Odoo ERP

Mục lục

**[I. Module trong Odoo 2](#_Toc25287)**

**[1. Định nghĩa 2](#_Toc8462)**

**[II. Cấu trúc Model 2](#_Toc17794)**

**[1. Định nghĩa 2](#_Toc19775)**

**[2. Cấu trúc của model 2](#_Toc24046)**

**[III. Khởi tạo Data 4](#_Toc24218)**

**[1. Master Data 4](#_Toc1307)**

**[2. Demo Data 4](#_Toc5620)**

**[3. So sánh giữa Master Data và Demo Data 4](#_Toc7302)**

**[IV. Decoration 5](#_Toc19180)**

**[V. Security 7](#_Toc28956)**

**[1. Access Control Lists (ACLs) 7](#_Toc17238)**

**[2. Record Rules 7](#_Toc13175)**

**[VI. Basic View 8](#_Toc20250)**

**[VII. Contrains 9](#_Toc5834)**

**[1. SQL Constraints 10](#_Toc5091)**

**[2. Python Constraints 10](#_Toc29778)**

**[VIII. Inherit View 10](#_Toc3910)**

**[IX. SQL View 12](#_Toc32595)**

**[X. Widget 13](#_Toc19293)**

**[XI. Controller 14](#_Toc13793)**

**[XII. Snippets 16](#_Toc24796)**

**[XIII. Polish-notations 17](#_Toc23814)**

**[XIII. Một vài widget 17](#_Toc31531)**

# Module trong Odoo

## 1. Định nghĩa

· **Module** trong Odoo là một **gói chức năng** bổ sung được thiết kế để mở rộng hoặc thay đổi các tính năng mặc định của Odoo.

· Module có thể bao gồm các **model**, **view**, **controller**, và **logic** để thêm mới hoặc tùy chỉnh chức năng.

1. Cấu trúc

Thư mục module sẽ được tạo với các file chính:

* \_\_init\_\_.py: Khai báo các file Python.
* \_\_manifest\_\_.py: Thông tin metadata của module (name, version, dependencies,...).
* models/: Chứa logic Python cho các model.
* views/: Chứa file XML cho giao diện (form, tree, kanban,...).
* security/: Quyền truy cập (access rights và record rules).

# II. Cấu trúc Model

## 1. Định nghĩa

Model trong Odoo là thành phần đại diện cho các đối tượng của module và được ánh xạ với các bảng trong cơ sở dữ liệu thông qua ORM (Object-Relational Mapping). Mỗi model tương ứng với một bảng trong cơ sở dữ liệu, giúp việc thao tác với dữ liệu trở nên trực quan và dễ dàng hơn.

## 2. Cấu trúc của model

**2.1 Phần định nghĩa**

· **Kế thừa từ** models.Model: Mỗi model trong Odoo cần kế thừa lớp models.Model của Odoo, đây là lớp cơ bản cung cấp các phương thức làm việc với cơ sở dữ liệu.

· **Thuộc tính** \_name: Đây là tên của model, đồng thời cũng là tên của bảng trong cơ sở dữ liệu. Tên model nên được đặt dưới dạng một chuỗi với định dạng phân cách bằng dấu chấm (ví dụ: 'education.student').

· **Thuộc tính** \_description: Dùng để mô tả về model. Thuộc tính này là một chuỗi văn bản mô tả mục đích của model.

· **Thuộc tính** \_order: Dùng để xác định thứ tự mặc định khi truy vấn các bản ghi của model. Nếu không chỉ định, mặc định sẽ không có thứ tự.

· **Thuộc tính** \_inherit: Dùng để kế thừa các model khác. Nếu model của bạn cần mở rộng hoặc tùy chỉnh một model đã có sẵn trong Odoo, bạn có thể sử dụng \_inherit để kế thừa từ model đó và thêm hoặc thay đổi các trường, phương thức, và hành vi.

**2.2 Phần trường của model (Field)**

Các loại trường trong Odoo có thể chia thành hai nhóm chính: **Trường đơn giản** và **Trường quan hệ**.

#### ****Trường đơn giản****:

* **Char**: Chuỗi ký tự (tương tự kiểu VARCHAR trong cơ sở dữ liệu).
* **Text**: Chuỗi văn bản dài, không có giới hạn độ dài.
* **Html**: Chuỗi văn bản định dạng HTML.
* **Integer**: Số nguyên.
* **Float**: Số thập phân.
* **Boolean**: Kiểu dữ liệu đúng/sai (True/False).
* **Date**: Ngày (định dạng: yyyy-mm-dd).
* **Datetime**: Ngày và giờ (định dạng: yyyy-mm-dd hh:mm:ss).
* **Selection**: Danh sách các giá trị chọn từ trước, dùng cho các lựa chọn cố định.

