Odoo ERP

Mục lục

[I. Cấu trúc Model 2](#_Toc183810882)

[1. Định nghĩa 2](#_Toc183810883)

[2. Cấu trúc của model 2](#_Toc183810884)

[II. Khởi tạo Data 3](#_Toc183810885)

[1. Master Data 3](#_Toc183810886)

[2. Demo Data 3](#_Toc183810887)

[3. So sánh giữa Master Data và Demo Data 4](#_Toc183810888)

[III. Decoration 4](#_Toc183810889)

[IV. Security 4](#_Toc183810890)

[1. Access Control Lists (ACLs) 5](#_Toc183810891)

[2. Record Rules 5](#_Toc183810892)

[V. Basic View 5](#_Toc183810893)

[VI. Contrains 6](#_Toc183810894)

[1. SQL Constraints 6](#_Toc183810895)

[2. Python Constraints 6](#_Toc183810896)

[VI. Inherit View 7](#_Toc183810897)

[VII. SQL View 8](#_Toc183810898)

[VIII. Widget 8](#_Toc183810899)

[IX. Controller 10](#_Toc183810900)

[X. Snippets 10](#_Toc183810901)

[XI. Giảm thiểu độ phức tạp trong lập trình 10](#_Toc183810902)

[XII. Polish-notations 13](#_Toc183810903)

[XIII. Một vài widget 13](#_Toc183810904)

[XIV. Module sản xuất (Manufacturing) 14](#_Toc183810905)

# I. Cấu trúc Model

## 1. Định nghĩa

Model là thành phần đại diện cho các đối tượng cần thiết cho module và được ánh xạ qua các table tầng database thông qua ORM.

## 2. Cấu trúc của model

**2.1 Phần định nghĩa**

* Một model sẽ được kế thừa từ models.Model.
* Được đặt tên bằng thuộc tính \_name
* Được mô tả bằng thuộc tính \_description
* Có thể sắp xếp bằng thuộc tính \_order
* Kế thừa thuộc tính từ bảng khác bằng thuộc tính \_inherit

**2.2 Phần trường của model (Field)**

**Trường đơn giản:**

* **Char**: Chuỗi ký tự (tương tự VARCHAR).
* **Text**: Chuỗi văn bản dài.
* **Html**: Văn bản định dạng HTML.
* **Integer**: Số nguyên.
* **Float**: Số thập phân.
* **Boolean**: Kiểu đúng/sai.
* **Date**: Ngày (yyyy-mm-dd).
* **Datetime**: Ngày giờ (yyyy-mm-dd hh:mm:ss).
* **Selection**: Danh sách các giá trị chọn.

**Trường quan hệ:**

* **Many2one**: Liên kết đến một bản ghi trong model khác (quan hệ N:1).
* **One2many**: Liên kết đến nhiều bản ghi khác (quan hệ 1:N).
* **Many2many**: Liên kết đến nhiều bản ghi trong model khác (quan hệ N:N).

**2.3 Phần thuộc tính của trường (Field Attribute)**

* **string:** Tên hiển thị của trường trên giao diện người dùng.
* **required**: Bắt buộc người dùng phải nhập giá trị cho trường.
* **default:** Giá trị mặc định của trường.
* **help:** Ghi chú hoặc hướng dẫn cho người dùng về trường.
* **readonly:** Chỉ cho phép đọc, không thể chỉnh sửa.
* **index:** Tạo chỉ mục trên trường trong cơ sở dữ liệu để tăng hiệu suất truy vấn.
* **compute:** Định nghĩa hàm tính toán giá trị tự động cho trường.
* **store:** Lưu giá trị đã tính toán vào cơ sở dữ liệu.
* **inverse:** Hàm dùng để cập nhật giá trị tính toán ngược.
* **search:** Hàm dùng để tùy chỉnh cách tìm kiếm giá trị.
* **related:** Liên kết giá trị từ trường khác.
* **ondelete:** Kiểm soát hành vi khi xóa bản ghi liên quan (cascade, restrict, set null).
* **domain:** Giới hạn giá trị có thể chọn cho trường (Many2one, Many2many).
* **translate:** Cho phép dịch giá trị của trường sang nhiều ngôn ngữ.
* **tracking:** Theo dõi các thay đổi trên trường.
* **copy:** Cho phép sao chép giá trị của trường khi tạo bản ghi mới.
* **groups:** Giới hạn quyền truy cập dựa trên nhóm người dùng.
* **widget:** Xác định kiểu giao diện hiển thị của trường .

