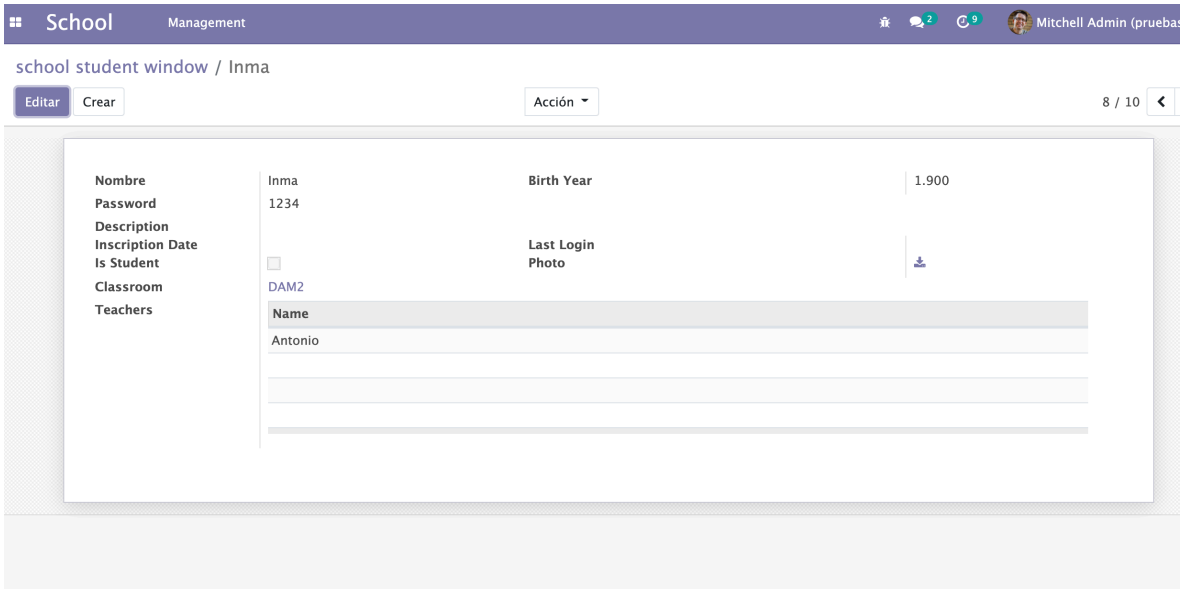
FIELDS CALCULADOS

- Un field calculado o computado es un field normal pero que es calculado sobre la marcha.
- Vamos a añadir al estudiante un campo password y vamos a hacer que siga un campo calculado. No tiene mucho sentido pero va a ser generado como ejemplo.

```python
class student(models.Model):
    _name = 'school.student'
    _description = 'Los alumnos'

    name = fields.Char(string="Nombre", readonly=False, required=True, help='Este es el nombre')
    birth_year = fields.Integer()
    password = fields.Char(compute='_get_password')
    description = fields.Text()
    inscription_date = fields.Date()
    last_login = fields.Datetime()
    is_student = fields.Boolean()
    photo = fields.Image(max_width=300, max_height=300) # Field binario pero específico para imágenes.
    classroom = fields.Many2one('school.classroom', ondelete='set null', help='Clase a la que pertenece')
    teachers = fields.Many2many('school.teacher', related='classroom.teachers', readonly=True)

    # Le puede entrar uno o más listas de estudiantes
    # Self será una lista de estudiantes, si sólo es un estudiante, self será una lista de un estudiante
    def _get_password(self):
        # Para ver qué está ocurriendo, en este caso podemos imprimir en el terminal (aparecerá en el log del
        # servicio)
        print(self)
        # Todas las funciones que calculan campos deben recorrer toda la lista de estudiantes
        # Si solo es un estudiante vendrá un único estudiantes
        for student in self:
            # student es una instancia del modelo student.
            student.password = '1234' #Como ejemplo estamos asignando a todos los usuarios la misma contraseña
            print(student)
```

- Para crear el campo calculado, ponemos el argumento **compute** y la función que se va a encargar de calcularlo
- La función la ponemos con _ previamente para que sea privada. Por defecto recibe una lista de instancias del modelo, en este caso, una lista de estudiantes, aunque tenga uno solo, será una lista con un estudiante. Si quisiésemos que sólo recibiese uno, tendría que tener un decorador **@api.one**
- Si ponemos print, mostraremos por la consola mensajes de debug. En este caso podría ser correcto ya que es una aplicación que por consola muestra determinados mensajes de log, aunque lo más correcto sería volcarlos al fichero de log correspondiente, pero por ahora nos pueden dar información de debug.

