# Minipresentación Lógica Aplicación

Algunas capturas del libro recomendado

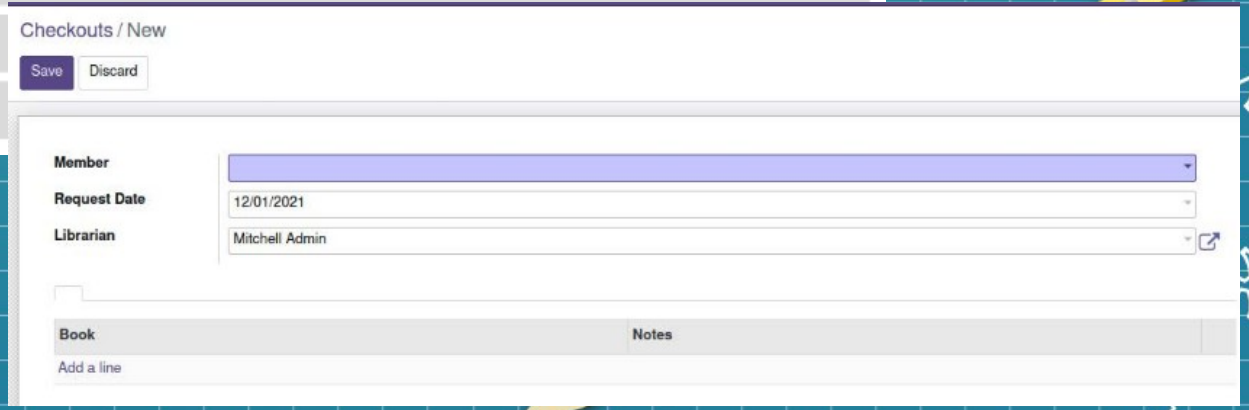
# Modelo cabeceras líneas

```
    line_ids = fields.One2many(
        "library.checkout.line",
        "checkout_id",
        string="Borrowed Books",
    )
class CheckoutLine(models.Model):
    _name = "library.checkout.line"
    _description = "Checkout Request Line"

    checkout_id = fields.Many2one(
        "library.checkout",
        required=True,
    )
    book_id = fields.Many2one("library.book",
      required=True)
    note = fields.Char("Notes")
```

# Modelo Cabecera Lineas

```
<notebook>
  <page name="lines">
    <field name="line_ids">
      <tree editable="bottom">
        <field name="book_id" />
        <field name="note" />
      </tree>
    </field>
  </page>
</notebook>
```



# Shell odoo

```
usuario@baseserver22:~$ cat shellodoo.sh
sudo -u odoo odoo shell -c /etc/odoo/odoo.conf -d emp02
usuario@baseserver22:~$ bash shellodoo.sh
[sudo] password for usuario:
env: <odoo.api.Environment object at 0x7fb19d0fdae0>
odoo: <module 'odoo' from '/usr/lib/python3/dist-packages/odoo/__init__.py'>
openerp: <module 'odoo' from '/usr/lib/python3/dist-packages/odoo/__init__.py'>
self: res.users(1,)
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(Console)
>>> lista=self.env['lista_tareas.lista']
>>> lista
lista_tareas.lista()
>>> todastareas=lista.search([])
>>> todastareas
lista_tareas.lista(2, 4, 5, 6, 11, 8, 9, 10, 7)
>>>
```

# Recordsets

```
>>> self.env['res.partner']
res.partner
>>> self.env['res.partner'].search([['is_company', '=', True], ['customer', '=', True]])
res.partner(7, 18, 12, 14, 17, 19, 8, 31, 26, 16, 13, 20, 30, 22, 29, 15, 23, 28, 74)
```

```
>>> # searches the current model
>>> self.search([('is_company', '=', True), ('customer', '=', True)])
res.partner(7, 18, 12, 14, 17, 19, 8, 31, 26, 16, 13, 20, 30, 22, 29, 15, 23, 28, 74)
>>> self.search([('is_company', '=', True)], limit=1).name
'Agrolait'
```

```
>>> self.browse([7, 18, 12])
res.partner(7, 18, 12)
```

Los “recordsets” son iterables como una lista/array y tienen, además, operaciones de conjuntos que permiten facilitar el trabajo con ellos:

