DIAGRAM SOFTWARE

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

PROGRAMACIÓN DE OPENERP/ODOO

1.2. MODELOS

José Zambudio Bernabeu
jose.zambudio@diagram.net

NubeaERP global cloud solutions

DIAGRAM SOFTWARE

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ÍNDICE

- Tipos de elementos
- Introducción
- Definiendo un modelo
- Atributos
- Methods
- Fields
  - Parameters
  - Simples
  - Relacionales
  - Funcionales
  - Special fields names

INTRODUCCIÓN MODELO

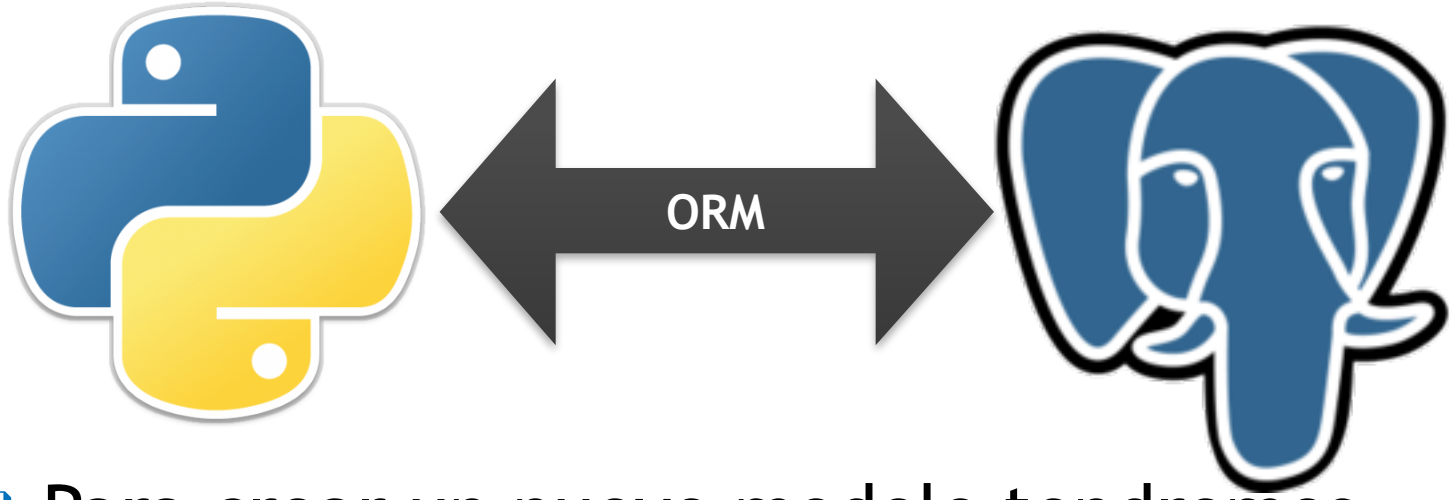
- Tipos de elementos
  1. Business object
     - Clases de python las cuales extienden de la clase orm.Model. La gestión en base de datos de dichos módulos la mantiene el ORM.
  2. Data
     - Vienen dados por ficheros XML o CSV, y nos proporcionan todos aquellos datos que serán introducidos en base de datos. Entre estos tenemos: la declaración de vistas, workflows, datos demo, datos de configuración…
  3. Reports
     - Ficheros RML, HTML/MAKO o templates de OpenOffice con los que junto con la información de nuestro módulo, utilizaremos para generar informes PDF, HTML o ODT.

INTRODUCCIÓN MODELO

- Tenemos un objeto por cada tipo de elemento, no por cada instancia de este.
  - Es decir, tenemos un objeto res.partner para controlar TODOS los partner, no un objeto por cada partner.
  - Por lo tanto… los métodos de un objeto son comunes entre todas sus instancias. Por ello, un parámetro común en todos los métodos es "ids".
    - ➡ids: Lista de ids en los que el método va a ser ejecutado.
    - ➡Ejemplo:
      - openerp/addons/base/res/res_partner.py:701

```
def email_send(self, cr, uid, ids, email_from, subject, body, on_error=''):
```

DEFINIENDO UN MODELO



- Para crear un nuevo modelo tendremos que definir una clase de Python que herede de **orm.Model (7.0)**

< 7.0	7.0	8.0
osv.osv	*orm.Model*	*models.Model*
osv.osv_memory	*orm.TransientModel*	*models.TransientModel*
osv.osv_abstract	*orm.AbstractModel*	*models.AbstractModel*

DEFINIENDO UN MODELO

```python
from osv import fields, orm


class mi_nuevo_modelo(orm.Model):
    _name = 'mi_modulo.mi_nuevo_modelo'
    _columns = {
        ...
    }
    ...
mi_nuevo_modelo()
```

```python
from osv import fields, osv


class mi_nuevo_modelo(osv.osv):
    _name = 'mi_modulo.mi_nuevo_modelo'
    _columns = {
        ...
    }
    ...
mi_nuevo_modelo()
```

