TECHNICAL TRAINING

odoo10

INSTALASI - KONFIGURASI - STRUKTUR MODUL - ORM - MENU - DATABASE - INHERITANCE - CONSTRAINT - VIEWS - STATE - SECURITY - ACCESS RIGHT - RECORD RULES - WIZARD - REPORTING

STUDI KASUS MODUL TRAINING

DAFTAR ISI

Introduction 1
Object Relational Mapping (ORM) 7
Module 15
Field Relation 20
Inheritance 28
Domain 33
Constraints 42
Views 47
Security 62
Wizard 66
Report 71

INTRODUCTION

Bismillah ...

Insya Allah saya akan memulai lagi untuk berbagi sedikit ilmu yang saya miliki khususnya seputar technical odoo 10 yang telah menggunakan API baru. Materi saya ambil dari berbagai ebook dan web yang menjelaskan technical odoo (terutama di website resminya).

Odoo di kembangkan dengan framework OpenObject dengan bahasa python. Olehkarna itu, sebelum kita masuk ke frameworknya, pastikan kita sudah berkenalan dengan python, minimal mengetahui syntax dasarnya. Karna framework OpenObject banyak perbedaan disana dengan bahasa python, sudah banyak campuran dengan ORM dan 'peringkasan' atau 'pembungkusan' python code. Silahkan mecoba tutorial python yang sangat interaktif di :

https://codesaya.com/python/

atau

https://www.codecademy.com/learn/python

Untuk tutorial ini, saya menggunakan OS Linux Ubuntu 16.04, jika anda berbeda OS, maka silahkan untuk menyesuaikannya. Selanjutnya adalah Instalasi Odoo. Ada banyak sekali tata cara instalasi odoo, silahkan pilih cara yang terbaik bagi anda, saya juga pernah membahasnya pada halaman ini.

Setelah Odoo berjalan dengan baik, maka untuk memulainya kita harus membuat sebuah modul baru. Pada versi terbaru, odoo telah menyiapkan 'shortcut' yang bernama "SCAFFOLD", untuk membuat modul secara instan yang mengandung 'komponen' default pada sebuah modul odoo.

Buat terlebih dahulu file odoo-bin yang diambil dari folder setup (odoo). Rename namanya menjadi odoo-bin lalu berikan akses untuk bisa di eksekusi (chmod a+x) lalu copas keluar folder setup. Kita langsung coba ketikan :

 # python [path_source_odoo]/odoo-bin scaffold -t default [nama_modul] [path_addons]

python ~/odoo/odoo-10.0/odoo-bin scaffold -t default training_odoo ~/odoo/addons/DEMO10/

Perintah diatas digunakan untuk membuat modul default pada odoo, sedangkan jika kita ingin membuat modul theme, maka perintahnya seperti ini :

 # python [path_source_odoo]/odoo-bin scaffold -t theme [nama_modul] [path_addons]

python ~/odoo/odoo-10.0/odoo-bin scaffold -t theme theme_training_odoo ~/odoo/addons/DEMO10/

Untuk windows sudah terdapat file odoo-bin pada folder instalasi odoo, jadi jalankan file tersebut dari cmd (tanpa syntax python) :

odoo-bin scaffold -t default [nama_modul] [path_addons]

Silahkan di sesuaikan untuk **[path_source_odoo]**, **[nama_modul]** dan **[path_addons]** nya masing-masing. Untuk kelancaran tutorial, sebaiknya [nama_modul] disamakan.

Setelah kita mengeksekusi perintah di atas, maka pada addons kita akan tercipta sebuah modul baru dengan struktur sebagai berikut :

training_odoo
. **controllers**
.... __init__.py
.... controllers.py
. **demo**
.... demo.xml
. **models**
.... __init__.py
.... models.py

. **security**

.... ir.model.access.csv

. **views**

.... templates.xml

.... views.xml

. **__init__.py**

. **__manifest__.py**

Pada versi ini, struktur MVC (Model View Controller) sudah terlihat, dengan adanya pembagian per folder. Sama halnya seperti konsep MVC yang lain, masing-masing folder memiliki fungsi tersendiri, penjelasan ringkasnya sebagai berikut :

A. Controllers = Berisi file **python** yang mengandung perintah untuk komunikasi model & view (untuk website)

B. Demo = Berisi file **xml** yang mengandung data untuk demo (saat create database centang "Load Demostration Data")

C. Models = Berisi file **python** yang mengandung perintah untuk ORM dan lainnya

D. Security = Berisi file **csv** yang mengandung akses right pada setiap object (tabel) dan file **xml** untuk membuat group sesuai kebutuhan akses right

E. Views = Berisi file **xml** yang mengandung syntax untuk membuat tampilan suatu view yaitu list, form, kanban, calendar, pivot, dan graph

Nantinya kita juga bisa menambahkan beberapa folder lagi sesuai kebutuhan seperti report, wizard, i18n, static, dll. Seperti berikut :

F. Report = Berisi file **python** dan file pembuat report seperti **xml**, **rml**, **mako**, dll

G. Wizard = Berisi file **python** dan **xml** untuk membuat object & view dari suatu wizard (object on the fly)

H. i18n = Berisi file translator

I. Static = Berisi file pendukung web seperti javascript, css, dll

Disamping folder, modul yang telah kita buat juga mengandung 2 file yaitu : __init__.py dan __manifest__.py seperti penjelasan berikut ini :

J. __init__.py

File ini berfungsi untuk menjadikan folder biasa menjadi modul python. Karna Odoo dikembangkan dengan python, maka modul odoo harus dikenali oleh python. File ini juga berfungsi sebagai **constructor** sebagaimana pada file python, Odoo akan membaca file ini pertama kali saat diinstal. Pada file ini kita harus mengimport semua **folder** yang mengandung file **python** dan mengimport file **python** yang sejajar dengan file __init__ ini jika ada

K. __manifest__.py

Sebagaimana file __init__.py, file ini juga berfungsi untuk menjadikan folder biasa menjadi modul odoo. fungsinya sama seperti file __openerp__.py pada versi odoo sebelumnya. Isinya mengandung informasi bagi user saat ingin menginstall modul terkait. Isinya seperti berikut :

```
{
    'name': "training_odoo",

    'summary': """
        Short (1 phrase/line) summary of the module's purpose, used
as
        subtitle on modules listing or apps.openerp.com""",

    'description': """
        Long description of module's purpose
    """,

    'author': "My Company",
    'website': "http://www.yourcompany.com",

    # Categories can be used to filter modules in modules listing
    # Check
https://github.com/odoo/odoo/blob/master/odoo/addons/base/module/modu
le_data.xml
    # for the full list
    'category': 'Uncategorized',
    'version': '0.1',

    # any module necessary for this one to work correctly
    'depends': ['base'],

    # always loaded
    'data': [
        # 'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
    ],
    # only loaded in demonstration mode
```

```
    'demo': [
        'demo/demo.xml',
    ],
}
```

Penjelasan ringkasnya sebagai berikut :

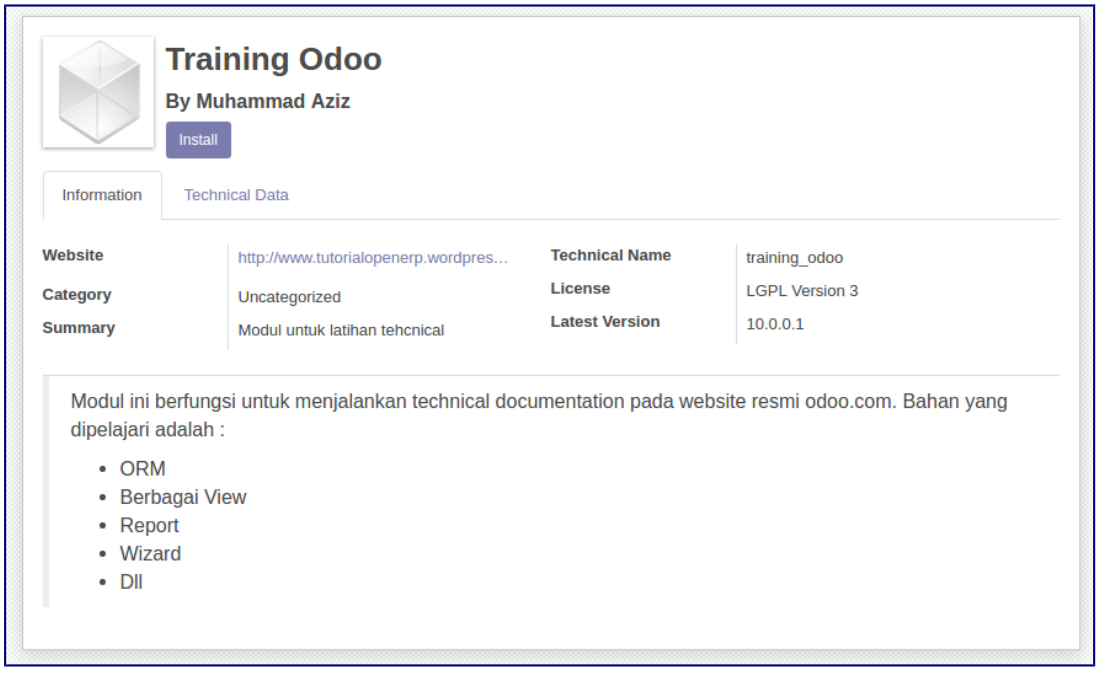
name : (String) # Nama modul
summary : (String) # Ringkasan fungsi modul
description : (String) # Penjelasan fungsi modul
author : (String) # Pembuat modul
website : (String) # Website modul
category : (String) # Category modul
version : (String) # Versi modul
depends : (List) # Modul yang diisi akan otomatis terinstall
data : (List) # Akan meload file xml & csv (Isinya file dari folder security, views, data, report, wizard)
demo : (List) # Akan meload file xml & csv (Isinya dari folder demo)
image : (List) # Gambar atau screenshoot dari modul
qweb : (List) # Akan meload file xml (Isinya file dari folder static)
css : (List) # Akan meload file css (Isinya file dari folder static)
installable : (Boolean) # Jika diisi True, maka modul dapat diinstal
application : (Boolean) # Jika diisi True, modul akan terfilter Apps
auto_install : (Boolean) # Jika diisi True, modul akan otomatis terinstall saat pembuatan database

Kedua file python ini (__init__ & __manifest__) wajib ada pada setiap modul odoo. Dan tanpa salah satunya, maka odoo tidak bisa mengenalinya. Selanjutnya kita akan merubah isi dari file __manifest__, sesuai dengan informasi yang kita inginkan. Seperti berikut ini :

```
{
    'name': "Training Odoo",
    'summary': """
        Modul untuk latihan tehcnical """,

    'description': """
        Modul ini berfungsi untuk menjalankan technical documentation pada website resmi odoo.com. Bahan yang dipelajari adalah :\n
        - ORM\n
        - Berbagai View\n
        - Report\n
        - Wizard\n
        - Dll
    """,
```

```
    'author': "Muhammad Aziz",
    'website': "http://www.tutorialopenerp.wordpress.com",

    # Categories can be used to filter modules in modules listing
    # Check
https://github.com/odoo/odoo/blob/master/odoo/addons/base/module/module_data.xml
    # for the full list
    'category': 'Uncategorized',
    'version': '0.1',

    # any module necessary for this one to work correctly
    'depends': ['base'],

    # always loaded
    'data': [
        # 'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
    ],
    # only loaded in demonstration mode
    'demo': [
        'demo/demo.xml',
    ],
}
```

Lanjutkan dengan proses instalasi modul, yaitu klik **Apps > Update Apps List**. Cari modul sesuai nama "training_odoo" lalu klik install seperti gambar berikut :



ORM (OBJECT RELATIONAL MAPPING)

Sebelum kita praktek untuk membuat object, menu, dll. Sebaiknya kita review kembali teori dan penjelasan materi terkait.

Setiap class di odoo dia akan menginherite super class model, class model inilah kunci dari ORM yang ada di Odoo. Diantara attribute yang dimiliki oleh class model ini adalah :

_name : untuk membuat sebuah object atau table di database

contoh : _name = "sale.order"

_description : memberikan informasi singkat mengenai object atau model

contoh : _description = "Sales Orders"

_rec_name : jika kita tidak memiliki field 'name', maka isikan attribute ini dengan field penggantinya. Secara default, field 'name' akan digunakan sebagai nilai pada method name_get() (field many2one)

contoh : _rec_name = 'complete_name'

_inherit : untuk inheritance object terkait. Biasanya kita gunakan untuk custom baik penambahan field maupun override method

contoh : _inherit = "stock.move"

_inherits : untuk 'pendelegasian' object terkait. Contohnya adalah object **'res.users'** yang menginherits object **'res.partner'**. Ketika membuat user, maka partner (customer/supplier) juga akan tercipta otomatis (tidak sebaliknya) dengan **tambahan** sebuah field 'jembatan' antara 'res.user' dan 'res.partner' yaitu **'partner_id'** seperti contohberikut:

contoh : _inherits = {'res.partner': 'partner_id'}

_order : untuk proses sorting, isi dengan nama salah satu field. Jika tidak diisi maka defaultnya field 'id'

contoh : _order = 'date desc'

_sql_constraints : untuk membuat unik nilai suatu field

contoh : _sql_constraints = [('code_uniq', 'unique (code)', 'The code of the account must be unique !')]

_constraints : untuk membuat pengecekan nilai suatu field dengan method python yang kita bikin

contoh : _constraints = [(_check_required_if_provider, 'Required fields not filled', [])]

Semenjak api baru, attribute _constraint digantikan dengan decorator

@api.constrains('nama_field'), contohnya : @api.constrains('line_ids'). Selain itu, pada api baru ini, attribute **_columns** untuk mendefinisikan field telah dihapus. Sedangkan **_defaults** juga telah dihapus dan dimasukan ke attribute field

Jika kita ingin membuat duplikasi dari object yang sudah ada tetapi dengan nama yang berbeda, maka gunakan attribute **_inherit** dan **_name** secara bersamaan. Contohnya :

```
class AnggotaPerpustakaan(models.Model):
    _inherit = 'res.partner'
    _name = 'anggota.perpustakaan'
```

Setelah kita membuat object, langkah selanjutnya adalah membuat field atau column dari object tersebut. Odoo menyediakan berbagai jenis field, diantaranya :

```
# Char = Field free character, biasanya digunakan untuk inputan
string. Contoh :
name = fields.Char(string='Order Reference', required=True,
copy=False, readonly=True)

# Boolean = Field ini bernilai True atau False, berbentuk cekbox pada
list/form. Contoh :
sale_ok = fields.Boolean('Can be Sold', default=True, help="Check if
the product can be selected in a sales order line.")

# Integer = Field dengan nilai bilangan bulat. Contoh :
sequence = fields.Integer('Sequence', default=1)

# Float = Field dengan nilai bilangan decimal. Contoh :
discount = fields.Float(string='Discount (%)',
digits=dp.get_precision('Discount'))

# Monetary = Field dengan nilai bilangan decimal (khusus keuangan)
dan otomatis dengan simbol currencynya (dalam 1 field). Contoh :
amount = fields.Monetary(currency_field='currency_id')

# Text = Field untuk inputan text berbentuk paragraf. Contoh :
term = fields.Text('Terms and conditions')

# Selection = Field berbentuk combobox/dropdown yang nilainya telah
ditentukan dengan hardcode. Contoh :
state = fields.Selection([('draft', 'Draft'),('done', 'Done')],
required=True)
```

```
# Html = Field seperti text dengan tambahan formating seperti html
note = fields.Html('Note')

# Date = Field inputan tanggal. Contoh :
date_invoice = fields.Date(string='Invoice Date')

# Datetime = Field inputan tanggal dan waktu. Contoh :
date_done = fields.Datetime(string="Closed On")

# Binary = Field untuk menyimpan sebuah data file, biasanya digunakan
untuk image. Contoh :
data_file = fields.Binary(string='Bank Statement File')

# Many2One = Sama seperti field selection tetapi nilainya diambil
dari sebuah object/tabel. Contoh :
partner_id = fields.Many2one('res.partner', string='Customer',
change_default=True, index=True, track_visibility='always')

# One2Many = Field berbentuk tabel pada sebuah form. Contoh :
order_line = fields.One2many('sale.order.line', 'order_id',
string='Order Lines')

# Many2Many = Field berbentuk tabel pada sebuah form (mirip one2many)
dengan pengisian nilainya seperti field many2one. Contoh :
tag_ids = fields.Many2many('crm.lead.tag', 'crm_lead_tag_rel',
'lead_id', 'tag_id', string='Tags')

# Reference = Sama seperti field many2one tetapi object/tabel
relasinya bebas untuk memilih (tampilan ada 2 combobox, milih object
& milih recordnya). Contoh :
document_id = fields.Reference(selection=_get_document_types,
string='Source Document', required=True)
```

Seperti contoh diatas, ada perbedaan pada api yang baru ini, yaitu tidak adanya field **compute** & field **related** seperti di versi sebelumnya. Karna field **compute** & field **related** telah di masukan menjadi attribute sebuah field, jadi semua field diatas bisa dijadikan field **compute** & field **related** dengan menambahkan attribute compute atau related. Disana juga ada tambahan attribute default, seperti penjelasan sebelumnya.

Ada beberapa field yang tercipta otomatis (default) ketika kita membuat object, walaupun kita tidak mendefiniskan field tersebut, diantara field itu adalah :

id = berisi sequence record (auto increment)
create_date = berisi tanggal record dibuat
create_uid = berisi user yang membuat record
write_date = berisi tanggal terakhir record di update
write_uid = berisi user yang mengedit record terakhir kali

Jika kita tidak mengingingkannya, maka kita bisa 'mematikan' field tersebut dengan memberikan attribute model **_log_access = False**

Ada juga spesial field yang bisa kita tambahkan, yaitu field 'Active', yang bernilai boolean (defaultnya True). Secara default, semua record yang ditampilkan bernilai True, jika kita ingin menghide sebuah record, maka set field 'Active' menjadi False. Contoh definisinya seperti ini :

active = fields.Boolean('Active', default=True) # Pada versi 10, sebagian form ada kotak "Archive / Active" untuk mengedit field ini

Jika kita perhatikan contoh diatas, setiap field memiliki attribute sebagaimana model, diantara attribute yang dimiliki field adalah :

string -- (string) berfungsi sebagai label pada GUI. Jika tidak di set, maka ORM memberikan label dari nama aslinya (db) dengan capital.

size -- (integer) berfungsi memberikan batasan jumlah karakter (khusus field Char)

help -- (string) berfungsi memberikan tooltip jika cursor di dekatkan

readonly -- (boolean) jika bernilai True, maka field tidak dapat di edit manual

required -- (boolean) jika bernilai True, maka field akan bersifat mandatory (wajib di isi)

index -- (integer) berfungsi untuk memberikan index pada database, mempercepat proses searching tetapi menambah alokasi database. Pengganti attribute **select** pada versi sebelumnya

default -- (sesuai tipe field) berfungsi memberikan nilai default (baik static maupun dinamis -dengan method-) saat user akan membuat record

states -- (dictionary) berfungsi untuk membuat readonly, required, & invisible suatu field berdasarkan nilai field statenya

domain -- (list) memberikan filter dari nilai yang dipilih (biasa digunakan field relasi -many2one dan many2many-). Seperti fungsi 'where' pada sql select

digits -- (tuple) untuk menentukan banyaknya digit decimal pada field float dan monetary

groups -- (string) memberikan akses suatu field kepada group terkait

copy -- (boolean) jika bernilai True, maka akan memberikan value yang sama pada saat record di duplikasi

translate -- (boolean) jika bernilai True, maka label field terkait dapat di terjemahkan

selection -- (list) untuk menentukan nilai pilihan baik static maupun dinamic (dengan method) pada field selection atau field reference

change_default -- (boolean) jika bernilai True, maka field ini bisa memberikan nilai default field lain (dengan setting khusus)

track_visibility -- ('always', 'onchange') membuat log history (message) yang biasanya terdapat di bawah setiap form untuk proses tracking. Jika diisi 'always' maka setiap ada perubahan apapun pada field lain, maka ia tetap membuat log nya (walaupun nilai field ini tidak berubah), sedangkan 'onchange' akan membuat log jika HANYA field ini yang mengalami perubahan nilainya

company_dependent -- (boolean) jika bernilai True, maka value dari field ini bisa berbeda nilai per company. Pengganti attribute **property** pada versi sebelumnya

related -- (string) field duplikasi dari object/tabel lain. Field dengan attribute ini akan selalu mengupdate nilai jika ada perubahan nilai pada field aslinya (sinkronisasi)

compute -- (string) berfungsi memberikan nilai suatu field dari sebuah method

inverse -- jika attribute 'compute' biasa digunakan untuk mengisi nilai field ini berdasarkan parameter dari field lain, sedangkan attribute 'inverse' melakukan fungsi sebaliknya, yaitu mengisi nilai field lain dari nilai field ini

store -- (boolean) jika bernilai True, maka value akan di simpan pada database, biasa digunakan berpasangan dengan attribute related dan compute

ondelete -- ('set null', 'restrict', 'cascade') digunakan pada field many2one, jika bernilai 'cascade' maka record (yang mengandung field ini) akan ikut terhapus jika record yang dipilih dari object referensinya dihapus, jika bernilai 'restrict' maka record pada object referensinya tidak akan bisa dihapus sebelum record yang mengandung field ini dihapus, jika bernilai 'setnull' maka field akan otomatis bernilai null jika record pada object referensinya dihapus

search -- memberikan proses search pada field lain yang berhubungan dengan field ini (biasanya field yang terkait dari attribute compute atau inverse) -seperti domain-

Ada yang baru pada versi api ini dalam mengoverride field, ketika kita ingin menambahkan attribute sebuah field dari parent class, kita cukup menginherite parent class dan memberikan nama & type yang sama, sebagaimana field pada parent class tersebut. Lalu kita sisipkan HANYA attribute yang ingin ditambahkan, contoh :

```
class Parent(models.Model):
    _name = 'orang.tua'
    state = fields.Selection([('draft', 'Draft'),('done', 'Done')],
required=True)

class Child(models.Model):
    _inherit = 'orang.tua'
    state = fields.Selection(help="Ini attribut tambahan dari class
anak")
```

Pada contoh diatas, kita hanya menambahkan attribute **help**, tanpa harus mengcopas keseluruhan attribute parent.