#### ****Trường quan hệ****:

* **Many2one**: Liên kết đến một bản ghi trong model khác (quan hệ N:1). Ví dụ: Nhiều đơn hàng có thể liên kết đến một khách hàng.
* **One2many**: Liên kết đến nhiều bản ghi trong model khác (quan hệ 1:N). Ví dụ: Một khách hàng có thể có nhiều đơn hàng.
* **Many2many**: Liên kết đến nhiều bản ghi trong model khác (quan hệ N:N). Ví dụ: Một sản phẩm có thể xuất hiện trong nhiều đơn hàng.

**2.3 Thuộc tính của trường (Field Attributes)**

Các thuộc tính giúp điều chỉnh hành vi và giao diện của trường trong Odoo:

* **string**: Tên hiển thị của trường trên giao diện người dùng.
* **required**: Đặt trường là bắt buộc (phải nhập giá trị khi tạo hoặc chỉnh sửa bản ghi).
* **default**: Giá trị mặc định của trường khi không có dữ liệu đầu vào.
* **help**: Hướng dẫn hoặc ghi chú cho người dùng về cách sử dụng trường.
* **readonly**: Chỉ cho phép đọc, không thể chỉnh sửa giá trị.
* **index**: Tạo chỉ mục cho trường trong cơ sở dữ liệu để tối ưu hóa truy vấn.
* **compute**: Định nghĩa một hàm tính toán giá trị tự động cho trường khi có sự thay đổi liên quan.
* **store**: Lưu giá trị tính toán vào cơ sở dữ liệu (nếu trường tính toán được lưu trữ).
* **inverse**: Hàm cập nhật ngược lại khi giá trị tính toán thay đổi.
* **search**: Hàm tùy chỉnh cách thức tìm kiếm giá trị của trường trong giao diện người dùng.
* **related**: Liên kết giá trị của trường với một trường khác từ model khác.
* **ondelete**: Kiểm soát hành vi khi xóa bản ghi liên quan (các tùy chọn như cascade, restrict, set null).
* **domain**: Định nghĩa điều kiện giới hạn giá trị có thể chọn cho các trường Many2one và Many2many.
* **translate**: Cho phép dịch giá trị của trường sang nhiều ngôn ngữ.
* **tracking**: Theo dõi các thay đổi của trường để phục vụ mục đích báo cáo hoặc audit.
* **copy**: Cho phép sao chép giá trị trường khi tạo bản ghi mới.
* **groups**: Giới hạn quyền truy cập trường dựa trên nhóm người dùng.
* **widget**: Xác định kiểu giao diện người dùng (UI widget) để hiển thị trường.

**2.4 Các Hàm Đặc Biệt (Decorators)**

Các hàm đặc biệt trong Odoo giúp điều chỉnh hành vi của các trường hoặc phương thức:

* **@api.depends**: Được sử dụng để tính toán lại giá trị của các trường tính toán (computed fields) khi một hoặc nhiều trường phụ thuộc vào sự thay đổi.
* **@api.depends\_context**: Tương tự như @api.depends, nhưng có thể tính toán dựa trên các giá trị trong context.
* **@api.onchange**: Được sử dụng để kích hoạt hành động khi một trường nào đó thay đổi giá trị trong giao diện người dùng.
* **@api.constrains**: Dùng để kiểm tra các ràng buộc dữ liệu khi lưu bản ghi (validation) nhằm đảm bảo dữ liệu hợp lệ.
* **@api.autovacuum**: Dùng để dọn dẹp dữ liệu không cần thiết hoặc giải phóng tài nguyên.

# III. Khởi tạo Data

## **1. Master Data**

**Master Data** (Dữ liệu chính) là các thông tin cơ bản, không thay đổi thường xuyên và đóng vai trò quan trọng trong hệ thống. Những bản ghi này thường đại diện cho các đối tượng cốt lõi của doanh nghiệp, như khách hàng, sản phẩm, nhà cung cấp, nhân viên, v.v. Các dữ liệu này cung cấp thông tin nền tảng cho các quy trình và hoạt động trong các module của hệ thống.

