



odoo

**UT06: DESARROLLO DE
COMPONENTES. MODELO VISTA
CONTROLADOR Y HERENCIA**

ÍNDICE

- 1.- Modelo en Odoo: campos básicos
- 2.- Modelo en Odoo: campos relacionales
- 3.- Modelo en Odoo: campos calculados
- 4.- Modelo en Odoo: restricciones
- 5.- Vistas en Odoo
- 6.- Vistas de tipo lista
- 7.- Vistas de tipo formulario

1

MODELO EN ODOO: CAMPOS BÁSICOS



Como se mencionó en el apartado anterior, el **modelo** en Odoo permite representar en código Python la estructura que tendrá la base de datos (ORM).

Un modelo se define como una clase Python que **hereda de la clase `models.Model`**.

Un modelo representa una tabla entera en la base de datos, mientras que las propiedades que son instancias de **`fields.Field`** serán los campos de dicha tabla.

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

```
from odoo import models, fields, api
```

```
class SchoolStudent(models.Model):
```

```
    _name= 'school.student'
```

```
    _description = 'Datos alumno'
```

```
    name = fields.Char(string='name', required=True)
```

Crearé una tabla que se llamará *school_student*

Y tendrá un campo de tipo texto llamado *name*

Los diferentes tipos de campos disponibles son:

- fields.**Boolean**: encapsula un booleano
- fields.**Char**: cadena de texto
- fields.**Float**: número de punto flotante
- fields.**Integer**: número entero
- fields.**Binary**: contenido en binario, por ejemplo, un fichero
- fields.**Html**: contenido en código HTML
- fields.**Image**: extiende Binary y puede contener una imagen
- fields.**Monetary**: encapsula un flotante expresado en una moneda
- fields.**Selection**: un valor de entre una lista
- fields.**Text**: similar a Char, pero para contenidos más largos.
- fields.**Date**: una fecha
- fields.**Datetime**: fecha y hora
- fields.**Many2one**: una relación muchos a uno en la base de datos

- `fields.One2many`: relación uno a muchos
- `fields.Many2many`: relación muchos a muchos

En la documentación de Odoo puedes ver información más detallada sobre cada uno de estos tipos de campos

<https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#basic-fields>



Cuando creamos uno de estos campos le podemos pasar una serie de **parámetros** para modificar su comportamiento por defecto.

Muchos parámetros son comunes para todos (son parámetros de la clase **fields.Field**, de la que heredan todas las demás), mientras que hay otros que son específicos según el tipo de datos que encapsulen.

Veamos algunos de los más comunes, si quieres información extra sobre el resto de los parámetros puedes ir al enlace arriba indicado.

Parámetro de *fields.Field*: **string**

La etiqueta del campo que se mostrará a los usuarios. Si no se indica Odoo tomará el nombre del campo capitalizado (con la primera letra de cada palabra en mayúsculas)

```
name = fields.Char( string='Nombre' )
```

Parámetro de *fields.Field*: **help**

El contenido del *tooltip* que se mostrará a los usuarios cuando coloquen el cursor sobre el campo

```
name = fields.Char( string='Nombre',  
                    |         |         |  
                    |         |         |         help='Nombre del alumno' )
```

Parámetro de *fields.Field*: **readonly**

Indica si el campo es de **solo lectura**. Esto solo tiene efecto en la vista (no se mostrará al usuario la opción de editar el campo), pero se le podrá asignar valor mediante código.

```
exp = fields.Char( string='Número de expediente',  
                    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  
                    readOnly=True)
```

Parámetro de *fields.Field*: **required**

Indica que el campo es obligatorio

```
name = fields.Char( string='Nombre',  
                    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  
                    help='Nombre del alumno',  
                    required=True )
```


Parámetro de *fields.Field*: **default**

Valor por defecto del campo

```
repeater = fields.Boolean( string='Repetidor',  
                             default=False)
```

Además de los parámetros comunes a todos los objetos de tipo Field, hay una serie de parámetros específicos a cada uno de los tipos de datos disponibles.

fields.Boolean

Encapsula un valor booleano

fields.Char

Un campo de cadena que puede tener limitado su tamaño. Se suele representar en la vista en una única línea de texto.

Parámetros:

- **size:** el tamaño máximo de los valores almacenados

fields.Float

Encapsula un número flotante.

Parámetros:

- digits: precisión del número, se indica como una tupla (total, decimal)

```
precio = fields.Float(  
    string="Precio",  
    digits=(10, 2), # Total 10 dígitos, con 2 decimales  
    help="Precio del producto con precisión de dos decimales."  
)
```

fields.Integer

Encapsula un número entero.

fields.Binary

Encapsula contenido binario, por ejemplo, un fichero

Parámetros:

- **attachment:** booleano que indica si debe almacenarse como ir_attachment o en una columna de la tabla del modelo

```
archivo = fields.Binary(string="Archivo Adjunto", attachment=True)
```

Archivo binario ?

SUBA SU ARCHIVO

fields.Html

Encapsula contenido en código HTML

Parámetros:

- **sanitize:** si el valor debe ser sanitizado (booleano). La sanitización es un mecanismo para evitar ataques XSS (Cross-Site Scripting)

```
contenido = fields.Html(string="Contenido", sanitize=True)
```

fields.Image

Es una extensión de Binary que encapsula una imagen

Parámetros:

- **max_width**: el ancho máximo de la imagen, si su tamaño fuera superior se redimensionaría manteniendo el ratio de aspecto
- **max_height**: la altura máxima de la imagen

fields.Monetary

Se utiliza para manejar valores monetarios asociados a una moneda específica, lo que facilita el tratamiento de cualquier dato financiero.

Se puede vincular a un campo de tipo **many2one** que apunta al modelo de monedas (res.currency)

Parámetros:

- **currency_field**: aquí indicamos la moneda de que se trata mediante su indicador. si no indicamos este campo tomará la moneda por defecto.

Podemos ver información sobre las monedas en *Facturación/Contabilidad* -> *Configuración* -> *Monedas*

The screenshot shows the Odoo web interface. The top navigation bar includes 'Facturación / Contabilidad', 'Clientes', 'Proveedores', 'Informes', and 'Configuración'. The 'Facturación / Contabilidad' menu is open, showing options like 'Facturas', 'NUEVO', 'SUBIR', and 'Información de compañía'. The 'Configuración' dropdown is also open, showing options like 'Ajustes', 'Facturación / Contabilidad', 'Condiciones de pago', 'Incoterms', 'Bancos', 'Agregar una cuenta bancaria', 'Contabilidad', 'Impuestos', 'Diarios', 'Monedas', and 'Posiciones fiscales'. The 'Monedas' option is selected, displaying a list of currencies.

<input type="checkbox"/>	Moneda	Símbolo	Nombre	Última actualización	Tipo actual	Activo	
<input type="checkbox"/>	EUR	€	Euro	01/01/2010	1,000000	<input checked="" type="checkbox"/>	
<input type="checkbox"/>	AED	د.ا	United Arab Emirates dirham	01/01/2010	4,512532	<input type="checkbox"/>	
<input type="checkbox"/>	AFN	Afs	Afghan afghani	01/01/2010	59,330000	<input type="checkbox"/>	
<input type="checkbox"/>	ALL	L	Albanian lek	01/01/2010	138,000000	<input type="checkbox"/>	
<input type="checkbox"/>	AMD	դր.	Armenian dram	01/01/2010	506,020000	<input type="checkbox"/>	
<input type="checkbox"/>	ANG	f	Netherlands Antillean guilder	01/01/2010	2,110000	<input type="checkbox"/>	
<input type="checkbox"/>	AOA	Kz	Angolan kwanza	01/01/2010	117,080000	<input type="checkbox"/>	

fields.Selection

Muestra un campo selector con varios valores

Parámetros:

- **selection:** especifica los posibles valores para este campo, se indica mediante una lista de tuplas (valor, etiqueta)

```
nombre = fields.Char(string="Nombre del Producto", required=True)
categoria = fields.Selection(
    [
        ('electronica', 'Electrónica'),
        ('ropa', 'Ropa'),
        ('alimentacion', 'Alimentación')
    ],
    string="Categoría",
    required=True,
    default='electronica'
)
```

fields.Date

Se utiliza para almacenar fechas sin información de hora.

