



Informàtica

ICB0 Desenvolupament d'aplicacions multiplataforma

M10 Sistemes de gestió empresarial
UF2 Sistemes ERP-CRM. Explotació i adequació.

Odoo17: Desenvolupament.

Isidre Guixà – Ester Marsal

Curs 2024/2025



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 17, apareixeran paràgrafs acolorits com aquest, que contenen informació per versions anteriors o informació prescindible d'usar en aquest curs, però que és interessant mantenir.

[Llista de reproducció YouTube](#) amb molts temes de desenvolupament en Odoo15

Índex

La BD de l'Odoo	5
Pràctica 1	8
Instal·lació PyCharm com IDE per Odoo17 en Windows	8
• I si els PDF d'Odoo surten en blanc? Causa i solució.....	9
Desenvolupament en Odoo 17 - Backend.....	10
Estructura d'un mòdul.....	10
Nomenclatura bàsica	11
Camps bàsics i vistes bàsiques (tree i form)	11
• Vistes tree editables.....	15
• Com desactivar creació-eliminació-modificació-duplicació en vistes?	15
• Com canviar el nom d'un camp en el model?	15
• Camps traduïbles	15
Camps relacionals	16
• Camp Many2one.....	16
• Camp One2many.....	17
• Camp Many2many.....	18
• Giny one2many o many2many?.....	19
Pestanyes en un formulari.....	21
Relacions reflexives.....	21
Controlador: Camps calculats / representació textual de registre / restriccions	23
• Com depurar el codi?	24
• Camps calculats	24
• Sobreescritura del mètode _compute_display_name / name_get	25
• Restriccions d'Odoo	26
• Com controlar recursivitat infinita en una classe reflexiva?.....	27
Disseny de relació *: * amb atributs.....	27
Ordre de visualització de registres	30
Camps related.....	30
Controlador: Mètode onchange.....	31
Unicitat d'un o varis camps.....	32
• Restricció d'unicitat vs majúscules/minúscules	32
Filtres en model i/o vistes – domain context.....	34
• Com desactivar active en una graella? – Com distingir registres via format visual?	35
• Camps calculats en filtres.....	36
Exercicis pre-parcial (pràctica 2 i versió 10 d'school).....	36
Atributs readonly-invisible-required en vistes (Odoo 16-).....	38
Atributs readonly-invisible-required en vistes (Odoo 17+).....	39
Vistes calendar.....	39
Vistes graph	40
Vistes search.....	41
Controlador: Mètodes search/read.....	42
Controlador: Sobreescritura de mètode unlink.....	43
Controlador: Sobreescritura de mètode create.....	44
Controlador: Sobreescritura de mètode write.....	44
Gestió dels valors Datetime.....	46



Herència de classe i de vista	47
Pràctica 3.....	48
Incorporació de dades (demo i no demo)	48
Informes QWeb.....	50
• Compte a l'hora de traduir	52
• Altres eines de reporting.....	52
Definició de l'esquema de seguretat.....	53
• Com incorporar-usar usuaris demo en un mòdul?	54
Assistents	55
Models no persistents.....	56
Quadres de comandament	57
Traducció de mòdul	59
Pràctica 4.....	60

La BD de l'Odoo

Seguim l'apartat 1.2 del dossier DAM_M10_UF2_B1 que és vigent.

Objectiu final: **Facilitar accés de lectura a la BD d'una empresa per a què si pugui accedir des d'eines ofimàtiques o, en cas d'usuaris avantatjats, des d'eines clients de PostgreSQL.**

Conceptes a conèixer:

- Detectar taules i camps on resideix la informació, sense consultar el codi font de l'Odoo.
- La majoria de taules tenen el camp `id` autonumèric com a clau primària.
- Eines client PostgreSQL pensades per a ser usades per usuaris informàtics:
 - *pgAdmin*
 - *psql*. Mínimes nocions en apartat 2.4.4. del [dossier de M10-UF1](#).
Ordres a conèixer:
 - `\h` instruccióSQL per obtenir informació sobre la sintaxi de l'instrucció indicada
 - `\?` per veure totes les ordres que reconeix la consola *psql*.
 - `\dt` per veure totes les taules
 - `\du` per veure els usuaris
 - `\l` per veure les bases de dades
 - `\d nomTaula` per veure l'estructura de la BD (camps, índexs, claus foranes i altres taules que fan referència a la taula indicada)
 - `\q` per abandonar la consola
- Esquemes de les BD de PostgreSQL:
 - Una BD de PostgreSQL s'organitza en esquemes, que són col·leccions d'estructures lògiques de dades (taules, vistes, funcions, procediments, disparadors, seqüències,...)
 - La creació d'una BD de PostgreSQL incorpora un esquema anomenat `public` que, com el seu nom indica, és públic i hi té accés qualsevol usuari.
 - Una BD de PostgreSQL pot contenir més esquemes, que cal crear via `CREATE SCHEMA`.
 - Els esquemes tenen propietari, el qual haurà d'establir permisos d'accés als continguts de l'esquema per la resta d'usuaris.
 - Qualsevol usuari en connectar a una BD (PostgreSQL 14-) de la que no n'és el propietari, està usant l'esquema `public`, en el que:
 - Pot crear taules i ser-ne el propietari
 - Pot veure els noms de les estructures (taules, vistes,...) existents dins l'esquema
 - No pot veure accedir al contingut de les taules mentre no se li concedeixi els privilegis adequats.
- Odoo emmagatzema totes les seves estructures lògiques (taules, vistes, seqüències,...) dins l'esquema `public` i, per tant, qualsevol usuari de PostgreSQL 14- pot accedir-hi i "embrutar-lo" afegint taules alienes a l'Odoo. Seria millor que Odoo creés un esquema per a les seves estructures, però no és el cas.

PostgreSQL 15+ canvia els permisos sobre l'esquema `public`. Bona explicació [aquí](#).

Fixeu-vos que a la part final de l'article també es comenta la conveniència de només usar l'esquema `public` per a usos causals...

En conseqüència, si es vol facilitar l'accés de consulta a usuaris diferents de l'usuari que usa l'Odoo per connectar amb el PostgreSQL14-, es fa necessari "protegir" l'esquema `public` d'accessos no desitjats.



`revoke all privileges on schema public from public;`

per eliminar els privilegis (només n'hi ha dos: `usage` i `create`) sobre l'esquema `public` a tots els usuaris (excepte el propietari de l'esquema que, en PostgreSQL 14- no és el propietari de la BD sinó el propietari de la base de dades `template1` conegut com a *bootstrap user*); aquesta instrucció l'ha d'executar l'usuari propietari de l'esquema o un superusuari

Després d'executar la instrucció anterior, l'usuari propietari de la BD si no coincideix amb l'usuari propietari de l'esquema, deixa de tenir accés a l'esquema públic, cosa que pot ser problemàtica i, en aquest cas, tenim dues possibilitats:

- a) Com a usuari propietari de l'esquema, donar la propietat de l'esquema `public` a l'usuari propietari de la BD:

`alter schema public owner to <usuariPropietariDeLaBD>;`

- b) Com a usuari propietari de l'esquema, donar tots els privilegis sobre l'esquema `public` al'usuari propietari de la BD (probablement millor l'opció anterior):

`grant all privileges on schema public to <usuariPropietariDeLaBD>;`

Per facilitar l'ús d'un esquema a un usuari concret, una vegada ja s'ha eliminat l'accés a tothom:

`grant usage on schema public to <usuariQueInteressiDonarAccés>;`

- Facilitar accés de només lectura a la BD a usuaris determinats (detall en dossier i aquí resumit):
 - Crearem usuaris de Postgres que es puguin connectar i sense cap altre privilegi, per a cada usuari de l'organització a qui es vulgui donar accés de lectura a la BD.
 - Configurarem servidor Postgres (`postgresql.conf`) per admetre connexions remotes per IP(s) local(s) que correspinguin.
 - Facilitarem accés remot (`pg_hba.conf`) als usuaris creats a Postgres des de les IPs que corresponguin i només per a la BD a les que hagin de poder accedir.
 - Protegirem l'esquema `public` de les BD de l'Odoo, com s'ha indicat prèviament.
 - Facilitarem accés de lectura (`SELECT`) a les taules o columnes de taules que interressi.

La possibilitat de concedir accés a columnes concretes apareix a la versió 8.4 de PostgreSQL.

- Millor que facilitar accés de lectura a les taules o columnes de taules per a que l'usuari pugui executar consultes, és dissenyar vistes amb les consultes ja dissenyades i donar accés a les vistes.

Aquestes vistes es poden enregistrar en el mateix esquema o en un altre esquema. En cas de residir en el mateix esquema, cal usar noms que no puguin interferir amb els noms que usa Odoo pels seus objectes; per exemple, usant el nom de l'organització com a prefix dels nom (`mif_nomVista`)

- Concedirem privilegi de lectura a cada vista per a cada usuari que correspongui:


```
grant select on <nomVista> to <nomUsuari>;
```

- Camps `jsonb` per Odoo 16+:

Els camps traduïbles són emmagatzemats a la BD en columnes de tipus `jsonb`. Per tant, si en una consulta SQL incorporem un camp traduïble, veurem el contingut del camp `jsonb`. Per exemple:

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

Evidentment cal gestionar camps `jsonb`: <https://www.postgresql.org/docs/12/functions-json.html>.

Així, per exemple si per un camp `xxx` volem veure el valor en català, escriurem `xxx->>'ca_ES'`.

- Com accedir des d'aplicacions ofimàtiques a la BD

L'[annex 03 - Accés a SGBD corporatives des d'eines ofimàtiques](#) ho detalla en la BD d'OpenERP 6.1 i utilitzant les aplicacions ofimàtiques coetànies de l'OpenERP 6.1. Les explicacions continuen sent aplicables en aplicacions ofimàtiques actuals.

- Des d'una BD de LibreOffice
 - Via connector PostgreSQL facilitat per PostgreSQL
 - Via connexió JDBC, prèvia detecció, descàrrega i instal·lació del connector JDBC adequat
 - Via ODBC, prèvia instal·lació del connector ODBC (veure comentaris següents)

Alerta: Si una consulta escrita en LibreOffice conté algun camp `jsonb` i s'usen operadors json, cal indicar a LibreOffice que executi la consulta SQL directament, fet que s'aconsegueix activant la corresponent icona:



Aquest problema no té lloc si la consulta es crea com una vista a la BD i des de la consulta de LibreOffice s'invoca la vista.