## **2. Demo Data**

**Demo Data** (Dữ liệu mẫu) là dữ liệu được tạo ra để minh họa cách thức hoạt động của hệ thống. Dữ liệu này được sử dụng chủ yếu trong các buổi thử nghiệm, đào tạo hoặc trong các bản demo để khách hàng dễ dàng hiểu và trải nghiệm các tính năng của hệ thống. Demo data không mang tính chính thức và có thể thay đổi hoặc xóa bỏ sau khi mục đích thử nghiệm được hoàn thành.

## **3. So sánh giữa Master Data và Demo Data**

· **Mục đích sử dụng**

· **Master Data**: Là dữ liệu nền tảng cung cấp thông tin cơ bản cho các hoạt động trong hệ thống, hỗ trợ các quy trình nghiệp vụ chính thức.

* **Demo Data**: Dữ liệu mẫu được sử dụng để trình diễn, thử nghiệm hoặc đào tạo, không phục vụ cho các quy trình nghiệp vụ thực tế.

· **Ảnh hưởng đến hệ thống**

* **Master Data**: Có ảnh hưởng trực tiếp đến các business logic trong hệ thống và các quy trình nghiệp vụ. Việc thay đổi Master Data có thể thay đổi cách thức hoạt động của hệ thống.
* **Demo Data**: Không ảnh hưởng đến các quy trình nghiệp vụ trong hệ thống. Dữ liệu này chỉ dùng cho mục đích thử nghiệm và trình diễn, không tác động đến hoạt động chính của hệ thống.

· **Nơi khai báo**

* **Master Data**: Được khai báo trong thẻ data của module.
* **Demo Data**: Được khai báo trong thẻ demo của module.

· **Lúc khởi tạo**

* **Master Data**: Được cài đặt hoặc nâng cấp khi thực hiện cài đặt hoặc nâng cấp module.
* **Demo Data**: Được cài đặt khi lần đầu tiên cài đặt module và chỉ phục vụ cho mục đích thử nghiệm, đào tạo.

# IV. Decoration

Decoration được sử dụng để định dạng hoặc thay đổi cách hiển thị các trường và bản ghi trên giao diện người dùng (views), giúp cải thiện trải nghiệm người dùng bằng cách làm nổi bật các trạng thái hoặc thông tin quan trọng. Dưới đây là các loại decoration phổ biến:

decoration-danger:

**Mô tả**: Hiển thị bản ghi với màu đỏ.

**Mục đích**: Thường dùng để chỉ lỗi hoặc cảnh báo, giúp người dùng dễ dàng nhận biết các vấn đề nghiêm trọng cần chú ý.

decoration-warning:

**Mô tả**: Hiển thị bản ghi với màu cam.

**Mục đích**: Dùng để nhắc nhở người dùng về một tình huống tiềm ẩn rủi ro hoặc cần sự chú ý nhưng không phải là vấn đề nghiêm trọng.

decoration-success:

**Mô tả**: Hiển thị bản ghi với màu xanh lá cây.

**Mục đích**: Dùng để chỉ trạng thái hoàn thành hoặc thành công, biểu thị rằng mọi thứ đã diễn ra đúng cách.

decoration-muted:

**Mô tả**: Hiển thị bản ghi với màu xám.

**Mục đích**: Thường dùng để chỉ trạng thái không hoạt động hoặc đã bị ẩn, giúp giảm tầm quan trọng của bản ghi.

decoration-primary:

**Mô tả**: Hiển thị bản ghi với màu xanh dương.

**Mục đích**: Dùng để làm nổi bật các bản ghi quan trọng hoặc có trạng thái cần sự chú ý đặc biệt từ người dùng.

decoration-info:

**Mô tả**: Hiển thị bản ghi với màu xanh nhạt.

**Mục đích**: Dùng để cung cấp thông tin, giúp người dùng nhận diện các bản ghi mang tính thông báo hoặc dữ liệu cần tham khảo.

decoration-bf:

**Mô tả**: Hiển thị bản ghi với chữ đậm (Bold).

**Mục đích**: Dùng để làm nổi bật các trường thông tin quan trọng hoặc cần sự chú ý đặc biệt.

decoration-it:

**Mô tả**: Hiển thị bản ghi với chữ nghiêng (Italic).

**Mục đích**: Thường dùng để hiển thị thông tin bổ sung hoặc phụ thuộc, có thể biểu thị các yếu tố không chính thức.

decoration-strike:

**Mô tả**: Hiển thị bản ghi với chữ gạch ngang.

**Mục đích**: Dùng để chỉ các trạng thái bị hủy bỏ hoặc không còn hiệu lực, giúp người dùng nhận diện các mục không còn giá trị.