Setelah kita mempelajari berbagai attribute model dan field. Maka selanjutnya adalah pembahasan tentang method suatu class atau model. Method dan attribute adalah dua hal yang tidak bisa kita pisahkan (keduanya saling melengkapi). Jika kita analogikan dengan sebuah object mobil, maka **attribute** seperti warna, merk, tahun pembuatan, dll. Sedangkan **method** seperti maju(), mundur(), berhenti(), dst. Diantara method yang dimiliki oleh class model adalah :

a. search()

Sama seperti fungsi **select** pada SQL, bedanya pada hasil yang diberikan. Contoh :

```
partner = self.env('res.partner').search([('is_company', '=', True),
('customer', '=', True)])
print partner # res.partner(7, 18, 12, 14, 17, 20)

partner = self.env('res.partner').search([('is_company', '=', True),
('customer', '=', True)], limit=3)
print partner # res.partner(7, 18, 12)

print partner[0].id # 7
print partner[0].name # Agrolait
print partner[1].id, partner[1].name # 18, Asustek
```

b. create()

Method ini berfungsi untuk membuat sebuah record pada object tertentu. Method ini membutuhkan parameter value (dictionary) berisi masing-masing field object tersebut beserta nilainya. Contoh :

```
self.env('res.partner').create({'name': 'Umar'})
```

c. write()

Method ini berfungsi untuk mengupdate satu atau beberapa field dari sebuah object. Parameter yang diperlukan adalah id dari record terkait dan value (dictionary) dari field-field yang akan diupdate. Contoh :

```
partner = self.env('res.partner').search([('name', '=', 'Umar')],
limit=1)
partner.write({'name': 'Abdullah', 'street': 'Jl. Gunung Salak'})
```

d. unlink()

Method ini digunakan untuk mendelete record pada object. Contoh :

```
partner = self.env('res.partner').browse([13, 14])
partner.unlink()
```

e. copy()

Method yang digunakan untuk menduplicate sebuah record yang menghasilkan record dengan nilai default (atau kita set) dari masing-masing field.

```
partner = self.env('res.partner').browse([15])
partner.copy()
# ATAU
partner.copy({'name': 'Ukasyah'})
```

f. browse()

Method yang berfungsi untuk mengambil sebuah record dan digunakan sebagai object, dengannya kita bisa mengexplore semua field dan relasi antar table pada database tersebut ‘tanpa batas’. Contoh :

```
partner = self.env('res.partner').browse([7, 18, 12])
print partner # res.partner(7, 18, 12)
print partner[0].account_receivable_id.company_id.currency_id.name #
USD
```

g. read()

Method yang memiliki fungsi sama seperti fungsi **select** pada SQL. Contoh :

```
partner = self.env('res.partner').browse([33, 34, 35])

data = partner.read(['nama'])
print data # [{'name': 'Ali'}, {'name': 'Abdurrahman'}, {'name':
'Uwais'}]

data = partner.read()
print data # [{'name': 'Ali', 'street': 'Jl. Danau Toba', 'city':
'Bekasi'}, {'name': 'Abdurrahman', 'street': 'Jl. Pisang Ambon',
'city': 'Depok'}, {'name': 'Uwais', 'street': 'Jl. Ahmad Yani',
'city': 'Jakarta'}]
```

h. ref()

Method yang memiliki fungsi mendapatkan record dari id xml. Contoh :

```
address_form_id = self.env.ref('base.res_partner_1')
print address_form_id # res.partner(7)
```

Diantara perbedaan dengan api yang lama adalah :

1. Telah hilangnnya parameter **cr, uid,** dan **ids** pada method. Semuanya digantikan dengan environment (**self.env**), termasuk **'self.pool.get'**

2. Return dari method diatas mayoritas adalah **recordset**, terutama untuk method **search()** -sebelumnya hanya return list integer-, jika kita biasa menggunakan method **search()** dan **browse()** bersamaan, maka dengan api yang baru ini, kita cukup menggunakan method **search()** saja

3. Method **read()** bisa kita gunakan untuk 'membaca' **beberapa** field yang kita inginkan saja, tidak seperti sebelumnya yang mereturn **semua** field dari record yang kita baca

4. Karna hilangnya parameter **ids** yang biasa digunakan untuk **read(), write(), copy()**, dan **unlink()**. Maka pastikan kita harus membuat recordset terlebih dahulu, sebelum memanggil method tersebut. Biasanya dengan search() atau browse()

MODULE

Insya Allah kita akan mulai untuk praktek membuat object berserta tampilannya. Modul yang akan kita gunakan adalah modul hasil dari scaffold pada pembahasan sebelumnya. Sumber materi paling dominan saya ambil dari sini :

Download Ebook
Sumber : http://www.odoo.com/documentation/10.0/index.html

Selanjutnya kita akan membuat object **'training.kursus'** dengan mengedit file python **models.py** yang berada di dalam folder **models**, sehingga hasilnya seperti dibawah ini :

```
from odoo import models, fields, api


class Kursus(models.Model):
    _name = 'training.kursus'

    name = fields.Char(string="Judul", required=True)
    description = fields.Text()
```

Bagian atas adalah hasil contoh dari scaffold, jadi kita abaikan saja. Nantinya nama object **'training.kursus'** akan berubah menjadi tabel **'training_kursus'** di databasenya. Untuk awalan, kita juga membuat 2 field yaitu **name** dan **description**. Setelah itu, kita lanjutkan dengan membuat tampilannya, agar user bisa mengaksesnya, yaitu edit file **views.xml** yang berada dalam folder **views**, seperti di bawah ini :

```
<odoo>
  <data>

        <!-- ### Membuat Tampilan Tree/List ### -->

        <record model="ir.ui.view" id="kursus_tree_view">
            <field name="name">training.kursus.tree</field>
            <field name="model">training.kursus</field>
            <field name="arch" type="xml">
                <tree string="Kursus List">
                    <field name="name"/>
                    <field name="description"/>
                </tree>
            </field>
        </record>
```

```
        <!-- ### Membuat Tampilan Form ### -->

        <record model="ir.ui.view" id="kursus_form_view">
            <field name="name">training.kursus.form</field>
            <field name="model">training.kursus</field>
            <field name="arch" type="xml">
                <form string="Kursus Form">
                    <sheet>
                        <group>
                            <field name="name"/>
                        </group>
                        <notebook>
                            <page string="Keterangan">
                                <field name="description"/>
                            </page>
                        </notebook>
                    </sheet>
                </form>
            </field>
        </record>

        <!-- ### Membuat Action/Event Object Kursus ### -->

       <record model="ir.actions.act_window" id="kursus_list_action">
            <field name="name">Kursus</field>
            <field name="res_model">training.kursus</field>
            <field name="view_type">form</field>
            <field name="view_mode">tree,form</field>
            <field name="help" type="html">
                <p class="oe_view_nocontent_create">Buatlah kursus
                pertamamu ...</p>
            </field>
        </record>

        <!-- ### Membuat Menu Bar ### -->

        <menuitem id="main_training_odoo_menu" name="Training Odoo"/>

         <!-- ### Membuat Menu Title ### -->

        <menuitem id="training_odoo_menu" name="Training"
                  parent="main_training_odoo_menu"/>

        <!-- ### Membuat Sub Menu ### -->

        <menuitem id="kursus_menu" name="Kursus"
                  parent="training_odoo_menu"
                  action="kursus_list_action"/>

  </data>
</odoo>
```

Bagian atas adalah hasil contoh dari scaffold, jadi kita abaikan saja. Pada file xml ini, kita membuat beberapa bagian, diantaranya :

Bagian 1 : Membuat Tampilan Tree/List. Jika tidak kita buat, maka defaultnya hanya field **name** saja yang tampil.

Bagian 2 : Membuat Tampilan Form. Setiap field harus berada di dalam tag **"group"**, jika tidak maka akan tidak ada labelnya (seperti field **description**). Lalu setiap tag **"page"** harus berada di dalam tag **"notebook"**, untuk menghasilkan tampilan dengan **multi tab** seperti di browser.

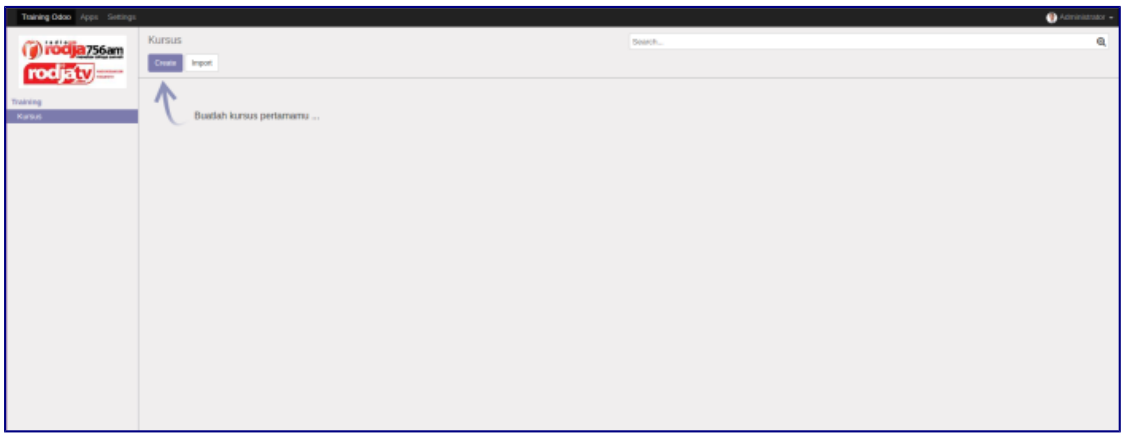
Bagian 3 : Membuat Action/Event Object Kursus. **Action** ini dijalankan ketika user mengklik submenu **Kursus**. Bagian **Action** ini juga tidak kalah pentingnya, disana kita bisa langsung memfilter (dengan value field) record yang ditampilkan, contohnya adalah menu **Customer** dan **Vendor**, kedua menu ini memiliki object yang sama, tetapi kita bisa membedakannya dengan **Action**. Begitu juga dengan penganturan tampilan, setiap object kita bisa tentukan tampilan (view) apa saja yang ingin kita sajikan.

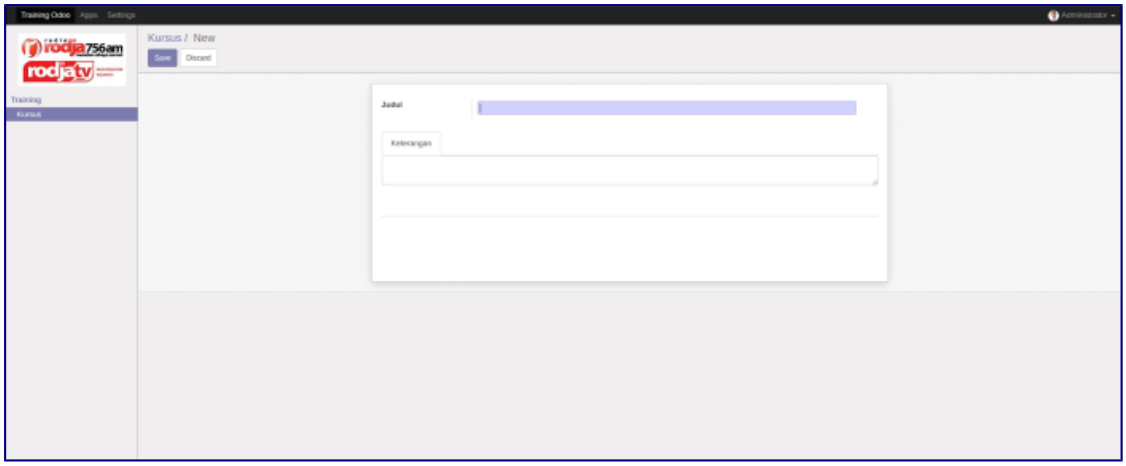
Bagian 4 : Membuat **Menu Bar**. Menu yang dihasilkan berada paling atas (**Top Menu**). Hal ini dikarenakan menu ini tidak memiliki parent menu.

Bagian 5 : Membuat **Menu Title**. Menu yang berada 1 tingkat dibawah **Top Menu**, fungsinya sebagai pengkategorian submenu.

Bagian 6 : Membuat **Sub Menu**. Tingkatan menu yang paling kecil, tempat dimana **Action** di tentukan. Semakin banyak hirarki parent yang dimiliki, maka akan semakin bawah hirarkinya.

Jika telah sudah selesai, maka lakukan upgrade modul jika memang modul ini sudah pernah diinstal seperti di tutorial sebelumnya, jika belum maka lakukan proses instalasi. Hasilnya seperti berikut :



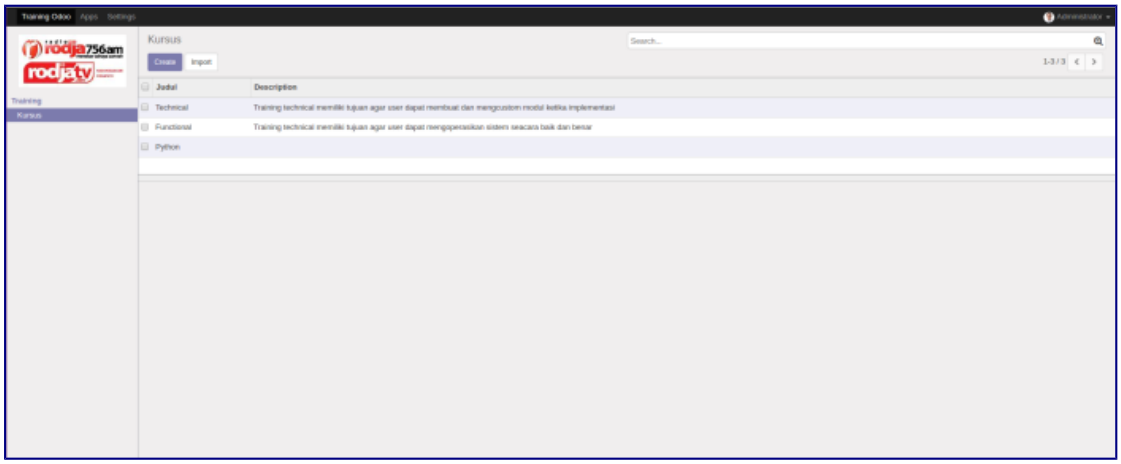


Kita juga bisa membuat data demo ketika modul pertama kali diinstal (pastikan cekbox "load demonstration data" di centang saat membuat db), caranya dengan mengedit file **demo.xml** pada folder **demo**, seperti berikut :

```
<odoo>
    <data>

        <record model="training.kursus" id="kursus_1">
            <field name="name">Technical</field>
            <field name="description">Training technical memiliki
                        tujuan agar user dapat membuat dan mengcustom
                        modul ketika implementasi</field>
        </record>

        <record model="training.kursus" id="kursus_2">
            <field name="name">Functional</field>
            <field name="description">Training technical memiliki
                        tujuan agar user dapat mengoperasikan sistem
                        seacara baik dan benar</field>
        </record>

        <record model="training.kursus" id="kursus_3">
            <field name="name">Python</field>
            <!-- tidak ada tambahan description -->
        </record>

    </data>
</odoo>
```

Membuat data demo sangat mudah, cukup mendefinikan objectnya kemudian field yang ingin diisi. Hal ini berguna pada beberapa kondisi, seperti membuat scheduler, sequence, dll.

Sehingga menghasilkan data seperti ini :



FIELD RELATION

Sebuah record dalam suatu model, biasanya memiliki hubungan dengan record di model lain, contohnya model **sale_order** memiliki relasi dengan **sale_order_line** dengan field penghubung yaitu **order_line** dan **order_id**. Odoo menyiapkan beberapa field relasi untuk kebutuhan ini, diantaranya adalah **many2one**, **one2many** dan **many2many**.

Object **sales_order** memiliki field **one2many (order_line)** yang berbentuk tabel product yaitu object **sale_order_line**, fungsinya untuk memperinci product apa saja yang di jual dalam **Sales Order** terkait, di dalam Odoo, setiap field **one2many** harus memiliki field **many2one** di tabel tujuan. Maka di object **sale_order_line**, ada field **order_id** sebagai foreign key dari tabel **sale_order**. Oleh karna itu setiap kita mendefinikan field **one2many**, disana kita harus menentukan field **many2one** yang akan menjadi relasi antar kedua tabel. Kita perhatikan contoh relasi kedua tabel tersebut :

```
class SaleOrder(models.Model):

    _name = "sale.order"

    order_line = fields.One2many('sale.order.line', 'order_id',
string='Order Lines', states={'cancel': [('readonly', True)], 'done':
[('readonly', True)]}, copy=True)


class SaleOrderLine(models.Model):

    _name = 'sale.order.line'

    order_id = fields.Many2one('sale.order', string='Order
Reference', required=True, ondelete='cascade', index=True,
copy=False)
```

Pada training kali ini, kita juga akan membuat relasi antar tabel seperti contoh diatas, yaitu antara tabel **training_kursus** dan **training_sesi**. Object/Tabel **training_sesi** berfungsi sebagai perinci tabel **training_kursus**. Contohnya jika kita mengikuti sebuah kursus, maka setiap kursus memiliki beberapa sesi. Jika kursusnya adalah **"Functional Odoo"**, maka sesinya adalah **"Introduction"**, **"Sales Management"**, **"Case Study"**, dll. Sekarang kita praktekan dengan membuat object **training_sesi** seperti dibawah ini (lanjutkan copas di bawah class **kursus()** sebelumnya) :

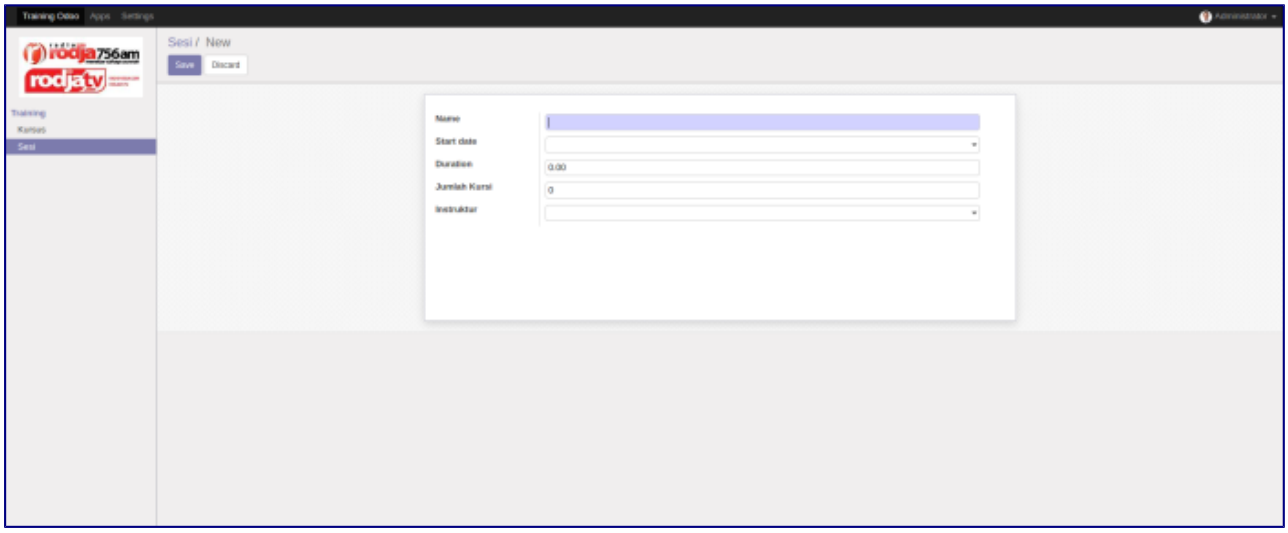
```
class Sesi(models.Model):
    _name = 'training.sesi'

    name = fields.Char(required=True)
    start_date = fields.Date()
    duration = fields.Float(digits=(6, 2), help="Durasi Hari")
    seats = fields.Integer(string="Jumlah Kursi")
    instructor_id = fields.Many2one('res.partner',
                            string="Instruktur")
```

