Generalitat de Catalunya
Departament d'Educació
Institut Milà i Fontanals. Igualada

Dossier

Informàtica

ICB0 Desenvolupament d'aplicacions multiplataforma

M10 Sistemes de gestió empresarial

UF2 Sistemes ERP-CRM. Explotació i adequació.

Odoo16: Desenvolupament.

Isidre Guixà – Ester Marsal

Curs 2023/2024

Milà i Fontanals IGUALADA
Unió Europea
Fons Social Europeu
ISO 9001 BUREAU VERITAS Certification

Pròleg

Aquest dossier pretén ser la posta al dia de:

- apartats 1.3 i 2 del dossier **Sistemes ERP-CRM. Explotació i adequació I (DAM_M10_UF2_B1.pdf)**
- la totalitat del dossier **Sistemes ERP-CRM. Explotació i adequació I (DAM_M10_UF2_B2.pdf)**

donat que fan referència al programari obsolet OpenERP 6.1.

L'alumne només hauria de consultar dels dossiers esmentats, aquelles parts que aquí s'indiquin que continuen sent vàlides.

En aquest dossier, actualitzat per Odoo 16, apareixeran paràgrafs acolorits com aquest, que contenen informació per versions anteriors o informació prescindible per usar en aquest curs, però que és interessant mantenir.

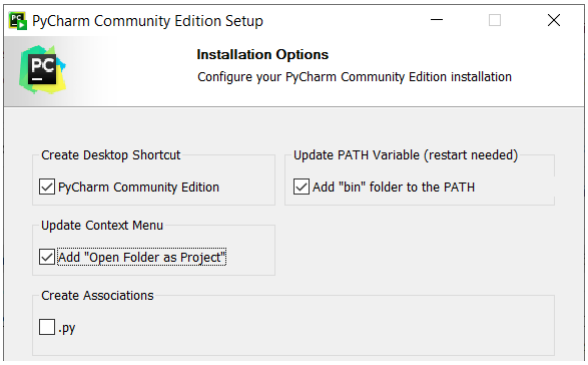
Llista de reproducció YouTube amb molts temes de desenvolupament en Odoo15

Índex

Instal·lació PyCharm com IDE per Odoo16 en Windows

Es suposa que l'alumnat té els coneixements de Python indicats abans de l'inici de la UF.

- Odoo16 està desenvolupat en Python 3.10.11 que resideix a la carpeta `python` dins `path_Odoo`.
- Com tot Python, hi ha alguns guions (`pip`, per exemple) dins subcarpeta `Scripts`
- En primer lloc, incorporar en el path del Windows les dues carpetes:
  ```
  pathOdoo\python
  pathOdoo\python\Scripts
  ```
  D'aquesta manera, des de qualsevol aplicació del sistema (`cmd`, per exemple) es podrà invocar qualsevol de les utilitats residents en aquestes carpetes.
- Comprovem funcionament de les eines. Obrim un `cmd`:
  - Executem `python`. Entrem a l'intèrpret de Python i s'informa de la versió. Sortim amb CTRL-Z
  - Executem `pip`. Ens informa de les ordres d'execució.
  - Si es vol, es pot instal·lar iPython (`pip install ipython`). Cal executar `pip` des de `cmd` elevat.
- Instal·lació i configuració de *PyCharm Community Edition*:
  - Descarreguem de la web i procedim a instal·lació.
  - En opcions d'instal·lació, marquem com la imatge.



  - Cal executar *PyCharm* com administrador. Per això és aconsellable configurar-ho en l'accés directe que queda a l'escriptori.
  - Posem en marxa *PyCharm*
  - Obrim (no crear) un projecte i seleccionem la carpeta `pathOdoo\server`.
  - *PyCharm* carrega tot el contingut de la carpeta `server` com a projecte i, per tant, tenim accés a totes les subcarpetes i fitxers. Per exemple, podem consultar i editar el fitxer `odoo.conf`.
  - Si es vol, es pot batejar el projecte `server` com a Odoo16 (`File | Rename Project`).
  - *PyCharm* es pot usar només com editor o, també, per executar i depurar codi.. En aquest 2n cas, cal aturar i desactivar el servei Odoo de Windows i configurar *PyCharm* com diu aquest video a partir del minut 16:46, quan ja tenim el projecte carregat a *PyCharm*, tenint en compte que:
    - Com a *Python Interpret* seleccionarem Python.exe instal·lat amb Odoo i que ja té totes les llibreries que necessita Odoo.
    - Cal crear una configuració per posar en marxa (llançadora) el projecte on:
      - *Name*: Odoo16 (o qualsevol nom que us sembli adequat)
      - *Script path*: `odoo-bin`
      - *Parameters*: `-c odoo.conf`
      - *Working directory:* ruta de la carpeta `server` (ruta del projecte)

      En *Script path* i *Parameters* no cal indicar la ruta per què els dos arxius es troben en el *Working directory* del projecte

  El vídeo configura *PyCharm* en un Odoo en Linux, però es passos són perfectament vàlids per Windows, on no serà necessari crear cap enllaç simbòlic, doncs tot està dins la carpeta `server`.

  Per comprovar que *PyCharm* està ben configurat, podem posar en marxa l'Odoo des de *PyCharm* (*Run Project*). A la part baixa de *PyCharm* (consola) veurem que l'Odoo s'ha posat en marxa però no surt cap missatge, per què estem bolcant els missatges al fitxer `odoo.log`. És altament recomanable comentar la propietat `logfile` dins `odoo.conf` (`;` davant) per a visualitzar els missatges dins la consola de *PyCharm*.

Dreceres de teclat interessants:
- Ctrl + símbolDividirTeclatNumèric = Comentar/Descomentar (Ctrl + / segons IDE)
- Ctrl + Alt + L = Reformatar el codi

- **I si els PDF d'Odoo surten en blanc? Causa i solució**

Si teniu un Odoo en Windows engegat de PyCharm i demaneu qualsevol informe dels que facilita Odoo (comanda, factura,...) és possible (no sempre) que us aparegui el PDF en blanc. Comproveu que si engegueu l'Odoo com a servei, l'informe es genera perfectament.

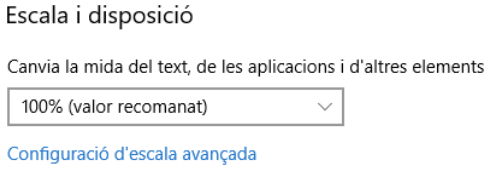
Situació: Tenir l'Odoo engegat com a procés (cosa que succeeix quan l'engeguem des de PyCharm)

Causa:

- En Odoo els PDF són generats per l'eina `wkhtmltopdf` (versió 0.12.1.2) que utilitza un valor DPI (Dots per Inch) per fer la conversió del codi HTML generat per Odoo cap el codi PDF.
- Quan Odoo està engegat com a servei, desconec d'on `wkhtmltopdf` agafa el valor DPI (i es generen els PDF correctament), però en cas que Odoo està engegat com a procés, `wkhtmltopdf` agafa el valor DPI de la configuració de pantalla de la sessió Windows que està executant el procés i si aquest valor no és adequat, el PDF es genera en blanc. Cada vegada que es demana un PDF, `wkhtmltopdf` avisa (en fitxer log o consola *PyCharm*) amb missatge:
  Generating PDF on Windows platform require DPI >= 96. Using 96 instead.
  Aquest missatge sembla que és un avís i informa que utilitzarà 96 per garantir el funcionament, però no és així, doncs no sempre genera els PDF correctament.
- En escriptoris Isard sembla que el problema només apareix quan usem protocol RDP i des de determinats monitors (sobretot, alguns tipus de portàtils). Per què???

Solució:

- O canviar el valor DPI a l'escriptori de Windows, cosa que en principi es fa via configuració de pantalla (botó secundari de ratolí sobre l'escriptori) i anant a canviar la mida del text en el camp que mostra la imatge.
  Si es tracta d'escriptori Isard i estem accedint a la màquina via RDP, Windows no permet modificar la mida del text...
  Podem intentar accedir a la màquina via "navegador" o via "SPICE" i canviar el valor. En aquest enllaç s'explica altres mecanismes per modificar els DPI de l'escriptori.
  En teoria 100% correspon a 96DPI. En Windows, el valor pot anar des de 96 fins 480 (500%).



- O, sembla que també funciona i és més senzill, entrar en "Configuració d'escala avançada", i desactivar la correcció de l'escala de les aplicacions.

Desenvolupament en Odoo 16 - Backend

El desenvolupament en Odoo té 2 vessants:
- Backend, basat en patró de disseny MVC (el què veiem a DAM)
- Frontend, que permet "donar vida" a la vista dissenyada en el backend via desenvolupament web, que precisa de coneixements de Javascript, HTML5 i CSS (no ho veurem a DAM)

L'aprenentatge d'aquest mòdul l'efectuarem dissenyant mòduls, que aniran evolucionant a mida que hi anem afegint contingut. Per veure l'evolució, aquest dossier anirà acompanyat de les diverses versions, seguint la següent nomenclatura.

Donat un mòdul de nom tècnic `xxx` (nom de la carpeta que el conté), donat que la carpeta no pot canviar de nom, les diferents versions del mòdul que acompanyaran aquest dossier tindran els noms:
`01_xxx.zip` `02_xxx.zip` `03_xxx.zip` `...`
però cada arxiu contindrà en el seu interior la carpeta de nom `xxx`.

Documentació oficial: https://www.odoo.com/documentation/16.0/

Normes importants de codificació:
- Tot en UTF-8 sense BOM
- Si es codifica via editor "simple", configurar-lo per a que els tabuladors siguin substituïts per espais en blanc (habitualment 4)

Estructura d'un mòdul

En aquest vídeo (fins minut 20:30) s'explica com crear un mòdul, les parts que té, on ubicar-lo,... en un Odoo en Linux. És perfectament vàlid per a Odoo en Windows. Mostra +/- el què s'explica a continuació.

Dissenyem l'esquelet bàsic per a un mòdul de nom `school` que anirem desenvolupant al llarg de la UF:

- Arxiu `__manifest__.py`
- Carpetes `models` i `views`, de moment buides.
- Arxiu `__init__.py`, tenint en compte la necessitat de la seva existència a cada carpeta que contingui codi Python

Desenvolupem una primera versió de mòdul `school` per a la gestió d'una escola, que conté únicament la informació bàsica del mòdul (`__manifest__.py`).

Si dins `manifest` hi tenim `application` a `True`, el veurem com aplicació a l'Odoo i si `application` no hi és o està a `False`, el veurem com a mòdul.

Si es vol assignar una icona, ha d'estar a ruta `static/description` amb nom `icon.png`. En versions anteriors (Odoo13-15) la icona havia d'estar en format `svg` (*Scalable Vector Graphics*) i en aquests casos habitualment aquesta carpeta també incorporava la icona en format `png` per a veure-la des del sistema de fitxers. En Odoo16 no cal `svg`. A la web hi ha moltes aplicacions gratuïtes de conversió `png` a `svg`, com per exemple https://convertio.co/png-svg/

On incorporar la informació del mòdul?
- En un fitxer de nom `index.html` ubicat dins `static/description`.
- El camp `description` de `__manifest__.py` és obsolet i només s'utilitza (probablement per què no s'han adonat) quan no existeix `index.html` i el mòdul no és `application`.

Per tant, incorporarem la informació del mòdul en el fitxer `index.html`.

Versió 1 del mòdul: `01_school.zip`.

Després d'actualitzar la llista d'aplicacions, apareix com a mòdul o com a aplicació segons valor dins `manifest`. El podem instal·lar, però no apreciarem res des de l'Odoo.

Si no s'instal·la, mireu final de fitxer `odoo.log` (o la consola de *PyCharm* si hem desactivat el log)

Oddo proporciona la possibilitat de generar un esquelet bàsic per a un mòdul. Per Odoo en Windows:

1) Obrir una consola de sistema (`cmd` com administrador o terminal en *PyCharm*).
2) Situar-se a la carpeta on hi ha l'Odoo
   En el cas d'Odoo 16, dins hi trobem les carpetes `python` i `server` entre altres.
3) Cal executar el servidor d'Odoo (`odoo-bin`) que es troba dins la carpeta `server` passant-li 3 paràmetres. El servidor `odoo-bin` és un programa Python i ha de ser executat amb `python.exe` que es troba a la carpeta `python`. Per tant, per usar l'eina (`scaffold`) que crea el mòdul:

   Des de `pathOdoo`: `python\python.exe server\odoo-bin scaffold nomModul rutaDesti`
   Des de la carpeta `server`: `..\python\python.exe odoo-bin scaffold nomModul rutaDesti`
   on a `rutaDesti` cal indicar on l'eina ha de crear la carpeta `nomModul` amb el contingut inicial.

Nomenclatura bàsica

- En anglès.
- Disseny de model en fitxers Python.
- Nom de la classe segueix notació Java i acostuma a coincidir amb el valor de la clàusula `_name`.
- Clàusula `_name`: en minúscula, amb mots compostos separats per . i amb el nom del mòdul com a prefix. Exemples: `school.teacher`, `school.course`, `school.subject`

Camps bàsics i vistes bàsiques (`tree` i `form`)

Camps: https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#fields

- Els paràmetres de la classe inicial `odool.fields.Field` són els comuns per a qualsevol tipus de dada que ens permet utilitzar Odoo
- Els tipus de dada que proporciona Odoo, els podem veure a les classes posteriors, amb els paràmetres propis de cada tipus de dada, que s'afegeixen als comuns:
  `Char`, `Boolean`, `Integer`, `Float`, `Text`, `Selection`, `Html`, `Date`, `Datetime`

Exercici:

Incorporar al mòdul `school` la gestió de les classes del disseny `02_school_UML.pdf` adjunt.

A tenir en compte:

Decidir classes en Odoo: `02_school_UML_Odoo.pdf` adjunt

Primera utilització de camps especials: `active`

Utilització de paràmetre `default` per introduir valor per defecte. Important en camps `active`, per assegurar que en crear un registre, quedi activat, que és el lògic.

Per actualitzar un mòdul, cal:

1. Substituir en `addons` carpeta per nova versió
2. Reiniciar servidor Odoo
3. Actualitzar llista de mòduls
4. Actualitzar el mòdul

Això és un procés molt llarg si estem desenvolupant...

- Procés d'actualització de mòdul en versions anteriors a Odoo 13 sense usar *PyCharm:*
  - Servei Odoo aturat
  - Des de consola de sistema, engegar Odoo com a procés indicant què actualitzar:
    `python\python.exe server\odoo-bin –u<nomMòdul> -d<nomBD>`
    Si s'està desenvolupant varis mòduls simultàniament i es vol actualitzar tots:
    `python\python.exe server\odoo-bin –u<nomMòdul1,nomMòdul2,...> -d<nomBD>`
  - Per aturar-lo, prémer `CTRL-C` dues vegades a la consola per matar/avortar el procés.

- Procés d'actualització de mòdul en versió Odoo 13-16 **si no usem un IDE com *PyCharm***.
  - Via reiniciar servei Odoo retocant la instrucció que SO executa en activar el servei.

  En Odoo13-16, el servei instal·lat en Windows corresponent a Odoo, està gestionat per:
  `<RutaOnEsOdoo>\nssm\win64\nssm.exe`

  1. Atureu servei Odoo.
  2. Obriu una consola de sistema i anar a la carpeta on és l'executable anterior.
  3. Executar `nssm edit odoo-server-versio` (nom del servei de Windows)
  4. S'obre una finestra amb la informació del servei. En el camp *Arguments* afegir, al final (després de les " de la ruta on es troba odoo-bin) els arguments:
     `-u<nomMòdul1,nomMòdu2,...> –d<nomBD>`

Alerta: `<nomMòdul>` és el nom (tècnic) de la carpeta que conté el mòdul

5. Enregistreu els canvis prement el botó *Edit Service*.

ATENCIÓ: Ara, cada vegada que engegueu-reinicieu el servei, s'actualitzarà sempre el(s) mòdul(s) indicat en la base de dades indicada. Si voleu usar Odoo de manera "normal", torneu a editar el servei i elimineu els arguments indicats.