Además, la clase `fields.Date` proporciona una serie de funciones para la manipulación de fechas:

- `fields.Date.today()`: devuelve la fecha actual

```
fecha_creacion = fields.Date(default=fields.Date.today)
```

- `fields.Date.to_string()`: convierte un objeto de tipo `Date` en una cadena de texto en formato ISO (YYYY-MM-DD)

```
from datetime import date  
fecha_cadena = fields.Date.to_string(date(2024, 11, 22))  
print(fecha_cadena)  # "2024-11-22"
```

- `fields.Date.from_string()`: convierte una cadena de texto en formato YYYY-MM-DD a un objeto de tipo `Date`

```
fecha_objeto = fields.Date.from_string("2024-11-22")  
print(fecha_objeto)  # datetime.date(2024, 11, 22)
```

- `fields.Date.add(days=0, months=0, years=0)`: permite sumar o restar días, meses o años a una fecha.

```
from datetime import date  
fecha_inicial = date(2024, 11, 22)  
fecha_modificada = fields.Date.add(fecha_inicial, days=30)  
print(fecha_modificada)  # 2024-12-22
```

- `fields.Date.delta(start_date, end_date)`: calcula la diferencia entre dos fechas en días

```
dias_diferencia = fields.Date.delta("2024-11-22", "2024-12-22")  
print(dias_diferencia)  # 30
```

fields.Datetime

Similar a Date, pero almacena también la hora.

De forma análoga a Date, también tiene una serie de métodos estáticos para manipular fechas y horas.

- `fields.Datetime.now()`: devuelve la fecha y hora actual

```
fecha_hora_actual = fields.Datetime.now()
print(fecha_hora_actual)  # 2024-11-22 10:30:45
```

- `fields.Datetime.today()`: devuelve la fecha actual a medianoche (00:00:000) en formato UTC

```
fecha_hora_actual = fields.Datetime.now()
print(fecha_hora_actual)  # 2024-11-22 10:30:45
```

- `fields.Date.to_string()`: convierte una cadena de texto en formato ISO (YYYY-MM-DD HH:MM:SS)

```
from datetime import datetime
fecha_cadena = fields.Datetime.to_string(datetime(2024, 11, 22, 14, 30))
print(fecha_cadena) # "2024-11-22 14:30:00"
```

- `fields.Date.from_string(value)`: convierte una cadena de texto en formato ISO a un objeto datetime

```
fecha_objeto = fields.Datetime.from_string("2024-11-22 14:30:00")
print(fecha_objeto) # datetime.datetime(2024, 11, 22, 14, 30)
```

- `fields.Date.add(days=0, hours=0, minutes=0, months=0, years=0)`: suma o resta

```
from datetime import datetime
fecha_inicial = datetime(2024, 11, 22, 10, 0)
fecha_modificada = fields.Datetime.add(fecha_inicial, days=5, hours=3)
print(fecha_modificada) # 2024-11-27 13:00:00
```

- `fields.Datetime.to_datetime(value, format)`: convierte una cadena a un objeto datetime utilizando un formato específico

```
fecha = fields.Datetime.to_datetime("22/11/2024 14:30", format="%d/%m/%Y %H:%M")
print(fecha) # datetime.datetime(2024, 11, 22, 14, 30)
```

- `fields.Datetime.subtract(start_time, end_time)`: calcula la diferencia entre dos objetos datetime y devuelve un objeto timedelta

```
from datetime import datetime
diferencia = fields.Datetime.subtract(datetime(2024, 11, 22), datetime(2024, 11, 20))
print(diferencia.days) # 2 días
```

2

MODELO EN ODOO: CAMPOS RELACIONALES



Los campos relacionales en Odoo nos permitirán establecer relaciones entre los diferentes modelos.

Hay tres tipos de campos relacionales:

- **Many2one**
- **One2many**
- **Many2many**

Campo Many2one

Indica que el modelo en el que está tiene una relación **muchos a uno** con otro modelo.

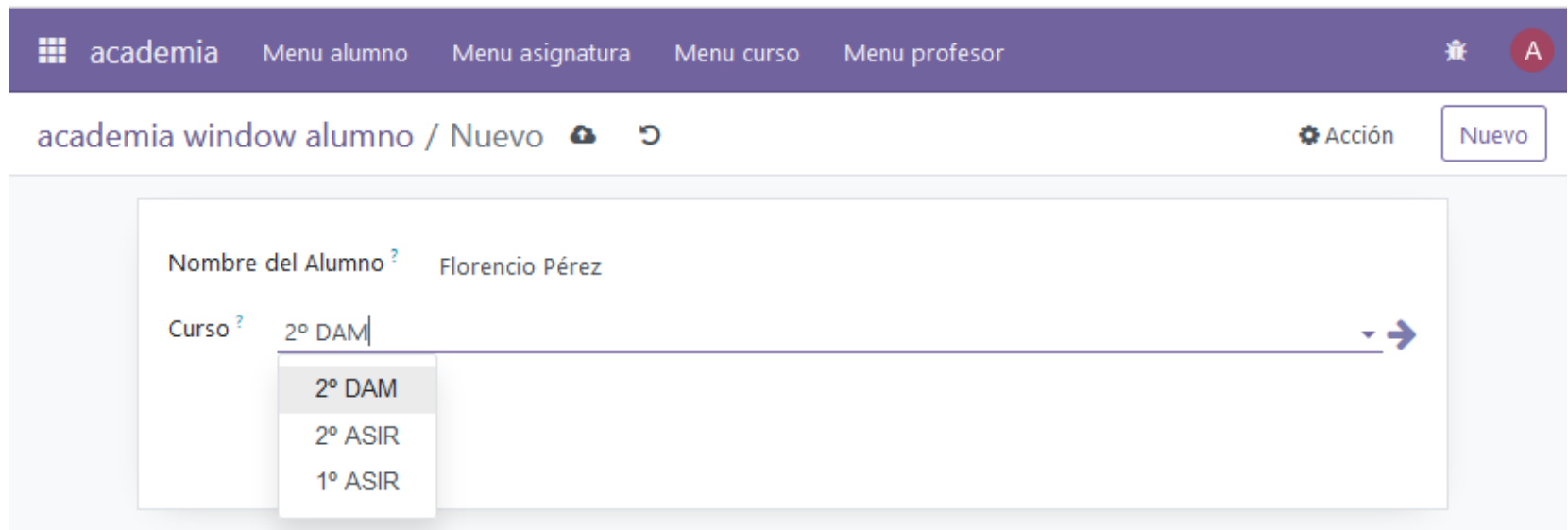
Ejemplo: un modelo que almacene información de alumnos podrá tener un campo de tipo Many2one que relacione cada alumno con el curso en el que está.

```
class Alumno(models.Model):
    _name = 'academia.alumno'
    _description = 'Modelo de Alumnos'

    name = fields.Char(
        string='Nombre del Alumno',
        required=True)
    curso_id = fields.Many2one(
        comodel_name='academia.curso',
        string='Curso')
```

Algunos parámetros que admite son:

- **comodel_name**: nombre del modelo con el que está relacionado.
- **onDelete**: indica como será el comportamiento si el registro al que se refiere es eliminado. Las opciones son: 'set null', 'restrict' y 'cascade'



The screenshot shows the 'New Student' form in the 'academia' module. The form has a purple header bar with navigation links: 'academia', 'Menu alumno', 'Menu asignatura', 'Menu curso', and 'Menu profesor'. On the right of the header are a star icon and a red circle with the letter 'A'. Below the header, the breadcrumb 'academia window alumno / Nuevo' is followed by a lock icon, a refresh icon, a gear icon labeled 'Acción', and a 'Nuevo' button. The form itself is a light gray box containing two fields: 'Nombre del Alumno' with the value 'Florencio Pérez' and a help icon, and 'Curso' with the value '2º DAM' and a help icon. A dropdown menu is open for the 'Curso' field, showing three options: '2º DAM' (highlighted), '2º ASIR', and '1º ASIR'. A blue arrow icon is at the end of the dropdown list.

En este ejemplo hemos creado nosotros los dos modelos, pero podríamos utilizar un modelo de los ya existentes en Odoo.

Por ejemplo, si quiero tener un campo país lo normal sería relacionarlo con el modelo **res_country**

Campo One2Many

Es el inverso de Many2one, ya que establece una relación **uno a muchos**.

Requiere que exista un campo Many2one en el otro modelo

```
class Curso(models.Model):
    _name = 'academia.curso'
    _description = 'Modelo de Cursos'

    name = fields.Char(
        string='Nombre del Curso',
        required=True)

    alumno_ids = fields.One2many(
        comodel_name='academia.alumno',
        inverse_name='curso_id',
        string='Alumnos')
```

Parámetros:

- **comodel_name**: nombre del modelo con el que está relacionado.
- **inverse_name**: nombre del campo correspondiente de tipo Many2one en el otro modelo.

The screenshot shows the Odoo web interface for the 'academia' module. The top navigation bar includes 'academia', 'Menu alumno', 'Menu asignatura', 'Menu curso', and 'Menu profesor'. The user is logged in as 'Administrator (odoo)'. The breadcrumb trail is 'academia window curso / 2º DAM'. The main content area displays a table with the following structure:

Nombre del Curso ? 2º DAM		Profesor ?
Alumnos ?	Nombre del Alumno	Curso
	Pepe Fernández	2º DAM
	Florencio Pérez	2º DAM
Agregar una línea		

Each row in the table has a delete icon (trash can) in the rightmost column. The interface also includes a 'Nuevo' button and a '1 / 3' indicator.