Object **training_sesi** hanya memiliki 4 field dengan tipe yang berbeda-beda, selanjutnya kita tentukan menu dan view nya agar user bisa mengaksesnya (lanjutkan copas di bawah menu **kursus** sebelumnya) :

```
        <!-- ### Membuat Tampilan Tree/List Sesi ### -->

        <record model="ir.ui.view" id="sesi_tree_view">
            <field name="name">training.sesi.tree</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                <tree string="Sesi List">
                    <field name="name"/>
                    <field name="start_date"/>
                    <field name="duration"/>
                    <field name="seats"/>
                    <field name="instructor_id"/>
                </tree>
            </field>
        </record>



        <!-- ### Membuat Tampilan Form Sesi ### -->

        <record model="ir.ui.view" id="sesi_form_view">
            <field name="name">training.sesi.form</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                <form string="Sesi Form">
                    <sheet>
                        <group>
                              <field name="name"/>
                              <field name="start_date"/>
                              <field name="duration"/>
                              <field name="seats"/>
                              <field name="instructor_id"/>
                        </group>
                    </sheet>
                </form>
            </field>
        </record>
```

```
        <!-- ### Membuat Action/Event Object Sesi ### -->

        <record model="ir.actions.act_window" id="sesi_list_action">
                <field name="name">Sesi</field>
                <field name="res_model">training.sesi</field>
                <field name="view_type">form</field>
                <field name="view_mode">tree,form</field>
        </record>

        <!-- ### Membuat Sub Menu Sesi ### →

        <menuitem id="sesi_menu" name="Sesi"
                  parent="training_odoo_menu"
                  action="sesi_list_action"/>
```

Bila kita upgrade modul training kita, maka hasilnya seperti berikut :



Pada object **training_sesi**, kita belum membuat relasi apapun terhadap **training_kursus**. Sekarang kita tambahkan field relasinya seperti contoh relasi antar object **sale_order** dan **sale_order_line**. Seperti berikut :

```
class Kursus(models.Model):

    _name = 'training.kursus'

    ...

    session_ids = fields.One2many('training.sesi', 'course_id',
string="Sesi")
```

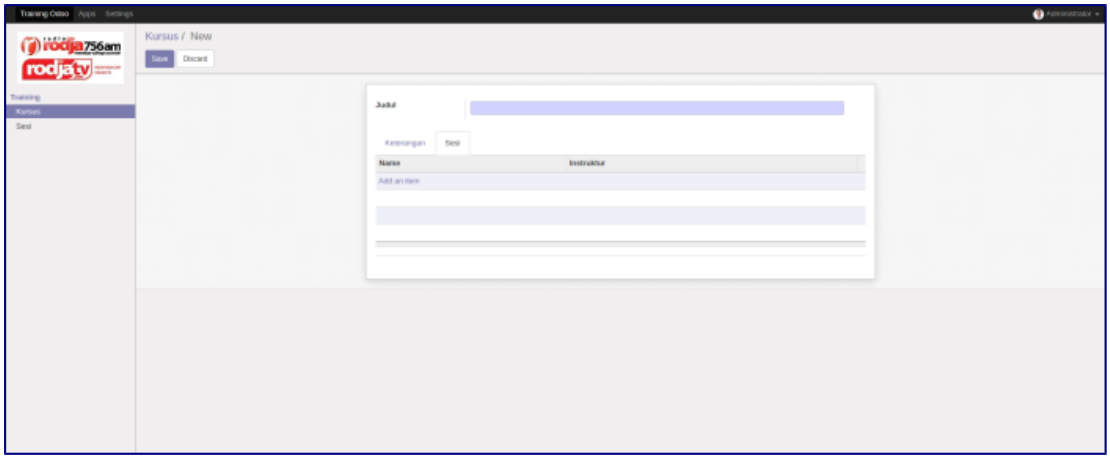
```
class Sesi(models.Model):

    _name = 'training.sesi'

    ...

    course_id = fields.Many2one('training.kursus',
ondelete='cascade', string="Kursus", required=True)
```

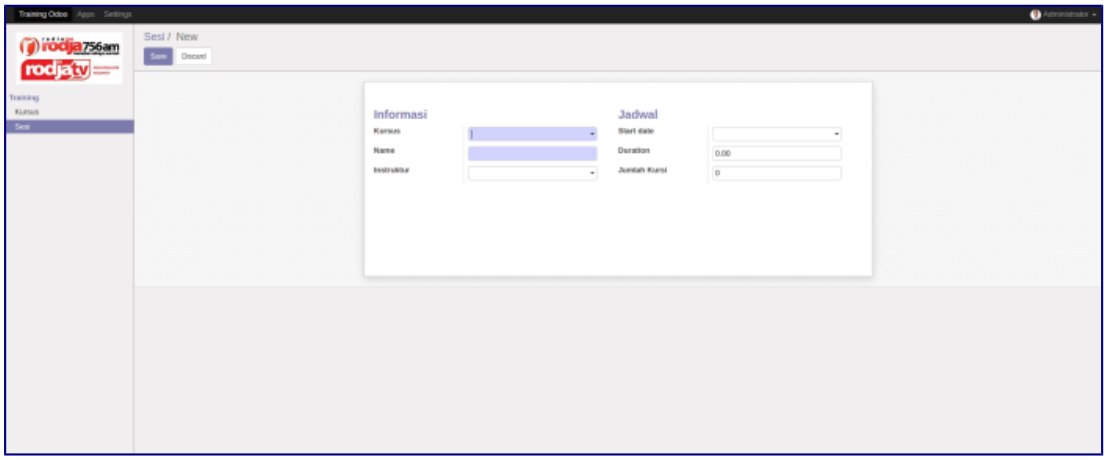
Attribute **ondelete='cascade'** memiliki arti jika record kursus dihapus, maka semua record sesi yang menginduk ke kursus tersebut juga akan dihapus. Kemudian tampilannya juga harus kita rubah agar sesuai dengan relasinya (replace sesuai **id xml** terkait) :

```
        <!-- ### Membuat Tampilan Form Kursus ### -->

        <record model="ir.ui.view" id="kursus_form_view">
            <field name="name">training.kursus.form</field>
            <field name="model">training.kursus</field>
            <field name="arch" type="xml">
            <form string="Kursus Form">
                  <sheet>
                  <group>
                      <field name="name"/>
                  </group>
                  <notebook>
                        <page string="Keterangan">
                            <field name="description"/>
                        </page>
                        <page string="Sesi">
                            <field name="session_ids">
                                <tree string="Daftar Sesi">
                                    <field name="name"/>
                                    <field name="instructor_id"/>
                                </tree>
                                <form>
                                   <group string="Informasi">
                                       <field name="name"/>
                                       <field name="instructor_id"/>
                                   </group>
                                   <group string="Jadwal">
                                       <field name="start_date"/>
                                       <field name="duration"/>
                                       <field name="seats"/>
                                   </group>
                                </form>
                            </field>
                        </page>
                    </notebook>
                    </sheet>
                </form>
            </field>
        </record>
```

```
        <!-- ### Membuat Tampilan Tree/List Sesi ### -->

        <record model="ir.ui.view" id="sesi_tree_view">
            <field name="name">training.sesi.tree</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                <tree string="Sesi List">
                    <field name="name"/>
                    <field name="course_id"/>
                    <field name="start_date"/>
                    <field name="duration"/>
                    <field name="seats"/>
                    <field name="instructor_id"/>
                </tree>
            </field>
        </record>

        <!-- ### Membuat Tampilan Form Sesi ### -->

        <record model="ir.ui.view" id="sesi_form_view">
            <field name="name">training.sesi.form</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
            <form string="Sesi Form">
                  <sheet>
                  <group>
                        <group string="Informasi">
                              <field name="course_id"/>
                              <field name="name"/>
                              <field name="instructor_id"/>
                        </group>
                        <group string="Jadwal">
                              <field name="start_date"/>
                              <field name="duration"/>
                              <field name="seats"/>
                        </group>
                  </group>
                  </sheet>
             </form>
             </field>
        </record>
```

Hasil dari perubahan coding di atas sebagai berikut :





Setelah kita berhasil menambahkan field relasi **one2many** dan **many2one**, maka ada 1 field relasi yang tesisa, yaitu **many2many**. Terakhir kita tambahkan field **many2many** kepada object **training_sesi** seperti berikut :

```
class Sesi(models.Model):

    _name = 'training.sesi'

    ...

    attendee_ids = fields.Many2many('res.partner', string="Peserta")
```

Sebelum kita update view xmlnya, sebagai penutup tutorial kali ini, saya ingin tambahkan lagi field relasi yang terakhir yaitu field **many2one** seperti berikut :

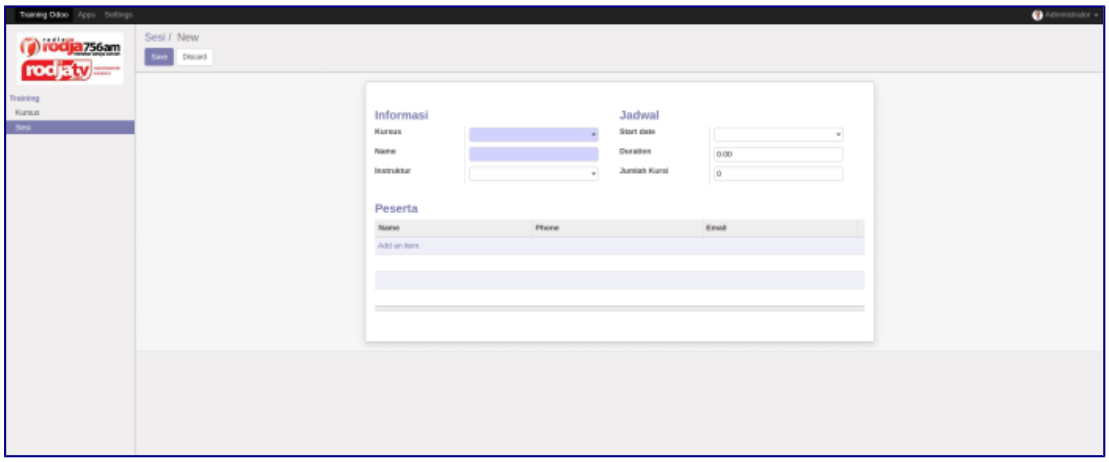
```
class Kursus(models.Model):

    _name = 'training.kursus'

    ...

    responsible_id = fields.Many2one('res.users',
      ondelete='set null', string="Penanggung Jawab", index=True)
```

Sekarang kita update tampilannya (perhatikan **id xml** nya untuk pengeditan) :

```
<!-- ### Membuat Tampilan Tree/List Kursus ### -->

<record model="ir.ui.view" id="kursus_tree_view">
    <field name="name">training.kursus.tree</field>
    <field name="model">training.kursus</field>
    <field name="arch" type="xml">
        <tree string="Kursus List">
            <field name="name"/>
            <field name="responsible_id"/>
            <field name="description"/>
        </tree>
    </field>
</record>

<!-- ### Membuat Tampilan Form Kursus ### -->

<record model="ir.ui.view" id="kursus_form_view">
    <field name="name">training.kursus.form</field>
    <field name="model">training.kursus</field>
    <field name="arch" type="xml">
        <form string="Kursus Form">
            <sheet>
                <group>
                    <field name="name"/>
                    <field name="responsible_id"/>
                </group>
                <notebook>
                    <page string="Keterangan">
                        <field name="description"/>
                    </page>
                    <page string="Sesi">
                        <field name="session_ids">
                            <tree string="Daftar Sesi">
                                <field name="name"/>
                                <field name="instructor_id"/>
                            </tree>
                            <form>
                              <group string="Informasi">
                                <field name="name"/>
                                <field name="instructor_id"/>
                              </group>
                              <group string="Jadwal">
                                <field name="start_date"/>
                                <field name="duration"/>
                                <field name="seats"/>
                              </group>
                            </form>
                        </field>
                    </page>
                </notebook>
            </sheet>
        </form>
    </field>
</record>
```

```
<!-- ### Membuat Tampilan Form Sesi ### -->

<record model="ir.ui.view" id="sesi_form_view">
    <field name="name">training.sesi.form</field>
    <field name="model">training.sesi</field>
    <field name="arch" type="xml">
        <form string="Sesi Form">
        <sheet>
            <group>
                <group string="Informasi">
                    <field name="course_id"/>
                    <field name="name"/>
                    <field name="instructor_id"/>
                </group>
                <group string="Jadwal">
                    <field name="start_date"/>
                    <field name="duration"/>
                    <field name="seats"/>
                </group>
                <group string="Peserta" colspan="2">
                    <field name="attendee_ids" nolabel="1"/>
                </group>
            </group>
        </sheet>
        </form>
    </field>
</record>
```

Hasilnya seperti gambar berikut :



INHERITANCE

Konsep **inheritance** adalah salah satu konsep **OOP** (Pemrograman Berorientasi Object), untuk penjelasan lanjutnya bisa baca disini.

Odoo menyediakan 2 jenis inheritance, yaitu :

1. Merubah 'behavior' suatu object dari modul lain (class anak). Diantaranya adalah :

- Menambahkan fields dan methods baru
- Mengoverride attribute atau method suatu field
- Mengoverride method parent
- Menambah constraints baru

2. Delegasi, yaitu memungkinkan untuk menghubungkan setiap record suatu model kepada record model parent dan memberikan akses kepada field record parent tersebut (hanya field saja).

Point pertama menggunakan attribute **_inherit**, sedangkan yang kedua menggunakan **_inherits**. Tujuan utama dari fitur ini adalah kita ingin **menambah** atau **merubah** hal-hal diatas pada object yang telah ada yaitu **dengan membuat modul baru**. Karna kita di 'haramkan' untuk langsung mengedit modul existing seperti sale, purchase, stock, dll. Ada banyak hal yang bisa kita inherite di Odoo, diantaranya model (python), view (xml), access right (csv), sampai report (qweb).

Selanjutnya kita akan menginherite object **res_partner**. Object ini digunakan oleh beberapa menu (view) yaitu menu Customers dan Vendors, hal ini bisa kita lakukan dengan bantuan fitur **Action** seperti penjelasan pertemuan sebelumnya. Kita juga telah melakukan hal yang sama, yaitu object **res_partner** kita gunakan untuk 2 hal yaitu sebagai **Instruktur** dan sebagai **Peserta**. Jika Customer dan Vendor menggunakan field **customer (Is a Customer)** dan **supplier (Is a Vendor)** sebagai pembeda, maka kita akan menggunakan field **instructor (Instruktur)** untuk membedakannya.

Setiap file python bisa mengandung banyak class, tidak seperti java dan lainnya yang hanya memiliki 1 class dalam setiap file. Tetapi dalam tutorial kali ini, saya ingin memisahkan class yang akan kita inherit kedalam file baru agar terlihat rapi dan dapat dijalankan pada windows. Baik kita mulai dengan membuat file baru yaitu **partner.py** (samakan pathnya dengan **models.py**) yang isinya seperti berikut :

```
from odoo import models, fields, api

class Partner(models.Model):
    _inherit = 'res.partner'

    instructor = fields.Boolean("Instruktur")
    session_ids = fields.Many2many('training.sesi',string="Menghadiri
Sesi", readonly=True)
```

Pada coding diatas kita menginherite object **res_partner** dengan tujuan untuk menambahkan 2 field baru yaitu **instructor** dan **session_ids**. Field **instructor** berfungsi sebagai filter antara instruktur dan peserta. Sedangkan field **session_ids** berfungsi sebagai relasi dan menjadikan informasi bahwa partner terkait telah mengikuti sesi apa saja (field akan terisi otomatis dari menu sesi).

Dan pastikan kita telah mengimport file **partner.py** pada file **__init__.py** di dalam folder **models**. Seperti berikut :

```
from . import models
from . import partner
```

Lalu kita buat file xml nya dengan nama **partner.xml** (samakan pathnya dengan **views.py**) agar field dapat di tampilkan, yang isinya seperti berikut :

```
<!-- Menambahkan field pada view form partner -->

<record model="ir.ui.view" id="instruktur_form_view">
<field name="name">res.partner.instruktur</field>
<field name="model">res.partner</field>
<field name="inherit_id" ref="base.view_partner_form"/>
<field name="arch" type="xml">
        <notebook position="inside">
            <page string="Sessions">
                <group>
                      <field name="instructor"/>
                      <field name="session_ids"/>
                </group>
            </page>
        </notebook>
</field>
</record>

<!-- Membuat action baru untuk menu baru-->

<record model="ir.actions.act_window" id="kontak_list_action">
        <field name="name">Kontak</field>
        <field name="res_model">res.partner</field>
        <field name="view_mode">tree,form</field>
</record>

<!-- ### Membuat menu title baru ### -->

<menuitem id="konfigurasi_menu" name="Konfigurasi"
          parent="main_training_odoo_menu"/>

<!-- ### Membuat Submenu Kontak ### →

<menuitem id="kontak_menu" name="Kontak"
          parent="konfigurasi_menu"
          action="kontak_list_action"/>
```

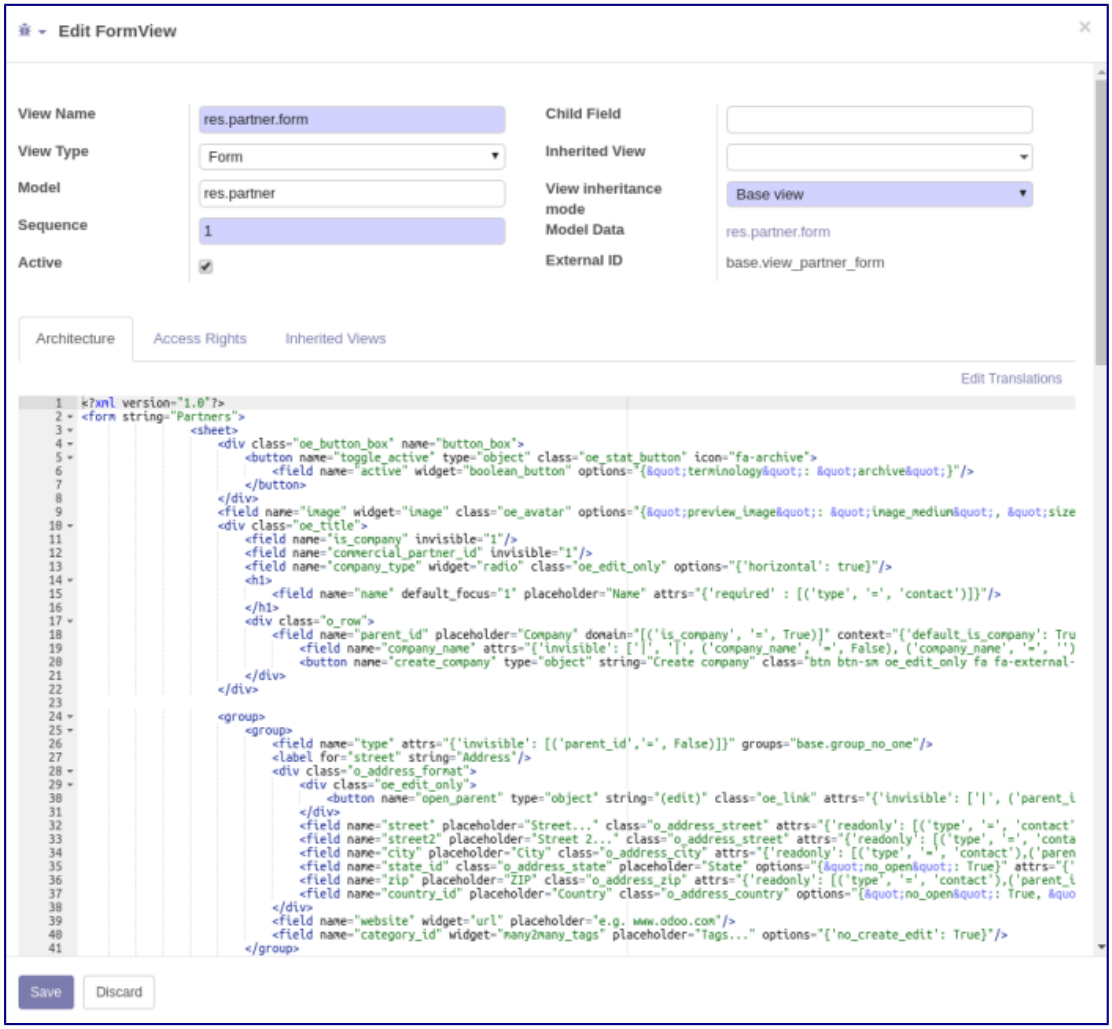
Pada bagian pertama file xml diatas, kita melakukan inherit dari sebuah tampilan atau view. Jika pada object/model kita menggunakan nama objectnya pada attribute **_inherit**, sedangkan untuk menginherit view kita menggunakan **id view** terkait pada field **inherit_id**. Caranya terletak pada baris ini :

```
<field name="inherit_id" ref="base.view_partner_form"/>
```

Bagaimana kita mengetahui **id view** tersebut ? Yaitu dengan :

1. Aktifkan developer mode
2. Pilih/buka view yang di inginkan untuk kita update
3. Klik icon '**bug**' pada pojok kanan atas, lalu pilih '**Edit FormView**'

Hasilnya seperti gambar dibawah ini :



Dari gambar diatas kita bisa banyak mengambil informasi, untuk inherite kita gunakan value dari field '**External ID**'. Setelah kita mengetahui id viewnya, selanjutnya kita sisipkan field tambahan yang telah dibuat sebelumnya. Ada 2 cara untuk melakukan hal tersebut sebagaimana contoh dibawah ini :

```
<xpath expr="//field[@name='description']" position="after">
        <field name="instructor" />
</xpath>

<!-- ATAU-->

<field name="description" position="after">
        <field name="instructor" />
</field>
```

