifier}">

<field name="{a field name}">{a value}</field>

</record>

</odoo>

model es el nombre para un registro en el modelo de Odoo.

id es un external identifier (identificador único), que permite hacer referencia a un registro (sin necesidad de conocer su identificador en base de datos).

<field> los elementos tienen un nombre que es el nombre del campo en el modelo (por ejemplo, description). Su cuerpo es el valor de campo.

Los archivos de datos tienen que ser declarados en el archivo manifiesto para que puedan cargarse, ellos pueden ser declarados en la lista 'data' (la cual siempre se carga) o en la lista 'demo' (que se cargará solo cuando modo demostración esté habilitado).

Ejercicio

Define los datos de demostración

Crea los datos de demostración llenando el model Courses con algunos pocos cursos.

Edita el archivo openacademy/demo/demo.xml para incluir algunos datos.

openacademy/demo.xml

<odoo>

<record model="openacademy.course" id="course0">

<field name="name">Course 0</field>

<field name="description">Course 0's description

Can have multiple lines

</field>

</record>

<record model="openacademy.course" id="course1">

<field name="name">Course 1</field>

<!-- no description for this one -->

</record>

<record model="openacademy.course" id="course2">

<field name="name">Course 2</field>

<field name="description">Course 2's description</field>

</record>

</odoo>

Acciones y menús

Las acciones y menús son registros regulares en la base de datos, usualmente a través de archivos de datos. Las acciones pueden ser disparadas en tres formas:

haciendo click en los menús (enlazados a una acción específica)

haciendo click en botones en las vistas (si éstos están conectados a acciones)

como acciones contextuales en el objeto

Ya que los menús son algo complejos de declarar, hay un acceso directo <menuitem> para declarar un ir.ui.menu y así enlazarlo con mayor facilidad a la acción correspondiente.

<record model="ir.actions.act\_window" id="action\_list\_ideas">

<field name="name">Ideas</field>

<field name="res\_model">idea.idea</field>

<field name="view\_mode">tree,form</field>

</record>

<menuitem id="menu\_ideas" parent="menu\_root" name="Ideas" sequence="10"

action="action\_list\_ideas"/>

Peligro

La acción debe declararse antes de declarar el menú correspondiente en el archivo XML.

Los archivos de datos se ejecutan secuencialmente, el id de la acción debe estar presente en la base de datos antes de que se pueda crear el menú.

Ejercicio

Define nuevas entradas de menú

Define nuevas entradas de menú para acceder a cursos bajo la entrada del menú OpenAcademy. Un usuario debería ser capaz de :

ver una lista de todos los cursos

crear/modificar cursos

Crea openacademy/views/openacademy.xml con una acción y los menús para activar la acción

Añádelo a la lista de data de openacademy/\_\_manifest\_\_.py

openacademy/\_\_manifest\_\_.py

'data': [

# 'security/ir.model.access.csv',

'templates.xml',

'views/openacademy.xml',

],

# only loaded in demonstration mode

'demo': [

openacademy/views/openacademy.xml

<?xml version="1.0" encoding="UTF-8"?>

<odoo>

<!-- window action -->

<!--

The following tag is an action definition for a "window action",

that is an action opening a view or a set of views

-->

<record model="ir.actions.act\_window" id="course\_list\_action">

<field name="name">Courses</field>

<field name="res\_model">openacademy.course</field>

<field name="view\_mode">tree,form</field>

<field name="help" type="html">

<p class="oe\_view\_nocontent\_create">Create the first course

</p>

</field>

</record>

<!-- top level menu: no parent -->

<menuitem id="main\_openacademy\_menu" name="Open Academy"/>

<!-- A first level in the left side menu is needed

before using action= attribute -->

<menuitem id="openacademy\_menu" name="Open Academy"

parent="main\_openacademy\_menu"/>

<!-- the following menuitem should appear \*after\*

its parent openacademy\_menu and \*after\* its

action course\_list\_action -->

<menuitem id="courses\_menu" name="Courses" parent="openacademy\_menu"

action="course\_list\_action"/>

<!-- Full id location:

action="openacademy.course\_list\_action"

It is not required when it is the same module -->

</odoo>

Al final, después de que se haga el commit, como se ve en el video,

agrega un "git remote" a tu repositorio de github (el que creaste en la lección anterior desde https://github.com/new)

Solo ejecuta dentro de la carpeta ~/openacademy-repo lo siguiente:

cd ~/openacademy-repo

git init

git remote add origin <URL-SSH-DE-TU-PROYECTO>

Luego ejecuta

git push origin master

Refresca la página de github, y ahí deberás de tener tu código.

See video 2!!

## Vistas Básicas

Las vistas definen el modo en que se muestran los registros de un modelo. Cada tipo de vista representa un modo de visualización (una lista de registros, un gráfico, …). Las vistas se pueden solicitar ya sea de manera genérica a través de su tipo (por ejemplo una lista de clientes) o especificando su id. Para solicitudes genéricas, se utilizará la vista con el tipo y la prioridad más baja (por lo que la vista de la prioridad más baja de cada tipo es la vista predeterminada para ese tipo).

[La herencia de Vistas](http://vauxoo.github.io/odoo/reference/views.html#reference-views-inheritance) permite modificar vistas declaradas en otros lugares (agregar o quitar contenido).

### **Declaración de vista genérica**

Una vista es declarada como un registro del modelo **ir.ui.view**. El tipo de vista es implícita en el elemento raíz del campo **arch**:

**<record model="ir.ui.view" id="view\_id">**

**<field name="name">view.name</field>**

**<field name="model">object\_name</field>**

**<field name="priority" eval="16"/>**

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

**<!-- view content: <form>, <tree>, <graph>, ... -->**

**</field>**

**</record>**

**Peligro**

El contenido de la vista es XML.

El campo **arch** debe ser declarado como **type =“xml”** para que sea analizado correctamente.

### **Vistas de árbol**

Vistas de árbol, también llamadas vistas de lista, mostrarán registros en forma de tabla.

Su elemento raíz es **<tree>**. La forma más simple de la vista de árbol simplemente lista todos los campos a mostrar en la tabla (cada campo como una columna):

**<tree string="Idea list">**

**<field name="name"/>**

**<field name="inventor\_id"/>**

**</tree>**

### **Vistas de formulario**

Los formularios se utilizan para crear y editar registros individualmente.

Su elemento raíz es **<form>**. Están compuestos por una estructura de alto nivel (groups, notebooks) y los elementos interactivos (buttons y fields):

**<form string="Idea form">**

**<group colspan="4">**

**<group colspan="2" col="2">**

**<separator string="General stuff" colspan="2"/>**

**<field name="name"/>**

**<field name="inventor\_id"/>**

**</group>**

**<group colspan="2" col="2">**

**<separator string="Dates" colspan="2"/>**

**<field name="active"/>**

**<field name="invent\_date" readonly="1"/>**

**</group>**

**<notebook colspan="4">**

**<page string="Description">**

**<field name="description" nolabel="1"/>**

**</page>**

**</notebook>**

**<field name="state"/>**

**</group>**

**</form>**

**Ejercicio**

Personaliza la vista de formulario usando XML

Crea una vista de formulario propia para el objeto **Course**. Los datos mostrados deben ser: el nombre y la descripción del curso.

openacademy/views/openacademy.xml

**<?xml version="1.0" encoding="UTF-8"?>**

**<odoo>**

**<record model="ir.ui.view" id="course\_form\_view">**

**<field name="name">course.form</field>**

**<field name="model">openacademy.course</field>**

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

**<form string="Course Form">**

**<sheet>**

**<group>**

**<field name="name"/>**

**<field name="description"/>**

**</group>**

**</sheet>**

**</form>**

**</field>**

**</record>**

**<!-- window action -->**

**<!--**

**The following tag is an action definition for a "window action",**

**Ejercicio**

Cuaderno de Notas (Notebooks)

En la vista formulario de un curso, poner el campo **description** en una pestaña, de tal manera que será más fácil añadir más tarde otras pestañas que contengan información adicional.

Modifica la vista de formulario de curso como sigue:

openacademy/views/openacademy.xml

**<sheet>**

**<group>**

**<field name="name"/>**

**</group>**

**<notebook>**

**<page string="Description">**

**<field name="description"/>**

**</page>**

**<page string="About">**

**This is an example of notebooks**

**</page>**

**</notebook>**

**</sheet>**

**</form>**

**</field>**

Las vistas de formulario también pueden utilizar HTML plano para diseños más flexibles:

**<form string="Idea Form">**

**<header>**

**<button string="Confirm" type="object" name="action\_confirm"**

**states="draft" class="oe\_highlight" />**

**<button string="Mark as done" type="object" name="action\_done"**

**states="confirmed" class="oe\_highlight"/>**

**<button string="Reset to draft" type="object" name="action\_draft"**

**states="confirmed,done" />**

**<field name="state" widget="statusbar"/>**

**</header>**

**<sheet>**

**<div class="oe\_title">**

**<label for="name" class="oe\_edit\_only" string="Idea Name" />**

**<h1><field name="name" /></h1>**

**</div>**

**<separator string="General" colspan="2" />**

**<group colspan="2" col="2">**

**<field name="description" placeholder="Idea description..." />**

**</group>**

**</sheet>**

**</form>**

### **Vistas de búsqueda**

Las vistas de búsqueda personalizan los campos de búsqueda asociados con la vista de lista (y otras vistas agregadas). Su elemento raíz es **<search>** y están compuestos por los campos definidos, por los cuáles pueden ser buscados:

**<search>**

**<field name="name"/>**

**<field name="inventor\_id"/>**

**</search>**

Si no existe ninguna vista de búsqueda para el modelo, Odoo genera automáticamente una, que permite la búsqueda por el campo **name**.

**Ejercicio**

Buscar Cursos

Permite buscar cursos basados en el título o la descripción.

openacademy/views/openacademy.xml

**</field>**

**</record>**

**<record model="ir.ui.view" id="course\_search\_view">**

**<field name="name">course.search</field>**

**<field name="model">openacademy.course</field>**

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

**<search>**

**<field name="name"/>**

**<field name="description"/>**

**</search>**

**</field>**

**</record>**

**<!-- window action -->**

**<!--**

**The following tag is an action definition for a "window action",**

See video 3

## Relaciones entre modelos

Un registro de un modelo puede estar relacionado con un registro de otro modelo. Por ejemplo, un registro de orden de venta se relaciona con un registro de cliente que contiene los datos del cliente; también está relacionado con los registros de línea de orden de venta.

