

DAM - M10

UF2. Sistemes ERP-CRM. Explotació i adequació.



Activitats a fer

Durant les diferents sessions de classe anireu seguint les diferents fases les quals us permetran entendre el gestor Odoo i conèixer les seves possibilitats.

Conceptes previs

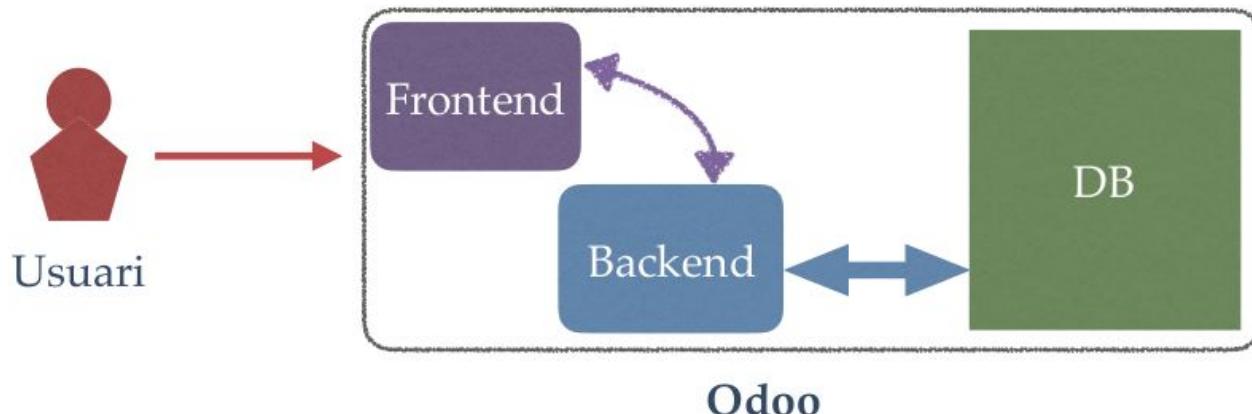


Conceptes

- *Enterprise Resource Planning*: Software de gestió empresarial que s'integra en totes les facetes del negoci (planificació, fabricació, vendes...).
- *Customer Relationship Management*: Software que permet gestionar les interaccions entre l'empresa i el client, fent possible una millor experiència del client, i una pressa de decisions més orientada a client.
- *Business Intelligence*: Presentar informació als usuaris empresarials per a que puguin utilitzar-la per guanyar coneixement.
- Datawarehouse: Magatzem de informació separat de les operacions transaccionals del dia a dia, i optimitzat per l'accés i l'anàlisi.

Arquitectura Odoo

- Amb el client ens podrem connectar al nostre servidor a través del navegador amb: **http://IP Odoo server:8069**
- També podem establir connexió a la BBDD d'Odoo amb el PgAdmin4
- Odoo utilitza *Object Relational Mapping* com a *backend*.



Arquitectura Odoo

- *La capa de Model del framework d'Odoo és correspon bàsicament amb les dades que formen part del ERP, i en Odoo tot això s'emmagatzemarà en la BBDD PostgreSQL*
- *La capa Vista del framework d'Odoo s'implementa, igual que en molts casos, mitjançant fitxers xml que Odoo s'encarregarà de renderitzar per mostrar a l'usuari en un navegador web.*
- *La capa de Control del framework d'Odoo s'ha realitzat mitjançant el llenguatge de programació Python, i s'encarrega d'aplicar la lògica del negoci i les regles del diferents fluxos de treball.*

Arquitectura Odoo

- En general tots els ERPs treballen mitjançant mòduls que es van afegint per poder ajustar-se al màxim número d'empreses, i Odoo no és cap excepció.
- El mòduls habituals que podem trobar en un ERP serien els relacionats amb vendes, compres, logística, finances, RRHH i fabricació entre d'altres.
- Mòduls a carregar de moment en el nostre cas són; gestió vendes i MRP.

Instal·lació Odoo v13

Pots seguir la Fase 1. Instal·lació d'Odoo Community

i també la documentació oficial:

<https://www.odoo.com/documentation/13.0/setup/install.html>

Per cambiar una IP estàtica ubuntu 18.04:

<https://websiteforstudents.com/configure-static-ip-addresses-on-ubuntu-18-04-beta/>

Inserim dades a Odoo

- Per començar a recordar el vostre treball a la UF1 amb Odoo, anem a fer algunes gestions e inserir algunes dades bàsiques per començar a treballar.

Pots seguir:

- Fase 2. Gestió d'aplicacions
- Fase 3. Flux de gestió

Odoo - Mode debug

De moment havem recordat l'ús general d'Odoo però en aquesta part del curs el nostre interès és centrarà en saber com ajustar l'Odoo a les necessitats concretes d'una empresa. Haureu de treure la vostra vessant de desenvolupadors d'aplicacions.

Una de les eines bàsiques per poder començar a desenvolupar amb Odoo és conèixer com treballar amb el mode debug.