- Des de Microsoft Acces, via ODBC (veure comentaris següents)
- En referència a les connexions ODBC, cal tenir en compte:
 - Diferències respecte si l'aplicació que usa ODBC és de 32 bits o de 64 bits:
 - ✓ Si l'aplicació que connecta és de 32 bits, cal tenir DSN declarat en Administrador d'orígens de dades ODBC (32 bits). MsAccess de l'escola és de 32 bits.
 - ✓ Si l'aplicació que connecta és de 64 bits, cal tenir DSN declarat en Administrador d'orígens de dades ODBC (64 bits). LibreOffice de l'escola és de 64 bits.
 - Problemes de posta en marxa de l'Administrador d'orígens de dades ODBC que pertorqui (32 bits o 64 bits):
 - ✓ L'explicació del 2n paràgraf de la pàgina 11 de l'annex, és vàlida en Windows 7 (i potser 8 ¿?¿?) i això portava molt mals de cap a l'hora de definir els DSN
 - ✓ Windows 10 ha millorat la gestió i facilita dues aplicacions diferents:
 - Administrador d'orígens de dades ODBC (32 bits)
 - Administrador d'orígens de dades ODBC (64 bits)que poden estar simultàniament obertes i queden identificades pel títol de la part superior de la finestra.
 - LibreOffice (que tenim a escola) via ODBC, si la vista a la que es vol accedir treballa amb més d'una taula, apareix error:

`ERROR: column "ctid" does not exist`

Totes les taules de PostgreSQL tenen una columna oculta de nom "ctid" i la connexió de LibreOffice via ODBC obliga a que la vista accedida tingui una columna "ctid", cosa que no succeeix amb MsAccess via ODBC.

Solució: Retocar la definició de la vista incorporant columna "ctid" que coincideixi amb "ctid" d'una de les taules de la vista.

- **Pràctiques** desenvolupades a classe:



- S'ha creat dins el PostgreSQL un usuari `pepe` amb privilegi de connexió.
 - S'ha configurat PostgreSQL per a que `pepe` es pugui connectar des de les IPs que corresponguin a la base de dades que correspongui.
 - S'ha "protegit" l'esquema `public` donant accés només a usuari que Odoo usa per connectar.
 - S'ha donat accés `usage` a l'usuari `pepe` a l'esquema `public`.
 - S'ha creat vista `mif_pepe_partner_simple` que mostra el nom del tercer i l'idioma, que és la informació a la que ha de tenir accés `pepe`.
 - S'ha comprovat, via `psql` o `pgAdmin` que l'usuari `pepe`, des de les IPs que corresponguin, es pot connectar a la BD i pot executar la vista desenvolupada.
 - S'ha creat base de dades de *LibreOffice* que connecta amb servidor PostgreSQL via connector per a PostgreSQL incorporat dins *LibreOffice*.
 - S'ha creat base de dades de *LibreOffice* que connecta amb servidor PostgreSQL via connector JDBC per a PostgreSQL, prèvia descàrrega del driver JDBC per a PostgreSQL adequat i s'ha instal·lat en LibreOffice.
 - S'ha comprovat que l'*Administrador d'ODBC* del nostre Windows (escola) no disposa del controlador per a PostgreSQL. En conseqüència, s'ha descarregat els drivers ODBC per a PostgreSQL i s'han instal·lat en Windows, de manera que apareixen a la pestanya *controladors* de l'Administrador d'ODBC.
 - S'ha creat DSN de 32 bits per connectar aplicació de 32 bits amb servidor PostgreSQL via ODBC i s'ha comprovat connexió des de *MsAccess* de l'escola que és de 32bits, accedint a la vista de PostgreSQL via taula vinculada.
 - S'ha creat DSN de 64 bits per connectar aplicació de 64 bits amb servidor PostgreSQL via ODBC i s'ha comprovat connexió des de *LibreOffice* de l'escola que és de 64bits i per accedir a la vista de PostgreSQL s'ha hagut de retocar la vista afegint columna "ctid".
 - S'ha comprovat connexió des d'*Excel* de l'escola que és de 32 bits, creant una taula vinculada a la vista de PostgreSQL.
- Com actualitzar la informació de l'origen de dades (IP, port, usuari o DSN) en una BD ofimàtica?
 - En LibreOffice, via opció *Edita | Base de dades | Tipus Connexió*
 - En MsAccess, via opció *Administrador de tablas vinculadas* (ubicació segons versió MsAccess)

Pràctica 1

Veure enunciat en PDF específic.

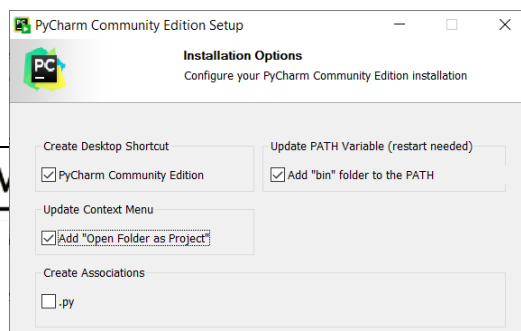
Instal·lació PyCharm com IDE per Odoo17 en Windows

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

- Odoo17 està desenvolupat en Python 3.12.3 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'interpret 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 `Odoo17` (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 vídeo](#) a partir del minut 16:46, quan ja tenim el projecte carregat a *PyCharm*, tenint en compte que:
 - ✓ Com a *Python Interpret* no cal crear un nou interpret virtual (cosa que fariem si tiguéssim diversos Python per diverses aplicacions) sinó que n'hi ha prou en seleccionar interpret existent i seleccionar `Python.exe` de l'Odoo i que ja té totes les llibreries que necessita Odoo.
 - ✓ Cal crear una configuració per posar en marxa (llançadora) el projecte on:
 - *Name*: `Odoo17` (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.

Escala i disposició

Canvia la mida del text, de les aplicacions i d'altres elements

100% (valor recomanat) ▾

[Configuració d'escala avançada](#)

Desenvolupament en Odoo 17 - 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/17.0/>

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.

La instrucció per a que el propi Odoo creï un esquelet bàsic per a un mòdul:

```
python odoo-bin scaffold <nomMòdul> <pathCarpetaOnCrearLo>
```

que es pot executar des d'una consola de sistema o des d'una terminal de PyCharm. Podem usar aquesta opció o generar l'estructura manualment.

Ens centrem en l'esquelet bàsic per a un mòdul de nom `school` que anirem desenvolupant durant 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)

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/17.0/developer/reference/backend/orm.html#fields>

- Els paràmetres de la classe inicial `odoo.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,nomMòdul2,...> -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 (que ja està instal·lat) **si usem PyCharm**:
 1. Servei Odoo aturat (com sempre que s'estigui usant les llançadores de PyCharm)
 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 cap vista, no s'observa que el mòdul està instal·lat.

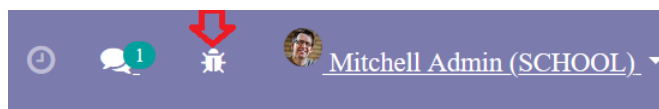


Desenvolupament bàsic de vista en fitxer XML

- ✓ 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`.

Doc: https://www.odoo.com/documentation/17.0/developer/reference/user_interface/view_architectures.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 d'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 `subject/course`
- Important emprar l'atribut `sequence` en els `menuitem` per assegurar l'ordre de visualització.
- Atribut `invisible="condicio"` en un camp per definir quan el camp (en un registre de la graella) ha de ser invisible. La columna es veu, però el contingut només pels camps que compleixen la condició. En Odoo 16- només podia ser `"True"` o `"1"` o `"False"` o `"0"`.
- En Odoo 17+, atribut `column_invisible="condició"` per fer invisible una columna sencera.

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



Doc: https://www.odoo.com/documentation/17.0/developer/reference/user_interface/view_architectures.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):

The screenshot shows a form with a title bar 'Title 1'. Below it, there's a group 'Title 1.1' containing two fields. Below that, there's another group 'Title 1.2' containing two fields. To the right of 'Title 1.2', there's a group 'Title 1.3' containing two fields. The form is rendered with dashed lines indicating the structure.

```
<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>
```

The screenshot shows a form with a title bar 'custom'. Below it, there's a group 'label b' containing two fields. Below that, there's another group containing two fields. To the right of the second group, there's a group containing two fields. The form is rendered with solid lines indicating the structure.

```
<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 colspan="12">
  <group colspan="8">
    <field name="h" />
  </group>
  <group colspan="4">
    <field name="i" />
  </group>
</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:

PERSONAL DATA				OTHER DATA	
First Name ?	???			eMail ?	???
Last Name ?	???			Phone ?	???
Gender ?	???	Birthdate ?	???	Tax ID ?	???
				Salary ?	0

A tenir en compte:

- Utilització de ginyes (widgets) per als camps que ho precisen (email, url,...).
Relació de ginyes 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

• 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 hem aplicat a la vista `list` de `school.course` dins `02_school`. En aquest cas, no és necessari dissenyar la vista `form` i si s'hi manté, Odoo no la utilitza.

• Com desactivar creació-eliminació-modificació-duplicació en vistes?

- [Atributs disponibles pel node `tree`](#) que ho faciliten
- [Atributs disponibles pel node `form`](#) que ho faciliten

• 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 el 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>

Atenció!!! En camps traduïbles és complicat pretendre unicitat (veurem més endavant com s'assoleix)

Ho hem aplicat en el camp `name` de cursos i assignatures de `02_school`.

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/17.0/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? La seva representació textual que:

En Odoo 16-, és el valor que calcula el mètode `name_get` existent a totes les classes i que, si no es sobreescriu, mostra:

En Odoo 17+, és el valor del camp `display_name` existent a totes les classes, que es pot modificar sobreescrivint el mètode `_compute_display_name` (més endavant ho veurem) i que per defecte mostra:

- Contingut del camp indicat en la clàusula `_rec_name` si s'ha definit en el disseny de la classe
- Si no hi ha clàusula `_rec_name`, el contingut del camp `name` de la classe
- Si la classe no conté camp `name`, el muntatge: `nomDeLaClasse, idDelRegistre`

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 té camp `name`. Observeu què mostra el camp `manager_id` a la vista formulari dels cursos si no incorporem la clàusula `_rec_name`:

`<school.teacher,núm>`.

El lògic serà que mostri la concatenació del cognom i del nom del registre apuntat però per aconseguir-ho ens caldrà implementar algun mètode. De moment, per sortir del pas, podem fer que es mostri el cognom incorporant `_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 (com és lògic en el model relacional).

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

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

Recomanable en el model incorporar `readonly=True` per a que en tots els formularis on aparegui la graella, aquesta sigui de només consulta i, si cal que sigui editable, s'hi afegirà `readonly=False`.

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 ... />
    ....
  </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" ...>
```

- Un camp `One2many` no fa cas de l'atribut `required`. En una graella editable, per exigir que contingui algun element, caldrà usar una restricció (veure més endavant).
- En cas que la graella sigui editable, observem que al final dels registres apareix l'opció *Afegir una línia* que sembla que hauria 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 interressi 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 -->
  </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.

Recomanable en el model incorporar `readonly=True` per a que en tots els formularis on aparegui la graella, aquesta sigui de només consulta i, si cal que sigui editable, s'hi afegirà `readonly=False`.

Un camp `Many2many` no fa cas de l'atribut `required`. En un vista amb camp editable, per exigir que contingui algun element, caldrà usar una restricció (veure més endavant).

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 giny `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 registres de la classe X susceptibles de ser escollits, amb opció 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 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 línia* 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 pegen d'un registre de la classe RY que no pot canviar.