1. Menggunakan **xpath**
2. Menggunakan **element** terkait. Bisa berupa **field, button, notebook, page, groups,** dll

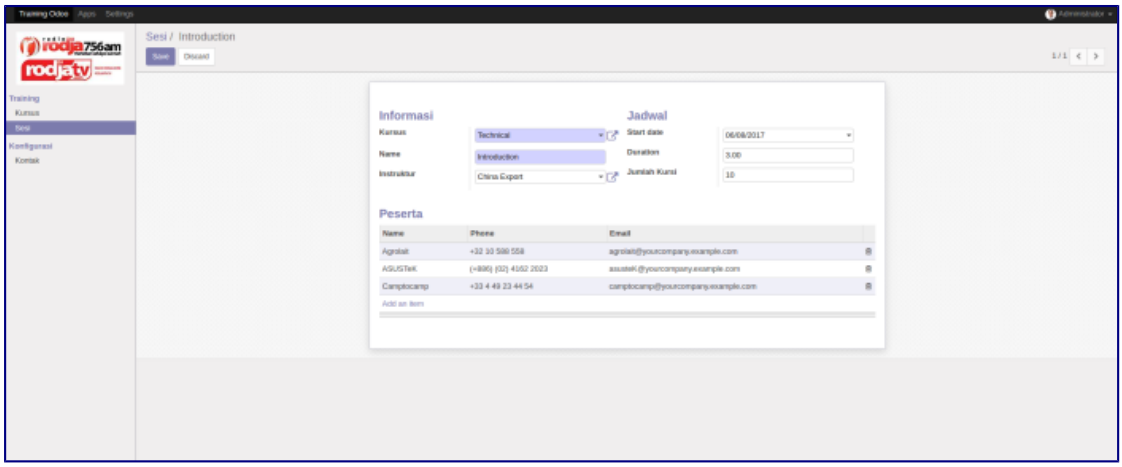
Kedua cara diatas menghasilkan tampilan yang sama dan menggunakan attribute yang sama yaitu **position**. Attribute position memiliki beberapa value yaitu :

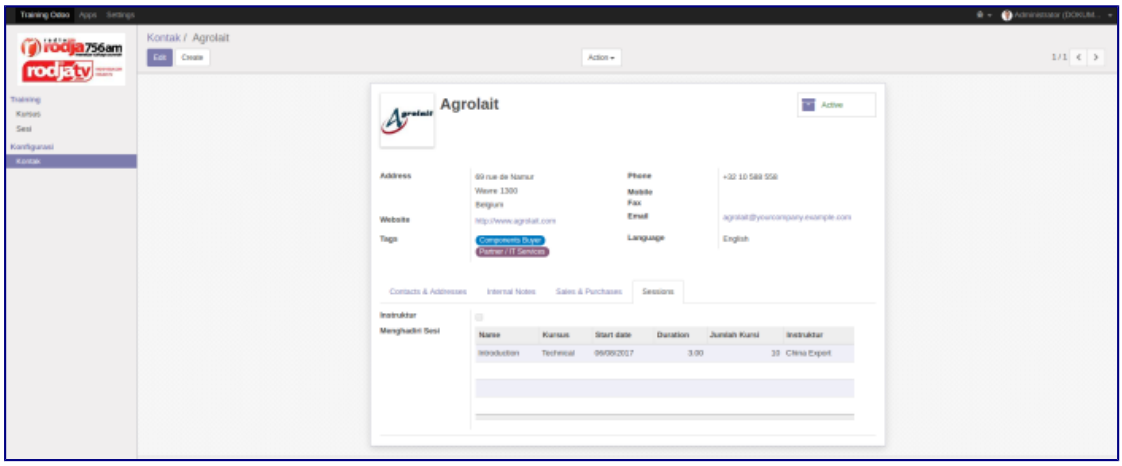
a. **after** = menambahkan element **setelah** element yang dituju
b. **before** = menambahkan element **sebelum** element yang dituju
c. **replace** = menambahkan element dengan **menghapus** element yang dituju
d. **inside** = menambahkan element **di dalam** element yang dituju
e. **attributes** = menambahkan attribute dari element yang dituju. Misal : **readonly, string, editable, domain, options,** dll

Pada contoh di atas kita memasukan sebuah element **page** ke dalam element **notebook**. Kemudian pastikan kita telah mengupdate file **partner.xml** pada file **__manifest__.py** seperti berikut :

```
...

    # always loaded
    'data': [
        # 'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
        'views/partner.xml',
    ],


...
```

Jika semua telah di lakukan, **restart service** dan **upgrade modul**, hasilnya akan terlihat seperti berikut :





DOMAIN

Di odoo, domain berfungsi sebagai filter suatu kondisi pada record, sebagaimana halnya syntax where pada sql. Domain biasa di gunakan pada field-field relasi seperti Many2one, One2many, dan Many2many. Domain berbentuk list python (array), yang isinya kumpulan tuple dan setiap tuple mengandung pasangan antara nama field dan values nya. Contohnya seperti ini :

```
domain = [('sale_ok', '=', True), ('product_type', '=', 'service'),
('unit_price', '>=', 1000)]
```

Domain bisa kita tentukan di model dan view. Domain juga memiliki beberapa operator yaitu **&** (AND), **|** (OR) dan **!** (NOT). Jika kita tidak mendefiniskan, maka defaultnya operator **&** (AND). Cara menggunakannya kita set di awal domain, contohnya seperti ini :

```
domain = ['|', ('product_type', '=', 'service'), ('unit_price', '>',
1000)]

# Searching product yang tipenya service ATAU harganya diatas 1000

domain = ['&', ('product_type', '=', 'service'), ('unit_price', '<',
1000)]

# Searching product yang tipenya service DAN harganya dibawah 1000

domain = ['|', ('product_type', '=', 'service'), '!', '&',
('unit_price', '>=', 1000), ('unit_price', '<=', 2000)]

# Searching product yang tipenya service ATAU harganya yang BUKAN di
antara 1000 dan 2000

domain = [('name','=','ABC'), ('language.code','!=','en_US'), '|',
('country_id.code','=','be'), ('country_id.code','=','de')]

# Searching partner yang namanya ABC DAN bahasanya SELAIN en_US
(english) DAN berasal dari negara be (belgia) ATAU de (jerman)
```

Jika kita perhatikan, diantara nama field dan valuenya ada operator. Diantara operator yang di sediakan odoo adalah :

= : sama dengan

!= : tidak sama dengan

> : lebih besar dari

>= : lebih besar dari atau sama dengan

< : lebih kecil dari

<= : lebih kecil dari atau sama dengan

=? : menghasilkan nilai True jika valuenya None atau False

=like : sama seperti operator =, bisa ditambah valuenya dengan underscore (_) -imbuhan 1 karakter- dan persentage (%) -imbuhan banyak karakter-

like : yang mengandung (**case sensitive**)
[('name', 'like', 'apa')], hasilnya record = 'apa', 'bapak', 'apakah', dst. Bukan 'Apa', 'APA', dst
[('name', 'like', 'apa')] **sama seperti** [('name', '=like', '%apa%')]

not like : tidak mengandung (**case sensitive**)

ilike : yang mengandung (tidak case sensitive)
[('name', 'ilike', 'apa')], hasilnya record = 'apa', 'Apa', 'APA', 'Apakah', 'bapak', dst.
[('name', 'ilike', 'apa')] **sama seperti** [('name', '=ilike', '%apa%')]

not ilike : tidak mengandung (tidak case sensitive)

=ilike : gabungan antara operator = dan ilike
[('name', '=ilike', 'apa')], hasilnya record = 'apa', 'APA', 'Apa', 'APa', ApA', 'aPa', dst. Bukan 'Apakah', 'papah', 'Bapak', dst

in : menjadi bagian value dari list/array

not in : tidak menjadi value bagian dari list/array

child_of : anak dari _parent_name (record yang hirarki)

Penambahan **i** pada like menjadikan tidak case sensitive (mengabaikan huruf besar kecil), sedangkan penambahan = menjadikan hasil querynya sesuai jumlah digit value parameternya

Sekarang kita aplikasikan pada materi training kita. Sebelumnya kita telah membuat field **instructor** (**Instruktur**) pada object **'training_sesi'** yang merefer ke menu **Kontak**. Tidak ada filter sama sekali disana, sehingga valuenya tidak memiliki perbedaan dengan field **attendee_ids** (**Peserta**). Oleh karna itu, kita akan memberikan filter pada field **instructor** (**Instruktur**) seperti berikut :

```
class Sesi(models.Model):
    ...

    instructor_id = fields.Many2one('res.partner',
string="Instruktur", domain=[('instructor', '=', True)])
    attendee_ids = fields.Many2many('res.partner', string="Peserta",
domain=[('instructor', '=', False)])

    ...
```

Setelah kita restart service odoo, maka buatlah beberapa kontak baru, lalu centang cekbox **Instruktur** kepada salah satunya. Maka kita akan melihat perbedaan saat pemilihan **Intruktur** dan **Peserta**. Jika **Instruktur** di centang, maka ia tidak dapat di pilih menjadi **Peserta**, begitu juga sebaliknya. Kemudian pada form **Kontak** di bawah field **Instruktur** ada tabel **Menghadiri Sesi**, kita ingin tabel ini mengikuti kondisi nilai field **Instruktur**, ketika dia di centang maka table **Menghadiri Sesi** harus menghilang (invisible), karna instrukur tidak bisa menjadi peserta. Caranya dengan menggunakan attribut **attrs** seperti dibawah ini :

```
<record model="ir.ui.view" id="instruktur_form_view">
    <field name="name">res.partner.instruktur</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>
    <field name="arch" type="xml">
        <notebook position="inside">
          <page string="Sessions">
            <group>
              <field name="instructor"/>
              <field name="session_ids"
    attrs="{'invisible': [('instructor', '=', True)]}"/>
            </group>
          </page>
        </notebook>
    </field>
</record>
```

Update modul kita dengan code xml diatas (perhatikan id xml nya). Lakukan restart dan upgrade modul **training_odoo**, lalu refresh browser dan silahkan di coba. Attribut **attrs** diantaranya :

a. **invisible** : field akan menjadi tidak terlihat jika kondisi terpenuhi
b. **readonly** : field akan menjadi tidak bisa di edit jika kondisi terpenuhi
c. **required** : field akan menjadi mandatory jika kondisi terpenuhi

Attribut diatas dapat digunakan bersamaan seperti contoh :

```
<field name="partner_id" attrs="{
      'invisible': [('communication', '=', True)],
      'required': [('payment_type', 'in', ('inbound', 'outbound'))],
      'readonly': [('state', '!=', 'draft')]}}" />
```

Pada contoh sebelumnya kita telah membuat domain field Instruktur, sekarang kita akan tambahkan lagi domainnya agar lebih complex. Silahkan di update :

```
class Sesi(models.Model):
    ...

    instructor_id = fields.Many2one('res.partner',
string="Instruktur", domain=['|', ('instructor', '=', True),
('category_id.name', 'ilike', "Pengajar")])

    ...
```

Domain yang kita tambahkan di atas artinya, field **Instuktur** nilainya dapat kita pilih **JIKA** kontak telah di centang cekbox Instrukturnya **ATAU** kontak terkait termasuk **Category** yang memiliki nama '**Pengajar**'. **Category** kontak bisa kita tentukan pada field **Tags**.

Karna kita belum membuat category / tags kontak, sekarang kita buat terlebih dahulu agar kita bisa pilih untuk mencoba domain yang kita buat. Tambahkan xml berikut pada file **partner.xml** :

```
        <menuitem id="kontak_tags_menu" name="Kontak Tags"
                  parent="konfigurasi_menu"
                  action="base.action_partner_category_form"/>

        <record model="res.partner.category" id="teacher1">
                <field name="name">Pengajar / Basic</field>
        </record>

        <record model="res.partner.category" id="teacher2">
                <field name="name">Pengajar / Advanced</field>
        </record>
```

Perhatikan coding diatas, terdiri dari 3 bagian. Pertama, membuat menu **Kontak Tags**. Kita bisa langsung **HANYA** membuat menu jika action dan view lainnya sudah di definiskan pada modul lain, karna category / tags merupakan object default odoo (res_partner_category) sebagaimana kontak (res_partner) maka kita cukup **menginheritnya** saja. Kecuali kita ingin menambahkan domain, view, dll pada action, maka kita kita perlu membuat ulang actionnya. Perintah inherite berada pada attribut :

```
action="base.action_partner_category_form"
```

Artinya menu tags yang kita buat ketika di klik, maka akan memanggil action atau event yang ada di modul **base** dengan id xml **action_partner_category_form**. Kita juga bisa menginherite attribute **parent** jika kita ingin meletakan menu kita di bawah menu terkait.

Bagian kedua dan ketiga adalah proses insert record / baris ke menu Category / Tags. Selain input manual user, kita juga bisa menambahkan record dari xml. Sekarang silahkan dicoba untuk memilih salah satu dari 2 kondisi yaitu centang cekbox Instruktur atau tentukan tags Pengajar.

Selanjutnya kita akan membuat field **compute** dengan nama **'taken_seats'** seperti berikut :

```
class Sesi(models.Model):

    ...

    taken_seats = fields.Float(string="Kursi Terisi",
                               compute='_taken_seats')

    ...

    @api.depends('seats', 'attendee_ids')
    def _taken_seats(self):
        for r in self:
            if not r.seats:
                r.taken_seats = 0.0
            else:
                r.taken_seats = 100.0 * len(r.attendee_ids) / r.seats
```

Field compute diatas berfungsi untuk menghitung persentase antara jumlah kursi yang disediakan dengan total peserta yang ada. Method yang digunakan berada pada attribute compute yaitu **'_taken_seats'**. Method itu mengandung decorator **@api.depends()**, artinya method akan selalu dijalankan ketika ada perubahan nilai pada 2 field terkait yaitu **'seats'** dan **'attendee_ids'**.

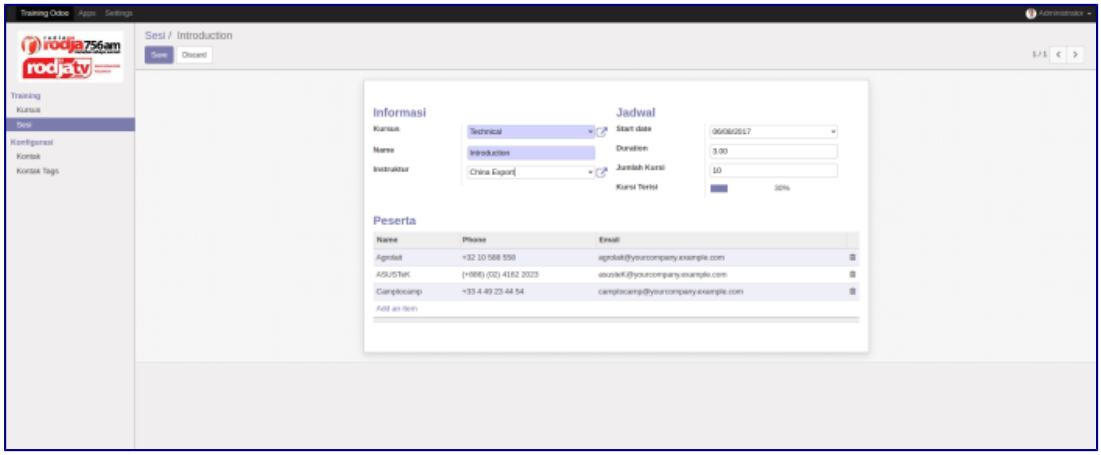
Penjelasan decorator penulis pernah sampaikan pada artikel ini.

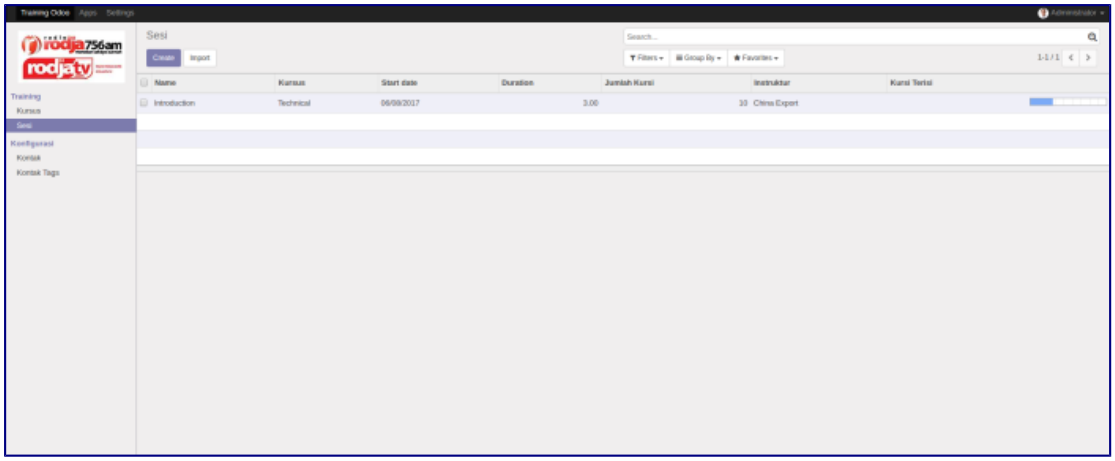
Kemudian kita tampilkan field compute tersebut pada list view dan form view object training_sesi. Seperti berikut (perhatikan id xmlnya) :

```
        <record model="ir.ui.view" id="sesi_tree_view">
            <field name="name">training.sesi.tree</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                <tree string="Sesi List">
                    <field name="name"/>
                    <field name="course_id"/>
                    <field name="start_date"/>
                    <field name="duration"/>
                    <field name="seats"/>
                    <field name="instructor_id"/>
                    <field name="taken_seats" widget="progressbar"/>
                </tree>
            </field>
        </record>
```

```
<record model="ir.ui.view" id="sesi_form_view">
    <field name="name">training.sesi.form</field>
    <field name="model">training.sesi</field>
    <field name="arch" type="xml">
        <form string="Sesi Form">
            <sheet>
            <group>
                <group string="Informasi">
                    <field name="course_id"/>
                    <field name="name"/>
                    <field name="instructor_id"/>
                </group>
                <group string="Jadwal">
                    <field name="start_date"/>
                    <field name="duration"/>
                    <field name="seats"/>
                    <field name="taken_seats"
                           widget="progressbar"/>
                </group>
                <group string="Peserta" colspan="2">
                    <field name="attendee_ids" nolabel="1"/>
                </group>
            </group>
            </sheet>
        </form>
    </field>
</record>
```

Hasilnya seperti foto berikut (perhatikan field '**Kursi Terisi**') :





Ada banyak jenis widget yang odoo sediakan dan masing-masing widget memiliki tampilan yang berbeda-beda. Untuk menggunakannya kita bisa set pada attribut **widget**, diantaranya :

char
id
email
url
text
char_domain
date
datetime
selection
radio
reference
boolean
boolean_button
toggle_button
float
percentpie
integer
float_time
progressbar
image
binary
statusbar
monetary

priority
kanban_state_selection
many2many_tags
kanban_label_selection
handle
statinfo
timezone_mismatch
label_selection
ace

Selanjutnya kita akan tentukan nilai default pada field '**start_date**' seperti contoh berikut :

```
class Sesi(models.Model):

    ...

    start_date = fields.Date(default=fields.Date.today)

    ...
```

Terakhir, saat kita membuka form **Kursus**, maka disana ada tabel **Sesi**. Untuk menambahkan sesi kita harus mengklik text **"Add an item"** sebagaimana semua tabel di odoo. Setelah kita klik, maka akan terbuka **pop up** form dari Sesi. Agar kita tidak perlu ada **pop up** dan cukup menambahkannya langsung pada table, maka kita gunakan attribut **editable**. Attribute **editable** memiliki value **top** dan **bottom**, silahkan mengupdate view xml seperti di bawah ini :

```
<record model="ir.ui.view" id="kursus_form_view">
      <field name="name">training.kursus.form</field>
      <field name="model">training.kursus</field>
      <field name="arch" type="xml">
          <form string="Kursus Form">
              <sheet>
                  <group>
                      <field name="name"/>
                      <field name="responsible_id"/>
                  </group>
                  <notebook>
                      <page string="Keterangan">
                          <field name="description"/>
                      </page>
                      <page string="Sesi">
                          <field name="session_ids">
                              <tree string="Daftar Sesi"
                                    editable="bottom">
                                  <field name="name"/>
                                  <field name="instructor_id"/>
                              </tree>
```

```
                                <form>
                                  <group string="Informasi">
                                    <field name="name"/>
                                    <field name="instructor_id"/>
                                  </group>
                                  <group string="Jadwal">
                                      <field name="start_date"/>
                                      <field name="duration"/>
                                      <field name="seats"/>
                                  </group>
                                </form>
                            </field>
                        </page>
                    </notebook>
                </sheet>
            </form>
        </field>
    </record>
```

Silahkan di restart + upgrade modul dan lihat perbedaannya.

CONSTRAINT

InsyaAllah kita lanjutkan kembali serial tutorial technical ini. Pada pertemuan kali ini, insyaAllah hal yang pertama kita bahas adalah fitur **onchange**. Mekanisme **onchange** berbentuk sebuah method yang berfungsi memberikan cara 'komunikasi' kepada user setiap kali mengupdate nilai field suatu form TANPA menyimpan apapun ke database. Pada umumnya onchange digunakan untuk 3 hal berikut :

1. **value** = mengisi nilai suatu field berdasarkan inputan user
2. **domain** = memberikan domain suatu field berdasarkan inputan user
3. **warning** = memberikan pesan informasi kepada user ketika proses penginputan data

Kali ini kita akan gunakan point 1 dan 3 pada object **training_sesi** kita. Silahkan update file model sesi seperti berikut :

```
class Sesi(models.Model):
    ...


    @api.onchange('seats', 'attendee_ids')
    def _verify_valid_seats(self):
        if self.seats <= 0: # cek nilai field seats, jika dibawah 0
(negatif), maka masuk kondisi if
            return {
                    'value': {
                                'seats': len(self.attendee_ids) or 1
                               # mengisi field seats dengan jumlah
peserta atau 1
                             },
                    'warning': {
                                'title': "Nilai Jumlah Kursi Salah",
                                'message': "Jumlah Kursi Tidak Boleh
Negatif"
                                }
                    }

        if self.seats < len(self.attendee_ids): # cek nilai field
seats (jumlah kursi) apakah lebih kecil dari field attendee_ids
(jumlah peserta), jika iya maka masuk kondisi if
            return {
                    'value': {
                                'seats': len(self.attendee_ids) #
mengisi field seats dengan nilai jumlah peserta
                             },
                    'warning': {
                                'title':"Peserta Terlalu Banyak",
                                'message':"Tambahkan Kursi atau
Kurangi Peserta"
                                }
                    }
```

Ada perubahan signifikan antara api lama dan baru pada fitur onchange ini, jika api lama maka kita harus mendefiniskan nama method onchange pada setiap field yang dinginkan di view file xml. Dengan api baru ini, kita tidak perlu repot untuk melakukan hal itu, cukup gunakan decorator **@api.onchange** dan masukan field mana saja yang kita inginkan untuk mengalami onchange terkait. Contoh diatas kita memberikan fitur onchange pada field '**seats**' dan '**attendee_ids**'.