El més adequat seria deixar el servei original `odoo-server-versió` aturat i en mode manual i crear un nou servei X com a còpia de l'original, i usar aquest servei X per activar Odoo. Hi ha versions més noves de `nssm` que ho permeten...

- Procés d'actualització de mòdul en versions 13-16 **si usem *PyCharm***:
  1. Servei Odoo aturat
  2. Crear una nova configuració de llançadora del projecte, configurant els paràmetres com: `-c odoo.conf -u<nomMòdul1,nomMòdul2,...> -d<nomBD>`

  ATENCIÓ: Ara, cada vegada que poseu en marxa el projecte amb aquesta llançadora, s'actualitzaran el(s) mòdul(s) indicat(s) en la base de dades indicada.

ATENCIÓ: En qualsevol dels processos indicats, quan l'Odoo s'engega amb `-d<nomBD>`, només permet treballar amb aquesta BD. Les altres no apareixeran en el selector de BD. Les podreu crear (i es crearan, cosa que es pot comprovar via *pgAdmin* o *psql*) però no es poden usar, a no ser que s'elimini el paràmetre `-d` de l'arrencada o que canviï el `<nomBD>` pel nom de nova BD.

Una vegada instal·lat un mòdul amb alguna classe, comprovar a la BD l'aparició de les taules!!!

Camps d'auditoria: `create_uid, create_date, write_uid i write_date`

- Es pot desactivar l'aparició via clàusula `_log_access=False`
- No la desactivarem per què apareixen problemes (comprovats en Odoo11)

Si no hi ha vista, no s'observa que el mòdul està instal·lat.

Desenvolupament bàsic de vista en fitxer XML

- Vistes `list(tree)` i `form`
- Cal invocar-les des d'una `action`
- Arbre de menús amb elements `menuitem`.
- Les opcions de menú (`menuitem`) que executen una programa, han d'invocar una `action`.
- Un menú o submenú que no incorpori cap opció que executi alguna vista, no és visible!!!
- No es pot fer referència a una vista-action-menu que no estigui definida prèviament, doncs Odoo carrega tots aquests elements en el sistema de forma seqüencial !!!

Els mòduls desenvolupats, per a ser accessibles des dels usuaris, hauran de tenir un esquema de seguretat (grups de privilegis...). Però això es fa al final, quan s'ha desenvolupat tot el mòdul. Llavors, com visualitzar el mòdul que estem desenvolupant? Doncs activant-nos com a "superusuari", seguint els passos següents:

1. Activar mode desenvolupador
2. Prémer damunt la "marieta" que apareix a la barra superior, a l'esquerra del nom d'usuari
3. Escollir l'opció "Converteix-te en superusuari", que activa l'usuari OdooBot i permet accés total a tots els mòduls instal·lats.

- Desenvolupament de les vistes `tree` per `Course`, `Subject` i `Teacher`.

  Documentació: https://www.odoo.com/documentation/16.0/developer/reference/backend/views.html#list

- Comentari respecte la vista `tree` de `school.course`:

  Hem incorporat els camps d'auditoria per demostrar que són camps que es poden visualitzar. Si observem el contingut dels camps dins al taula `school_course`, observem:
  - Els camps `_uid` contenen el `id` de l'usuari que els ha creat. Contenen 1 que és l'identificador de l'usuari `OdooBot` (superusuari), com es pot veure a la taula `res_users`. Però en incorporar-los a una vista, no mostra el valor numèric, sinó el nom de l'usuari!!!
  - Els camps `_date` contenen el valor UTC del moment en què es va produir la creació o la modificació. Si no es fa cap tipus de format d'aquests camps, l'usuari els veu segons la zona horària del seu ordinador.

- Els nodes `<tree>` i `<form>` de les vistes `tree` i `form`, poden incorporar l'atribut `string="..."` amb l'objectiu de donar nom a aquestes vistes. De moment observem que malgrat posem aquest atribut, no s'aprecia cap canvi en visualitzar les vistes; apareix el títol que s'hagi donat a l'acció que invoca les vistes. Per tant, de moment no l'usarem i potser, més endavant, ens trobarem amb la necessitat.

- Les vistes `tree` permeten, des de Odoo13, incorporar columnes opcionals, accessibles via icona a l'extrem dret de la fila de títols de la vista tree, només visible si la vista incorpora camps opcionals. Aquestes columnes es poden fer visibles o no visibles. Cal usar l'atribut `optional` amb valors `"show"` o `"hide"`. Ho hem aplicat en la vista `tree` de `teacher`.

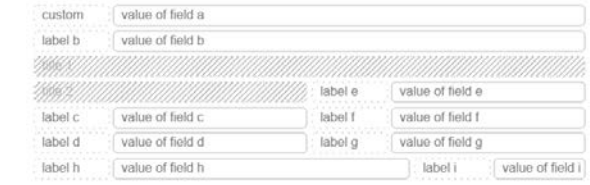
- Important emprar l'atribut `sequence` en els `menuitem` per assegurar l'ordre de visualització.

- Desenvolupament de les vistes `form` per `Course`, `Subject` i `Teacher`.

  Documentació: https://www.odoo.com/documentation/16.0/developer/reference/backend/views.html#form

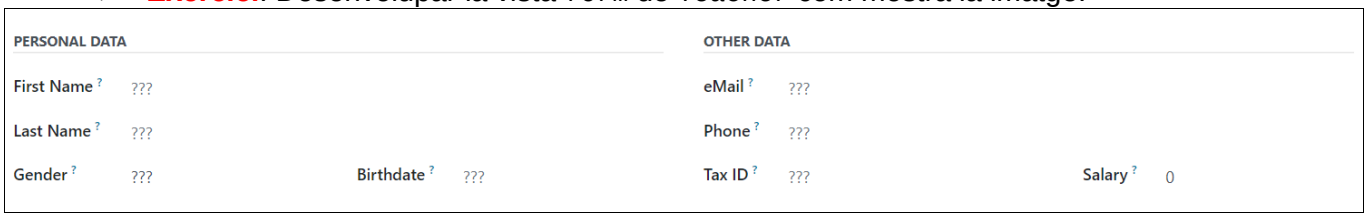
  - Principals elements a tenir en compte en vistes `form`: `sheet`, `group`, `separator`, `newline`
  - Posar especial atenció amb l'explicació de l'element `group`.
  - Respecte l'element `newline`, l'explicació és errònia/incompleta, doncs permet obligar a que el `group` següent canviï de fila, però no té efecte entre els `fields` de dins un `group`.
  - Exemples de renderització (extrets de la documentació d'Odoo 17):


```
<form>
    <group string="Title 1">
        <group string="Title 1.1">...</group>
        <newline/>
        <group string="Title 1.2">...</group>
        <group string="Title 1.3">...</group>
    </group>
</form>
```




```
<group>
    <field name="a" string="custom"/>
    <field name="b"/>
</group>
<group string="title 1">
    <group string="title 2">
        <field name="c"/>
        <field name="d"/>
    </group>
    <group>
        <field name="e"/>
        <field name="f"/>
        <field name="g"/>
    </group>
</group>
<group col="12">
    <group colspan="8">
        <field name="h"/>
    </group>
    <group colspan="4">
        <field name="i"/>
    </group>
</group>
```

- Dins un `group` només hi pot haver `field` o `group`, però no poden conviure `group` i `field` dins un group, doncs els efectes visuals són fatídics.
- Sembla que usar un 3r nivell de `group` també provoca efectes visuals fatídics.
- **Exercici**: Desenvolupar la vista `form` de `Teacher` com mostra la imatge:



A tenir en compte:

- Utilització de ginys (*widgets*) per als camps que ho precisen (`email`, `url`,...).
  Relació de ginys possibles dins taula 2.3 del dossier `DAM_M10_UF2_B1.pdf`
  +info: https://www.cybrosys.com/blog/widgets-in-odoo
  +info: https://odoo-dev.readthedocs.io/en/latest/widgets/field_widgets.html
- Entre els atributs que es pot assignar a un camp en el model, està `readonly`, per defecte té valor `False`. En cas d'utilitzar-lo, aquest serà el valor per defecte a totes les vistes en les que aparegui aquest camp, però a cada vista es pot canviar el seu valor.
- Atributs que es poden usar en un `field` a la vista:
  - `readonly` amb valors `"1"/"True"` (activat) o `"0"/"False"` (desactivat)
  - `required` amb valors `"1"/"True"` (activat) o `"0"/"False"` (desactivat)
  - `invisible` amb valors `"1"/"True"` (activat) o `"0"/"False"` (desactivat)
  - `placeholder`, per visualitzar la posició del camp fent-hi aparèixer un text adequat.
  - `string`, per canviar el valor per defecte de l'etiqueta (definida en el model)

Resultat: `02_school.zip`

- **Com desactivar creació-eliminació-modificació en vistes?**

- L'atribut `create="0"` en node `tree` desactiva el botó per crear nous registres.
- L'atribut `delete="0"` en nodes `tree`/`form` desactiva l'opció per eliminar registres.
- L'atribut `edit="0"` en node `form` desactiva l'opció d'edició del registre.

- **Com canviar el nom d'un camp en el model?**

Des de Odoo8 fins Odoo12, si un camp de nom `xxx` ja instal·lat (per tant, a la corresponent taula hi ha la columna xxx) es volia reanomenar per nom `yyy`, es disposava del paràmetre `oldname` com segueix:

```
yyy = fields.Tipus (..., oldname='xxx')
```

i en la següent instal·lació del mòdul, Odoo s'encarregava de reanomenar la columna `xxx` per `yyy`, sense perdre les dades existents.

En Odoo13+ s'ha eliminat aquesta funcionalitat. La incorporació del paràmetre `oldname` no té cap efecte. Per tant, en cas de voler canviar el nom d'un camp prèviament instal·lat, caldrà fer-ho manualment:

- Canviar en nom en el model i en les vistes afectades
- Aturar el servidor Odoo.
- Anar manualment a la BD i reanomenar la columna afectada. Per exemple:
  `alter table <nomTaula> rename column <nomVell> to <nomNou>;`
- Engegar el servidor Odoo i actualitzar el mòdul

- **Camps traduïbles**

Odoo sempre ha permès tenir camps definits com a traduïbles, els quals es poden traduir als diversos idiomes que tingui activada la BD-empresa. Si la BD-empresa només treballa en 1 idioma, no te sentit traduir el valor del camp i no s'activa el mecanisme per fer-ho.

Quan la BD-empresa té varis idiomes actius, els camps traduïbles mostren una icona idiomàtica a la dreta del camp d'edició. Aquesta icona ha anat canviant en les versions d'Odoo. En versions més antigues era una bola del món mentre que ara és el codi ISO de l'idioma que té actiu l'usuari. Prement aquesta icona s'obre una finestra on es pot traduir el valor als diversos idiomes actius.

Fins Odoo15, els camps traduïbles eren emmagatzemats a la BD en camps `character varying` (cadena) i les traduccions s'emmagatzemaven en una taula de traduccions.

En Odoo16+, els camps traduïbles s'emmagatzemen a la BD en camps `jsonb`. Per exemple:

```
{"ca_ES": "Desenvolupament", "es_ES":"Desarrollo","en_US":"Development"}
```

En inserir un nou registre, Odoo emmagatzema el valor del camp traduïble en l'idioma que està usant l'usuari que efectua l'alta i també incorpora el mateix valor en idioma `"en_US"`, que s'usarà en cas que s'activi un nou idioma a la BD-empresa i encara no estiguin els valors traduïts al nou idioma.

Per a fer un camp traduïble, cal incorporar l'atribut `translate=True` en la seva definició en el model.

En cas de convertir un camp no traduïble (amb registres creats a la BD) en un camp traduïble, la columna passa a ser `jsonb` i els valors existents es transformen en `{"en_US":valor}`.

En cas de convertir un camp traduïble (amb registres creats a la BD) en un camp no traduïble, la columna passa a ser character varying amb el valor que tenia en idioma `"en_US"`.

Com gestionar camps `jsonb`.: https://www.postgresql.org/docs/12/functions-json.html

Camps relacionals

La implementació en Odoo de relacions entre classes s'efectua amb camps relacionals.

Exercici:

Incorporar al mòdul `school` les classes del disseny `03_school_UML.pdf` adjunt.

A tenir en compte:

La relació `*:1` s'implementa amb camp `Many2one`
La relació `1:*` s'implementa amb camp `One2many` i necessita l'existència de la `Many2one` inversa.
La relació `*:*` s'implementa amb camp `Many2many` i obliga a dir nom de la taula per PostgreSQL

+Info: https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#relational-fields

Nomenclatura per camp `Many2one`: `nomEntenedorOnApunta_id`
Nomenclatura per camps `One2many` i `Many2many`: `nomEntenedorOnApunta_ids`

Corresponent disseny de classes en Odoo: `03_school_UML_Odoo.pdf`

Atenció!!!

Si els camps relacionals apunten a una classe d'un altre mòdul M, és obligatori que aquest mòdul M estigui informat a l'apartat `depends` del fitxer `__manifest__.py`, malgrat el mòdul M estigui instal·lat a l'empresa. Del contrari, provoca errors diversos (quan s'aplica `domain` sobre el camp, quan es fa un camp `related` a partir del camp... conceptes que es veuran més endavant).

- **Camp `Many2one`**

`fields.Many2one('classeApuntada','etiqueta',...)`

En la corresponent taula de PostgreSQL és un camp FK de la taula corresponent a la classe apuntada.

El podem ubicar a qualsevol vista.

Quina informació mostra del registre de la classe apuntada? El primer possible dels següents casos:

- Resultat del mètode `name_get` si està definit a la classe apuntada
- Camp indicat per la clàusula `_rec_name` en el disseny de la classe
- Camp `name` en el disseny de la classe
- Nom de la classe seguit del valor del camp `id` del registre apuntat.

En l'exercici que cal desenvolupar, hem d'incorporar a la classe `school.course` el camp `Many2one` de nom `manager_id` apuntant a `school.teacher`. Aquesta classe no te camp `name`. Observeu què mostra el camp a les vistes si no incorporem `_rec_name` (`<school.teacher,1>`...). El lògic serà que mostri la concatenació del cognom i del nom del registre apuntat, fet que es pot aconseguir amb el mètode `name_get`, que dissenyarem més endavant. Ara incorporem `_rec_name='last_name'`.

- **Camp `One2many`**

Necessita l'existència de camp `Many2one` invers a la classe apuntada.

`fields.One2many('classeApuntada','campInvers','etiqueta',...)`

En la corresponent taula de PostgreSQL no deixa cap rastre.

En una vista `tree` (no habitual), mostra el nombre de registres apuntats.

En una vista `form` mostra una graella amb els registres apuntats.

El giny per defecte que porten assignat és `one2many`.

- Per defecte, la graella conté totes les columnes de la classe.
- Per indicar les columnes que ha d'incorporar la graella, cal incorporar un element `tree` dins l'element `field`:

```
<field name="campOne2many" ...>
  <tree>
    <!-- camps_a_visualitzar -->
    <field ... />
    <field ... />
    ....
  </tree>
</field>
```

- Per defecte, el camp ocuparà 2 columnes (1 per l'etiqueta i 1 per la graella). Les graelles acostumen (no obligatori) a fer-se visibles sense etiqueta (atribut `nolabel="1"`) i llavors expandeixen a 2 columnes i via element `separator`, que per defecte ocupa 1 columna, es posa un títol a la fila anterior:

```
<separator string="títol" colspan.../>
<field name="campOne2many" ...>
```

- La graella és editable si i només es dona una de les condicions següents:
  - En el model el camp incorpora `readonly=False` (o no té `readonly` -per defecte `False`-).
  - Si el camp està definit amb `readonly=True` però la vista incorpora atribut `readonly="0"`

- En cas que la graella sigui editable, observem que al final dels registres apareix l'opció *Afegir una línia* que sembla que hauría de permetre accedir als cursos existents per tal de seleccionar-ne un i assignar-lo al professor en el que estem ubicats. En canvi, si premem aquesta opció, passem a crear un nou curs que quedarà assignat al professor actual. Aquest és el funcionament del giny `one2many` que porta associat un camp `one2many`. En versions antigues d'Odoo, l'opció que apareixia no era *Afegir una línia* sinó *Crear*, fet que deixava més clara la funcionalitat.