**Ejercicio**

Crea un modelo **Session**

Para el módulo Open Academy, consideramos un modelo sessions: una sesión es cuando un curso es impartido en un momento dado para un público determinado.

Crea un modelo para sessions. Una sesión tiene un nombre, una fecha de inicio, una duración y un número de asientos. Agrega una acción y un elemento de menú a los mismos. Visualizar el nuevo modelo a través de un menú.

1. Crea la clase Session en **openacademy/models/models.py**.
2. Añade el acceso al objeto session en **openacademy/view/openacademy.xml**.

openacademy/models.py

**name = fields.Char(string="Title", required=True)**

**description = fields.Text()**

**class Session(models.Model):**

**\_name = 'openacademy.session'**

**name = fields.Char(required=True)**

**start\_date = fields.Date()**

**duration = fields.Float(digits=(6, 2), help="Duration in days")**

**seats = fields.Integer(string="Number of seats")**

openacademy/views/openacademy.xml

**<!-- Full id location:**

**action="openacademy.course\_list\_action"**

**It is not required when it is the same module -->**

**<!-- session form view -->**

**<record model="ir.ui.view" id="session\_form\_view">**

**<field name="name">session.form</field>**

**<field name="model">openacademy.session</field>**

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

**<form string="Session Form">**

**<sheet>**

**<group>**

**<field name="name"/>**

**<field name="start\_date"/>**

**<field name="duration"/>**

**<field name="seats"/>**

**</group>**

**</sheet>**

**</form>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="session\_list\_action">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_mode">tree,form</field>**

**</record>**

**<menuitem id="session\_menu" name="Sessions"**

**parent="openacademy\_menu"**

**action="session\_list\_action"/>**

**</odoo>**

**Nota**

**digits=(6, 2)** especifica la precisión de un número flotante: 6 es el número total de dígitos, mientras que 2 es el número de dígitos después de la coma. Ten en cuenta que, en el resultado, los dígitos antes de la coma pueden tener un máximo de 4

### **Campos Relacionales**

Los campos relacionales vinculan registros, ya sea del mismo modelo (jerarquías) o entre diferentes modelos.

Los tipos de campos relacionales son:

[**Many2one(other\_model, ondelete='set null')**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.fields.Many2one)

Un simple enlace a otro objeto:

**print foo.other\_id.name**

**Ver también**

[**foreign keys**](http://www.postgresql.org/docs/9.3/static/tutorial-fk.html) (llaves foráneas)

[**One2many(other\_model, related\_field)**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.fields.One2many)

Una relación virtual, inverso a un [**Many2one**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.fields.Many2one). Un [**One2many**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.fields.One2many) se comporta como contenedor de registros, accediendo a sus resultados en un (posiblemente vacío) conjunto de registros:

**for other in foo.other\_ids:**

**print other.name**

**Peligro**

Debido a que un [**One2many**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.fields.One2many) es una relación virtual, debe haber un campo [**Many2one**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.fields.Many2one) en **other\_model**, y su nombre debe ser **related\_field**

[**Many2many(other\_model)**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.fields.Many2many)

Relación bidireccional múltiple, cualquier registro en un lado puede estar relacionada con cualquier número de registros en el otro lado. Se comporta como un contenedor de registros, al acceder a él también se puede traducir en una lista vacía de registros:

**for other in foo.other\_ids:**

**print other.name**

**Ejercicio**

Relaciones Many2one

Usando un many2one, modificar los modelos Course y Session para reflejar su relación con otros modelos:

* Un curso tiene un usuario responsable (responsible); el valor de ese campo es un registro del modelo integrado **res.users**.
* Una sesión tiene un instructor; el valor de ese campo es un registro del modelo integrado **res.partner**.
* Una sesión está relacionada con un curso (course); el valor de ese campo es un registro del modelo **openacademy.course** y es requerido.
* Adapta las vistas.

1. Añade los campos **Many2one** correspondientes a los modelos, y
2. añádelos en las vistas.

openacademy/models.py

**name = fields.Char(string="Title", required=True)**

**description = fields.Text()**

**responsible\_id = fields.Many2one('res.users',**

**ondelete='set null', string="Responsible", index=True)**

**class Session(models.Model):**

**\_name = 'openacademy.session'**

**start\_date = fields.Date()**

**duration = fields.Float(digits=(6, 2), help="Duration in days")**

**seats = fields.Integer(string="Number of seats")**

**instructor\_id = fields.Many2one('res.partner', string="Instructor")**

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

openacademy/views/openacademy.xml

**<sheet>**

**<group>**

**<field name="name"/>**

**<field name="responsible\_id"/>**

**</group>**

**<notebook>**

**<page string="Description">**

**</field>**

**</record>**

**<!-- override the automatically generated list view for courses -->**

**<record model="ir.ui.view" id="course\_tree\_view">**

**<field name="name">course.tree</field>**

**<field name="model">openacademy.course</field>**

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

**<tree string="Course Tree">**

**<field name="name"/>**

**<field name="responsible\_id"/>**

**</tree>**

**</field>**

**</record>**

**<!-- window action -->**

**<!--**

**The following tag is an action definition for a "window action",**

**<form string="Session Form">**

**<sheet>**

**<group>**

**<group string="General">**

**<field name="course\_id"/>**

**<field name="name"/>**

**<field name="instructor\_id"/>**

**</group>**

**<group string="Schedule">**

**<field name="start\_date"/>**

**<field name="duration"/>**

**<field name="seats"/>**

**</group>**

**</group>**

**</sheet>**

**</form>**

**</field>**

**</record>**

**<!-- session tree/list view -->**

**<record model="ir.ui.view" id="session\_tree\_view">**

**<field name="name">session.tree</field>**

**<field name="model">openacademy.session</field>**

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

**<tree string="Session Tree">**

**<field name="name"/>**

**<field name="course\_id"/>**

**</tree>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="session\_list\_action">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**Ejercicio**

Relaciones One2Many inversas

Utilizando el campo relacional inverso one2many, modifica los modelos para reflejar la relación entre los cursos y sesiones.

1. Modifica la clase **Course**, y
2. añade el campo en la vista de formulario del curso.

openacademy/models.py

**responsible\_id = fields.Many2one('res.users',**

**ondelete='set null', string="Responsible", index=True)**

**session\_ids = fields.One2many(**

**'openacademy.session', 'course\_id', string="Sessions")**

**class Session(models.Model):**

openacademy/views/openacademy.xml

**<page string="Description">**

**<field name="description"/>**

**</page>**

**<page string="Sessions">**

**<field name="session\_ids">**

**<tree string="Registered sessions">**

**<field name="name"/>**

**<field name="instructor\_id"/>**

**</tree>**

**</field>**

**</page>**

**</notebook>**

**</sheet>**

**Ejercicio**

Relaciones Many2Many múltiples

Utilizando el campo relacional Many2Many, modifica el modelo de Session para relacionar cada sesión a un conjunto de asistentes (attendees). Los asistentes estarán representados por los registros partners, por lo que se relacionan con el modelo **res.partner**. Adapta las vistas en consecuencia.

1. Modifica la clase `` Session``, y
2. agrega el campo en la vista de formulario del curso.

openacademy/models.py

**instructor\_id = fields.Many2one('res.partner', string="Instructor")**

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

openacademy/views/openacademy.xml

**<field name="seats"/>**

**</group>**

**</group>**

**<label for="attendee\_ids"/>**

**<field name="attendee\_ids"/>**

**</sheet>**

**</form>**

**</field>**

Ver video 4

## Herencia

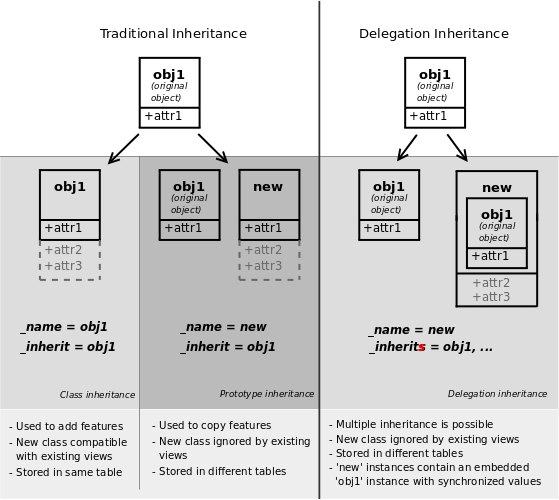
### **Herencia de modelos**

Odoo proporciona dos mecanismos de herencia para extender un modelo existente de forma modular.

El primer mecanismo de herencia permite a un módulo modificar el comportamiento de un modelo definido en otro módulo:

* agrega campos a un modelo,
* sobrescribir la definición de campos en un modelo,
* añade restricciones a un modelo,
* añade métodos a un modelo,
* reemplazar los métodos existentes en un modelo.

El segundo mecanismo de herencia (delegación) permite vincular cada registro de un modelo a un registro en un modelo padre, y proporciona un acceso transparente a los campos del registro padre.



**Ver también**

* [**\_inherit**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.models.Model._inherit)
* [**\_inherits**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.models.Model._inherits)

### **Herencia de vistas**

En lugar de modificar las vistas existentes en el lugar (sobrescribiéndolos), Odoo proporciona herencia de vistas, donde las vistas hijas **extensión** se aplican en la parte superior de la vista de la raíz, y se puede añadir o eliminar contenido de sus padres.

Una vista de extensión hace referencia a su padre con el campo **inherit\_id**, y en lugar de una vista el campo **arch** se compone de cualquier número de **xpath**, elementos que seleccionan y modifican el contenido de su vista padre:

**<!-- improved idea categories list -->**

**<record id="idea\_category\_list2" model="ir.ui.view">**

**<field name="name">id.category.list2</field>**

**<field name="model">idea.category</field>**

**<field name="inherit\_id" ref="id\_category\_list"/>**

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

**<!-- find field description and add the field**

**idea\_ids after it -->**

**<xpath expr="//field[@name='description']" position="after">**

**<field name="idea\_ids" string="Number of ideas"/>**

**</xpath>**

**</field>**

**</record>**

**expr**

Una expresión [XPath](http://w3.org/TR/xpath) es la selección de un solo elemento en la vista padre. Genera un error si no coincide con ningún elemento o más de uno

**position**

Operación para aplicar al elemento coincidente:

**inside**

anexa el cuerpo de **xpath** en el extremo del elemento emparejado

**replace**

reemplaza el elemento que empate con el cuerpo del **xpath**, reemplazando cualquier ocurrencia de nodo **$0** en el nuevo cuerpo con el elemento original

**before**

inserta el cuerpo del **xpath** como un hermano antes del elemento emparejado

**after**

inserta el cuerpo de **xpaths** como un hermano después del elemento emparejado

**attributes**

altera los atributos del elemento coincidente utilizando elementos **attribute** especiales en el cuerpo de **xpath**

**Truco**

En la concordancia de un solo elemento, el atributo **position** se puede establecer directamente sobre el elemento que se encuentra. Ambas herencias a continuación le dará el mismo resultado.