Campo Many2Many

Se trata de una relación **muchos a muchos**.

En la base de datos se gestiona con una tabla intermedia con claves ajenas a las dos tablas. Esta tabla intermedia es generada automáticamente por Odoo y nosotros no tenemos que preocuparnos de ella para nada.

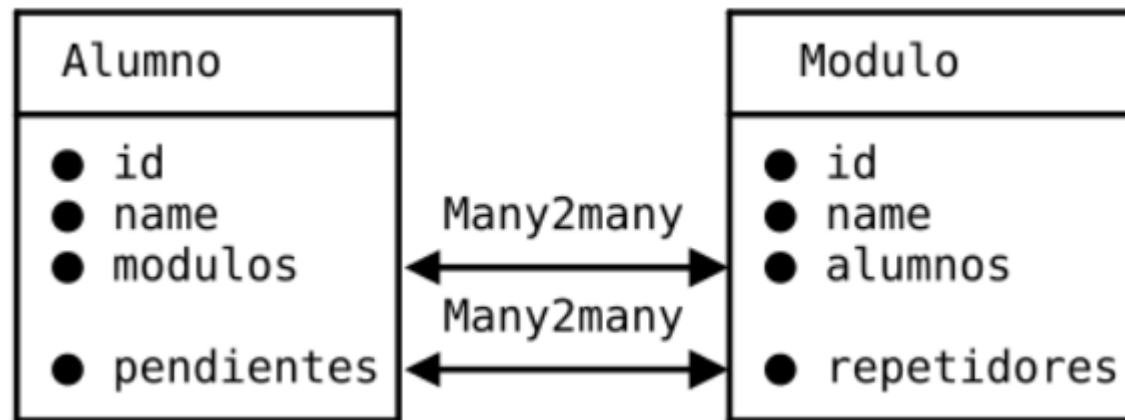
Cuando hay dos modelos que tienen un campo Many2many apuntando al otro modelo asumirá que ambos corresponden a la misma relación y creará la tabla intermedia con un nombre generado a partir del nombre de los dos modelos.

```
class Curso(models.Model):  
    _name = 'academia.curso'  
    _description = 'Modelo de Cursos'  
  
    asignatura_ids = fields.Many2many(  
        'academia.asignatura',  
        string='Asignaturas')
```

```
class Asignatura(models.Model):  
    _name = 'academia.asignatura'  
    _description = 'Modelo de Asignaturas'  
  
    name = fields.Char(  
        string='Nombre de la Asignatura',  
        required=True)  
  
    curso_ids = fields.Many2many(  
        'academia.curso',  
        string='Cursos Asociados')
```

El problema lo podemos encontrar cuando tenemos dos modelos que están relacionados por dos campos Many2many entre ellos.

Ejemplo: tenemos dos tablas: alumnos y módulos. La tabla *alumnos* tiene un campo *módulo* que está relacionada muchos a muchos con la tabla *módulos*, pero también tiene un campo *pendiente* que también está relacionado muchos a muchos con la tabla de módulos.




```
alumnos_ids = fields.Many2many(comodel_name='modulo.alumno',  
    relation='modulos_alumnos', # El nombre de la tabla intermedia  
    column1='modulo_id', # El nombre en la tabla intermedia de la clave a este modelo  
    column2='alumno_id') # El nombre de la clave al otro modelo.  
repetidores_ids = fields.Many2many(comodel_name='modulo.alumno',  
    relation='modulos_alumnos_repetidores', # El nombre de la tabla intermedia  
    column1='modulo_id', # El nombre en la tabla intermedia de la clave a este modelo  
    column2='alumno_id') # El nombre de la clave al otro modelo.  
modulos_ids = field.Many2many(comodel_name='modulo.modulo',  
    relation='modulos_alumnos', # El nombre de la tabla intermedia  
    column1='alumno_id', # El nombre en la tabla intermedia de la clave a este modelo  
    column2='modulo_id') # El nombre de la clave al otro modelo.  
pendientes_ids = field.Many2many(comodel_name='modulo.modulo',  
    relation='modulos_alumnos_repetidores', # El nombre de la tabla intermedia  
    column1='alumno_id', # El nombre en la tabla intermedia de la clave a este modelo  
    column2='modulo_id') # El nombre de la clave al otro modelo.
```

3

MODELO EN ODOO: CAMPOS CALCULADOS



Un **campo calculado** es un campo cuyo valor se determina automáticamente mediante una función Python.

No se almacena en la base de datos a menos que explícitamente lo configures para que sea almacenado (*store=True*).

Los campos calculados se definen en dos partes:

- En el campo que queremos que se calcule automáticamente debe tener un atributo **compute** cuyo valor sea el nombre de la función que realizará el cálculo.
- La función debe tener el **decorador** **@api.depends()** referenciando los campos de los que depende

```
class Producto(models.Model):  
    _name = 'mi_modulo.producto'  
    _description = 'Modelo de Producto'
```

```
    nombre          = fields.Char()  
    precio_unitario = fields.Float()  
    iva              = fields.Float()  
    precio_total    = fields.Float( compute="_compute_precio_total" )
```

```
@api.depends('precio_unitario', 'iva')
```

```
def _compute_precio_total(self):
```

```
    for producto in self:
```

```
        producto.precio_total = producto.precio_unitario * (1 + producto.iva / 100)
```

Atributo compute para que Odoo sepa que este campo se calcula automáticamente a partir de otros

La función que calculará el valor del campo

Si se modifican los valores precio_unitario y/o iva se recalcula el valor del campo.

La función siempre comienza iterando sobre *self*

Las funciones de los campos calculados tienen el decorador `@api.depends`

Por defecto, los campos calculados no se almacenan en la base de datos, sino que se calculan en tiempo real.

Si queremos mejorar el rendimiento almacenando su valor en la base de datos podemos usar el parámetro **store=True**

```
total = fields.Float(  
    string="Total",  
    compute="_compute_total",  
    store=True )
```

La principal **ventaja** de esto es que mejora el rendimiento en consultas y filtrados, ya que no hay que recalcular el valor cada vez.

El **inconveniente** es que ocupa más espacio en la base de datos y que será necesario recalcular todos los valores cada vez que se cambian las dependencias.

En ocasiones, nos interesará que los campos calculados puedan ser **editables** (que el usuario pueda cambiar manualmente su valor).

Esto se soluciona con el parámetro **inverse**, que indicará cuál es la función que escribirá los campos originales.

```
price = fields.Float('Precio')
quantity = fields.Integer('Cantidad')
total = fields.Float(
    string = 'Total',
    readonly = False,
    compute="_compute_total",
    inverse="_inverse_total")

@api.depends('price', 'quantity')
def _compute_total(self):
    for record in self:
        record.total = record.price * record.quantity

def _inverse_total(self):
    for record in self:
        if record.quantity != 0:
            record.price = record.total / record.quantity
```

4

MODELO EN ODOO: RESTRICCIONES



Las **restricciones** son mecanismos utilizados para asegurar la integridad de los datos y el correcto funcionamiento del sistema, aplicando **condiciones o reglas** que deben cumplirse en los registros.

Estas restricciones pretenden controlar la introducción de datos, evitando situaciones no deseadas como la duplicación o el ingreso de valores inválidos.

Las restricciones se pueden aplicar a diferentes niveles:

- A nivel de base de datos (SQL Constrains)
- A nivel de modelo (Python Constrains)
- En la interfaz de usuario
- Mediante seguridad de acceso con reglas de seguridad
- Con validaciones a nivel de flujo de trabajo

Nosotros veremos únicamente las dos primeras

Odoo permite introducir directamente restricciones SQL en el modelo mediante la propiedad **`_sql_constraints`**.

Se utilizan cuando queremos realizar validaciones simples, por ejemplo:

- Que el valor de un campo sea único
- Validaciones que se pueden expresar con una condición SQL, como que un campo sea positivo o que cumpla un formato específico.