# V. Security

Odoo cung cấp nhiều cơ chế bảo mật khác nhau để kiểm soát quyền truy cập và đảm bảo tính an toàn của dữ liệu trong hệ thống. Các cơ chế bảo mật bao gồm **quyền truy cập cấp độ mô-đun**, **record rules** (quy tắc bản ghi) và **Access Control Lists (ACLs)**.

## **Access Control Lists (ACLs)**

**1.1 Định nghĩa**

**Access Control Lists (ACLs)** trong Odoo là một cơ chế bảo mật giúp xác định quyền truy cập của người dùng đối với các mô-đun hoặc các đối tượng trong hệ thống. ACLs cho phép định nghĩa các quyền truy cập mà một nhóm người dùng có đối với các **models** (mô hình dữ liệu) cụ thể, bao gồm các quyền như:

* **Create** (Tạo mới)
* **Read** (Đọc)
* **Write** (Chỉnh sửa)
* **Delete** (Xóa)

****1.2.Cấu trúc của Access Control List (ACL)****

Cấu trúc của **Access Control List (ACL)** trong Odoo xác định các quyền truy cập của người dùng đối với các mô hình dữ liệu trong hệ thống. Mỗi ACL có các thành phần sau:

* **model\_id**: Mô hình dữ liệu mà ACL này áp dụng. Mỗi ACL sẽ xác định quyền truy cập cho một model cụ thể (ví dụ: res.partner, sale.order, v.v.).
* **group\_id**: Nhóm người dùng mà ACL này áp dụng. Quyền truy cập sẽ được cấp cho một nhóm người dùng cụ thể trong hệ thống (ví dụ: Nhóm quản trị viên, Nhóm nhân viên bán hàng).
* **perm\_read**: Quyền đọc, cho phép người dùng trong nhóm này xem dữ liệu của model.
* **perm\_write**: Quyền sửa, cho phép người dùng chỉnh sửa dữ liệu trong model.
* **perm\_create**: Quyền tạo mới, cho phép người dùng tạo các bản ghi mới trong model.
* **perm\_unlink**: Quyền xóa, cho phép người dùng xóa các bản ghi khỏi model.

**Ví dụ**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id | name | model\_id:id | group\_id:id | perm\_read | perm\_write | perm\_create | perm\_unlink |
| access\_estate | Estate admin | model\_estate\_property | base.group\_user | 1 | 1 | 1 | 1 |

## **2. Record Rules**

**Record Rules** (Quy tắc bản ghi) là một cơ chế bảo mật trong Odoo, giúp kiểm soát quyền truy cập của người dùng vào các bản ghi cụ thể trong các mô hình dữ liệu. Các quy tắc này sử dụng **biểu thức domain** (điều kiện lọc) để xác định những bản ghi mà người dùng có thể xem hoặc thao tác, giúp phân quyền truy cập chi tiết và linh hoạt hơn.

#### ****Cấu trúc của Record Rules:****

**model\_id**:

* 1. **Mô tả**: Mô hình dữ liệu mà quy tắc này áp dụng. Quy tắc sẽ giới hạn quyền truy cập cho một mô hình dữ liệu cụ thể, ví dụ như res.partner, sale.order, v.v.

**domain\_force**:

* 1. **Mô tả**: Điều kiện lọc (domain) mà người dùng có quyền truy cập. Đây là biểu thức xác định những bản ghi mà người dùng có thể xem hoặc chỉnh sửa, ví dụ: [('user\_id', '=', user.id)].

**groups**:

* 1. **Mô tả**: Các nhóm người dùng mà quy tắc này áp dụng. Quy tắc này sẽ chỉ áp dụng cho các người dùng thuộc nhóm cụ thể, ví dụ: nhóm quản trị viên, nhóm nhân viên bán hàng, v.v.

# VI. Basic View

Odoo cung cấp nhiều loại **view** khác nhau để hiển thị dữ liệu (record set) theo các định dạng cụ thể, giúp người dùng dễ dàng thao tác và quản lý thông tin. Dưới đây là các loại view cơ bản trong Odoo:

1. **Form View**:

**Chức năng**: Hiển thị chi tiết của một bản ghi. Cho phép người dùng xem và chỉnh sửa thông tin của bản ghi đó.

**Ứng dụng**: Dùng để hiển thị thông tin chi tiết của một bản ghi duy nhất, ví dụ như chi tiết một đơn hàng, một khách hàng, v.v.