- La Definiendo de un modelo siempre seguirá dicha forma.
- Los modelos vienen definidos por atributos los cuales se declaran en la clase del objeto.
- Los atributos “_name” y “_columns” son los únicos **obligatorios**

DEFINIENDO UN MODELO

```
from osv import fields, orm


class mi_nuevo_modelo(orm.Model):
    _name = 'mi_modulo.mi_nuevo_modelo'
    _columns = {
        ...
    }
    ...
mi_nuevo_modelo()
```

```
from osv import fields, osv


class mi_nuevo_modelo(osv.osv):
    _name = 'mi_modulo.mi_nuevo_modelo'
    _columns = {
        ...
    }
    ...
mi_nuevo_modelo()
```

Por convención

- Nombres en minúsculas
- Separados por _
- Deben comenzar por el nombre del módulo al que pertenecen.

- La Definiendo forma.
- Los modelos cuales se declaran en la clase del objeto.
- Los atributos “_name” y “_columns” son los únicos **obligatorios**

ATRIBUTOS DE UN MODELO

- **_auto** = Determinamos si el ORM debe generar automáticamente la tabla PostgreSQL.
  - orm.Model -> default = True
  - orm.TransientModel -> default = True
  - orm.AbstractModel -> default = False
- **_register** = No visible para el ORM, es decir, hablamos de un modelo que sólo queremos herencia en python.
  - orm.Model -> default = False
  - orm.TransientModel -> default = False
  - orm.AbstractModel -> default = False
- **_transient** = Únicamente indica que el modelo mantiene los datos en ddbb de forma temporal.
  - No tocar, utilizar TransientModel para indicar dicho uso.

ATRIBUTOS DE UN MODELO

- **_name** = (Obligatorio) Indica el nombre del modelo.
  - default = None
- **_columns** = (Obligatorio) Donde vamos a indicar los atributos del modelo.
  - type = dict
  - default = {}
- **_constraints** = Definimos constraints del modelo.
  - type = list
  - default = []
- **_sql_constraints** = Definimos constraints del modelo.
  - type = list
  - default = []

ATRIBUTOS DE UN MODELO

- **_name** = (Obligatorio) Indica el nombre del modelo.
  - default = None
- **_columns** = (Obligatorio) Donde vamos a indicar los atributos del modelo.
  - type = dict
  - default = {}
- **_constraints** = Definimos constraints del modelo.
  - type = list
  - default = []
- **_sql_constraints** = Definimos constraints del modelo.
  - type = list
  - default = []

MENOS CUANDO INDICAMOS _INHERITS

NOSOTROS INDICAMOS LA LÓGICA

UNIQUE CONSTRAINT

ATRIBUTOS DE UN MODELO

- **_defaults** = Valores por defecto para los atributos del modelo (definidos en _columns)
  - type = dict
  - default = {}
- **_inherit** = Nombre del modelo (orm.Model) del que vamos a heredar.
  - En caso de especificarse, podemos no especificar el atributo _name, lo veremos más adelante…
  - En caso de especificarse _name y _inherit CAMBIA la forma en la que vamos a heredar.
  - type = String | list(strings)
- **_inherits** = Lista de modelos (orm.Model) de los que vamos a heredar. “Poliformismo” en OpenERP…
  - type = dict
  - default = {}

ATRIBUTOS DE UN MODELO

- **_log_access** = Determina si OpenERP debe registrar la creación y escritura de los registros del modelo.
  - True:
    - create_uid: Quién crea el registro.
    - create_date: Cuándo se ha creado el registro.
    - write_uid: Quién ha sido el último en modificar el registro.
    - write_date: Cuándo se ha modificado por última vez el registro.
  - Estos campos se obtienen del método perm_read del ORM.
  - default = mismo valor que el field _auto

ATRIBUTOS DE UN MODELO

- **_order** = Donde indicamos el orden que va a utilizar el ORM a la hora de listar los registros del modelo.
  - type = string
  - default = 'id'
  - ASC - DESC
  - Los métodos "search" y "read" del ORM listarán los elementos en dicho orden.
  - Ejemplo:
    - _order = 'name desc'
- **_rec_name** = Indicamos el field a mostrar para referenciar visualmente los registros del modelo.
  - default = 'name'
  - Con el método "name_get" podemos especificar el contenido de este atributo de forma dinámica.