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

Exercici:

Finalitzar la implementació de la versió 03 del mòdul incorporant:

- Dins els formulari de Cursos, la graella d'assignatures
- Dins el formulari d'Assignatures, la graella de Cursos i de Professors
- Dins el formulari de Professors, la graella de Cursos i 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 gestió 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 Thematic`:

<input type="checkbox"/>	Name	Parent Thematic	Child Thematics	Courses
<input type="checkbox"/>	Informàtica		1 registre	1 registre
<input type="checkbox"/>	Programació	Informàtica	Cap registre	Cap registre

- Vista form de `Thematic`:

Name	Informàtica		
Parent Thematic			
Child Thematics		Courses	
	Name	Hours	Teacher Manager
	Programació	DAM	2.000 Pepe

- Vista form de `Teacher`:

PERSONAL DATA				OTHER DATA	
First Name ???				eMail ???	
Last Name ???				Phone ???	
Gender ???	Birthdate ???			Tax ID ???	
				Citizenship ???	
				Salary 0	
Courses Managed		Subjects authorized			
Name				Hours	
Afegir una línia					

- Vista tree de `Course`:

Courses				Cercar...
<input type="button" value="Crear"/>	<input type="button" value="Importa"/>	<input type="button" value="Imprimir"/>		
<input type="button" value="Filtres"/> <input type="button" value="Agrupar per"/> <input type="button" value="Favorits"/>				1-4 / 4 < >
<input type="checkbox"/>	Name	Hours	Course Manager	Thematic
<input type="checkbox"/>	Desenvolupament d'aplicacions multiplataforma	1.999	Gotera	Programació
<input type="checkbox"/>	Desenvolupament d'aplicacions web	2.000	Brufau	
<input type="checkbox"/>	Sistemes microinformàtics a la xarxa	2.000	Brufau	
<input type="checkbox"/>	Jocs de rol	2.000	Orellana	



- Vista form de Course:

Name	Desenvolupament d'aplicacions multiplataforma		
Thematic	Programació	Hours	1.999
Teacher Manager	Gotera		

Subjects

Name	Hours
Sistemes de gestió empresarial	99
Programació	500

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` podem 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` podem 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 coherència 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.

Alerta! De moment la reflexivitat en les temàtiques permet entrar en recursivitat infinita, cosa que no podem permetre i haurem de solucionar. És a dir, ara permet situacions errònies com les següents:

- Temàtica X sigui filla d'ella mateixa
- Temàtica X sigui filla de temàtica Y i aquesta sigui filla de temàtica X
- Temàtica X sigui filla de temàtica Y i aquesta filla de temàtica Z i aquesta filla de temàtica X

Controlador: Camps calculats / representació textual de registre / restriccions

Conceptes d'Odoo a conèixer:

- Recordset:
<https://www.odoo.com/documentation/17.0/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/17.0/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.

En treballar amb PyCharm, com que l'Odoo s'executa com a procés, podem incorporar missatges amb la funció `print()`, que apareixeran a la consola de l'Odoo.

- 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/17.0/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 o usar la funció `print()` i veure els missatges per la consola.

• Camps calculats

Info: <https://www.odoo.com/documentation/17.0/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 `store=True` en la definició del camp.

- **Alerta!** El codi del mètode ha de tenir en compte si pot generar error en cas que algun dels camps que formen part del càlcul no tinguin valor (estigui buit = `False` en Python) i en tals situacions, usar `if` per distingir la situació i generar resultat que no provoqui error.

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 `manager_id` dins `school.course => usar-lo en _rec_name.`

Alerta en usar un camp calculat en `rec_name`:

En mostrar un registre en una vista `form`, sota el títol de la vista (extrem superior esquerra) apareix la representació textual del registre facilitada pel camp `display_name` (**mètode `name_get` en Odoo 16-**), que si no està sobreescrit, mostra el contingut del camp indicat en `_rec_name`.

Si s'usa un camp calculat en `_rec_name` cal tenir en compte que tots els camps estan inicialment buits (excepte els que tinguin valor `default`) i, en conseqüència, cal vigilar que els càlculs del mètode no generin error en aquesta situació. Ho hem hagut de tenir en compte en el càlcul de `full_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:

PERSONAL DATA				OTHER DATA	
First Name ?	Pepa			eMail ?	pepe.gotera@gmail.com
Last Name ?	Gotera			Phone ?	938050000
Birthdate ?	15/05/1981	Age ?	42	Citizenship ?	Andorra
Gender ?	Female			Tax ID ?	ES12345
				Salary ?	1.000

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.

• Sobreescritura del mètode `_compute_display_name` / `name_get`

Anteriorment ja s'ha comentat que allà on es necessiti la representació textual d'un registre, es mostra:

En Odoo 16-, el valor que calcula el mètode `name_get` existent a totes les classes i que, si no es sobreescriu, mostra:

En Odoo 17+, el valor del camp `display_name` existent a totes les classes, que es pot modificar sobreescrivint el mètode `_compute_display_name` (més endavant ho veurem) i que per defecte mostra:



- Contingut del camp indicat en la clàusula `_rec_name` si s'ha definit en el disseny de la classe
- Si no hi ha clàusula `_rec_name`, el contingut del camp `name` de la classe
- Si la classe no conté camp `name`, el muntatge: `nomDeLaClasse`, `idDelRegistre`

Per això, a la classe `SchoolTeacher` que no disposa de camp `name` varem usar el camp calculat `full_name` a la clàusula `_rec_name`, però haguéssim pogut:

- En Odoo 16-, desenvolupar mètode `name_get`, que ha de retornar una llista de tuples (`id`, `càlculQueInteressi`) per a cada registre afectat:

```
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
```

- En Odoo 17+, sobre escriure mètode `_compute_display_name` que és l'encarregat de calcular el valor que mostra el camp automàtic `display_name`:

```
def _compute_display_name(self):
    for record in self:
        record.display_name = càlculQueInteressi
```

Incorporem aquest mètode a la classe `Teacher`. (07_school)

Quan una classe té el mètode `display_name` (`name_get` en Odoo16-) és innecessari tenir definida la clàusula `_rec_name`.

Es eliminar el camp `full_name`?

En Odoo16- no, per què un mètode (`name_get`) no es podia usar en una vista.

En Odoo17+ sí, doncs el camp `display_name` es pot usar en les vistes.

• 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`. (07_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 informàtic 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 vàlids.
- Es pot aconseguir tenir `check` a la BD (més endavant ho veurem) però és millor que comprovi Odoo.

Exercicis (07_school):

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

La solució incorpora a l'arrel un fitxer `utils.py` amb una utilitat (trobad 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` s'explica, 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.

• Com controlar recursivitat infinita en una classe reflexiva?

Odoo facilita un mètode per controlar la recursivitat infinita dins una classe reflexiva on els camps que defineixen la reflexivitat s'anomenen `parent_id` i `child_ids`. Per això és interessant usar aquests camps en les classes reflexives.

El mètode és `_check_recursion`. Es pot veure a la classe `SchoolThematic` en el mòdul `07_school`, on és usat en definir restricció sobre el camp `parent_id`.

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 `08_school_UML`.

Pràctica: Evolucionar el mòdul `school` segons disseny `08_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 `08_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:

Name ? Desenvolupament d'aplicacions multiplataforma Thematic ? Desenvolupament Hours ? 1.000 Course Manager ? Gotera, Pepa	Summary ? Aquests estudis postobligatoris capaciten per desenvolupar, implantar, documentar i mantenir aplicacions informàtiques multiplataforma, utilitzant tecnologies i entorns de desenvolupament específics, garantint l'accés a les dades de forma segura i complint els criteris de usabilitat i qualitat exigides en els estàndards establerts.																
SUBJECTS <table border="1"> <thead> <tr> <th>Number</th> <th>Subject</th> </tr> </thead> <tbody> <tr> <td>3</td> <td>Programació</td> </tr> <tr> <td>10</td> <td>Sistemes de gestió empresarial</td> </tr> <tr> <td>1</td> <td>Sistemes Informàtics</td> </tr> <tr> <td>2</td> <td>Bases de dades</td> </tr> </tbody> </table> Afegir una línia	Number	Subject	3	Programació	10	Sistemes de gestió empresarial	1	Sistemes Informàtics	2	Bases de dades	EDITIONS <table border="1"> <thead> <tr> <th>Edition name</th> <th>Init date</th> <th>End date</th> </tr> </thead> <tbody> <tr> <td colspan="3">Afegir una línia</td> </tr> </tbody> </table>	Edition name	Init date	End date	Afegir una línia		
Number	Subject																
3	Programació																
10	Sistemes de gestió empresarial																
1	Sistemes Informàtics																
2	Bases de dades																
Edition name	Init date	End date															
Afegir una línia																	

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 interressi 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_fname 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 ruta `OnHiHaOdoo\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.



PERSONAL DATA		OTHER DATA	
First Name ?	Pepe	eMail ?	pepe.gotera@gmail.com
Last Name ?	Gotera	Phone ?	938050000
Birthdate ?	08/02/1990	Citizenship ?	Andorra
Age ?	34	Tax ID ?	ES12345678Z
Gender ?	Male	Salary ?	2.000
<input type="button" value="Courses"/> <input type="button" value="Subjects"/>			

MANAGED COURSES	
Name	Hours
Desenvolupament d'aplicacions multiplataforma	2.000 ✕
Desenvolupament d'aplicacions web	2.000 ✕
Afegir una línia	



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 08_school: 08_school.zip

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`.

Respecte els camps traduïbles, la clàusula `order` actua sobre els valors de l'idioma que té activat l'usuari. Fantàstic!

Exercici (09_school): Cursos, assignatures, temàtiques i edicions de curs es mostrin ordenats per nom; professorat ordenat per cognom-nom i assignatures de 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 9 del mòdul school (09_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. Si se'ls desactiva readonly, en un formulari permeten l'edició però els canvis no tenen efecte; per tant, millor no desactivar-ho.