**2. Tree View**:

**Chức năng**: Hiển thị các bản ghi dưới dạng danh sách (lưới). Thường dùng để hiển thị nhiều bản ghi trong cùng một lúc.

**Ứng dụng**: Thường được sử dụng để hiển thị các danh sách lớn, như danh sách khách hàng, sản phẩm, đơn hàng, v.v.

**3. Kanban View**:

**Chức năng**: Hiển thị dữ liệu dưới dạng bảng điều khiển với các thẻ (card) có thể di chuyển, hữu ích cho các quy trình công việc.

**Ứng dụng**: Dùng để quản lý công việc, nhiệm vụ hoặc các quy trình theo dạng thẻ, có thể kéo thả thẻ giữa các trạng thái khác nhau (ví dụ: Từ "Chưa làm" sang "Đang làm" hay "Hoàn thành").

**4. Calendar View**:

**Chức năng**: Hiển thị dữ liệu dưới dạng lịch (calendar). Thường được sử dụng cho các sự kiện hoặc nhiệm vụ có ngày và thời gian.

**Ứng dụng**: Dùng để hiển thị và quản lý các sự kiện, cuộc hẹn, hoặc nhiệm vụ có thời gian, ví dụ như lịch làm việc, cuộc họp, v.v.

1. **Graph View**:

**Chức năng**: Hiển thị dữ liệu dưới dạng đồ thị (biểu đồ). Phù hợp để thể hiện số liệu, như biểu đồ thanh hoặc biểu đồ đường.

**Ứng dụng**: Dùng để phân tích và trực quan hóa dữ liệu, ví dụ như biểu đồ doanh thu, số lượng bán hàng theo thời gian, v.v.

1. **Pivot View**:

**Chức năng**: Hiển thị dữ liệu dưới dạng bảng điều khiển, giúp phân tích dữ liệu thông qua các phép tính tổng hợp (sum, count, average).

**Ứng dụng**: Dùng để phân tích số liệu chi tiết, như tổng doanh thu, số lượng sản phẩm bán ra, v.v. theo nhiều tiêu chí.

1. **Dashboard View**:

**Chức năng**: Cung cấp cái nhìn tổng quan về dữ liệu thông qua các widget, biểu đồ, và bảng điều khiển.

**Ứng dụng**: Dùng để hiển thị tổng hợp dữ liệu từ nhiều nguồn khác nhau trên một giao diện duy nhất, giúp người dùng theo dõi các chỉ số quan trọng trong thời gian thực.

# VII. Contrains

Trong Odoo, có hai loại **constraints** (ràng buộc) dùng để đảm bảo tính hợp lệ của dữ liệu: **SQL Constraints** và **Python Constraints**. Mỗi loại có cách sử dụng và phạm vi kiểm tra khác nhau.

## **1. SQL Constraints**

SQL constraints áp dụng trực tiếp ở cấp cơ sở dữ liệu để đảm bảo dữ liệu tuân thủ các điều kiện nhất định trước khi được lưu vào cơ sở dữ liệu. Các ràng buộc này giúp ngăn ngừa dữ liệu không hợp lệ ở mức độ cơ sở dữ liệu.

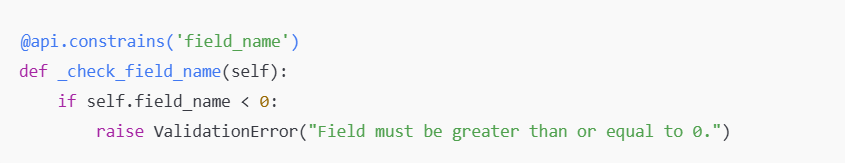
**Cú pháp**:



* constraint\_name: Tên của ràng buộc (unique trong mỗi mô hình).
* SQL\_condition: Điều kiện kiểm tra ràng buộc, thường là một câu lệnh SQL.
* Error\_message: Thông báo lỗi khi điều kiện ràng buộc không thỏa mãn.

## **2. Python Constraints**

Python constraints áp dụng trong mã Python để kiểm tra dữ liệu trước khi lưu vào cơ sở dữ liệu. Chúng có thể được sử dụng trong các phương thức của Odoo và được khai báo thông qua decorator **@api.constrains**. Phương thức này sẽ kiểm tra tính hợp lệ của dữ liệu và đưa ra thông báo lỗi nếu không thỏa mãn điều kiện.