Ventajas:

- Se ejecutan directamente en la base de datos, por lo que son más eficientes
- Útiles con datos críticos que necesitan **integridad de datos** a nivel de base de datos
- Evitan entradas incorrectas desde fuera de Odoo

Sintaxis:

```
_sql_constraints = [ ( name, sql_definition, message) ]
```

donde el valor es una lista de tuplas, cada una con los siguientes campos:

- `name`: el nombre de la restricción
- `sql_definition`: expresión con la restricción
- `message`: mensaje de error

En <https://www.postgresql.org/docs/12/ddl-constraints.html> puedes ver ejemplos de restricciones en PostgreSQL

Ejemplo: campo con valor único

```
class Book(models.Model):  
    _name = 'library.book'  
    _description = 'library.bool'  
    _sql_constraints = [  
        ('unique_name', 'unique(name)', 'El nombre debe ser único')  
    ]  
  
    name = fields.Char()
```

Ejemplo: campo con valor positivo

```
class Book(models.Model):  
    _name = 'library.book'  
    _description = 'library.book'  
    _sql_constraints = [  
        ('check_price', 'CHECK(price > 0)', 'El precio debe ser positivo')  
    ]  
  
    price = fields.Integer()
```

Las **restricciones Python** las usaremos:

- Cuando queramos realizar **validaciones más complejas** o que involucren lógica de negocio. Como validaciones que dependen de varios campos, relaciones o cálculos. Ejemplo: fecha de inicio sea anterior a fecha de finalización.
- Cuando queramos **personalizar los mensajes de error**. Con **@api.constrains** podemos lanzar mensajes personalizados y manejar excepciones con `ValidationError`.
- En **validaciones dependientes de registros relacionados**, por ejemplo, que no haya dos registros activos con el mismo cliente en una relación Many2one.

Si usamos excepciones
debemos importar la librería

```
from odoo.exceptions import ValidationError
```

```
class ProductTemplate(models.Model):
```

```
    _name = 'product.template'
```

```
    price = fields.Float('price')
```

Las restricciones se indican con el
decorador **@api.constrains**

```
@api.constrains('price')
```

```
def _check_price(self):
```

```
    for record in self:
```

```
        if record.price < 0:
```

```
            raise ValidationError('El precio debe ser > 0')
```

Aquí indicamos a qué campo afectan las
restricciones. La función se ejecutará cada vez
que este campo sea modificado

5

VISTAS EN OD00



La **vista** en Odoo se encarga de todo lo que tenga que ver con la interacción con el usuario.

La vista se define en ficheros XML que se guardan en la base de datos y son consultados como cualquier otro modelo. Estos ficheros XML (que en el ejemplo anterior vimos que se guardaban en el directorio *views*) son referenciados en el **__manifest__.py**



La **vista** tiene varios elementos necesarios para funcionar:

- **Definiciones de vistas:** son las propias definiciones de las vistas, guardadas en el modelo *ir.ui.views*. Contiene por lo menos los *fields* que se van a mostrar y pueden tener información sobre la disposición, el comportamiento o el aspecto de los *fields*.
- **Menús:** se distribuyen de forma jerárquica y se guardan en el modelo *ir.ui.menus*.
- **Actions:** las acciones enlazan una acción del usuario (como pulsar un menú) con una llamada al servidor desde el cliente para pedir algo (p.e. cargar una vista). Las *actions* se guardan en varios modelos según el tipo.

Vamos a comenzar con las **definiciones de vistas**.

Hay muchos tipos de vistas (imagen de la derecha), cada una de ellas representando un modo de visualización diferente, pero nos centraremos en las más comunes.

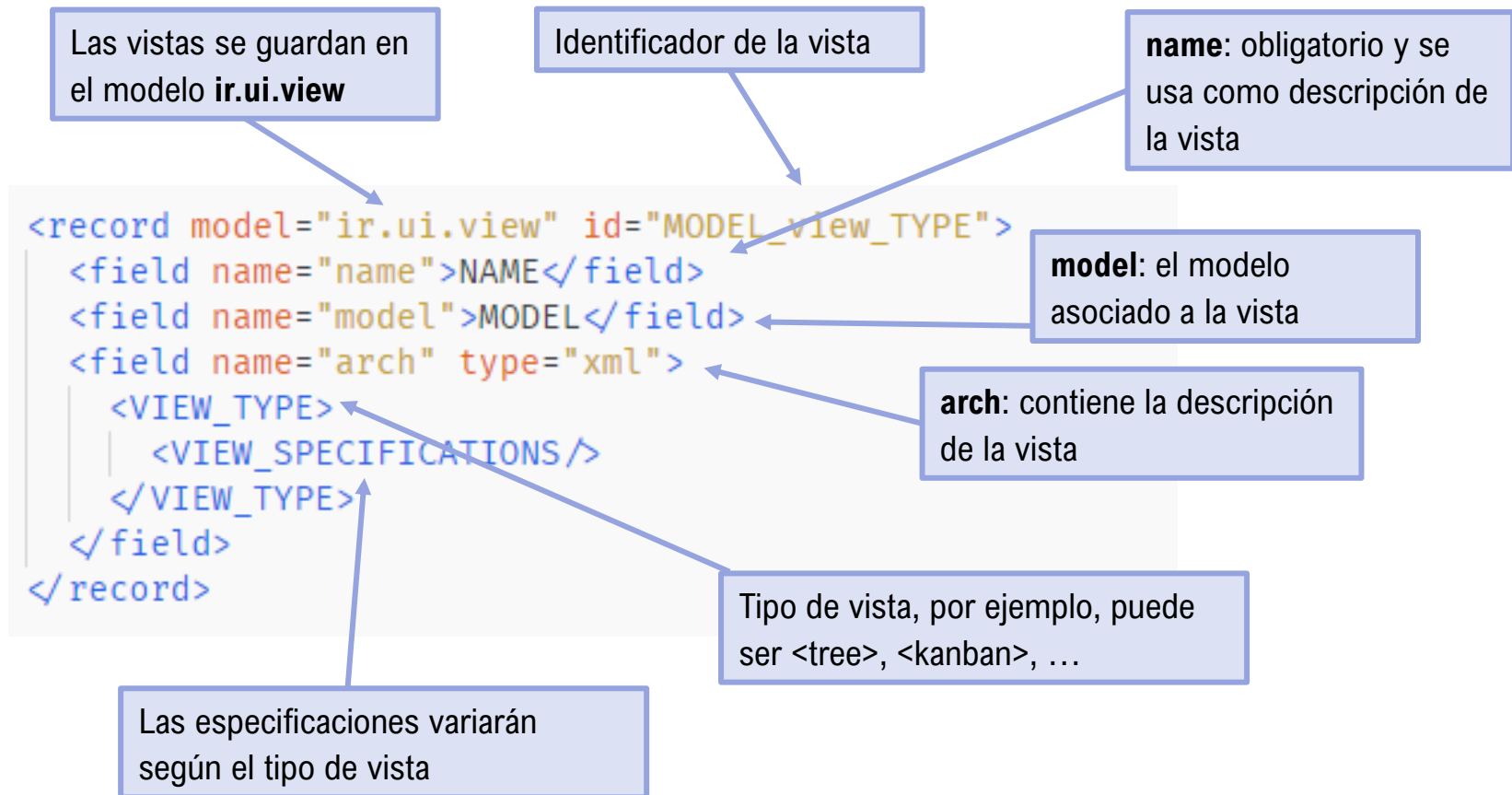
En la documentación oficial de Odoo puedes verlas en detalle:

<https://www.odoo.com/documentation/16.0/developer/reference/backend/views.html#view-types>

- ▼ View types
 - Activity
 - Calendar
 - Cohort
 - > Form
 - Gantt
 - Graph
 - > Grid
 - Kanban
 - List
 - > Map
 - Pivot
 - QWeb
 - > Search

Estructura general de una vista

Por norma general, todas las vistas comparten la siguiente estructura.



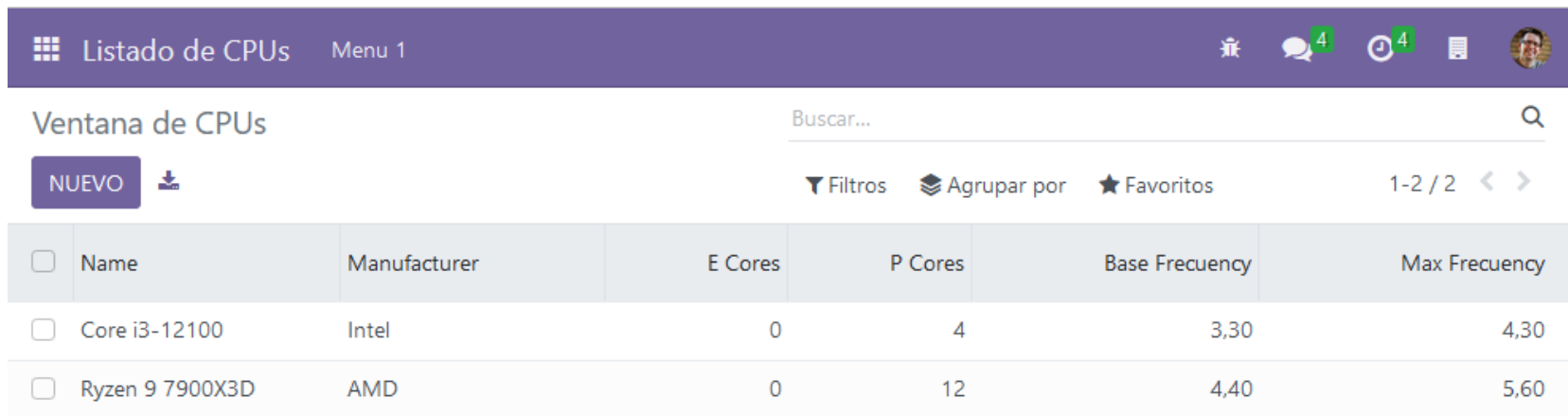
6

VISTAS DE TIPO LISTA



Vista tree (lista)

La **vista tree** o de **lista** es de las más utilizadas. Muestra los elementos del modelo en una tabla donde cada fila corresponde a un elemento y cada columna a un campo del mismo.