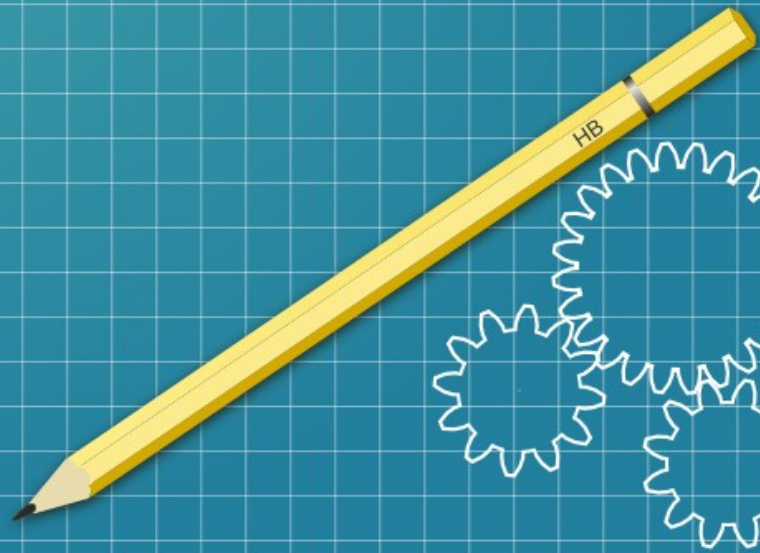
- **record in set:** retorna True si “record” está en el conjunto (set).
- **set1 | set2:** unión de conjuntos (sets). También funciona el signo +.
- **set1 & set2:** intersección de conjuntos (sets).
- **set1 - set2:** diferencia de conjuntos (sets).

# Operaciones recordsets

```
self.env['mail.channel'].browse(self.group_private.id).unlink()
```

```
>>> self.create({'name': "New Name"})
res.partner(78)
```

```
self.write({'name': "Newer Name"})
```



# Write con campos Many2many

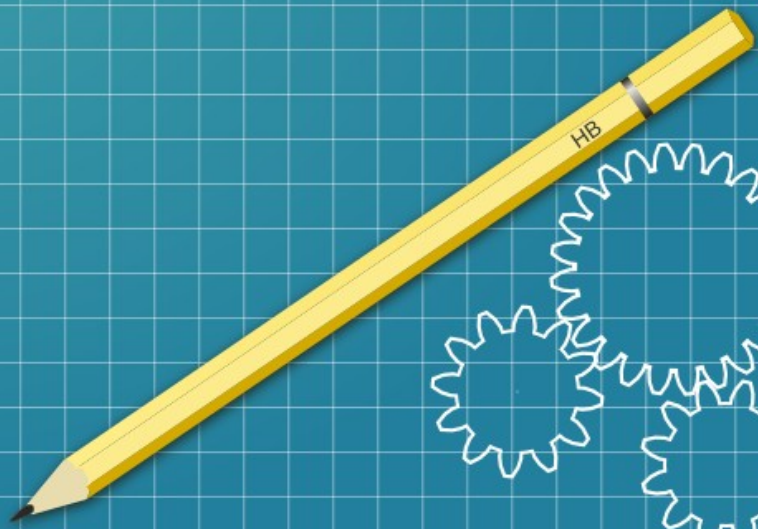
- **(0,_ ,{'field': value}):** crea un nuevo registro y lo vincula.
- **(1,id,{'field': value}):** actualiza los valores de un registro ya vinculado.
- **(2,id,_):** desvincula y elimina el registro.
- **(3,id,_):** desvincula, pero no elimina el registro de la relación.
- **(4,id,_):** vincula un registro que ya existe.
- **(5,_,_):** desvincula, pero no elimina todos los registros.
- **(6,_,[ids]):** reemplaza la lista de registros.

```
# Crear dos nuevas etiquetas y asociarlas a los productos
nuevas_etiquetas = [
    (0, 0, {'name': 'Nueva Etiqueta 1'}),
    (0, 0, {'name': 'Nueva Etiqueta 2'})
]
productos.write({'etiqueta_ids': nuevas_etiquetas})
```

```
# Añadir una etiqueta existente a los productos
etiqueta_existente_id = 456
productos.write({'etiqueta_ids': [(4, etiqueta_existente_id)]})
```

```
# Eliminar todas las etiquetas asociadas a los productos
productos.write({'etiqueta_ids': [(5)]})
```

```
# Reemplazar todas las etiquetas con un nuevo conjunto
nuevas_etiquetas_ids = [123, 456, 789]
productos.write({'etiqueta_ids': [(6, 0, nuevas_etiquetas_ids)]})
```