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 (09_school): La vista form de cursos mostri la informació del manager similar a:

Name ? Desenvolupament d'aplicacions multiplataforma		Summary ?																											
Thematic ? Desenvolupament		Aquests estudis postobligatoris capaciten per desenvolupar, implantar, documentar i mantenir aplicacions informàtiques multiplataforma, utilitzant tecnologies i entorns de desenvolupament específics, garantint l'accés a les dades de forma segura i complint els criteris de usabilitat i qualitat exigides en els estàndards establerts.																											
Hours ? 2.000		Tenen una durada de 2.000 hores (1.683 lectives en un centre educatiu i 317 de pràctiques en un centre de treball) distribuïdes en dos cursos acadèmics.																											
MANAGER																													
Teacher ? Gotera, Pepe	Aquest estudi substitueix el cicle Desenvolupament d'Aplicacions Informàtiques (LOGSE)																												
Phone ?																													
eMail ? pgotera@gmail.com																													
Citizenship ? Andorra																													
SUBJECTS		EDITIONS																											
<table border="1"> <thead> <tr> <th>Number</th> <th>Subject</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Sistemes Informàtics</td> </tr> <tr> <td>2</td> <td>Bases de dades</td> </tr> <tr> <td>3</td> <td>Programació</td> </tr> <tr> <td>4</td> <td>Llenguatges de marques</td> </tr> <tr> <td>10</td> <td>Sistemes de gestió empresarial</td> </tr> </tbody> </table>	Number	Subject	1	Sistemes Informàtics	2	Bases de dades	3	Programació	4	Llenguatges de marques	10	Sistemes de gestió empresarial	<table border="1"> <thead> <tr> <th>Edition name</th> <th>Init date</th> <th>End date</th> </tr> </thead> <tbody> <tr> <td>DAM - 23/24</td> <td>12/09/2023</td> <td>23/06/2024</td> </tr> <tr> <td>DAM - 24/25</td> <td>12/09/2024</td> <td></td> </tr> <tr> <td>DAM - 22/23</td> <td>05/09/2022</td> <td></td> </tr> <tr> <td colspan="3">Afegir una línia</td> </tr> </tbody> </table>		Edition name	Init date	End date	DAM - 23/24	12/09/2023	23/06/2024	DAM - 24/25	12/09/2024		DAM - 22/23	05/09/2022		Afegir una línia		
Number	Subject																												
1	Sistemes Informàtics																												
2	Bases de dades																												
3	Programació																												
4	Llenguatges de marques																												
10	Sistemes de gestió empresarial																												
Edition name	Init date	End date																											
DAM - 23/24	12/09/2023	23/06/2024																											
DAM - 24/25	12/09/2024																												
DAM - 22/23	05/09/2022																												
Afegir una línia																													
Afegir una línia																													

Controlador: Mètode onchange

Info: <https://www.odoo.com/documentation/17.0/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.

Exercici (09_school):

- Camp tin de la classe SchoolTeacher sempre quedi en majúscules.
- Nom de cursos, assignatures i temàtiques comencin en majúscula i la resta minúscules.
- Nom d'edicions tingui tots els mots amb inicial en majúscula i resta en minúscules.

PERILL! Aquest mecanisme 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, podem 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, podem 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.

Exercici (09_school): 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 09_school) la verificació de que salari sigui positiu. No és habitual incorporar les restriccions `check` a la BD doncs s'acostumen a programar com a restriccions (`@api.constrains`) 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`. Hi afegeix el prefix `nom_taula` per distingir mateix nom de restricció en diverses taules, doncs no poden conviure en una BD índexs amb mateix nom.

Atenció!

- Pels camps traduïbles (`jsonb`), definir una restricció d'unicitat és possible però problemàtic, doncs la comprovació d'unicitat es fa a nivell de tot el camp `jsonb`. Per exemple, observar els 2 valors següents corresponents a un camp:


```
{"ca_ES": "Computació", "en_US": "Computació", "es_ES": "computación"}
{"ca_ES": "Computació", "en_US": "Computació", "es_ES": "Computación"}
```

 Per Odoo no hi ha repetició, doncs certament són diferents, però en català i anglès són iguals...
- 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`) fins que es deixi d'infringir
- En cas de canviar una restricció, Odoo podria no eliminar (depèn de versió d'Odoo) la restricció existent anteriorment a la taula (millor comprovar-ho) i caldria eliminar-la manualment via:


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

Qüestió: El mecanisme `_sql_constraints` amb `unique` permet garantir unicitat en el camp `name` de cursos, assignatures, temàtiques i edicions i en el camp `tin` del professorat?

No, per què són camps alfanumèrics i la constraint `unique(camp)` és case insensitive. Llavors???

• 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 versió 17.2) no permet aquest tipus de `constraint`.

En canvi, PostgreSQL Sí permet crear un índex `UNIQUE` usant `lower(camp)` o `upper(camp)` en la instrucció de creació d'índex:

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

Sembla una incoherència de PostgreSQL, doncs permet usar les funcions `upper` i `lower` en la creació d'índexs i, en canvi, no en permet l'ús en `constraint unique`.

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

❖ Mecanisme 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 pot ser molt costosa en accedir a TOTS els registres de la taula.

❖ Mecanisme 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()
```

Podríem aplicar la tècnica `onchange + _sql_constraints` en camp `tin` de la classe `SchoolTeacher` (sempre en majúscules) i en camps `name` de cursos, assignatures, temàtiques i edicions on hem incorporat el mètode `onchange`, 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 textuales que no verifiquin unicitat *case insensitive*.

❖ Mecanisme 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. Per aconseguir-ho, cal sobreescrivre el mètode `_auto_init` (que s'executa en actualitzar el mòdul) amb el següent contingut, on s'invoca la utilitat `create_unique_index` que Odoo facilita en el seu mòdul `tools`, que caldrà importar:

```
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

Atenció! PostgreSQL12 no permet aplicar la funció `lower` sobre camps `jsonb`. Per tant, aquest mecanisme no és factible en camps traduïbles.

Podem aplicar aquest mecanisme en el camp `tin` del professorat i en el camp `name` d'edicions, que no són traduïbles i en el camp `name` de cursos, assignatures i temàtiques podríem usar el mecanisme 2.

En usar el mecanisme 3, 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:

Error de validació

```
duplicate key value violates unique constraint "school_unique_lower_name"  
DETAIL: Key (lower(name::text))=(sistemes) already exists.
```

El missatge no és prou clar, oi... Com que no s'ha programat amb una `_sql_constraints` on es pot introduir el missatge en cas d'incompliment, Odoo només pot mostrar el missatge que rep de 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 desplegable 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 varies 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 desplegable 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.

En una vista (codi XML):

- El `domain` cal escriure'l entre dobles cometes: `domain=" [...]"`
- L'operador `&` cal escriure'l escapat: `&`

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'acció que invoca la vista, l'element `context` amb el següent contingut:

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

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

Aquest context en acció afecta els `active` a nivell de totes les vistes invocades per l'acció. 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 | En Odoo 15-17 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/17.0/developer/tutorials/backend.html?highlight=decoration%20success#id3>

+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'acció) i distingeix via color els actius (verd) dels arxivats (vermell). Però això implica que el camp `active` formi part de la vista i l'hem incorporat amb atribut `column_invisible="1"` per a que no sigui visible.



Veure vista `form` de classe `SchoolTeacher`, en graella que mostra els cursos assignats, hauria de distingir via color, 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 a la vista.

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)

Exercicis pre-parcial (pràctica 2 i versió 10 d'school)

- Pràctica 2 d'accés a SGBDR corporatius des de BD ofimàtiques: Veure enunciat en PDF específic
- Evolucionar el mòdul `school` a la versió 10 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) Introduir la nova classe `Teaching` per gestionar la docència dels professors en les diverses assignatures per cada edició de curs, seguint el disseny de `10_school_UML.pdf` i `10_school_UML_Odoo.pdf` adjunts.
 - 5) Incorporar una opció de menú `Teaching`, entre `Teachers` i `Configuration` per poder gestionar la docència.

Solució en 10_school:

Comentaris a la solució:

- Dins la classe `SchoolCourseEdition` apareixen les clàusules `_sql_constraints` i `_order` per aconseguir els requeriments 1-2 anteriors.
- Pel requeriment 3, ha estat necessari afegir dins la classe `CourseSubject` el camp `subject_hours` (related via `subject_id`) i retocar la vista `school_course_form`.
- Pels requeriments 4 i 5, s'ha creat la classe `SchoolTeaching` amb una vista `tree` editable (només `tree`) i la corresponent opció de menú, ubicada on ens demanen amb l'ajut de `sequence`.

No es necessària una vista `form`, ja que només tenim 3 camps a editar. Respecte la vista `tree`:



- 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 definir el camp `display_name` 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 camp `display_name` 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 interressi que aparegui el nom del curs. La bona opció és definir el camp `display_name` 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 afecta 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 (Odoo 16-)

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.
- En
- 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 10 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 seleccionada

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` (comprobat necessari en Odoo13+; versions anteriors???)

Atributs `readonly`-`invisible`-`required` en vistes (Odoo 17+)

- Els elements `field` de les vistes poden incorporar els atributs `readonly`, `invisible` i `required` que poden contenir:
 - Valor `"0"` o `"False"` per desactivar-los
 - Valor `"1"` o `"True"` per activar-los
 - Valor resultant d'una condició Python amb resultat booleà.
- En vistes `tree`, per fer invisible una columna cal usar `column_invisible` amb valor `"1"`.
- 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` o `column_invisible` activats, 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.

Exemple de `attrs` en vistes: retocs finals en la vista `view_teaching_tree` en la versió 10 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 seleccionada

Per aconseguir el funcionament desitjat:

- El selector d'assignatura hauria d'estar "inactiu" en cas que no hi hagués curs seleccionat.
`readonly="edition_id==False"`
- El selector d'edició hauria d'estar "inactiu" en cas que ja hi hagi assignatura seleccionada.
`readonly="subject_id!=False"`
- El selector de professor hauria d'estar "inactiu" en cas que no hi hagi assignatura seleccionada.
`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:
`readonly="edition_id==False or 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.

Vistes `calendar`

Info: https://www.odoo.com/documentation/17.0/developer/reference/user_interface/view_architectures.html#calendar

Es pot usar camps `Date` i camps `Datetime` i en aquest darrer cas mostra l'hora de l'esdeveniment.

Segons la documentació, `date_delay` es mesura en dies, però en realitat espera hores (Odoo 16+).

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 11 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!

En Odoo16+, si s'usa `date_stop`, no mostra els objectes que no contenen valor en el camp indicat. Per solucionar-ho, podem introduir un camp calculat (`q_days`) que, pels objectes que tinguin dates inici i final en calculi els dies transcorreguts i que pels que no tinguin data final els calculi 1 i podem usar aquest camp en `date_delay` (eliminant `date_stop`)... tenint en compte que haurem de multiplicar per 24 per què `date_delay` (Odoo 16+) espera hores. La versió 11 incorpora camp `q_days` com a exemple.

Una vista `calendar` permet, per defecte, crear, eliminar i modificar els esdeveniments. Segons la documentació, es pot desactivar amb atributs `create`, `delete` i `edit` a valor 0, però només funciona en `create` i `delete`. Llavors, pel cas que l'usuari vulgui editar l'esdeveniment, Odoo mostra la vista `form` de la vista i si no n'hi ha cap, en crea una per defecte; ergo cal incorporar una vista `form` adequada.

Exercici

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 celebra 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 celebra en el dia.
 - Dissenyar la vista `calendar` a partir del camp `data` creat mostrant l'edat que celebra.
- 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

Solució dins versió 11 del mòdul `school`:

- S'ha creat els camps calculats `birthday` i `birthday_txt` a la classe `SchoolTeacher`.
- S'ha creat la vista `calendar` a `SchoolTeacher`.

Vistes `graph`

Info: https://www.odoo.com/documentation/17.0/developer/reference/user_interface/view_architectures.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 11 de mòdul `school`.

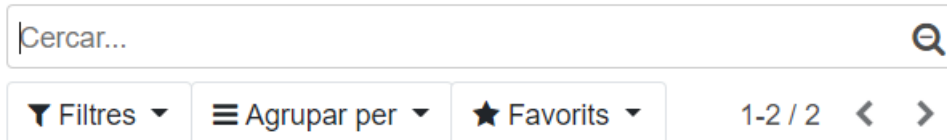
Observacions:

- 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 diversos professorats 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.