The screenshot shows a web application interface for a CPU list. At the top is a purple header bar with a grid icon, the text 'Listado de CPUs', 'Menu 1', and several utility icons. Below the header, the main content area has a title 'Ventana de CPUs' and a search bar labeled 'Buscar...'. On the left, there is a 'NUEVO' button and a download icon. On the right, there are links for 'Filtros', 'Agrupar por', and 'Favoritos', along with a pagination indicator '1-2 / 2'. The main part of the interface is a table with columns for Name, Manufacturer, E Cores, P Cores, Base Frequency, and Max Frequency. Two rows are visible: 'Core i3-12100' by Intel and 'Ryzen 9 7900X3D' by AMD. Each row has a checkbox in the first column.

<input type="checkbox"/>	Name	Manufacturer	E Cores	P Cores	Base Frequency	Max Frequency
<input type="checkbox"/>	Core i3-12100	Intel	0	4	3,30	4,30
<input type="checkbox"/>	Ryzen 9 7900X3D	AMD	0	12	4,40	5,60

Admite un gran número de atributos que permite personalizar la presentación.

NOTA: para los ejemplos siguientes he creado un módulo nuevo con el modelo de la derecha.

Igualmente, solo incluiré capturas del contenido de la etiqueta <tree> de la vista. Pero la vista deberá tener también definidos los menús y el *action window*.

Puedes basarte en el ejemplo de la lista de tareas para crear ese contenido si quieres ir probando el código que voy a exponer aquí.

```
class cpus(models.Model):
    _name = 'cpus.cpus'
    _description = 'cpus.cpus'

    name = fields.Char()
    manufacturer = fields.Char()
    total_cores = fields.Integer()
    p_cores = fields.Integer()
    e_cores = fields.Integer()
    base_frecuency = fields.Float()
    max_frecuency = fields.Float()
```

Atributo de <tree>: **editable**="top|bottom"

Por defecto, hacer click en una fila abre la vista de formulario correspondiente para modificar los datos.

Este atributo modifica el comportamiento de forma que se puede **editar directamente** en la vista de lista.

```
<tree editable='bottom'>
  <field name="name" />
  <field name="manufacturer" />
  <field name="e_cores" />
```

Por otro lado, al hacer click en *Nuevo* permite añadir los datos del nuevo registro en la propia lista en lugar de abrir un formulario

Ventana de CPUs

GUARDAR

DESCARTAR



<input type="checkbox"/>	Name	Manufacturer
<input type="checkbox"/>	Core i3-12100	Intel
<input type="checkbox"/>	Ryzen 9 7900X3D	AMD

Intel


Los valores *top* y *bottom* hacen referencia a dónde se crea el registro nuevo cuando pulsamos en *Nuevo*.

Es obligatorio poner uno de los dos valores.

```
<tree editable='top'>
  <field name="name" />
  <field name="manufacturer" />
  <field name="e_cores" />
```


```
<tree editable='bottom'>
  <field name="name" />
  <field name="manufacturer" />
  <field name="e_cores" />
```

Ventana de CPUs

GUARDAR DESCARTAR 

<input type="checkbox"/>	Name	Manufacturer
<input type="checkbox"/>		
<input type="checkbox"/>	Core i3-12100	Intel
<input type="checkbox"/>	Ryzen 9 7900X3D	AMD
<input type="checkbox"/>		

Ventana de CPUs

GUARDAR DESCARTAR 

<input type="checkbox"/>	Name	Manufacturer
<input type="checkbox"/>	Core i3-12100	Intel
<input type="checkbox"/>	Ryzen 9 7900X3D	AMD
<input type="checkbox"/>		

Atributo de <tree>: **default_order**

Permite modificar el orden en el que se muestran los registros.

El valor es una lista separada por comas de campos, seguidos de la palabra **desc** si se quiere que estén en orden inverso.

```
<tree default_order="max_frecuency desc">  
  <field name="name" />  
  <field name="manufacturer" />
```

Ventana de CPUs

Buscar...

NUEVO



Filtros

Ag

<input type="checkbox"/>	Name	Manufacturer	E Cores	P Cores	Base Frequency	▼	Max Frequency
<input type="checkbox"/>	Ryzen 9 7900X3D	AMD	0	12	4,40		5,60
<input type="checkbox"/>	Core i9-13900K	Intel	16	8	3,00		5,40
<input type="checkbox"/>	Ryzen 7 5700G	AMD	0	8	3,80		4,60
<input type="checkbox"/>	Core i3-12100	Intel	0	4	3,30		4,30
<input type="checkbox"/>	Ryzen 5 4600G	AMD	0	6	3,70		4,20

Atributo de <tree>: **decoration-{\$name}**

Permite cambiar el color de una fila en base al valor de un atributo.

{ \$name } puede ser **bf** (bold), **it** (italic) o cualquier color contextual de Bootstrap: **danger**, **info**, **muted**, **primary**, **success** o **warning**.

El valor de esta propiedad tiene que ser una expresión en Python.

Por ejemplo, si queremos que ponga resaltados los procesadores cuyo fabricante sea AMD pondríamos el atributo de la forma:

```
decoration-danger="manufacturer=='AMD'"
```

Observa cómo se combinan comillas simples y dobles para que sea sintácticamente correcto. Si fuera un número no serían necesarias las comillas

```
<tree decoration-danger="manufacturer='AMD'">  
  <field name="name" />  
  <field name="manufacturer" />
```

El operador de igualdad es == (fíjate que son dos caracteres=)

Ventana de CPUs

NUEVO



<input type="checkbox"/>	Name	Manufacturer	E Cores	P Cores
<input type="checkbox"/>	Core i3-12100	Intel	0	4
<input type="checkbox"/>	Ryzen 9 7900X3D	AMD	0	12
<input type="checkbox"/>	Core i9-13900K	Intel	16	8
<input type="checkbox"/>	Ryzen 7 5700G	AMD	0	8
<input type="checkbox"/>	Ryzen 5 4600G	AMD	0	6

Atributo de <tree>: **limit**

Indica el número de filas que se mostrarán en cada página.

```
<tree limit="3">
  <field name="name" />
  <field name="manufacturer" />
```

Navegación entre páginas

Ventana de CPUs

NUEVO



Filtros

Agrupar por

Favoritos

1-3 / 5



<input type="checkbox"/>	Name	Manufacturer	E Cores	P Cores	Base Frequency	Max Frequency
<input type="checkbox"/>	Core i3-12100	Intel	0	4	3,30	4,30
<input type="checkbox"/>	Ryzen 9 7900X3D	AMD	0	12	4,40	5,60
<input type="checkbox"/>	Core i9-13900K	Intel	16	8	3,00	5,40

Las **vistas tree** pueden tener dos tipos de nodos hijo: **<field>** y **<button>**

<field>

Define una columna donde el campo correspondiente se muestra para cada registro. Admite un gran número de atributos

Atributo de <field>: *name*

El que hemos usado hasta ahora, hace referencia al campo correspondiente en el modelo.

Atributo de <field>: *string*

El título de la columna. Por defecto utiliza el valor de ***string*** si se ha indicado en el modelo. Si no se indica ni la propiedad en el modelo ni el atributo en la vista toma el nombre del campo. Por ejemplo: p_cores lo toma como P Cores

Modelo

```
total_cores = fields.Integer()  
p_cores = fields.Integer(string='Cores de Rendimiento')  
e_cores = fields.Integer()  
base_frecuency = fields.Float()
```

Vista

```
<field name="manufacturer" />  
<field name="total_cores" />  
<field name="e_cores" string="Cores de Eficiencia" />  
<field name="p_cores" />  
<field name="base_frecuency" />
```

No hay *string* ni en modelo ni vista, por lo que muestra
Total Cores

Toma el valor indicado
en la vista

Toma el valor indicado en el
modelo porque en la vista no
hay ninguno definido

Total Cores	Cores de Eficiencia	Cores de Rendimiento
4	0	4
12	0	12
24	16	8

Atributo de <field>: invisible

En ocasiones necesitamos tener un campo en la vista, pero no queremos que se muestre. Con este atributo impediremos que se muestre.

```
<tree decoration-dánger='total cores ≥ 16'>
  <field name="name" />
  <field name="manufacturer" />
  <field name="total_cores" invisible="1" />
  <field name="e_cores" string="Cores de Eficiencia" />
  <field name="p_cores" string="Cores de Rendimiento" />
  <field name="base_frequency" />
```

Aquí necesitamos *total_cores* para colorear las CPUs con más de 16 en total, pero no queremos que se muestre

Observa cómo hay que poner `invisible="1"` para que el XML sea correcto

Ventana de CPUs

NUEVO



Buscar...