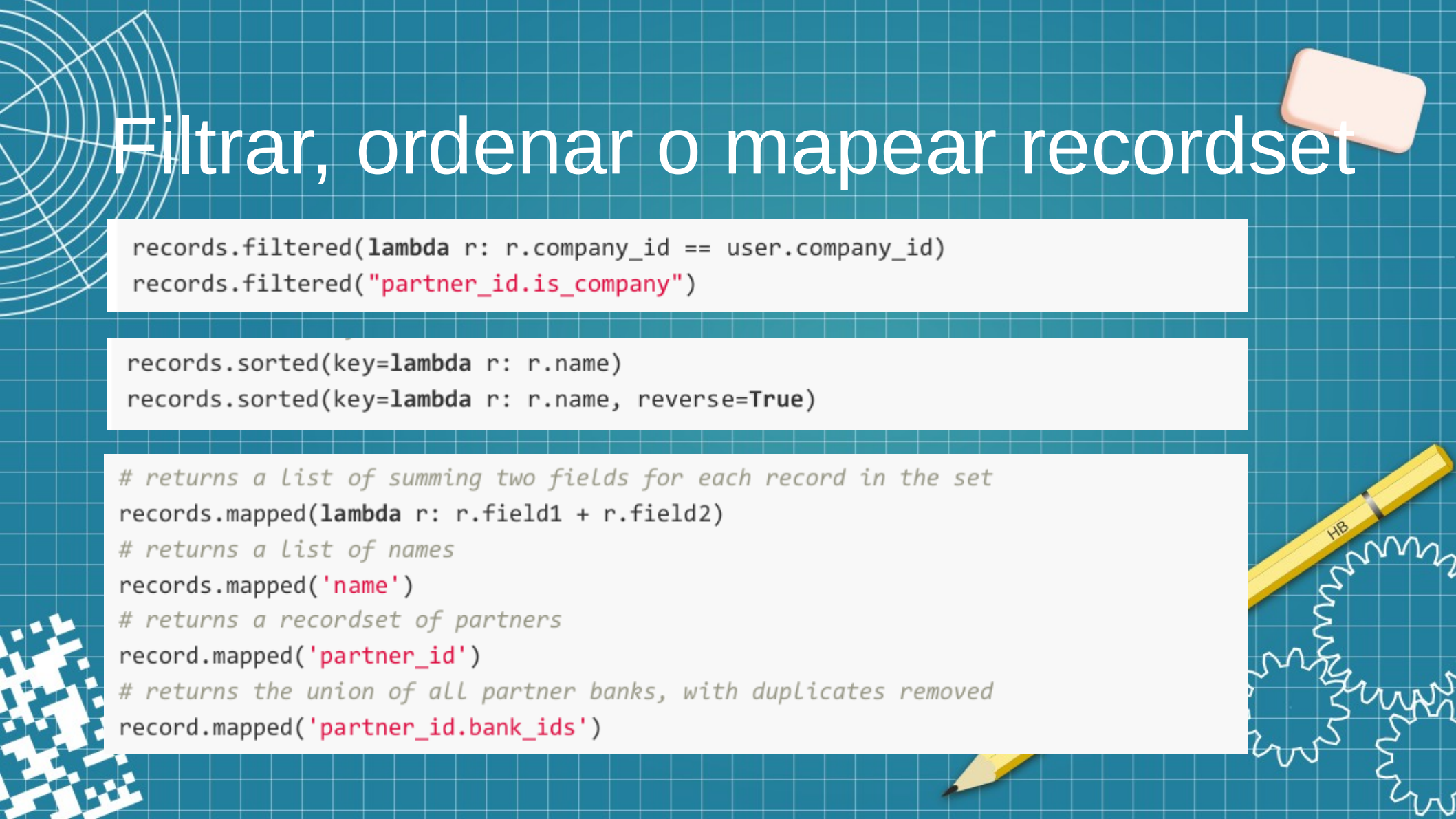


# Filtrar, ordenar o mapear recordset

```
records.filtered(lambda r: r.company_id == user.company_id)
records.filtered("partner_id.is_company")
```

```
records.sorted(key=lambda r: r.name)
records.sorted(key=lambda r: r.name, reverse=True)
```

```
# returns a list of summing two fields for each record in the set
records.mapped(lambda r: r.field1 + r.field2)
# returns a list of names
records.mapped('name')
# returns a recordset of partners
record.mapped('partner_id')
# returns the union of all partner banks, with duplicates removed
record.mapped('partner_id.bank_ids')
```



# Funciones recordsets

```
>>> rs0 = self.env["res.partner"].search([("display_name",
"like", "Azure")])
>>> len(rs0)  # how many records?
4
>>> rs0.filtered(lambda r: r.name.startswith("Nicole"))
res.partner(27,)
>>> rs0.filtered("is_company")
res.partner(14,)
>>> rs0.mapped("name")
['Azure Interior', 'Brandon Freeman', 'Colleen Diaz', 'Nicole
Ford']
>>> rs0.sorted("name", reverse=True).mapped("name")
['Nicole Ford', 'Colleen Diaz', 'Brandon Freeman', 'Azure
Interior']
>>> rs0.mapped(lambda r: (r.id, r.name))
[(14, 'Azure Interior'), (26, 'Brandon Freeman'), (33, 'Colleen
Diaz'), (27, 'Nicole Ford')]
```

# Valor por defecto otra tabla

The function for finding the default stage should return the record that will be used as the default value:

```
    @api.model
    def _default_stage_id(self):
        Stage = self.env["library.checkout.stage"]
        return Stage.search([("state", "=", "new")],
          limit=1)
```

This returns the first record in the stage model. Since the stage model is ordered by sequence, it will return the one with the lowest sequence number.