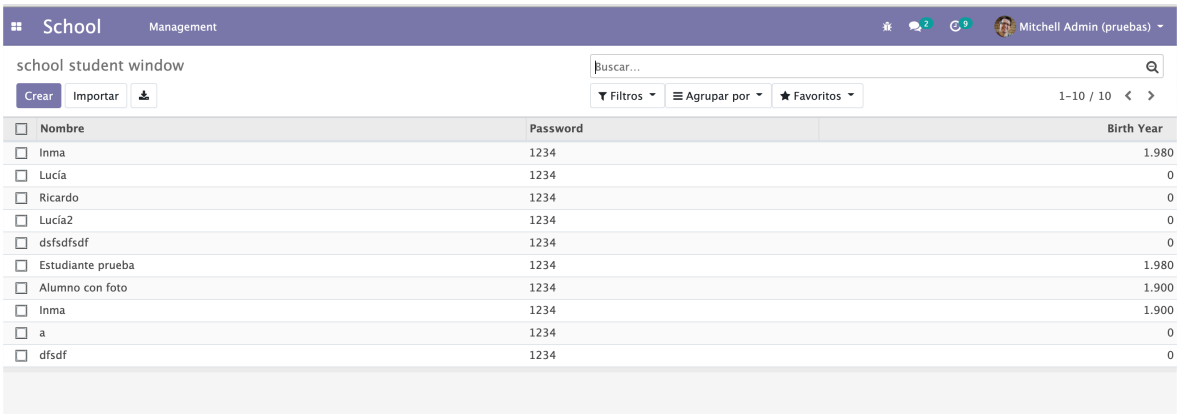
```
2021-02-10 11:32:54,011 1153 INFO pruebas werkzeug: 192.168.203.137
2021-02-10 11:32:55,056 1153 INFO pruebas werkzeug: 192.168.203.137
school.student(1,)
school.student(1,)
2021-02-10 11:32:56,860 1153 INFO pruebas werkzeug: 192.168.203.137
```



- Podemos hacer que en la vista de tree, además de el campo nombre y fecha de nacimiento me muestre el password.

```
<!-- explicit list view definition -->

<record model="ir.ui.view" id="school.student_list">
  <field name="name">school.student list</field>
  <field name="model">school.student</field>
  <field name="arch" type="xml">
    <tree>
      <field name="name"/>
      <field name="password"/>
      <field name="birth_year"/>
    </tree>
  </field>
</record>
```

- Nota: A veces no coge los cambios al modificar las vistas, si ocurre, se puede borrar desde odoo para forzar que se regenere, buscando en **Ajustes > Técnico > Interfaz de usuario > Vistas**, borrando la vista en este caso student list y al reiniciar el servicio se volverá a generar.



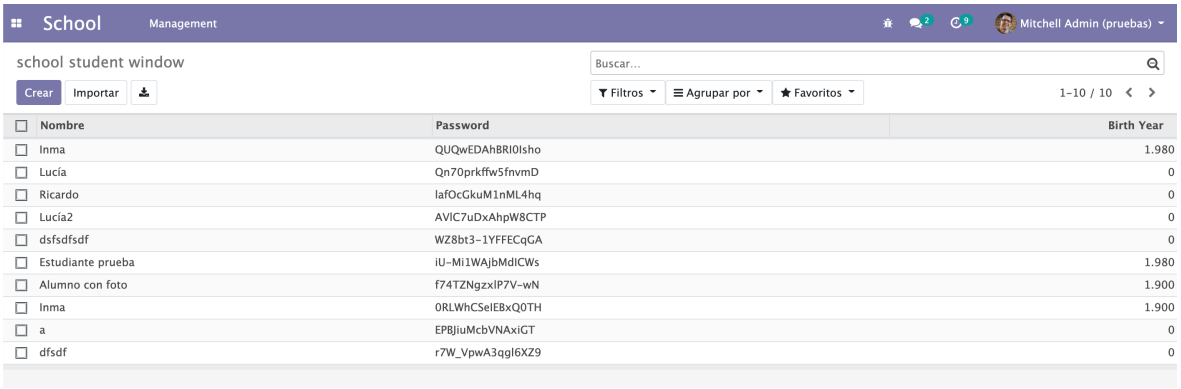
Nombre	Password	Birth Year
Inma	1234	1.980
Lucía	1234	0
Ricardo	1234	0
Lucía2	1234	0
dsfsdfsdf	1234	0
Estudiante prueba	1234	1.980
Alumno con foto	1234	1.900
Inma	1234	1.900
a	1234	0
dfsdf	1234	0

- Vamos a generar la contraseña de forma más segura utilizando funciones de Python. Vamos a importar la librería **secrets**.