Filtros

Agrupar por

Favoritos

1-5 / 5

<input type="checkbox"/>	Name	Manufacturer	Cores de Eficiencia	Cores de Rendimiento	Base Frequency	Max Frequency
<input type="checkbox"/>	Core i3-12100	Intel	0	4	3,30	4,30
<input type="checkbox"/>	Ryzen 9 7900X3D	AMD	0	12	4,40	5,60
<input type="checkbox"/>	Core i9-13900K	Intel	16	8	3,00	5,40
<input type="checkbox"/>	Ryzen 7 5700G	AMD	0	8	3,80	4,60
<input type="checkbox"/>	Ryzen 5 4600G	AMD	0	6	3,70	4,20

Atributo de <field>: **sum** y **avg**

Con estos atributos se muestra un total de los valores de la columna, en el caso de **sum** la suma de dichos valores y en el caso de **avg** la media.

```
<field name="max_frecuency" avg="1" />
```

Cores de Eficiencia	Cores de Rendimiento	Base Frecuency	Max Frecuency
0	4	3,30	4,30
0	12	4,40	5,60
16	8	3,00	5,40
0	8	3,80	4,60
0	6	3,70	4,20
			4,82

Nuevamente, hay que poner avg="1" para que el XML sea correcto

Algo importante es que, para el cálculo, se usan solo los registros mostrados en la página

Atributo de <field>: **decoration-{\$name}**

Su uso es análogo a este mismo atributo en la etiqueta <field>, pero solo coloreará el campo correspondiente.

```
<field name="max_frequency"
      decoration-success="max_frequency<=5.0" />
```

Observa que aquí utilizo la **referencia a entidad XML** como alternativa al uso del símbolo <

Rendimiento	Base Frequency	Max Frequency
4	3,30	4,30
12	4,40	5,60
8	3,00	5,40
8	3,80	4,60
6	3,70	4,20

Referencias a entidades en XML		
Carácter	Entidad	Referencia a entidad
< (menor que)	lt (less than)	<
> (mayor que)	gt (greater than)	>
" (comilla doble)	quot (quotation mark)	"
' (comilla simple)	apos (apostrophe)	'
& (ampersand)	amp (ampersand)	&

<https://www.abrirllave.com/xml/referencias-a-entidades.php>



Atributo de <field>: **widget**

Los **widgets** permiten modificar la presentación de un campo. Hay muchos tipos de widgets para cada tipo de datos, por lo que no vamos a entrar en detalle con todos ellos.

En la siguiente URL tienes una relación con todos los tipos de datos que hay en Odoo 16 y los widgets disponibles para cada uno de ellos.






<https://www.cybrosys.com/blog/field-types-and-widgets-in-odoo-16>



A modo de ejemplo, vamos a ver tres *widgets*: **CopyClipboardChar**, **progressbar** y **percentpie**, aunque no estaría mal que hicieras pruebas para ver cómo funcionan otros widgets.

El widget **CopyClipboardChar** se aplica a los campos de tipo texto y añade un botón a la derecha del mismo que permite copiar el contenido de dicho campo en el portapapeles.

```
<tree>  
  <field name="name" widget="CopyClipboardChar" />
```

<input type="checkbox"/>	Name		Manufacturer	Cores de Eficiencia	Cores de Rendimiento
<input type="checkbox"/>	<input type="text" value="Core i3-12100"/>	 COPIAR	Intel	0	4
<input type="checkbox"/>	<input type="text" value="Ryzen 9 7900X3D"/>	 COPIAR	AMD	0	12
<input type="checkbox"/>	<input type="text" value="Core i9-13900K"/>	 COPIAR	Intel	16	8
<input type="checkbox"/>	<input type="text" value="Ryzen 7 5700G"/>	 COPIAR	AMD	0	8
<input type="checkbox"/>	<input type="text" value="Ryzen 5 4600G"/>	 COPIAR	AMD	0	6

Los widgets **progressbar** y **percentpie** son muy similares, ya que ambos permiten mostrar un valor numérico como un porcentaje en una barra de progreso o en un gráfico circular.

Estos widgets esperan que el valor numérico esté comprendido entre 0 y 100, así que muchas veces se utilizan en combinación con un campo calculado.

Ejemplo: vamos a modificar nuestro código para que las frecuencias de las CPUs se muestren como un porcentaje sobre un máximo de frecuencia (por ejemplo, supondremos un máximo de 6GHz que equivaldrá al 100%)

Realizo estos cambios
en el **modelo**

Añado dos campos calculados
en el modelo, uno para cada
frecuencia

```
base_frequency = fields.Float()
max_frequency = fields.Float()
percent_max_frequency = fields.Float( compute="_percent_max_frec", store=True)
percent_base_frequency = fields.Float( compute="_percent_base_frec", store=True)

# Pasamos la frecuencia a un porcentaje donde 0 GHz es 0% y 6 GHz es 100%
@api.depends('max_frequency')
def _percent_max_frec(self):
    for record in self:
        record.percent_max_frequency = (float(record.max_frequency)/6)*100

@api.depends('base_frequency')
def _percent_base_frec(self):
    for record in self:
        record.percent_base_frequency = (float(record.base_frequency)/6)*100
```

Como la frecuencia es un
valor entre 0 y 6, lo paso a un
valor entre 0 y 100

Y estos en la vista

Ya no necesito mostrar el campo original, lo he comentado

```
<!-- <field name="base_frequency" /> -->
<field name="percent_base_frequency"
| | | widget="progressbar"
| | | string="Frecuencia base" />

<!-- <field name="max_frequency" /> -->
<field name="percent_max_frequency"
| | | widget="percentpie"
| | | string="Frecuencia máxima" />
```

Como ejemplo, muestro una como *progressbar*



Y el otro como *percentpie*



Cambio el nombre del título de la columna para que sea más representativo

Ventana de CPUs

NUEVO



Buscar...



Filtros

Agrupar por

★ Favoritos

1-5 / 5 < >

<input type="checkbox"/>	Name	Manufacturer	Cores de Eficiencia	Cores de Rendimiento	Frecuencia base	Frecuencia máxima
<input type="checkbox"/>	<div>Core i3-12100</div> <div>COPIAR</div>	Intel	0	4	<div><div></div></div> 55 %	<div>72%</div> Frecuencia máxima
<input type="checkbox"/>	<div>Ryzen 9 7900X3D</div> <div>COPIAR</div>	AMD	0	12	<div><div></div></div> 73 %	<div>93%</div> Frecuencia máxima
<input type="checkbox"/>	<div>Core i9-13900K</div> <div>COPIAR</div>	Intel	16	8	<div><div></div></div> 50 %	<div>90%</div> Frecuencia máxima
<input type="checkbox"/>	<div>Ryzen 7 5700G</div> <div>COPIAR</div>	AMD	0	8	<div><div></div></div> 63 %	<div>77%</div> Frecuencia máxima
<input type="checkbox"/>	<div>Ryzen 5 4600G</div> <div>COPIAR</div>	AMD	0	6	<div><div></div></div> 62 %	<div>70%</div> Frecuencia máxima

Atributo de <field>: `attrs`

Permite aplicar reglas condicionales a los elementos de la interfaz.

Estas reglas determinan cómo y cuándo un elemento debe estar visible, habilitado o adquirir ciertas características.

El valor es una expresión basada en un diccionario que define el comportamiento del elemento según las condiciones especificadas.

El formato básico es:

```
attrs="{ 'atributo': [ ('campo', 'operador', 'valor')] }"
```

atributo: define qué acción aplicar al elemento.

Puede ser:

- **invisible:** oculta el elemento si la condición se cumple.
- **readonly:** solo lectura si la condición se cumple
- **required:** obligatorio si la condición se cumple

campo: nombre del campo del modelo a evaluar

```
attrs="{ 'atributo': [ ('campo', 'operador', 'valor') ] }"
```

operador: un operador Python que puede ser: ==, !=, >, <, >=, <=, in o not in

valor: valor contra el que se compara el campo

Ejemplos:

```
<!-- Se oculta el campo si el valor de state es done -->
<field name="partner_id"
  | | | attrs="{ 'invisible': [('state', '=', 'done')] }"/>

<!-- Hace el campo email obligatorio si is_company es True -->
<field name="email"
  | | | attrs="{ 'required': [('is_company', '=', True)] }"/>

<!-- Condiciones combinadas. Si las condiciones están en una lista se combinan con AND -->
<field name="name"
  | | | attrs="{ 'invisible': [('state', '=', 'done'), ('priority', '=', 'low')] }"/>

<!-- También se pueden combinar con OR usando |. Observa bien la sintaxis -->
<field name="description"
  | | | attrs="{ 'invisible': ['|', ('state', '=', 'done'), ('priority', '=', 'low')] }"/>
```

<button>

El nodo **<button>** permite agregar botones dentro de las filas de la vista tree, habilitando acciones directas sobre los registros.