# On change

```
    @api.onchange("member_id")
    def onchange_member_id(self):
        today_date = fields.Date.today()
        if self.request_date != today_date:
            self.request_date = today_date
            return {
                "warning": {
                    "title": "Changed Request Date",
                    "message": "Request date changed to
                      today!",
                }
            }
```

The previous `onchange` method is triggered when the `member_id` field is set on the user interface. The actual method name is not relevant, but the convention is for its name to begin with the `onchange_` prefix.

# Mas Onchange

One is that it works disconnected from the server-side events. Onchange is only played when requested by the form view and is not called as a consequence of an actual `write()` value change. This forces the server-side business logic to explicitly replay the relevant onchange methods.

```
    request_date = fields.Date(
        default=lambda s: fields.Date.today(),
        compute="_compute_request_date_onchange",
        store=True,
        readonly=False,
    )
```

# Mas Onchange

```
@api.depends("member_id")
def _compute_request_date_onchange(self):
    today_date = fields.Date.today()
    if self.request_date != today_date:
        self.request_date = today_date
        return {
            "warning": {
                "title": "Changed Request Date",
                "message": "Request date changed to
                  today!",
            }
        }
```

# Read_Group

```python
from odoo import models, api, _
from datetime import datetime, timedelta

class SaleOrderLine(models.Model):
    _inherit = 'sale.order.line'

    def check_total_sales_in_last_month(self):
        # Calcular la fecha de hace un mes
        one_month_ago = datetime.now() - timedelta(days=30)

        # Agrupar las líneas de pedido por cliente y calcular el total de ventas en el último mes
        grouped_sales = self.env['sale.order.line'].read_group(
            [('order_id.date_order', '>=', one_month_ago)],  # Dominio: Pedidos realizados en el último mes
            ['partner_id'],
            ['partner_id', 'total_sales:sum(price_subtotal)']
        )

        # Establecemos el límite de venta
        sale_limit = 10000

        for group in grouped_sales:
            if group['total_sales'] > sale_limit:
                raise UserError(_("El cliente %s ha excedido el límite de venta de %s euros.") % (group['partner_id'][1], sale_limit))

    # Llamamos a este método desde un botón o acción en la interfaz de usuario
    def button_check_sales(self):
        self.check_total_sales()
```

```
[
    {'partner_id': (3, 'Acme Corporation'), 'total_sales': 15000.0},
    {'partner_id': (5, 'Global Tech'), 'total_sales': 8000.0},
    {'partner_id': (7, 'Small Business'), 'total_sales': 2500.0}
]
```

# Wizards

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

<record id='liga_equipo_wizard_form' model='ir.ui.view'>
    <field name='name'>Wizard para introducir un Equipo</field>
    <field name='model'>liga.equipo.wizard</field>
    <field name='arch' type='xml'>
        <form string="Introducir datos de un equipo">
            <sheet>
                <group>
                    <field name='nombre'/>
                </group>
                <group>
                    <field name='descripcion'/>
                </group>
            </sheet>
            <footer>
                <button string='Añadir' name='add_liga_equipo' class='btn-primary' type='object'/>
                <button string='Cancel' class='btn-default' special='cancel'/>
            </footer>
        </form>
    </field>
</record>

<act_window id="action_wizard_liga_equipo" name="Añadir equipo" res_model="liga.equipo.wizard" view_mode="form" target="new" />
<menuitem id="menu_wizard_liga_equipo" parent="liga_base_menu" action="action_wizard_liga_equipo" sequence="20" />
```

# Wizards

```python
# -*- coding: utf-8 -*-
from odoo import models, fields

#Esta clase observamos que hereda de "models.TransientModel" una clase especial
#que crea un modelo, pero es temporal y no hacer permanente sus datos en la base de datos.
#Se utiliza para "mientras dura el Wizard"
class LigaEquipoWizard(models.TransientModel):
    _name = 'liga.equipo.wizard'
    #Campos del modelo que usaremos en el Wizard
    nombre = fields.Char()
    descripcion = fields.Html('Descripción', sanitize=True, strip_style=False)

    #Funcion que se llamara desde el Wizard, para utilizando este modelo temporal
    #y con el crear un nuevo registro en el modelo destino
    def add_liga_equipo(self):
        #Obtenemos referencia al modelo destino
        ligaEquipoModel = self.env['liga.equipo']
        #Tenemos que recorrer porque recordamos self referencia a todo el modelo
        for wiz in self:
            #Por cada elemento (en verdad, este Wizars solo tendra uno)
            #Creamos un registro en "liga.equipo"
            ligaEquipoModel.create({
                'nombre': wiz.nombre,
                'descripcion': wiz.descripcion,
            })
```