Vistes search

Odoo incorpora, per a totes les classes, la vista `search` similar a:

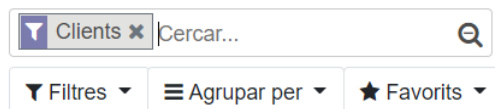


usable en vistes no form (tree, graph, calendar,...) amb els funcionaments per defecte:

- El camp de text (cercar) permet efectuar cerques pel contingut del camp indicat a la clàusula `rec_name` (si existeix i és a la BD) o del camp `name` (si existeix). Si no hi ha `name/rec_name...` ☹
Error en Odoo 16+: En cas de camp a `rec_name` no emmagatzemat a la BD, Odoo mostra la possibilitat d'efectuar la cerca, però no filtra res.
- L'apartat *filtres* permet que l'usuari defineixi filtres personalitzats
- L'apartat *agrupar per* permet que l'usuari defineixi grups personalitzats
- L'apartat *favorits* permet, entre altres, que l'usuari guardi la cerca actual.

Odoo permet definir vistes `search` més sofisticades i llavors desapareix el funcionament per defecte anterior.

Observem la vista `search` de qualsevol de les classes dels mòduls que incorpora Odoo, per exemple, la vista `search` en les vistes dels clients.



Recordem que la classe és `res.partner`, que incorpora clients i proveïdors. Fixem-nos que d'entrada incorpora el filtre `Clients`.

Dins “*filtres*” i “*agrupar per*” hi ha un munt d'opcions, amb separadors delimitadors.

Codi de la vista `search` que permet tota aquesta funcionalitat? Podeu veure quin és el codi d'aquesta vista, amb el mode desenvolupador activat, obrint qualsevol vista de clients que mostri la zona de cerca (totes excepte les vistes `form`) i via apartat “eines del desenvolupador”, editar la vista de cerca (poseu atenció en no modificar res que no vulgueu modificar!!!).

Info: https://www.odoo.com/documentation/17.0/developer/reference/user_interface/view_architectures.html#search

L'arquitectura d'una vista `search` és similar a la resta de vistes i acostuma a conviure amb elles:

```
<record model="ir.ui.view" id="view_....">
  <field name="name">...</field>
  <field name="model">nomModel</field>
  <field name="arch" type="xml">
    <search string="títol">
      <!-- elements field / filter / group / separator -->
    </search>
  </field>
</record>
```

- Element `field` per definir camps sobre els que es pot efectuar la cerca, a més del camp `name`

```
<field name="nomCamp" string="títolQueMostrarà".../>
```

- `name` és l'únic atribut obligatori
- La documentació explica la resta d'atributs que es pot incorporar. Especialment interessants són:
 - `operator`, que permet sobreescriure l'operador per defecte que actua a la cerca.
 - `filter_domain`, per completar el domini de cerca.

- Element `filter` per definir filtres (simples o agrupats)

```
<filter name="..." string="titolQueMostrarà" domain="condicioFiltre" ... />
```

- `name` és atribut obligatori; és un nom del filtre que es pot utilitzar per activar el filtre per defecte (més avall en parlem)
- `domain` serveix per indicar què ha de filtrar el filtre. No és obligatori per què el filtre pot funcionar en base a altres atributs, però és clar que d'alguna manera o altra cal definir què filtrar.
- La documentació explica la resta d'atributs que es pot incorporar. Especialment interessant és:
 - `context`, que és un diccionari que permet definir maneres de generar el domini de cerca. Concretament, si utilitzem la clau `group_by`, el filtre apareixerà en el menú *agrupar per*. El valor de `group_by` pot ser un camp de la classe o una llista de camps
- Element `separator`, per separar filtres i/o grups dins la vista.
- Element `group` per separar grups de filtres, més entenedor que `separator`, en vistes `search` complexes. No té res a veure amb l'apartat *agrupar per*.

Per activar una vista `search` en una `action`, en cas que hi hagi vàries vistes `search` sobre el mateix model, cal introduir-hi:

```
<field name="search_view_id" ref="idVistaSearch"/>
```

A més, es pot activar la vista `search` amb un filtre inicial, per camps i/o filtres incorporats a la vista `search`, incorporant a la `action`:

```
<field name="context">{'search_default_nomCamp_o_nomFiltre':valor1,  
                        'search_default_nomCamp_o_nomFiltre':valor2,...}</field>
```

- Aquí es veu la necessitat de l'atribut `name` en un `filter`.
- En el cas de `nomFiltre`, el valor ha de ser el resultat que s'espera de la corresponent condició.

Exemple: Incorporar una vista `search` per al professorat, de manera que:

- Es pugui cercar pels camps `last_name`, `tin` i `birthdate`.
- Hi hagi filtre per `gender`: Male/Female
- Hi hagi filtre per `salary`: >=3000/<3000
- Hi hagi filtre per `arxivament`: Active/Archived
- Es pugui agrupar per `arxivament`
- Es pugui agrupar per `gender`.

Solució: Vista `school_teacher_graph` en versió 11 de mòdul `school`.

Controlador: Mètodes `search/read`

Info: <https://www.odoo.com/documentation/17.0/developer/reference/backend/orm.html#search-read>

Odoo facilita mètodes per obtenir registres que compleixin una determinada condició (`domain`). Entre ells, ens cal conèixer els que ja fa vàries versions d'Odoo que existeixen (ara n'hi ha molts més):

- `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 de registres que compleixin una determinada condició. Mireu sintaxis en documentació d'Odoo.

En Odoo 16- contemplava paràmetre `count` que podia 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 de `search` però que només compte els registres.
- En Odoo 16- equivalia a `search` amb `count=True`
En Odoo 17+ el mètode `search` ja no té el paràmetre `count` per què ja hi ha aquest mètode.
- ✓ 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 utilitzar el mètode `search` amb el paràmetre `count` o el mètode `search_count`.

Controlador: Sobreescritura 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.

Per sobreescriure el mètode `unlink`:

```
def unlink(self):
    # self conté la llista dels registres a eliminar
    ... # Aquí posarem les comprovacions que calgui efectuar abans d'eliminar
    ... # Si algun registre a eliminar no pot ser eliminat, s'avorta tot
    # La sobreescritura d'aquest mètode, com qualsevol sobreescritura en POO,
    # elimina l'execució del mètode heretat... Per tant, quan ja estem segurs que es
    # pot procedir a l'eliminació, cal invocar el mètode de la classe base, fent:
    q = super(nomClassePython,self).unlink()
    # En Python3 és equivalent invocar super() sense arguments
    ... # Aquí posarem accions que calgui efectuar una vegada eliminat, si n'hi ha
    # El mètode unlink ha de retornar la qtat de registres eliminats.
    # Per això recuperem la qtat de registres eliminats i la retornem:
    return q
```

Alerta! És `nomClassePython` (per exemple `SchoolTeacher`) i no pas `classeOdoo` (`school.teacher`).



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 13 del mòdul `school`. Observeu les diverses possibilitats d'actuació... i de situacions problemàtiques... comentades dins el codi.

Controlador: Sobreescritura 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
    # Per procedir a la inserció, fem:
    r = super(nomClassePython, self).create(values) # Aquí efectuem la creació
    # En Python3 és equivalent invocar super() sense arguments
    # 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, si n'hi ha
    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: Sobreescritura 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(nomClassePython, self).write(values) # Aquí efectuem la modificació
    # En Python3 és equivalent invocar super() sense arguments
    # r un booleà indicant si l'operació s'ha pogut efectuar
    ... # Aquí posarem accions que calgui efectuar una vegada modificats, si n'hi ha
    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.

Exercicis:

- Assegurar que el camp `tin` en `Teacher` sempre tingui el format `upper()` i que el camp `name` en `CourseEdition` sempre tingui el format `title()`. No són camps traduïbles!
- `capitalize()` en el camp traduïble `name` de `SchoolCourse`, `SchoolSubject` i `SchoolThematic`.



Solució en la versió 14 del mòdul `school`:

- Per apartat 1, veure els mètodes `create` i `write` de la classe `SchoolTeacher` (camp `tin`). El mètode `_onchange_tin` no té necessitat d'existir i per això l'hem deixat comentat. I el mateix caldria fer per la classe `SchoolCourseEdition` (camp `name`).
- Per apartat 2, on el camp `name` és traduïble, segurament interessa que `capitalize()` s'apliqui en el valor que l'usuari incorpora en el formulari així com en les traduccions introduïdes en el camp.
 - En la creació d'un registre, no tenim problema, doncs Odoo omple totes les traduccions dels camps traduïbles amb el valor que hagi introduït l'usuari, al qual s'ha aplicat el mètode `capitalize()`.
 - En introduir les traduccions via la finestra *popup* que facilita Odoo, comencen a aparèixer els problemes, doncs el mètode `write` només s'aplica sobre el valor del camp corresponent a l'idioma de l'usuari. El funcionament sembla correcte mentre no s'hagi introduït cap traducció, cas en que Odoo mostra el valor en l'idioma de l'usuari per a tots els idiomes, però quan es comença a introduir traduccions, podem comprovar que el diccionari `values` del mètode `write` només conté, pel camp traduïble, el valor corresponent a l'idioma de l'usuari. Com incidir en els valors pels altres idiomes?

En Odoo15- existia la classe `IrTranslation` encarregada de les traduccions, que quedaven emmagatzemades en taula `ir_translation`. Per aconseguir el nostre propòsit hauríem de sobreescrivir el mètode `write` en una herència de la classe `IrTranslation` en el nostre mòdul (cal conèixer herència de classe, més endavant).

En Odoo16+ no existeix la classe `IrTranslation` i Odoo, amb algun mecanisme intern, s'encarrega de recuperar les traduccions incorporades en el `popup` i, en prémer el botó *Desar* del `popup`, executa el mètode `write`, enregistrant tots els canvis però si es comprova el diccionari, es veu que `values['campTraduïble']` només conté el valor en l'idioma de l'usuari. Com aconseguir veure el valor en els altres idiomes actius a l'empresa i modificar-los? Una vegada enregistrats els valors amb el `write`, el paràmetre `self` conté la llista dels registres ja modificats. En cas que s'hagi modificat el camp traduïble, cal fer un recorregut pels valors ja modificats del `self` i veure la manera d'obtenir el valor del camp en els altres idiomes actius a l'empresa, per poder-hi actuar.

Pistes:

- ✓ Com esbrinar els idiomes actius en una empresa:
`langs = self.env['res.lang'].search([('active', '=', True)])`
`langs` és una llista amb els idiomes; cada idioma té el codi en el camp `code`.
- ✓ Com esbrinar l'idioma de l'usuari actiu:
`self.env['res.users'].browse(self.env.uid).lang`
 Aquesta instrucció retorna el codi (`code`) de l'idioma de l'usuari (`self.env.uid`)
- ✓ Com obtenir el valor d'un camp traduïble d'un registre `r` en idioma diferent del de l'usuari:
`r.with_context(lang=codiIdioma).nomCampTraduïble`
- ✓ Com modificar el valor de camp traduïble d'un registre `r` en idioma diferent del de l'usuari:
`r.with_context(lang=codiIdioma).write({'nomCampTraduïble':nouValor})`

Amb la informació anterior estem en condicions de sobreescrivir el mètode `write` de manera que compleixi les nostres expectatives.