**2.4 Hàm đặc biệt (decorator)**

* **@api.depends :** Được dùng để tính toán các trường tính toán (computed fields).
* **@api.depends\_context :** Được dùng để tính toán như hàm depend những sẽ phụ thuộc vào context
* **@api.onchange:** Tính toán lại khi một trường được thay đổi trong giao diện người dùng.
* **@api.constrains :**Dùng để kiểm tra ràng buộc dữ liệu khi lưu bản ghi (validation)
* **@api.autovacuum :**Dùng để dọn dẹp dữ liệu.

# II. Khởi tạo Data

## 1. Master Data

Là các dữ liệu chính, được sử dụng trong hệ thống. Các bản ghi này cung cấp dữ liệu cho hoạt động cho module.

## 2. Demo Data

Là dữ liệu mẫu, dùng để minh họa cách thức hoạt động của hệ thống, phục vụ cho việc thử nghiệm, đào tạo hoặc tạo bản demo cho khách hàng.

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

|  |  |  |
| --- | --- | --- |
|  | **Master Data** | **Demo** |
| **Mục đích sử dụng** | |  | | --- | | Dữ liệu nền tảng cho các hoạt động trong hệ thống. |  |  | | --- | |  | | |  | | --- | | Dữ liệu mẫu dùng để trình diễn, thử nghiệm hoặc đào tạo. |  |  | | --- | |  | |
| **Ảnh hưởng đến hệ thống** | |  | | --- | | Có ảnh hưởng trực tiếp đến các business logic trong hệ thống. |  |  | | --- | |  | | |  | | --- | | Không ảnh hưởng đến các quy trình | |
| **Nơi khai báo** | Trong thẻ data | Trong thẻ demo |
| **Lúc khởi tạo** | Install hoặc Upgrade Module | Install Module |

# III. Decoration

Decoration được sử dụng để định dạng hoặc thay đổi hiển thị của các trường và bản ghi trên giao diện

* **decoration-danger:** Hiển thị bản ghi với màu đỏ, thường dùng để chỉ lỗi hoặc cảnh báo.
* **decoration-warning:** Hiển thị bản ghi với màu cam, dùng để nhắc nhở người dùng.
* **decoration-success:** Hiển thị bản ghi với màu xanh lá cây, dùng để chỉ trạng thái hoàn thành.
* **decoration-muted:** Hiển thị bản ghi với màu xám, thường dùng để chỉ trạng thái không hoạt động hoặc ẩn.
* **decoration-primary:** Hiển thị bản ghi với màu xanh dương, dùng để làm nổi bật hoặc cho trạng thái quan trọng.
* **decoration-info:** Hiển thị bản ghi với màu xanh nhạt, dùng để cung cấp thông tin.
* **decoration-bf:** Hiển thị bản ghi với chữ đậm (Bold).
* **decoration-it:** Hiển thị bản ghi với chữ nghiêng (Italic).
* **decoration-strike:** Hiển thị bản ghi với chữ gạch ngang, thường dùng cho trạng thái bị hủy hoặc không còn hiệu lực.

# IV. Security

Odoo cung cấp nhiều cơ chế bảo mật khác nhau, bao gồm quyền truy cập cấp độ mô-đun, record rules (quy tắc bản ghi), và access control lists (ACLs).

## 1. Access Control Lists (ACLs)

ACLs trong Odoo là một cách để xác định quyền truy cập của người dùng vào các mô-đun hoặc các đối tượng trong hệ thống. ACLs xác định quyền mà một nhóm người dùng có đối với các mô hình dữ liệu (models), chẳng hạn như **create**, **read**, **write**, **delete** (CRUD).

**Cấu trúc của ACL:**

* **model\_id**: Mô hình dữ liệu mà ACL này áp dụng.
* **group\_id**: Nhóm người dùng có quyền này.
* **perm\_read**: Quyền đọc.
* **perm\_write**: Quyền sửa.
* **perm\_create**: Quyền tạo mới.
* **perm\_unlink**: Quyền xóa.

**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 là các quy tắc bảo mật giúp kiểm soát quyền truy cập vào các bản ghi cụ thể trong các mô hình dữ liệu. Các quy tắc này có thể được định nghĩa bằng cách sử dụng biểu thức domain (điều kiện lọc) để xác định những bản ghi nào người dùng có thể truy cập.

Cấu trúc của Record Rules:

* **model\_id:** Mô hình dữ liệu mà quy tắc này áp dụng.
* **domain\_force:** Điều kiện lọc dữ liệu mà người dùng có quyền truy cập.
* **groups:** Các nhóm người dùng mà quy tắc này áp dụng.