Los botones pueden ejecutar **métodos específicos del modelo** o realizar **acciones predefinidas**.

Ejemplo: botón que ponga de oferta un microprocesador indicando un 10% de descuento.

Añadimos primero un campo *discount* para almacenar el descuento

```
discount = fields.Float('Descuento')
```

En la vista añadimos el campo *discount* para ver cómo hace efecto y un nodo de tipo `<button>`

Nombre del método que se ejecutará al hacer click. El método se tiene que definir en el modelo

El tipo de acción a realizar. Puede ser **object** si se va a ejecutar un método definido en el modelo, o **action** si queremos ejecutar una acción específica

Texto que se mostrará en el botón

Le podemos dar estilos CSS










Y también podemos indicar un icono

```
<field name="discount"/>
<button name="action_offer"
  type="object"
  string="Oferta"
  class="btn-primary"
  icon="fa-gift"/>
```

Los iconos son los disponibles en **Font Awesome**.

Algunos de los más comunes son:

Acciones Generales

- `fa-check` :  Aprobación o confirmación.
- `fa-times` :  Cancelación o cierre.
- `fa-edit` :  Edición.
- `fa-trash` :  Eliminación.
- `fa-save` :  Guardar.
- `fa-plus` :  Añadir o crear.
- `fa-minus` :  Eliminar o reducir.
- `fa-arrow-up` :  Subir.
- `fa-arrow-down` :  Bajar.

Navegación y Visualización

- `fa-search`: 🔍 Buscar.
- `fa-eye`: 👁 Mostrar o visualizar.
- `fa-eye-slash`: 🚫 👁 Ocultar.
- `fa-bars`: ☰ Menú.





Archivos y Documentos

- `fa-file`: 📄 Documento.
- `fa-upload`: ⬆ Subir archivo.
- `fa-download`: ⬇ Descargar archivo.
- `fa-folder`: 📁 Carpeta.
- `fa-folder-open`: 📁 Carpeta abierta.





Indicadores de Estado

- `fa-info-circle`: ⓘ Información.
- `fa-warning` o `fa-exclamation-triangle`: ⚠ Advertencia.
- `fa-check-circle`: ✔ Aprobación en estado.
- `fa-times-circle`: ✖ Error o rechazo.





Gestión de Usuarios

- `fa-user` :  Usuario.
- `fa-users` :  Grupo de usuarios.
- `fa-sign-out` :  Salir.
- `fa-sign-in` :  Iniciar sesión.

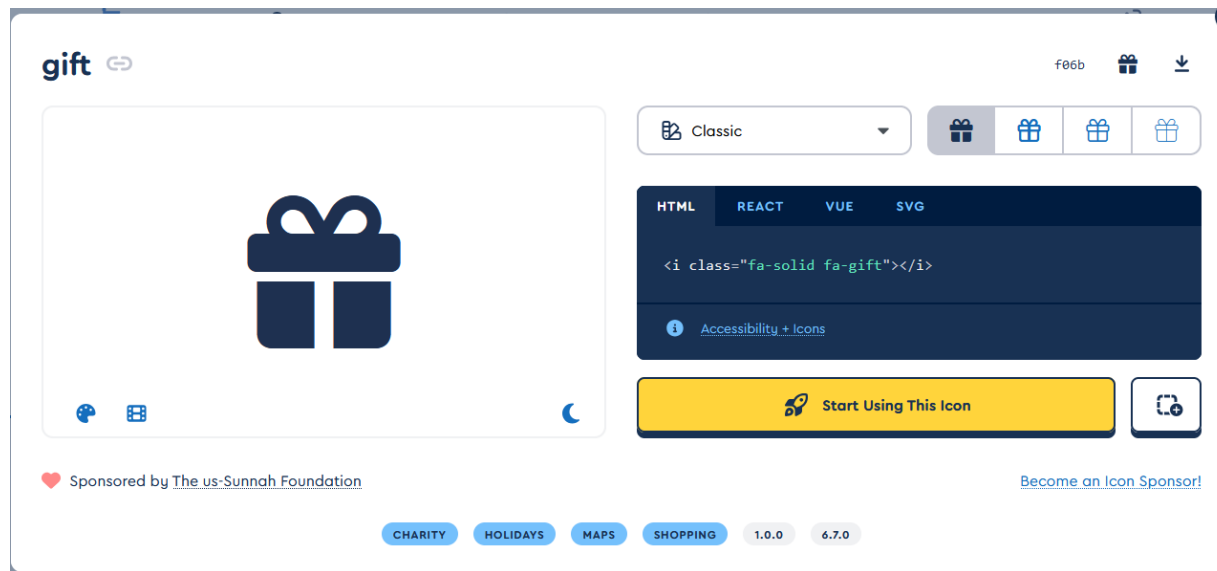
Pagos y Finanzas

- `fa-money` :  Dinero.
- `fa-credit-card` :  Tarjeta de crédito.
- `fa-bank` :  Banco.
- `fa-shopping-cart` :  Compra o pedido.

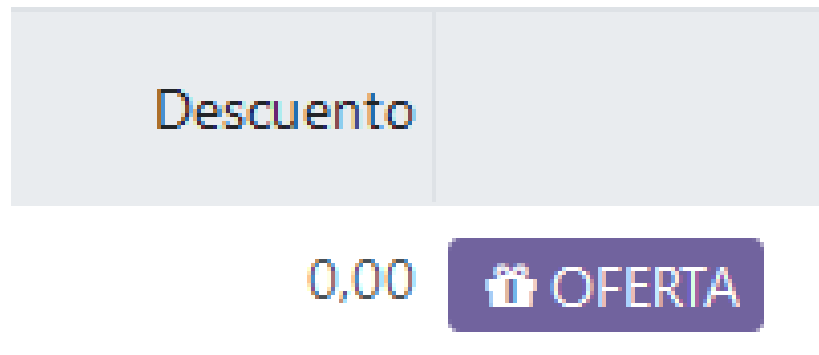
Otras Categorías

- `fa-cogs` :  Configuración.
- `fa-calendar` :  Fecha o calendario.
- `fa-envelope` :  Correo electrónico.
- `fa-phone` :  Teléfono.

En este caso he buscado uno directamente en la página Web de Font Awesome (<https://fontawesome.com/>) y he mirado la clase que hay que incluir.



Con esto ya tenemos definido el botón.



Ahora nos queda definir el método en el modelo. En este ejemplo vamos a hacer que, al pulsar el botón, se actualice el campo *discount* a un valor de 0.10

```
def action_offer(self):  
    for cpu in self:  
        cpu.discount = 0.10;
```

Como en otros métodos (por ejemplo, en los campos calculados), comenzamos con un iterador



En este ejemplo simplemente actualizamos el valor del campo *discount*

Y ya estaría, ahora, si hacemos click en el botón veremos como el campo de descuento se actualiza automáticamente a 0,10

cpu_management window

NUEVO



<input type="checkbox"/>	Nombre	^	Fabricante	Número de núcleos	Descuento	
<input type="checkbox"/>	Intel i7		Intel	24	0,00	 OFERTA
<input type="checkbox"/>	Ryzen 7		AMD	12	0,10	 OFERTA

Ejemplo: botón que cambia de color según el estado del registro.

Para ver este ejemplo modificaremos un campo al modelo que indicará si un producto está disponible o no. Según esté disponible o no se mostrará un botón para alternar la disponibilidad.

Comenzamos cambiando el modelo.