Veure els mètodes `write` i `_write_aux` en la classe `SchoolCourse`.

Hem usat el mètode `_write_aux` per aconseguir que el recorregut per les traduccions per actuar-hi amb el `capitalize()` només s'efectuï una vegada.

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`)

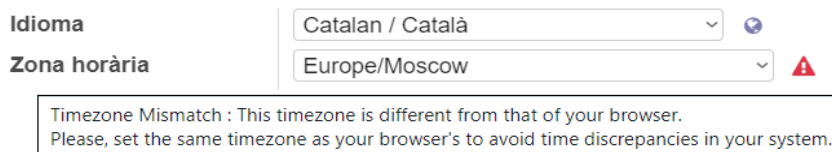
Però Odoo està dissenyat per què tots els valors `datetime` siguin transformats al valor en el fus horari UTC (*Coordinated Universal Time* - fus horari situat sobre el meridià de Greenwich) i:

- 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, via codi 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:



The screenshot shows the 'Idioma' (Language) and 'Zona horària' (Timezone) settings. The language is set to 'Catalan / Català' and the timezone is 'Europe/Moscow'. A warning box below states: 'Timezone Mismatch : This timezone is different from that of your browser. Please, set the same timezone as your browser's to avoid time discrepancies in your system.'

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. Tenim dues opcions (tenint en compte que no és editable des de cap formulari):

- Cadena, guardant el valor en format `HH:MM`
- Float, guardant el valor en format centesimal (18,5 per les 18:30) i usant el giny `float_time` que visualitzarà el valor en format `HH:MM` (18:30).

Per esbrinar la data i hora locals, podem usar:

```
momentActual = fields.Datetime.context_timestamp(self,datetime.now())
```

doncs Odoo, en invocar `datetime.now()` facilita l'hora en UTC, cosa que no desitgem i a partir de `momentActual`, aplicant mètode `strftime` podem obtenir data, hora, ... i emplenar els camps `date` i `time`.

- 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`

Solució en la versió 15 del mòdul `school`:

Respecte exercici 1:

- Sobreescritura 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`.
- Sobreescrivre mètode `create` a la classe `Teacher`, per enregistrar el salari inicial del professor.
- Sobreescrivre 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, seria convenient sobreescrivre el mètode `unlink` a la nova classe, per si algun altre mètode intenta eliminar els registres. Això no afecta el `ondelete='cascade'` incorporat en el camp `teacher_id`.

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/17.0/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/17.0/developer/reference/user_interface/view_records.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 3

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/17.0/es/developer/reference/backend/data.html#data-files>

Exemple: `16_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 nouupdate="1"> ...dades... </data>
```

 Si hi ha dades a reinstal·lar a cada actualització, han d'estar fora del node `<data nouupdate="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 interressi 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 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)
```

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 creen 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`.
 - 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"/>`

En el 2n cas, el fitxer imatge ha de residir en alguna carpeta dins el mòdul i la ruta s'ha d'indicar des de la carpeta del mòdul; en el nostre cas, per exemple: `school/data/images/...`

La versió 16 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 16 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 16 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 202...	Computer systems	Peres Peres, Maria
Cross-platform application development - Edition 202...	Access to data	Peres Peres, Maria
Cross-platform application development - Edition 202...	Web development in client environment	Rodriguez Rodriguez, Josep

Solució: Versió 17 del mòdul `school`, que incorpora els 3 registres de la classe `school.teaching` al final del fitxer `school_demo.xml`.

⇒ **Ja es pot desenvolupar l'exercici 1 de la pràctica 4.**

Informes QWeb

QWeb és una eina que permet combinar plantilles amb dades per obtenir documents finals.

En Odoo es pot utilitzar aquesta eina en el disseny de les vistes (no ho hem fet) i en el disseny d'informes (no tenim altra possibilitat dins Odoo).

Info de QWeb:

<https://www.odoo.com/documentation/17.0/developer/reference/frontend/qweb.html#qweb-templates>

Tutorial complet de QWeb: <https://youtu.be/YkE9YqB5VPA>

Info de QWeb Reports:

<https://www.odoo.com/documentation/17.0/developer/reference/backend/reports.html#qweb-reports>

Tutorial introductori de QWeb Reporting: <https://youtu.be/zWxmPbfb2t8>

Tutorial sobre com crear i/modificar un QWeb Report: <https://youtu.be/-nkx3pbFvLo>

Passos per aconseguir un informe QWeb:

- 1) Definir l'informe en un fitxer `.xml`, que acostuma a residir dins la carpeta `report` i aquest fitxer (amb la ruta `report/...`), incorporar-lo a l'apartat `data` de `__manifest__.py`.

Cada informe es defineix en una `ir.actions.report` i invoca una plantilla, que pot estar en el mateix fitxer o en un altre, cas en el que cal incorporar-lo dins l'apartat `data` de `__manifest__.py`.

- 2) Definir la plantilla en què es basa l'informe.

Revisem els exemples en la versió 18 del mòdul `school`:

- Fitxer `school_report_qweb.xml` conté la definició de cada informe amb la seva plantilla (podria estar en un altre fitxer).
- Informe senzill "Llistat de cursos": `qweb_report_course`, que invoca plantilla `report_course_template`.
- Aquesta plantilla conté una taula HTML amb els títols de les columnes a la capçalera i amb el contingut de cada curs en una fila. Això s'aconsegueix amb `t-foreach="docs"`, on `docs` és el mot que cal utilitzar per referir-se a cada registre Odoo pel que es demana la impressió.
- Observar que dins la pàgina, a la part superior dreta, s'incorpora la data/hora del moment d'impressió invocant la funció Python: `datetime.datetime.now()`.

Aquesta crida imprimeix data-hora tal i com ho retorna Python. Per aconseguir que es vegi segons el format de l'idioma que té actiu l'usuari que invoca l'informe, cal acompanyar-lo de l'atribut:

`t-options="{ 'widget': 'datetime' }"`, si es vol data-hora

`t-options="{ 'widget': 'date' }"`, si només es vol data

- La paginació ja ve incorporada en la plantilla estàndard dels informes externs, on segurament també es podria incorporar la data-hora d'impressió i no caldria afegir-ho a cada informe.
- Veure, dins el codi, els comentaris referents a tipus d'informe (pdf/html) i com aconseguir la negreta.
- Informe mestre-detall "Fitxa de professor": `qweb_report_teacher_card`, que invoca plantilla `report_teacher_card_template`.
- En aquest cas, per cada `teacher` es vol una pàgina diferent. Per tant, el `t-foreach="docs"` ha d'estar per fora de la marca `<div class="page">`.
- La capçalera és una taula on cada cel·la conté els diversos camps. Destinem una cel·la per l'etiqueta i una cel·la per al valor del camp, utilitzant `colspan` per indicar l'amplada que ha d'ocupar cada cel·la. Així, per exemple, podríem confeccionar una capçalera com:

Name:	o.full_name			TIN:	o.tin
Birthdate:	o.birthdate	Age	o.age	eMail:	o.email
Gender:	o.gender			Phone:	o.phone
Citizenship:	o.country_id			Salary:	o.salary



Per aconseguir aquesta taula, cap pensar en una taula amb 6 cel·les per fila, i les cel·les de camps `name`, `gender` i `country_id` amb `colspan="3"`.

Però... I si es vol incorporar la imatge a la dreta... de manera que utilitzi +/- l'altura de les 4 files?

Cal pensar en una columna més... Per tant, la taula passaria a ser de 7 cel·les per fila i a la setena cel·la de la primera fila li apliquem un `rowspan="4"` de manera que combini les 4 cel·les de la novena columna en una sola cel·la i allà hi posem el camp imatge, seguint la nomenclatura:

```

```

Si volem que la imatge tingui per altura +/- la suma de les altures de les 4 files, haurem de "tantejar" i assignar-li una altura. Per exemple:

```

```

Si no existeix la imatge, la cel·la no ocuparà espai, però fa un efecte estrany... Per solucionar-ho, via un `<t t-if="o.photo">` es pot controlar que la cel·la existeixi només si hi ha imatge.

Hi ha molt poca documentació (o no l'he sabut trobar) sobre les classes predefinides dins QWeb per formatar les taules i la resta d'elements.

- La zona de línies és una altra taula, on els registres sorgeixen a partir d'un recorregut pel camp `One2many` o `Many2many`.

• Compte a l'hora de traduir

Odoo mostra l'informe en el llenguatge de l'usuari. Per a que això sigui possible, cal que els seus textos hagin estat traduïts seguint el mecanisme de traducció d'Odoo, que veurem més endavant.

Els textos que dins la plantilla QWeb s'hagin introduït entre etiquetes HTML com ``, no són traduïts directament i en el fitxer de traduccions veurem els textos `text` que caldrà traduir-los mantenint les etiquetes HTML.

En <https://www.odoo.com/documentation/17.0/developer/reference/backend/reports.html#qweb-reports> s'explica com es pot aconseguir que un informe es visualitzi en un idioma diferent al de l'usuari.

• Altres eines de reporting

Quan Odoo era OpenERP, l'eina de reporting que aportava OpenERP tenia moltes mancances i l'empresa [NaN-tic](#), que distribuïa OpenERP, va elaborar el mòdul `jasper_reports` que permet generar informes amb Jasper.

NaN-tic ja no distribueix Odoo i altres col·laboradors/distribuïdors d'Odoo se n'han fet càrrec. En el moment d'escriure aquest document, la versió més recent del mòdul és de [JayVora-SerpentCS](#) i es pot descarregar a https://github.com/JayVora-SerpentCS/Jasperreports_odoo.

Avantatge respecte QWeb:

Fàcil d'usar per desenvolupadors que ja coneixin els informes Jasper

Inconvenient:

El mòdul JasperReport necessita anar evolucionant per a que vagi funcionant en les diverses versions d'Odoo i no hi ha garantia que algú se n'encarregui. Però amb una mica de paciència es pot aconseguir el funcionament en les noves versions.

Exercici

Desenvolupar un informe QWeb 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.

Per visualitzar bé un camp amb contingut `html` en informe QWeb, usar `t-raw` enlloc de `t-field`.

Solució: Versió 19 del mòdul `school`, que incorpora l'informe en fitxer `exercici_report_qweb.xml`.

⇒ **Ja es pot desenvolupar l'exercici 2 de la pràctica 4.**

Definició de l'esquema de seguretat

Informació: Apartat 1.2 del dossier [DAM M10 UF2 B2](#).