**<xpath expr="//field[@name='description']" position="after">**

**<field name="idea\_ids" />**

**</xpath>**

**<field name="description" position="after">**

**<field name="idea\_ids" />**

**</field>**

**Ejercicio**

Alterar el contenido existente

* Utilizando el modelo de herencia, modifica el modelo Partner y añade un campo boolean llamado **Instructor** y un campo many2many que corresponde a la relación de session-partner
* Mediante herencia de vista, muestra este campo en la vista formulario de partner

**Nota**

Esta es la oportunidad para presentar el modo desarrollador para inspeccionar la vista, conocer su ID externo y el lugar para poner el nuevo campo.

1. Crea un archivo **openacademy/models/partner.py** e impórtalo en **\_\_init\_\_.py**
2. Crea un archivo **openacademy/views/partner.xml** y agrégalo al **\_\_manifest\_\_.py**

openacademy/\_\_init\_\_.py

**# -\*- coding: utf-8 -\*-**

**from . import controllers**

**from . import models**

**from . import partner**

openacademy/\_\_manifest\_\_.py

**# 'security/ir.model.access.csv',**

**'templates.xml',**

**'views/openacademy.xml',**

**'views/partner.xml',**

**],**

**# only loaded in demonstration mode**

**'demo': [**

openacademy/partner.py

**# -\*- coding: utf-8 -\*-**

**from odoo import fields, models**

**class Partner(models.Model):**

**\_inherit = 'res.partner'**

**# Add a new column to the res.partner model, by default partners are not**

**# instructors**

**instructor = fields.Boolean("Instructor", default=False)**

**session\_ids = fields.Many2many('openacademy.session',**

**string="Attended Sessions", readonly=True)**

openacademy/views/partner.xml

**<?xml version="1.0" encoding="UTF-8"?>**

**<odoo>**

**<!-- Add instructor field to existing view -->**

**<record model="ir.ui.view" id="partner\_instructor\_form\_view">**

**<field name="name">partner.instructor</field>**

**<field name="model">res.partner</field>**

**<field name="inherit\_id" ref="base.view\_partner\_form"/>**

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

**<notebook position="inside">**

**<page string="Sessions">**

**<group>**

**<field name="instructor"/>**

**<field name="session\_ids"/>**

**</group>**

**</page>**

**</notebook>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="contact\_list\_action">**

**<field name="name">Contacts</field>**

**<field name="res\_model">res.partner</field>**

**<field name="view\_mode">tree,form</field>**

**</record>**

**<menuitem id="configuration\_menu" name="Configuration"**

**parent="main\_openacademy\_menu"/>**

**<menuitem id="contact\_menu" name="Contacts"**

**parent="configuration\_menu"**

**action="contact\_list\_action"/>**

**</odoo>**

#### **Dominios**

En Odoo, [Domains](http://vauxoo.github.io/odoo/reference/orm.html#reference-orm-domains) (dominios) son valores que representan condiciones en los registros. Un dominio es una lista de criterios utilizados para seleccionar un subconjunto de registros de un modelo. Cada criterio es una tripleta con un nombre de campo, un operador y un valor.

Por ejemplo, cuando se usa en el modelo product el siguiente dominio selecciona todos los services con un precio unitario mayor 1000:

**[('product\_type', '=', 'service'), ('unit\_price', '>', 1000)]**

Por defecto se combinan criterios con un AND implícito. Los operadores lógicos **&** (AND), **|** (OR) and **!** (NOT) se pueden utilizar para combinar explícitamente criterios. Se utilizan en posición de prefijo (el operador se inserta antes de sus argumentos, y no entre). Por ejemplo para seleccionar productos "que son servicios OR tienen un precio unitario que NOT está entre 1000 y 2000":

**['|',**

**('product\_type', '=', 'service'),**

**'!', '&',**

**('unit\_price', '>=', 1000),**

**('unit\_price', '<', 2000)]**

Un parámetro **dominio** puede ser añadido a campos relacionales para limitar registros válidos para esa relación cuando se intenta seleccionar registros en la interfaz del cliente.

**Ejercicio**

Dominios en campos relacionales

Al seleccionar el instructor para una Session, sólo instructores (partners cuyo campo **instructor** es **True**) deben ser visibles.

openacademy/models.py

**duration = fields.Float(digits=(6, 2), help="Duration in days")**

**seats = fields.Integer(string="Number of seats")**

**instructor\_id = fields.Many2one('res.partner', string="Instructor",**

**domain=[('instructor', '=', True)])**

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**Nota**

Un dominio declarado como una lista literal se evalúa del lado del servidor y no puede hacer referencia a valores dinámicos en el lado derecho en su declaración, un dominio declarado como una cadena es evaluado del lado del cliente y permite nombres de campo en el lado derecho

**Ejercicio**

Dominios más complejos

Crea un par de nuevas categorías Teacher / Level 1 y Teacher / Level 2. El instructor de una sesión puede ser un instructor o un profesor (de cualquier nivel).

1. Modifica el dominio del modelo Session
2. Modifica **openacademy/view/partner.xml** para obtener acceso a las categorías que acabamos de crear:

openacademy/models.py

**seats = fields.Integer(string="Number of seats")**

**instructor\_id = fields.Many2one('res.partner', string="Instructor",**

**domain=['|', ('instructor', '=', True),**

**('category\_id.name', 'ilike', "Teacher")])**

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

openacademy/views/partner.xml

**<menuitem id="contact\_menu" name="Contacts"**

**parent="configuration\_menu"**

**action="contact\_list\_action"/>**

**<record model="ir.actions.act\_window" id="contact\_cat\_list\_action">**

**<field name="name">Contact Tags</field>**

**<field name="res\_model">res.partner.category</field>**

**<field name="view\_mode">tree,form</field>**

**</record>**

**<menuitem id="contact\_cat\_menu" name="Contact Tags"**

**parent="configuration\_menu"**

**action="contact\_cat\_list\_action"/>**

**<record model="res.partner.category" id="teacher1">**

**<field name="name">Teacher / Level 1</field>**

**</record>**

**<record model="res.partner.category" id="teacher2">**

**<field name="name">Teacher / Level 2</field>**

**</record>**

**</odoo>**

Watch video 5

## Campos y valores calculados

Hasta ahora los campos han sido almacenados y obtenidos directamente de la base de datos. Los campos también pueden ser calculados. En ese caso, el valor del campo no es obtenido de la base de datos sino calculado al vuelo llamando a un método en el modelo.

Para crear un campo calculado, crea un campo y establece su atributo: **compute** con el nombre del método. El método de cómputo simplemente debe establecer el valor del campo calculado en cada registro en **self**.

**Peligro**

**self** es una (collection) colección

El objeto **self** es un recordset, es decir, una colección ordenada de registros. Es compatible con las operaciones estándar de Python en colecciones, como **len(self)** e **iter(self)**, además de operaciones extra set como **recs1 + recs2**.

Iterando sobre **self** los registros uno por uno, donde cada registro es una colección de tamaño 1. Puedes acceder/asignar campos a registros individuales mediante el uso de la notación de puntos, algo así **record.name**.

**import random**

**from odoo import models, fields, api**

**class ComputedModel(models.Model):**

**\_name = 'test.computed'**

**name = fields.Char(compute='\_compute\_name')**

**@api.multi**

**def \_compute\_name(self):**

**for record in self:**

**record.name = str(random.randint(1, 1e6))**

### **Dependencias**

El valor de un campo calculado generalmente depende de los valores de otros campos en el registro calculado. El ORM espera que el programador especifique estas dependencias en el método de cálculo con el decorador [**depends()**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.api.depends). Las dependencias dadas son utilizadas por el ORM para desencadenar el recálculo del campo cada vez que se han modificado algunas de sus dependencias:

**from odoo import models, fields, api**

**class ComputedModel(models.Model):**

**\_name = 'test.computed'**

**name = fields.Char(compute='\_compute\_name')**

**value = fields.Integer()**

**@api.depends('value')**

**def \_compute\_name(self):**

**for record in self:**

**record.name = "Record with value %s" % record.value**

**Ejercicio**

Campos y valores calculados

* Agrega el porcentaje de asientos ocupados para el modelo session
* Muestra ese campo en la vista de árbol y formulario
* Muestra el campo como una barra de progreso

1. Agrega un campo calculado a una Session
2. Muestra el campo en la vista Session:

openacademy/models.py

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**taken\_seats = fields.Float(string="Taken seats", compute='\_taken\_seats')**

**@api.depends('seats', 'attendee\_ids')**

**def \_taken\_seats(self):**

**for r in self:**

**if not r.seats:**

**r.taken\_seats = 0.0**

**else:**

**r.taken\_seats = 100.0 \* len(r.attendee\_ids) / r.seats**

openacademy/views/openacademy.xml

**<field name="start\_date"/>**

**<field name="duration"/>**

**<field name="seats"/>**

**<field name="taken\_seats" widget="progressbar"/>**

**</group>**

**</group>**

**<label for="attendee\_ids"/>**

**<tree string="Session Tree">**

**<field name="name"/>**

**<field name="course\_id"/>**

**<field name="taken\_seats" widget="progressbar"/>**

**</tree>**

**</field>**

**</record>**

### **Valores por defecto**

Cualquier campo puede tener un valor por defecto. En la definición de campo, se agrega la opción **default=X** donde **X** es un valor literal Python (boolean, integer, float, string), o una función que toma un conjunto de registros y devuelve un valor:

**name = fields.Char(default="Unknown")**

**user\_id = fields.Many2one('res.users', default=lambda self: self.env.user)**

**Nota**

El objeto **self.env** da acceso a solicitar parámetros y otras cosas útiles:

* **self.env.cr** o **self.\_cr** es el objeto cursor de la base de datos; se utiliza para consultar la base de datos
* **self.env.uid** o **self.\_uid** es el id de la base de datos del usuario actual
* **self.env.user** es el registro del usuario actual
* **self.env.context** o **self.\_context** es el Diccionario de contexto
* **self.env.ref(xml\_id)** devuelve el registro correspondiente a un id XML
* **self.env[model\_name]** devuelve una instancia del modelo dado

**Ejercicio**

Objetos activos – valores por defecto

* Define el valor por defecto de start\_date como hoy (ver [**Date**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.fields.Date)).
* Agrega un campo **active** en la clase de Session y has las sesiones activas por defecto.

openacademy/models.py

**\_name = 'openacademy.session'**

**name = fields.Char(required=True)**

**start\_date = fields.Date(default=fields.Date.today)**

**duration = fields.Float(digits=(6, 2), help="Duration in days")**

**seats = fields.Integer(string="Number of seats")**

**active = fields.Boolean(default=True)**

**instructor\_id = fields.Many2one('res.partner', string="Instructor",**

**domain=['|', ('instructor', '=', True),**

openacademy/views/openacademy.xml

**<field name="course\_id"/>**

**<field name="name"/>**

**<field name="instructor\_id"/>**

**<field name="active"/>**

**</group>**

**<group string="Schedule">**

**<field name="start\_date"/>**

**Nota**

Odoo tiene reglas incorporadas que hacen que registros con el campo **active** establecido a **False** sean invisibles.