ATRIBUTOS DE UN MODELO

- **_sequence** = Nombre de la secuencia SQL que gestiona los ids del modelo.
  - default = None
  - != ir_sequence
- **_sql** = Indicamos la query SQL a ejecutar después de la creación de la tabla en la base de datos.
  - type = string
  - Únicamente se ejecuta si _auto = True
  - Puede ejecutar varias queries separadas por ';'
- **_table** = (DEPRECATED) => _model
  - Indicamos el nombre de la tabla de la ddbb.
  - type = string
  - default = valor del campo _name substituyendo los '.' por '_'

ATRIBUTOS DE UN MODELO

- **_parent_name** = Indicamos nombre del atributo del modelo utilizado en _columns el cuál será utilizado para indicar el padre de dicho registro.
  - default = 'parent_id'
- **_parent_store** = Indica que guarde en base de datos el id del padre para no tener que recalcularlo.
  - default = False
- **_parent_order** = Orden a aplicar dentro de la jerarquía.
  - type = string
  - default = False
  - En caso de no indicarse se obtiene del field _order
- **_date_name** = Se necesita para la vista calendar creada por defecto.
  - Valores que puede contener:
    - date | date_start | x_date | x_date_start
  - default = 'date'

ATRIBUTOS DE UN MODELO

- **_description** = Indicamos el comentario que se añadirá a la tabla en ddbb.
  - En caso de no indicarse el valor será el mismo que el field _name
  - default = None
- **_group_by_full** = TODO
  - type = dict
  - default = {}

ORM METHODS

read
write
create
default_get
perm_read
unlink
fields_get
fields_view_get
search
name_get
distinct_field_get
name_search
copy
copy_data
import_data
search_count
exists
browse

COLUMNS FIELDS

- Un modelo puede contener varios tipos de fields.
- Podemos definir 3 categorías de tipos:
  - Fields simples
    - ***boolean*, *integer*, *float*, *char*, *text*, *date*, *datetime*, *binary*, *selection*, *html*.**
  - Fields relacionales
    - Representan relaciones entre los modelos.
    - ***one2many*, *many2one*, *many2many***
  - Fields funcionales.
    - ***function***
    - Tipo especial. Pueden no guardarse en ddbb, o si, según lo definamos, recalculando el valor del campo o obteniéndolo de un calculo previo.

PARAMETERS - COLUMNS FIELDS

- Parámetros comunes a todos los fields:
  - **change_default** = Indicamos que dicho field va a ser utilizado para calcular el default de otro field.
    - type = boolean
    - default = False
  - **help** = Indicamos una explicación de cómo funciona, cómo debe utilizarse…etc, dicho campo. Esta ayuda es la que se muestra por un tooltip al poner el cursor por encima del label de dicho campo.
    - type = string
    - default = ‘’

PARAMETERS - COLUMNS FIELDS

- **ondelete** = Indicamos al ORM cómo debe de tratar los borrados en el registro relacionado.
  - type = string
  - values = “restrict”, “no action”, “cascade”, “set null”, “set default”
  - default = “set null”
- **readonly** = Indicamos si dicho campo debe ser únicamente de lectura o no.
  - type = boolean
  - default = False

PARAMETERS - COLUMNS FIELDS

- **required** = Indicamos si el campo es obligatorio para poder crear un registro en el modelo.
  - type = boolean
  - default = False
- **size** = Indicamos el tamaño del campo en DDBB
  - type = integer
- **string** = Indicamos el nombre del campo a mostrar en el label de la vista, o en la cabecera de la columna.
  - type = unicode -> u”Teléfono”
  - default = ‘unknown’

PARAMETERS - COLUMNS FIELDS

- **states** = Nos permite modificar el comportamiento del resto de parámetros en un estado específico del registro.
  - type = dict
  - default = None
  - Por ejemplo:

```
'mi_campo': fields.many2one(
    'otro_modulo.otro_modelo','Mi campo', readonly=True, states={
        'posted':[('readonly', False)]
    }),
```

PARAMETERS - COLUMNS FIELDS

- **translate** = Indicamos si el valor del campo debe ser traducible.
  - type = boolean
  - default = False
- **priority** = Está definido en el core… está… pero no sirva para nada :S
- **domain** = Indicamos una restricción de los valores accesibles por dicho campo relacional.
  - type = list | string
  - default = []

PARAMETERS - COLUMNS FIELDS

- **invisible** = Indicamos si dicho campo debe estar oculto en la interfaz del usuario.
  - type = boolean
  - default = False
- **relation** = Cuando el campo es una referencia a otro registro de otro modelo (por id), con este parámetro indicamos la tabla relación.
  - type = string
  - default = ‘’

PARAMETERS - COLUMNS FIELDS

- **select** = Indicamos el valor por defecto para el campo en la vista.
  - type = integer
  - default = False = 0
  - values
    - 1 basic search
    - 2 advanced search
- **deprecated** = Indicamos que dicho campo está obsoleto. OpenERP lo indicará con un Warning en el log.
  - type = boolean
  - default = False
- **groups** = TODO
  - type = string
  - default = ‘’