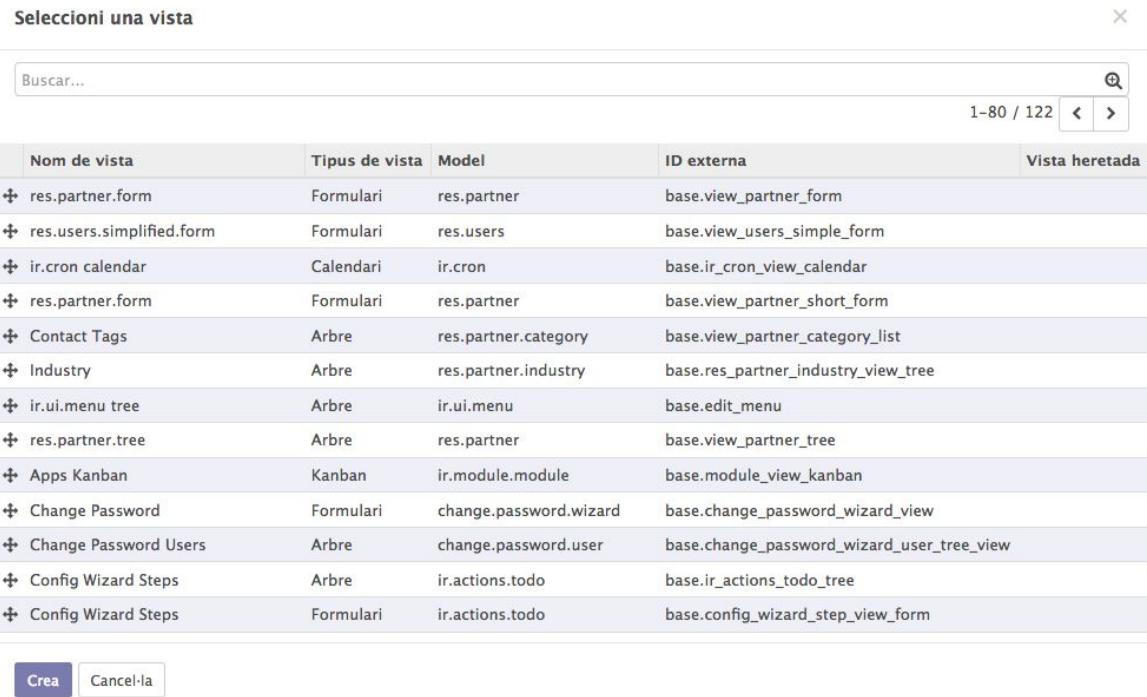
Activarem el mode debug des de la finestra "About" d'Odoo, al menú de configuració.

Odoo - Mode debug

- Mitjançant “Debug mode” podem visualitzar i modificar ràpidament els camps que veiem a Odoo.
- En el mode “Debug” al parar el cursor sobre un camp ens mostra informació relacionada amb el codi d’Odoo.
 - Field — Camp dins de PostgreSQL
 - Object — Objecte Python
 - Type — Tipus de dada
 - Modifiers — Modificadors del comportament

Odoo - Mode debug

- Podem modificar les vistes gràcies a l'opció “Manage Views” del mode debug.
- Obrirà una pantalla on estarà seleccionada la nostra pantalla.



Repàs Python

- El llenguatge de programació usat per odoo es Python.
- A continuació revisarem les bases de Phyton:
- [Manual Phyton](#)

Odoo – Creació de mòduls

- Hi ha vegades que modificar el comportament d'un objecte d'Odoo no serà suficient (editar la seva vista), i els canvis demanaran la creació d'un nou mòdul.
- El framework d'Odoo guarda tots els mòduls dins del directori "addons". Podeu identificar on està ubicat el vostre directori "addons" en **/etc/odoo/odoo.conf**
- També podeu afegir nous directoris per a la creació de nou mòduls mitjançant la variable **addons_path = path_folder**

Odoo – Creació de mòduls

- Els mòduls són els components bàsics per a les aplicacions Odoo. Un mòdul pot afegir noves característiques a Odoo, o modificar les existents. És un directori que conté un manifest, o arxiu descriptor, anomenat **`__manifest__.py`**, més els arxius restants que implementen les seves característiques.
- Les aplicacions són la forma en què s'afegeixen les principals característiques a Odoo. Proporcionen els elements bàsics per a una àrea funcional, com Comptabilitat o RH, en funció de quines característiques de mòduls complementaris modifiquen o amplien. A causa d'això, es destaquen al menú Apps de Odoo.

Odoo – Creació de mòduls

- Crear un nou mòdul implica també crear un nou directori (que permeti identificar el mòdul) dins de “addons”.
- Dins d'aquest nou directori serà obligatori afegir els fitxers **`__init__.py`** i **`__manifest__.py`**
- **`__init__.py`** és el fitxer on haurem d'indicar els fitxers en codi Python que formaran part del nostre mòdul.
- **`__manifest__.py`** és un fitxer de descripció del nostre mòdul, on haurem de crear un diccionari de Python amb les claus: name, version, description, author, depends, data, demo, installable, auto_install.

Odoo – Creació de mòduls

- El seu contingut serà similar al següent:

```
# -*- coding: utf-8 -*-
# Part of Odoo. See LICENSE file for full copyright and licensing details.

from . import controllers
from . import models
from . import report
from . import wizard
```

```
# -*- coding: utf-8 -*-
# Part of Odoo. See LICENSE file for full copyright and licensing details.
{
    'name': 'Sales Channels',
    'version': '1.0',
    'category': 'Sales',
    'summary': 'Sales Channels',
    'description': """
Using this application you can manage Sales Channels with CRM and/or Sales
=====
"""
    'website': 'https://www.odoo.com/page/crm',
    'depends': ['base', 'mail'],
    'data': ['security/sales_team_security.xml',
             'security/ir.model.access.csv',
             'data/sales_team_data.xml',
             'views/crm_team_views.xml',
             'views/sales_team_dashboard.xml',
             ],
    'demo': ['data/sales_team_demo.xml'],
    'installable': True,
    'auto_install': False,
}
```

Odoo – __init__.py

- En aquest fitxer importarem els fitxers Python que formen part del nostre mòdul. Exemples:

import tasques_a_fer

from . import tasques_a_fer

from altre_directori import tasques_a_fer

Odoo – __manifest__.py

name - nom que apareixerà en Odoo pel nostre mòdul.

version - permet especificar el número de versió.

category - permet especificar la categoria (secció) del mòdul.

sequence

summary - Paraules clau per resumeixen la funció del mòdul.

description - entrada que apareix quan algú vol instal·lar el mòdul, indica informació suficient per entendre que farà el mòdul en qüestió.

website - web oficial del mòdul

depends - dependències d'aquest mòdul amb altres mòduls d'Odoo (com a mínim ha d'incloure *base*).

data - especifica els fitxers xml que formaran part del mòdul.

demo - permet dades de demo per poder treballar amb el mòdul (es pot deixar en blanc).

installable - marcarem si volem permetre que el mòduls s'instal·li (true).

application - marcarem si volem permetre que el mòdul es visualitzi com una App (true).

auto_install - permet definir si volem que s'instal·li aquest mòdul si les dependències ja estan instal·lades.

