Lecture 03


Gestión de Compras
Gestión de Ventas
Gestión de Almacén
odoo
Gestión Financiera
Y mucho más...
Gestión de Personal

Odoo

You can access the course materials via this link

https://maharatech.gov.eg/course/view.php?id=244

Contents

- API constraints
- SQL constraints
- Computed fields
- Model Inheritance
- Views Inheritance

API Constraints

- A Python constraint is defined as a method decorated with ***@api.constrains***
- Invoked on a recordset.
- The decorator specifies which fields are involved in the constraint, so that the constraint is **automatically evaluated** when one of them is modified.
- The method is expected to **raise an exception** if its invariant is not satisfied

```
@api.constrains("age")
def _check_age(self):
    for record in self:
        if record.age > 30:
            raise ValidationError(f"Your record is too old: {record.age}")
    # all records passed the test, don't return anything
```

SQL Constraints

- A SQL constraint is defined as an attribute list called **_sql_constraints**
- The constraint is being invoked with each create/update from database side
- Allowed SQL Constraints **UNIQUE, CHECK**
- Each constraint is a tuple consisting of (“Title”, “Constraint”, “Error msg”)

```
_sql_constraints = [
    ("Duplicate Name", "UNIQUE(name)", "The name you entered already exist"),
    ("Invalid Age", "CHECK(age >= 10, age <= 40)", "The age you entered is invalid"),
]
```

Computed Fields

- Fields can be computed instead of read straight from the database using the compute parameter.
- It **must** assign the computed value to the field.
- If it uses the values of other fields, it should specify those fields using ***depends()*** decorator

```
from odoo import api
total = fields.Float(compute='_compute_total')

@api.depends('value', 'tax')
def _compute_total(self):
    for record in self:
        record.total = record.value + record.value * record.tax
```

Computed Fields

- computed fields are not stored by default, they are computed and returned when requested.
- Setting store=True will store them in the database and automatically enable searching
- multiple fields can be computed at the same time by the same method, just use the same method on all fields and set all of them

Model Inheritance

1. Add the module that you need to extend/modify to the **depends** key in the manifest file
2. Add new model to your module classes
3. Use _inherit to inherit already existing model
4. You can add new fields/methods or override old ones
5. In case you used a new value for _name attribute it will create new model

```
class HrEmployeeInherit(models.Model):
    _name = 'hr.employee'
    _inherit = 'hr.employee'

    my_new_field = fields.Char()
```

Views Inheritance

- Use **inherit_id** to inherit already existing view
- Use **xpath** tag to define the position of the field/group/page… you need to modify
- Available positions are [before, after, replace, inside, attributes]

```
<!-- Partner form -->
<record id="view_res_partner_filter_assign_tree" model="ir.ui.view">
    <field name="name">res.partner.geo.inherit.tree</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_tree"/>
    <field name="arch" type="xml">
        <field name="user_id" position="after">
            <field name="date_review_next"/>
        </field>
    </field>
</record>
```

Views Inheritance

- Using inside position and xpath

```
<xpath expr="//group[@name='sale']" position="inside">
    <field name="assigned_partner_id" groups="base.group_no_one"/>
</xpath>
```

- Using attributes position and xpath

```
<xpath expr="//group[@name='invoicing']" position="attributes">
    <attribute name="invisible">0</attribute>
</xpath>
```