El nuevo campo se llamará *available* y, por defecto, tendrá el valor *no*

```
available = fields.Char(default="no")
```

```
def action_make_available(self):  
    self.available = 'yes'
```

```
def action_make_unavailable(self):  
    self.available = 'no'
```

Creamos ya las funciones que definirán el comportamiento de los botones: una para hacer el producto disponible y otra para lo contrario

Y ahora la vista

Mostramos el nuevo campo

Método que se ejecutará al pulsar el botón

Los estilos que se aplicarán al botón (clases Bootstrap)

Se ejecutará una función en lugar de una acción Odoo

Oculto el botón si el producto ya está disponible





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



<button name="action_make_available"
         type="object"
         string="Hacer disponible"
         class="btn-success"
         attrs="{ 'invisible': [('available', '!=', 'no')] }"/>

<button name="action_make_unavailable"
         type="object"
         string="Quitar disponibilidad"
         class="btn-danger"
         attrs="{ 'invisible': [('available', '!=', 'yes')] }"/>
```

Creo dos botones: uno para cada estado de disponibilidad

El resultado final sería como este, al pulsar un botón modificará el valor de *available* y, además, hará que cambie el botón

<input type="checkbox"/>	Nombre	Fabricante	Número de núcleos	Descuento		Available	
<input type="checkbox"/>	Ryzen 7	AMD	12	0,10	 OFERTA	no	 Hacer disponible
<input type="checkbox"/>	Intel i7	Intel	24	0,00	 OFERTA	yes	 Quitar disponibilidad

Available	
no	 Hacer disponible
yes	 Quitar disponibilidad

7

VISTAS DE TIPO FORM



Vista form

La **vista form** es utilizado cuando se quieren mostrar los campos de un único registro.

Se definen mediante la etiqueta XML **<form>**

La vista **form** tiene una serie de **elementos estructurales** y otros **semánticos**.

Elementos estructurales:

- notebook
- group
- newline
- separator
- sheet
- header

Elementos semánticos:

- button
- field
- label

La vista de formulario se crea de forma análoga a la vista de lista pero utilizando la etiqueta **form**.

Se puede definir en el mismo fichero que la vista de lista o en un fichero diferente.

```
<record model="ir.ui.view" id="cpu_management.form">
  <field name="name">cpu_management form</field>
  <field name="model">cpu_management.cpu</field>
  <field name="arch" type="xml">
    <form string="Producto">
      <!-- Aquí pondremos la estructura del formulario -->
      <field name="name"/>
    </form>
  </field>
</record>
```

<field>

Al igual que en la vista de tipo lista, identifican cada uno de los campos del modelo que se mostrará en la vista.

```
<form string="Producto">
  <field name="name"/>
  <field name="manufacturer"/>
  <field name="total_cores"/>
  <field name="p_cores"/>
  <field name="e_cores"/>
</form>
```

Por defecto, los campos los distribuye en columnas indicando únicamente el valor del cada campo.

cpu_management window / Ryzen 7

Ryzen 7

AMD

12

0

0

Si queremos que se muestre una etiqueta deberemos utilizar **<label>**

El atributo **for** es obligatorio y hace referencia al nombre o identificador de la etiqueta que acompaña

Con **string** indicamos el texto de la etiqueta

```
<label for="name" string="Nombre"/>
<field name="name"/>

<label for="man" string="Fabricante"/>
<field name="manufacturer" id="man"/>
```

Si **for** señala un identificador, este debe definirse en el field mediante el atributo **id**

<sheet>

Se puede usar como hijo directo de <form> y proporciona una vista más estrecha y *responsive* para el formulario.

Sin la etiqueta
<sheet>

cpu_management window / Ryzen 7

Acción

1 / 1

< >

Nuevo

CORES

Número de núcleos 12

P Cores 0

E Cores 0

Con la etiqueta
<sheet>

cpu_management window / Ryzen 7

Acción

1 / 1

< >

Nuevo

CORES

Número de núcleos 12

P Cores 0

E Cores 0

<group>

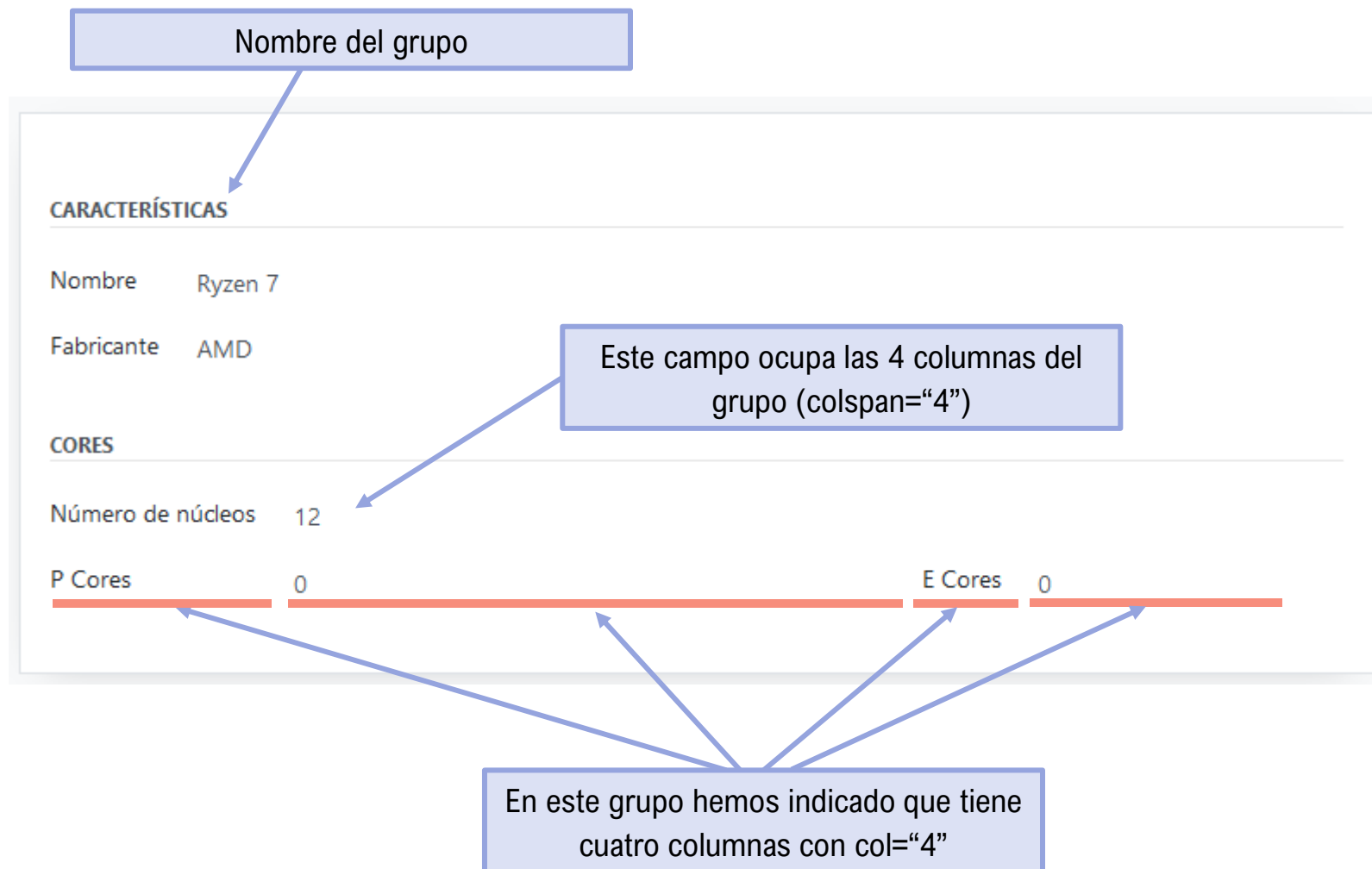
Organiza los campos en columnas. Algo importante es que cuando tenemos campos dentro de un grupo incluyen la etiqueta por defecto

Se puede asignar nombre al grupo con la etiqueta **<group>**

Con **<col>** indicamos en cuántas columnas se distribuirá el grupo.

```
<group string="Características">
  <field name="name"/>
  <field name="manufacturer"/>
</group>
<group string="Cores" col="4">
  <field name="total_cores" colspan="4"/>
  <field name="p_cores"/>
  <field name="e_cores"/>
</group>
```

<colspan> indica cuántas columnas ocupará un campo



<separator>

Espaciado horizontal en el formulario, admite el atributo **string** para indicar un título

```
<group>
  <field name="name"/>
  <field name="manufacturer" id="man"/>
</group>
<separator string="Características técnicas"/>
<group>
  <field name="total_cores"/>
  <field name="p_cores"/>
  <field name="e_cores"/>
</group>
```

cpu_management window / Ryzen 7

Nombre Ryzen 7

Fabricante AMD

CARACTERÍSTICAS TÉCNICAS

Número de núcleos 12

P Cores 0

E Cores 0

<notebook> y <page>

Define una sección con pestañas. Cada pestaña se define mediante <page>

```
<notebook>
  <page string="Características">
    <group>
      <field name="name"/>
      <field name="manufacturer" id="man"/>
    </group>
  </page>

  <page string="Cores">
    <group>
      <field name="total_cores"/>
      <field name="p_cores"/>
      <field name="e_cores"/>
    </group>
  </page>
</notebook>
```

Características	Cores
Nombre	Ryzen 7
Fabricante	AMD

Características	Cores
Número de núcleos	12
P Cores	0
E Cores	0