Odoo – Creació de mòduls

Els diccionaris de Python son un tipus de variable que forma arrays de tipus clau-valor (hashtable).

La clau ha de ser un string

El valor pot ser qualsevol tipus de dada

Es generen mitjançant,

```
var_dicc = { 'clau1': valor1 , 'clau2': valor2, ...}
```

Odoo – Creació de mòduls

Un cop generat el nostre mòdul local haurem de carregar aquest en Odoo. Els passos a seguir són:

1. Reiniciar el servidor d'Odoo després de la generació del directori del mòdul i els fitxers **`__init__.py`** i **`__manifest__.py`**.
2. Connectar-se amb mode “debug” a Odoo.
3. Des del menú d'Apps seleccioolem “Update Apps List”.
4. Fem una cerca per buscar el nom de la vostra App, la instal·lem.

Odoo – Creació de mòduls

- En l'exemple que seguirem, crearem un nou mòdul amb el menor nombre possible de dependències.
- No obstant això, aquest no serà el cas típic. Principalment es modifiquen o s'estenen funcionalitats d'un mòdul ja existent.
- Es considera una mala pràctica modificar els mòduls existents canviant el seu codi font directament. En el seu lloc, hem de crear els mòduls d'extensió que s'instal·laran al costat dels mòduls que volem modificar, implementant els canvis que necessitem. De fet, un dels principals punts forts de Odoo és el mecanisme **d'herència**, que permet mòduls personalitzats per estendre els mòduls existents.

Creació d'un mòdul



Odoo – Creació de mòduls

- Per mantenir les coses ordenades, crearem un nou directori al nostre home ~.
- Odoo inclou una ordre scaffold per crear automàticament un nou directori de mòdul, amb una estructura bàsica ja establerta. Pots obtenir més informació al respecte amb la següent comanda:

```
$ ~/odoo-dev/odoo/odoo-bin scaffold --help
```

Odoo – Creació de mòduls

Crearem una nova carpeta amb un nom tècnic pel mòdul **todo_app**. Aquest nom ha de ser un identificador vàlid per Phyton: començant per lletra i sols contenint lletres, números i barres baixes. A dintre del nostre nou directori, crearem els dos fitxers principals del mòdul **__init__.py** i **__manifest__.py**.

```
{  
    'name': 'To-Do Application',  
    'description': 'Manage your personal To-Do tasks.',  
    'summary': 'Explanation tittle',  
    'website': 'Website contact',  
    'author': 'My Name and Surnames',  
    'depends': ['base'],  
    'application': True,  
}
```

Odoo – Creació de mòduls

En cas de trobar alguna errada en el codi d'un mòdul (lògica o d'interfície), Odoo no s'inicia correctament.

Per poder resoldre els errors trobats haurem de revisar el fitxer de Logs i buscar entre les darreres entrades al fitxer.

El fitxer de logs d'Odoo es troba a **/var/log/odoo/odoo-server.log**
Podeu visualitzar les vint darreres línies mitjançant la comanda:

```
sudo cat /var/log/odoo/odoo-server.log | tail -20
```

Odoo – Model

Els models descriuen objectes de negoci, la informació del negoci que volem desar: clients, proveïdors, etc. Un model té una llista d'atributs i també pot definir el seu negoci específic.

Els models s'implementen utilitzant una classe Python derivada d'una classe de plantilla Odoo. Es tradueixen directament a objectes de base de dades, i Odoo s'encarrega d'això automàticament a l'instal·lar o actualitzar el mòdul. El mecanisme responsable d'això és el **Model Relacional d'Objectes (ORM)**.

El nostre mòdul serà una aplicació molt simple per mantenir les tasques pendents. Aquestes tasques tindran un sol camp de text per a la descripció i una casella de verificació per a marcar-les com a completes. Més endavant ampliarem funcionalitats.

Odoo – Model. Atributs estructurals

És necessari importar la classe “model” d’odoo que ens dona aquests atributs.

La classe model d’Odoo que ens dona diversos atributs a usar, tot i que l’únic obligatori és **_name**. Per definir el nom del model a dintre d’Odoo.

Altres possibles atributs són:

- **_rec_name** — marca el nom utilitzat per les vistes per referenciar el model (records).
- **_order** — permet marcar en funció de quins atributs s’ordenarà el model.
- **_description** — permet marcar una descripció pel mòdul.
- **_inherit** — crea una herència de la classe referenciada.
- **_inherits** — permet fixar quins atributs i/o mètodes s’hereten de cada classe
- **_sql_constraints** — permet fixar les restriccions de SQL
- **_constraints** — permet fixar les restriccions de l’objecte

Odoo - Model. atributs simples.

Els atributs simples permeten guardar valors concrets. Aquests utilitzen la classe `field` per generar els diferents camps a la taula de PostgreSQL.

- **char** — Permet emmagatzemar una cadena de text.
- **text** — Permet emmagatzemar una cadena de text que inclogui múltiples línies.
- **selection** — Permet emmagatzemar les dades per un menú desplegable.
- **html** — Permet emmagatzemar un camp de text amb etiquetes HTML
- **integer** — Permet emmagatzemar un enter
- **float** — Permet emmagatzemar un real
- **date/datetime** — Permet emmagatzemar una data / data incorporant hora.
- **boolean** — Permet emmagatzemar un valor lògic.
- **binary** — Permet emmagatzemar un fitxer.

Odoo - Model. atributs simples.