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


class student(models.Model):
    _name = 'school.student'
    _description = 'Los alumnos'

    name = fields.Char(string="Nombre", readonly=False, required=True, help='Este es el nombre')
    birth_year = fields.Integer()
    password = fields.Char(compute='_get_password')
    description = fields.Text()
    inscription_date = fields.Date()
    last_login = fields.Datetime()
    is_student = fields.Boolean()
    photo = fields.Image(max_width=300, max_height=300) # Field binario pero específico para imágenes.
    classroom = fields.Many2one('school.classroom', ondelete='set null', help='Clase a la que pertenece')
    teachers = fields.Many2many(                                    eachers', readonly=True)

    # Le puede entrar uno o más listas de estudiantes
    # Self será una lista de estudiantes, si sólo es un estudiante, self será una lista de un estudiante
    def _get_password(self):
        # Para ver qué está ocurriendo, en este caso podemos imprimir en el terminal (aparecerá en el log del
        # servicio)
        print(self)
        # Todas las funciones que calculan campos deben recorrer toda la lista de estudiantes
        # Si solo es un estudiante vendrá un único estudiantes
        for student in self:
            # student es una instancia del modelo student.
            student.password = secrets.test_token_urlsafe(12) # Generará un token de 12 bytes
            print(student)
```

```
models.models
class school.classroom
```

- De este modo genera contraseñas seguras, pero cada vez que reinicio las vuelve a generar. Cada vez que refresco se recalcula, pero es sólo un ejemplo.



Nombre	Password	Birth Year
Inma	QUQwEDAhBRI0Isho	1.980
Lucía	Qn70prkffw5fnvmD	0
Ricardo	lafOcGkuM1nML4hq	0
Lucía2	AVlC7uDxAhpW8CTP	0
dsfsdfsdf	WZ8bt3-1YFFECqGA	0
Estudiante prueba	iU-Mi1WAjbMdICWs	1.980
Alumno con foto	f74TZNgzxlP7V-wN	1.900
Inma	0RLWhCSeIEBxQ0TH	1.900
a	EPBjiuMcbVNAxiGT	0
dfsdf	r7W_VpwA3qgl6XZ9	0

- Para que sólo se calcule una vez hacemos **store=True.** Cuando hacemos esto, al ser un campo calculado le tenemos que decir cuándo se debe calcular, en este caso, con el decorador @api.depends le estamos diciendo que cada vez que se cree o modifique el campo nombre, calcule la password.

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


class student(models.Model):
    _name = 'school.student'
    _description = 'Los alumnos'

    name = fields.Char(string="Nombre", readonly=False, required=True, help='Este es el nombre')
    birth_year = fields.Integer()
    password = fields.Char(compute='_get_password', store=True)
    description = fields.Text()
    inscription_date = fields.Date()
    last_login = fields.Datetime()
    is_student = fields.Boolean()
    photo = fields.Image(max_width=300, max_height=300) # Field binario pero específico para imágenes.
    classroom = fields.Many2one('school.classroom', ondelete='set null', help='Clase a la que pertenece')
    teachers = fields.Many2many('school.teacher', related='classroom.teachers', readonly=True)

    # Le puede entrar uno o más listas de estudiantes
    # Self será una lista de estudiantes, si sólo es un estudiante, self será una lista de un estudiante
    @api.depends('name') #Se calculará cuando cambie o se cree el campo nombre
    def _get_password(self):
        # Para ver qué está ocurriendo, en este caso podemos imprimir en el terminal (aparecerá en el log del
        # servicio)
        print(self)
        # Todas las funciones que calculan campos deben recorrer toda la lista de estudiantes
        # Si solo es un estudiante vendrá un único estudiantes
        for student in self:
            # student es una instancia del modelo student.
            student.password = secrets.token_urlsafe(12) # Generará un token de 12 bytes
            print(student)
```

- Esto es sólo un ejemplo, no sería una solución bien planteada para la contraseña.

