**all attribute are inside a models.py in this class**

class BaseModel(metaclass=MetaModel):

**1. \_order Attribute**

the \_**order** model attribute is used to define the default sorting order of records when retrieving them from the database. This attribute is specified inside a model class and determines how records are sorted when no specific order is provided in a query.

**Syntax:**

class YourModel(models.Model):

\_name = "your.model"

\_order = "field\_name ASC, another\_field\_name DESC"

**How It Works**

* The \_order attribute takes a string containing one or more field names followed by ASC (ascending) or DESC (descending).
* If ASC or DESC is not specified, ASC is used by default.
* If multiple fields are specified, they are used in sequence for sorting.

**Example:**

class ProductCategory(models.Model):

\_name = "product.category"

\_order = "name ASC"

class Task(models.Model):

\_name = "project.task"

\_order = "sequence ASC, name DESC"

**Default Behavior**

If \_order is not defined, Odoo sorts records by the id field in ascending order (id ASC).

**Overriding the Order in Queries**

tasks = self.env['project.task'].search([], order="date\_deadline DESC")

**2. \_rec\_name Attribute** **(Similar to name\_get Method)**

the \_rec\_name ORM attribute is used to specify which field should be used as the **display name** of a record in **Many2one fields** and other selection lists.

**How \_rec\_name Works?**

* By default, Odoo uses the **name field** as the display name.
* If a model **does not** have a name field, Odoo will throw an error in Many2one fields.
* \_rec\_name allows you to define a different field (e.g., email, code, etc.) as the display name.

**Example:**

**1. Using \_rec\_name to Show Email Instead of Name**

from odoo import models, fields

class Student(models.Model):

\_name = "practice.student"

\_rec\_name = "email" # Use 'email' as the display name in Many2one fields

name = fields.Char(string="Student Name")

email = fields.Char(string="Email")

Output: In a Many2one field, students will be displayed using their **email** instead of their name.

**2. Using \_rec\_name for a Model Without a name Field**

class ProductCategory(models.Model):

\_name = "product.category"

\_rec\_name = "code" # Use 'code' as the display name

code = fields.Char(string="Category Code", required=True)

description = fields.Text(string="Description")

Output: In a Many2one field, categories will be displayed using their code instead of an undefined name.

**Comparison: \_rec\_name vs name\_get**

| **Feature** | **\_rec\_name** | **name\_get** |
| --- | --- | --- |
| Simplicity | Defines a single field for display | Can combine multiple fields for display |
| Customization | Limited (only one field) | Highly customizable (multiple fields, formatting) |
| Performance | Faster (Direct DB query) | Slightly slower (Python processing) |
| Example Output | student.email | student.name (student.age) |

**When to Use \_rec\_name vs name\_get?**

✅ Use \_rec\_name when you **only need one field** as the display name.  
✅ Use name\_get when you need a **custom format** combining multiple fields.

**3.** **\_rec\_name\_search Attribute (Similar to name\_search Method)**

he \_rec\_name\_search attribute is used in Odoo 16+ to define the fields that should be considered when searching for records in Many2one fields. It is an enhancement to the traditional name\_search method, allowing you to specify multiple fields to search by, rather than just relying on the default \_rec\_name field (usually name).

**How \_rec\_name\_search Works**

The \_rec\_name\_search attribute allows you to specify a list of fields that will be searched when a user types into a Many2one field search box. This gives you more flexibility in search behavior, especially when you want to search for records based on multiple fields.

**Syntax:**

class MyModel(models.Model):

\_name = 'my.model'

\_rec\_name = 'name' # Default display field

# Fields to search for when a user types into the Many2one search box

\_rec\_name\_search = ['name', 'email']

**Example:**

**1. Basic Usage of \_rec\_name\_search**

class Student(models.Model):

\_name = 'practice.student'

\_rec\_name = 'name'

name = fields.Char(string="Student Name", required=True)

email = fields.Char(string="Email")

\_rec\_name\_search = ['name', 'email'] # Search by both name and email

**2. earch Across Multiple Fields**

class Product(models.Model):

\_name = 'product.product'

\_rec\_name = 'product\_name'

product\_name = fields.Char(string="Product Name")

sku = fields.Char(string="SKU")

description = fields.Text(string="Description")