Els atributs simples poden utilitzar certs paràmetres per ajustar el seu comportament. Alguns d'aquests paràmetres serien,

- **string** — serà el títol del camps (nom que veurem en la UI).
- **default** — valor per defecte que volem donar a aquest camp.
- **size** — màxima mida permesa en les cadenes de text.
- **translate** — permet que el text sigui traduïble.
- **help** — serà la ajuda que es mostrerà en Odoo per aquest camp.
- **readonly** — permet fer un camp no editable.
- **required** — marca aquest camp com obligatori.
- **index** — crea un index per aquest camp en la BBDD.
- **groups** — permet marcar l'accés i la visibilitat només per a un grup d'usuaris.
- **states** — permet passar un diccionari Python on podem assignar als diferents valors del camp “state” un atribut que modifiqui el seu comportament en la interfície d'usuari.

Odoo - Model. atributs simples.

Hi ha certs noms d'atributs que estàn reservats i que els genera automàticament Odoo

- **id** — número únic que identifica cada entrada.
- **create_uid** — usuari que ha creat la entrada
- **create_date** — data i hora de creació de l'entrada.
- **write_uid** — últim usuari que ha modificat l'entrada
- **write_date** — data i hora de l'última modificació.

També hi ha certs atributs que encara que no siguin reservats son utilitzats per dur a terme certes funcions dins d'Odoo.

- **name** — s'utilitza per defecte pel nom a mostrar en les vistes
- **active** — permet marcar com inactiu aquest objecte.
- **sequence** — permet marcar l'ordre en que es mostra un objecte.
- **state** — representa el cicle de vida d'un objecte.
- parent_id / parent_left / parent_right — representa la jerarquia d'un objecte.

Odoo - Model. atributs simples.

- Python és molt estricta amb l'indexació!!!

```
File Edit View Search Terminal Help
GNU nano 2.9.3

# -*- coding: utf-8 -*-
from odoo import models, fields
class TodoTask(models.Model):
    _name = 'todo.task'
    _description = 'To-do Task'
    name = fields.Char('Description', required=True)
    is_done = fields.Boolean('Done?')
    active = fields.Boolean('Active?', default=True)█
```

Odoo - La capa de vista.

La capa de vista descriu la interfície d'usuari. Les vistes es defineixen mitjançant XML, que és utilitzat pel marc de client web per generar vistes HTML amb dades.

Tenim elements de menú que poden activar accions que poden fer vistes.

Per exemple: l'opció de menú **Usuaris** processa una acció també anomenada Usuaris, que al seu torn genera una sèrie de vistes. Hi ha diversos tipus de vista disponibles, com les vistes de llista i formulari i les opcions de filtre també disponibles, estan definides per un tipus particular de vista, la vista de cerca.

Odoo - La capa de vista.

Les directrius de desenvolupament d'Odoo estableixen que els arxius XML que defineixen la interfície d'usuari han de col·locar-se dins d'un subdirectorí `views/`.

Així que començarem per a la interfície d'usuari per a la nostra aplicació de tasques pendents, creant un nou subdirectorí al nostre mòdul anomenat `views`.

```
todo
├── __init__.py
├── manifest__.py
└── models
    ├── __init__.py
    ├── todo_models.py
    └── todo_models.py.save
└── views
    └── todo_menu.xml
```

Odoo - La capa de vista.

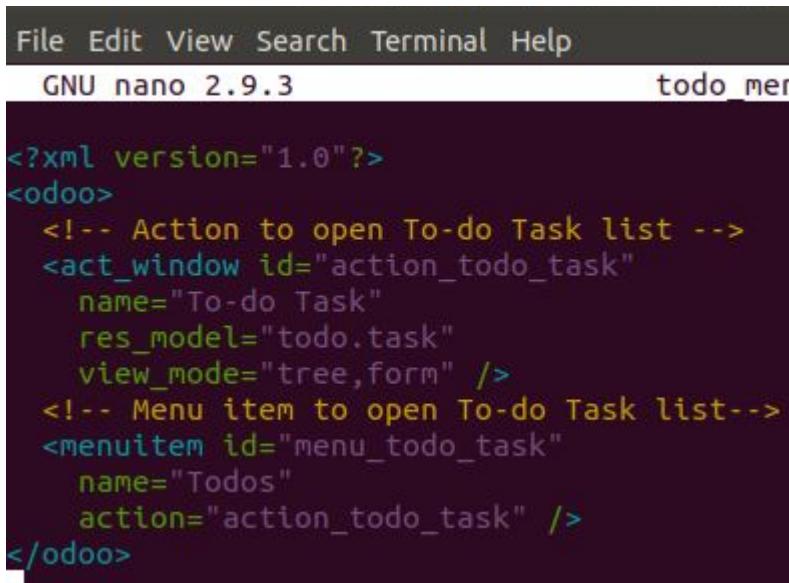
Ara que tenim ja un model per emmagatzemar les nostres dades, havem de fer disponible una interfície d'usuari amb la que interactuar.

Per poder-ho aconseguir, havem d'afegir una opció de menú per obrir el model **To-do Task** perquè pugui utilitzar-se.

Crearem el fitxer **views/todo_menu.xml** per definir un element de menú i l'acció realitzada per ell:

Odoo - La capa de vista.

- Necessitem el següent fitxer xml per poder veure el nostre primer mòdul d'Odoo.



A screenshot of a terminal window titled "todo_mer". The window shows a file named "todo_mer" being edited with "GNU nano 2.9.3". The content of the file is an XML snippet defining an action and a menu item for a "To-do Task" list.

```
<?xml version="1.0"?>
<odoo>
    <!-- Action to open To-do Task list -->
    <act_window id="action_todo_task"
        name="To-do Task"
        res_model="todo.task"
        view_mode="tree,form" />
    <!-- Menu item to open To-do Task list-->
    <menuitem id="menu_todo_task"
        name="Todos"
        action="action_todo_task" />
</odoo>
```

<act_window> defineix una acció de finestra del costat del client que obrirà el model **todo.task** amb les vistes d'arbre i formulari habilitades, en aquest ordre.

<menuitem> defineix un element de menú superior que crida a l'acció **action_todo_task**, que es va definir anteriorment.