COLUMNS FIELDS - SIMPLE

- **fields.boolean(...)**
  - view -> input checkbox
  - pgtype = bool
  - default = False
  - required=True no sirve para nada...
- **fields.integer(...)**
  - view -> input text
  - pgtype = int4
  - default = 0
- **fields.float(...)**
  - view -> input text
  - default = 0.0
  - ...digits=...
    - type = tuple
      - (nº dígitos enteros, nº dígitos decimales)
  - ...digits_compute=...
    - procedimiento que devuelve el valor a aplicar en digits
  - pgtype
    - if digits => float8
    - else => numeric

COLUMNS FIELDS - SIMPLE

- **fields.char(...)**
  - ...size=...
  - ...translate=...
  - view -> input text
  - pgtype = varchar
- **fields.text(...)**
  - ...translate=...
  - view -> input textarea
  - pgtype = text
- **fields.html(...)**
  - hereda de fields.text(...)
  - view -> input textarea
  - pgtype = text

COLUMNS FIELDS - SIMPLE

- **fields.date(...)**
  - view -> input text + jQuery.datepicker()
  - pgtype = date
  - static methods
    - today(*args) > :rtype: str
      - Devuelve la fecha actual con el formato tools.DEFAULT_SERVER_DATE_FORMAT
    - context_today(model, cr, uid, context=None, timestamp=None) :rtype: str
      - En caso de no especificarse timestamp devuelve today
      - Devuelve la fecha con respecto el timezone del cliente con el formato tools.DEFAULT_SERVER_DATE_FORMAT
- **fields.datetime(...)**
  - view -> input text + jQuery.datepicker()
  - pgtype = timestamp
  - static methods
    - now(*args) > :rtype: str
      - tools.DEFAULT_SERVER_DATETIME_FORMAT
    - context_timestamp(cr, uid, timestamp, context=None)
      - :rtype: datetime

COLUMNS FIELDS - SIMPLE

- **fields.binary(…)**
  - Cadena binaria que OpenERP introduce en ddbb como cadenas codificadas en base64.
  - …filters=…
    - Filtros para la selección
    - type = string
    - "*.png,*.gif"
  - view -> input file
  - pgtype = bytea
- **fields.selection(…)**
  - view -> input selection
  - …values=…
    - type = tuple of tuples
      - (…('key_or_value', 'string_to_display'),…)
    - Podemos definir una función que devuelva dicha tupla
  - pgtype = Según elemento 0 de la tupía
    - if isinstance int => int4
    - else => varchar

COLUMNS FIELDS - RELATIONAL

- **fields.many2one(name="", string=""…)**
  - …ondelete=…
  - …select=True…
  - 1er parámetro: name debe ser un _name de un modelo válido.
  - 2do parámetro: string
  - …opcionales…

(0, 0, {fields})	*Crea un nuevo registro*
(1, ID, {fields})	*Actualiza campos del registro ID*
(2, ID)	*Borra el registro ID*
(3, ID)	*Borra la relación con ID*
(4, ID)	*Añade un registro ya existente.*
(5)	*Borra todos los elementos (one2many)*

COLUMNS FIELDS - RELATIONAL

- **fields.one2one(…)**
  - No existe dicha relación, la simulamos con many2one.

COLUMNS FIELDS - RELATIONAL

- **fields.one2many(…)**
  - Inversa de many2one, requiere de dicho field.
  - No puede contener el parámetro:
    - change_defaults
  - 1 parámetro: Indicamos el _name del modelo al que realizamos la relación.
  - 2 parámetro: Indicamos el nombre del field inverso a nuestro one2many.
  - 3 parámetro: String
  - …opcionales…

COLUMNS FIELDS - RELATIONAL

- **fields.many2many(…)**
  - 1er parámetro: Modelo al que realizamos la relación (_name)
  - 2do parámetro: (opcional) Indicamos el nombre de la tabla que el ORM generará para relacionar ambos modelos. En caso de no especificarla OpenERP generará un nombre basado en la regla:
    - modelA_modelB_rel
  - 3er parámetro: (opcional) indicamos el nombre de la columna que hará referencia al id del modelo actual. En caso de no especificarse OpenERP genera un nombre tal que:
    - src_model_id
  - 4to parámetro: (opcional) indicamos el nombre de la columna que hará referencia al id del modelo al que realizamos la relación.
    - dest_model_id
  - 5to parámetro: String
  - …Opcionales…
  - Para hacer dicho field bidireccional, debemos definirlo en ambos modelos. ¡Cuidado a la hora de definir el 3er y 4to parámetro!