Jadi saat 2 field tersebut mengalami perubahan nilai oleh user, maka method **_verify_valid_seats()** akan dijalankan.

Sebenarnya field **compute** secara otomatis mengalami fitur **onchange** tetapi dengan keterbatasan yaitu mengupdate field saja. Sedangkan untuk pengecekan nilai inputan user, dll kita harus menggunakan onchange. Pada contoh diatas kita akan mengecek inputan user, diantaranya :

1. Pengecekan jumlah kursi, apakah dibawah 0 atau tidak. Jika 0 dan dibawah 0, maka jumlah kursi akan terisi 1 atau sebanyak jumlah peserta beserta pesan errornya

2. Pengecekan apakah jumlah kursi lebih sedikit dari jumlah peserta. Jika iya maka akan ada pesan error dan nilai jumlah kursi di isi sesuai jumlah peserta

Jika kita hanya mereturn **value** saja tanpa **warning** atau **domain**, maka di v10 ini kita bisa menggunakan perintah **self.update()** seperti contoh berikut :

```
self.update({'nama_field': 'nilai'})
```

```
self.update({
             'partner_id': self.partner_id.id,
             'fiscalyear_id': self.partner_id.fiscalyear_id.id
            })
```

Silahkan dicoba dan cek hasilnya.

Selanjutnya kita akan bahas **constraints**. Ada 2 jenis untuk model constraints ini, yaitu :

1. Python Constraints

Sebuah constraints yang dibuat oleh method python. Biasanya digunakan untuk pengecekan nilai antar field pada record yang sama. Cara menggunakannya dengan decorator **@api.constrains()** dengan parameter nama field. Contohnya :

```
@api.constrains('umur')
def _cek_umur(self):
    for r in self:
        if r.umur < 17:
            raise ValidationError("Umur anda masih terlalu muda")

#  ATAU

@api.constrains('name', 'code')
def _cek_name_code(self):
    for r in self:
        if r.name == r.code:
            raise ValidationError("Name dan Code tidak boleh sama")
```

2. SQL Constraints

Sebuah constraints yang dibuat seperti syntax SQL. Biasanya digunakan untuk pengecekan nilai antar field pada record yang sama atau record yang berbeda. Cara menggunakannya dengan attribute model **_sql_constraints** dengan parameter tuple berisi **nama_constraints, definis_sql,** dan **pesan_error**. Contohnya :

```
_sql_constraints = [
                    ('nama_constraints_bebas', 'CHECK(name !=
description)', 'Pesan error pengecekan field name dan description
pada record yang sama')
]

# ATAU

_sql_constraints = [
                    ('nama_constraints_bebas', 'UNIQUE(name)', 'Pesan
error pengecekan field name pada semua record (harus unik)')
]

# ATAU


_sql_constraints = [
                    ('name_description_check', 'CHECK(name !=
description)', 'The title of the course should not be the
description'),
                    ('name_unique', 'UNIQUE(name)', 'The course title
must be unique')
]
```

Pada contoh diatas kita bisa menggunakan constraints dengan perintah **CHECK(), UNIQUE(),** dll sebagaimana syntax SQL. Kita juga bisa mendefiniskan beberapa constraints sekaligus.

Sekarang kita buat constraints pada object training kita, yaitu Kursus() dan Sesi() seperti berikut :

```
class Kursus(models.Model):
    ...

    _sql_constraints = [
            ('name_description_cek', 'CHECK(name != description)',
'Judul kursus dan keterangan tidak boleh sama '),
            ('name_unik', 'UNIQUE(name)', 'Judul kursus harus unik')
    ]


class Sesi(models.Model):
    ...


    @api.constrains('instructor_id', 'attendee_ids')
    def _check_instructor_not_in_attendees(self):
        for r in self:
            if r.instructor_id and r.instructor_id in r.attendee_ids:
# Jika field instructor_id (Instruktur) diisi DAN instructor_id ada
di tabel attendee_ids (Peserta), maka munculkan pesan error
                raise exceptions.ValidationError("Seorang instruktur
tidak boleh menjadi peserta")
```

Pada contoh diatas kita membuat 2 constraints, pertama pada **class Kursus()**, yaitu constraints dengan tipe **SQL Constraints**, fungsinya untuk mengecek nilai field name (Judul) dan description (Keterangan) tidak boleh sama dalam 1 record.

Lalu pengecekan ke semua record bahwa field name (Judul Kursus) harus unik. Kedua adalah **Python Constraints** pada **class Sesi()**, fungsinya untuk pengecekan bahwa seorang Instruktur tidak boleh menjadi Peserta.

Setelah kita menambahkan contraints unik untuk judul kursus, maka hal ini bisa membuat masalah ketika kita menggunakan fitur **duplicate** pada object kursus.

Untuk itu kita harus mengoverride **method duplicate** yaitu **copy()** dan 'merevisi' dengan beberapa tambahan, seperti berikut :

```
class Kursus(models.Model):
    ...


    @api.multi
    def copy(self, default=None):
        default = dict(default or {})
        copied_count = self.search_count([('name', '=like', "Copy of
{}%".format(self.name))]) # Searching judul kursus yang sama dan
menyimpannya pada variable copied_count

        if not copied_count: # Cek isi variable copied_count (hasil
searching)
            new_name = "Copy of {}".format(self.name) # Jika proses
searching judul yang sama tidak ditemukan, maka judul baru akan
dikasih imbuhan 'Copy of'
        else:
            new_name = "Copy of {} ({})".format(self.name,
copied_count) # Jika proses searching judul yang sama ditemukan, maka
judul baru akan dikasih imbuhan 'Copy of' dan angka terakhir
duplicate

        default['name'] = new_name # Mereplace value field name
dengan yang sudah di sesuaikan
        return super(Kursus, self).copy(default)
```

Fungsi dari **overriding** diatas adalah untuk menambahkan imbuhan setiap record kursus di duplicate, sehingga judul kursus selalu unik.

VIEWS

Insya Allah kali ini kita akan membahas tentang pembuatan berbagai view, seperti list, form, calendar, kanban, graphs, dll.

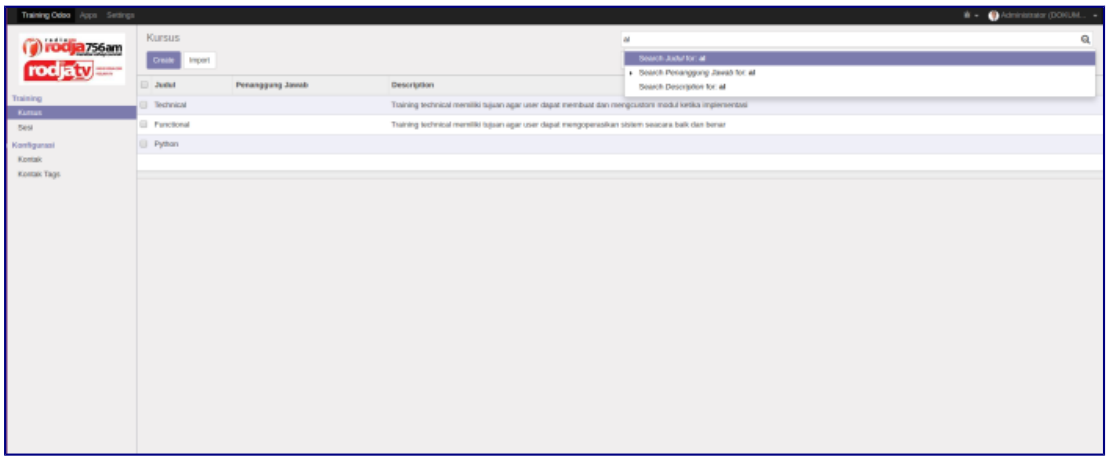
A. Search Views

Untuk yang pertama kita akan membahas **Search View**. Perbedaan mendasar dari versi sebelumnya adalah fitur **Group By** otomatis ada pada **Advanced Search** untuk semua field sebagaimana filter. Secara default field **name** akan menjadi filter jika kita tidak membuat search view. Sekarang kita tambahkan search view object Kursus kita seperti berikut :

(tambahkan coding dibawah ini DI ATAS action view kursus)

```
<!-- ### Membuat Search View Kursus ### -->

<record model="ir.ui.view" id="kursus_search_view">
    <field name="name">training.kursus.search</field>
    <field name="model">training.kursus</field>
    <field name="arch" type="xml">
        <search>
            <field name="name"/>
            <field name="responsible_id"/>
            <field name="description"/>
        </search>
    </field>
</record>
```

Setelah kita tambahkan code diatas, maka search boxnya akan mengalami perubahan seperti berikut :



Kemudian kita tambahkan **custom filter** dan **custom group** seperti berikut :

```
<!-- ### Membuat Search View Kursus ### -->

<record model="ir.ui.view" id="kursus_search_view">
    <field name="name">training.kursus.search</field>
    <field name="model">training.kursus</field>
    <field name="arch" type="xml">
        <search>
            <field name="name"/>
            <field name="responsible_id"/>
            <field name="description"/>
            <filter name="my_courses" string="Kursus Saya"
                    domain="[('responsible_id', '=', uid)]"/>
            <group string="Group By">
                <filter name="by_responsible"
                string="Penanggung Jawab"
                context="{'group_by': 'responsible_id'}"/>
            </group>
        </search>
    </field>
</record>
```

Sehingga menu dropdown **Filters** dan **Group By** pada Advanced Search bertambah pilihannya.

B. Tree/List View

Kita telah membuat tampilan tree view pada kedua object Kursus dan Sesi. Sekarang kita akan berikan 'dekorasi' yang dimiliki oleh view ini. Ada beberapa attribute tree/list view yang bisa kita gunakan, diantaranya :

Merubah font menjadi bold

decoration-bf="[nama_field][operator][nilai_field]"

Merubah font menjadi italic

decoration-it="[nama_field][operator][nilai_field]"

Merubah font dengan berbagai warna

decoration-info="[nama_field][operator][nilai_field]"

decoration-muted="[nama_field][operator][nilai_field]"

decoration-danger="[nama_field][operator][nilai_field]"

decoration-primary="[nama_field][operator][nilai_field]"

decoration-success="[nama_field][operator][nilai_field]"

decoration-warning="[nama_field][operator][nilai_field]"

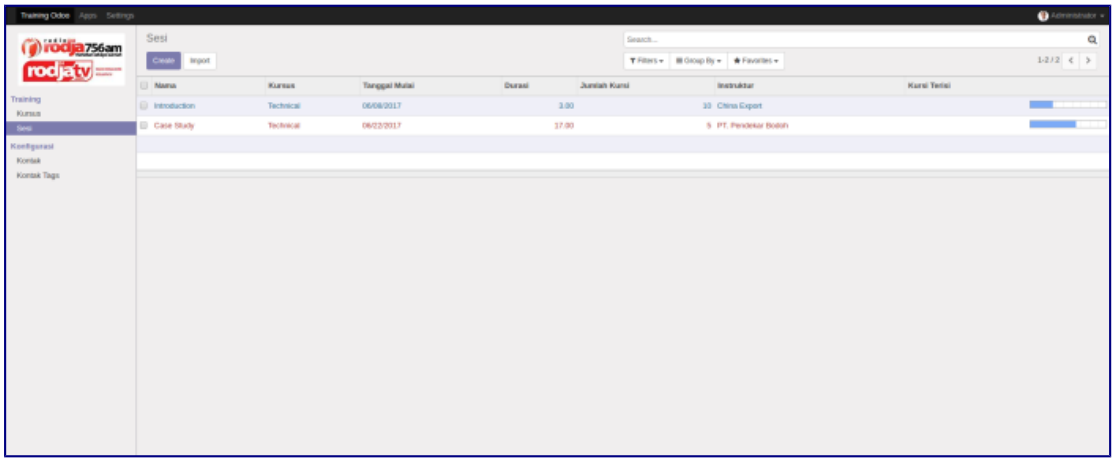
Contohnya seperti ini :

```
<tree string="Quotation"
      decoration-bf="state=='confirm'"
      decoration-info="state=='draft'"
decoration-warning="date_action and (date_action &lt; current_date)">
        <field name="name"/>
        <field name="date_action"/>
        <field name="state"/>
</tree>
```

Selanjutnya kita tambahkan 'dekorasi' warna berdasarkan field durasi, yaitu jika durasi harinya kurang dari 5 hari maka berwarna biru dan jika durasinya lebih dari 15 hari maka berwarna merah. Kita tambahkan pada object Sesi seperti berikut :

```
        <!-- ### Membuat Tampilan Tree/List Sesi ### -->

        <record model="ir.ui.view" id="sesi_tree_view">
            <field name="name">training.sesi.tree</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                <tree string="Sesi List"
                      decoration-info="duration&lt;5"
                      decoration-danger="duration&gt;15">
                                <field name="name"/>
                                <field name="course_id"/>
                                <field name="start_date"/>
                                <field name="duration"/>
                                <field name="seats"/>
                                <field name="instructor_id"/>
                                 <field name="taken_seats"
                                        widget="progressbar"/>
                </tree>
            </field>
        </record>
```

Hasilnya akan terlihat seperti ini :



Selain hal diatas, tree view juga memiliki attribute-attribute di bawah ini :

```
<!-- Menghilangkan pop up form saat create dan edit -->
<tree string="" editable="bottom">
<!-- Mensorting tabel berdasarkan nama field -->
<tree string="" default_order="name desc">
<!-- Menghilangkan tombol create -->
<tree string="" create="false">
<!-- Menghilangkan tombol edit -->
<tree string="" edit="false">
<!-- Menghilangkan tombol delete -->
<tree string="" delete="false">
```

C. Calendar Views

Calendar view berfungsi untuk menampilkan record dalam bentuk calendar. Kita juga bisa langsung membuat record dengan memilih salah satu atau beberapa tanggal (drag) yang kita inginkan seperti halnya pembuatan Leaves (cuti) dll. Selain itu, calendar view juga memudahkan kita untuk mengedit field tanggal dengan fitur drag and drop, tanpa harus masuk kedalam form view terkait.

Sekarang kita akan tambahkan calendar view pada object Sesi, ikuti langkah-langkah berikut :

```
from datetime import timedelta
```

Kita import library baru yaitu **timedelta**. Library ini kita butuhkan untuk pembuatan field **end_date**, dimana field ini akan menghitung secara otomatis tanggal waktu berakhir sesi dari tanggal mulai ditambah durasi hari.

```
class Sesi(models.Model):
    ...

    end_date = fields.Date(string="Tanggal Selesai", store=True,
compute='_get_end_date', inverse='_set_end_date')


    @api.depends('start_date', 'duration')
    def _get_end_date(self):
        for r in self:
            # Pengecekan jika field duration & start_date tidak
diisi, maka field end_date akan di update sama seperti field
start_date
            if not (r.start_date and r.duration):
                r.end_date = r.start_date
                continue

            # Membuat variable start yang isinya tanggal dari field
start_date
            start = fields.Datetime.from_string(r.start_date)

            # Membuat variable duration yang isinya durasi hari dari
field duration
            # Durasi hari dikurangi 1 detik agar start_date masuk
kedalam durasi hari
            duration = timedelta(days=r.duration, seconds=-1)

            # Mengupdate field end_date dari perhitungan variabel
start ditambah variabel duration
            r.end_date = start + duration

    def _set_end_date(self):
        for r in self:
            if not (r.start_date and r.end_date):
                continue
  # Membuat variable start_date yang isinya dari field start_date
            start_date = fields.Datetime.from_string(r.start_date)

  # Membuat variable end_date yang isinya tanggal dari field end_date
            end_date = fields.Datetime.from_string(r.end_date)

            # Mengupdate field duration (jika ada perubahan dari
field end_date) dari perhitungan variabel end_date dikurangi variabel
start_date (ditambah 1 hari agar end_date termasuk durasi hari)
            r.duration = (end_date - start_date).days + 1
```

Kita menambahkan field dengan attribute **compute** dan **inverse**, sehingga disana ada 2 method. Method **_get_end_date** (compute) untuk menupdate field **end_date** berdasarkan perhitungan **start_date** ditambah **duration**. Sedangkan method **_set_end_date** (inverse) berfungsi untuk menentukan field lain (duration) dari field end_date.

Kita perhatikan juga, untuk perhitungan tanggal, kita harus mengconvert tanggal kepada object datetime, karna field date atau datetime yang di simpan odoo dengan format string.

Sekarang kita update viewnya untuk melihat hasil dan mencobanya :

```
<!-- ### Membuat Tampilan Tree/List Sesi ### -->

<record model="ir.ui.view" id="sesi_tree_view">
    <field name="name">training.sesi.tree</field>
    <field name="model">training.sesi</field>
    <field name="arch" type="xml">
        <tree string="Sesi List"
              decoration-info="duration&lt;5"
              decoration-danger="duration&gt;15">
            <field name="name"/>
            <field name="course_id"/>
            <field name="start_date"/>
            <field name="end_date"/>
            <field name="duration"/>
            <field name="seats"/>
            <field name="instructor_id"/>
            <field name="taken_seats" widget="progressbar"/>
        </tree>
    </field>
</record>


<!-- ### Membuat Tampilan Form Sesi ### -->

<record model="ir.ui.view" id="sesi_form_view">
    <field name="name">training.sesi.form</field>
    <field name="model">training.sesi</field>
    <field name="arch" type="xml">
        <form string="Sesi Form">
            <sheet>
                <group>
                    <group string="Informasi">
                        <field name="course_id"/>
                        <field name="name"/>
                        <field name="end_date"/>
                        <field name="instructor_id"/>
                    </group>
```

```
                        <group string="Jadwal">
                            <field name="start_date"/>
                            <field name="duration"/>
                            <field name="seats"/>
                            <field name="taken_seats"
                                   widget="progressbar"/>
                        </group>
                        <group string="Peserta" colspan="2">
                         <field name="attendee_ids" nolabel="1"/>
                        </group>
                    </group>
                </sheet>
            </form>
        </field>
    </record>
```

Attribute yang ada pada calendar view ini adalah :

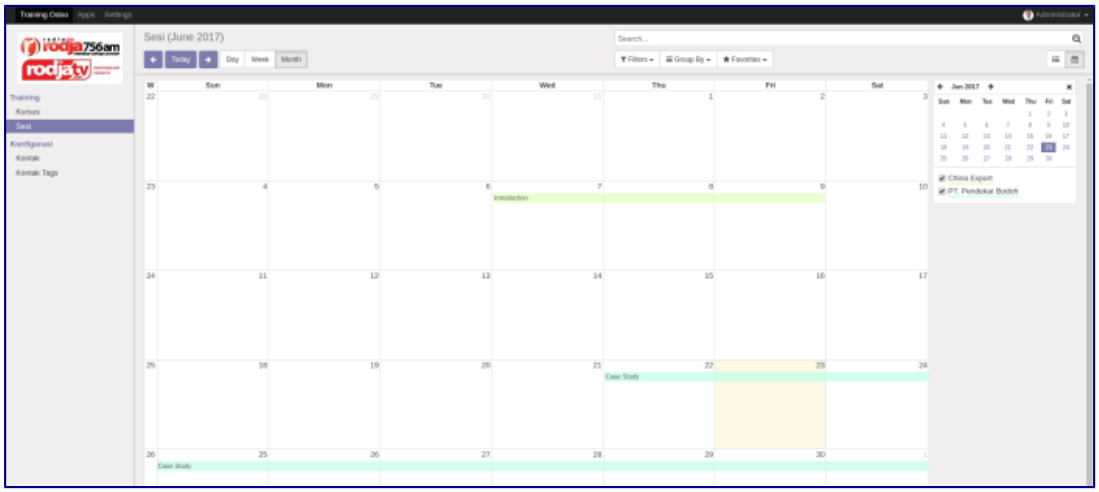
color
date_start
date_stop
date_delay
event_open_popup
quick_add
display
all_day
mode

Kita membuat field **end_date** karna field ini dibutuhkan attribute **date_stop**. Jika kita tentukan attribute **date_stop**, maka calendar view dapat menampilkan durasi harinya. Terakhir, kita buat calendar viewnya seperti berikut :

```
    <!-- ### Membuat Tampilan Calendar Sesi ### -->

    <record model="ir.ui.view" id="sesi_calendar_view">
            <field name="name">training.sesi.calendar</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                    <calendar string="Sesi Calendar"
                              date_start="start_date"
                              date_stop="end_date"
                              color="instructor_id"
                              mode="month">
                            <field name="name"/>
                    </calendar>
            </field>
    </record>
```

```
<!-- ### Membuat Action/Event Object Sesi ### -->

<record model="ir.actions.act_window" id="sesi_list_action">
        <field name="name">Sesi</field>
        <field name="res_model">training.sesi</field>
        <field name="view_type">form</field>
        <field name="view_mode">tree,form,,calendar</field>
</record>
```