Per fer menus i submenus:

```
<menuitem  
    id="todo_submenu"  
    name="Submenus To-Do"  
    parent="todo_menu_task"/>
```

```
<menuitem  
    id="sub_sub_menu"  
    name="Tareas a realizar"  
    parent="todo_submenu"  
    action="action_todo_task"/>
```

Amb la comanda parent es poden fer més submenus.

Primer menu a la part de dalt.

Següents sebmenus a la part de l'esquerra

Odoo - Vista.

La **interfície d'usuari**, incloses les opcions i les accions de menú, s'emmagatzemen en les **taules de la base de dades**.

L'arxiu XML creat és un arxiu de dades utilitzat per carregar aquestes definicions a la base de dades quan el mòdul s'instal·la o actualitza. El codi anterior és un arxiu de dades Odoo, que descriu dos registres per afegir a Odoo.

Tots dos elements inclouen un atribut id. Aquest atribut **id** és molt important: s'utilitza per identificar de forma única cada element de dades dins del mòdul, i pot ser utilitzat per altres elements per referenciar-lo. En aquest cas, l'element **<menuitem>** necessita fer referència a l'acció per processar, i necessita fer ús de la ID per això.

Odoo - Vista.

El nostre mòdul encara no sap que hi ha un fitxer XML el qual servirà d'intermediari entre ell i l'usuari. Per fer-li-ho saber, tindrem que editar el nostre fitxer **__manifest__.py** a l'atribut que conté la llista de fitxers a carregar pel mòdul:

```
'data': ['views/todo_menu.xml'],
```

Finalment necessitarem tornar a carregar Odoo i actualitzar el mòdul per veure que els nostres canvis.

Vistas Tree / Form



Odoo - vista

Associem la Vista (arquitectura MVC) amb la interfície d'usuari.

- ❖ Odoo guarda aquesta informació en fitxers xml que s'emmagatzemen a PostgreSQL mitjançant entrades ("records") en diferents taules.
- ❖ Els tipus principals de vistes que trobem a Odoo són:
 - ❖ Tree – Vista utilitzant una UI de tipus llistat.
 - ❖ Form – Vista utilitzant una UI de tipus formulari.
 - ❖ Search – Vista utilitzada en les cerques quan mostrem llistats.
- ❖ També tenim altres vistes alternatives: "Calendar", "Kanbas", "Graphs", "Gantt".

Odoo - vista - Definició fitxer XML

- Les vistes d'Odoo han d'estar definides dins de les etiquetes <odoo> i <data> respectivament.
- Utilitzem l'etiqueta record per generar les diferents vistes. I haurem d'indicar els atributs model i id per aquesta etiqueta.
 - model — Les vistes d'Odoo és generen com a records (registres) del model “**ir.ui.view**”. Només els menús, i les accions és defineixen utilitzant un tipus diferent de model.
 - id — Aquest serà el XML ID (identificador XML) que utilitzarem per referenciar a aquest registre xml des d'altres vistes o accions.
- Dins del record (registre) utilitzarem l'etiqueta field per afegir o introduir camps en aquesta vista.

Odoo - vista - Definició <record>

<record> és cadascun dels registres disponibles en PostgreSQL i trobem diferents models de registre relacionats amb la vista de la UI.

- Dins del record utilitzarem l'etiqueta field per afegir o introduir camps en aquesta vista.
- Alguns camps importants dins de la vista són:
 - name — Nom que es visualitzarà en la vista.
 - model — Referència interna del Model Python (atribut _name)
 - priority — Valor numèric que permet fixar quina vista és carrega primer (s'utilitza en les herències).
 - arch — Tipus d'arquitectura, en el nostre fitxers sempre escollirem type="xml". Dins d'aquest camp definirem el tipus de vista, els camps a mostrar i l'organització de tots els elements.

Odoo - vista - Definició <record>

```
<?xml version="1.0"?>
<odoo>
    <!-- Esta es la vista formulario, -->
    <record id="view_form_todo_task" model="ir.ui.view">
        <field name="name">view name</field>
        <field name="model">object.name</field>
        <field name="arch" type="xml">
            <!-- aqui va contenido -->
        </field>
    </record>
```

Odoo - vista Formulari- header / sheet

Odoo ens permet donar una estructura de document als formularis mitjançant les etiquetes següents.

- ❖ header — Aquest element em permet agrupar els elements que formaran part de la capçalera (habitualment botons i barres d'estat).
- ❖ sheet — Aquest element permet mostrar el cos del document.

```
<form>
    <header>
        <!-- Buttons go here-->
    </header>
    <sheet>
        <!-- Content goes here: -->
        <field name="name"/>
        <field name="is_done"/>
    </sheet>
</form>
```

Odoo - vista Formulari - groups

Odoo ens permetrà estructura la vista del formulari en diferents files i columnes mitjançant l'etiqueta <group>.

❖ group — Cada etiqueta pare de <group> permet definir agrupacions en línies dels diferents camps. Si dins d'aquestes etiquetes pare inserim un nova etiqueta <group> estarem definint diferents columnes. També podem utilitzar els atributs col i colspan per definir exactament el nombre de columnes.

❖ col — indica el nombre de columnes
❖ colspan — indica el nombre de columnes que ocuparà un element.

❖ string — permet mostrar un títol en la vista per a cada agrupament.

```
<group name="group_top">
    <group name="group_left">
        <field name="name"/>
    </group>
    <group name="group_right">
        <field name="is_done"/>
        <field name="active" readonly="1"/>
    </group>
</group>
```

Odoo - vista - Record act_window

Aquest menús realitzen accions que definirem en un tipus especial de record anomenat <act_window>.

- ❖ Els atributs que té aquesta etiqueta són:
- ❖ name — Títol mostrat en les vistes obertes per aquesta acció.
- ❖ res_model — Nom del model (Python) que utilitzarem
- ❖ view_mode — Tipus de vistes que podrem mostrar. Per defecte són “tree” i “form”.
- ❖ limit — Nombre màxim de registre a mostrar en la vista “tree”.
- ❖ context — Valors globals que permet emmagatzemar Odoo i utilitzar en diferents vistes.
- ❖ domain — Permet filtrar els registres que es mostraran en les vistes a carregar.