# V. Basic View

Các basic view có chức năng hiển thị các record set theo một định dạng nhất định tùy theo mỗi view và cách định nghĩa view :

* **Form View:** Hiển thị chi tiết của một bản ghi. Cho phép người dùng xem và chỉnh sửa thông tin của bản ghi đó.
* **Tree View:** Hiển thị các bản ghi dưới dạng danh sách (lưới). Thường dùng để hiển thị nhiều bản ghi trong cùng một lúc.
* **Kanban View:** 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.
* **Calendar View:** Hiển thị dữ liệu dưới dạng lịch (calendar). Thường được sử dụng cho các sự kiện hoặc nhiệm vụ có ngày và thời gian.
* **Graph View:** 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.
* **Pivot View:** Hiển thị dữ liệu dưới dạng bảng điều khiển, giúp phân tích dữ liệu thông qua các phép tính tổng hợp (sum, count, average).
* **Dashboard View:** Cung cấp cái nhìn tổng quan về dữ liệu thông qua các widget, biểu đồ, và bảng điều khiển.

# VI. Contrains

## 1. SQL Constraints

SQL constraints được sử dụng để đảm bảo tính hợp lệ của dữ liệu trực tiếp ở cấp cơ sở dữ liệu. Chúng có thể được áp dụng khi bạn tạo một model và cần đảm bảo rằng dữ liệu phải tuân theo một số điều kiện nhất định.

SQl được khai báo trong model theo cú pháp :

* \_sql\_constraints=[(`name`,`condition `,`string error`)]

## 2. Python Constraints

Python constraints được sử dụng trong mã Python để kiểm tra dữ liệu và đưa ra thông báo lỗi nếu cần. Chúng có thể sử dụng trong các phương thức (method) của Odoo để xác minh dữ liệu trước khi lưu vào cơ sở dữ liệu.

Python constraints được khai báo thông qua decorator @api.constrains(‘field\_name’)

*Lưu ý: Mỗi field chỉ nên sử dụng một trong hai constraints để tránh tiêu hao hiệu suất*

Ví dụ:

@api.constrains('age')

def \_check\_age(self):

for record in self:

if record.age < 18:

raise ValidationError('Age must be greater than or equal to 18!')

# VI. Inherit View

Trong Odoo, việc inherit view cho phép kế thừa và sửa đổi một view đã có trong hệ thống mà không cần phải thay đổi trực tiếp mã nguồn của nó.

Các thuộc tính của một inherit view

* name : tên của inherit view
* model : model được định nghĩa ở view gốc
* inherit\_id : id của view gốc cần inherit. Nếu view nằm ở module khác thì cấu trúc trỏ đến là module.inherit\_id\_view
* xpath : trỏ đến node cần chèn trong view gốc
* position : xác định vị trí xung quanh node được trỏ đến. Có 4 loại position :
  + position="replace": Thay thế phần tử được chọn bởi phần tử mới.
  + position="before": Thêm phần tử mới trước phần tử được chọn.
  + position="after": Thêm phần tử mới sau phần tử được chọn.
  + position="inside": Thêm phần tử vào bên trong phần tử được chọn.

Ví dụ :

<odoo>

<data>

<record id="custom\_view\_form\_inherit" model="ir.ui.view">

<field name="name">custom.view.form.inherit</field>

<field name="model">res.partner</field>

<field name="inherit\_id" ref="base.view\_partner\_form"/>

<field name="arch" type="xml">

<xpath expr="//field[@name='name']" position="before">

<field name="code"/>

</xpath>

</field>

</record>

</data>

</odoo>

# VII. SQL View

Trong Odoo, SQL View là một loại đối tượng mà có thể tạo trong cơ sở dữ liệu để truy vấn dữ liệu từ các bảng khác nhau mà không cần phải tạo ra một mô hình Odoo mới. Điều này giúp tối ưu hóa các truy vấn phức tạp mà không cần phải viết mã Python phức tạp. SQL View thực chất là một bảng ảo, có thể giúp hiển thị kết quả của một truy vấn SQL.

Phần view xml và action vẫn được khởi tạo và định nghĩa như các view khác. Mỗi khi gọi đến view xml và hàm init sẽ được khởi chạy và nhận về các record set để hiển thị lên view.