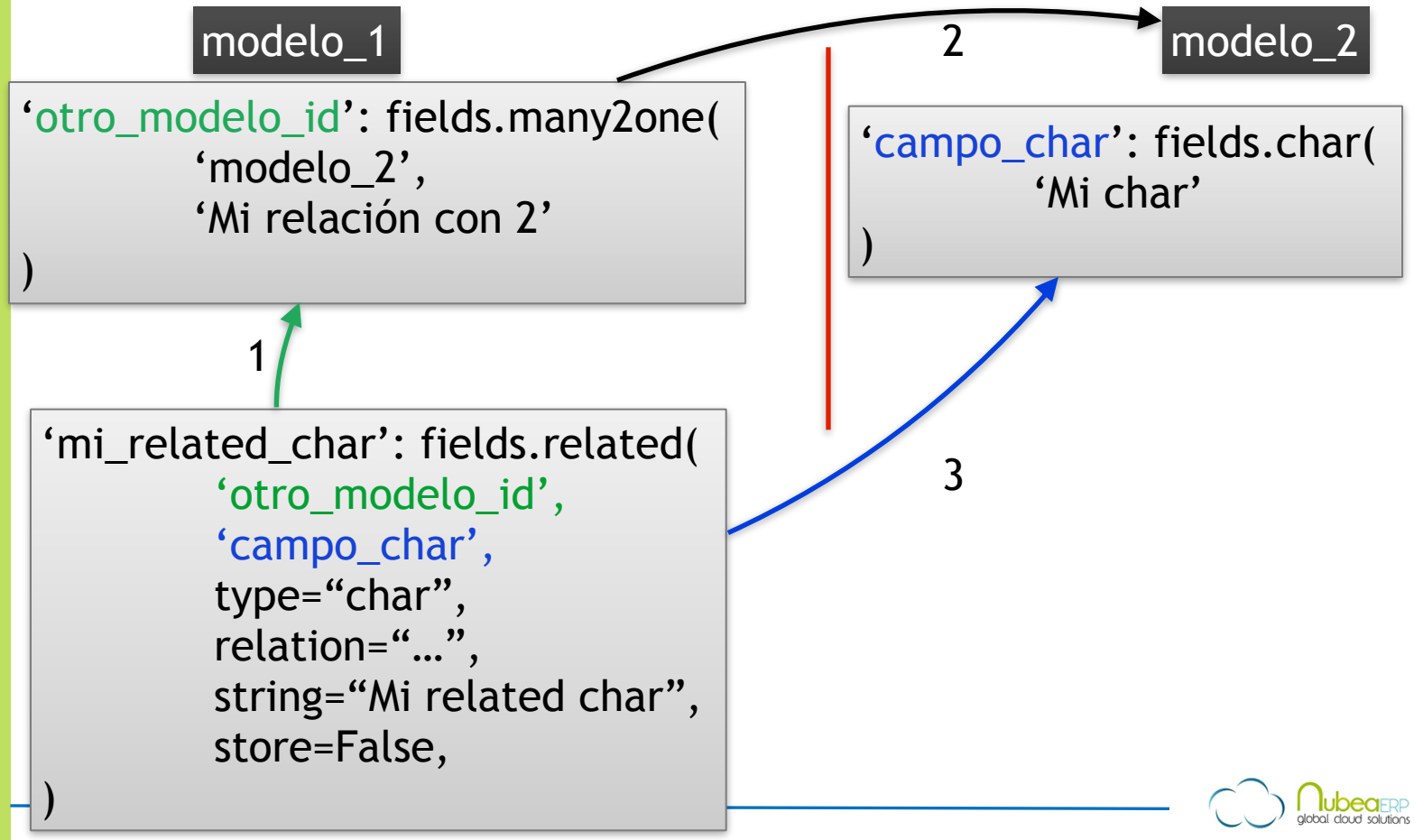
COLUMNS FIELDS - RELATIONAL

- **fields.one2many(...) && fields.many2many(...)**

(0, 0, {fields})	*Crea un nuevo registro*
(1, ID, {fields})	*Actualiza campos del registro ID*
(2, ID)	*Elimina la relación con el registro ID, y además borra también el registro ID*
(3, ID)	*Borra la relación con ID, pero no el registro ID*
(4, ID)	*Añade un registro (ya existente) a la relación.*
(5)	*Borra todos los elementos utilizando la opción (3, ID)*
(6, 0, [IDs])	*Reemplaza todos los registros existentes realizando un (5), por unos ya creados realizando un (4, ID) de los IDs que pasamos.*