Odoo - vista - Record act_window

```
<record id="discos_action_view" model="ir.actions.act_window">
    <field name="name">discos.action.view</field>
    <field name="res_model">disc</field>
    <field name="view_type">form</field>
    <field name="view_mode">tree,form</field>
    <field name="view_id" ref="discos_view_tree"/>
</record>

<record id="author_action_view" model="ir.actions.act_window">
    <field name="name">author.action.view</field>
    <field name="res_model">author</field>
    <field name="view_type">form</field>
    <field name="view_mode">tree,form</field>
    <field name="view_id" ref="author_view_tree"/>
</record>
```

Odoo - vista - Formulari

- ❖ La vista de tipus formulari ens permet mostrar informació detallada d'un registre, o bé crear un de nou.
- ❖ L'etiqueta <field> permet definir quins camps del model estaran disponibles en la vista.

Exemple vista formulari.
Autor de disco!

```
<record id="author_view_form" model="ir.ui.view">
    <field name="name">author.view.form</field>
    <field name="model">author</field>
    <field name="arch" type="xml">
        <form string="Autor">
            <group>
                <field name="name"/>
                <field name="birthday"/>
                <field name="age"/>
            </group>
        </form>
    </field>
</record>
```

Odoo - vista - Tree

La vista de tipus llistat <tree> mostra un llistat dels diferents registres que trobem d'un model concret.

- ❖ La informació que podrem veure en el llistat d'aquest registre la definirem mitjançant l'ús de <field>
- ❖ Els següents atributs es poden utilitzar en aquesta vista:
 - ❖ colors — Permet modificar el color d'una camp atenent a alguna condició. Utilitzant una sintaxi de tipus colors="nom_color:condició_Python"
 - ❖ fonts — Igual que colors però modificant la font entre bold, italic o underline.
 - ❖ default_order — Permet modificar el camp per fer l'ordenació de la llista i el tipus d'ordenació.

```
<tree default_order="sequence, name desc">
```

Odoo - vista - Tree

```
<record id="discos_view_tree" model="ir.ui.view">
    <field name="name">discos.view.tree</field>
    <field name="model">disc</field>
    <field name="arch" type="xml">
        <tree string="Mis Discos Musicales">
            <field name="name"/>
            <field name="published_date"/>
            <field name="author_id"/>
        </tree>
    </field>
</record>
```

Odoo - vista

Modifica vista tipus formulari “todo.task” perquè quedi:

The screenshot shows the Odoo interface for creating a new task. On the left, there is a sidebar with a logo placeholder, a 'Your logo' section, and a 'todo submenu' with 'Todo Task' selected. The main area has a title 'todo.task.action.view / Nuevo' with 'Guardar' and 'Descartar' buttons. The form itself has fields for 'Description' (a long input field), 'Done?' (an unchecked checkbox), 'Active?' (a checked checkbox), and an 'Owner' section. The 'Owner' section includes a table with columns 'Nombre', 'Edad', 'Apellidos', and 'Task'. A button 'Añadir un elemento' is present in the 'Owner' section.

todo.task.action.view / Nuevo

Guardar Descartar

Description

Done?

Active?

Owner

Nombre	Edad	Apellidos	Task
Añadir un elemento			

Odoo - vista

Modifica vista tipus tree perquè quedi:

todo.task.action.view

Buscar...

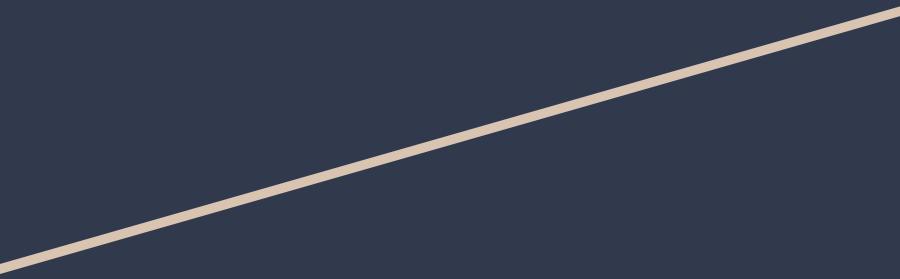
Crear Importar

Description	Done?	Owner
<input type="checkbox"/> ir a comprar	<input type="checkbox"/>	1 registro
<input type="checkbox"/> comer	<input checked="" type="checkbox"/>	2registros
<input type="checkbox"/> ir al cine	<input checked="" type="checkbox"/>	2registros
<input type="checkbox"/> Vacaciones	<input type="checkbox"/>	2registros

Description

- **Campo:** name
- **Objeto:**
- **Tipo:** char
- **Modificadores:** {"required":true}

Base de dades PostgreSQL



PostgreSQL

Podem consultar l'estrucuta de la nostra base de dades a la ruta: Ajustes -> Técnico -> Estructura de la base de datos.
o fent servir el terminal.

Gestió de la venda de components - Odoo - Mozilla Firefox

dmorenoar/odoo How to Install PostgreSQL pgAdmin 4

192.168.1.173:8069/web#id=223&action=16&model=ir.model&view_type=form&cids=&menu_id=4

Ajustes Opciones Generales Usuarios y compañías Traducciones Técnico

Modelos / Gestió de la venta de components

Editar Crear Imprimir Acción

Descripción del modelo	Gestió de la venta de components	Tipo	Objeto base
Modelo	venda.component	En las aplicaciones	venda_component
Modelo transitorio	<input type="checkbox"/>		
Hilo de mensajes	<input type="checkbox"/>		
Actividad de correo	<input type="checkbox"/>		
Lista negra de correo	<input type="checkbox"/>		

Campos Permisos de acceso Reglas de registro Notas Vistas

Nombre de campo	Etiqueta de campo	Tipo de campo	Requerido	Sólo lectura	Indexado	Tipo
_last_update	Last Modified on	Fecha y hora	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base
color	Color	Selección	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Campo base
create_date	Created on	Fecha y hora	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base
create_uid	Created by	many2one	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base
display_name	Display Name	Carácter	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base
id	ID	entero	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base
is_offer	Té descompte?	booleano	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Campo base
marca	Marca	Carácter	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Campo base
name	Nom del component	Carácter	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Campo base
write_date	Last Updated on	Fecha y hora	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base
write_uid	Last Updated by	many2one	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base

Connexions remotes PostgreSQL

Conectar-te a la BD en remot.