\_rec\_name\_search = ['product\_name', 'sku', 'description'] # Search by name, SKU, and description

**Difference Between \_rec\_name\_search and name\_search**

| **Feature** | **\_rec\_name\_search** | **name\_search** |
| --- | --- | --- |
| **Odoo Version** | Available in Odoo 16+ | Available in all versions of Odoo |
| **Use Case** | Specify which fields to search in Many2one | Full customization of search logic (joins, filtering) |
| **Customization** | Only simple list of fields to search | Can include complex search logic (AND, OR, custom filters) |
| **Performance** | More efficient for basic use cases | Potentially slower for complex queries |

**When to Use \_rec\_name\_search?**

* **Odoo 16+** introduces this feature for simpler search customization.
* Use it when you want to **search across multiple fields** in a **Many2one field** without needing to override the name\_search method.
* It’s **faster** and **easier** for basic use cases compared to name\_search.

**When to Use name\_search Instead?**

* If you need **advanced search logic**, such as filtering by specific conditions, handling joins with related models, or more complex search queries, you should override the **name\_search** method.
* **name\_search** is more flexible and can be used when the search behavior needs to be more complex than just checking fields.

**Summary**

* **\_rec\_name\_search** simplifies the process of specifying which fields should be used for searching in **Many2one** fields. It is available in **Odoo 16+** and offers an easy way to search across multiple fields without the need to override the name\_search method.
* Use **\_rec\_name\_search** when you only need to search across multiple fields. For more advanced filtering and logic, override the name\_search method.

**Odoo Model Attributes with Examples**

**\_name**

* **Description**: Defines the technical name of the model (used as the database table identifier).
* **Example**:
* class Partner(models.Model):
* \_name = 'res.partner'

**\_description**

* **Description**: Provides a human-readable description of the model.
* **Example**:
* class Partner(models.Model):
* \_name = 'res.partner'
* \_description = 'Partner Information'

**\_inherit**

* **Description**: Used to extend the functionality of an existing model.
* **Example**:
* class ExtendedPartner(models.Model):
* \_inherit = 'res.partner'
* extra\_field = fields.Char(string="Extra Field")

**\_inherits**

* **Description**: Implements delegation inheritance to reuse fields from another model in the current model.
* **Example**:
* class Employee(models.Model):
* \_name = 'hr.employee'
* \_inherits = {'res.partner': 'partner\_id'}
* partner\_id = fields.Many2one('res.partner', required=True, ondelete="cascade")
* job\_position = fields.Char(string="Job Position")

**\_table**

* **Description**: Specifies a custom name for the database table.
* **Example**:
* class CustomModel(models.Model):
* \_name = 'custom.model'
* \_table = 'custom\_table'

**\_log\_access**

* **Description**: Determines if access logs (create\_uid, create\_date, write\_uid, write\_date) are automatically managed.
* **Example**:
* class SimpleModel(models.Model):
* \_name = 'simple.model'
* \_log\_access = False

**\_rec\_name**

* **Description**: Specifies the field used as the display name for records.
* **Example**:
* class CustomModel(models.Model):
* \_name = 'custom.model'
* \_rec\_name = 'custom\_field'
* custom\_field = fields.Char(string="Custom Field")

**\_order**

* **Description**: Defines the default sorting order of records.
* **Example**:
* class Product(models.Model):
* \_name = 'product.template'
* \_order = 'name asc'

**\_constraints**

* **Description**: A list of Python constraints (methods) to validate business rules.
* **Example**:
* class CustomModel(models.Model):
* \_name = 'custom.model'
* name = fields.Char(string="Name")
* value = fields.Integer(string="Value")
* \_constraints = [
* (lambda self: self.value > 0, 'Value must be positive!', ['value'])
* ]

**\_sql\_constraints**

* **Description**: Specifies SQL constraints, such as unique constraints or checks.
* **Example**:
* class Product(models.Model):
* \_name = 'product.template'
* \_sql\_constraints = [
* ('field\_name', 'unique(field\_name)', 'Field must be unique!')
* ]

**\_mail\_thread**

* **Description**: Adds messaging features to the model.
* **Example**:
* class SaleOrder(models.Model):
* \_name = 'sale.order'
* \_inherit = ['mail.thread']