- Con esta sintaxis, conseguimos que los mensajes por consola aparezcan de color:

```
    classroom = fields.Many2one('school.classroom', ondelete='set null', help='Clase a la que pertenece')
    teachers = fields.Many2many('school.teacher', related='classroom.teachers', readonly=True)

    # Le puede entrar uno o más listas de estudiantes
    # Self será una lista de estudiantes, si sólo es un estudiante, self será una lista de un estudiante
    @api.depends('name') #Se calculará cuando cambie o se cree el campo nombre
    def _get_password(self):
        # Todas las funciones que calculan campos deben recorrer toda la lista de estudiantes
        # Si solo es un estudiante vendrá un único estudiantes
        for student in self:
            # student es una instancia del modelo student.
            student.password = secrets.token_urlsafe(12) # Generará un token de 12 bytes
            print('\033[94m', student, '\033[0m')

class classroom(models.Model):
```

- El 94 indica el color azul y el 0 vuelve a colocar el color negro. \033 es la secuencia de escape que indica que cambiamos de color.

```
school.student(<NewId 0x7f54485e5280>,)
2021-02-10 12:23:40,094 2400 INFO pruebas werkzeug:
school.student(12,)
2021-02-10 12:23:40,187 2400 INFO pruebas werkzeug:
2021-02-10 12:23:40,201 2400 INFO pruebas werkzeug:
2021-02-10 12:24:22,543 2400 INFO pruebas werkzeug:
```

- Si importamos lo siguiente, podemos lanzar excepciones, warnings, etc:

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

from odoo import models, fields, api
from odoo import _
from odoo.exceptions import Warning
import secrets
```

```python
class student(models.Model):
    _name = 'school.student'
    _description = 'Los alumnos'

    name = fields.Char(string="Nombre", readonly=False, required=True, help='Este es el nombre')
    birth_year = fields.Integer()
    password = fields.Char(compute='_get_password', store=True)
    description = fields.Text()
    inscription_date = fields.Date()
    last_login = fields.Datetime()
    is_student = fields.Boolean()
    photo = fields.Image(max_width=300, max_height=300) # Field binario pero específico para imágenes.
    classroom = fields.Many2one('school.classroom', ondelete='set null', help='Clase a la que pertenece')
    teachers = fields.Many2many('school.teacher', related='classroom.teachers', readonly=True)

    # Le puede entrar uno o más listas de estudiantes
    # Self será una lista de estudiantes, si sólo es un estudiante, self será una lista de un estudiante
    @api.depends('name') #Se calculará cuando cambie o se cree el campo nombre
    def _get_password(self):
        # Todas las funciones que calculan campos deben recorrer toda la lista de estudiantes
        # Si solo es un estudiante vendrá un único estudiantes
        for student in self:
            # student es una instancia del modelo student.
            student.password = secrets.token_urlsafe(12) # Generará un token de 12 bytes
            print('\033[94m', student, '\033[0m')
            raise Warning(_('Se ha producido un Warning!')) # Aquí no tiene sentido pero es para verlo como ejemplo
```