Một SQL View sẽ có \_auto = False trong phần định nghĩa model và hàm init để truy vấn SQL:

Ví dụ:

@api.model\_cr

def init(self):

self.env.cr.execute("""

CREATE OR REPLACE VIEW my\_sql\_view AS (

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

FROM product\_template p

WHERE p.active = TRUE

);

""")

# VIII. Widget

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

Các kiến thức cần chú ý

* Một widget sẽ kế thừa từ một AbstractField

var ColorPickerWidget = AbstractField.extend({

* Có các sự kiện được khai báo trước

events: {  
 'click .o\_color\_pill': 'clickColorPill',  
},

* Các hàm quan trọng để render widget :

\_init: Khởi tạo các biến cần thiết để widget sử dụng

\_renderEdit : render khi view ở trạng thái edit

\_renderReadonly : render khi view ở trạng thái readonly

* Widget cũng có thể sử dụng RPC để thực hiện các tương tác với DB để nhận về thông tin mong muốn phục vụ quy trình render.

\_getFunction: function () {  
 var self = this;  
 return rpc.query({  
 model: model',  
 method: 'model\_method',  
 args: []  
 }).then(function (result) {  
 self.variable = result;   
 });  
},

# IX. Controller

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

Các tham số trong @http.route

**@http.route**: Đây là một decorator được sử dụng cho mục đích định tuyến (routing), cho phép điều hướng đến các trang cụ thể hoặc liên kết một URL nhất định với một trang web.

Các tham số chính của decorator này bao gồm:

* **/url**: URL của controller.
* **type**: Loại yêu cầu, có thể là yêu cầu 'http' hoặc 'json'.
* **auth**: Xác định ai có quyền truy cập URL này hoặc xem trang web liên kết với URL đó.
* **website=True**: Xác định rằng controller này liên kết với một trang web.
* **methods**: Dãy các phương thức HTTP mà route này áp dụng. Nếu không chỉ định, tất cả các phương thức sẽ được phép (ví dụ: methods=['GET', 'POST']).
* **cors**: Chia sẻ tài nguyên qua các nguồn khác (Cross-Origin Resource Sharing), cho phép truy cập tài nguyên từ các miền khác (ví dụ: cors="\*").
* **csrf**: Xác định xem có bảo vệ CSRF cho route này không, mặc định là True

# X. Snippets

Snippet liên quan đến giao diện người dùng (UI) và được sử dụng trong module Website Builder. Snippet là các khối nội dung hoặc thành phần giao diện được thiết kế sẵn, giúp người dùng kéo-thả và tùy chỉnh dễ dàng khi xây dựng trang web.

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

1. **XML**: Định nghĩa giao diện và cấu trúc của snippet.
   * Nằm trong file XML, thường sử dụng t-call hoặc div để tạo khối giao diện.
2. **CSS**: Định nghĩa kiểu dáng (style) cho snippet.
   * Nằm trong file .scss hoặc .css.
3. **Snippet Template**: Định nghĩa cách snippet được nhận diện trong giao diện kéo-thả.
   * Nằm trong file XML và kế thừa website.snippets và sẽ tùy theo thể loại snippets mà ta đưa (xem bảng dưới) và kế đến là thẻ

<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'] |

# XI. Giảm thiểu độ phức tạp trong lập trình

**SOLID**

* **S** - **Single Responsibility Principle (SRP)**: Mỗi lớp chỉ nên có một lý do để thay đổi, tức là mỗi lớp chỉ nên có một trách nhiệm duy nhất.
* - **Open/Closed Principle (OCP)**: Các lớp nên được mở để mở rộng nhưng đóng để sửa đổi, có nghĩa là bạn có thể thêm tính năng mới mà không cần thay đổi mã hiện tại.
* **L** - **Liskov Substitution Principle (LSP)**: Các lớp con phải có thể thay thế các lớp cha mà không làm thay đổi tính đúng đắn của chương trình.
* **I** - **Interface Segregation Principle (ISP)**: Không nên ép các lớp implement các giao diện mà chúng không sử dụng. Tạo ra các giao diện nhỏ, chuyên biệt hơn là tốt hơn.
* **D** - **Dependency Inversion Principle (DIP)**: Các mô-đun cấp cao không nên phụ thuộc vào các mô-đun cấp thấp, mà thay vào đó cả hai nên phụ thuộc vào các abstractions (giao diện).