Sehingga hasilnya seperti ini :

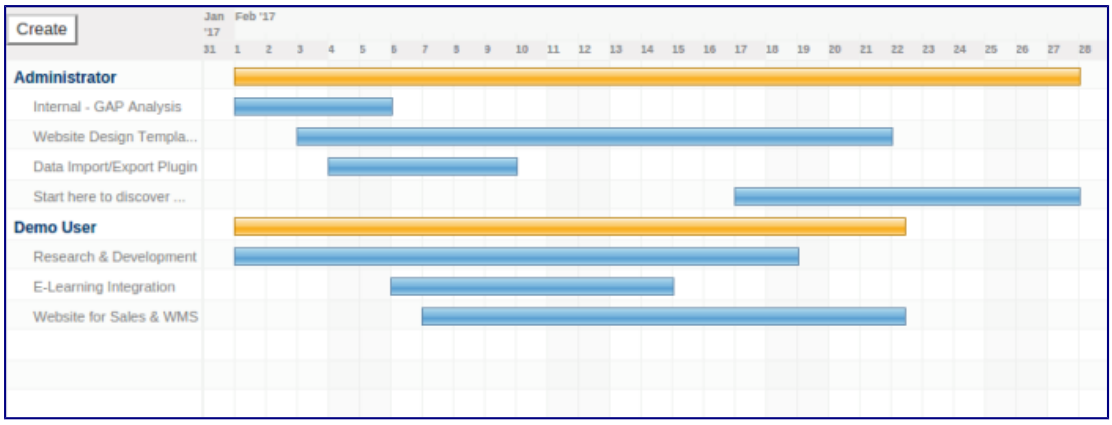


D. Gantt Views

Gantt view berbentuk seperti bar chart yang biasa digunakan untuk membuat timeline seperti progress project, production, leaves, dll. Semenjak v9, fitur ini tidak ada di versi community, salah satu alasannya seperti pembahasan disini :

"The library we used for the gantt view was GPL, we removed it from the LGPL community edition. For enterprise we use a proprietary edition of the library (used in the task planner)." - Antony Lesuisse (CTO in Odoo)

Penampakan Gantt Chart view seperti ini :



Bagi yang memiliki versi enterprise mungkin bisa melanjutkan dengan mengikuti coding di bawah ini :

```
class Sesi(models.Model):
    ...

    hours = fields.Float(string="Durasi Jam", compute='_get_hours',
inverse='_set_hours')

    @api.depends('duration')
    def _get_hours(self):
        for r in self:
            r.hours = r.duration * 24

    def _set_hours(self):
        for r in self:
            r.duration = r.hours / 24
```

Kemudian update file xmlnya (perhatikan id xmlnya - replace jika sama) :

```
        <!-- ### Membuat Tampilan Tree/List Sesi ### -->

        <record model="ir.ui.view" id="sesi_tree_view">
            <field name="name">training.sesi.tree</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                <tree string="Sesi List"
                      decoration-info="duration&lt;5"
                      decoration-danger="duration&gt;15">
                    <field name="name"/>
                    <field name="course_id"/>
                    <field name="start_date"/>
                    <field name="end_date"/>
                    <field name="duration"/>
                    <field name="hours"/>
                    <field name="seats"/>
                    <field name="instructor_id"/>
                    <field name="taken_seats" widget="progressbar"/>
                </tree>
            </field>
        </record>

        <!-- ### Membuat Tampilan Gantt Sesi ### -->

        <record model="ir.ui.view" id="sesi_gantt_view">
                <field name="name">training.sesi.gantt</field>
                <field name="model">training.sesi</field>
                <field name="arch" type="xml">
                    <gantt string="Sesi Gantt" color="course_id"
                           date_start="start_date"
                           date_delay="hours" progress="progress"
                           default_group_by='instructor_id'>
                        <field name="name"/>
                    </gantt>
                </field>
        </record>
```

```
        <!-- ### Membuat Action/Event Object Sesi ### -->

        <record model="ir.actions.act_window" id="sesi_list_action">
                <field name="name">Sesi</field>
                <field name="res_model">training.sesi</field>
                <field name="view_type">form</field>
                <field name="view_mode">tree,form,calendar,
                                        gantt</field>
        </record>
```

E. Graph Views

Pada tampilan ini (seperti BI sederhana) kita bisa mengambil informasi sebagai overview untuk analisa dan pengambilan keputusan strategis selanjutnya. Ada 3 tipe grafik pada view ini yaitu **bar** (default), **pie** dan **line**. Graph view juga sebagai agregasi data dari database, jadi field compute yang nilainya tidak disimpan (on the fly), maka tidak bisa dijadikan parameter pada tampilan ini. Sebelum kita membuatnya, kita akan tambahkan dulu beberapa field yang dibutuhkan pada tampilan ini, yaitu :

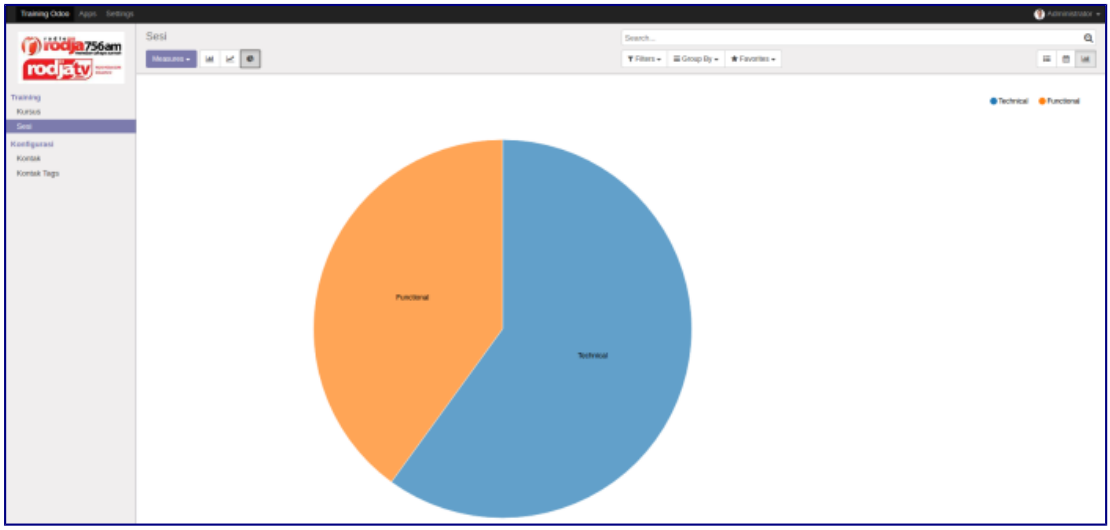
```
class Sesi(models.Model):
    ...


    attendees_count = fields.Integer(string="Jumlah Peserta",
compute='_get_attendees_count', store=True)


    @api.depends('attendee_ids')
    def _get_attendees_count(self):
        for r in self:
            # Mengupdate field attendees_count berdasarkan jumlah
record di tabel peserta
            r.attendees_count = len(r.attendee_ids)
```

Kita buat field compute baru beserta methodnya yang fungsinya menjumlahkan semua Peserta yang ada. Setelah itu kita langsung buat Graph view tanpa menampilkan field tersebut pada form dan list view :

```
        <!-- ### Membuat Tampilan Graph Sesi ### -->

        <record model="ir.ui.view" id="sesi_graph_view">
                <field name="name">training.sesi.graph</field>
                <field name="model">training.sesi</field>
                <field name="arch" type="xml">
                    <graph string="Peserta by Kursus" type="pie">
                       <field name="course_id"/>
                       <field name="attendees_count" type="measure"/>
                    </graph>
                </field>
        </record>
```

```
<!-- ### Membuat Action/Event Object Sesi ### -->

<record model="ir.actions.act_window" id="sesi_list_action">
        <field name="name">Sesi</field>
        <field name="res_model">training.sesi</field>
        <field name="view_type">form</field>
        <field name="view_mode">tree,form,calendar,gantt,
                                graph</field>
</record>
```

Setelah kita update modulnya, maka hasilnya seperti ini :



F. Pivots Views

Tampilan ini pada versi sebelumnya di gabung dengan graph view, karna datanya saling berkaitan. Sesuai namanya, maka tampilan ini memberikan view seperti pivots yang biasa kita lihat pada pada excel. Karna menggunakan data yang sama seperti graph view, maka kita langsung tambahkan saja dengan mengupdate file xml seperti berikut :

```
<!-- ### Membuat Tampilan Pivot Sesi ### -->

<record model="ir.ui.view" id="sesi_pivot_view">
    <field name="name">training.sesi.pivot</field>
    <field name="model">training.sesi</field>
    <field name="arch" type="xml">
        <pivot string="Peserta by Kursus"
               disable_linking="True"
               display_quantity="true">
            <field name="course_id"/>
            <field name="attendees_count" type="measure"/>
        </pivot>
    </field>
</record>
```

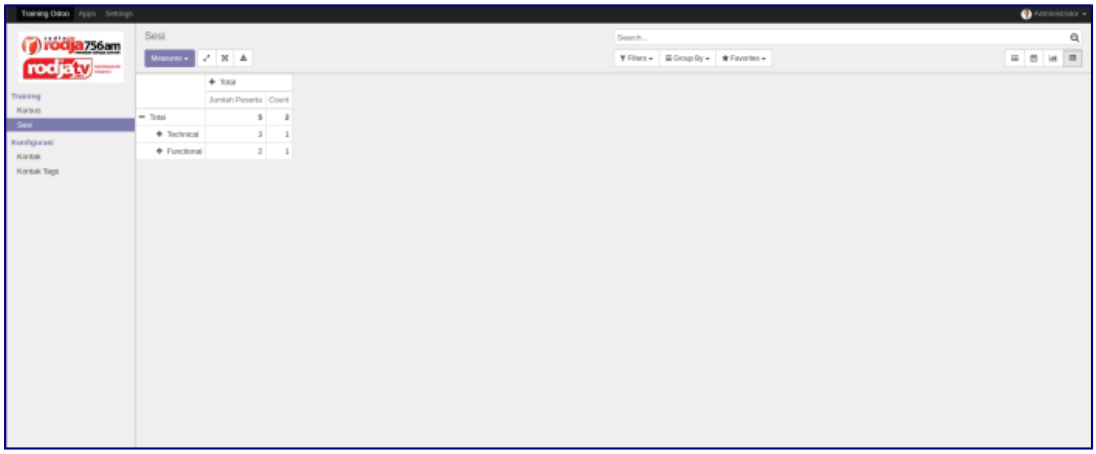
```
<!-- ### Membuat Action/Event Object Sesi ### -->

<record model="ir.actions.act_window" id="sesi_list_action">
    <field name="name">Sesi</field>
    <field name="res_model">training.sesi</field>
    <field name="view_type">form</field>
    <field name="view_mode">tree,form,calendar,gantt,graph,
                            pivot</field>
</record>
```

Pivot view memiliki 2 attribute yaitu :

disable_linking (boolean) = Nilai defaultnya **False**, jika diisi **True** maka akan menghapus link ke data terkait ketika cell di klik

display_quantity (boolean) = Nilai defaultnya **false**, jika diisi **true** maka akan menambah 1 kolom sebagai summary (count) dari data aggregat

Hasil dari tampilan ini adalah :



G. Kanban Views

Kanban view seperti tampilan tumbnails dengan tambahan beberapa informasi. Kanban view menampilkan record seperti 'kartu' yang merupakan 'gabungan' antara form view dan list view. Dengan kanban view kita juga dapat merubah state record hanya dengan drag and drop. Sekarang kita akan tambahkan kanban view pada object Sesi kita seperti berikut :

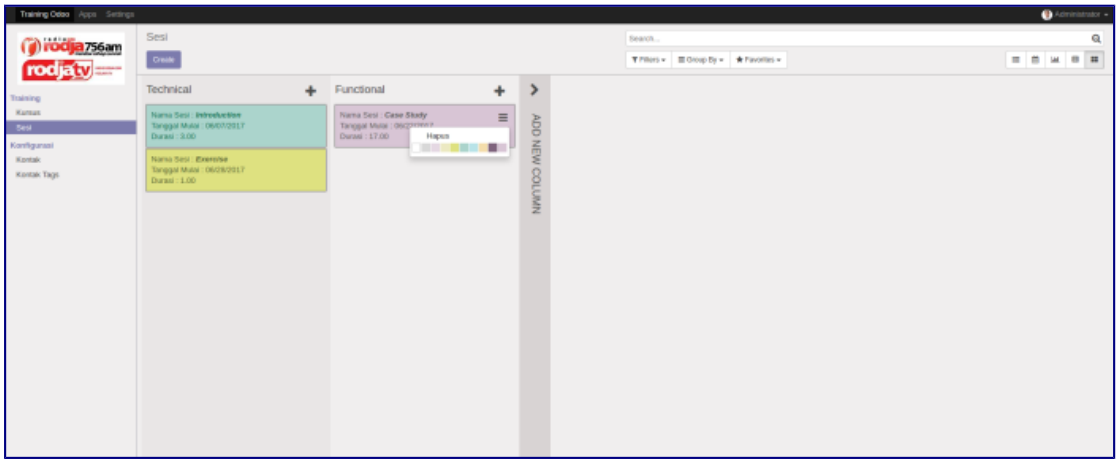
```
class Sesi(models.Model):
    .....

    color = fields.Integer('Warna')
```

Field color berfungsi untuk menyimpan warna pada kanban, field tersebut akan otomatis terisi ketika kita memberikan warna pada kanban kita di mode kanban view. Selanjutnya kita update file xml kita (replace dengan id xml yang sama) :

```
        <!-- ### Membuat Tampilan Tree/List Sesi ### -->

        <record model="ir.ui.view" id="sesi_tree_view">
            <field name="name">training.sesi.tree</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                <tree string="Sesi List" decoration
                      info="duration&lt;5"
                      decoration-danger="duration&gt;15">
                    <field name="name"/>
                    <field name="course_id"/>
                    <field name="start_date"/>
                    <field name="end_date"/>
                    <field name="duration"/>
                    <field name="hours"/>
                    <field name="seats"/>
                    <field name="instructor_id"/>
                    <field name="color"/>
                    <field name="taken_seats" widget="progressbar"/>
                </tree>
            </field>
        </record>
```

```
        <!-- ### Membuat Tampilan Kanban Sesi ### -->

        <record model="ir.ui.view" id="sesi_kanban_view">
            <field name="name">training.sesi.kanban</field>
            <field name="model">training.sesi</field>
            <field name="arch" type="xml">
                <kanban default_group_by="course_id">
                    <field name="color"/>
                    <templates>
                        <t t-name="kanban-box">
                          <div t-attf-
class="oe_kanban_color_{{kanban_getcolor(record.color.raw_value)}}
oe_kanban_global_click_edit oe_semantic_html_override oe_kanban_card
{{record.group_fancy==1 ? 'oe_kanban_card_fancy' : ''}}">
                            <div class="o_dropdown_kanban dropdown">
                                <!-- MENU DROPDOWN -->
                                <a class="dropdown-toggle btn"
                                    data-toggle="dropdown" href="#" >
                                     <span class="fa fa-bars fa-lg"/>
                                 </a>
                               <ul class="dropdown-menu" role="menu"
                                   aria-labelledby="dLabel">
                                  <li><a type="delete">Hapus</a></li>
                                  <li><ul class="oe_kanban_colorpicker"
                                           data-field="color"/></li>
                               </ul>
                              </div>

                              <div t-attf-class="oe_kanban_content" >
                                 <!-- JUDUL -->
                        Nama Sesi:<b><i><field name="name"/></i></b><br/>
                        Tanggal Mulai:<field name="start_date"/><br/>
                        Durasi : <field name="duration"/>
                                 </div>
                            </div>
                        </t>
                    </templates>
                </kanban>
            </field>
        </record>

  <!-- ### Membuat Action/Event Object Sesi ### -->

  <record model="ir.actions.act_window" id="sesi_list_action">
     <field name="name">Sesi</field>
     <field name="res_model">training.sesi</field>
     <field name="view_type">form</field>
     <field name="view_mode">tree,form,calendar,gantt,graph,
                             pivot,kanban</field>
  </record>
```

Hasil kanban view diatas seperti ini :



SECURITY

Pembahasan workflow kita lewatkan karna pada versi 10 ini, fitur workflow telah dihapus. Dan kita cukupkan dengan field **state** untuk melihat progress suatu document. Diantara fungsi/policy security yang bisa kita manfaatkan adalah :

1. Mengatur akses menu, tab, field, button, dll berdasarkan group
2. Mengatur akses object atau model sampai tingkat CRUD (Create, Read, Update, Delete)
3. Mengatur akses record atau data suatu object berdasarkan group atau user (User A hanya bisa melihat data diri sendiri, User B dapat melihat data User C, D, dst)

Fitur security pada Odoo dibagi menjadi 2, yaitu **Access Right** dan **Record Rules**. Kedua fitur ini sangat bermanfaat dan saling melengkapi untuk bisa menghasilkan point-point diatas. Sebuah **groups** memiliki keterkaitan terhadap 2 hal ini. Jika kita tidak mendefinsikan group pada kedua hal ini, maka **Access Right** dan **Record Rules** berlaku kepada semua group.

Untuk yang pertama kita akan membahas **Access Right**.

A. Access Right

Jika kita sering melihat file **ir.model.access.csv** di suatu direktori modul Odoo, maka itu adalah cara untuk mendefinisikan access right. Selain itu, access right juga dapat kita definisikan sebagaimana pembuatan record. Kita bisa langsung menginputnya pada menu **Setting > Security > Access Controls List** atau pada menu **Setting > Groups** di tab **Access Right**.

Access Right sangat erat kaitannya dengan model atau object. Fitur ini fungsi utamanya untuk mengatur akses 'permissions' object atau model seperti **Create, Read, Update, dan Delete** kepada setiap groups. Hal pertama yang kita lakukan adalah membuat **group** dengan langkah-langkah berikut :

1. Buat file baru dengan nama **security.xml** di folder **security**

2. Isi file **security.xml** dengan code di bawah ini :

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
        <record id="group_training_create" model="res.groups">
                <field name="name">Training / Akses Create</field>
        </record>

        <record id="group_training_read" model="res.groups">
                <field name="name">Training / Akses Read</field>
        </record>

        <record id="group_training_update" model="res.groups">
                <field name="name">Training / Akses Update</field>
        </record>

        <record id="group_training_delete" model="res.groups">
                <field name="name">Training / Akses Delete</field>
        </record>

        <record id="group_training_user" model="res.groups">
                <field name="name">Training / User</field>
        </record>

        <record id="group_training_manager" model="res.groups">
            <field name="name">Training / Manager</field>
            <field name="implied_ids"
            eval="[(4, ref('training_odoo.group_training_user'))]"/>
            <field name="users" eval="[(4, ref('base.user_root'))]"/>
        </record>
</odoo>
```

Pada file di atas kita membuat 6 groups yaitu :

Training / Akses Create
Training / Akses Read
Training / Akses Update
Training / Akses Delete
Training / User
Training / Manager

Group **Training / Manager** menginherit group **Training / User**, sehingga apapun yang bisa dilakukan oleh group user, maka bisa dilakukan group manager. Dan group manager otomatis kita tambahakan ke user administrator.

3. Update file **__manifest__.py** pada key dictionary berikut :

```
# always loaded
    'data': [
        'security/security.xml',
        'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
        'views/partner.xml',
    ],
```

4. Update file **ir.model.access.csv** di folder **security** seperti berikut :

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
kursus_user,kursus.user,model_training_kursus,group_training_user,1,1,1,1
sesi_user,sesi.user,model_training_sesi,group_training_user,1,1,1,1
kursus_create,kursus.create,model_training_kursus,group_training_create,1,0,1,0
kursus_read,kursus.read,model_training_kursus,group_training_read,1,0,0,0
kursus_update,kursus.update,model_training_kursus,group_training_update,1,1,0,0
kursus_delete,kursus.delete,model_training_kursus,group_training_delete,1,0,0,1
sesi_create,sesi.create,model_training_sesi,group_training_create,1,0,1,0
sesi_read,sesi.read,model_training_sesi,group_training_read,1,0,0,0
sesi_update,sesi.update,model_training_sesi,group_training_update,1,1,0,0
sesi_delete,sesi.delete,model_training_sesi,group_training_delete,1,0,0,1
```

Pada file diatas, kita menambahkan hak akses seperti berikut :

Training / Akses Create : read dan create
Training / Akses Read : read
Training / Akses Update : read dan update
Training / Akses Delete : read dan delete
Training / User : read, create, update, dan delete

Untuk mencoba dan mensimulasikannya, silahkan membuat 1 user baru, dan masukan ke salah satu group secara bergantian sampai semua group. Dan silahkan perhatikan perbedaaanya.