Cú pháp:   


**Lưu ý**: Mỗi trường chỉ nên sử dụng một loại ràng buộc (SQL hoặc Python) để tránh làm giảm hiệu suất của hệ thống.

# VIII. Inherit View

Trong Odoo, việc **inherit view** cho phép kế thừa và sửa đổi một **view** đã có mà không cần thay đổi trực tiếp mã nguồn của nó. Điều này giúp bạn tùy chỉnh giao diện mà không làm thay đổi hệ thống gốc, bảo đảm tính linh hoạt và dễ dàng nâng cấp module.

#### ****Các thuộc tính của một Inherit View****:

name:

**Tên của inherit view**, giúp phân biệt các view khác nhau. Đây là trường bắt buộc.

model:

**Model** được định nghĩa trong view gốc mà inherit view này sẽ làm việc với. Thông thường, model này sẽ phải tương thích với model của view gốc.

inherit\_id:

**ID của view gốc** cần được inherit. Nếu view nằm trong module khác, bạn cần phải tham chiếu view theo cú pháp module.inherit\_id\_view.

**Ví dụ**: inherit\_id="module\_name.view\_id".

xpath:

**XPath** được sử dụng để xác định node (phần tử) trong view gốc mà bạn muốn thao tác hoặc thay đổi. XPath cho phép bạn chỉ định vị trí của phần tử mà bạn muốn sửa đổi trong view gốc.

position:

* 1. **Xác định vị trí xung quanh node (phần tử)** mà bạn muốn chèn thêm hoặc thay thế. Có 4 loại **position** như sau:
     1. position="replace": Thay thế phần tử được chọn bằng phần tử mới.
     2. position="before": Thêm phần tử mới trước phần tử được chọn.
     3. position="after": Thêm phần tử mới sau phần tử được chọn.
     4. position="inside": Thêm phần tử mới vào bên trong phần tử đã được chọn.



# IX. SQL View

Trong Odoo, **SQL View** là một đối tượng cơ sở dữ liệu được sử dụng để truy vấn dữ liệu từ nhiều bảng khác nhau mà không cần phải tạo ra một mô hình Odoo mới. Nó giúp tối ưu hóa các truy vấn phức tạp mà không cần viết mã Python. **SQL View** thực chất là một **bảng ảo** giúp hiển thị kết quả của một truy vấn SQL mà không cần phải lưu trữ dữ liệu thực tế.

Các đặc điểm của SQL View:

**Tạo và sử dụng SQL View**:

SQL View cho phép bạn kết hợp dữ liệu từ nhiều bảng trong cơ sở dữ liệu mà không cần phải sử dụng mô hình Odoo truyền thống.

Khi sử dụng SQL View, bạn chỉ cần định nghĩa một mô hình với truy vấn SQL cần thiết để truy xuất dữ liệu.

**Đặc điểm của SQL View**:

\_auto = False: Trong mô hình Odoo, bạn cần định nghĩa \_auto = False để chỉ rõ rằng đây không phải là một bảng thực sự mà là một bảng ảo (SQL View).

init(): Hàm init() sẽ được sử dụng để thực thi truy vấn SQL khi SQL View được gọi, giúp lấy dữ liệu và trả về các record set cần hiển thị.

**Định nghĩa View XML và Action**:

Phần **view XML** và **action** cho SQL View vẫn được định nghĩa giống như các view khác trong Odoo. Điều này bao gồm việc hiển thị kết quả của SQL View dưới dạng các bản ghi trong giao diện người dùng.

Mỗi khi view được gọi, hàm init() sẽ được thực thi để truy vấn dữ liệu từ cơ sở dữ liệu và kết quả sẽ được hiển thị trên giao diện.

Ví dụ:

@api.model\_cr

def init(self):

self.env.cr.execute("""

CREATE OR REPLACE VIEW my\_sql\_view AS (

SELECT

row\_number() OVER () AS id,

p.id, p.name, p.list\_price AS price

FROM product\_template p

WHERE p.active = TRUE

);

""")

**Lưu ý** : Phải có

row\_number() OVER () AS id,

đảm bảo rằng mỗi bản ghi trong view có một ID duy nhất

# X. Widget

**Widgets** là các thành phần giao diện người dùng (UI) tùy chỉnh, được sử dụng để cải tiến và mở rộng cách thức người dùng tương tác với hệ thống. Chúng giúp hiển thị thông tin theo một cách đặc biệt (như lịch, bảng điều khiển, các bảng tương tác) và cung cấp các chức năng đặc biệt như chọn ngày tháng, kéo thả, hoặc hiển thị dữ liệu động.