```
2021-02-12 12:07:34,349 1216 INFO pruebas werkzeug: 192.168.203.137 - - [12/Feb/2021 12:07:34] "POST /lo
2021-02-12 12:07:44,086 1216 INFO pruebas werkzeug: 192.168.203.137 - - [12/Feb/2021 12:07:44] "POST /we
 school.student(<NewId 0x7f0db02bf820>,)
2021-02-12 12:07:44,105 1216 WARNING pruebas odoo.http: ('Se ha producido un Warning!', '')
2021-02-12 12:07:44,109 1216 INFO pruebas werkzeug: 192.168.203.137 - - [12/Feb/2021 12:07:44] "POST /we
2021-02-12 12:07:44,233 1216 INFO ? werkzeug: 192.168.203.137 - - [12/Feb/2021 12:07:44] "GET /web/stati
2021-02-12 12:07:44,244 1216 INFO ? werkzeug: 192.168.203.137 - - [12/Feb/2021 12:07:44] "GET /web/stati
2021-02-12 12:07:45,693 1216 INFO pruebas werkzeug: 192.168.203.137 - - [12/Feb/2021 12:07:45] "POST /lo
```

- Elimino esto porque era sólo para un ejemplo. Si lo que quiero es mandar información de log, que se muestra en el terminal y también se almacena en el fichero de log, lo haría así:

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

from odoo import models, fields, api
import secrets
import logging

_logger = logging.getLogger(__name__)
```

```python
class student(models.Model):
    _name = 'school.student'
    _description = 'Los alumnos'

    name = fields.Char(string="Nombre", readonly=False, required=True, help='Este es el nombre')
    birth_year = fields.Integer()
    password = fields.Char(compute='_get_password', store=True)
    description = fields.Text()
    inscription_date = fields.Date()
    last_login = fields.Datetime()
    is_student = fields.Boolean()
    photo = fields.Image(max_width=300, max_height=300) # Field binario pero específico para imágenes.
    classroom = fields.Many2one('school.classroom', ondelete='set null', help='Clase a la que pertenece')
    teachers = fields.Many2many('school.teacher', related='classroom.teachers', readonly=True)

    # Le puede entrar uno o más listas de estudiantes
    # Self será una lista de estudiantes, si sólo es un estudiante, self será una lista de un estudiante
    @api.depends('name') #Se calculará cuando cambie o se cree el campo nombre
    def _get_password(self):
        # Todas las funciones que calculan campos deben recorrer toda la lista de estudiantes
        # Si solo es un estudiante vendrá un único estudiantes
        for student in self:
            # student es una instancia del modelo student.
            student.password = secrets.token_urlsafe(12) # Generará un token de 12 bytes
            print('\033[94m', student, '\033[0m')
            _logger.debug('\033[94m'+str(student)+'\033[0m')
```

- Al arrancar el servicio, le tengo que indicar esto si quiero que salgan los mensajes de debug, el resto sí salen.

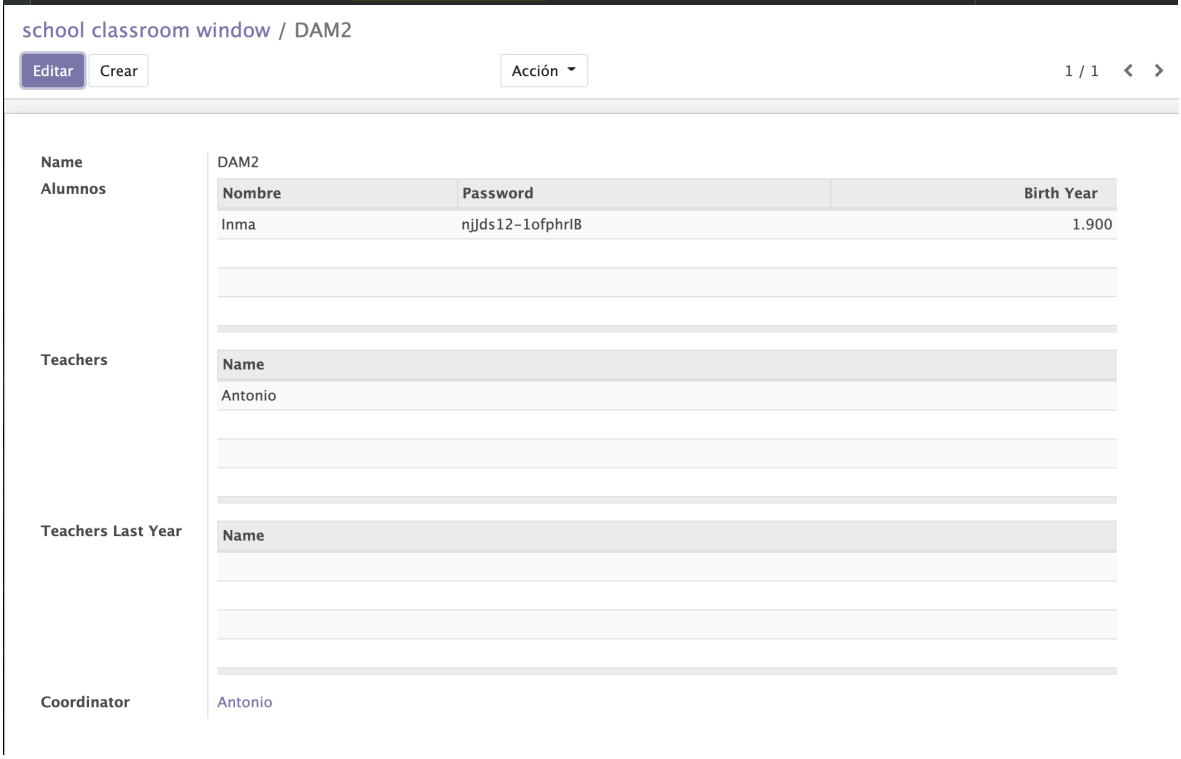
```
odoo@alumnodam:~$ odoo -u school -d pruebas --log-level=debug
school.student(<NewId 0x7f75081a7b20>,)
2021-02-12 12:18:57,428 1402 DEBUG pruebas odoo.addons.school.models.models: school.student(<NewId 0x7f75081a7b20>,)
2021-02-12 12:18:57,430 1402 INFO pruebas werkzeug: 192.168.203.137 - - [12/Feb/2021 12:18:57] "POST /web/dataset/call_kw/school.s
2021-02-12 12:19:08,921 1402 DEBUG pruebas odoo.modules.registry: Multiprocess signaling check: [Registry - 114 -> 114] [Cache - 2
```