  Per tant, és molt aconsellable canviar el giny per defecte pel giny `many2many`, que també mostra opció *Afegir una línia* però que en aquest cas, en permer-la, apareix la relació de tots els cursos existents –menys els que ja té assignats el professor- per poder seleccionar els que calgui i, a més, facilita un botó *Crear* per si es vol procedir a crear un nou curs.

  Informació més detallada a l'apartat *Giny `one2many` o `many2many`?*

- En cas de graella editable, si procedim a crear un nou registre de la classe apuntada, apareix la vista `form` de la classe apuntada, fet que possiblement no interessi doncs apareixen camps redundants. Es pot canviar el disseny de la vista `form` a visualitzar, incorporant un element `form` sota l'element `tree` dins el camp `field`:

```
<field name="campOne2many" ...>
  <tree>
    <!-- camps_a_visualitzar -->
    <field ... />
    <field ... />
    ....
  </tree>
  <form>
    <!-- disseny que correspongui -->
  </form>
</field>
```

- **Camp `Many2many`**

Les classes d'Odoo s'implementen en PostgreSQL en taules que tenen per clau primària el camp `id`.

La implementació d'un camp `Many2many` dins PostgreSQL genera una nova taula que, de forma similar a les entitats N:N del model E-R, té per clau primària la parella de claus primàries de les taules relacionades. Odoo obliga, en la definició del camp `Many2many`, a:

- Donar nom a la taula, que acostuma a contenir els noms de les classes relacionades.
- Donar nom a cadascuna de les dues columnes de la clau primària
- Cada columna de la clau primària és FK de la taula corresponent a la classe apuntada.

```
fields.Many2many('classeApuntada','nomTaula','colA_PK','colB_PK','etiqueta',...)
```

Exemple de camps `Many2many` entre les classes `Teacher` i `Subject` (disseny `03_school`):

- A la classe `Teacher`:

```
subject_ids = fields.Many2many('school.subject','school_subject_teacher_rel',
                               'teacher_id','subject_id','Subjects authorized'...)
```

- A la classe `Subject`:

```
teacher_ids = fields.Many2many('school.teacher','school_subject_teacher_rel',
                               'subject_id','teacher_id',Authorized Teachers',...)
```

No és obligatori l'existència dels dos camps; només n'hi haurà 2 si la relació és bidireccional.

El nom de la taula ha de ser el mateix en els dos camps (si existeixen els 2 camps). S'acostuma a anomenar amb el nom del mòdul (`school`) seguit pels noms de les dues classes (`subject_teacher` o `teacher_subject`) seguit del sufix `_rel`.

El tercer paràmetre és el nom de la columna que conté l'identificador (FK) de la pròpia taula.
El quart paràmetre és el nom de la columna que conté l'identificador (FK) de la taula apuntada.

La visualització dels camps `Many2many` en les vistes és similar a la dels camps `One2many`.

El giny per defecte que porten assignat és `many2many`.

- **Giny `one2many` o `many2many`?**

Un camp `Many2many`, si no s'indica el contrari, en una vista form té assignat un giny `many2many` i un camp `One2many` té assignat un giny `one2many`, però aquesta assignació per defecte es pot canviar i cal tenir clar quin és el comportament dels camps `Many2one`/`One2many` i dels ginys `many2one`/`one2many` quan el camp és editable, doncs si és de només consulta, no hi ha distinció en la visualització.

Suposem que tenim registre RY de classe Y amb un camp `_ids` que apunta a la classe X.

- Si el camp és `Many2many`, independent del giny que tingui assignat, mostrarà:
  - Els registres RX apuntats de la classe X, amb una creueta *x* a la seva dreta.

    Aquesta creueta serveix per eliminar la relació que hi ha entre el registre RY i el registre RX de la graella. No elimina ni el registre RX ni el registre RY. A nivell de taules, elimina la fila de la taula "pont" entre les classes X i Y.
  - L'opció *Afegir una línia* al final.

  La creueta *x* i l'opció *Afegir una línia* no es poden fer desaparèixer.
  - Si el giny que acompanya el camp és `many2many` (per defecte), en prémer l'opció *Afegir una línia* apareix una finestra amb tots els registre de la classe X susceptibles de ser escollits, amb la possibilitat de seleccionar-los o, de *Crear* un nou registre de la classe X per a ser seleccionat.

    L'opció *Crear* es pot fer desaparèixer en cas que el camp vagi acompanyat d'un node `tree` per indicar els camps de la classe X que ha de mostrar la graella, incorporant l'atribut `create="0"` en la seva definició. És a dir:

```
<field name="..._ids" readonly="0" widget="many2many">
   <tree create="0">
      ...
   </tree>
</field>
```

  - Si el giny que acompanya el camp és `one2many`, en prémer l'opció *Afegir una línia* apareix el formulari per crear un nou objecte de la classe X que, en cas de proseguir, quedarà incorporat a la graella. El formulari que s'invoqui ha d'incorporar tots els camps de la classe X que siguin obligatoris i que no tinguin valor assignat per defecte.
- Si el camp és `One2many`, mostrarà:
  - Els registres RX apuntats de la classe X,
    - Amb una icona *paperera* a la seva dreta si el giny assignat és `one2many`

    Aquesta *paperera* serveix per intentar eliminar de la BD el registre RY seleccionat de la graella.

    L'icona *paperera* es pot fer desaparèixer en cas que el camp vagi acompanyat d'un node `tree` per indicar els camps de la classe X que ha de mostrar la graella, incorporant l'atribut `delete="0"` en la seva definició. És a dir:

```
<field name="..._ids" readonly="0" widget="one2many">
   <tree delete="0">
```

```
      ...
   </tree>
</field>
```

- Amb amb una creueta *x* a la seva dreta si el giny assignat és `many2many`

Aquesta creueta serveix per eliminar l'assignació del registre RX al registre RY, fet que serà possible únicament si el camp `Many2One` de RX a RY invers de `_ids` no és obligatori. No elimina el registre RX, simplement intenta deixar-lo sense assignació a RY.

La creueta *x* no es pot fer desaparèixer.

- L'opció *Afegir una línia* al final.
  - Si el giny que acompanya el camp és `many2many`, en prémer l'opció *Afegir una línia* apareix una finestra amb tots els registre de la classe X susceptibles de ser escollits, amb la possibilitat de seleccionar-los o, de *Crear* un nou registre de la classe X per a ser seleccionat.

    En cas d'optar per *Crear*, apareix el formulari per crear un nou objecte de la classe X que, en cas de proseguir, quedarà incorporat a la graella. El formulari que s'invoqui ha d'incorporar tots els camps de la classe X que siguin obligatoris i que no tinguin valor assignat per defecte.

    L'opció *Crear* es pot fer desaparèixer en cas que el camp vagi acompanyat d'un node `tree` per indicar els camps de la classe X que ha de mostrar la graella, incorporant l'atribut `create="0"` en la seva definició. És a dir:

```
<field name="..._ids" readonly="0" widget="many2many">
   <tree create="0">
      ...
   </tree>
</field>
```

  - Si el giny que acompanya el camp és `one2many`, en prémer l'opció *Afegir una línia* apareix el formulari per crear un nou objecte de la classe X que, en cas de proseguir, quedarà incorporat a la graella. El formulari que s'invoqui ha d'incorporar tots els camps de la classe X que siguin obligatoris i que no tinguin valor assignat per defecte.

    L'opció *Afegir una linia* es pot fer desaparèixer en cas que el camp vagi acompanyat d'un node `tree` per indicar els camps de la classe X que ha de mostrar la graella, incorporant l'atribut `create="0"` en la seva definició. És a dir:

```
<field name="..._ids" readonly="0" widget="one2many">
   <tree create="0">
      ...
   </tree>
</field>
```

En definitiva, és aconsellable:

- Usar giny `many2many` quan es vulgui facilitar seleccionar registres de la classe RX ja existents per a fer una assignació.
- Usar giny `one2many` quan es vulgui facilitar la creació de registres de la classe RX per a fer una assignació, per exemple en una relació `One2many` resultat d'una composició, on els registres de la classe RX només tenen raó d'existir si pengen d'un registre de la classe RY que no pot canviar.

+Info sobre ginys: https://www.cybrosys.com/blog/many2many-fields-and-its-widgets-odoo

Exercici:

Finalitzar la implementació de la versió 03 del mòdul incorporant:

- Dins el formulari d'Assignatures, la graella de Cursos i de Professors
- Dins el formulari de Professors, la graella d'Assignatures

Requeriments per a les relacions `one2many` – `many2many` existents en disseny `03_school`:

- Des de `Course` poder veure/assignar/crear assignatures que en formen part
- Des de `Subject` poder veure cursos però no assignar-crear-eliminar
- Des de `Subject` poder veure el professorat que pot impartir, sense assignar-crear-eliminar
- Des de `Teacher` poder veure/assignar les assignatures que pot impartir, sense crear-eliminar
- Des de `Teacher` poder veure/assignar els cursos amb responsabilitat, sense crear-eliminar

Resultat del disseny `03_school`: `03_school.zip`.

Observacions a la solució:

- En relació a la gestio a través de les relacions `one2many` i `many2many`, suposem que la relació `one2many` i les 4 relacions `many2many` tenen `readonly=True` en el model. Llavors, per aconseguir els requeriments demanats:
  - En `course_form`, camp `subject_ids` amb `readonly="0"`.
  - En `subject_form`, camp `course_ids` sense res a nivell de `readonly` (ja està en el model)
  - En `subject_form`, camp `teacher_ids` sense res a nivell `readonly` (ja està en el model)
  - En `teacher_form`, camp `subject_ids` amb `readonly="0"` i `tree create="0" delete="0"`.
  - En `teacher_form`, camp `course_ids` amb `widget="many2many"` i `readonly="0"` i `tree create="0"`
- Com aconseguir que dues graelles (camps `One2many` o `Many2many`) apareguin en una vista `form` una al costat de l'altra? Definint cada graella (i altres camps que poguessin interessar) en un `group` dins el `group` principal (veure `view_subject_form` i `view_teacher_form`).

Pestanyes en un formulari

Una zona de pestanyes en una vista `form` s'acostuma a utilitzar quan el formulari reuneix molta informació que és difícil fer encabir en una pantalla i, a més, permet una millor organització de la informació en el formulari. Veure, per exemple, la vista `form` de clients/proveïdors.

Per generar una zona de pestanyes, cal usar l'estructura:

```
<notebook colspan="...">
  <page [name="..."] string="...">
     ...contingut de la pestanya
  </page>
  <page [name="..."] string="...">
     ...contingut de la pestanya
  </page>
  ... i així amb totes les pestanyes que calgui...
</notebook>
```

És aconsellable donar nom a les pestanyes (atribut `name`) per facilitar ubicació en cas d'herència (més endavant). El valor dels atributs `name` no es tradueixen mai (i per això es poden referenciar en el codi), mentre que el valor dels atributs `string` es tradueixen en el procediment de traducció.

Si una pestanya no conté atribut `string`, Odoo mostrarà el contingut de l'atribut `name` per

En la versió `04_school` s'ha reubicat les dues graelles que hi havia en el formulari de professorat, en una zona amb dues pestanyes, on cada pestanya recull una de les graelles inicials i, a més, ocupant només la meitat esquerra de la pestanya, per si cal afegir més contingut a la meitat dreta.

Relacions reflexives

Una relació reflexiva s'implementa amb una parella de camps `Many2many` o amb una parella de camps `Many2One` i `One2many`.

Exercici:

Incorporar al mòdul `school` les novetats del disseny `05_school_UML.pdf` adjunt.
Feu que el nom de temàtica sigui traduïble.

A tenir en compte:

- Seria del tot il·lògic crear una classe per gestionar els països, quan Odoo ja la té (`res.country`).
- La classe `res.country` forma part del mòdul base que s'instal·la sempre en la creació d'una empresa. Per tant, no cal incorporar cap altre mòdul a la llista `depends` de `__manifest__.py`.
- Respecte la relació reflexiva sobre la classe `Temàtica`, els camps podrien anomenar-se `supertheme_id` i `subthemes_ids`, però Odoo incorpora dos camps específics per aquest menester, que ens facilitaran funcionalitats més endavant: `parent_id` i `child_ids`.

Per tant, el model UML en Odoo és `05_school_UML_Odoo.pdf`.

Respecte la vista, incorporar:

- Opció `Thematics` dins menú `Configuration`.
- Vista `tree` de `Themàtic`:

Name	Parent Thematic	Child Thematics	Courses
Informàtica		1 registre	1 registre
Programació	Informàtica	Cap registre	Cap registre

- Vista `form` de `Thematic`:

Name Informàtica
Parent Thematic

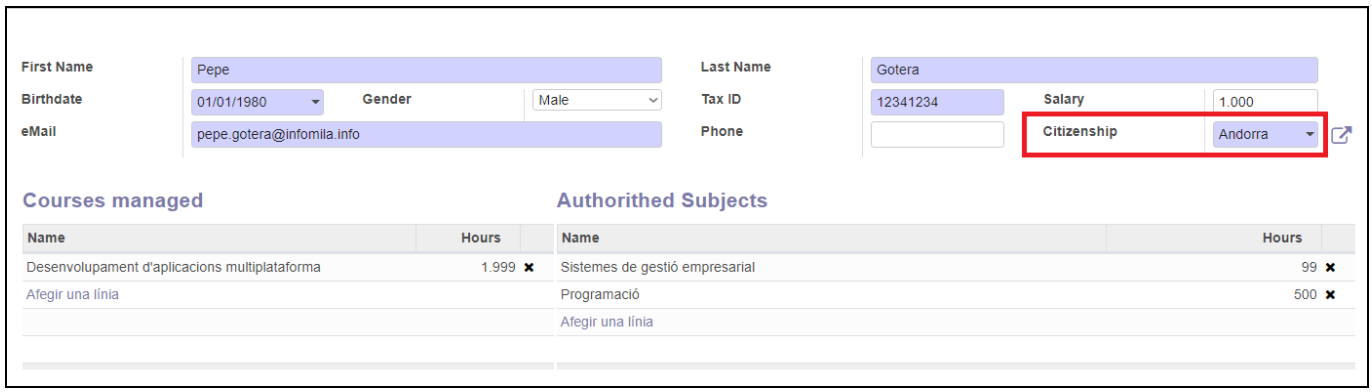
Child Thematics

Name
Programació

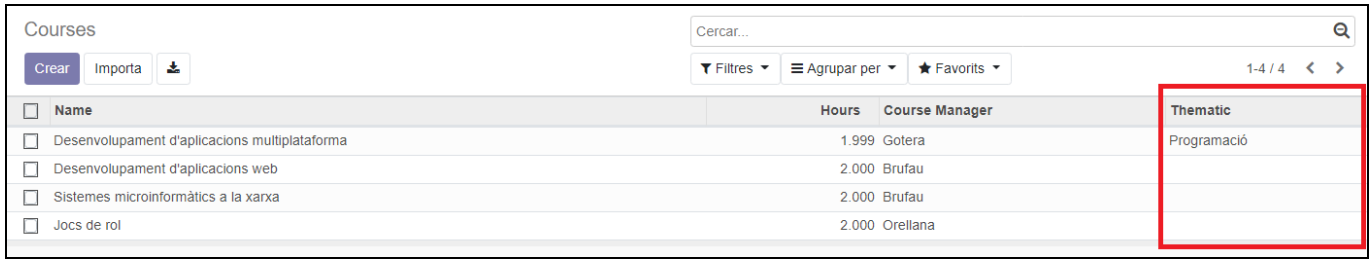
Courses

Name	Hours	Teacher Manager
DAM	2.000	Pepe

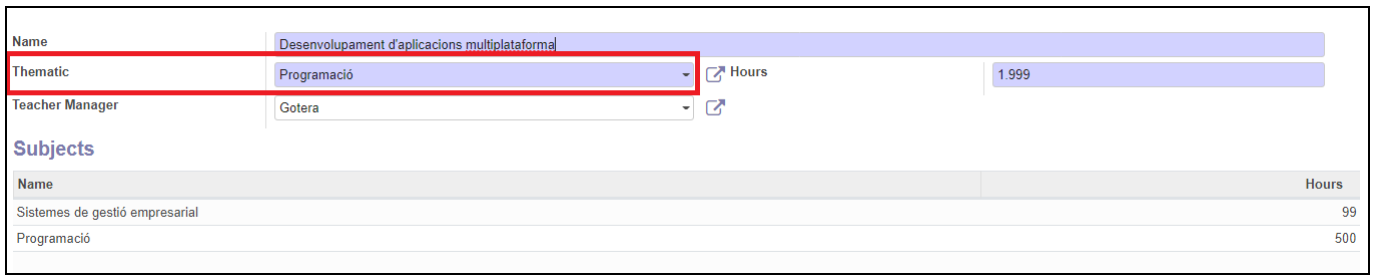
- Vista `form` de `Teacher`:



- Vista `tree` de `Course`:



- Vista `form` de `Course`:



Alerta! Retoqueu (respecte la versió 04) la vista form de `Teacher` de manera que permeti crear cursos. En conseqüència, si es permet afegir un curs, cal incorporar en el corresponent formulari, el camp `thematic_id`, doncs és obligatori.