Modificar postgres per poder-nos conectar desde fora:

```
sudo nano /etc/postgresql/10/main/postgresql.conf
```

```
listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for $
```

```
sudo nano /etc/postgresql/10/main/pg_hba.conf
```

```
# IPv4 local connections:
host    all            all            0.0.0.0/0            md5
```

1. Comprovar si està el socket obert: netstat -an | grep 5432

PostgreSQL

```
sudo su postgres
psql
\conninfo                               \? (ajuda)
\l - listar
\c odoo (conectar a BBDD odoo)
\dt - list tables
\d todo_task (veure
select * from todo_task;
```

```
(remot) psql -h 34.242.208.216 -U odoo2019 -d odoo
```

```
odoo-# \d todo_task
```

Table "public.todo_task"

Column	Type	Collation	Nullable	Default
id	integer		not null	nextval('todo_task_id_seq'::regclass)
name	character varying		not null	
is_done	boolean			
active	boolean			
create_uid	integer			
create_date	timestamp without time zone			
write_uid	integer			
write_date	timestamp without time zone			

Indexes:

"todo_task_pkey" PRIMARY KEY, btree (id)

Foreign-key constraints:

"todo_task_create_uid_fkey" FOREIGN KEY (create_uid) REFERENCES res_users(id) ON DELETE SET NULL

"todo_task_write_uid_fkey" FOREIGN KEY (write_uid) REFERENCES res_users(id) ON DELETE SET NULL

```
odoo=# select * from todo_task;
 id |      name      | is_done | active | create_uid |           create_date           | write_uid |           write_date
---+-----+-----+-----+-----+-----+-----+-----+
 1 | ir a comprar | f     | t     |          1 | 2019-03-11 14:03:43.920504 |         1 | 2019-03-11 14:03:43.920504
 2 | comer        | t     | t     |          1 | 2019-03-11 14:16:27.26391  |         1 | 2019-03-11 14:16:27.26391
 3 | ir al cine   | t     | t     |          1 | 2019-03-11 14:16:42.612513 |         1 | 2019-03-11 14:16:42.612513
(3 rows)
```

Instal·lació Pg admin 4 (voluntari)

Si volem gestionar d'una forma més còmode PostgreSQL podem instal·lar pgAdmin 4.

Instal·lació en ubuntu desktop 18.04:

<https://askubuntu.com/questions/831262/how-to-install-pgadmin-4-in-desktop-mode-on-ubuntu>

<https://www.howtoforge.com/how-to-install-postgresql-and-pgadmin4-on-ubuntu-1804-lts/>

Aplicaciones - Odoo dmorenoar/odoo How to Install PostgreSQL pgAdmin 4

PGAdmin Archivo Objeto Herramientas Ayuda

Navegador Tablero Propiedades SQL Estadísticas Dependencias Dependientes

Bienvenido

Rico en características | Maximiza PostgreSQL | Código Abierto

pgAdmin Management Tools for PostgreSQL

Enlaces Rápidos

Agregar un Nuevo Servidor

Empezando

Documentación PostgreSQL Página web de pgAdmin Planeta PostgreSQL

Crear - Servidor

General Conexión SSL Túnel SSH Avanzado

Nombre **odoo**

Grupo de servidores Servers

Segundo plano

Primer plano

¿Conectar ahora?

Comentarios

i ? Cancelar Restaurar Salvar

Crear - Servidor

General Conexión SSL Túnel SSH Avanzado

Nombre/Dirección de servidor **localhost**

Puerto **5432**

Base de datos de mantenimiento **postgres**

Nombre de usuario **postgres**

Contraseña **.....**

¿Salvar contraseña?

Rol

Servicio

i ? Cancelar Restaurar Salvar

Recordeu modificar la contrasenya de l'usuari postgresql per poder fer la configuració:

```
sudo su postgresql
```

```
psql
```

```
postgres=# ALTER USER postgres PASSWORD '12345'
```

```
postgres=# \q
```

pgAdmin Archivo ▾ Objeto ▾ Herramientas ▾ Ayuda ▾

danielmorenoarellano@gmail.com ▾

Navegador

Servers (1)

odoo

Bases de Datos (2)

odoo

Catálogos (2)

Contenedores de Datos Foráneos

Conversiones

Disparadores por evento

Esquemas (1)

public

Analizadores FTS

Colaciones

Configuraciones FTS

Diccionarios FTS

Dominios

Funciones

Funciones disparadoras

Plantillas FTS

Secuencias

Tablas (216)

base_document_layout

base_import_import

base_import_mapping

base_import_tests_models_char

base_import_tests_models_char_noreadonly

base_import_tests_models_char_READONLY

base_import_tests_models_char_REQUIRED

base_import_tests_models_char_STATES

base_import_tests_models_char_STILLREADONLY

base_import_tests_models_complex

base_import_tests_models_float

base_import_tests_models_m2o

Query Editor Historial de Consu...