B. Record Rules

Record rules berfungsi untuk memberikan batasan terhadap hak akses kumpulan record suatu model. Kita bisa membuat record rules langsung dari menu **Setting** > **Security** > **Record Rules** atau seperti biasa kita gunakan xml yang membuatnya. Pada field **domain** inilah pembatasan akses right dilakukan. Sekarang kita akan buat record rule untuk model kursus seperti berikut :

1. Group **Training / User** : hanya bisa melihat kursus yang responsiblenya diri sendiri
2. Group **Training / Manager** : bisa melihat semua kursus tanpa memperdulikan responsiblenya

Sekarang kita update file **security.xml** kita seperti berikut (tambahkan dari bawah) :

```
<record id="kursus_see_own_rule" model="ir.rule">
   <field name="name">Own Kursus</field>
   <field name="model_id" ref="model_training_kursus"/>
   <field name="domain_force">['|',
                  ('responsible_id','=',user.id),
                  ('responsible_id','=',False)]</field>
   <field name="groups" eval="[(4, ref('group_training_user'))]"/>
</record>

<record id="kursus_see_all_rule" model="ir.rule">
   <field name="name">All Kursus</field>
   <field name="model_id" ref="model_training_kursus"/>
   <field name="domain_force">[(1,'=',1)]</field>
   <field name="groups" eval="[(4, ref('group_training_manager'))]"/>
</record>
```

Untuk mensimulasikannya, silahkan masukan user yang telah kita buat ke group **Training / User** dan group **Training / Manager** secara bergantian, dan perhatikan perbedaannya.

Access right juga bisa kita tentukan di record rules ini, tetapi dia tidak akan menjadi prioritas. Prioritas tetap access right di group, kecuali kita tidak membuatnya di group, maka kita bisa buat di record rule.

Untuk memfilter label, div, menu, tab, field, button, dll berdasarkan group, maka kita bisa tambahkan attribute groups pada element terkait, seperti contoh berikut :

```
<div groups="account.group_account_manager"
     class="row o_kanban_card_settings">

<label for="currency_id" groups="base.group_multi_currency"/>

<field name="account_id" groups="account.group_account_user"/>

<button name="action_invoice_open" type="object" states="draft"
        string="Validate" groups="account.group_account_invoice"/>

<menuitem id="menu_bard_journal" name="Dashboard"
          parent="menu_finance" action="open_journal_dashboard"
          groups="group_account_user"/>

<page string="Analytic Lines"
      groups="analytic.group_analytic_accounting">
```

WIZARD

Alhamdulillah kita bisa melanjutkan lagi tutorial ini. Selanjutnya kita akan membahas tentang wizard di Odoo.

Wizard di Odoo digunakan untuk komunikasi interaktif dengan user, biasanya berbentuk popup atau dialog box, yang tampilannya bersifat dinamis. Wizard sama seperti model lain tetapi lebih simple dan memiliki ciri khas seperti berikut :

- Wizard menggunakan (inherite) class ***TransientModel()*** bukan ***Model()***

- Karna menginherite ***TransientModel()***, maka sesuai namanya, recordnya bersifat sementara, pada periode waktu tertentu akan dihapus secara otomatis dari database

- Model atau object dari wizard tidak memiliki akses right, semua user bisa mengaksesnya

- Sebuah record dari wizard bisa memiliki field relasi ke model lain yang biasa, tetapi tidak sebaliknya

Sekarang kita akan membuat sebuah wizard untuk menambah jumlah peserta pada sesi. Ikuti langkah-langkah berikut :

1. Buat folder baru bernama **wizard** di dalam modul training_odoo

2. Update file **__init__.py** untuk menambahkan (import) folder wizard yang baru kita buat

```
from . import wizard
```

3. Buat 2 file python baru di dalam folder wizard

A. file **__init__.py** yang isinya :

```
from . import wizard
```

B. file **wizard.py** untuk membuat object wizard beserta fieldnya seperti berikut :

```
from odoo import models, fields, api

class Wizard(models.TransientModel):
    _name = 'training.wizard'

    def _default_sesi(self):
        return self.env['training.sesi'].browse(
                                self._context.get('active_id'))

    session_id = fields.Many2one('training.sesi', string="Sesi",
                                required=True, default=_default_sesi)
```

```
    attendee_ids = fields.Many2many('res.partner', string="Peserta")


    @api.multi
    def tambah_peserta(self):
        self.session_id.attendee_ids |= self.attendee_ids
        return {}
```

4. Buat file xml baru didalam folder wizard dengan nama **wizard_view.xml** yang isinya :

```
<odoo>
  <data>

        <!-- Membuat form wizard -->

        <record model="ir.ui.view" id="wizard_form_view">
            <field name="name">training.wizard.form</field>
            <field name="model">training.wizard</field>
            <field name="arch" type="xml">
                <form string="Tambah Peserta">
                    <group>
                        <field name="session_id"/>
                        <field name="attendee_ids"/>
                    </group>
                    <footer>
                        <button name="tambah_peserta" type="object"
                                string="Tambah"
                                class="oe_highlight"/>
                        or
                        <button special="cancel" string="Batal"/>
                    </footer>
                </form>
            </field>
        </record>


        <!-- Membuat action baru-->

        <act_window id="launch_session_wizard"
                                name="Tambah Peserta"
                                src_model="training.sesi"
                                res_model="training.wizard"
                                view_mode="form"
                                target="new"
                                key2="client_action_multi"/>

  </data>
</odoo>
```

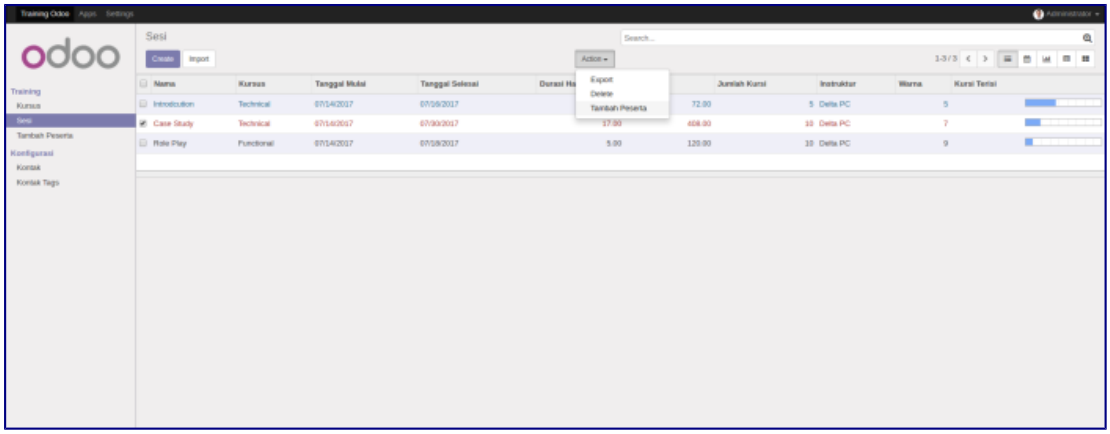
Wizard dijalankan dengan event sama seperti model lain dengan tambahan field **target="new"** (sehingga berbentuk popup). Cara diatas merupakan salah satu cara lain untuk menjalankan wizard, yaitu dari tombol **Action** model terkait. Cirinya dengan adanya field **src_model** dan **key2**.

Pada wizard diatas juga terdapat button khusus yaitu **special="cancel"**. Fungsinya untuk keluar dari wizard tanpa melakukan penyimpanan atau pemrosesan data.

5. Terakhir kita update file __manifest__.py agar view wizard_view.xml bisa di 'render' :

```
# always loaded
    'data': [
        'security/security.xml',
        'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
        'views/partner.xml',
        'wizard/wizard_view.xml',
    ],
```

Setelah kita restart dan upgrade modulnya, silahkan dicoba menambahkan peserta dengan wizard diatas, caranya dengan masuk ke tampilan list/tree sesi, lalu centang salah satu record, nanti akan muncul button dropdown **Action** pilih **"Tambah Peserta"**. Nanti akan muncul wizard yang telah kita buat, dan lakukan penambahan peserta pada sesi terkait.



Cara kedua untuk menjalankan wizard adalah dengan membuat menu item seperti biasa. Kita akan membuat menu item untuk wizard ini, bedanya dengan tambahan bahwa wizard dari menu item ini bisa menambahkan peserta langsung ke banyak sesi. Silahkan update code kita seperti berikut :

```
from odoo import models, fields, api

class Wizard(models.TransientModel):
    _name = 'training.wizard'

    def _default_sesi(self):
        return self.env['training.sesi'].browse(
                                    self._context.get('active_id'))

    session_id = fields.Many2one('training.sesi', string="Sesi",
                                    default=_default_sesi)
    attendee_ids = fields.Many2many('res.partner', string="Peserta")
    session_ids = fields.Many2many('training.sesi', string="Sesi")

    @api.multi
    def tambah_peserta(self):
        self.session_id.attendee_ids |= self.attendee_ids
        return {}


    @api.multi
    def tambah_banyak_peserta(self):
        for sesi in self.session_ids:
            sesi.attendee_ids |= self.attendee_ids
        return {}
```

Diatas kita tambahkan satu field baru yaitu **session_ids** yang tipenya many2many fungsinya agar bisa menambahkan peserta ke banyak sesi (karna sebelumnya hanya 1 sesi -many2one-). Kemudian kita update file xml kita seperti berikut :

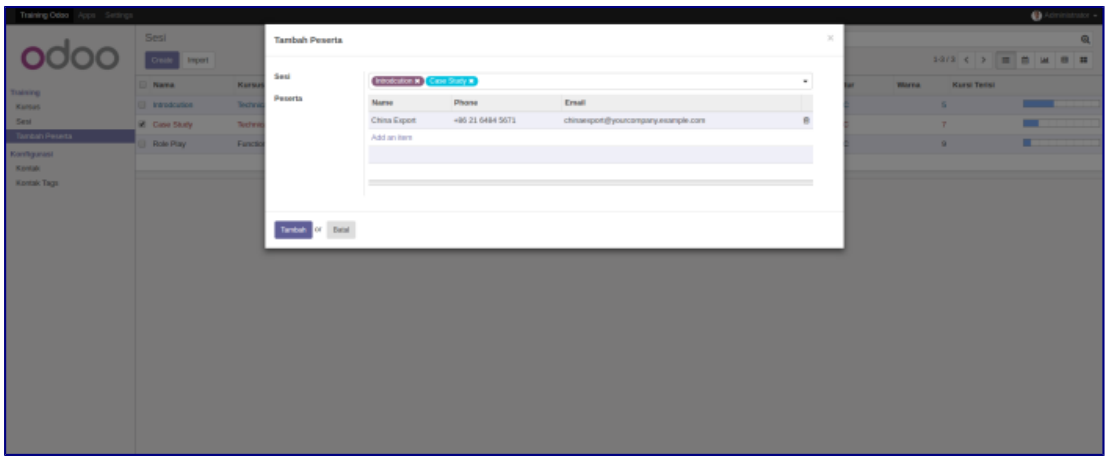
```
        <!-- Membuat form wizard dari menu item -->

        <record model="ir.ui.view" id="wizard_form_view_menu">
            <field name="name">training.wizard.menu.form</field>
            <field name="model">training.wizard</field>
            <field name="arch" type="xml">
                <form string="Tambah Peserta">
                    <group>
                        <field name="session_ids"
                                widget="many2many_tags"/>
                        <field name="attendee_ids"/>
                    </group>
                    <footer>
                        <button name="tambah_banyak_peserta"
                                type="object"
                                string="Tambah"
                                class="oe_highlight"/>
                        or
                        <button special="cancel" string="Batal"/>
                    </footer>
                </form>
            </field>
        </record>
```

```
<!-- Membuat action wizard dari menu item -->

<act_window id="sesi_wizard_menu_action"
                        name="Tambah Peserta"
                        res_model="training.wizard"
                        view_mode="form"
                        view_id="wizard_form_view_menu"
                        target="new"/>


<!-- ### Membuat Sub Menu Wizard ### -->

 <menuitem id="wizard_menu" name="Tambah Peserta"
            parent="training_odoo_menu"
            action="sesi_wizard_menu_action"/>
```

Kita definisikan ulang pembuatan form wizard karna ada field yang berbeda yaitu **session_id** dan **session_ids**. Secara default jika kita telah mendefiniskan view pada suatu model, maka action akan mengambil view tersebut, akan tetapi karna kita memiliki 2 view form dan kita ingin menentukan form mana yang akan kita tampilkan, maka kita harus mendefinisikan id form terkait pada action, yaitu pada baris ini :

```
view_id="wizard_form_view_menu"
```

Kita juga membuat method baru pada button **"Tambah"**, karna ada sedikit perbedaan dari kedua cara penambahan peserta. Silahkan upgrade modulnya dan lakukan simulasi dari menu item baru di bawah Sesi yaitu **Tambah Peserta**.



REPORT

Ada banyak tools yang bisa kita gunakan untuk membuat report pada Odoo 10 seperti Jasper, webkit, dll. Tetapi saya akan cukupkan pembahasan dengan report default yang telah Odoo sediakan yaitu RML dan QWEB.

Sebelum kita masuk ke pembahasan utama yaitu report. Sedikit kita bahas tentang state pada Odoo. Seperti penjelasan pada artikel sebelumnya, **workflow** telah di hapus semenjak v10 dan 'dicukupkan' dengan **state**. Untuk object training ini, kita akan tambahkan 3 state yaitu **draft, open,** dan **done**. Awalan kita update class Kursus() menjadi :

```
class Kursus(models.Model):

    _name = 'training.kursus'

    ...

    state = fields.Selection([
        ('draft', 'Draft'),
        ('open', 'Open'),
        ('done', 'Done'),
        ], string='Status', readonly=True, copy=False,
           default='draft')


    @api.multi
    def action_confirm(self):
        self.write({'state': 'open'})

    @api.multi
    def action_cancel(self):
        self.write({'state': 'draft'})

    @api.multi
    def action_close(self):
        self.write({'state': 'done'})
```

Pada code diatas, kita membuat sebuah field yaitu **state** dan 3 method yaitu **action_confirm(), action_cancel(),** dan **action_close()**. Fungsi dari 3 method tersebut berfungsi hanya untuk merubah sebuah nilai field **state**. Dan ketiga method tersebut akan di panggil oleh 3 button yang berbeda. Sekarang kita sesuaikan view dari penambahan field dan button seperti berikut :

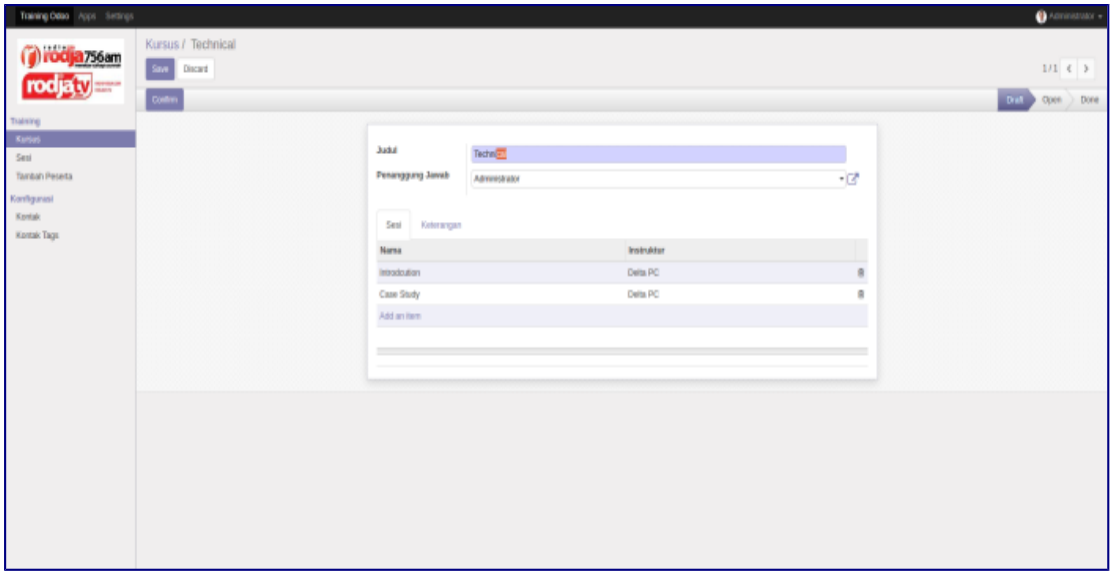
```
<!-- ### Membuat Tampilan Tree/List Kursus ### -->

<record model="ir.ui.view" id="kursus_tree_view">
    <field name="name">training.kursus.tree</field>
    <field name="model">training.kursus</field>
    <field name="arch" type="xml">
        <tree string="Kursus List">
            <field name="name"/>
            <field name="responsible_id"/>
            <field name="description"/>
            <field name="state"/>
        </tree>
    </field>
</record>



<!-- ### Membuat Tampilan Form Kursus ### -->

<record model="ir.ui.view" id="kursus_form_view">
    <field name="name">training.kursus.form</field>
    <field name="model">training.kursus</field>
    <field name="arch" type="xml">
        <form string="Kursus Form">
        <header>
            <button name="action_confirm" type="object"
                string="Confirm" states="draft" class="btn-primary"/>
            <button name="action_cancel" type="object"
                  string="Cancel" states="open"/>
            <button name="action_close" type="object" string="Close"
                  states="open" class="btn-primary"/>
            <field name="state" widget="statusbar"
                  statusbar_visible="draft,open,done"/>
        </header>
        <sheet>
            <group>
                <field name="name"/>
                <field name="responsible_id"/>
            </group>
            <notebook>
                <page string="Sesi">
                    <field name="session_ids">
                        <tree string="Daftar Sesi" editable="bottom">
                            <field name="name"/>
                            <field name="instructor_id"/>
                        </tree>
                        <form>
                            <group string="Informasi">
                                <field name="name"/>
                                <field name="instructor_id"/>
                            </group>
                            <group string="Jadwal">
                                <field name="start_date"/>
                                <field name="duration"/>
                                <field name="seats"/>
                            </group>
                        </form>
                    </field>
                </page>
```

```
                <page string="Keterangan">
                    <field name="description"/>
                </page>
            </notebook>
        </sheet>
        </form>
    </field>
</record>
```

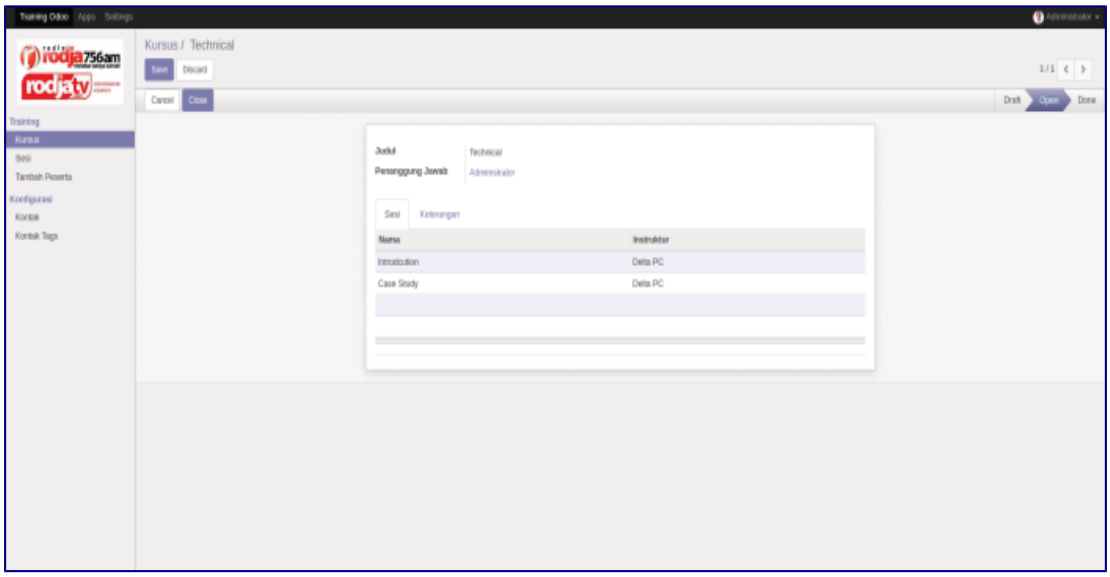
Perubahan coding diatas hanya merubah field state, sedangkan kita juga ingin mempengaruhi field-field yang lain. Yaitu ketika statenya berubah maka field yang lain menjadi **readonly**. Untuk merealisasikan hal tersebut, maka kita harus menambahkan attribute **state** dan **readonly** pada setiap field sehingga hasilnya seperti berikut :

```
class Kursus(models.Model):
    _name = 'training.kursus'

    name = fields.Char(string="Judul", required=True, readonly=True,
                       states={'draft': [('readonly', False)]})
    description = fields.Text(readonly=True,
                       states={'draft': [('readonly', False)]})
    session_ids = fields.One2many('training.sesi', 'course_id',
                       string="Sesi", readonly=True,
                       states={'draft': [('readonly', False)]})
    responsible_id = fields.Many2one('res.users',
                       ondelete='set null',
                       string="Penanggung Jawab", index=True,
                       readonly=True,
                       states={'draft': [('readonly', False)]})
```

Status Draft



Status Open



Untuk selanjutnya kita akan menambahkan log history perubahan yang dilakukan user pada object kursus kita yang biasanya berada di bawah form, seperti form sales order. Sebelumnya pastikan kita telah menginstall modul **Discuss** (mail). Update file __manifest__.py :

```
'depends': ['base', 'mail'],
```

Lanjutkan dengan update di class kursus() kita :

```
class Kursus(models.Model):
    _name = 'training.kursus'
    _inherit = 'mail.thread'


    name = fields.Char(string="Judul", required=True, readonly=True,
                       states={'draft': [('readonly', False)]},
                       track_visibility='always')

    description = fields.Text(readonly=True,
                       states={'draft': [('readonly', False)]},
                       track_visibility='onchange')

    session_ids = fields.One2many('training.sesi', 'course_id',
                       string="Sesi", readonly=True,
                       states={'draft': [('readonly', False)]},
                       track_visibility='onchange')

    responsible_id = fields.Many2one('res.users',
                       ondelete='set null',
                       string="Penanggung Jawab", index=True,
                       readonly=True,
                       states={'draft': [('readonly', False)]},
                       track_visibility='onchange')

    state = fields.Selection([
        ('draft', 'Draft'),
        ('open', 'Open'),
        ('done', 'Done'),
        ], string='Status', readonly=True, copy=False,
            default='draft', track_visibility='onchange')

...
```