Resultat del disseny **`05_school`: `05_school.zip`**

Observació a la instal·lació dels retocs:

- La incorporació de `country_id` dins `SchoolTeacher` amb `required=true`, implica que la taula `school_teacher` incorpori la columna `country_id` amb `not null`, però això no és possible si en actualitzar el mòdul, la taula `school_teacher` ja conté dades. Per aquest motiu, Odoo afegeix la columna `country_id` però no li posa la restricció `not null` i en el fitxer `odoo.log` podrem veure el missatge: `Table 'school_teacher': unable to set NOT NULL on column 'country_id'`
- La incorporació de `thematic_id` dins `SchoolCourse` amb `required=true`, implica que la taula `school_course` incorpori la columna `thematic_id` amb `not null`, però això no és possible si en actualitzar el mòdul, la taula `school_course` ja conté dades. Per aquest motiu, Odoo afegeix la columna `thematic_id` però no li posa la restricció `not null` i en el fitxer `odoo.log` podrem veure el missatge: `Table 'school_course': unable to set NOT NULL on column 'thematic_id'`

Si editem un curs o un professor, Odoo obliga a introduir valor pels camps amb `required=true` encara que a la corresponent taula no hagi pogut incorporar la restricció `not null`.

Odoo, en cada actualització, comprova la coherencia de les classes del model amb les taules de la BD i intentarà incorporar les restriccions `not null` que siguin necessàries. En el moment que totes les files de les taules afectades ja continguin valor per la columna que ha de ser `not null`, Odoo incorpora la restricció en la següent actualització.

Controlador: Camps calculats / restriccions / mètode `name_get`

Conceptes d'Odoo a conèixer:

- Recordset:
  https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#recordsets
  Conjunt ordenat de registres d'un mateix model (classe), sobre el que s'executa qualsevol mètode.
- Decorators:
  https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#module-odoo.api
  Els decoradors són una eina molt potent i útil a Python, ja que permet als programadors modificar el comportament d'una funció o classe. Els decoradors ens permeten embolicar una funció per tal d'ampliar el comportament de la funció embolicada, sense modificar-la permanentment.

Classes de Python a conèixer:

- Mòdul `datetime`, amb les classes `date`, `datetime`, ...
- Mòdul `dateutil`, amb les classes `relativedelta.relativedelta`, ...

- **Com depurar el codi?**

- Via el depurador de l'entorn de desenvolupament que s'utilitzi: En el nostre cas, si usem PyCharm, es l'opció més fàcil, incorporant punts de ruptura.
- Però Odoo, per si mateix, incorpora un mecanisme de depuració, consistent en utilitzar el paquet `logging` per introduir missatges que, en temps d'execució, apareixeran en el fitxer `odoo.log`.

  Per activar-ho, cal:

  - A la part superior del fitxer .py on es vulgui usar, incorporar:
    ```
    import logging
    _logger = logging.getLogger(__name__)
    ```
  - Dins els mètodes, on es vulgui incorporar un missatge, caldrà escriure:
    `_logger.xxxx('missatge')` o `_logger.xxxx(nom_variable)`

    on `xxxx` pot ser: `info`, `warning`, `debug`, `error` i `critical`.

  Si usem `info`, apareixeran dins el fitxer `odoo.log` catalogats com a `INFO` i serà difícil de distingir-los dels missatges d'informació que habitualment enregistra Odoo. El mateix passa amb les altres categories de missatges.

  El lògic és usar `debug` i en tal cas cal retocar el fitxer de configuració d'Odoo, canviant l'entrada `log_level = info` per `log_level = debug` i, d'aquesta manera els nostres missatges `debug` quedaran enregistrats dins el fitxer `odoo.log` catalogats com a `DEBUG`. A més, en fer-ho amb `debug`, si queden dins el codi, no es veuran en una empresa on tinguin el nostre mòdul en explotació, doncs no tindran, suposadament, `log_level = debug`.

  Informació oficial: https://www.odoo.com/documentation/16.0/developer/reference/cli.html#logging

  Es pot comprovar funcionament en els primers mètodes que desenvolupem, tot i que és més fàcil usar les eines de depuració de PyCharm.

- **Camps calculats**

Info: https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#computed-fields

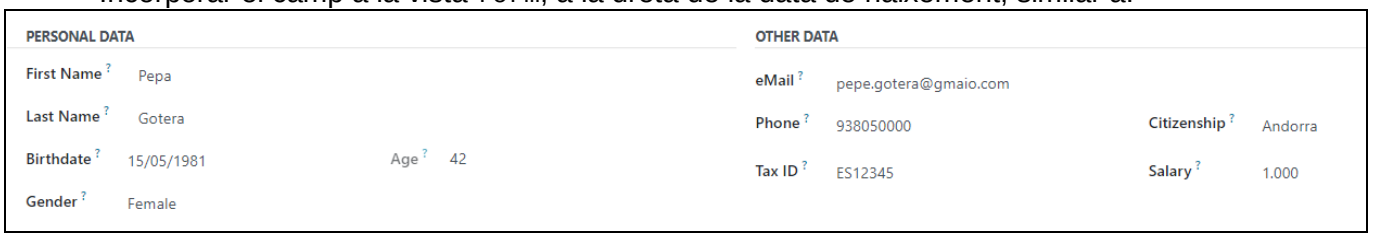
- Cal indicar-ho amb paràmetre `compute` en la definició del camp, que cal emplenar amb el nom del mètode que ha d'utilitzar per calcular el valor.
- Cal definir el mètode amb decorador `@api.depends` per indicar els camps X dels que depèn el càlcul de manera que quan l'usuari canviï el valor d'algun dels camps X, Odoo refresqui immediatament en pantalla, el valor del camp calculat, sense haver d'esperar a l'enregistrament.
- Els camps calculats no apareixen, per defecte, dins la taula de la BD, però hi ha casos en els que interessa que quedin enregistrats, fet que s'assoleix amb el paràmetre `strore=True` en la definició del camp.

Pràctica: Incorporar camp `full_name` (contingut `cognoms, nom`) a la classe `Teacher`. (`06_school`)

- Substituir `first_name` i `last_name` a la vista `teacher_tree` pel nou camp.
- Comprovar –via depuració– que en visualitzar la vista `tree`, s'executa el mètode i comprovar contingut de `self` i, posteriorment, dins el `for`, el contingut de `teacher` a cada iteració.
- Activar la vista form de `teacher` (on no hi ha `full_name`), reiniciar servidor i actualitzar pàgina. Com que `full_name` no apareix a la vista `form`, no s'executa el mètode. En navegar a la vista `tree`, podem comprovar la seva execució.
- Aprofitar el nou camp `full_name` per a que es mostri quan un camp relacional ha d'accedir a un professor (per exemple, camp `teacher_id` dins `school.course` => usar-lo en `_rec_name`).

Exercici (`06_school`)**:**

- Incorporar camp edat pels professors, calculant-lo a partir de la data de naixement.
- Incorporar el camp a la vista `form`, a la dreta de la data de naixement, similar a:



Està clar que `age` no tindria cap sentit tenir-lo enregistrat a la taula, doncs és un valor calculat canviant.

Quan un camp calculat es declara amb enregistrament a la BD, en actualitzar el mòdul, Odoo crea la columna i si ja hi havia registres, l'emplena aplicant el mètode de càlcul.

En cas que es decideixi eliminar l'enregistrament a la BD d'un camp calculat amb `store=True` (fet que s'assoleix eliminant aquest paràmetre o posant-lo a `False`), en actualitzar el mòdul, alguna versió d'Odoo no elimina la columna i no la continuarà emplenant ni actualitzant. Millor comprovar si en el procés d'actualització del mòdul s'ha eliminat i, si conve, eliminar-la via `alter table...`

En parlar de filtres (`domains`) es veurà la conveniència d'utilitzar `store=True` en camps calculats per poder utilitzar-los en els filtres.

- **Restriccions d'Odoo**

- Cal crear un mètode amb decorador `@api.constrains` indicant els camps afectats per la restricció.

Pràctica: Incorporem validació sobre el camp `salary` de `Teacher`. (`06_school`)

- Aquestes restriccions les comprova Odoo en enregistrar els canvis i només s'executen quan algun(s) dels camps indicats en `@api.constrains` ha canviat el valor.

- NO són restriccions `check` a la taula de la BD. Per tant, un informàtica desaprensiu podria accedir a la BD i incorporar valors que no verifiquessin la restricció, però això és suposa que no es fa. NO es traspassen a la BD per què és el servidor Odoo qui s'encarrega de la comprovació i no envia la instrucció SQL a la BD si no es tracta de valors vàlides.
- Es pot tenir `check` a la BD (més endavant ho veurem) però és més eficient que comprovi Odoo.

Exercicis (`06_school`):

- Validar que el camp `hours` de `Course` i `Subject` sigui estrictament positiu.
- Validar que el telèfon d'un professor només pugui contenir dígits.
- Validar el format del correu electrònic d'un professor.

  La solució incorpora a l'arrel un fitxer `utils.py` amb una utilitat (trobada a la web) que valida el format d'un correu electrònic. Aquest fitxer `utils.py` no conté cap classe per Odoo i, en conseqüència, no cal posar-lo dins `__init__.py`.

  Per altra banda, dins `school.py` es veu, a l'inici, com declarar la seva importació, en funció de la seva ubicació, per poder usar la funció `is_valid_email`.

  Si s'ubica tota la utilitat dins el mateix `school.py`, no cal importar-lo.

- **Sobreescriptura del mètode `name_get` d'una classe**

Tota classe d'Odoo disposa del mètode `name_get` per defecte (no sobreescrit) que, aplicat sobre un recordset d'objectes, retorna una llista de parelles `(id, representació_textual_de_registre)`, on representació textual és:

1r. Contingut del camp informat a la clàusula `_rec_name`.
2n. Si no hi ha clàusula `_rec_name`, contingut del camp `name` del registre
3r. Si no existeix camp `name`, el valor *nom_classe, idRegistre*.

I quan Odoo ha de visualitzar un camp `Many2One`, mostra el valor que retorna aquest mètode.

Per això, si a la classe `SchoolTeacher` que no disposa de camp `name`, tampoc informem `_rec_name`, quan cal visualitzar un `Teacher` des de `Course`, mostra *school_teacher, n* on n és l'identificador del `Teacher`. Quan hem incorporat el camp calculat `full_name`, l'hem aprofitat per usar-lo en `_rec_name`, però no sempre tindrem un camp (calculat o no) invocat pel mètode `name_get` per defecte.

Aquest funcionament del mètode `name_get` el podem canviar sobreescrivint el mètode. Simplement haurem de sobreescriure'l retornant una llista de parelles `(id,representació_textual)` on la representació textual sigui la que interessi. És a dir:

```
def name_get(self):
  result = []  # Llista buida que anirem emplenant per cada teacher
  for record in self:
    tupla = (record.id, càlculQueInteressi)
    result.append( tupla )
  return result
```

Incorporem aquest mètode a la classe `Teacher`. (`06_school`)

Quan una classe té el mètode `name_get`, és innecessari tenir definida la clàusula `_rec_name`.

Ara, en la classe `Teacher`, tenim el camp calculat `full_name` i el mètode `name_get` que fan el mateix... Podem eliminar el camp `full_name`? NO si volem mostrar en alguna vista el nom sencer d'un professor (com ara tenim a la vista `tree`). Les vistes només poden mostrar camps (`fields`) definits a la classe. El mètode `name_get` només s'utilitza per mostrar el valor en camps relacionals `Many2one`.

Disseny de relació `*:*` amb atributs.

En Odoo, la implementació d'una relació `*:*` entre dues classes A-B, amb atributs (amb classe associativa A-B), s'ha d'efectuar amb relacions `One2many` d'A a A-B i de B a A-B.

Com a exemple, considerem que en el nostre disseny del mòdul `school`, on una assignatura es pot impartir en molts cursos i un curs pot tenir moltes assignatures, ens interessa enumerar les assignatures dins cada curs. Apareix un atribut a la relació `*:*`. Veure disseny `07_school_UML`.

Pràctica: Evolucionar el mòdul `school` segons disseny `07_school_UML`.

- Aparició de classe associativa `CursAssignatura`.
- Aparició de nova classe `ConvocatòriaCurs` per gestionar les diverses convocatòries d'un curs, amb `dataInici` obligatòria i `dataFinal` optativa, però si existeix, ha de ser posterior a la data inicial o, com a molt, iguals (per un curs que comença i finalitza el mateix dia).
  Observar que la relació entre `Curs` i `ConvocatòriaCurs` és una composició (diamant ple), fet que indica que una `ConvocatòriaCurs` no pot existir sense el corresponent `Curs`. En conseqüència, la relació `Many2One` té sentit que inclogui *eliminació en cascada*.
- Aparició de camp `Fotografia` a la classe `Professor`.
- Aparició de camp `sinopsi` a la classe `Curs`, pensat per a incorporar explicació (grandària il·limitada) sobre el curs.
- Un curs passa a tenir obligatòriament un professor responsable (fins ara era opcional).

Implementació en Odoo seguint disseny `07_school_UML_Odoo`.

- **Fase 1:** *Implementació de la classe associativa `SchoolCourseSubject`*

- L'aparició de la classe `SchoolCourseSubject` enlloc de la `Many2many` entre `SchoolCourse` i `SchoolSubject`, provoca la creació de la taula `school_course_subject` i, la taula `school_course_subject_rel` que havíem definit en la relació `Many2many` deixa de tenir sentit i cal eliminar-la manualment (`drop table`) de la BD.
- Els camps `Many2one` i `Many2many` poden incorporar el paràmetre `ondelete` amb valors `'cascade'`, `'set null'` i `'restrict'`, que indiquen l'actuació que ha de tenir Odoo quan s'intenti eliminar el registre apuntat per la relació `Many2one` o `Many2many`.