Watch Video 6

## Onchange

El mecanismo "onchange" proporciona una forma para que la interfaz del cliente actualice los datos de un formulario cuando el usuario ha llenado un valor en un campo, sin guardar nada en la base de datos.

Por ejemplo, supóngase que un modelo tiene tres campos **amount**, **unit\_price** y **price**, y se desea actualizar el precio en el formulario cuando cualquiera de los otros campos es modificado. Para ello, se define un método donde **self** representa el registro en la vista formulario y se decora con [**onchange()**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.api.onchange) para especificar en qué campo se activará. Cualquier cambio que se realice en **self** será reflejado en el formulario.

**<!-- content of form view -->**

**<field name="amount"/>**

**<field name="unit\_price"/>**

**<field name="price" readonly="1"/>**

**# onchange handler**

**@api.onchange('amount', 'unit\_price')**

**def \_onchange\_price(self):**

**# set auto-changing field**

**self.price = self.amount \* self.unit\_price**

**# Can optionally return a warning and domains**

**return {**

**'warning': {**

**'title': "Something bad happened",**

**'message': "It was very bad indeed",**

**}**

**}**

Para campos calculados, el comportamiento **onchange** está incorporado como se aprecia trabajando con el formulario de una sesión (Session): cambia el número de asientos o los participantes, y se actualiza automáticamente la barra de progreso de **taken\_seats**.

**Ejercicio**

Cuidado

Añade un onchange explícito para advertir sobre valores no válidos, como un número negativo de asientos o más participantes que asientos.

openacademy/models.py

**r.taken\_seats = 0.0**

**else:**

**r.taken\_seats = 100.0 \* len(r.attendee\_ids) / r.seats**

**@api.onchange('seats', 'attendee\_ids')**

**def \_verify\_valid\_seats(self):**

**if self.seats < 0:**

**return {**

**'warning': {**

**'title': "Incorrect 'seats' value",**

**'message': "The number of available seats may not be negative",**

**},**

**}**

**if self.seats < len(self.attendee\_ids):**

**return {**

**'warning': {**

**'title': "Too many attendees",**

**'message': "Increase seats or remove excess attendees",**

**},**

**}**

## Restricciones

Odoo ofrece dos formas de configurar automáticamente invariantes: [**Python constraints**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.api.constrains) y [**SQL constraints**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.models.Model._sql_constraints).

Una restricción de Python se define como un método decorado con [**constrains()**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.api.constrains) y es invocado en un recordset. El decorador especifica los campos que están involucrados en la restricción, por lo que la restricción es automáticamente evaluada cuando uno de ellos es modificado. El método levantará una excepción si no se satisface su invariante:

**from odoo.exceptions import ValidationError**

**@api.constrains('age')**

**def \_check\_something(self):**

**for record in self:**

**if record.age > 20:**

**raise ValidationError("Your record is too old: %s" % record.age)**

**# all records passed the test, don't return anything**

**Ejercicio**

Agrega restricciones de Python

Agrega una restricción que comprueba que el instructor no está presente en los asistentes de su propia sesión.

openacademy/models.py

**# -\*- coding: utf-8 -\*-**

**from odoo import models, fields, api, exceptions**

**class Course(models.Model):**

**\_name = 'openacademy.course'**

**'message': "Increase seats or remove excess attendees",**

**},**

**}**

**@api.constrains('instructor\_id', 'attendee\_ids')**

**def \_check\_instructor\_not\_in\_attendees(self):**

**for r in self:**

**if r.instructor\_id and r.instructor\_id in r.attendee\_ids:**

**raise exceptions.ValidationError("A session's instructor can't be an attendee")**

Las restricciones SQL se definen mediante el atributo modelo: [**\_sql\_constraints**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.models.Model._sql_constraints). Este último se le asigna a una lista de tripletas de cadenas **(name, sql\_definition, message)`, donde ``name** es un nombre de restricción válido de SQL, **sql\_definition** es una expresión de [table\_constraint](http://www.postgresql.org/docs/9.3/static/ddl-constraints.html), y **message** es el mensaje de error.

**Ejercicio**

Agrega restricciones SQL

Con la ayuda de la [**Documentación de PostgreSQL**](http://www.postgresql.org/docs/9.3/static/ddl-constraints.html), añade las siguientes restricciones:

1. Comprueba que la descripción del curso y el título del curso son diferentes
2. Haz el nombre del Curso ÚNICO

openacademy/models.py

**session\_ids = fields.One2many(**

**'openacademy.session', 'course\_id', string="Sessions")**

**\_sql\_constraints = [**

**('name\_description\_check',**

**'CHECK(name != description)',**

**"The title of the course should not be the description"),**

**('name\_unique',**

**'UNIQUE(name)',**

**"The course title must be unique"),**

**]**

**class Session(models.Model):**

**\_name = 'openacademy.session'**

**Ejercicio**

Ejercicio 6 - Añade una opción para duplicar

Puesto que hemos añadido una restricción para la unicidad del nombre de curso, no es posible utilizar la función **duplicate** (**Form ‣ Duplicate**).

Reimplementa tu propio método de "copy" que permita duplicar el objeto del curso, cambiando el nombre original por "Copy of [original name]".

openacademy/models.py

**session\_ids = fields.One2many(**

**'openacademy.session', 'course\_id', string="Sessions")**

**@api.multi**

**def copy(self, default=None):**

**default = dict(default or {})**

**copied\_count = self.search\_count(**

**[('name', '=like', u"Copy of {}%".format(self.name))])**

**if not copied\_count:**

**new\_name = u"Copy of {}".format(self.name)**

**else:**

**new\_name = u"Copy of {} ({})".format(self.name, copied\_count)**

**default['name'] = new\_name**

**return super(Course, self).copy(default)**

**\_sql\_constraints = [**

**('name\_description\_check',**

**'CHECK(name != description)',**

Watch Video 7

Vistas Avanzadas

Vistas de árbol

Las vistas de árbol pueden tener atributos suplementarios para personalizar su comportamiento:

decoration-{$name}

permite cambiar el estilo de texto de una fila basada en atributos del registro en cuestión.

Los valores son expresiones Python. Para cada registro, la expresión se evalúa con los atributos del registro como valores del contexto y si es true, se aplica el estilo correspondiente a la fila. Otros valores de contexto son uid (el id del usuario actual) y current\_date (la fecha actual como una cadena de la forma yyyy-MM-dd).

{$name} puede ser bf (font-weight: bold), it (font-style: italic), o cualquier color contextual de bootstrap (danger, info, muted, primary, success or warning).

<tree string="Idea Categories" decoration-info="state=='draft'"

decoration-danger="state=='trashed'">

<field name="name"/>

<field name="state"/>

</tree>

editable

Ya sea "top" o "bottom". Hace la vista de árbol modificable (en lugar de tener que pasar por la vista formulario), el valor representa la posición donde aparecen nuevas filas bien sea arriba o abajo por su significado en inglés.

Ejercicio

Colores en la Lista

Modifica la vista de árbol de una sesión de tal manera que las sesiones que duran menos de 5 días sean de color azul, y los que duran más de 15 días sean de color rojo.

Modifica la vista de árbol de sesión:

openacademy/views/openacademy.xml

<field name="name">session.tree</field>

<field name="model">openacademy.session</field>

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

<tree string="Session Tree" decoration-info="duration&lt;5" decoration-danger="duration&gt;15">

<field name="name"/>

<field name="course\_id"/>

<field name="duration" invisible="1"/>

<field name="taken\_seats" widget="progressbar"/>

</tree>

</field>

Calendarios

Muestra los registros como calendario de eventos. Su elemento raíz es <calendar> y sus atributos más comunes son:

color

El nombre del campo utilizado para colorear. Los colores se distribuyen automáticamente a los eventos y acontecimientos en el mismo segmento de color (los registros que tienen el mismo valor para su campo @color) se les dará el mismo color.

date\_start

campo del registro con la fecha de inicio del evento

date\_stop (opcional)

campo de registro con la fecha final del evento

string

campo del registro para definir la etiqueta de cada evento del calendario

<calendar string="Ideas" date\_start="invent\_date" color="inventor\_id">

<field name="name"/>

</calendar>

Ejercicio

Vista de calendario

Agrega una vista de calendario al modelo Session que permita visualizar los eventos asociados a la OpenAcademy.