1 SELECT * FROM public.venda_component

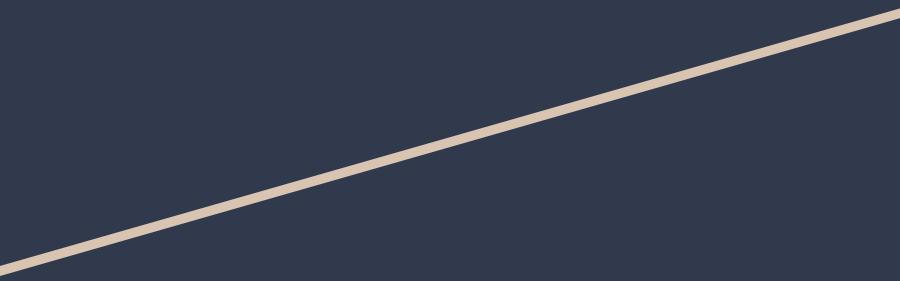
2

Scratch Pad

Salida de Datos Explain Mensajes Notificaciones

	id [PK] integer	name character varying	marca character varying	color character varying	is_offer boolean	create_uid integer	create_date timestamp without time zone
1		1 Marco 2DIN Corolla	TOYOTA	negre	true	[null]	[null]
2		2 Marco 1DIN Altea	SEAT	gris	false	[null]	[null]

Models atributs relacionals



- Els atributs relacionals indiquen relació entre diferents models, de manera semblant al model entitat-relació de les bases de dades.
- Les diferents relacions tenim pels camps de Odoo són,
 - **Many2one** — molts objectes de la classe que conté l'atribut poden estar relacionats amb un mateix objecte de la classe referenciada.
 - **One2many** — un objecte de la classe que conté l'atribut pot estar relacionat amb molts objectes de la classe referenciada. És una relació inversa del many2one.
 - **Many2many** — molts objectes de la classe A poden estar relacionats amb molts objectes de la classe B, i a l'inrevés.

Manyzone && One2many

- Generem atributs relacionals many2one mitjançant (només són obligatoris els camps en negreta),
 - nom_atribut=fields.Many2one(**comodel_name='other_object_name'**, ‘Field name’, optional parameters)
 - Exemple (si estigués en la classe Disc):
author_id = fields.Many2one(comodel_name="author", string="Autor del disco")
author_id = fields.Many2one('author', ‘autor del disco’)
- Generem atributs relacionals one2many mitjançant
 - nom_atribut=fields.One2many(**‘other_object_name’**, ‘Field relational name’,‘Field name’, optional parameters)

Many2many

- Generem atributs relacionals many2many mitjançant,
 - nom_atribut=fields.Many2many('Other Object name','Relational Object', 'actual object id', 'other object id', 'Field name')
- El paràmetres opcionals poden ser,
string='nom assignat a la relació'

required=[True|False]
readonly=[True|False]

```
arel_ids = fields.Many2many('res.users')
arel_ids = fields.Many2many(comodel_name='res.users',
                           relation='table_name',
                           column1='col_name',
                           column2='other_col_name')
```

Ex:

```
class Author(models.Model):
    _name = 'author'

    name = fields.Char(string="Nombre", required=True)
    birthday = fields.Date(string="Fecha nacimiento")
    age = fields.Integer(string="Edad")
```

Exemple Model

many2one
many2many

amb Autors, Discs i
Cançons

```
class Disc(models.Model):
    _name = 'disc'

    name = fields.Char(string="Título", required=True)
    published_date = fields.Date(string="Fecha publicación")
    author_id = fields.Many2one(comodel_name="author", string="Autor del disco")
    song_ids = fields.Many2many(comodel_name='song',
                                relation='disc_song_rel',
                                column1='disc_id',
                                column2='song_id')

class Song(models.Model):
    _name = 'song'

    name = fields.Char(string="Nombre de la canción", required=True)
    duracion = fields.Integer(string="duración de la canción")
    author_id = fields.Many2one(comodel_name="author", string="Autor de la canción")
    disc_ids = fields.Many2many(comodel_name='disc',
                                relation='disc_song_rel',
                                column1='song_id',
                                column2='disc_id')
```

Herència



Odoo - Herència

Odoo permet utilitzar l'herència per aprofitar mòduls, controladors, models i/o vistes ja disponibles.

- La herència en el Model ens permet ampliar un model afegint nous atributs, sobreescriure alguns atributs.
- Li hem de dir al arxiu `__manifest__.py` del nostre mòdul, que depen del mòdul pare.
- La herència en el Model es realitza mitjançant l'atribut:
- `_inherit` — herència tradicional, permet ampliar un objecte o bé fer una còpia.
 - `_inherit=objecte1`

```
# License AGPL-3.0 or later (http://www.gnu.org/licenses/agpl.html)

from odoo import fields, models

class Disc(models.Model):
    _inherit = 'disc'

    remastered = fields.Boolean(string="Remasterizado?")
```

Odoo - Herència

La capa de Vista en Odoo també permet l'herència entre una vista pare i una vista fill mitjançant l'ús de l'etiqueta <inherit_id>. On haurem d'indicar el XML ID de la vista pare.

També haurem d'indicar com buscar un camp de la vista pare a partir de la qual farem les modificacions. Utilitzarem l'etiqueta <field> i els atributs "name" i "position" amb aquest objectiu.

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

    <record model="ir.ui.view" id="gestion_musical_2_view_form">
        <field name="name">gestion.musical.2.view.form</field>
        <field name="model">disc</field>
        <field name="inherit_id" ref="gestion_musical.discos_view_form" />
        <field name="arch" type="xml">
            <field name="published_date" position="after">
                <field name="remastered"/>
            </field>
        </field>
    </record>
</odoo>
```

Aquí li dius després de quin atribut anirà

Odoo - Herència

L'atribut “position” permet ajustar com modificarem la vista pare des de el fill. Les opcions que em dona aquest atribut són:

inside — és el valor per defecte, annexa el camp al final de la vista pare.

replace — modifica el camp de la vista pare amb l'actual.

after — posiciona el camp actual darrera del camp pare.

before — posiciona el camp actual abans del camp pare.

attributes — modifica l'atribut indicat en el pare pel que indiquem en la vista fill.

Bibliografia

<https://odoo-new-api-guide-line.readthedocs.io/en/latest/>

<https://www.odoo.com/documentation/13.0/index.html>

<https://fundamentos-de-desarrollo-en-odoo.readthedocs.io/es/latest/capitulos/herencia-extendiendo-funcionalidad-aplicaciones-existentes.html>