Exemple: `20_school.zip`

- L'esquema de seguretat només s'hauria d'instal·lar en instal·lar el mòdul (opció `-i nomModul`), cosa que s'aconsegueix incorporant `<data noupdate="1">`.
- Carpeta `security` conté arxius amb l'esquema de seguretat:
 - Fitxer(s), habitualment XML, per:
 - Definir els grups de privilegis, englobats en una categoria definida prèviament en el mateix mòdul o en un altre mòdul que ha d'estar instal·lat prèviament (apartat `depends` dins `__manifest__.py`)
 - Assignar -si es creu oportú- l'usuari administrador (`base.user_admin`) i l'usuari OdooBoot (`base.user_root`) a un grup (habitualment `manager`), de manera que quan s'instal·li el mòdul, l'usuari administrador ja tingui algun tipus d'accés (habitualment total) sobre el mòdul.
 - Fitxer(s), habitualment CSV, per assignar els privilegis a cada grup. S'aconsella gestionar-lo amb LibreOffice (format UTF-8 amb la coma com a separador i sense delimitador). S'acostuma a anomenar `ir.model.access.csv`.
- Cal incorporar aquests arxius a l'apartat `data` de `__manifest__.py`. Cal tenir cura en l'ordre d'introducció dins `data`:
 - El fitxer que conté la definició dels grups ha d'anar ubicat abans que el fitxer que conté l'assignació dels privilegis de cada grup.
 - En la definició de la vista del mòdul (vistes, accions, menús) és possible indicar quin grup/grups hi tenen accés (no ho havíem fet encara). Si s'usa aquesta funcionalitat, dins `data`, el fitxer `_views` hauria d'anar ubicat després del fitxer que defineix els grups.

La versió 20 del mòdul `School` incorpora:

- 3 grups/rols: `Manager`, `Teacher`, `Nothing` i mostra com `Manager` podria hereta dels altres rols.
- Privilegis totals per `Manager` sobre totes les classes, excepte sobre la història dels salaris del professorat on només ha de tenir privilegi de consulta.
- Privilegis de lectura per `Teacher` sobre totes les classes, excepte sobre la història dels salaris del professorat, on no ha de tenir cap privilegi.
- Cap privilegi per `Nothing` (per això hem posat aquest nom)
- El camp `salary` de les vistes `tree` i `form` de `school.teacher` incorporen l'atribut `groups` per limitar que el salari només sigui visible pels usuaris amb rol `manager`.
- La vista `school_teacher_graph` que mostra els salaris, també hauria de ser només visible pels usuaris amb rol `manager`. Segons documentació, en aquesta vista es pot incorporar: `<field name="groups_id" eval="[(6,0,[ref('school.group_school_manager')])]" />` per indicar que només la poden visualitzar els usuaris del rol `manager`. En Odoo16+ no funciona. De fet, podem comprovar via l'apartat *Grups* de l'Odoo, d'assignar vistes a un rol en concret i la resta de rols continuen veient-la. La solució per aconseguir l'objectiu és:
 - Crear una `action` i un `menuitem` que invoquin totes les vistes i assignar-lo al rol `manager`.
 - Crear una `action` i un `menuitem` que invoquin les vistes que calgui i assignar-lo al rol `teacher`.

És la solució adoptada en la versió 20 del mòdul.

Atenció!!! En utilitzar `groups` dins un XML hem de tenir en compte que dins l'apartat data de `__manifest__.py`, aquest XML estigui ubicat amb posterioritat a l'XML o CSV on es declaren els grups indicats a `groups`, doncs del contrari, en instal·lar el mòdul, Odoo generarà error indicant que no coneix els grups indicats.

Comprovació:

- Veure que l'usuari administrador té accés a tot.
- Crear un usuari `ttt` amb rol `Teacher` i veure que només pot consultar les dades
- Crear un usuari `nnn` amb rol `Nothing` i veure que no pot veure res.

No té cap sentit tenir un rol com `Nothing`, doncs el mateix s'aconsegueix en qualsevol usuari sense assignar-li un rol. Ha estat, únicament, com exercici acadèmic per comprovar que si un rol no té privilegis assignats, no pot fer res de res.

Respecte el fitxer `csv` on assignar privilegis a cada grup:

- La columna `model_id:id` ha de contenir sempre el nom de la classe substituint `'` per `_` i amb prefix `model_`. Per exemple, per referir a la classe `school.teacher` escriurem `model_school_teacher`.
- En cas d'haver de fer referència a una classe definida en un altre mòdul, cal indicar el nom del mòdul com a prefix a l'identificador de la classe. Per exemple, si cal fer referència a la classe `school.teacher` definida en mòdul `school`, escriurem: `school.model_school_teacher`.
- En cas d'haver de fer referència a un grup que s'hagi creat en un altre mòdul, cal indicar el nom del mòdul com a prefix a l'identificador del grup. Per exemple, si cal fer referència al rol `group_school_manager` definit en mòdul `school`, escriurem: `school.group_school_manager`.

Compte! A partir d'ara, en incorporar una nova classe en el mòdul hem de pensar en anar actualitzant de manera adequada el sistema de seguretat i, si s'implementa un mòdul que hereti de `School`, caldrà incorporar un esquema de seguretat que completi l'esquema de seguretat del mòdul `School`.

• Com incorporar-usar usuaris demo en un mòdul?

Com aconseguir que en instal·lar el mòdul `School` amb dades demo, l'usuari `demo` d'Odoo tingui assignat el rol `Teacher` pel mòdul `School`? Com crear un altre usuari demo, per exemple, `teacher`, que també tingui assignat el rol `Teacher`?

Solució: Ens estan demanant d'assignar un rol a una dada demo. Per tant, com que és un fet que afecta a dades demo, NO ho podem fer en el `school_security.xml` que genera l'esquema de seguretat, sinó que ho hem de fer en un xml incorporat a les dades demo. Veure fitxer `school_security_demo.xml` que incorporem dins la zona demo en el fitxer `__manifest__.py`.

Respecte el nou usuari, ens podem basar en la creació de l'usuari `base.user_demo` que podem veure dins de `addons/base/data/res_users_demo.xml`. Cal posar atenció que en Odoo un `user` està sempre associat a un `partner` i, per això, cal crear el `partner` abans que `user`.

Problema! Si està instal·lat el mòdul `mail` (que per exemple s'instal·la si el mòdul de `Vendes` està instal·lat), la classe `res.users` conté un camp `notification_type` obligatori (es declara dins el mòdul `mail`) i això provoca que calgui indicar, en crear un nou usuari, el valor per `notification_type`, però si i només si està instal·lat el mòdul `mail`. Per assegurar que no hi hagi problema, podem forçar que el mòdul `school` depengui del mòdul `mail` i així segur que en instal·lar el mòdul `school` tindrem prèviament instal·lat el mòdul `mail`.

⇒ **Ja es pot desenvolupar l'exercici 3 de la pràctica 4.**

Assistents

Un assistent informàtic (*wizard* en anglès) és un programa pensat per abreujar o canalitzar els passos a seguir per dur a terme una tasca o, com a mínim, explica els passos detalladament. En moltes ocasions s'utilitza per guiar als usuaris inexperts.

En Odoo, un assistent és una successió de passos. Cada pas es compon de diverses accions:

1. Mostra un formulari a l'usuari amb alguns botons.
2. Recupera la informació introduïda per l'usuari i el botó seleccionat.
3. Executa les accions que corresponguin
4. Envia una nova acció al client (formulari, informe,...)

Exemple:

Es vol una utilitat que informi quantes edicions hi ha d'un curs que s'hagin iniciat entre un interval de dates determinat.

Solució: Assistent `how_many_editions_between_dates` en la versió 21 del mòdul `school`.

- Normalment, els assistents s'ubiquen en una carpeta `wizard`, tant les vistes com el model en el que es basa l'assistent.
- El model (fitxer `.py`) caldrà indicar-lo dins fitxer `__init__.py` dins carpeta `wizard` i afegir aquesta carpeta en `__init__.py` de l'arrel del mòdul.
- La vista (fitxer `.xml`) caldrà afegir-la a l'apartat `data` de `__manifest__.py` del mòdul.
- El model ha d'incorporar tots els camps necessaris per a la utilitat:
 - Camp per a que l'usuari seleccioni el curs
 - Camps per a que l'usuari introdueixi l'interval de dates
 - Camp per calcular el número d'edicions i mostrar-lo per pantalla.
 - Camp `state` per saber en quin moment del procés ens trobem i poder visualitzar uns camps o uns altres en el formulari. Aquest és un camp especial d'Odoo (com `name` i `active`) i està sincronitzat amb l'atribut `states` que es pot aplicar a camps i botons. És de tipus `selection`.
- El model deriva de `models.TransientModel`.
- El formulari ha de contenir:
 - Per una banda, els camps que l'usuari ha d'emplenar, acompanyats d'un botó per procedir a l'execució i un botó per cancel·lar l'acció.
 - Per altra banda, els camps on cal mostrar el resultat.
 - Es podria fer tots els camps visibles permanentment, però en ocasions, per donar sensació de procediment, es visualitzen en funció del camp `state`.
 - Un botó per procedir a la finalització i tancament de l'assistent.
- Incorporem l'execució del menú en menú `Wizards` submenú del menú `Configuration` (al final).
- El model ha de contenir el mètode a executar, que en aquest cas ha d'efectuar el càlcul de les edicions del curs entre les dues dates i el corresponent percentatge.
- Les classes definides en els assistents, que són `models.TransientModel`, en les primeres versions d'OpenERP, no generaven cap taula a la BD, doncs... quines dades ha de guardar? Des de ja fa unes versions, Odoo genera la corresponent taula (comprovar-ho a la BD) on s'hi guarda les dades amb les que es va executant l'assistent. Evidentment, aquestes dades no tenen cap importància i de les taules corresponents als assistents se'n pot fer neteja sense problema. Pot servir per conèixer qui-quan-com ha executat l'assistent.



- Les classes definides en els assistents, en Odoo14+ (no en Odoo13) estan controlades per l'esquema de seguretat del mòdul i cal completar adequadament el fitxer on s'assigna els permisos a cada rol. I els rols que hi tinguin accés han de tenir privilegis de "creació" sobre la classe (doncs cada execució de l'assistent provoca una creació de registre a la corresponent taula) i de "modificació" (doncs per canviar d'estat, es provoca una modificació de registre a la taula).
- Cal incorporar dins `ir.model.access.csv`, per a la classe `how_many_editions_between_dates`, privilegis totals pel rol `manager` i privilegis `read-create-write` pel rol `teacher`. L'usuari `admin` (que tenim assignat al rol `manager`) pot executar l'assistent sense problema. En canvi un usuari amb rol `teacher`, (per exemple l'usuari de demostració `ttt` creat prèviament) si només tingués `read`, veuria l'assistent, però en executar-lo li petaria per no tenir el permís de creació, i si tingués el privilegi `create` sobre la classe, l'execució petaria per no tenir el permís de modificació, és a dir, cal donar-li permisos `1,1,1,0` sobre la classe.

⇒ **Ja es pot desenvolupar l'exercici 4 de la pràctica 4.**

Models no persistents

Odoo permet dissenyar objectes no persistents, basats en vistes de PostgreSQL, que s'acostumen a utilitzar en el desenvolupament de gràfics, informes estadístics i quadres de comandament.