**Các yếu tố quan trọng về Widgets**:

* **Kế thừa từ AbstractField**:

Một widget thường kế thừa từ AbstractField, giúp quản lý các thuộc tính cơ bản và các phương thức cần thiết cho các widget.



* **Sự kiện (Events)**:

Các widget có thể khai báo trước các sự kiện để xử lý hành động người dùng. Những sự kiện này có thể bao gồm click, hover, hoặc bất kỳ hành động nào khác của người dùng.



* **Các hàm render quan trọng**:

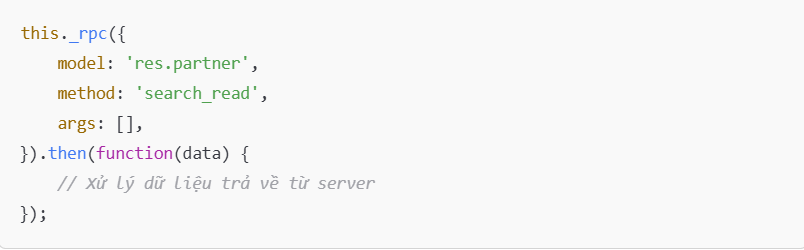
\_init: Hàm khởi tạo các biến và cấu hình ban đầu cần thiết cho widget.

\_renderEdit: Hàm được gọi khi view đang ở trạng thái **edit** (chỉnh sửa), dùng để render widget trong chế độ chỉnh sửa.

\_renderReadonly: Hàm được gọi khi view đang ở trạng thái **readonly** (chỉ đọc), dùng để render widget trong chế độ chỉ đọc.

* **Sử dụng RPC để tương tác với DB**:

Widgets có thể sử dụng **RPC (Remote Procedure Call)** để tương tác với cơ sở dữ liệu và lấy thông tin cần thiết cho việc render. Điều này giúp widget có thể cập nhật dữ liệu hoặc thực hiện các thao tác trực tiếp với server mà không cần phải tải lại toàn bộ trang.



# XI. Controller

Trong Odoo, **controller** (bộ điều khiển) là một thành phần quan trọng trong kiến trúc web framework, giúp xử lý các yêu cầu HTTP và định tuyến (routing) các URL đến các chức năng tương ứng. Controllers cho phép xây dựng các trang web động, xử lý các biểu mẫu (form), và cung cấp các dịch vụ web (web services).

Các tham số trong @http.route

@http.route là một decorator trong Odoo dùng để định tuyến các yêu cầu HTTP tới các phương thức trong controller. Nó kết nối một URL với một hàm xử lý cụ thể.

Các tham số chính của decorator này bao gồm:

/url:

Đây là URL mà controller sẽ lắng nghe và xử lý yêu cầu đến.

Ví dụ: /my/custom/page

type:

Xác định loại yêu cầu. Có thể là http cho các yêu cầu HTTP thông thường hoặc json cho yêu cầu trả về dữ liệu dạng JSON.

Ví dụ: type='json' sẽ cho phép controller trả về dữ liệu JSON.

auth:

Xác định ai có quyền truy cập vào URL hoặc trang web liên kết với URL đó. Có thể là user (yêu cầu đăng nhập người dùng) hoặc public (có thể truy cập công khai).

Ví dụ: auth='user' yêu cầu người dùng phải đăng nhập để truy cập.

website=True:

Nếu tham số này được đặt là True, controller sẽ liên kết với một trang web. Điều này thường được sử dụng khi xây dựng các trang web động trong Odoo.

Ví dụ: website=True giúp xác định rằng route này là một phần của trang web.

methods:

Đây là một danh sách các phương thức HTTP (như GET, POST, PUT, DELETE) mà route này áp dụng. Nếu không chỉ định, mặc định sẽ chấp nhận tất cả các phương thức HTTP.

Ví dụ: methods=['GET', 'POST'] chỉ định rằng route này chấp nhận cả yêu cầu GET và POST.

cors:

Xác định quyền truy cập tài nguyên từ các nguồn khác (Cross-Origin Resource Sharing). Điều này cho phép tài nguyên từ một miền có thể được truy cập bởi các trang web từ các miền khác.