  En una `Many2one`, si el camp no és obligatori, el valor per defecte és `'set null'`, però si és obligatori, `'restrict'`. En una `Many2many`, el valor per defecte és `'cascade'`.

  En el nostre cas, té molt sentit que l'eliminació d'un curs (`SchoolCourse`) impliqui l'eliminació automàtica de les assignatures assignades (`SchoolCourseSubject`) i de les convocatòries de curs (`SchoolCourseEdition`). Per això, aquestes classes cal que tinguin `ondelete='cascade'` en el camp `course_id`.
- El camp `number` ha de ser estrictament positiu.
- ALERTA! És clar que hauríem de controlar que en un curs no hi hagi dues assignatures amb el mateix `number` ni assignatures repetides. Encara no estem en situació de controla-ho. Pendent!
- Des del formulari de `SchoolCourse` s'ha de poder continuar amb la gestió (altes-baixes-consultes-modificacions) de les seves assignatures, però ara, des de `SchoolCourse` no s'accedeix a `SchoolSubject`, sinó a `SchoolCourseSubject` i, per tant, els camps a mostrar no

són els de `SchoolSubject`. I cal mostrar el número que ocupa dins el curs i el nom de l'assignatura. I interessaria veure les assignatures ordenades per número, no? Pendent!

- Des del formulari de `SchoolSubject` s'ha de poder continuar amb la consulta dels seus cursos i els professors habilitats, però ara, des de `SchoolSubject` no s'accedeix a `SchoolCourse` sinó a `SchoolCourseSubject` i, per tant, els camps a mostrar no són els de `SchoolCourse`. I cal mostrar el nom del curs i el número que l'assignatura ocupa dins el curs.

- **Fase 2**: *Implementació de la classe `SchoolCourseEdition` i camp summary dins `SchoolCourse`*

- Respecte el camp `summary` de la classe `SchoolCourse`:
  - Cal declarar el camp com `fields.Text` o `fields.Html`
  - En un form, si és `Text`, s'edita amb editor de text pla o, amb editor HTML si s'assigna `widget="html"`.
  - En un form, el camp "creix" en funció del seu contingut. En cas de visualització `html`, es pot usar l'atribut `options="{'resizable':true}"` per definir la grandària i disposar de barra de desplaçament vertical.

- Des del formulari de `SchoolCourse` s'ha d'incorporar la gestió (altes-baixes-consultes-modificacions) de les edicions.

- La data final d'una edició considerarem que no és obligatòria però, en cas de tenir valor, ha de ser igual o posterior a la data d'inici, que sí és obligatòria.

- La vista formulari de `SchoolCourse` ha de quedar similar a:

Observar que l'etiqueta del camp `summary` no apareix a l'esquerra,sino al damunt cosa que s'assoleix usant l'element `<label>`, que permet mostrar l'etiqueta d'un camp on interessi i es pot sobreescriure.

Sintaxi: `<label for="nomCamp" [string="novaEtiqueta"]>`

L'element `<separator>` és un títol que no té res a veure amb la label d'un camp i utilitza una font major.

- **Fase 3**: *Implementació d'obligatorietat de `manager_id` i camp `photo` en `SchoolTeacher`*

- El fet que el camp `manager_id` de la classe `Course` passi a ser obligatori implica que la graella `course_ids` de la vista form de `Teacher`, que és editable, presenta els problemes:

  ➢ Si un usuari demana *Afegir una línia* i, en lloc de seleccionar un curs existent, prem el botó *Crear* per a crear un nou curs, el formulari que s'obre no permet –evidentment- la introducció

del professor i... en ser un camp obligatori, presentaria problemes en enregistrar la transacció. Atenció! El problema té lloc quan Odoo ja ha pogut indicar `NOT NULL` a la taula.

Solució 1: Desactivar el botó *Crear* posant `create="0"` en el tree que defineix la graella.

Solució 2: Incorporar l'atribut `context="{'default_manager_id':id}"` en el camp `course_ids`, que indica amb quin valor s'ha d'emplenar, per defecte, el camp `manager_id`, que no és altre que el camp `id` del registre `Teacher` en curs. Optem per aquesta solució.

- Si un usuari decideix prémer la creueta *x* per desassignar un curs al professor actiu, Odoo llença un error, doncs tot curs ha de tenir un professor responsable. Caldria poder desactivar la creueta *x*, però sembla que no és possible en Odoo13+. El problema és que Odoo llança un error difícil de comprendre per un usuari. Si el poguéssim interceptar i canviar... Pendent!

- Respecte el camp `photo` a la classe `SchoolTeacher`.
  - Cal declarar el camp com `fields.Binary`
  - Cal mostrar-lo en un `form` amb `widget="image"`

Alerta: Fins Odoo12 la imatge quedava dins columna de tipus `bytea` a la taula corresponent. En Odoo13+ es guarda com un adjunt (repassar UF1) i si es vol continuar com a columna, cal afegir a la declaració del camp l'atribut: `attachment=False`.

Recordatori de UF1: Els adjunts es guarden a la taula `ir_attachment`, que entre altres té les columnes `res_model`, `res_field` i `res_id` on hi ha la informació de la classe-camp-registre a la que correspon l'adjunt. Així, si es vol accedir a les imatges dels professors, caldrà escriure:

```
select mimetype, store_fnane from ir_attachment
where res_model='school.teacher' and res_field='photo';
```

i la columna `mimetype` ens dirà el tipus d'imatge i la columna `store_fname` ens dirà la ubicació.

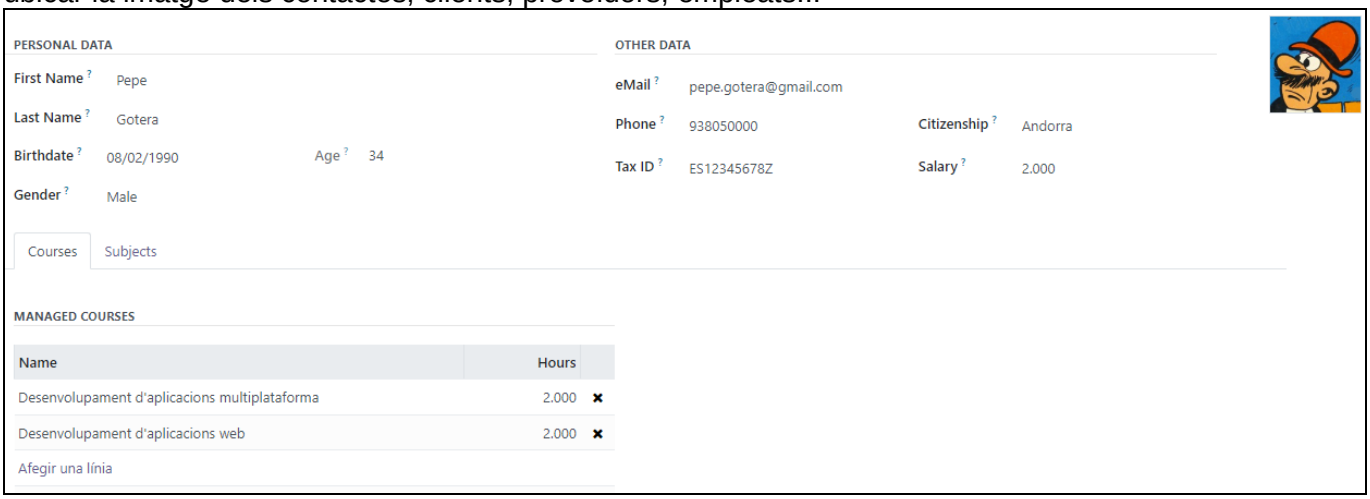
Recordem, de la UF1, la ubicació física dels adjunts en Odoo16:

- En Windows, a `rutaOnHiHaOdoo\sessions\filestore\<nomBD>`
- En Linux, a `/var/lib/odoo/.local/share/Odoo/filestore\<nomBD>`

Odoo proporciona camps `Image` específics pel tractament d'imatges (tema a explorar).

- La vista formulari de `SchoolTeacher` ha de quedar similar a la imatge següent.

Per aconseguir que el camp `photo` es mostri a la part superior dreta de la vista `form`, poseu-lo dins l'element `sheet` i abans de l'element `group` principal, que és el criteri que Odoo16 usa per ubicar la imatge dels contactes, clients, proveïdors, empleats...



Resultat del disseny **`07_school`: `07_school.zip`**

Més sobre vistes

- **Vistes `tree` editables**

Les vistes `tree` es poden fer editables (atribut `editable` amb valor `top` o `bottom`) sent llavors innecessària la vista `form`, funcionalitat útil quan la classe té uns pocs camps que es visualitzen tots a la graella. Ho podem aplicar (versió 8 de mòdul `school`) a les graelles `course_subject_ids` i `edition_ids` del formulari de `SchoolCourse`, ja que la graella mostra tots els camps que hi hauria a la vista `form`. En aquest cas, no és necessari dissenyar la vista `form` vinculada a la vista `tree`.

- **Ordre de visualització de registres**

Odoo permet definir un criteri d'ordre en la visualització dels registres d'una classe. Per assolir-ho cal emplenar la clàusula `_order = 'camp1 [desc], camp2 [desc], camp3 [desc],...'` dins la classe.

Si no existeix la clàusula `_order`, Odoo mostra els registres ordenats pel seu `id`.

La partícula `desc` és optativa i s'usa per indicar, com en SQL, ordenació en descendent.

Només es pot usar camps que existeixin a la taula. Per tant, no es pot usar un camp calculat no emmagatzemat a la taula.

En els camps de text, l'ordre és lexicogràfic i, per tant, les minúscules apareixen després de les majúscules i els caràcters especials (accents, ç, ñ,...) després de les minúscules. Hi ha solució? Malgrat PostgreSQL permet usar les funcions `lower-upper` en la clàusula `order by` (no solucionaria el cas de caràcters especials), Odoo 16 no permet usar-les en la clàusula `_order`.

Exercici: Aconseguir que cursos, assignatures, temàtiques i edicions d'un curs es mostrin ordenades per nom, professors ordenats per cognom-nom i assignatures d'un curs ordenades pel seu número.

Camps `related`

En ocasions interessa incorporar en una vista d'una classe A que conté algun camp `xxx_id` de tipus `Many2one` cap una altra classe B, camps del registre de la classe B apuntat per `xxx_id`.

Per exemple, ens interessa incorporar a les vistes `tree` i `form` de la classe `Course`, el telèfon i el correu electrònic del professor responsable del curs, en cas que en tingui.

Per aconseguir això es disposa dels camps `related`, que permeten incorporar a la classe A, una còpia de camps de la classe B, per via del camp `xxx_id`.

Un camp `yyyA` a la classe A, còpia de camp `yyyB` de la classe B es defineix com el camp `yyyB` original afegint el paràmetre `related = 'xxx_id.yyyB'`. I si `yyyB` fos també un `Many2one`, ens permetria navegar fins un camp de la classe C apuntat per ell, escrivint `'xxx_id.yyy_id.zzzC'`. És a dir, els camps `Many2one` ens permeten anar navegant de classe en classe.

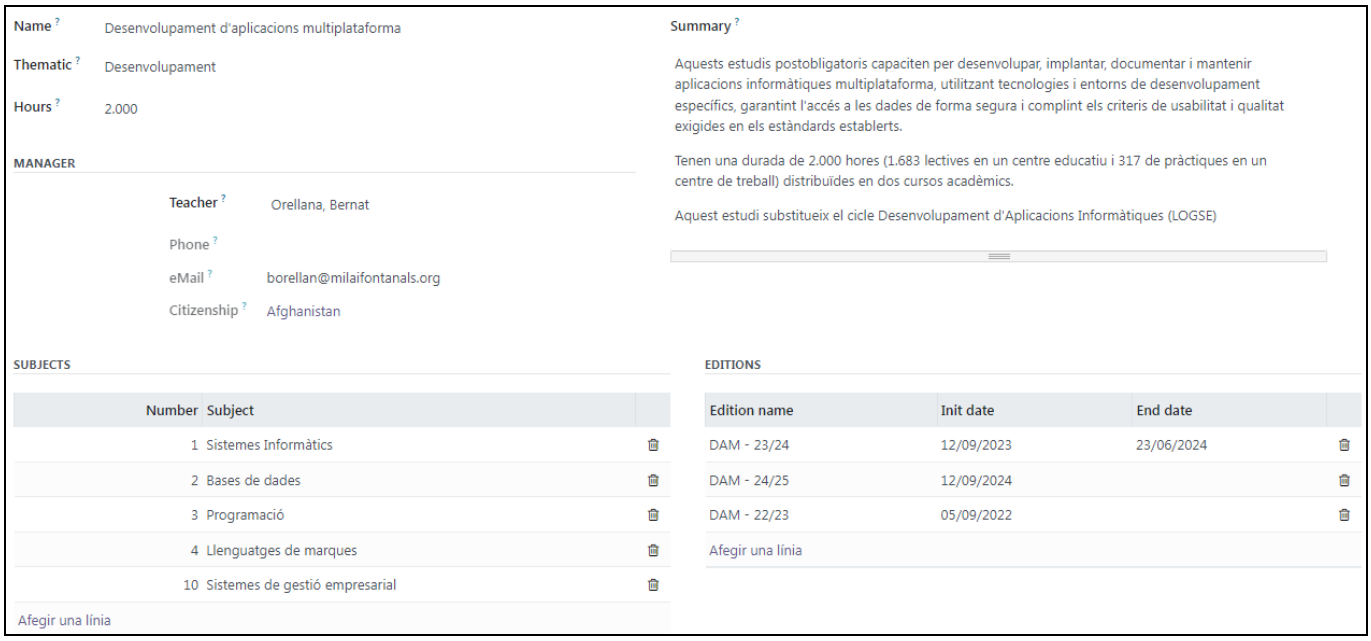
La versió 8 del mòdul `school` (`08_school.zip`) incorpora els camps `manager_email`, `manager_phone` i `manager_citizenship` a la classe `SchoolCourse` que donen accés al correu electrònic, telèfon i nacionalitat del professor i una vegada incorporats ja es poden usar a les vistes de `SchoolCourse`.

Els camps `related` són de només lectura.

Observar que per incorporar la nacionalitat del `teacher` responsable d'un curs, es pot:

- Declarar camp `related` de tipus `Many2one` amb navegació `manager_id.country_id`
  En aquest cas, el primer paràmetre ha de ser la classe a la que apunta el camp (`res.country`)
  La diferència respecte la possibilitat següent, és que en tractar-se d'un camp `Many2one`, quan aparegui en una vista, des d'ell es podrà navegar fins la vista form de la seva classe.
- Declarar camp `related` de tipus `Char` amb navegació `manager_id.country_id.name`

Exercici: Aconseguir que la vista `form` de cursos mostri la informació del `manager` similar a:



Controlador: Mètode onchange

Info: https://www.odoo.com/documentation/16.0/developer/tutorials/backend.html#onchange

El mecanisme `onchange` permet actualitzar algun(s) camp(s) d'una vista en funció de canvis que es produeixin en altres camps (o en els mateixos). **NOMÉS** actua en vistes.

Els camps calculats porten integrat aquest mecanisme.

Exemple en el camp `tin` de la classe `SchoolTeacher` (versió 8) per a que sempre quedi en majúscula.

PERILL!

Aquest mètode NO assegura que totes les incorporacions a la corresponent classe entrin amb el valor que genera el mètode `onchange`. Aquest mètode NOMÉS actua en vistes i, per codi, es pot generar registres en qualsevol taula. Per assegurar que totes les incorporacions, via Odoo, a la taula corresponent, entrin en un determinat format, podrem sobreescriure el mètode `create` a la classe.

I... per codi també es pot generar modificacions als registres en qualsevol taula i per assegurar que totes les modificacions, via Odoo, a la taula corresponent, entrin en un determinat format, podrem sobreescriure el mètode `write` a la classe.