**KISS**

Nguyên tắc KISS, viết tắt của "Keep It Simple, Stupid", đã được sử dụng trong nhiều bối cảnh để truyền tải ý tưởng rằng các hệ thống thường hoạt động tốt nhất nếu được giữ đơn giản. Nguyên tắc này có thể áp dụng cho việc thiết kế các hệ thống phần mềm. Một nhóm phát triển nên cố gắng không làm cho các giải pháp của họ trở nên phức tạp hơn cần thiết.

**Nhưng đừng làm nó quá đơn giản**

**Ví dụ tình huống:** Bạn đang phát triển một hệ thống quản lý đơn hàng cho một cửa hàng trực tuyến.

**Thiết kế đơn giản:** Bạn quyết định rằng người dùng chỉ cần nhập tên sản phẩm và số lượng để tạo một đơn hàng. Bạn loại bỏ tất cả các trường khác như địa chỉ giao hàng, thông tin thanh toán, và phương thức giao hàng vì bạn nghĩ rằng điều này sẽ giúp thiết kế đơn giản hơn.

**Vấn đề:** Dù thiết kế này đơn giản nhưng nó dẫn đến nhiều vấn đề:

* Người dùng không thể cung cấp thông tin cần thiết để xử lý đơn hàng.
* Hệ thống không thể xác định được nơi nào để giao hàng, gây ra sự nhầm lẫn và chậm trễ.
* Không có cách nào để xác minh thanh toán, dẫn đến việc có thể xảy ra gian lận.

**Thiết kế hợp lý:** Thay vì chỉ yêu cầu tên sản phẩm và số lượng, bạn giữ lại các trường quan trọng như địa chỉ giao hàng, thông tin thanh toán và phương thức giao hàng. Tuy nhiên, bạn có thể sử dụng các mẫu điền tự động hoặc dropdown để người dùng dễ dàng lựa chọn và điền thông tin, nhằm giảm bớt sự phức tạp

**DRY**

Nguyên tắc DRY khuyến khích các nhà phát triển giảm thiểu tối đa việc viết lại cùng một đoạn mã hoặc logic tại nhiều nơi khác nhau trong dự án. Mỗi phần thông tin hoặc logic nên có một đại diện duy nhất, đảm bảo rằng khi cần thay đổi hoặc sửa lỗi, chỉ cần chỉnh sửa tại một chỗ. Điều này giúp mã nguồn dễ đọc, dễ bảo trì và giảm thiểu lỗi phát sinh khi có các thay đổi.

**Các hình thức trùng lặp cần tránh**

* **Sao chép và dán (Copy-and-Paste Programming):** Việc sao chép mã từ chỗ này sang chỗ khác dễ dẫn đến trùng lặp logic, và khi cần thay đổi, có thể dẫn đến lỗi do thiếu đồng nhất.
* **Chuỗi "magic strings":** Đây là những chuỗi tĩnh xuất hiện nhiều lần trong mã, chẳng hạn như các khóa, URL hoặc đường dẫn tệp. Việc sử dụng "magic strings" nhiều lần khiến hệ thống khó bảo trì, vì nếu cần thay đổi một chuỗi, phải tìm và sửa ở mọi nơi nó xuất hiện.

**Trùng lặp logic:** Khi logic xuất hiện lặp lại trong nhiều module hoặc lớp, rất dễ gây lỗi và khó duy trì nếu cần thay đổi logic này.

**YAGNI**

YAGNI (You Aren't Gonna Need It) là một nguyên tắc trong phát triển phần mềm, đặc biệt là trong phát triển Agile, nhằm nhắc nhở lập trình viên rằng chỉ nên tập trung vào những tính năng cần thiết và không nên phát triển các tính năng hoặc chức năng không cần thiết tại thời điểm hiện tại. Nguyên tắc này giúp giảm thiểu việc lãng phí tài nguyên và tránh làm cho phần mềm trở nên phức tạp hơn mức cần thiết.

* **Khi nào nên áp dụng YAGNI**:
  + Khi nhóm phát triển phải quyết định có nên thêm một tính năng chỉ vì "có thể sẽ cần" trong tương lai, YAGNI khuyên rằng không nên làm vậy. Chỉ nên phát triển những gì hiện tại cần ngay và đã có yêu cầu rõ ràng.
  + YAGNI cũng áp dụng khi thiết kế cơ sở dữ liệu và API, tránh việc tạo ra các cấu trúc phức tạp hoặc trường dữ liệu chưa có yêu cầu sử dụng.
* **Khi nào không nên áp dụng YAGNI**:
  + Đối với những hệ thống đòi hỏi tính bảo trì cao và khả năng mở rộng trong tương lai, đôi khi việc chuẩn bị trước là cần thiết để tránh chi phí điều chỉnh quá lớn. Ví dụ, thiết kế kiến trúc phần mềm cần tính đến khả năng mở rộng và thay đổi lâu dài để đảm bảo hệ thống dễ bảo trì và không bị giới hạn bởi các quyết định ban đầu.
  + Các yếu tố liên quan đến bảo mật, hiệu năng, và tính khả dụng của hệ thống thường cần phải được dự đoán trước để đảm bảo an toàn và trải nghiệm người dùng tốt nhất.