- Igual que debut, puedo escribir warnings, info … etc. No lo haremos **NUNCA** con print.

CAMPOS COMPUTADOS RELACIONALES

- Vamos como ejemplo a crear un campo computado relacional en el que aunque no sea lo común vamos a considerar que una clase tiene un coordinador que será un profesor, y que un profesor puede ser el coordinador de muchas clases.
- Para el ejemplo, vamos a considerar que el coordinador va a ser el primero de la lista de profesores de la clase. Tenemos que poner el id para que funcione correctamente, porque lo que necesita es el identificador de la clave ajena a la que apuntará.

```python
class classroom(models.Model):
    _name = 'school.classroom'
    _description = 'Las clases'

    name = fields.Char() # Todos los modelos deben tener un field name
    students = fields.One2many(string='Alumnos', comodel_name='school.student', inverse_name='classroom')
    teachers = fields.Many2many(comodel_name='school.teacher',
                                relation='teachers_classroom',
                                column1='classroom_id',
                                column2='teacher_id')
    teachers_last_year = fields.Many2many(comodel_name='school.teacher',
                                relation='teachers_classroom_ly',
                                column1='classroom_id',
                                column2='teacher_id')
    # Vamos a considerar para el ejemplo que una clase puede tener un coordinador (profesor) y que un mismo profesor
    # pudiera ser coordinador de varias clases
    coordinator = fields.Many2one('school.teacher', compute='_get_coordinator')

    def _get_coordinator(self):
        for classroom in self:
            if len(classroom.teachers) > 0:
                classroom.coordinator = classroom.teachers[0].id #Para el ejemplo, vamos a establecer como coordinador al primero de la lista
```



- Vamos a añadir otro campo calculado que no tiene mucho sentido porque sería redundante pero para practicarlo. Vamos a mostrar un campo **All teachers** en el que aparezcan todos los profesores (los actuales y los del año pasado):

```python
class classroom(models.Model):
    _name = 'school.classroom'
    _description = 'Las clases'

    name = fields.Char() # Todos los modelos deben tener un field name
    students = fields.One2many(string='Alumnos', comodel_name='school.student', inverse_name='classroom')
    teachers = fields.Many2many(comodel_name='school.teacher',
                                relation='teachers_classroom',
                                column1='classroom_id',
                                column2='teacher_id')
    teachers_last_year = fields.Many2many(comodel_name='school.teacher',
                                relation='teachers_classroom_ly',
                                column1='classroom_id',
                                column2='teacher_id')
    # Vamos a considerar para el ejemplo que una clase puede tener un coordinador (profesor) y que un mismo profesor
    # pudiera ser coordinador de varias clases
    coordinator = fields.Many2one('school.teacher', compute='_get_coordinator')

    all_teachers = fields.Many2many('school.teacher', compute='_get_teacher')

    def _get_coordinator(self):
        for classroom in self:
            if len(classroom.teachers) > 0:
                classroom.coordinator = classroom.teachers[0].id #Para el ejemplo, vamos a establecer como coordinador al primero de la lista

    def _get_teacher(self):
        for classroom in self:
            # Para trabajar, acepta o lista de id o recordset, las dos cosas le valen. En este caso le metemos recordset.
            classroom.all_teachers = classroom.teachers + classroom.teachers_last_year
```