Agrega un campo end\_date calculado a partir de start\_date y duration

Truco

la función inversa hace que el campo se pueda escribir y permite mover las sesiones (a través de arrastrar y soltar) en la vista Calendario

Agrega una vista de calendario al modelo Session

Y añade la vista de calendario a la acción del modelo Session

openacademy/models.py

# -\*- coding: utf-8 -\*-

from datetime import timedelta

from odoo import models, fields, api, exceptions

class Course(models.Model):

attendee\_ids = fields.Many2many('res.partner', string="Attendees")

taken\_seats = fields.Float(string="Taken seats", compute='\_taken\_seats')

end\_date = fields.Date(string="End Date", store=True,

compute='\_get\_end\_date', inverse='\_set\_end\_date')

@api.depends('seats', 'attendee\_ids')

def \_taken\_seats(self):

},

}

@api.depends('start\_date', 'duration')

def \_get\_end\_date(self):

for r in self:

if not (r.start\_date and r.duration):

r.end\_date = r.start\_date

continue

# Add duration to start\_date, but: Monday + 5 days = Saturday, so

# subtract one second to get on Friday instead

start = fields.Datetime.from\_string(r.start\_date)

duration = timedelta(days=r.duration, seconds=-1)

r.end\_date = start + duration

def \_set\_end\_date(self):

for r in self:

if not (r.start\_date and r.end\_date):

continue

# Compute the difference between dates, but: Friday - Monday = 4 days,

# so add one day to get 5 days instead

start\_date = fields.Datetime.from\_string(r.start\_date)

end\_date = fields.Datetime.from\_string(r.end\_date)

r.duration = (end\_date - start\_date).days + 1

@api.constrains('instructor\_id', 'attendee\_ids')

def \_check\_instructor\_not\_in\_attendees(self):

for r in self:

openacademy/views/openacademy.xml

</field>

</record>

<!-- calendar view -->

<record model="ir.ui.view" id="session\_calendar\_view">

<field name="name">session.calendar</field>

<field name="model">openacademy.session</field>

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

<calendar string="Session Calendar" date\_start="start\_date" date\_stop="end\_date" color="instructor\_id">

<field name="name"/>

</calendar>

</field>

</record>

<record model="ir.actions.act\_window" id="session\_list\_action">

<field name="name">Sessions</field>

<field name="res\_model">openacademy.session</field>

<field name="view\_mode">tree,form,calendar</field>

</record>

<menuitem id="session\_menu" name="Sessions"

Vistas de búsqueda

En la vista de búsqueda los elementos <field> pueden tener un @filter\_domain que reemplaza el dominio generado para la búsqueda en el campo de búsqueda. En el dominio dado, self representa el valor introducido por el usuario. En el siguiente ejemplo, se utiliza para buscar en campos name y description.

Las vistas de búsqueda también pueden contener elementos <filter>, que actúan como alternativa a búsquedas predefinidas. Los filtros deben tener uno de los siguientes atributos:

domain

añade el dominio dado a la búsqueda actual

context

añade algún contexto para la búsqueda actual; Utiliza la clave group\_by para agrupar los resultados por el campo nombre

<search string="Ideas">

<field name="name"/>

<field name="description" string="Name and description"

filter\_domain="['|', ('name', 'ilike', self), ('description', 'ilike', self)]"/>

<field name="inventor\_id"/>

<field name="country\_id" widget="selection"/>

<filter name="my\_ideas" string="My Ideas"

domain="[('inventor\_id', '=', uid)]"/>

<group string="Group By">

<filter name="group\_by\_inventor" string="Inventor"

context="{'group\_by': 'inventor\_id'}"/>

</group>

</search>

Para utilizar una vista de búsqueda no predeterminada en una acción, debe vincularse con el campo de search\_view\_id del registro de la acción.

La acción también puede establecer valores predeterminados para los campos de búsqueda a través de su campo context: las claves del contexto deben tener la forma search\_default\_field\_name la cual inicializará field\_name con el valor proporcionado. Los filtros de búsqueda deben tener un @name opcional para tener un valor por defecto y comportarse como datos lógicos (booleans) (que pueden sólo ser activadas por defecto).

Ejercicio

Vistas de búsqueda

Añade un botón en la vista de búsqueda para filtrar los cursos de tal forma que encuentre solo los registros donde el usuario actual sea el responsable del curso. Y que esté seleccionada por defecto.

Agrega un botón para agrupar los cursos por el usuario responsable.

openacademy/views/openacademy.xml

<search>

<field name="name"/>

<field name="description"/>

<filter name="my\_courses" string="My Courses"

domain="[('responsible\_id', '=', uid)]"/>

<group string="Group By">

<filter name="by\_responsible" string="Responsible"

context="{'group\_by': 'responsible\_id'}"/>

</group>

</search>

</field>

</record>

<field name="res\_model">openacademy.course</field>

<field name="view\_mode">tree,form</field>

<field name="context" eval="{'search\_default\_my\_courses': 1}"/>

<field name="help" type="html">

<p class="oe\_view\_nocontent\_create">Create the first course

</p>

Gantt

Advertencia

La vista gantt requiere del módulo web\_gantt, que está disponible en la versión enterprise edition.

Gráficos de barras horizontales normalmente se utilizan para mostrar la planificación del proyecto y su avance, su elemento raíz es <gantt>.

<gantt string="Ideas"

date\_start="invent\_date"

date\_stop="date\_finished"

progress="progress"

default\_group\_by="inventor\_id" />

Ejercicio

Diagramas de Gantt

Agrega un diagrama de Gantt que permita al usuario ver la programación de las sesiones relacionados con el módulo OpenAcademy. Las sesiones deben agruparse por instructor.

Crea un campo calculado expresando la duración de la sesión en horas

Agrega la definición de la vista de gantt y agrega la vista de gantt a la acción del modelo Session

openacademy/models.py

end\_date = fields.Date(string="End Date", store=True,

compute='\_get\_end\_date', inverse='\_set\_end\_date')

hours = fields.Float(string="Duration in hours",

compute='\_get\_hours', inverse='\_set\_hours')

@api.depends('seats', 'attendee\_ids')

def \_taken\_seats(self):

for r in self:

end\_date = fields.Datetime.from\_string(r.end\_date)

r.duration = (end\_date - start\_date).days + 1

@api.depends('duration')

def \_get\_hours(self):

for r in self:

r.hours = r.duration \* 24

def \_set\_hours(self):

for r in self:

r.duration = r.hours / 24

@api.constrains('instructor\_id', 'attendee\_ids')

def \_check\_instructor\_not\_in\_attendees(self):

for r in self:

openacademy/views/openacademy.xml

<record model="ir.ui.view" id="session\_gantt\_view">

<field name="name">session.gantt</field>

<field name="model">openacademy.session</field>

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

<gantt string="Session Gantt" color="course\_id"

date\_start="start\_date"

default\_group\_by='instructor\_id' date\_stop="end\_date">

<field name="name"/>

</gantt>

</field>

</record>

<record model="ir.actions.act\_window" id="session\_list\_action">

<field name="name">Sessions</field>

<field name="res\_model">openacademy.session</field>

<field name="view\_mode">tree,form,calendar,gantt</field>

</record>

<menuitem id="session\_menu" name="Sessions"

parent="openacademy\_menu"

action="session\_list\_action"/>

Vistas de Gráfico

Las vistas de gráfico permiten agregar resumen y análisis de modelos, su elemento raíz es <graph>.

Nota

Vista Pivote (elemento raíz <pivot>) es una tabla multidimensional, permite la selección de dimensiones para obtener el conjunto de datos adecuado para presentar los datos de forma más gráfica. La vista pivote comparte la misma definición contenida como vistas gráfico.

Vistas de gráfico tienen 4 modos de visualización, se selecciona el modo predeterminado mediante el atributo @type.

Bar (por defecto)

un gráfico de barras, la primera dimensión se utiliza para definir los grupos en el eje horizontal, las otras dimensiones definen las barras agregadas dentro de cada grupo.

Por defecto las barras están lado a lado, éstas pueden apilarse utilizando @stacked="True" en el <graph>

Line

Gráfica de Líneas XY

Pie

Gráfica de pastel de 2 dimensiones

Las vistas de gráfico contienen <field> con un atributo obligatorio @type tomando los valores:

row (default)

el campo debe agregarse por defecto

measure

el campo debe ser agregado en lugar de agrupar por él

<graph string="Total idea score by Inventor">

<field name="inventor\_id"/>

<field name="score" type="measure"/>

</graph>

Advertencia

Las vistas de gráfico suma valores de la base de datos, no funcionan con campos calculados que no estén almacenados.

Ejercicio

Vistas en Gráfico

Agrega una vista de gráfico en el objeto Session que muestre, para cada curso, el número de participantes bajo la forma de un gráfico de barras.

Agrega el número de asistentes como un campo calculado almacenado

Luego añade la vista relevante

openacademy/models.py

hours = fields.Float(string="Duration in hours",

compute='\_get\_hours', inverse='\_set\_hours')

attendees\_count = fields.Integer(

string="Attendees count", compute='\_get\_attendees\_count', store=True)

@api.depends('seats', 'attendee\_ids')

def \_taken\_seats(self):

for r in self:

for r in self:

r.duration = r.hours / 24

@api.depends('attendee\_ids')

def \_get\_attendees\_count(self):

for r in self:

r.attendees\_count = len(r.attendee\_ids)

@api.constrains('instructor\_id', 'attendee\_ids')

def \_check\_instructor\_not\_in\_attendees(self):

for r in self:

openacademy/views/openacademy.xml

<record model="ir.ui.view" id="openacademy\_session\_graph\_view">

<field name="name">openacademy.session.graph</field>

<field name="model">openacademy.session</field>

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

<graph string="Participations by Courses">

<field name="course\_id"/>

<field name="attendees\_count" type="measure"/>

</graph>

</field>

</record>

<record model="ir.actions.act\_window" id="session\_list\_action">

<field name="name">Sessions</field>

<field name="res\_model">openacademy.session</field>

<field name="view\_mode">tree,form,calendar,gantt,graph</field>

</record>

<menuitem id="session\_menu" name="Sessions"

parent="openacademy\_menu"

action="session\_list\_action"/>

Kanban

Usada para organizar tareas, procesos de producción, etc… su elemento raíz es <kanban>.

Una vista de kanban muestra un conjunto de tarjetas que pueden ser agrupadas en columnas. Cada tarjeta representa un registro y cada columna los valores de un campo de agregación.

Por ejemplo, se organizan las tareas del proyecto por etapa (cada columna es una etapa), o por responsable (cada columna es un usuario), y así sucesivamente.

Las vistas Kanban definen la estructura de cada tarjeta como una mezcla de elementos de un formulario (incluyendo HTML básico) y QWeb.

Ejercicio

Vista Kanban

Agrega una vista de Kanban que muestre sesiones agrupadas por curso (las columnas son cursos).