Unicitat d'un o varis camps

Odoo permet definir que un o varis camps no admetin repeticions. Per assolir-ho, cal introduir una restricció SQL d'unicitat, en camp `_sql_constraints` en la definició de la classe, seguint la sintaxis:

```
_sql_constraints=[('nomRestriccióUnicitat','restriccióSQL','missatgeSiIncompliment'),
                  ('nomRestriccióUnicitat','restriccióSQL','missatgeSiIncompliment'),
                  ...]
```

Com es veu, és una llista de tuples formades per tres elements:

- `nomRestriccióUnicitat`, que és el nom que tindrà la restricció a la taula de la BD
- `restriccióSQL`, que és el codi SQL que utilitzaríeu per indicar la unicitat en una instrucció SQL
- `missatgeSiIncompliment`, que és el missatge que Odoo mostrarà si es viola la unicitat.

La llista tindrà tantes tuples com restriccions d'unicitat calgui definir a la classe.

La versió 8 del mòdul `school` (`08_school.zip`) incorpora:

- Unicitat en el camp `tin` de la classe `SchoolTeacher`
- Unicitat en la classe `SchoolCourseSubject` per garantir que no es pugui repetir una assignatura en un curs i que no hi pugui haver dues assignatures amb el mateix número dins el curs.

La clàusula `_sql_constraints` també es pot usar per incorporar altres restriccions `check` a la BD. Per exemple (veure versió 8 del mòdul `school`) la verificació de que salari sigui positiu. No és habitual incorporar aquestes restriccions a la BD, ja que es posen com a restriccions en el codi d'Odoo i Odoo ja no enviarà mai un salari no positiu a la BD pel seu enregistrament.

Totes les restriccions `_sql_constraints` queden definides a la BD com a `constraints`. PostgreSQL, per una constraint `UNIQUE`, crea automàticament un `INDEX UNIQUE`, com molts SGBD i l'anomena: `nom_taula_nom_restricció`. Afegeix prefix `nom_taula` per distingir mateix nom de restricció en diverses taules, doncs no poden conviure en una BD índexs amb mateix nom.

Atenció!

- En introduir una restricció d'unicitat, cal estar segurs que la taula no conté dades contràries a la restricció, doncs l'actualització no tindrà lloc (missatge dins `odoo.log`).
- En cas de canviar una restricció d'unicitat, Odoo no elimina la restricció existent anteriorment a la taula i caldrà eliminar-la manualment via:

```
alter table <nom_taula> drop constraint <nom_constraint>
```

Exercici:

Aconseguir que no hi pugui haver temàtiques, cursos, assignatures i edicions d'un curs amb igual nom.

Bona solució? En aquests casos, fer-ho amb una `_sql_constraints` no és una bona solució (com tampoc en el cas del camp `tin` de `SchoolTeacher`, doncs `unique(camp)` és case insensitive.

- **Restricció d'unicitat vs majúscules/minúscules**

La restricció d'unicitat via `_sql_constraints` distingeix entre majúscules i minúscules i, en conseqüència, quan es vol aconseguir que un camp alfanumèric sigui únic, no es pot aconseguir via la restricció "unique" de PostgreSQL en `_sql_constraints`.

És a dir, si per una classe volem aconseguir que el típic camp `name` alfanumèric sigui únic, i definim:

```
_sql_constraints = [
    ('name_unique','unique(name)','The name must be unique!')
]
```

no aconseguirem prohibir que hi hagi registres amb "XXX", "xxx", "Xxx" en el camp `name`.

Estaria molt bé poder definir una restricció similar a:

```
_sql_constraints = [
    ('name_unique','unique(lower(name))','The name must be unique!')
]
```

però PostgreSQL (comprovat fins a versió15) no permet aquest tipus de `constraint`.

En canvi, PostgreSQL SÍ permet crear un índex `UNIQUE` indicant com a camp: `lower(camp)` o `upper(camp)` amb la instrucció de creació d'índex:

```
CREATE UNIQUE INDEX nomIndex ON nomTaula (LOWER(nomCamp)
```

És a dir, Odoo envia a PostgreSQL la `_sql_constraint` com una `check` i PostgreSQL no admet com a `check` una `unique(lower(...))` ni una `unique(upper(...))`, però PostgreSQL SÍ admet un índex amb `lower(...)` o amb `upper(...)`.

Tenim 3 possibilitats per aconseguir la unicitat *case insensitive* en camps alfanumèrics:

❖ **Possibilitat 1: via una restricció d'Odoo `@api.constrains`**

Dissenyar un mètode `check` que comprovi si ja hi ha, entre **TOTS** els recursos de la classe (no només els que proporciona `self`), algun amb mateix valor que el que estem verificant, excepte ell mateix.

Es pot fer, però és molt costosa, doncs ha d'accedira a TOTS els registres de la taula. Desestimada!

❖ **Possibilitat 2: via mètode `onchange` que transformi el valor del camp a un format estàndard**

Sigui quin sigui el text que introdueix l'usuari, en el formulari on s'introdueix el valor del camp pel que volem assegurar la unicitat, podem definir un mètode `onchange` que formati el contingut del camp deixant sempre el contingut del camp en un mateix format, independent de com l'hagi introduït l'usuari. Per això, ens ajudem de funcions de tractament de cadena de Python, com:

- `capitalize()`
- `lower()`
- ...
- `upper()`
- `title()`

Aplicada en camp `tin` de la classe `SchoolTeacher` (veure versió 7 de mòdul `school`) on és adequat que el tin sempre quedi enregistrat en majúscules i amb la combinació de `onchange` i `_sql_constraints` garantim la unicitat.

Però aquesta tècnica `onchange + _sql_constraints` no garanteix que una introducció/modificació de dades des d'Odoo sense passar pel formulari (que és on només actua `onchange`) o, fins i tot, una introducció/modificació de dades directament a la BD (cosa que no hauria de passar gairebé mai) introdueixin dades textuals que no verifiquin unicitat case insensitive.

❖ **Possibilitat 3: índex `UNIQUE` no sensible a majúscules-minúscules**

Més amunt s'ha dit que PostgreSQL NO permet restriccions `unique(lower(...))` o `unique(upper(...))`, però SÍ permet índexs `unique(lower(...))` i `unique(upper(...))`. La solució, doncs, està en no utilitzar la funcionalitat `_sql_constraints` i en el seu lloc obligar a Odoo que creï, si no existeix, un índex `UNIQUE` adequat i això Odoo ho sap fer via utilitats que té en el mòdul `tools`.

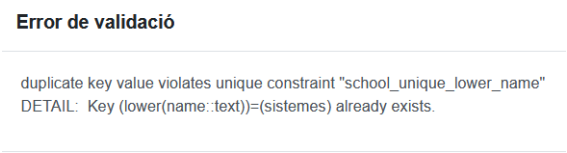
Hem aplicat aquesta possibilitat en el camp `name` de les temàtiques, cursos, assignatures i edicions (veure versió 7 del mòdul `school`), implementant el mètode `_auto_init` (que s'executa quan s'actualitza el mòdul) dins la classe:

```
def _auto_init(self):
  res = super(<nomClasse>, self)._auto_init()
  tools.create_unique_index(self._cr, '<nomIndex>',
                            self._table, ['<camp1>','<camp2>',...])
  return res
```

ALERTA!

És fonamental que `nomIndex` incorpori com a prefix `nom_taula_`, per distingir mateix nom de restricció en diverses taules, doncs no poden conviure en una BD índexs amb mateix nom

Amb aquesta opció, quan l'usuari intenta introduir un valor repetit, PostgreSQL no deixa efectuar l'operació i retorna el missatge al servidor Odoo que el mostra per pantalla:



El missatge no és prou clar, oi... Com que no s'ha programat amb una `_sql_constraint` on es pot introduir el missatge en cas d'incompliment, Odoo només pot mostrar el missatge que envia PostgreSQL. Més endavant veurem com interceptar aquest missatge sobreescrivint els mètodes `create` i `write` que són els que efectuen les altes i les modificacions a la BD.

Filtres en model i/o vistes – `domain` | `context`

En ocasions interessa incorporar filtres, de manera que els desplegables només mostrin un subconjunt de la totalitat de registres possibles o, fins i tot, canviar l'actuació per defecte del filtre `active`. Això s'aconsegueix definint domain.

Els `domain` es poden incorporar, si cal, en:

- Camps relacionals en la definició d'una classe => S'apliquen a qualsevol vista on s'utilitzi el camp, a no ser que es sobreescrigui el `domain` a la vista.
- Camps relacionals en la definició d'una vista => S'aplica només a la vista.
- En certs mètodes (es veurà més endavant) on calgui filtrar els registres

❖ Exemple d'utilització de `domain` en una classe per filtrar els registres:

Es demana que el professor responsable d'un curs sigui de nacionalitat espanyola. Tenim

```
manager_id = fields.Many2one('school.teacher','Manager',...)
```

La definició anterior permet que `manager_id` sigui qualsevol registre de `school.teacher`. Si volem assegurar que només puguin ser professors amb nacionalitat espanyola, cal incorporar un filtre (veure versió 9 de mòdul `school`):

```
manager_id = fields.Many2one('school.teacher','Manager',...,
                             domain=[('country_id.code','=','ES')])
```

Cada condició és una tupla amb 3 camps i un domain pot combinar vàries condicions amb els operadors lògics & (conjunció), | (disjunció) i ! (negació) aplicats amb notació polonesa (prefixa, no confondre amb notació polonesa inversa).

❖ Exemples d'utilització de `domain` per desactivar `active` en les recerques (desplegables):

No és massa habitual voler desactivar `active` en les recerques (desplegables) de camps relacionals, però en cas que es vulgui fer, caldrà escriure un `domain` adequat.

Suposem que en els desplegables del camp `manager_id` de la classe `SchoolCourse`, es vol que sempre es mostri tots els professors de nacionalitat espanyola, estiguin actius o arxivats. Amb el `domain` anterior, només apareixerien els actius. Possibilitats:

```
domain=[('country_id.code','=','ES'),'|',('active','=',True),('active','=',False)]
```

o, equivalent (donat que Odoo aplica la conjunció quan no hi ha operador):

```
domain=['&',('country_id.code','=','ES'),'|',('active','=',True),('active','=',False)]
```

La versió 9 del mòdul `school` incorpora aquest `domain` en el camp `manager_id` de la vista `form` de `SchoolCourse` i, per tant, en aquesta vista, sobreescriu el `domain` incorporat en la definició del camp `manager_id` de la classe `SchoolCourse`. Només tindrà efecte en aquesta vista.

El `domain` en una vista, cal escriure'l entre dobles cometes: `domain="[...]"`.

La incorporació dels camps `active` en els `domain` afecta únicament a les cerques (desplegables) però no pas a la visualització en les graelles (vistes `tree` i graelles `one2many` i `many2many`). Per desactivar `active` en la visualització... veure apartat següent.

- **Com desactivar `active` en una graella? – Com distingir registres via format visual?**

❖ Per obligar a que Odoo mostri tots els registres (actius i arxivats) en una vista `tree`, cal incorporar a l'`action` que invoca la vista, l'element `context` amb el següent contingut:

```
<field name="context">{'active_test': False}</field>
```

Veure `action` de classe `SchoolTeacher` en versió 9 de mòdul `school`.

Aquest context en `action` afecta els `active` a nivell de totes les vistes invocades per l'`action`. Així, en l'exemple de la classe `SchoolTeacher`, provoca que la vista `tree` mostri tots els professors (actius i arxivats) i quan seleccionem un professor, mostri tots els cursos assignats (actius i arxivats).

❖ Per obligar a que Odoo mostri tots els registres (actius i arxivats) en graelles (`One2many` o `Many2many`) de vistes `form`, hi ha dues possibilitats:
- Si es vol que afecti a totes les vistes on és el camp, cal incorporar `context={'active_test': False}` en la definició del camp en el model.
- Si es vol que afecti a una vista en concret, cal incorporar `context="{'active_test': False}"` en el camp dins la vista. (En Odoo 13-14-16 no funciona)

❖ Quan `active` està desactivat i es veuen tots els registres, potser interessa incorporar alguna informació per a que l'usuari pugui distingir els registres actius dels arxivats. Evidentment, es pot incorporar el camp `active` a la vista (apareix com a casella de selecció), però Odoo també facilita la distinció via el format de visualització, utilitzant l'atribut `decoration-{$name}` a la vista `tree`.

+Info: https://www.odoo.com/documentation/16.0/developer/reference/backend/views.html#attributes
+Info: https://www.iwesabe.com/blog/how-to-add-colors-to-tree-view-in-odoo

Atenció! Els camps utilitzats en les condicions per activar un `decoration`, han d'estar presents a la corresponent vista `tree`. Si no es vol que siguin visibles, disposem de l'atribut invisible.

Veure vista `tree` de classe `SchoolTeacher` en versió 9 de mòdul `school`, que mostra tots els professors (via `active_test` a l'`action`) i distingeix via color els activats (verd) dels arxivats (vermell). Ha calgut incorporar el camp `active` a la vista com a invisible per a que no aparegui.

Veure a la vista `form` de la mateixa classe, en la graella que mostra els cursos assignats, hauria de de distingeix via tipus de lletra, els activats (verd) dels arxivats (vermell), però en Odoo16 no funciona. Ha calgut incorporar el camp `active` dins la `tree` corresponent a la graella, com a invisible per a que no aparegui la columna.

Detectat funcionament anòmal en Odoo 13+:

Si una vegada desactivada la funcionalitat `active` en una `action`, el programador decideix eliminar la desactivació eliminant la línia on defineix `active_test` a `False`, en actualitzar el mòdul continua activa en l'empresa on s'havia activat. Evidentment, en instal·lar el mòdul en una nova empresa, no estarà activa.

Solucions:
- Desinstal·lar mòdul i tornar-lo a instal·lar (perdrem totes les dades de totes les taules!!!)
- Enlloc d'eliminar la línia on defineix `active_test` a `False`, mantenir-la posant `active_test` a `True`, que és el funcionament per defecte. En actualitzar el mòdul tornarà a estar activada i, si llavors es vol eliminar la línia, es podrà fer sense problema.

- **Camps calculats en filtres**

Per poder utilitzar un camp calculat en un `domain`, cal verificar una de les dues condicions:

- El camp estigui emmagatzemat a la BD (`store=True`)
- El camp incorpori el paràmetre `search=funció_search` (potser veurem algun exemple més endavant – Odoo avançat)

Vistes `calendar`

Info: https://www.odoo.com/documentation/16.0/developer/reference/backend/views.html#calendar

Important: Si hi ha definit `date_stop`, s'ignora possible definició de `date_delay`.

Exemple: Incorporar, a nivell d'edicions d'un curs, vista `calendar` per visualitzar el curs.

Solució: Vista `school_course_edition_calendar` en versió 9 de mòdul `school`.

Si en el codi no s'indica `date_stop` ni `date_delay`, en el calendari mostra només el dia d'inici. OK!

Errors/funcionaments estranys en Odoo16:
- Si en el codi s'indica `date_stop`, no mostra els objectes que no tenen valor en aquest camp. KO!
- Si a la classe s'introdueix un camp calculat que, pels objectes que tenen dates inici i final en calculi els dies transcorreguts i que pels que no tenen data final els calculi 1 i usem aquest camp en `date_delay` (eliminant `date_stop`)... KO! No fa cas del calcul establert i mostra intervals que res tenen a veure amb el què hauria de mostrar.

La vista `calendar` permet, per defecte, eliminar i modificar els esdeveniments, fet que es pot desactivar. En cas de demanar modificar l'esdeveniment, mostra la vista form de la vista i si no n'hi ha cap, en crea una per defecte. En el nostre cas, incorporem una vista form adequada.

Vistes `graph`

Info: https://www.odoo.com/documentation/16.0/developer/reference/backend/views.html#graph

En ocasions, els models existents incorporen camps numèrics sobre els que té sentit desenvolupar vistes `graph` (barres, formatge i línia). Només es pot utilitzar camps persistents (que resideixin a la BD) i no pas camps calculats que no tinguin `store=True`.