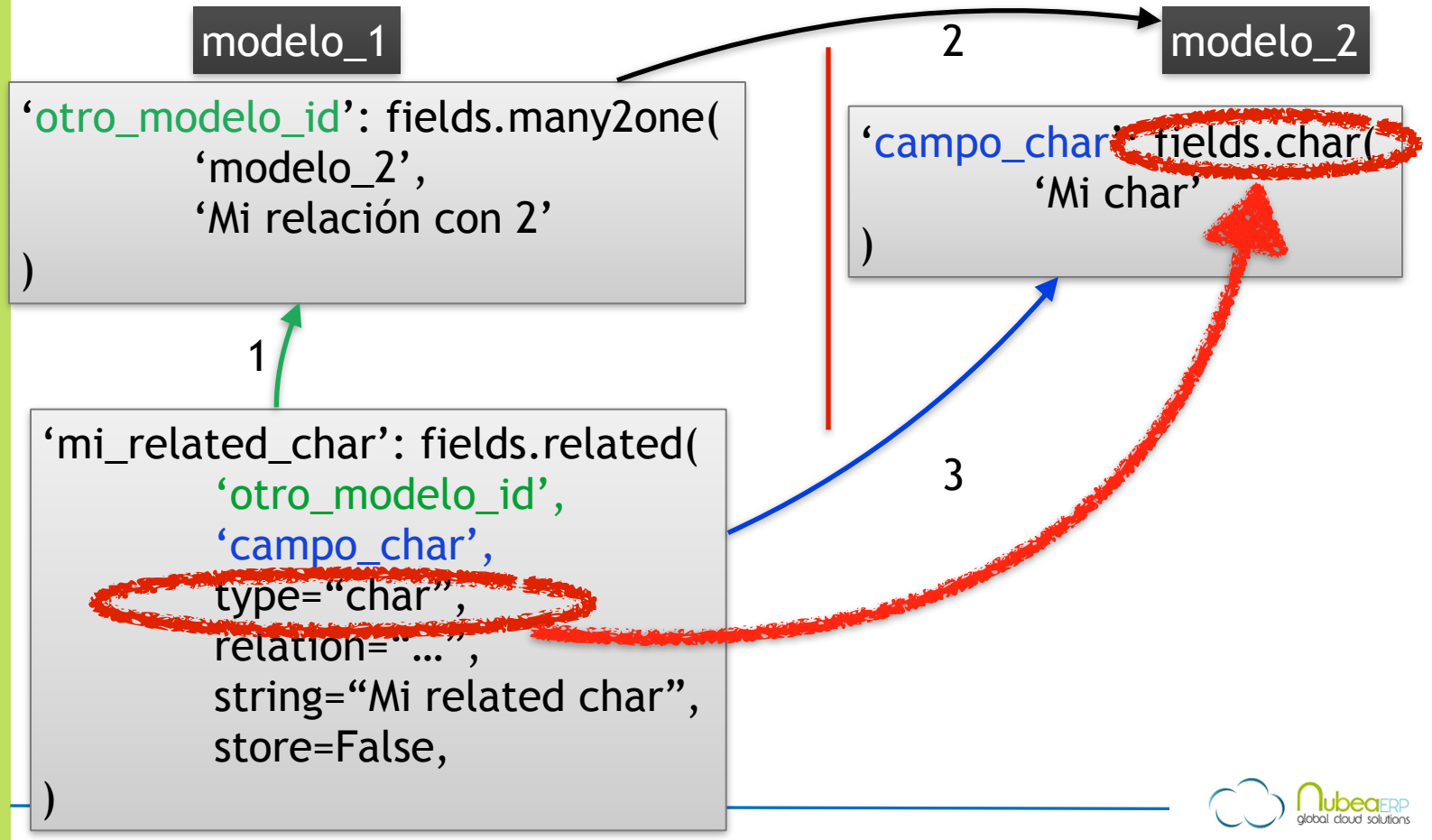
COLUMNS FIELDS - RELATIONAL

- **fields.related(...)**
  - Un campo que apunta a otro campo de otro modelo al cual tenemos relación.