Agrega un campo entero de color al modelo Session

Añade la opción de kanban y actualiza la acción

openacademy/models.py

duration = fields.Float(digits=(6, 2), help="Duration in days")

seats = fields.Integer(string="Number of seats")

active = fields.Boolean(default=True)

color = fields.Integer()

instructor\_id = fields.Many2one('res.partner', string="Instructor",

domain=['|', ('instructor', '=', True),

openacademy/views/openacademy.xml

<record model="ir.ui.view" id="view\_openacad\_session\_kanban">

<field name="name">openacad.session.kanban</field>

<field name="model">openacademy.session</field>

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

<kanban default\_group\_by="course\_id">

<field name="color"/>

<templates>

<t t-name="kanban-box">

<div

t-attf-class="oe\_kanban\_color\_{{kanban\_getcolor(record.color.raw\_value)}}

oe\_kanban\_global\_click\_edit oe\_semantic\_html\_override

oe\_kanban\_card {{record.group\_fancy==1 ? 'oe\_kanban\_card\_fancy' : ''}}">

<div class="oe\_dropdown\_kanban">

<!-- dropdown menu -->

<div class="oe\_dropdown\_toggle">

<i class="fa fa-bars fa-lg"/>

<ul class="oe\_dropdown\_menu">

<li>

<a type="delete">Delete</a>

</li>

<li>

<ul class="oe\_kanban\_colorpicker"

data-field="color"/>

</li>

</ul>

</div>

<div class="oe\_clear"></div>

</div>

<div t-attf-class="oe\_kanban\_content">

<!-- title -->

Session name:

<field name="name"/>

<br/>

Start date:

<field name="start\_date"/>

<br/>

duration:

<field name="duration"/>

</div>

</div>

</t>

</templates>

</kanban>

</field>

</record>

<record model="ir.actions.act\_window" id="session\_list\_action">

<field name="name">Sessions</field>

<field name="res\_model">openacademy.session</field>

<field name="view\_mode">tree,form,calendar,gantt,graph,kanban</field>

</record>

<menuitem id="session\_menu" name="Sessions"

parent="openacademy\_menu"

action="session\_list\_action"/>

Watch video 8

## Seguridad

Los mecanismos de control de acceso deben ser configurados para lograr una política coherente de seguridad.

### **Mecanismos de control de acceso basado en grupos**

Los grupos se crean como registros convencionales del modelo **res.groups** y garantizan el acceso a través de las definiciones de permisos en los menús. Sin embargo incluso si un objeto no tiene menú, éstos se les puede acceder indirectamente, entonces las reglas CRUD (leer, escribir, crear, borrar) deben definirse para los grupos a nivel de objetos. Estas reglas se insertan generalmente a través de archivos CSV dentro de los módulos. Es posible restringir el acceso a campos específicos de una vista o a un objeto utilizando el atributo **groups** del campo.

### **Derechos de acceso**

Los derechos de acceso se definen como registros del modelo **ir.model.access**. Cada derecho de acceso está asociado a un modelo, un grupo (o no grupo para el caso de un acceso global) y un conjunto de permisos: leer, escribir, crear, borrar. Estos derechos de acceso son creados generalmente por un archivo CSV con el nombre de su modelo: **ir.model.access.csv**.

**id,name,model\_id/id,group\_id/id,perm\_read,perm\_write,perm\_create,perm\_unlink**

**access\_idea\_idea,idea.idea,model\_idea\_idea,base.group\_user,1,1,1,0**

**access\_idea\_vote,idea.vote,model\_idea\_vote,base.group\_user,1,1,1,0**

**Ejercicio**

Añade control de acceso mediante la interfaz de Odoo

Crea un nuevo usuario “John Smith”. Crea un grupo "OpenAcademy / Session Read" con acceso de lectura al modelo Session.

1. Crea un nuevo usuario John Smith a través de **Settings ‣ Users ‣ Users**
2. Crea un nuevo grupo llamado **session\_read** a través del menú **Settings ‣ Users ‣ Groups**, que debe tener acceso de lectura sobre el modelo Session
3. Edita John Smith para que sea un miembro de **session\_read**
4. Inicie la sesión como John Smith para comprobar que los derechos de acceso son correctos

**Ejercicio**

Añade el control de acceso a través de los archivos de datos en el módulo

Usando archivos de datos,

* Crea un grupo OpenAcademy / Manager con pleno acceso a todos los modelos OpenAcademy
* Haz Session y Course con acceso de lectura por todos los usuarios

1. Crea un nuevo archivo **openacademy/security/security.xml** que tenga el grupo OpenAcademy Manager en él
2. Edita el archivo **openacademy/security/ir.model.access.csv** con los derechos de acceso a los modelos
3. Por último, actualiza **openacademy/\_\_manifest\_\_.py** para añadir los nuevos archivos de datos en él

openacademy/\_\_manifest\_\_.py

**# always loaded**

**'data': [**

**'security/security.xml',**

**'security/ir.model.access.csv',**

**'templates.xml',**

**'views/openacademy.xml',**

**'views/partner.xml',**

openacademy/security/ir.model.access.csv

**id,name,model\_id/id,group\_id/id,perm\_read,perm\_write,perm\_create,perm\_unlink**

**course\_manager,course manager,model\_openacademy\_course,group\_manager,1,1,1,1**

**session\_manager,session manager,model\_openacademy\_session,group\_manager,1,1,1,1**

**course\_read\_all,course all,model\_openacademy\_course,,1,0,0,0**

**session\_read\_all,session all,model\_openacademy\_session,,1,0,0,0**

openacademy/security/security.xml

**<odoo>**

**<record id="group\_manager" model="res.groups">**

**<field name="name">OpenAcademy / Manager</field>**

**</record>**

**</odoo>**

### **Reglas de Registro**

Una regla de registro restringe los derechos de acceso a un subconjunto de registros de un modelo dado. Una regla es un registro del modelo **ir.rule**, y se asocia a un modelo, un número de grupos (campo Many2Many), los permisos a los que se aplicarán, y un dominio. El dominio especifica sobre cuáles registros se limitará el acceso.

Aquí un ejemplo de una regla que impide la eliminación de los prospectos (leads) que no se encuentren en el estado **cancel**. Observe que el valor del campo **groups** debe seguir la misma convención como el método [**write()**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.models.Model.write) del ORM.

**<record id="delete\_cancelled\_only" model="ir.rule">**

**<field name="name">Only cancelled leads may be deleted</field>**

**<field name="model\_id" ref="crm.model\_crm\_lead"/>**

**<field name="groups" eval="[(4, ref('sales\_team.group\_sale\_manager'))]"/>**

**<field name="perm\_read" eval="0"/>**

**<field name="perm\_write" eval="0"/>**

**<field name="perm\_create" eval="0"/>**

**<field name="perm\_unlink" eval="1" />**

**<field name="domain\_force">[('state','=','cancel')]</field>**

**</record>**

**Ejercicio**

Regla de registro

Agrega una regla de registro para el modelo Course y el grupo "OpenAcademy / Manager", que restringe el **write** y **unlink** al responsable de un curso. Si un curso no tiene responsable, todos los usuarios del grupo deben ser capaces de modificarlo.

Crea una nueva regla en **openacademy/security/security.xml**:

openacademy/security/security.xml

**<record id="group\_manager" model="res.groups">**

**<field name="name">OpenAcademy / Manager</field>**

**</record>**

**<record id="only\_responsible\_can\_modify" model="ir.rule">**

**<field name="name">Only Responsible can modify Course</field>**

**<field name="model\_id" ref="model\_openacademy\_course"/>**

**<field name="groups" eval="[(4, ref('openacademy.group\_manager'))]"/>**

**<field name="perm\_read" eval="0"/>**

**<field name="perm\_write" eval="1"/>**

**<field name="perm\_create" eval="0"/>**

**<field name="perm\_unlink" eval="1"/>**

**<field name="domain\_force">**

**['|', ('responsible\_id','=',False),**

**('responsible\_id','=',user.id)]**

**</field>**

**</record>**

**</odoo>**

Watch video 9

## Asistentes

Los asistentes describen sesiones interactivas con el usuario (o cajas de diálogos) a través de formularios dinámicos. Un asistente es simplemente un modelo que extiende la clase **TransientModel** en vez de [**Model**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.models.Model). La clase **TransientModel** extiende [**Model**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.models.Model) y re-usa todos sus mecanismos existentes, con las siguientes particularidades:

* Los registros de un asistente no pretenden ser persistentes; se borran automáticamente de la base de datos después de cierto tiempo. Por esta razón se llaman transient.
* Un asistente no requiere permisos explícitos: los usuarios tienen todos los permisos en cualquier asistente.
* Los registros de un asistente pueden referirse a registros normales a través de la definición de campos many2one, pero los registros regulares no se pueden referir a los registros de un asistente a través de campos many2one.

Queremos crear un asistente que permita a los usuarios crear attendees (asistentes) para una determinada sesión, o para una lista de sesiones a la vez.

**Ejercicio**

Define el asistente

Crea un modelo asistente con una relación many2one al modelo Session y una relación many2many con el modelo Partner.

Agrega un nuevo archivo **openacademy/wizard.py**:

openacademy/\_\_init\_\_.py

**from . import controllers**

**from . import models**

**from . import partner**

**from . import wizard**

openacademy/wizard.py

**# -\*- coding: utf-8 -\*-**

**from odoo import models, fields, api**

**class Wizard(models.TransientModel):**

**\_name = 'openacademy.wizard'**

**session\_id = fields.Many2one('openacademy.session',**

**string="Session", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

### **Ejecutar un asistente**

Los asistentes se lanzan usando registros **ir.actions.act\_window** con el campo **target** en **new**. Este último abre la vista del asistente en una ventana emergente. La acción puede ser llamada por un elemento de menú.

Hay otra manera para iniciar el asistente y esta es usando un registro **ir.actions.act\_window** como arriba, pero con un campo extra **src\_model** que especifica en el contexto de cuál modelo la acción está disponible. El asistente aparecerá en la acción contextual del modelo, por encima de la vista principal. Debido a algunos disparadores internos en el ORM, tal acción es declarada en XML con la etiqueta **act\_window**.

**<act\_window id="launch\_the\_wizard"**

**name="Launch the Wizard"**

**src\_model="context.model.name"**

**res\_model="wizard.model.name"**

**view\_mode="form"**

**target="new"**

**key2="client\_action\_multi"/>**

Los asistentes usan vistas regulares y sus botones pueden usar el atributo **special="cancel"** para cerrar la ventana del asistente sin guardar los cambios.