Ví dụ: cors='\*' cho phép chia sẻ tài nguyên với tất cả các miền khác.

csrf:

Xác định xem có bảo vệ **CSRF** (Cross-Site Request Forgery) cho route này hay không. Mặc định, tham số này là True, giúp bảo vệ chống lại các cuộc tấn công CSRF.

Ví dụ: csrf=False sẽ tắt bảo vệ CSRF cho route này.

# XII. Snippets

Trong Odoo, **snippets** là các khối nội dung hoặc thành phần giao diện người dùng (UI) được thiết kế sẵn, cho phép người dùng kéo-thả và tùy chỉnh dễ dàng khi xây dựng trang web. Các snippets giúp người dùng tạo và chỉnh sửa các phần của trang web mà không cần phải có kiến thức về mã nguồn.

**Các thành phần cơ bản của một snippet**

1. **XML:**

* Định nghĩa giao diện và cấu trúc của snippet.
* Các snippet thường được định nghĩa trong file XML và có thể sử dụng các thẻ như t-call hoặc div để tạo cấu trúc giao diện.



1. **CSS**:

* Định nghĩa kiểu dáng (style) cho snippet.
* Các file CSS hoặc SCSS sẽ được sử dụng để tạo phong cách cho snippet, giúp thay đổi cách hiển thị và kiểu dáng của các phần tử trong snippet.



1. **Snippet Template**:

* Định nghĩa cách snippet được nhận diện trong giao diện kéo-thả.
* Snippet template được khai báo trong file XML và phải kế thừa từ website.snippets. Thông qua đó, ta có thể xác định các thuộc tính như template ID và hình ảnh thumbnail cho snippet.
* <t t-snippet="module.template\_id" t-thumbnail="path "/> để xác định template cho snippets

|  |  |
| --- | --- |
| Editor Tab | XPath Expression |
| Snippet structure | //div[@id='snippet\_structure'] |
| Content | //div[@id='snippet\_content'] |
| Feature | //div[@id='snippet\_feature'] |
| Effect | //div[@id='snippet\_effect'] |

# XIII. Polish-notations

**Prefix Notation** (hoặc **Polish Notation**) là cách viết biểu thức toán học trong đó **toán tử** được đặt **trước** các **toán hạng**. Cách viết này giúp loại bỏ việc sử dụng dấu ngoặc để xác định thứ tự thực hiện phép toán, đồng thời giải quyết vấn đề thứ tự ưu tiên của các phép toán.

Khi chuyển một biểu thức từ **Infix Notation** (dạng thông thường, ví dụ như A + B) sang **Prefix Notation**, bạn thực hiện theo các bước dưới đây:

**Cách chuyển từ Infix sang Prefix Notation:**

1. **Đặt toán tử trước các toán hạng**:

Trong Prefix Notation, toán tử sẽ luôn đứng ở vị trí đầu tiên, theo sau là các toán hạng.

1. **Đảm bảo thứ tự thực hiện các phép toán**:

Thứ tự các phép toán phải được duy trì giống như trong biểu thức Infix, nhưng không cần sử dụng dấu ngoặc để phân tách.

Các phép toán sẽ được thực hiện từ trái qua phải theo đúng thứ tự ưu tiên toán học.

Ví dụ:

* **Infix Notatio**: ((A AND B) OR C ) OR (D AND E)
* **Prefix Notation**: ['|', '|', '&', A, B, C, '&', D, E]

# XIII. Một vài widget

* **widget="char"** - Hiển thị dữ liệu văn bản.
* **widget="date"** - Hiển thị và chọn ngày.
* **widget="datetime"** - Hiển thị và chọn ngày giờ.
* **widget="many2one"** - Hiển thị mối quan hệ Many2one dưới dạng lựa chọn (dropdown).
* **widget="one2many"** - Hiển thị mối quan hệ One2many.
* **widget="many2many"** - Hiển thị mối quan hệ Many2many.
* **widget="selection"** - Hiển thị lựa chọn dưới dạng danh sách thả xuống.
* **widget="html"** - Hiển thị nội dung HTML.
* **widget="image"** - Hiển thị ảnh.
* **widget="progressbar"** - Hiển thị thanh tiến trình.
* **widget="boolean"** - Hiển thị giá trị boolean (True/False) dưới dạng checkbox.
* **widget="monetary"** - Hiển thị giá trị tiền tệ.
* **widget="file"** - Cho phép tải lên và hiển thị tệp tin.
* **widget="color"** - Hiển thị và chọn màu sắc.
* **widget="rating"** - Hiển thị đánh giá bằng các ngôi sao.