Exemple:

Interessa mostrar (en una vista o en un informe) el nombre d'edicions iniciades en cada curs per any natural.

Per aconseguir-ho, necessitem tenir una "classe" amb els camps:

- Curs
- Any
- Nombre d'edicions

Entenem que el contingut d'aquesta classe és calculat a partir de continguts en altres classes. Odoo ens permet crear una classe basada en una vista SQL que calculi les dades necessàries a partir de les dades existents a la BD. Evidentment, cal saber SQL i les taules implicades per aconseguir les dades necessàries. Es tracta d'una classe no persistent, que anomenarem `SchoolYearCourseQtyEditions`.

Veure mòdul `22_school` on tenim:

- La classe que podria conviure amb les altres classes persistents, però hi ha tendència a crear un `.py` per cada classe no persistent i així ho hem fet en el mòdul.
- La vista que també podria estar en el mateix `.xml` que la resta, però també la separem. Només calen vistes `tree` i `graph` i amb `delete="0"` i `create="0"` doncs les dades són calculades.
- Una opció de menú per executar-la, incorporada en una nova una opció `Statistics` en el menú principal del mòdul.
- Esquema de seguretat retocat, segons interressi, per permetre l'accés a la nova classe. No s'ha assignat accés al rol `teacher` i es pot veure que un usuari amb aquest rol no hi accedeix.
- I caldria ampliar la traducció del mòdul...

Respecte la classe, primer cal tenir clara la consulta que facilita la informació desitjada. En el nostre cas:

```
select sce.course_id, sc.name, date_part('year',sce.date_start), count(*)
from school_course_edition sce join school_course sc on sce.course_id = sc.id
group by sce.course_id, sc.name, date_part('year',sce.date_start);
```

La classe a definir ha de tenir les següents característiques:

`_auto = False`, per indicar que no s'ha de crear la taula

Columnes:

- Tantes com la vista

- Mateix ordre que la vista
- Nom igual que la columna de la vista => Cal posar "àlies" a les columnes que no són un camp
- Tipus coherent amb la columna de la vista

La consulta SQL ha de contenir com a primera columna, un columna de nom `id` que ha de fer el paper del camp `id` en les taules creades per Odoo.

- Ha de ser de tipus numèric enter
- Ha de garantir que no hi hagi valors repetits
- No cal que contingui valors correlatius

En molts ocasions s'utilitza un dels camps `id` de les taules implicades en la consulta. En el nostre cas:

- No és possible utilitzar `id` de `school_course` per què d'un mateix curs podem tenir diferents anys.
- No és possible utilitzar `id` de `school_course_edition` per què en un mateix curs-any podem tenir diferents edicions.
- Podem utilitzar `min` o `max` de `id` de `school_course_edition`.

```
select min(sce.id) as "id",  
       sc.name as "course_name",  
       date_part('year',sce.date_start) as "year",  
       count(*) as "qty_editions"  
from school_course_edition sce join school_course sc on sce.course_id = sc.id  
group by sce.course_id, sc.name, date_part('year',sce.date_start);
```

Odoo presenta problemes si la vista ja existeix i s'ha de tornar a crear; és a dir, té problemes en fer un `replace` de la `view`. Per solucionar-ho, enlloc de la instrucció `create or replace view`, hi ha una `create view` i prèviament una instrucció per eliminar la vista si existeix:

```
drop_view_if_exists(self.env.cr, self._table)
```

Per a que funcioni, cal haver afegit a l'inici del `.py`:

```
from odoo.tools.sql import drop_view_if_exists
```

Quadres de comandament

Odoo facilita la possibilitat de dissenyar senzills quadres de comandament, ja sigui a nivell d'usuari, personalitzant el seu Odoo (apartat 5.8 del PDF d'UF1), o a nivell d'incorporació en els mòduls per a la seva instal·lació. Aquí ens centrem en com incorporar-los en un mòdul.

Els quadres de comandament d'Odoo es confeccionen integrant qualsevol de les vistes existents a l'empresa, de manera similar als quadres de comandament que pot crear l'usuari, que són una combinació de vistes existents.

No confondre amb les vistes *dashboard* que incorpora la versió Enterprise d'Odoo.

Un quadre de comandament és una vista `form` sobre el model `board.board`, al que cal assignar l'estil:



L'atribut `style` ha de ser coherent amb la quantitat d'elements `column` dins l'element `board`., és a dir, si es defineix 1-1 no podem assignar 3 columnes.

El contingut del `form` ha de ser:

```
<form>  
  <board style="estil">  
    <column>    <!-- Per contenir les actions de la 1a columna -->
```



```
<action string="...títol..."
        name="% (nom_mòdul.nom_acció)d"
        domain="..." <!-- Només si es vol filtrar -->
        view_mode="tipusDeVistaDeLaAccióAVisualitzar"/>
<action...
/>
</column>
<column> <!-- Per contenir les actions de la 2a columna -->
...
</column>
</board>
</form>
```

En versions 13 i 15+, l'apartat `action` pot incorporar atribut `domain` per definir un filtre dels registres a visualitzar dins la classe sobre la que està definida la `action`. En Odoo14, en canvi, hi ha un canvi de funcionament i la incorporació de l'atribut `domain` s'interpreta com un filtre sobre la classe `board.board`, provocant error.

En el cas d'Odoo14 (no és el nostre cas), la solució per poder filtrar registres consisteix en crear prèviament una `action` que incorpori el `domain` per filtrar i dins la vista `board.board` invocar aquesta `action` que conté el filtre.

No es permet indicar `view_mode="tree"` que és el tipus de visualització per defecte (quan no s'indica).

Cal tenir el mòdul `board` instal·lat. Per tant, caldrà incorporar-lo a `depends`.

Com a exemple, considerem que volem un quadre de comandament amb:

- Columna esquerra:
 - Vista `tree` que mostri el professorat
 - Vista `tree` que mostri el professorat femení
 - Vista `tree` que mostri els professorat masculí
- Columna dreta:
 - Vista `tree` que mostri la docència
 - Vista `tree` que mostri la història dels salaris del professorat
 - Vista `graph` que mostra els salaris del professorat

Per fer visible una vista `board`, caldrà actuar com a la resta de vistes: `menuitem => action => view`

L'`action` que invoca la vista `board` ha de seguir:

```
<record model="ir.actions.act_window" id="action_board_teacher">
  <field name="name">...</field>
  <field name="res_model">board.board</field>
  <field name="view_mode">form</field>
  <field name="usage">menu</field>
  <field name="view_id" ref="idVistaBoard"/>
</record>
```

Solució: `23_school`, que incorpora el quadre de comandament dins `school_dashboards.xml`.

- La vista pot estar en el mateix `.xml` que la resta, però l'hem separat en `school_dashboards.xml` pensat per contenir els quadres de comandament.
- Per a que un usuari pugui executar un quadre de comandament, cal que tingui permisos adequats per les classes de les vistes que el quadre de comandament incorpora i, si no té els permisos adequats, Odoo genera un error en executar el quadre.



Es pot comprovar connectant amb usuari `teacher` i intentant executar el tauler dissenyat. Com que aquest usuari no té privilegis per veure la història dels salaris, es produeix un error en executar el tauler. Concretament:

No podeu accedir als registres 'Història dels salaris dels professors' (school.teacher.salary.history).

Aquesta operació es permet a aquests grups:
- School/Manager

En Odoo 14 i Odoo16+ aquest error es visualitza en pantalla. En Odoo 15 no es visualitzava en pantalla, tot i que quedava enregistrat en el log i, simplement, el tauler no es visualitzava.

Evidentment, si és conegut que un quadre de comandament invoca una vista d'una classe X per la que els usuaris amb rol R no tenen permís adequat, és il·lògic que als usuaris de rol R que no podran executar-lo, els aparegui el quadre de comandaments en l'arbre de menús (com en el cas de l'usuari `teacher`, que veu l'opció de menú i en executar-la es troba amb l'error de privilegi). Això es pot fer incorporant en el `menuitem` que invoca el quadre de comandaments, l'atribut `groups` amb la llista de rols que tenen accés al menú:

```
<menuitem ... groups="rol1, rol2,..."/>
```

El projecte `23_school`, tal i com es facilita, permet comprovar que per un usuari `teacher` apareix el menú `Dashboards` amb l'opció `Teachers` i en executar-la no s'observa cap tauler. En el codi XML està comentat com hauria de ser el codi del `menuitem Teachers` de manera que només es dona privilegi als usuaris que tenen el rol `group_school_manager` per a poder-lo executar. Es pot comprovar com ara l'usuari `teacher` no veu l'opció per executar el tauler.

Atenció!!! Si en algun moment, via codi, assignem `groups` a un `menuitem` i posteriorment, via codi, eliminem l'assignació i actualitzem mòdul, l'assignació prèvia haurà quedat activa. Cal eliminar-la manualment via *Configuració | Usuaris i Empreses | Groups* i opció *Menú* del grup en concret.

Atenció!!! En utilitzar `groups` dins un XML hem de tenir en compte que dins l'apartat `data de __manifest__.py`, aquest XML estigui ubicat amb posterioritat a l'XML o CSV on es declaren els grups indicats a `groups`., doncs del contrari, en instal·lar el mòdul, Odoo generarà error indicant que no coneix els grups indicats.

Traducció de mòdul

Informació: Apartat 1.5 del dossier [DAM_M10_UF2_B2](#).

Exemple: `24_school.zip`

- Extracció de dades:
 - En mode desenvolupador, *Configuració | Traduccions | Exportar traducció*, en format PO.
 - Indicar l'idioma pel que es vol efectuar la traducció i el mòdul a traduir.
 - Si el mòdul ja contenia textos traduïts a l'idioma indicat, els incorpora en el fitxer generat.
- Guardar fitxer generat en carpeta `i18n` del mòdul
- Utilitzar programa [poEdit](#) per efectuar la traducció. Hi ha versió de pagament *poEdit Pro* que permet utilitzar Google Traductor.
- Actualitzar mòdul

L'extracció de dades recull:

- Els textos (`label`, `help`) de la definició dels camps de les classes.
- Els textos de `_sql_constraints`
- Els textos dels camps `selection`
- Els textos de la vista (vistes, accions i menús)



Però... I els textos dels missatges dins els mètodes de les classes? Aquest NOMÉS es tradueixen si estan “marcats” per a ser traduïts.

Com marcar els textos dels missatges dels mètodes de les classes?

- A la part superior del fitxer `.py` que conté classes, cal incorporar: `from odoo import _`
- Cada text dins el codi dels mètodes que vulguem que sigui extret per a traducció, cal tancar-lo entre els símbols `_ (i)`, de manera que quedi `_ (' . . . ')`
- Actualitzar mòdul.
- Repetir el procés de traducció, que incorporarà els nous textos pendents de traduir.

Per activar els textos incorporats, cal anar a *Idiomes* i reactivar l'idioma que interressi (darrera columna).

⇒ **Ja es pot desenvolupar l'exercici 5 de la pràctica 4**

Pràctica 4

Veure enunciat en PDF específic.