**Ejercicio**

Inicia el asistente

1. Define una vista de formulario para el asistente.
2. Añade la acción para ejecutar en el contexto del modelo Session.
3. Define un valor predeterminado para el campo de sesión en el asistente; Utiliza el parámetro de **self.\_context** para recuperar la sesión actual.

openacademy/wizard.py

**class Wizard(models.TransientModel):**

**\_name = 'openacademy.wizard'**

**def \_default\_session(self):**

**return self.env['openacademy.session'].browse(self.\_context.get('active\_id'))**

**session\_id = fields.Many2one('openacademy.session',**

**string="Session", required=True, default=\_default\_session)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

openacademy/views/openacademy.xml

**action="session\_list\_action"/>**

**<record model="ir.ui.view" id="wizard\_form\_view">**

**<field name="name">wizard.form</field>**

**<field name="model">openacademy.wizard</field>**

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

**<form string="Add Attendees">**

**<group>**

**<field name="session\_id"/>**

**<field name="attendee\_ids"/>**

**</group>**

**</form>**

**</field>**

**</record>**

**<act\_window id="launch\_session\_wizard"**

**name="Add Attendees"**

**binding\_model="openacademy.session"**

**res\_model="openacademy.wizard"**

**view\_mode="form"**

**target="new"**

**/>**

**</odoo>**

**Ejercicio**

Registro de asistentes

Añade botones al asistente e implementa el método correspondiente para agregar a los asistentes a la sesión dada.

openacademy/views/openacademy.xml

**</group>**

**<footer>**

**<button name="subscribe" type="object"**

**string="Subscribe" class="oe\_highlight"/>**

**or**

**<button special="cancel" string="Cancel"/>**

**</footer>**

**</form>**

**</field>**

**</record>**

**<act\_window id="launch\_session\_wizard"**

openacademy/wizard.py

**session\_id = fields.Many2one('openacademy.session',**

**string="Session", required=True, default=\_default\_session)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**def subscribe(self):**

**self.session\_id.attendee\_ids |= self.attendee\_ids**

**return {}**

**Ejercicio**

Registro de asistentes a las sesiones múltiples

Modifica el modelo de asistente (wizard) para que los asistentes puedan registrarse a varias sesiones.

openacademy/views/openacademy.xml

**<group>**

**<field name="session\_ids"/>**

**<field name="attendee\_ids"/>**

**</group>**

**<footer>**

**<button name="subscribe" type="object"**

**string="Subscribe" class="oe\_highlight"/>**

openacademy/wizard.py

**class Wizard(models.TransientModel):**

**\_name = 'openacademy.wizard'**

**def \_default\_sessions(self):**

**return self.env['openacademy.session'].browse(self.\_context.get('active\_ids'))**

**session\_ids = fields.Many2many('openacademy.session',**

**string="Sessions", required=True, default=\_default\_sessions)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**def subscribe(self):**

**for session in self.session\_ids:**

**session.attendee\_ids |= self.attendee\_ids**

**return {}**

Watch video 10

## Internacionalización

Cada módulo puede tener sus propias traducciones dentro del directorio i18n, con solo tener archivos llamados LANG.po donde LANG es el código local del idioma, o el código del idioma y el país combinados cuando son diferentes (por ejemplo pt.po o pt\_BR.po). Las traducciones serán cargadas automáticamente por Odoo para todos los idiomas habilitados. Los desarrolladores deberían utilizar inglés cuando crean un módulo, una vez listo exportan los términos de ese módulo usando la característica de exportar archivos POT la cual usa **gettext** para tal fin (**Settings ‣ Translations ‣ Import/Export ‣ Export Translation** sin especificar un lenguaje) para crear el archivo plantilla POT del módulo, y después usarlo para crear las traducciones derivadas, que serían archivos PO. Varios IDEs tienen plugins o modos de edición y unión de archivos PO/POT, por ejemplo **poEdit**.

**Truco**

Los archivos del Objeto Potable generados por Odoo son publicados en [**Transifex**](https://www.transifex.com/odoo/public/), haciendo fácil la traducción del programa.

**|- idea/ # The module directory**

**|- i18n/ # Translation files**

**| - idea.pot # Translation Template (exported from Odoo)**

**| - fr.po # French translation**

**| - pt\_BR.po # Brazilian Portuguese translation**

**| (...)**

**Truco**

Por defecto la exportación de los POT en Odoo sólo extrae etiquetas dentro de archivos XML o dentro de las definiciones de campo en código Python, pero cualquier cadena Python puede ser traducida de esta manera envolviéndola con la función **odoo.\_()** (e.g. **\_("Label")**)

**Ejercicio**

Traduce un Módulo

Elige un segundo idioma para la instalación de Odoo. Traduce su módulo utilizando los servicios proporcionados por Odoo.

1. Crea un directorio **openacademy/i18n/**
2. Instala cualquier idioma que desee ( **Administration ‣ Translations ‣ Load an Official Translation**)
3. Sincroniza términos traducibles (**Administration ‣ Translations ‣ Application Terms ‣ Synchronize Translations**)
4. Crea un archivo de plantilla de traducción por exportación ( **Administration ‣ Translations -> Import/Export ‣ Export Translation**) sin especificar un idioma, guardar en **openacademy/i18n/**
5. Crea un archivo de traducción mediante la exportación ( **Administration ‣ Translations ‣ Import/Export ‣ Export Translation**) especifica un idioma. Guárdalo en **openacademy/i18n/**
6. Abre el archivo exportado de la traducción (con un editor de texto básico o un editor de fichero PO dedicado por ejemplo [**POEdit**](http://poedit.net/)) y traduce los términos que faltan
7. En **models.py**, agrega una instrucción import para la función **odoo.\_** y marca las cadenas restantes como traducibles
8. Repite los pasos 3-6

openacademy/models.py

**# -\*- coding: utf-8 -\*-**

**from datetime import timedelta**

**from odoo import models, fields, api, exceptions, \_**

**class Course(models.Model):**

**\_name = 'openacademy.course'**

**default = dict(default or {})**

**copied\_count = self.search\_count(**

**[('name', '=like', \_(u"Copy of {}%").format(self.name))])**

**if not copied\_count:**

**new\_name = \_(u"Copy of {}").format(self.name)**

**else:**

**new\_name = \_(u"Copy of {} ({})").format(self.name, copied\_count)**

**default['name'] = new\_name**

**return super(Course, self).copy(default)**

**if self.seats < 0:**

**return {**

**'warning': {**

**'title': \_("Incorrect 'seats' value"),**

**'message': \_("The number of available seats may not be negative"),**

**},**

**}**

**if self.seats < len(self.attendee\_ids):**

**return {**

**'warning': {**

**'title': \_("Too many attendees"),**

**'message': \_("Increase seats or remove excess attendees"),**

**},**

**}**

**def \_check\_instructor\_not\_in\_attendees(self):**

**for r in self:**

**if r.instructor\_id and r.instructor\_id in r.attendee\_ids:**

**raise exceptions.ValidationError(\_("A session's instructor can't be an attendee"))**

Watch Video 10

## Informes

### **Informes impresos**

Odoo, A partir de la versión 8.0, viene con un nuevo motor de informe basado en [QWeb](http://vauxoo.github.io/odoo/reference/qweb.html#reference-qweb), [Twitter Bootstrap](http://getbootstrap.com/) y [Wkhtmltopdf](http://wkhtmltopdf.org/).

Un informe es un elemento de la combinación de dos elementos:

* un **ir.actions.report**, para el cual se proporciona un enlace directo **<report>**, que configura diversos parámetros básicos para el informe (por defecto el tipo, si el informe debe guardarse en la base de datos después de su generación,…)
* **<report**
* **id="account\_invoices"**
* **model="account.invoice"**
* **string="Invoices"**
* **report\_type="qweb-pdf"**
* **name="account.report\_invoice"**
* **file="account.report\_invoice"**
* **attachment\_use="True"**
* **attachment="(object.state in ('open','paid')) and**
* **('INV'+(object.number or '').replace('/','')+'.pdf')"**

**/>**

* Un estándar [QWeb view](http://vauxoo.github.io/odoo/reference/views.html#reference-views-qweb) para el informe real:
* **<t t-call="report.html\_container">**
* **<t t-foreach="docs" t-as="o">**
* **<t t-call="web.external\_layout">**
* **<div class="page">**
* **<h2>Report title</h2>**
* **</div>**
* **</t>**
* **</t>**
* **</t>**
* **the standard rendering context provides a number of elements, the most**
* **important being:**
* **``docs``**
* **the records for which the report is printed**
* **``user``**

**the user printing the report**

Debido a que los informes son páginas web estándar, están disponibles a través de una URL y los parámetros de salida pueden ser manipulados a través de esta URL, por ejemplo la versión HTML de la factura está disponible a través de <http://localhost:8069/report/html/account.report_invoice/1> (si está instalado **account**) y la versión en PDF a través de <http://localhost:8069/report/pdf/account.report_invoice/1>.

**Peligro**

Si parece que el informe PDF carece de estilos (es decir, aparece el texto pero el estilo/diseño es diferente de la versión html), probablemente su proceso [**wkhtmltopdf**](http://wkhtmltopdf.org/) no puede encontrar el servidor web para convertir y descargar los estilos.

Si revisas los logs de tu servidor y observas que los estilos CSS no están siendo descargados al generar un informe PDF, seguramente ese es el problema.

El proceso de [**wkhtmltopdf**](http://wkhtmltopdf.org/) utiliza el parámetro del sistema **web.base.url** como la root path a archivos de todos los **links**, pero este parámetro se actualiza automáticamente cada vez que el administrador ha iniciado sesión. Si su servidor se encuentra detrás de algún tipo de proxy, no podrá llegar. Puede solucionar esto mediante la adición de uno de estos parámetros del sistema:

* **report.url**, apuntando a una URL accesible desde el servidor (probablemente ‘’ [**http://localhost:8069**](http://localhost:8069/)'' o algo parecido). Se utilizará para este propósito particular solamente.
* **web.base.url.freeze**, cuando se establece en **True**, dejará las actualizaciones automáticas de **web.base.url**.