Exemple: Incorporar, a nivell de professors, una gràfica de barres pel salari.

Solució: Vista `school_teacher_graph` en versió 9 de mòdul `school`.

Observacions:

- El camp `salary` és utilitzat com a mesura
- El camp `tin` és utilitzat com a etiqueta en l'eix de les X. No es pot utilitzar el camp `full_name` per què no resideix a la BD. A més, si utilitzéssim `full_name`, aquesta gràfica agruparia els salaris del professorat amb mateix `full_name`, i això no interessa, doncs hi pot haver divers professorat amb mateix nom-cognom. En canvi, com que el mòdul exigeix que el camp `tin` sigui únic, la seva utilització ens assegura veure cada `teacher` per separat.

- La utilització del camp `id` com a camp de l'eix de les X no provoca error però en Odoo 15+ provoca l'agrupació de tots els registres. ¿?¿?¿?

Per lluir una bona identificació, caldria disposar d'algun camp calculat que identifiqués bé a cada registre (per exemple, concatenació de `tin` amb `full_name` o de `id` amb `full_name`) i tenir-lo emmagatzemat a la BD.

Reporting amb Jasper (veure PDF específic)

Explicació detallada en PDF específic.

La versió 10 del mòdul `school` incorpora informes:

- Llistat de cursos
- Fitxa de professor amb els cursos dels que és responsable

La carpeta `report` inclou els informes així com un `llegiume` explicatiu.

Exercici de reporting en Jasper

Desenvolupar un infome Jasper mestre-detall de cursos amb les seves edicions. Requeriments:

- Per cada curs, es vol visualitzar el nom, les hores, si està actiu o no (valors `Yes|No`), el resum, la temàtica i nom complet, correu-e i telèfon del responsable del curs.
- Per cada edició, es vol visualitzar el nom, la data d'inici i la data final.
- En finalitzar les edicions, es vol mostrar el número total d'edicions del curs.
- Ha d'estar incorporat en el mòdul

Exercici per després de Setmana Santa:

Evolucionar el mòdul `school` per aconseguir:

1) No hi pugui haver dues edicions d'un mateix curs amb mateixa data d'inici.
2) En una llista d'edicions de cursos, apareguin ordenades per data inici.
3) En el formulari de cursos, la graella de les assignatures que el conformen mostri, a més del número i del nom, les hores de l'assignatura.
4) Es desitja, pels professors, incorporar una vista `calendar` que mostri la data del seu aniversari dins l'any actual, per així poder-lo felicitar ;) així com l'edat que compleix en aquell dia.
   Pista:
   - Crear, dins la classe `SchoolTeacher`, camp que contingui la data de l'aniversari dins l'any actual i... que el seu contingut sigui automàtic (no podem esperar que ningú el mantingui).
   - Crear, dins la classe `SchoolTeacher`, camp que mostri l'edat que compleix en el dia.
   - Dissenyar la vista `calendar` a partir d'aquest camp data i mostrant, també, l'edat.

   Info a tenir en compte: Legalment, per les persones nascudes un 29 de febrer, l'aniversari és:
   - Dia 29 de febrer pels anys que són de traspàs
   - Dia 1 de març pels anys que no són de traspàs
5) Introduir la nova classe `Docència` per gestionar els professors que, en una edició de curs, imparteixen alguna de les assignatures, seguint el disseny de `11_school_UML.pdf` i `11_school_UML_Odoo.pdf`.
6) Incorporar una opció de menú per introduir la docència i la/les corresponent/s vista/es

Solució: Versió 11 del mòdul `school`

Comentaris a la solució:

- Observar a les diverses classes l'aparició de clàusules `_sql_constraints` i `_order` per aconseguir els requeriments 1-2 anteriors.
- Pel requeriment 3, ha estat necessari afegir a la classe `CourseSubject` el camp `subject_hours` (`related` via `subject_id`)
- Pel requeriment 4 i seguint les pistes facilitades:
  - S'ha creat els camps calculats `birthday` i `birthday_txt` a la classe `SchoolTeacher`.
  - S'ha creat la vista `calendar` a `SchoolTeacher`.
- Pels requeriments 5 i 6, s'ha creat la classe `SchoolTeaching` amb una vista `tree` (només `tree`) i la corresponent opció de menú:
  - Les vistes `tree` es poden fer editables sent llavors innecessària la vista `form`, funcionalitat útil quan la classe té uns pocs camps que es visualitzen tots en la graella.

  Exemple: `view_teaching_tree`: atribut `editable` a l'element `tree`, amb valors `top` o `bottom`.
  - El desplegable de la classe `CourseEdition` (camp `edition_id`) només mostra el nom de l'edició i, en conseqüència, no sabem de quin curs és cada edició. S'imposa implementar el mètode `name_get` a la classe `school.course.edition`.
  - El desplegable de la classe `CourseSubject` (camp `subject_id`) mostra informació com `'school.course.subject, id'` per què la classe `school.course.subject` no té mètode `name_get` ni clàusula `_rec_name` ni camp `name`.

    En la vista que ens ocupa, és fàcil d'aconseguir que el desplegable mostri el nom de l'assignatura, si a la classe `CourseSubject` es defineix `_rec_name='subject_id'`,però això no és adequat, doncs en una altra vista pot interessar que aparegui el nom del curs.

    Igual d'inadequada és l'opció d'afegir un camp `subject_name` (`related` via `subject_id`) i utilitzar-lo a `_rec_name`, doncs tampoc és adequat quan interessi que aparegui el nom del curs.

    La bona opció és facilitar un mètode `name_get` que mostri curs-numero-assignatura.
  - El funcionament de la vista `tree` no és adequat, doncs donada una edició d'un curs, només cal poder assignar assignatures del corresponent curs i, posteriorment, assignar professorat que pugui impartir l'assignatura... Solucionem'ho:
    - El selector d'assignatura només hauria de mostrar assignatures del curs seleccionat. És a dir, cal incorporar `domain` amb condició similar a:

      ```
      subject_id.course_id = edition_id.course_id
      ```

      Però en una condició `(op1,op,op2)` en un camp relacional `xxx`, `op1` ha de ser forçosament un camp existent a la classe apuntada per `xxx` i `op2` ha de ser un valor fix o un valor d'un camp existent a la vista. En el nostre cas, com que aquesta condició cal posar-la a `subject_id`:

      ```
      domain = "[('course_id','=',valor)]"
      ```

      i valor??? Hem d'aconseguir que el valor `edition_id.course_id` estigui present a la vista.

      Cal incorporar el camp `edition_course_id` a la classe `Teaching` per poder-lo incorporar a la vista amb `invisible="1"` i així poder-lo usar com a valor en el tercer paràmetre de la condició i queda:

      ```
      domain="[('course_id','=',edition_course_id)]"
      ```

- El selector de professor només hauria de mostrar professors habilitats per impartir l'assignatura seleccionada. És a dir, cal incorporar `domain` amb condició similar a:

  `teacher_id.id in subject_id.subject_id.teacher_ids`

  En el nostre cas, com que aquesta condició cal posar-la a `teacher_id`:

  `domain = "[('id','in',valor)]"`

  i valor??? Cal que `subject_id.subject_id.teacher_ids` sigui present a la vista.

  Cal incorporar el camp `subject_teacher_ids` a la classe `Teaching` per poder-lo incorporar a la vista amb `invisible="1"` i així poder-lo utilitzar com a valor en el tercer paràmetre de la condició i queda:

  `domain="[('id','in',subject_teacher_ids)]"`

➢ I encara ens manca aconseguir que l'usuari hagi de seleccionar en primer lloc del desplegable `CourseEdition`, seguidament del desplegable `Subject` i finalment del desplegable `Teacher`. Per aconseguir-ho necessitem el següent apartat.

Atributs `readonly`-`invisible`-`required` en vistes

Recordem que:

- Els elements `field` de les vistes poden incorporar els atributs `readonly`, `invisible` i `required` amb valor `"0"` per desactivar-los i amb valor `"1"` per activar-los.
- Respecte els camps amb `readonly` activat, Odoo no els té en compte a l'hora d'enregistrar el registre a la BD, doncs "pensa" que en ser de només lectura, l'usuari no l'haurà modificat.
- Respecte els camps amb `invisible` activat, quin sentit té tenir un camp així en una vista si no és visible? Doncs per poder-lo utilitzar en condicions dins la pròpia vista.

Odoo permet que els atributs `readonly`, `invisible` i `required` siguin dinàmics, és a dir, prenguin valors segons condicions. En cas que algun d'ells hagi de ser dinàmic, cal incorporar un atribut `attrs` amb sintaxis:

`attrs = "{ 'readonly': condició, 'invisible': condició, 'required': condició}"`

Observem que es tracta d'un diccionari de parelles, on només incorporarem la parella o parelles que calgui. La condició segueix la mateixa sintaxis que en un `domain`: llista de tuples...

Exemple de `attrs` en vistes: retocs finals en la vista `view_teaching_tree` en la versió 11 del mòdul.

El funcionament lògic hauria d'obligar al següent ordre d'accions per part de l'usuari:

1) L'usuari selecciona una edició de curs (`edition_id`)
2) L'usuari selecciona una assignatura del curs corresponent al curs de l'edició
3) L'usuari selecciona un professor habilitat per impartir l'assignatura seleccioada

Per aconseguir el funcionament desitjat:

- El selector d'assignatura hauria d'estar "inactiu" en cas que no hi hagués curs seleccionat.

  `attrs="{'readonly':[('edition_id','=',False)]}"`

- El selector d'edició hauria d'estar "inactiu" en cas que ja hi hagi assignatura seleccionada.

  `attrs="{'readonly':[('subject_id','!=',False)]}"`

- El selector de professor hauria d'estar "inactiu" en cas que no hi hagi assignatura seleccionada.

  `attrs="{'readonly':[('subject_id','=',False)]}"`

- El selector d'assignatura hauria d'estar "inactiu" en cas que ja hi hagi professor seleccionat. Això implica retocar la condició `readonly` anterior:

  ```
  attrs="{'readonly':['|',('edition_id','=',False),('teacher_id','!=',False)]}"
  ```

- Per últim... la funcionalitat implementada provoca que en el moment d'enregistrar una alta o una modificació, els camps `edition_id` i `subject_id` estiguin en mode de només lectura i, en conseqüència, Odoo no els "envia" per enregistrar i provoca un error. La "trampa" per solucionar aquest funcionament és duplicar aquests camps dins la vista amb `readonly` desactivat i amb `invisible` activat.

  I, a més, com que `subject_id` té un `domain`, cal tornar-lo a posar en la segona aparició de `subject_id` (comprovat necessari en Odoo13+; versions anteriors???)

Controlador: Mètodes `search`/`read`

Info: https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#search-read

Odoo facilita mètodes per obtenir registres que compleixin una determinada condició (`domain`). Entre ells, ens cal conèixer:

- `read` - `browse` - `search` - `search_count`

Tots aquests mètodes cal executar-los sobre una classe:

- `self.mètode` per executar-lo sobre la pròpia classe
- `self.env['nomClasse'].mètode` per executar-lo sobre una altra classe

✓ El mètode `search` permet efectuar una cerca o comptatge de registres que compleixin una determinada condició. Mireu sintaxis en documentació d'0doo.

Paràmetre `count` que pot prendre valors `True` o `False`:

- Si `False` (valor per defecte), `search` retorna registres
- Si `True`, `search` retorna quantitat de registres

Paràmetre `limit`:

- Si `count=False`, màxim nombre de registres a recuperar
- Si `count=True`, no té efecte

✓ El mètode `search_count` és una variant que equival a `search` amb `count=True`

✓ El mètode `browse` permet obtenir un `recordset` a partir dels `id` dels objectes (cal tenir els `id`).

✓ El mètode `read` permet obtenir una llista de diccionaris que contenen els camps demanats més `id`.

Exercici:

Afegir a la vista `tree` de `Teacher` les columnes:

- Nombre de cursos dels que és responsable
- Nombre d'assignatures que pot impartir
- Nombre de docències assignades

La versió 12 del mòdul `school` incorpora els 3 camps calculats en el model de la classe `SchoolTeacher` i a la vista `tree`. El càlcul dels 2 primers és molt senzill per què tenim la sort que la classe `SchoolTeacher` té els camps `course_ids` i `subject_ids` i només mirant la quantitat d'elements que tenen aquestes llistes, ja tenim el camp calculat. Però el 3r càlcul és més complicat... No tenim més remei que situar-nos a la classe `SchoolTeaching` i allà comptar quants registres hi ha que tinguin per `teacher_id` al professor pel que estem efectuant el càlcul. Cal utilitar el mètode `search` amb el paràmetre `count` o el mètode `search_count`.

Controlador: Sobreescriptura de mètode `unlink`

En ocasions ens pot interessar sobreescriure el mètode `unlink` per canviar el funcionament per defecte.

Aquesta necessitat pot tenir lloc, per exemple:

- Per personalitzar el missatge que apareix en intentar eliminar un registre que té informació vinculada (via relació `Many2one` sense `ondelete='cascade'`), com passa en intentar eliminar un professor que té responsabilitat en algun curs.
- Per eliminar registres d'altres classes abans que procedir a l'eliminació del propi registre, com passa si es té registres vinculats en altres classes sense `ondelete='cascade'` en el model i es vol procedir a l'eliminació prèvia dels registres vinculats.

Exercici:

Aconseguir el següent funcionament en l'eliminació de professorat:

- Si algun professor és responsable d'algun curs, avortar informant amb missatge clar.
- Si cap professor és responsable de cap curs, procedir a la seva eliminació prèvia eliminació de les docències que pugui tenir assignades.

Solució en el mètode `unlink` de la classe `SchoolTeacher` de la versió 12 del mòdul `school`. Observeu les diverses possibilitats d'actuació... i de situacions problemàtiques... comentades dins el codi.

Controlador: Sobreescriptura de mètode `create`

El mètode create permet inserir molts registres, com a mínim 1.

Per sobreescriure el mètode `create`:

```
@api.model_create_multi
def create(self,values):
  # values és una llista de diccionaris amb els valors dels camps
  # Cada diccionari correspon a un registre a inserir
  # Com que ha de poder inserir molts, la llista permet tenir molts diccionaris
  ... # Aquí posarem les comprovacions/canvis que calgui efectuar abans de crear
  r = super(nomClasse,self).create(values)   # Aquí efectuem la creació
  # r és la llista dels objectes creats, que caldrà retornar obligatòriament
  # Els registres de "r" ja tenen "id", cosa que no tenen abans de la seva creació
  ... # Aquí posarem accions que calgui efectuar una vegada creat
  return r   # Per conveni, el mètode create retorna la llista "r" d'obj. creats
```

`values` pot ser una llista de diccionaris en cas que algun mètode enviés, en una sola transacció, un conjunt de registres a crear. Això no succeeix quan el mètode s'executa com a conseqüència de la creació d'un registre via vista `form`, però cal programar el mètode per si arriba una llista.

Controlador: Sobreescriptura de mètode `write`

Per sobreescriure el mètode `write`:

```
def write(self,values):
  # values és un diccionari que conté els valors dels camps a modificar
  # Només conté els camps que es modifiquen, NO tots
  # self conté els registres que es modificaran
  ... # Aquí posarem les comprovacions/canvis que calgui efectuar pre modificació
  r = super(nomClasse,self).write(values)   # Aquí efectuem la modificació
  # r un booleà indicant si l'operació s'ha pogut efectuar
  ... # Aquí posarem accions que calgui efectuar una vegada modificats
  return r   # Per conveni, el mètode write retorna el valor "r"
```

La modificació s'efectua per tots els registres dins `self` on se'ls aplica els mateixos `values`. Quan el mètode s'executa com a conseqüència de la modificació d'un registre via vista `form`, `self` només conté 1 registre, però cal programar el mètode tenint en compte que hi pot haver més d'1 registre, per si algun mètode intenta fer una modificació massiva.