Ví dụ :

Giả sử một công ty đang phát triển một ứng dụng quản lý bán hàng. Nhóm phát triển cân nhắc việc xây dựng chức năng hỗ trợ đa tiền tệ vì nghĩ rằng "có thể" mở rộng ra các thị trường quốc tế trong tương lai. Tuy nhiên, hiện tại, ứng dụng chỉ phục vụ khách hàng trong nước với một loại tiền tệ.

Theo nguyên tắc YAGNI, nhóm phát triển nên tập trung vào tính năng quản lý đơn vị tiền tệ hiện có thay vì xây dựng một hệ thống đa tiền tệ phức tạp mà chưa có yêu cầu cụ thể. Khi công ty mở rộng ra quốc tế và có nhu cầu rõ ràng, việc phát triển tính năng đó sẽ hợp lý hơn và sẽ phù hợp với nhu cầu cụ thể khi đó.

# XII. Polish-notations

Prefix Notation (hay còn gọi là Polish Notation) là một cách viết biểu thức toán học trong đó toán tử được đặt trước các toán hạng. Cách viết này không cần sử dụng dấu ngoặc để xác định thứ tự thực hiện các phép toán, giúp loại bỏ được các vấn đề về thứ tự ưu tiên của các phép toán.

Khi chuyển một biểu thức từ dạng Infix Notation (biểu thức thông thường mà chúng ta sử dụng, ví dụ như A + B) sang Prefix Notation, bạn thực hiện theo các bước sau:

1. Đặt toán tử trước các toán hạng.
2. Đảm bảo thứ tự thực hiện các phép toán theo quy tắc toán học mà không cần dấu ngoặc.

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.

# XIV. Module sản xuất (Manufacturing)

**Nguyên vật liệu (Bill Of Material)**

Bill of Materials (BOM) trong Odoo là một danh sách các nguyên liệu, linh kiện hoặc thành phần cần thiết để sản xuất một sản phẩm hoàn chỉnh. BOM là một phần quan trọng trong quy trình sản xuất, giúp quản lý và kiểm soát các vật liệu và quy trình cần thiết để tạo ra sản phẩm cuối cùng.

**Tuyến cung ứng**

Tuyến cung ứng là quá trình liên quan đến tất cả các hoạt động và hệ thống cần thiết để cung cấp sản phẩm hoặc dịch vụ từ nhà cung cấp đến khách hàng cuối cùng. Tuyến cung ứng bao gồm việc cung cấp nguyên liệu thô, sản xuất, vận chuyển, lưu trữ và phân phối sản phẩm.

Tuyến cung ứng cơ bản 3 bước trong Odoo sẽ là nguyên liệu sẽ được lấy từ kho đến tập trung ở Pre-Production sau đó sẽ được đưa đến Manufacture để sản xuất. Sản xuất sẽ chạy qua các giai đoạn sẽ được đưa về Post-Production để tập trung và đưa về Warehouse.

Các tuyến cung ứng sẽ còn tùy thuộc vào công ty sản xuất được tùy chỉnh sao cho phù hợp với quy trình sản xuất.

**Công đoạn**

Một công đoạn sản xuất (work center) là nơi mà các công việc sản xuất cụ thể diễn ra. Mỗi công đoạn này có thể liên quan đến một máy móc, một nhóm công nhân, hoặc một quy trình sản xuất đặc biệt. Công đoạn giúp phân chia quy trình sản xuất thành các bước cụ thể để theo dõi hiệu suất và chi phí của từng công đoạn.

**Lệnh sản xuất**

Lệnh sản xuất (Manufacturing Order) trong Odoo là một tài liệu quan trọng trong quá trình sản xuất, được sử dụng để quản lý và theo dõi sản xuất các sản phẩm từ nguyên liệu thô. Lệnh sản xuất bao gồm thông tin chi tiết về sản phẩm cần sản xuất, các công đoạn (work centers), nguyên liệu (BOM), và các bước thực hiện trong quy trình sản xuất.