**Ejercicio**

Crea un informe para el modelo de sesión

Para cada sesión, debe mostrar el nombre de sesión, su inicio y final y una lista de los asistentes de la sesión.

openacademy/\_\_manifest\_\_.py

**'version': '0.1',**

**# any module necessary for this one to work correctly**

**'depends': ['base', 'web'],**

**# always loaded**

**'data': [**

openacademy/\_\_manifest\_\_.py

**'templates.xml',**

**'views/openacademy.xml',**

**'views/partner.xml',**

**'reports.xml',**

**],**

**# only loaded in demonstration mode**

**'demo': [**

openacademy/reports.xml

**<odoo>**

**<report**

**id="report\_session"**

**model="openacademy.session"**

**string="Session Report"**

**name="openacademy.report\_session\_view"**

**file="openacademy.report\_session"**

**report\_type="qweb-pdf" />**

**<template id="report\_session\_view">**

**<t t-call="web.html\_container">**

**<t t-foreach="docs" t-as="doc">**

**<t t-call="web.external\_layout">**

**<div class="page">**

**<h2 t-field="doc.name"/>**

**<p>From <span t-field="doc.start\_date"/> to <span t-field="doc.end\_date"/></p>**

**<h3>Attendees:</h3>**

**<ul>**

**<t t-foreach="doc.attendee\_ids" t-as="attendee">**

**<li><span t-field="attendee.name"/></li>**

**</t>**

**</ul>**

**</div>**

**</t>**

**</t>**

**</t>**

**</template>**

**</odoo>**

### **Paneles de control**

**Ejercicio**

Define un panel de control

Define un tablero que contenga la vista gráfico que creaste, la vista de calendario de sesiones y una vista de lista de los cursos (conmutables a una vista de formulario). Este panel debe estar disponible a través de un menuitem en el menú y aparecerá automáticamente en el cliente web cuando se selecciona el menú principal de OpenAcademy.

1. Crea un archivo **openacademy/views/session\_board.xml**. Este debe contener la vista de tablero, las acciones referenciadas en dicha vista, una acción para abrir el tablero y una nueva definición de la opción del menú principal para agregar la acción del tablero

**Nota**

Los estilos disponibles de tableros son **1**, **1-1**, **1-2**, **2-1** y **1-1-1**

1. Actualiza **openacademy/\_\_manifest\_\_.py** para hacer referencia al nuevo archivo de datos

openacademy/\_\_manifest\_\_.py

**'version': '0.1',**

**# any module necessary for this one to work correctly**

**'depends': ['base', 'report', 'board'],**

**# always loaded**

**'data': [**

**'templates.xml',**

**'views/openacademy.xml',**

**'views/partner.xml',**

**'views/session\_board.xml',**

**'reports.xml',**

**],**

**# only loaded in demonstration mode**

openacademy/views/session\_board.xml

**<?xml version="1.0"?>**

**<odoo>**

**<record model="ir.actions.act\_window" id="act\_session\_graph">**

**<field name="name">Attendees by course</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_mode">graph</field>**

**<field name="view\_id"**

**ref="openacademy.openacademy\_session\_graph\_view"/>**

**</record>**

**<record model="ir.actions.act\_window" id="act\_session\_calendar">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_mode">calendar</field>**

**<field name="view\_id" ref="openacademy.session\_calendar\_view"/>**

**</record>**

**<record model="ir.actions.act\_window" id="act\_course\_list">**

**<field name="name">Courses</field>**

**<field name="res\_model">openacademy.course</field>**

**<field name="view\_mode">tree,form</field>**

**</record>**

**<record model="ir.ui.view" id="board\_session\_form">**

**<field name="name">Session Dashboard Form</field>**

**<field name="model">board.board</field>**

**<field name="type">form</field>**

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

**<form string="Session Dashboard">**

**<board style="2-1">**

**<column>**

**<action**

**string="Attendees by course"**

**name="%(act\_session\_graph)d"**

**height="150"**

**width="510"/>**

**<action**

**string="Sessions"**

**name="%(act\_session\_calendar)d"/>**

**</column>**

**<column>**

**<action**

**string="Courses"**

**name="%(act\_course\_list)d"/>**

**</column>**

**</board>**

**</form>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="open\_board\_session">**

**<field name="name">Session Dashboard</field>**

**<field name="res\_model">board.board</field>**

**<field name="view\_mode">form</field>**

**<field name="usage">menu</field>**

**<field name="view\_id" ref="board\_session\_form"/>**

**</record>**

**<menuitem**

**name="Session Dashboard" parent="base.menu\_reporting\_dashboard"**

**action="open\_board\_session"**

**sequence="1"**

**id="menu\_board\_session" icon="terp-graph"/>**

**</odoo>**

Watch Video 11

## WebServices

El módulo de servicios web ofrece una interfaz común para todos los servicios web:

* XML-RPC
* JSON-RPC

Los objetos de negocio también se pueden acceder mediante el mecanismo de objetos distribuidos. Pueden ser modificables mediante la interfaz de cliente con vistas contextuales.

Odoo es accesible a través de interfaces XML-RPC/JSON-RPC, hay bibliotecas en muchos lenguajes.

### **Librería XML-RPC**

El siguiente ejemplo es un programa Python que interactúa con un servidor de Odoo con la librería **xmlrpclib**:

**import xmlrpclib**

**root = 'http://%s:%d/xmlrpc/' % (HOST, PORT)**

**uid = xmlrpclib.ServerProxy(root + 'common').login(DB, USER, PASS)**

**print "Logged in as %s (uid: %d)" % (USER, uid)**

**# Create a new note**

**sock = xmlrpclib.ServerProxy(root + 'object')**

**args = {**

**'color' : 8,**

**'memo' : 'This is a note',**

**'create\_uid': uid,**

**}**

**note\_id = sock.execute(DB, uid, PASS, 'note.note', 'create', args)**

**Ejercicio**

Añade un nuevo servicio al cliente

Escribe un programa de Python capaz de enviar peticiones XML-RPC a un PC Odoo (tuyo o de tu instructor). Este programa debe mostrar todas las sesiones y su correspondiente número de asientos. También debe crear una nueva sesión para uno de los cursos.

**import functools**

**import xmlrpclib**

**HOST = 'localhost'**

**PORT = 8069**

**DB = 'openacademy'**

**USER = 'admin'**

**PASS = 'admin'**

**ROOT = 'http://%s:%d/xmlrpc/' % (HOST,PORT)**

**# 1. Login**

**uid = xmlrpclib.ServerProxy(ROOT + 'common').login(DB,USER,PASS)**

**print "Logged in as %s (uid:%d)" % (USER,uid)**

**call = functools.partial(**

**xmlrpclib.ServerProxy(ROOT + 'object').execute,**

**DB, uid, PASS)**

**# 2. Read the sessions**

**sessions = call('openacademy.session','search\_read', [], ['name','seats'])**

**for session in sessions:**

**print "Session %s (%s seats)" % (session['name'], session['seats'])**

**# 3.create a new session**

**session\_id = call('openacademy.session', 'create', {**

**'name' : 'My session',**

**'course\_id' : 2,**

**})**

En lugar de utilizar un id de curso en el código, el código puede buscar un curso por nombre:

**# 3.create a new session for the "Functional" course**

**course\_id = call('openacademy.course', 'search', [('name','ilike','Functional')])[0]**

**session\_id = call('openacademy.session', 'create', {**

**'name' : 'My session',**

**'course\_id' : course\_id,**

**})**

### **Biblioteca de JSON-RPC**

El siguiente ejemplo es un programa Python que interactúa con un servidor de Odoo con las librerías estándar de Python **urllib2** y **json**:

**import json**

**import random**

**import urllib2**

**def json\_rpc(url, method, params):**

**data = {**

**"jsonrpc": "2.0",**

**"method": method,**

**"params": params,**

**"id": random.randint(0, 1000000000),**

**}**

**req = urllib2.Request(url=url, data=json.dumps(data), headers={**

**"Content-Type":"application/json",**

**})**

**reply = json.load(urllib2.urlopen(req))**

**if reply.get("error"):**

**raise Exception(reply["error"])**

**return reply["result"]**

**def call(url, service, method, \*args):**

**return json\_rpc(url, "call", {"service": service, "method": method, "args": args})**

**# log in the given database**

**url = "http://%s:%s/jsonrpc" % (HOST, PORT)**

**uid = call(url, "common", "login", DB, USER, PASS)**

**# create a new note**

**args = {**

**'color' : 8,**

**'memo' : 'This is another note',**

**'create\_uid': uid,**

**}**

**note\_id = call(url, "object", "execute", DB, uid, PASS, 'note.note', 'create', args)**

Aquí está el mismo programa, utilizando la biblioteca [jsonrpclib](https://pypi.python.org/pypi/jsonrpclib):

**import jsonrpclib**

**# server proxy object**

**url = "http://%s:%s/jsonrpc" % (HOST, PORT)**

**server = jsonrpclib.Server(url)**

**# log in the given database**

**uid = server.call(service="common", method="login", args=[DB, USER, PASS])**

**# helper function for invoking model methods**

**def invoke(model, method, \*args):**

**args = [DB, uid, PASS, model, method] + list(args)**

**return server.call(service="object", method="execute", args=args)**

**# create a new note**

**args = {**

**'color' : 8,**

**'memo' : 'This is another note',**

**'create\_uid': uid,**

**}**

**note\_id = invoke('note.note', 'create', args)**

Los ejemplos pueden ser fácilmente adaptados de XML-RPC a JSON-RPC.

**Nota**

Hay una serie de APIs de alto nivel en varios idiomas para acceder a los sistemas de Odoo sin pasar explícitamente por XML-RPC o JSON-RPC, tales como:

* [**https://github.com/akretion/ooor**](https://github.com/akretion/ooor)
* [**https://github.com/syleam/openobject-library**](https://github.com/syleam/openobject-library)
* [**https://github.com/nicolas-van/openerp-client-lib**](https://github.com/nicolas-van/openerp-client-lib)
* [**http://pythonhosted.org/OdooRPC**](http://pythonhosted.org/OdooRPC)
* [**https://github.com/abhishek-jaiswal/php-openerp-lib**](https://github.com/abhishek-jaiswal/php-openerp-lib)

[[1]](http://vauxoo.thinkific.com/courses/take/odoo-backend/texts/1966776-introduccion-webservices#id2)

es posible [**desactivar la creación automática de algunos campos**](http://vauxoo.github.io/odoo/reference/orm.html#odoo.models.Model._log_access)

[[2]](http://vauxoo.thinkific.com/courses/take/odoo-backend/texts/1966776-introduccion-webservices#id1)

escribir consultas SQL directas es totalmente posible, pero requiere cuidado, ya que ignora todos los mecanismos de autenticación y seguridad y de Odoo.

Watch Video 12