Gestió dels valors `Datetime`

Els valors `Datetime` emmagatzemen moments temporals i *Python* facilita la classe `datetime.datetime` per a la seva gestió.

Si en una consola *Python* fem:

```
from datetime import datetime
print (datetime.now())
```

obtenim el moment actual del sistema operatiu (`yyyy-mm-dd hh:mm:ss.mil·lèsimesDeSegon`)

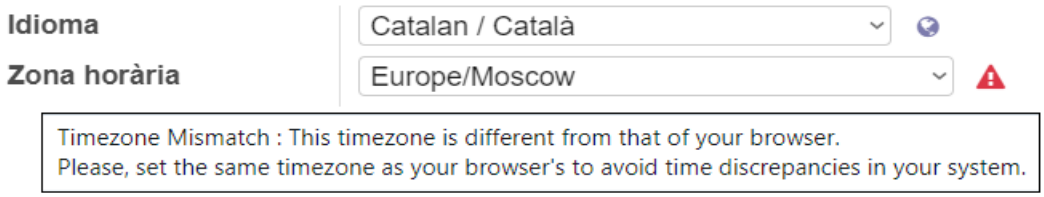
Però Odoo està dissenyat per què tots els valors `datetime` siguin transformats al valor en el fus horari UTC (*Coordinated Universal Time* - fus horari situat sobre el meridià de Greenwich) i:

- S'enregistren a la BD en UTC
- Es mostren en el navegador segons el fus horari de la màquina on s'executa el navegador.

En ocasions, però, es necessita disposar d'un valor `Datetime` en el fus horari de la màquina:

- Si es necessita en el desenvolupament client (*JavaScript*), es pot accedir al fus horari de la màquina.
- Si es necessita en el desenvolupament servidor (*Python*), no es pot accedir al fus horari de la màquina i hem d'usar el fus horari indicat en les preferències de l'usuari.

  Però per no tenir problemes és necessari que el fus horari de les preferències de l'usuari coincideixi amb el fus horari de la màquina on s'executa el navegador. Per això, si no hi ha coincidència entre els dos fusos horaris, el client web d'Odoo mostra la següent advertència:



En el desenvolupament servidor, per transformar un valor `datetime` (UTC) al fus horari de l'usuari (el que té seleccionat a les seves preferències), disposem de:

- `fields.Datetime.context_timestamp(self,<valor_datetime>)`
- O usar mètodes de conversió de *Python* a partir del fus horari (`tz`) de l'usuari, que es pot trobar de diverses maneres:
  - Valor de la clau `'tz'` en el diccionari `self.env.context`
  - Camp `res_user.res_partner.tz` per l'usuari actiu, que es coneix per `self.env.uid`.

Exercicis:

1. Si des del formulari de `Teacher` intenteu desassignar un dels seus cursos, l'Odoo llença un error doncs no és possible que un curs es quedi sense professor. El problema és que el text de l'error no és entenedor per a un usuari d'Odoo.

Intercepteu la situació i llenceu un error entenedor per a un usuari (p.e.: *No es pot deixar un curs sense professor responsable*).

Pista: L'error es produeix quan en desassignar el curs, Odoo ha d'actualitzar el curs a la BD sense professor responsable i això passa en executar-se el mètode `write` de `SchoolCourse`. Així doncs, cal sobreescriure aquest mètode, per detectar quan es produeixi el problema i llençar un error entenedor.

2. Es vol guardar la història de tots els salaris que han tingut els diversos professors, des del moment en què se'ls dona d'alta i en les successives modificacions. Per aconseguir-ho, es decideix:

- Crear la classe `SchoolTeacherSalaryHistory` per a que Odoo enregistri la història dels salaris, amb els camps: `teacher_id`, `user_id`, `date`, `time`, `salary`.

  Es vol que el camp `time` guardi l'hora local de la màquina on s'ha efectuat el moviment.

- Facilitar una vista `tree` (sense `form`) per poder consultar (només lectura) la història dels salaris, visualitzant-los ordenats per data-hora en descendent.

- Automatitzar els enregistraments en aquesta classe, que cal efectuar-los en dues situacions:
  - En donar d'alta un professor, per guardar el salari inicial.
  - En modificar el salari d'un professor, per guardar el canvi.

- No permetre l'eliminació de la història dels salaris.

Nota: L'usuari actual es pot obtenir amb l'expressió `self.env.uid`

La versió 13 del mòdul `school` incorpora la solució.

Respecte exercici 1:

- Sobreescriptura de `write` a `SchoolCourse`

Respecte exercici 2:

- Creació de la classe i vista indicades.
- Respecte el camp per guardar hora-minut, la solució mostra 2 possibilitats:
  `time_s`, com a "cadena"
  `time_f`, com a "`float`" i usem giny `float_time` per a que es visualitzi en format `hh:mm`
  En aquest cas, com que són camps de només consulta, és igual usar `time_s` que `time_f`, però en cas de camps editables, cal usar la versió "`float`" amb el giny, doncs controla que el què s'introdueixi sigui valor numèric factible, mentre que la versió "cadena" no controla res i caldria dissenyar un mètode `check`.
- Sobreescriure mètode `create` a la classe `Teacher`, per enregistrar el salari inicial del professor.
- Sobreescriure mètode `write` a la classe `Teacher`, per enregistrar qualsevol modificació de salari.
- Afegir a la vista `tree` atribut `delete="0"` per a que no aparegui la possibilitat d'eliminar registres. Aprofitem també per afegir atribut `create="0"` per a que no aparegui el botó de creació.
- Malgrat des de la vista `tree` no es pugui eliminar, és convenient sobreescriure el mètode `unlink` per si algún altre mètode intenta eliminar els registres. Això no afecta el `ondelete='cascade'` incorporat en el model.

Herència de classe i de vista

- Informació: Apartat 1.1 del dossier DAM_M10_UF2_B2.
- Utilitzarem herència de classe per ampliar/retocar classes ja existents en altres mòduls.

Info: https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#inheritance-and-extension

- Utilitzarem herència de vista per ampliar/retocar vistes ja existents en altres mòduls.

Info: https://www.odoo.com/documentation/16.0/developer/reference/backend/views.html#inheritance

Exemple:

Es vol incorporar, a la fitxa dels empleats, a la part final de la zona de dades privades, informació sobre algunes característiques físiques, com:

- Altura (en cm)
- Color dels ulls (a escollir entre un conjunt de noms de colors que no pot tenir valors repetits)

Solució en mòdul `hr_physical_characteristics` (`01_hr_physical_characteristics.zip`):

- Cal crear un nou mòdul: `hr_physical characteristics`
- Classe per definir els colors: `HrEmployeeEyeColour` (`hr.employee.eye.colour`)
- Classe `HrEmployeePrivatePhysicalCharacteristics` que hereta de `hr.employee` per incorporar les característiques físiques requerides.
- Vista `view_employee_physical_characteristics_form` que hereta de `hr.view_employee_form` per incorporar les característiques físiques requerides a la zona indicada de la fitxa dels empleats.
  En executar aquesta vista, la pestanya *Informació privada* incorpora a la part final la zona *Physical Characteristics*, amb els camps *Height* i *Eye colour*. Aquest darrer camp permet, com tots els camps `Many2one`, seleccionar un color dels existents o donar-ne un d'alta. La versió `01` del mòdul no ha incorporat cap opció de menú per gestionar els colors.

Exercici:

Retocar la versió `01` del mòdul `hr_physical_characteristics`, ampliant-lo amb:

- Crear vista `tree` editable per gestionar els colors.
- Dins apartat *Configuration* del menú *Empleats*, incorporar al final del subapartat (submenú) *Empleat*, l'opció de menú *Eye Colours*, que mostri la vista `tree` anterior.
- Ampliar la vista `tree` dels empleats de manera que contingui:
  Columna *Height* abans de la columna *Gerent*, de manera que l'usuari la pugui amagar.
  Columna *Eye colour* com a columna amagada que l'usuari pugui fer visible.

Atenció! Quan s'ha d'indicar un `id` d'una vista/menu ubicat en un altre mòdul, cal precedir l'`id` pel nom del mòdul on existeix la vista/menu al que es fa referència.

Solució en mòdul `hr_physical_characteristics` (`02_hr_physical_characteristics.zip`):

- Vista `view_employee_physical_characteristics_tree` que hereta de `hr.view_employee_tree` per incorporar columnes requerides a la zona indicada de la vista graella d'empleats.
- Vista `view_hr_employee_eye_colour_tree` per mostrar graella editable de colors d'ull d'empleats.
- *Action* `action_hr_employee_eye_colour` per invocar la vista anterior.
- *Menuitem* `menu_hr_employee_eye_colour` per invocar l'acció anterior i ubicat com a fill del menú `hr.menu_config_employee` (que és submenú de `hr.menu_human_resources_configuration`).
  Per localitzar el `id` del menú pare, ha calgut cercar-lo dins els fitxers XML del mòdul `hr`. Les eines de desenvolupament (marieta) no faciliten ajuda per localitzar els `id` de menús.

Pràctica 1

Veure enunciat en PDF específic.

Incorporació de dades (demo i no demo)

Informació:

- Apartat 1.3 del dossier DAM_M10_UF2_B2.
- https://www.odoo.com/documentation/16.0/developer/reference/backend/data.html#data-files

Exemple: `14_school.zip`

- Carpeta `data` conté arxius XML amb les dades. S'acostuma a utilitzar carpeta `data`.
- Dins el fitxer `_manifest_.py` cal incorporar:
  - Fitxers amb dades demo, a l'apartat `demo`.
  - Fitxers amb dades que cal incorporar obligatòriament en instal·lar el mòdul, a l'apartat `data`.
- Per aconseguir que les dades (demo i no demo) només s'instal·lin durant el procés d'instal·lació del mòdul i no pas en els processos d'actualització, cal introduir totes les dades dins un node:
  `<data noupdate="1"> ...dades... </data>`
  Si hi ha dades a reinstal·lar a cada actualització, han d'estar fora del node `<data noupdate="1">`.
  En desenvolupament, quan es vol comprovar la correcta instal·lació de les dades (demo o no demo), per no haver de desinstal·lar mòdul i tornar-lo a instal·lar, en la configuració que usem per reiniciar l'Odoo en mode actualització d'un mòdul, podem substituir `-u nomModul` per `-i nomModul`, i d'aquesta manera Odoo es creu que està instal·lant el mòdul i llavors hi instal·la les dades.
- Cada registre ha d'estar en un element `<record>` amb `id` identificatiu del registre. NO és l'identificador que PostgreSQL assignarà al registre dins la taula a la BD. És un valor alfanumèric i s'aconsella utilitzar com `id` un nom amb significat del registre al que correspon, doncs serà més fàcil de fer-ne referència (atribut `ref` explicat més endavant).
- Respecte els camps que s'ha d'emplenar en una càrrega de dades:
  - Els obligatoris que no tenen valor per defecte
  - Els obligatoris amb valor per defecte al que se'ls vol donar un valor diferent del valor per defecte
  - NO els `computed` ni els `related`
  - Els no obligatoris que interessi emplenar
- Les dades es poden introduir en anglès i es pot incorporar la seva traducció en traduir el mòdul.
- Com emplenar els camps relacionals:

  - Per camp `One2many`: S'aconsella no emplenar-los per aquí sinó pel camp `Many2one` invers. Sempre que hi ha un `One2many` és necessari l'existència del camp `Many2one` i és molt més fàcil introduir la dada en aquest camp.
  - Per camp `Many2one`: Cal usar atribut `ref` i cal introduir el `id` del registre apuntat.
    Per apuntar un registre d'una altra classe cal conèixer el seu `id` i ha d'estar instal·lat a la BD abans de fer-ne referència via `ref`. Si està en un altre mòdul, cal indicar `nomModul.id`. Veure, per exemple, com indicar la nacionalitat d'un professor (camp `country_id`)
  - Per camp `Many2many`, si és bidireccional, només s'introdueix per una de les dues classes afectades. Veure, per exemple, com s'emplena `teacher_ids` a la classe `school.subject`.

  Pel que respecta a la sintaxi a emprar pel camp `Many2many` (i també pel camp `One2many` si no es vol seguir el consell d'omplir-lo via el `Many2one` invers), cal seguir la següent sintaxi:

```
+ For a Many2many field, a list of tuples is expected.
  Here is the list of tuple that are accepted, with the corresponding semantics ::
     (0, 0,  { values }) link to a new record that needs to be created with the given values dictionary
     (1, ID, { values }) update the linked record with id = ID (write *values* on it)
     (2, ID)             remove and delete the linked record with id = ID (calls unlink on ID, that will delete
                         the object completely, and the link to it as well)
     (3, ID)             cut the link to the linked record with id = ID (delete the relationship between the two
                         objects but does not delete the target object itself)
     (4, ID)             link to existing record with id = ID (adds a relationship)
     (5)                 unlink all (like using (3,ID) for all linked records)
     (6, 0, [IDs])       replace the list of linked IDs (like using (5) then (4,ID) for each ID in the list of IDs)

     Example:
        [(6, 0, [8, 5, 6, 4])] sets the many2many to ids [8, 5, 6, 4]
```

```
+ For a One2many field, a lits of tuples is expected.
  Here is the list of tuple that are accepted, with the corresponding semantics ::
     (0, 0,  { values }) link to a new record that needs to be created with the given values dictionary
     (1, ID, { values }) update the linked record with id = ID (write *values* on it)
     (2, ID)             remove and delete the linked record with id = ID (calls unlink on ID, that will delete the
                         object completely, and the link to it as well)

     Example:
        [(0, 0, {'field_name':field_value_record1, ...}), (0, 0, {'field_name':field_value_record2, ...})]
```

- Atribut `eval` per introduir valors que es calculen en el moment d'inserir les dades. Cal utilitzar-lo en:
  - Introducció de dates dinàmiques (per crear-les en funció del moment en què s'instal·la el mòdul). Veure, per exemple, les dates inici i final de les dades `demo` per `school.course.edition`, on les dates es crearan en funció de la data d'instal·lació (cal utilitzar funcions de Python).
  - Definició de camps `One2many` i `Many2many`.
    Ja s'ha dit que no és habitual emplenar camps `One2many`. Més fàcil fer-ho via l'invers `Many2one`.
    Veure, per exemple, com s'emplena el camp `teacher_ids` a la classe `school.subject`.
  - També en camps booleans:
    `<field name="nomCamp" eval="True"/>`
    `<field name="nomCamp" eval="False"/>`
- Les imatges es poden introduir de dues maneres:
  - `<field name="campImatge">cadenaDeImatgeEnFormat_base64</field>`
  - `<field name="campImatge" type="base64" file="rutaAfitxerImatge"/>`

  La versió 14 del mòdul `school` mostra les dues possibilitats en dades demo de dos professors.
  A la web es troben convertidors de formats imatge a format `base64`, com per exemple aquest.
- Les dades demo de la versió 14 del mòdul `school` utilitza dos fitxers per què incorpora les fotos en format `base64` en un segon fitxer XML, i així el primer fitxer XML és més llegible. Això no s'hagués pogut fer si la foto fos obligatòria, doncs caldria incorporar-la quan es crea el professor.

Exercici

Ampliar el conjunt de dades demo que incorpora la versió 14 del mòdul `school` amb dades de docència, de manera que una vegada instal·lades s'apreciïn les docències:

Course Edition	Subject	Teacher
Cross-platform application development - Edition 2023	Computer systems	Peres Peres, Maria
Cross-platform application development - Edition 2023	Access to data	Peres Peres, Maria
Cross-platform application development - Edition 2023	Web development in client environment	Rodriguez Rodriguez, Josep

Solució: Versió 15 del mòdul `school`.