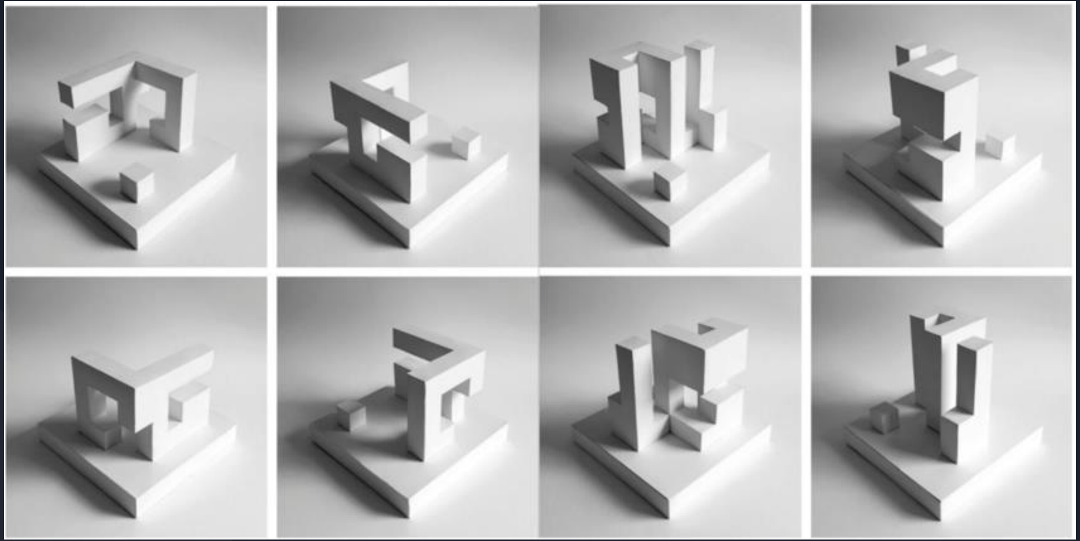
Odoo’s Models introduction



What is Model ?

A **Model** determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. In other words, a model is a table of information that can be bridged with other tables.

Models can be configured by setting attributes in their definition. The most important attribute is ***_name***, which is required and defines the name for the model in the **Odoo** system.

Here is a minimum definition of a model:

```
from odoo import models

class TestModel(models.Model):
    _name = "test.model"
```

Model Types

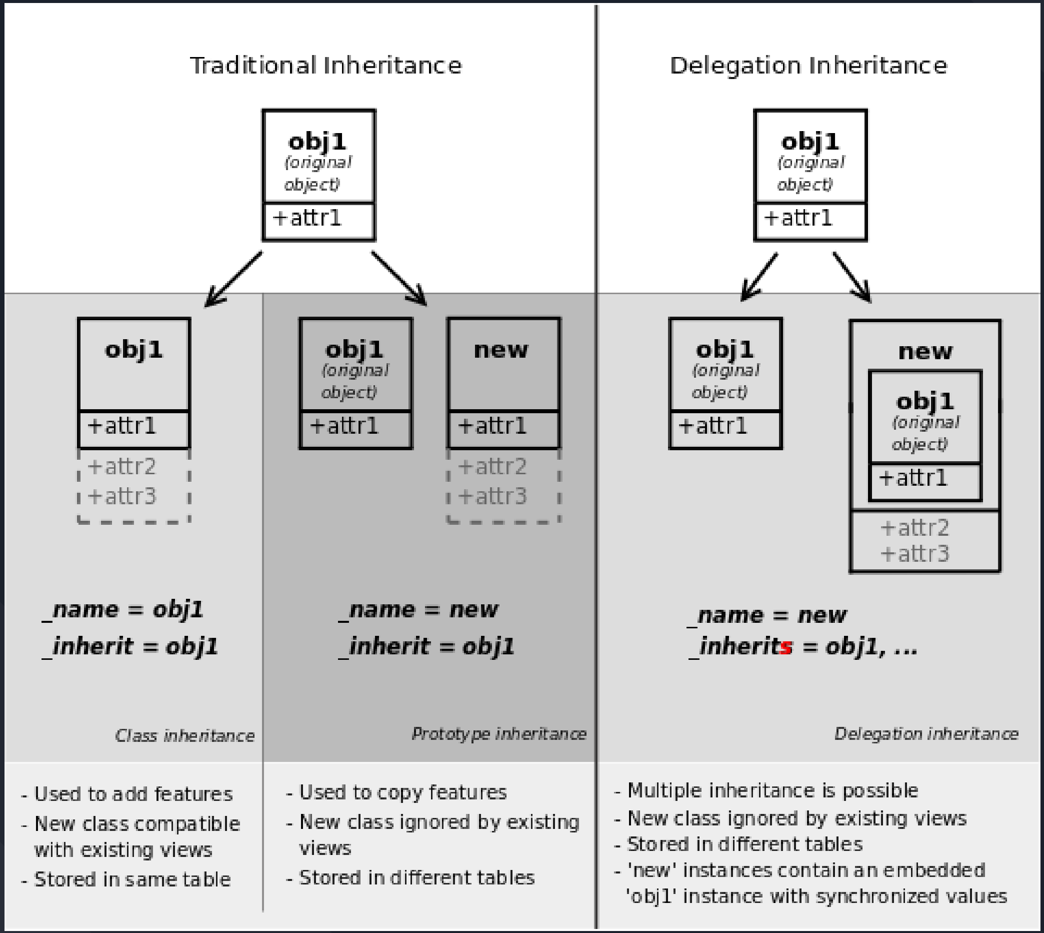
Odoo models are created by inheriting one of the following:

- Model for regular database-persisted models

- TransientModel for temporary data, stored in the database but automatically vacuumed every so often

- AbstractModel for abstract super classes meant to be shared by multiple inheriting models

Models Inheritance



- Used to add features
- New class compatible with existing views
- Stored in same table

- Used to copy features
- New class ignored by existing views
- Stored in different tables

- Multiple inheritance is possible
- New class ignored by existing views
- Stored in different tables
- 'new' instances contain an embedded 'obj1' instance with synchronized values

ModelClass

- The system automatically instantiates every model once per database. Those instances represent the available models on each database, and depend on which modules are installed on that database.

- The actual class of each instance is built from the Python classes that create and inherit from the corresponding model.

- Every model instance is a "recordset", i.e., an ordered collection of records of the model. **Recordsets** are returned by methods like **browse**, **search**, or field accesses. Records have no explicit representation: a record is represented as a recordset of one record.

Models Inheritance

- In the simplest case, the model's registry class inherits from cls and the other classes that define the model in a flat hierarchy.

- The registry contains the instance **model** (on the left). Its class, **ModelClass**, carries inferred metadata that is shared between all the model's instances for this registry only. Example:

```
class A1(Model):
    _name = 'a'

class A2(Model):
    _inherit = 'a'

class A3(Model):
    _inherit = 'a'
```

```
        Model
        / | \
      A3  A2  A1
        \ | /
     ModelClass
        /   \
   model     recordset
```

Models Inheritance

- When a model is extended by **_inherit**, its base classes are modified to include the current class and the other inherited model classes.

- We actually inherit from other **ModelClass**, so that extensions to an inherited model are immediately visible in the current model class, like in the following example:

```
class A1(Model):
    _name = 'a'

class B1(Model):
    _name = 'b'

class B2(Model):
    _name = 'b'
    _inherit = ['a', 'b']

class A2(Model):
    _inherit = 'a'
```

```
    Model
   / / \ \
  / A2 A1 \
 /   \ /   \
B2  ModelA  B1
 \    |    /
  \   |   /
   \  |  /
    ModelB
```

Common ORM

1. search()

Takes a search domain, returns a recordset of matching records. Can return a subset of matching records (offset and limit parameters) and be ordered (order parameter):

```
>>> # searches the current model
>>> self.search([('is_company', '=', True), ('customer',
'=', True)])
res.partner(7, 18, 12, 14, 17, 19, 8, 31, 26, 16, 13, 20,
30, 22, 29, 15, 23, 28, 74)
>>> self.search([('is_company', '=', True)], limit=1).name
'Agrolait'
```

Common ORM

2. create()

Takes a number of field values, and returns a recordset containing the record created:

```
>>> self.create({'name': "New Name"})
res.partner(78)
```

Common ORM

3. write()

Takes a number of field values, writes them to all the records in its recordset. Does not return anything:

```
self.write({'name': "Newer Name"})
```

Common ORM

4. browse()

Takes a database id or a list of ids and returns a recordset, useful when record ids are obtained from outside Odoo (e.g. round-trip through external system) or when calling methods in the old API:

```
>>> self.browse([7, 18, 12])
res.partner(7, 18, 12)
```

Common ORM

5. ref()

Environment method returning the record matching a provided external id:

```
>>> env.ref('base.group_public')
res.groups(2)
```

Common ORM

6. name_get()

Return the text representation of requested objects for x-to-many relationships

```
>>> self.name_get()
[(66, “My name”)]
```