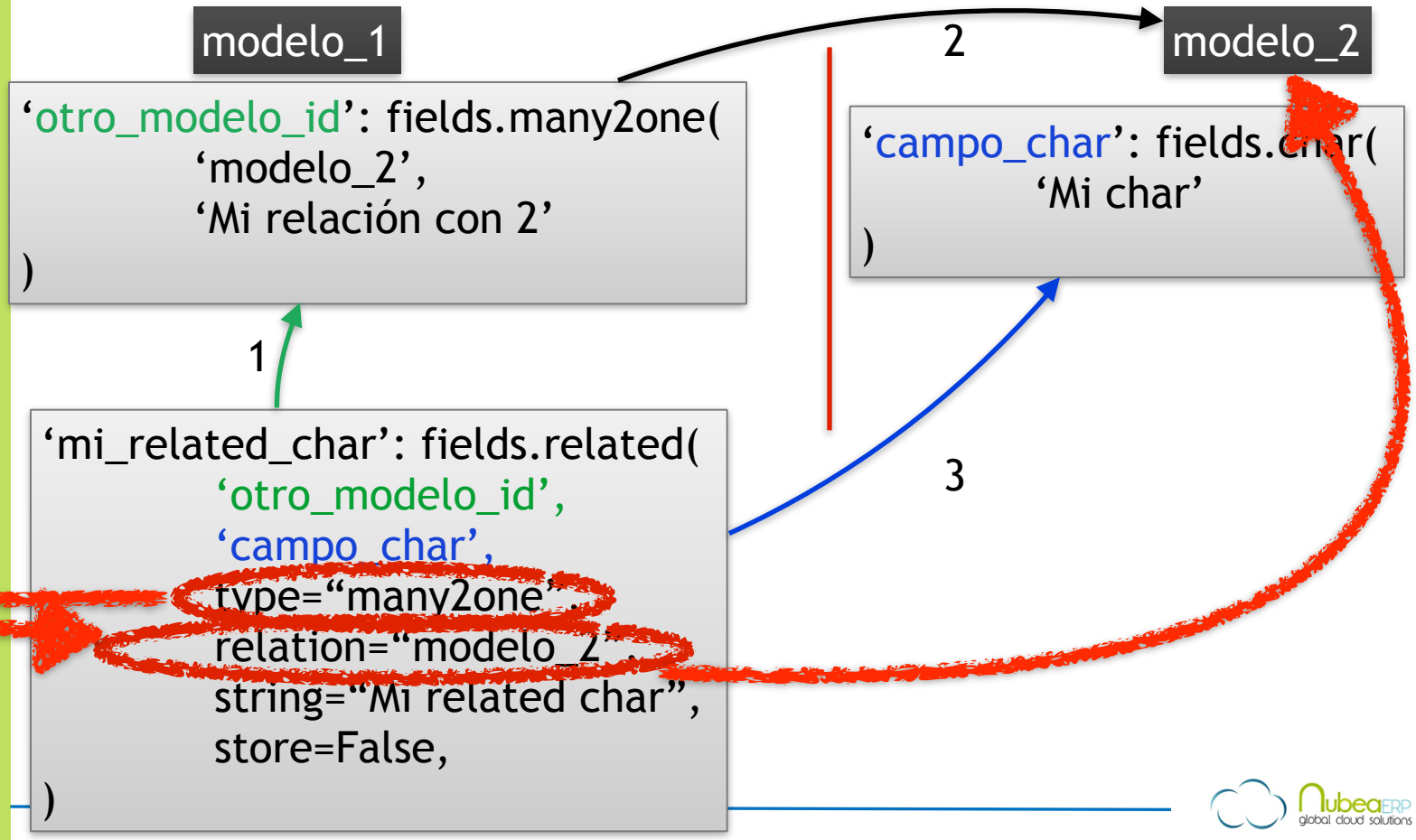
COLUMNS FIELDS - RELATIONAL

- **fields.related(...)**
  - Un campo que apunta a otro campo de otro modelo al cual tenemos relación.



COLUMNS FIELDS - RELATIONAL

- **fields.related(...)**
  - Un campo que apunta a otro campo de otro modelo al cual tenemos relación.



COLUMNS FIELDS - RELATIONAL

- **fields.reference(string, selection, size)**
  - Pseudo-many2one
  - Puede hacer referencia a registros de varios modelos. Es decir, cada registro contiene el modelo al que hace referencia y el ID al registro que hace referencia.
  - string…
  - selection:
    - Lista de modelos de puede contener dicho campo.
    - Igual que fields.selection:
      - key: Nombre del modelo
      - Name: El que queramos
  - size:
    - OpenERP indica que debe ser None
    - Se utiliza para restringir el tamaño del campo en base de datos.

COLUMNS FIELDS - FUNCIONALES

- Fields funcionales
  - **fields.function(...)**
  - **fields.property(...)**

COLUMNS FIELDS - FUNCIONALES

- **fields.function(**
  fnct,
  arg=None,
  fcnt_inv=None,
  fnct_inv_arg=None,
  type="float?",
  fnct_search=None,
  store=False,
  multi=False
)
  - **fnct:** Método que va a calcular el valor del campo. El método debe haber sido declarado antes que los fields.
  - **arg:** Valores arbitrarios que le pasamos a **fnct** para calcular su valor.

COLUMNS FIELDS - FUNCIONALES

- **fnct(self, cr, uid, ids, field_name, arg, context)**
  - Debe devolver un diccionario con:
    - key: id del registro
    - value: valor del registro
  - En caso de tener el parámetro multi=True debe devolver:
    - key: id del registro
    - value: diccionario:
      - key: nombre del field
      - value: valor del field.
    - Para el ejemplo anterior:
      - {
        1:{
        'tax': 1.05,
        'total_without_tax': 5.0,
        'total_with_tax': 6.05,
        } ,
        2:{…},
        …
        }

COLUMNS FIELDS - FUNCIONALES

- **fields.function(**
  fnct,
  arg=None,
  fcnt_inv=None,
  fnct_inv_arg=None,
  type="float?",
  fnct_search=None,
  store=False,
  multi=False
)
  - **fcnt_inv:** Método que va a permitir valores en este campo, en caso de no definirse pasa a considerarse readonly.
  - **fnct_inv_arg:** Valores arbitrarios que le pasamos a **fnct_inv** para calcular su valor.