Kita membutuhkan class **mail.thread()** untuk mengadopsi log history dan attribute **track_visibility**, kemudian kita update file xml viewnya :

```
<!-- ### Membuat Tampilan Form Kursus ### -->

<record model="ir.ui.view" id="kursus_form_view">
    <field name="name">training.kursus.form</field>
    <field name="model">training.kursus</field>
    <field name="arch" type="xml">
        <form string="Kursus Form">
        <header>
            <button name="action_confirm" type="object"
                string="Confirm" states="draft" class="btn-primary"/>
            <button name="action_cancel" type="object"
                  string="Cancel" states="open"/>
            <button name="action_close" type="object" string="Close"
                  states="open" class="btn-primary"/>
            <field name="state" widget="statusbar"
                  statusbar_visible="draft,open,done"/>
        </header>
        <sheet>
            <group>
                <field name="name"/>
                <field name="responsible_id"/>
            </group>
            <notebook>
                <page string="Sesi">
                    <field name="session_ids">
                        <tree string="Daftar Sesi" editable="bottom">
                            <field name="name"/>
                            <field name="instructor_id"/>
                        </tree>
                        <form>
                            <group string="Informasi">
                                <field name="name"/>
                                <field name="instructor_id"/>
                            </group>
                            <group string="Jadwal">
                                <field name="start_date"/>
                                <field name="duration"/>
                                <field name="seats"/>
                            </group>
                        </form>
                    </field>
                </page>
                <page string="Keterangan">
                    <field name="description"/>
                </page>
            </notebook>
        </sheet>
        <div class="oe_chatter">
            <field name="message_follower_ids"
                   widget="mail_followers"/>
            <field name="message_ids" widget="mail_thread"/>
        </div>
        </form>
    </field>
</record>
```

Silahkan lakukan perubahan (edit) lalu perhatikan field yang berattribute track_visibility='**onchange**' dan track_visibility='**always**'. Sekarang kita akan membuat report RML pada object kursus, lakukan perubahan seperti berikut :

A. Buat folder **report**

B. Lalu buat file **__init__.py** di dalam folder report dengan isi :

```
import report
```

C. Buat file **report.py** dengan isi :

```
import time
from openerp.report import report_sxw

class print_training(report_sxw.rml_parse):
    def __init__(self, cr, uid, name, context):
        super(print_training, self).__init__(cr, uid, name,
                                             context=context)
        self.localcontext.update({
            'time': time,
        })

report_sxw.report_sxw('report.laporan.kursus', 'training.kursus',
                      'addons/training_odoo/report/print_kursus.rml',
                      parser=print_training, header=False)
```

D. Buat file **report_view.xml** dengan isi :

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


        <report id="cetak_kursus" string="Report RML"
                        model="training.kursus" name="laporan.kursus"
                        rml="training_odoo/report/print_kursus.rml"
                        auto="False" menu="False"/>


    </data>
</odoo>
```

Fungsi baris code diatas adalah untuk membuat **action** dari button report yang akan kita buat.

E. Buat file template report **print_kursus.rml** yang isinya :

```
<?xml version="1.0"?>
<document filename="Form Kursus.pdf">

  <!-- DEFINISI LAYOUT (Ukuran Kertas) -->

  <template pageSize="(595.0, 842.0)" title="Report Kursus"
author="Radio Rodja 756 am (http://radiorodja.com)">
    <pageTemplate id="first">

      <!-- Menentukan luas dari canvas yang dapat kita 'tulis' dengan
parameter widht & height beserta titik awal penulisan identasi/margin
-->

      <frame id="first" x1="20.0" y1="485.0" width="538"
height="350"/>

      <header>
            <!-- Membuat text/image sesuai dengan presisi yang tepat
dengan menggunakan titik koordinat -->
            <pageGraphics>
                <setFont name="Helvetica" size="6"/>
                <drawString x="19.0cm" y="1cm">
<pageNumber/></drawString>
            </pageGraphics>
      </header>

    </pageTemplate>
  </template>

  <!-- DEFINISI STYLE -->

  <stylesheet>


    <!-- Mendefiniskan Style Tabel -->

    <blockTableStyle id="Table_String">
      <blockAlignment value="LEFT"/>
      <blockValign value="TOP"/>
    </blockTableStyle>


    <blockTableStyle id="Tabel_Sesi">
      <blockAlignment value="LEFT"/>
      <blockValign value="TOP"/>
        <!-- Kolom ke-1 -->
        <!-- atas --><lineStyle kind="LINEABOVE" colorName="#000000"
start="0,0" stop="0,0"/>
        <!-- kanan --><lineStyle kind="LINEBEFORE"
colorName="#000000" start="1,-1" stop="1,-1"/>
        <!-- bawah --><lineStyle kind="LINEBELOW" colorName="#000000"
start="0,-1" stop="0,-1"/>
        <!-- kiri --><lineStyle kind="LINEBEFORE" colorName="#000000"
start="0,0" stop="0,0"/>
        <!-- Kolom ke-2 -->
```

```
        <!-- atas --><lineStyle kind="LINEABOVE" colorName="#000000"
start="1,0" stop="1,0"/>
        <!-- kanan --><lineStyle kind="LINEBEFORE"
colorName="#000000" start="2,-1" stop="2,-1"/>
        <!-- bawah --><lineStyle kind="LINEBELOW" colorName="#000000"
start="1,-1" stop="1,-1"/>
        <!-- kiri --><lineStyle kind="LINEBEFORE" colorName="#000000"
start="1,0" stop="1,0"/>
    </blockTableStyle>

    <initialize>
      <paraStyle name="all" alignment="justify"/>
    </initialize>

    <!-- Mendefinisikan Style Huruf -->

    <paraStyle name="terp_header" fontName="Helvetica-Bold"
fontSize="12.0" leading="15" alignment="CENTER" spaceBefore="0.0"
spaceAfter="0.0"/>
    <paraStyle name="terp_table_header" fontName="Helvetica-Bold"
fontSize="7.0" leading="8" alignment="CENTER" spaceBefore="0.0"
spaceAfter="0.0"/>
    <paraStyle name="terp_default" fontName="Helvetica"
fontSize="7.0" leading="11" alignment="LEFT" spaceBefore="0.0"
spaceAfter="0.0"/>

    <images/>
  </stylesheet>

  <!-- DEFINISI DATA -->
  <story>
  <pto>

    <!-- DEFINISI OBJECT ALIAS -->

    <para style="terp_default">[[repeatIn(objects, 'kursus')]]
</para>

    <!-- Membuat Judul Report -->

    <blockTable colWidths="500.0" style="Table_String">
      <tr><td><para style="terp_header">REPORT
KURSUS</para></td></tr>
    </blockTable>

    <!-- Membuat Header Report -->
    <blockTable colWidths="330.0,170.0" style="Table_String">
      <tr>
        <td>
          <para style="terp_default">Judul Kursus :
[[ kursus.name ]]</para>
        </td>
        <td>
          <para style="terp_default">Penanggung Jawab :
[[ kursus.responsible_id.name ]]</para>
        </td>
      </tr>
    </blockTable>
```

```
    <!-- Membuat Gap/Space Baris Kosong -->
    <para style="terp_default"><font color="white">.</font></para>

    <!-- Membuat Header Tabel Sesi -->
    <blockTable colWidths="350.0,150.0" repeatRows="1"
style="Tabel_Sesi">
      <tr>
        <td><para style="terp_table_header">Nama</para></td>
        <td><para style="terp_table_header">Instruktur</para></td>
      </tr>
    </blockTable>

    <!-- Membuat Looping Data Tabel Sesi (Element Section) -->
    <section>

      <para style="terp_default">[[ repeatIn(kursus.session_ids, 'x')
]]</para>

      <blockTable colWidths="350.0,150.0" repeatRows="0"
style="Tabel_Sesi">
        <tr>
          <td><para style="terp_default">[[ x.name ]]</para></td>
          <td><para
style="terp_default">[[ x.instructor_id.name ]]</para></td>
        </tr>
      </blockTable>
    </section>
  </pto>
  </story>
</document>
```

F. Update file **__init__.py** di folder **training_odoo** (yang sejajar dengan __manifest__.py):

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

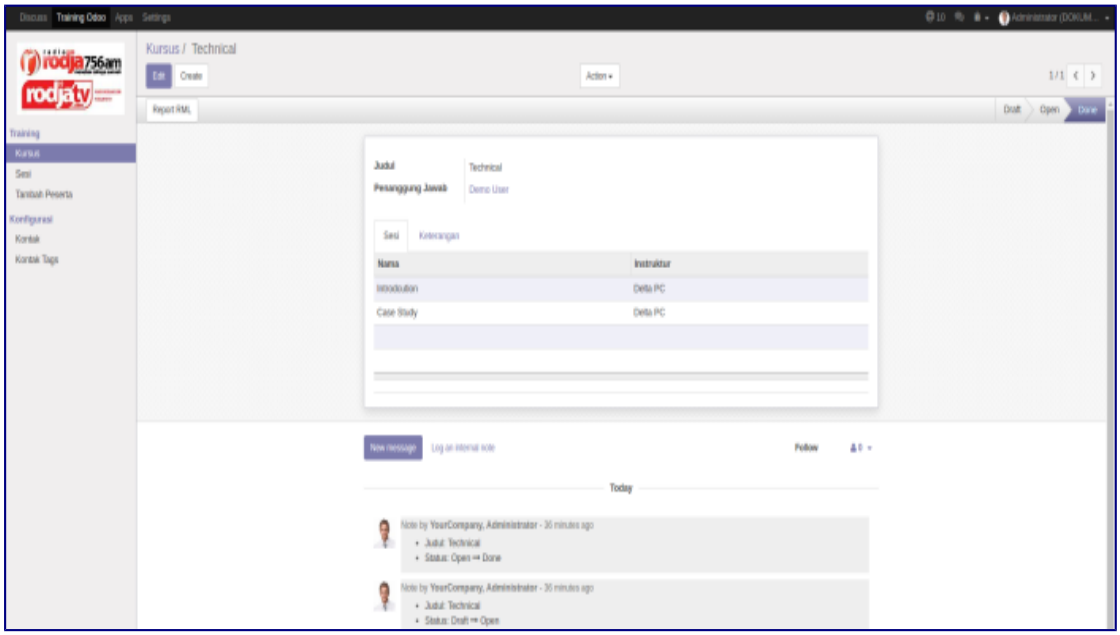
G. Update file **__manifest__.py** :

```
# always loaded
    'data': [
        'report/report_view.xml',
        'security/security.xml',
        'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
        'views/partner.xml',
        'wizard/wizard_view.xml',
    ],
```

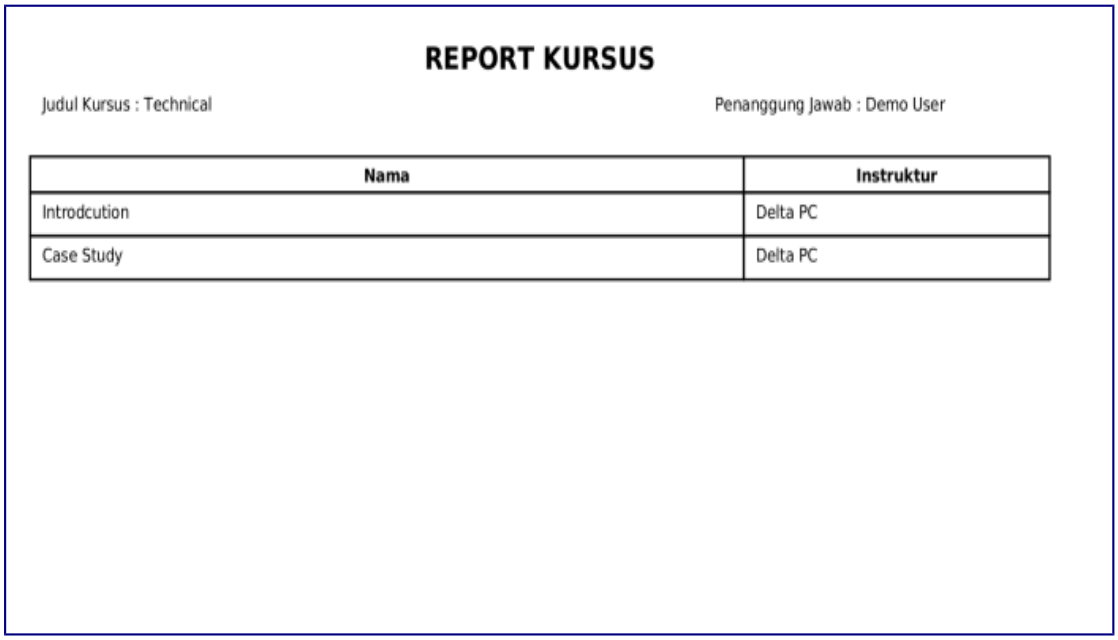
Terakhir kita buat button untuk mengenerate report di form kursus :

H. Update form kursus sebagai berikut :

```
<!-- ### Membuat Tampilan Form Kursus ### -->

    <record model="ir.ui.view" id="kursus_form_view">
        <field name="name">training.kursus.form</field>
        <field name="model">training.kursus</field>
        <field name="arch" type="xml">
            <form string="Kursus Form">
            <header>
                <button name="action_confirm" type="object"
                string="Confirm" states="draft" class="btn-primary"/>
                <button name="action_cancel" type="object"
                  string="Cancel" states="open"/>
                <button name="action_close" type="object"
                  string="Close" states="open" class="btn-primary"/>
                <button name="%(cetak_kursus)d" type="action"
                  states="done" string="Report RML"/>
                <field name="state" widget="statusbar"
                  statusbar_visible="draft,open,done"/>
            </header>
                <sheet>
                    <group>
                        <field name="name"/>
                        <field name="responsible_id"/>
                    </group>
                    <notebook>
                      <page string="Sesi">
                        <field name="session_ids">
                        <tree string="Daftar Sesi" editable="bottom">
                              <field name="name"/>
                               <field name="instructor_id"/>
                        </tree>
                        <form>
                              <group string="Informasi">
                                  <field name="name"/>
                                  <field name="instructor_id"/>
                               </group>
                               <group string="Jadwal">
                                    <field name="start_date"/>
                                    <field name="duration"/>
                                    <field name="seats"/>
                               </group>
                        </form>
                        </field>
                      </page>
                      <page string="Keterangan">
                          <field name="description"/>
                      </page>
                    </notebook>
                </sheet>
            <div class="oe_chatter">
                <field name="message_follower_ids"
                        widget="mail_followers"/>
                <field name="message_ids" widget="mail_thread"/>
            </div>
            </form>
        </field>
    </record>
```



Hasilnya seperti ini :



REPORT KURSUS

Judul Kursus : Technical

Penanggung Jawab : Demo User

Nama	Instruktur
Introdcution	Delta PC
Case Study	Delta PC

Selanjutnya kita akan membuat report QWEB pada object sesi, ikuti langkah-langkah berikut :

A. Buat button report di form sesi

```
<!-- ### Membuat Tampilan Form Sesi ### -->

  <record model="ir.ui.view" id="sesi_form_view">
      <field name="name">training.sesi.form</field>
      <field name="model">training.sesi</field>
      <field name="arch" type="xml">
      <form string="Sesi Form">
      <header>
          <button name="cetak_sesi" string="Report QWEB"
            type="object" class="btn-primary"/>
      </header>
      <sheet>
          <group>
              <group string="Informasi">
                  <field name="course_id"/>
                  <field name="name"/>
                  <field name="instructor_id"/>
                  <field name="seats"/>
              </group>
              <group string="Jadwal">
                  <field name="start_date"/>
                  <field name="duration"/>
                  <field name="end_date"/>
                  <field name="taken_seats"
                         widget="progressbar"/>
              </group>
              <group string="Peserta" colspan="2">
                  <field name="attendee_ids" nolabel="1"/>
              </group>
          </group>
      </sheet>
          </form>
      </field>
  </record>
```

B. Buat method dari button report diatas (karna typenya object) pada class sesi :

```
class Sesi(models.Model):
    _name = 'training.sesi'

    ....

    @api.multi
    def cetak_sesi(self):
        return self.env['report'].get_action(self,
'training_odoo.laporan_sesi')
```

C. Buat action yang akan di panggil dari method button diatas pada file **report_view.xml** :

```
<report
        id="cetak_sesi"
        string="Report QWEB"
        model="training.sesi"
        report_type="qweb-pdf"
        file="training_odoo.laporan_sesi"
        name="training_odoo.laporan_sesi"
    />
```

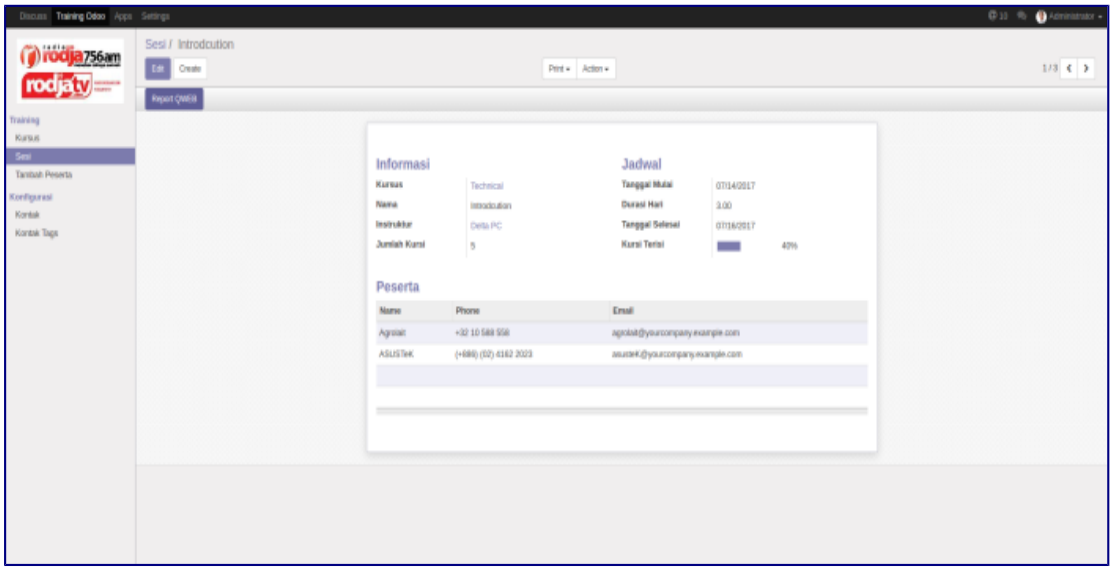
D. Buat file template report didalam folder **report** dengan nama **print_sesi.xml** yang isinya :

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


<template id="laporan_sesi">
    <t t-call="report.html_container">
        <t t-foreach="docs" t-as="o">
            <t t-call="report.external_layout">
                <div class="page">

                    <!-- JUDUL REPORT-->

                    <h2 class="text-center">REPORT SESI</h2>

                    <!-- INFORMASI HEADER -->

                    <table class="table">
                        <tr>
                              <td><strong>Nama</strong></td>
                               <td><span t-field="o.name"/></td>
                               <td><strong>Instruktur</strong></td>
                      <td><span t-field="o.instructor_id.name"/></td>
                        </tr>
                        <tr>
                            <td><strong>Tanggal Mulai</strong></td>
                            <td><span t-field="o.start_date"/></td>
                            <td><strong>Tanggal Akhir</strong></td>
                            <td><span t-field="o.end_date"/></td>
                        </tr>
                    </table>
                    <table class="table table-bordered"
                           style="table-layout:auto">
                        <thead>
                            <tr>
                                <th class="text-center">Name</th>
                                <th class="text-center">Phone</th>
                                <th class="text-center">Email</th>
                            </tr>
                       </thead>
                       <tbody>
```

```
                                <t t-foreach="o.attendee_ids" t-as="l">
                                    <tr>
                                        <td><span t-field="l.name"/></td>
                                        <td><span t-field="l.phone"/></td>
                                        <td><span t-field="l.email"/></td>
                                    </tr>
                                </t>
                            </tbody>
                        </table>
                    </div>
                </t>
            </t>
        </t>
</template>

</odoo>
```

E. Update file _manifest__.py :

```
# always loaded
    'data': [
        'report/report_view.xml',
        'report/print_sesi.xml',
        'security/security.xml',
        'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
        'views/partner.xml',
        'wizard/wizard_view.xml',
    ],
```



Report yang dihasilkan seperti berikut :

REPORT SESI

Nama	Introdcution	**Instruktur**	Delta PC
Tanggal Mulai	07/14/2017	**Tanggal Akhir**	07/16/2017

Name	Phone	Email
Agrolait	+32 10 588 558	agrolait@yourcompany.example.com
ASUSTeK	(+886) (02) 4162 2023	asusteK@yourcompany.example.com

Alhamdulillah wa syukurillah, akhirnya kita telah menyelesaikan ebook tutorial technical ini, bagi yang mengalami kendala, bisa mendownload hasil tulisan ini disini.

Yang benar datangnya dari Allah azza wa jalla, sedangkan kesalahan dari saya pribadi. Saran dan kritik yang membangun sangat saya harapkan. Semoga bermanfaat.

Wassalammu’alaikum