COLUMNS FIELDS - FUNCIONALES

- **fnct_inv(self, cr, uid, ids, field_name, field_value, fnct_inv_arg, context)**
  - Devuelve: True
  - Implementamos el write para dicho campo.
  - En esta función el parámetro “multi” es ignorado.

COLUMNS FIELDS - FUNCIONALES

- **fields.function(**
  fnct,
  arg=None,
  fcnt_inv=None,
  fnct_inv_arg=None,
  type="float?",
  fnct_search=None,
  store=False,
  multi=False
)
  - **type:** Nombre del tipo de campo devuelto por la función. Puede ser cualquiera exceptuando "function". En caso de ser un relacional, tenemos que añadir el parámetro "relation"
  - **fnct_search:** Definimos el método con la lógica para buscar por este campo.

COLUMNS FIELDS - FUNCIONALES

- **fnct_search(self, cr, uid, model, field_name, criterion, context)**
  - model: Tiene el mismo valor que self… tal cuál…
    - Es para compatibilizar con la función search internamente.
  - criterion: Lista de condiciones de búsqueda por defecto.
  - Returns: Nueva lista de condiciones de búsqueda. Esta lista debe basarse en registros insertados en base de datos. Se suele ejecutar el método **fcnt,** de dichos valores obtenemos los registros que cumplen la condición y devolvemos un domain tal que:
    - [('id', 'in', [1,2,3,4,5,6,7…])]

COLUMNS FIELDS - FUNCIONALES

- **fields.function(**
  fnct,
  arg=None,
  fcnt_inv=None,
  fnct_inv_arg=None,
  type="float?",
  fnct_search=None,
  store=False,
  multi=False
)
  - **store:** Indicamos si queremos almacenar el valor del campo.
  - **multi:** Se utiliza para agrupar llamadas de varios function. Por ejemplo calcular los totales:
    - Base imponibe
    - Total sin IVA
    - Total cliente

COLUMNS FIELDS - FUNCIONALES

- **store=…**
  - Este parámetro permite almacenar el valor del registro que tenemos en caché, en la base de datos. Además, podemos indicarle una serie de triggers para que re-calcule el valor y lo vuelva a insertar en base de datos.
  - Posibles valores:
    - False -> Este campo siempre se re-calculará, y no será almacenado en la base de datos.
    - True -> El campo se almacenará en base de datos, y será re-calculado siempre que se modifique algún dato del mismo registro.
    - Triggers!

COLUMNS FIELDS - FUNCIONALES

- **store=…**
  - Triggers

```
'amount_total': fields.function(_amount_all, digits_compute=dp.get_precision('Account'), string='Total',
    store={
        'sale.order': (lambda self, cr, uid, ids, c={}: ids, ['order_line'], 10),
        'sale.order.line': (_get_order, ['price_unit', 'tax_id', 'discount', 'product_uom_qty'], 10),
    },
    multi='sums', help="The total amount."),
```

- {'trigger_model': (mapping_function, ['trigger_fields',…], priority)}
- En el caso de la imagen, el primer trigger es exactamente el mismo caso que dejar store=True.
- **mapping_function(trigger_model, cr, uid, trigger_ids, context)**

```
def _get_order(self, cr, uid, ids, context=None):
    result = {}
    for line in self.pool.get('sale.order.line').browse(cr, uid, ids, context=context):
        result[line.order_id.id] = True
    return result.keys()
```

  - Devolvemos la lista de ids deregistros del model del field.function para que estos sean recalculados.

COLUMNS FIELDS - FUNCIONALES

- **store=…**
- **{'trigger_model': (mapping_function, ['trigger_fields',…], priority)}**
  - **trigger_fields:**
    - Lista de nombres de fields de nuestro **trigger_model** los cuales ejecutan el evento de re-cálculo de nuestro fields.function al ser modificados.
    - En caso de especificarse vacía la lista, OpenERP presupone que es para todos los fields.
  - **priority**
    - Por defecto 10
    - Indicamos el orden a ejecutar los triggers.

COLUMNS FIELDS - FUNCIONALES

- **fields.property(model, type="many2one", relation="model", string="…", group_name="…")**

SPECIAL FIELDS NAMES

id	*Identificador único para el registro*
name	*Campo a mostrar para mostrar el registros en los listados. Podemos cambiar este por el tag: _rec_name*
active	*Visibilidad del registro.*
sequence	*Permite drag&drop de los registros para reordenar estos desde el navegaor*
state	*Definición de los estados para el workflow.*
parent_id	*Se utiliza para definir una jerarquización de los registros. Si lo definimos nos habilita la búsqueda por hijos (child_of)*
parent_left, parent_right,	*Se utilizan junto con _parent_store en el modelo. Permite mayor velocidad de acceso en la estructura.*
create_date, create_uid, write_date, write_uid	*Por defecto en todos los modelos a no ser que le indiquemos lo contrario en _log_access. Los añade el ORM, podemos sobreescribirlos para tener acceso a estos.*