MK.Lab

Часть 4

«Модели данных»

Предисловие

Все дальнейшие уроки мы с Вами будем писать свое приложение и на его примере разберем основные моменты.

В литературе обычно разбирается пример с библиотекой или заказами. Это всё нам надоело. Давайте представим, что к Вам обратился председатель СНТ с просьбой помочь автоматизировать работу вашего дачного кооператива. Основная «боль», по словам председателя, это сбор годовых и целевых взносов, а также ведение протоколов собраний и голосование участников.
Сделаем допущение, что у нас современное СНТ и все наши дачники умеют пользоваться Интернетом и могут войти в Odoo.

Вы поразмыслили, и поняли, что Odoo подходит почти идеально: ведение лицевых счетов — это счета дебиторской задолженности, начисление взносов — это выставление счетов, отправка счетов есть, личный кабинет есть, осталось только «допилить» голосования и задача будет решена.
Но мы вспоминаем, что есть модуль Survey, который и онлайн голосование может сделать.
Значит нам осталось только сделать протоколы собраний.

Вы уже имеете опыт нормализации данных и уже прикинули, что у нас будут:
1. Собрания.
2. Протоколы собраний.
3. Вопросы на повестке.
4. Участники собраний.

Голосование будет поименным только онлайн, оффлайн будет голосование большинством. Поэтому в вопросах следует предусмотреть поля для указания количества людей, голосовавших «За».

MK.Lab

Приступаем к проектированию.

Начинаем делать приложение «Собрания»

Итак, сущности мы определили, давайте по сущностям строить модели.

Собрания:
Дата
Список вопросов
Список участников
Протокол

Протокол:
Дата и время
Собрание, к которому относится этот протокол
Текст протокола
Председатель
Секретарь

Вопросы:
Текст вопроса
Количество голосов ЗА
Итог: решение принято или нет (вычисляемое поле).

Участники собрания:
Пользователь участник

Запускаем Odoo, просим его сделать новый модуль snt-rassvet:

$python3 odoo-bin scaffold snt-rassvet

Переносим сделанный пустой модуль в папку наших самодельных addon-ов.
Давайте отредактируем __maninfest__.py:

__manifest__.py

```python
# -*- coding: utf-8 -*-
{
    'name': "СНТ \"Рассвет\"",

    'summary': "Собрания, протоколы, голосования",

    'description': """
Собрания, протоколы, голосования
    """,

    'author': "My Company",
    'website': "https://www.yourcompany.com",

    # Categories can be used to filter modules in modules listing
    # Check https://github.com/odoo/odoo/blob/15.0/odoo/addons/base/data/ir_module_category_data.xml
    # for the full list
    'category': 'Uncategorized',
    'version': '18.0.0.1',

    # any module necessary for this one to work correctly
    'depends': ['base'],

    # always loaded
    'data': [
        # 'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
    ],
    # only loaded in demonstration mode
    'demo': [
        'demo/demo.xml',
    ],
}
```

На момент написания первой редакции этого курса я использовал среду разработки (IDE) Eric6.
Довольно удобная среда, позволяет настроить подсветку синтаксиса, проверяет синтаксические ошибки в Python, можно настроить темную тему подсветки и пр.

Сейчас я использую PyCharm как более прогрессивную среду.

Итак, манифест готов. Мы в нем сейчас поправили только версию модуля и описание. Для odoo очень важно, чтобы версия модуля соответствовала версии ядра, т.е., если у Вас 18 ядро, модуль должен иметь версию, начинающуюся с 18, например 18.0.1. Кроме того, начиная с 17 версии, Odoo не будет запускаться, если в папках с модулями содержится хотя бы один модуль с некорректной версией. В таком случае в логе можно будет увидеть вот такую ошибку: *ValueError: Invalid version. Modules should have a version in format `x.y`, `x.y.z`, `18.0.x.y` or `18.0.x.y.z`.*

Мы отойдем от канонов комьюнити Odoo, которые нам предлагается соблюдать, а именно:
1. Каждая модель в отдельном файле.
2. Все названия полей на английском.
3. Наличие комментариев на английском.

Хотя требования — правила хорошего тона, мы не будем в нашем примере их соблюдать, и поместим в один файл несколько моделей.

MK.Lab

Переходим в файл models.py

Давайте создадим все классы:

models.py

```python
# -*- coding: utf-8 -*-

from odoo import models, fields, api, exceptions


class Meeting(models.Model):
    _name = 'snt.meeting'
    _description = 'Собрания'

    name = fields.Char(string='Наименование собрания', required=True)
    date = fields.Datetime(string='Дата и время собрания', required=True)
    question_ids = fields.One2many(comodel_name='snt.question',
                                   inverse_name='meeting_id',
                                   string='Вопросы',
                                   required=True)
    member_ids = fields.Many2many(comodel_name='res.users', string='Список участников')
    protocol_id = fields.Many2one(comodel_name='snt.protocol', string='Протокол')

class Protocol(models.Model):
    _name = 'snt.protocol'
    _description = 'Протоколы'

    name = fields.Char(string='Наименование протокола', required=True)
    date = fields.Datetime(string='Дата и время составления')
    meeting_id = fields.Many2one(comodel_name='snt.meeting', string='Собрание')
    text = fields.Html(string='Текст протокола')
    chairman = fields.Many2one(comodel_name='res.users', string='Председатель')
    secretary = fields.Many2one(comodel_name='res.users', string='Секретарь')

class Question(models.Model):
    _name = 'snt.question'
    _description = 'Вопросы'

    name = fields.Char(string='Текст вопроса', required=True)
    yes = fields.Integer(string='Голосовали "ЗА"')
    state = fields.Selection(selection=[('ok','Решение принято'),
                                        ('not_ok','Решение не принято')],
                             default='not_ok',
                             string='Статус решения')
    meeting_id = fields.Many2one(comodel_name='snt.meeting', string='Собрание')
```

Я настоятельно рекомендую сейчас прописать текст руками, так Вы быстрее поймете, что к чему. Это научно доказанный факт. В процессе написания руками Ваш мозг проработает каждый элемент, а моторная кора позволит обработать информацию быстрее и правильнее.
Первый раз будет долго, а потом Вы будете работать в разы быстрее.
Это особенно полезно новичкам в Python, ведь прописывать кортежи как элемент списка так приятно:) Скоро Вы в скобочках будете настоящим мастером!

Именно поэтому я дам Вам листинги в виде скриншотов и пока не дам ничего в виде файлов.

В нашем примере получилось так, что мы использовали много типов данных в полях. Давайте пройдем по важным моментам.

Типы данных в полях

Вся документация по полям находится по ссылке:
https://www.odoo.com/documentation/18.0/developer/reference/backend/orm.html

Нет смысла ее разбирать, остановимся только на важных вещах, которые могут оказаться не очевидными на первый взгляд.

Поля отношений.
Это три типа

- One2many
- Many2one
- Many2many

Конечно же, это отношения один ко многим, много к одному и много ко многим. Эти поля нужны для того, чтобы делать связку между моделями.
Например счет, но у счета есть ссылка на Заказчика. Нужно делать связь.
Тут есть свои нюансы.

Если мы хотим сделать поле, в котором только одно значение, например протокол у собрания, то пишем:

protocol_id = fields.Many2one('snt.protocol', 'Протокол')

Здесь все просто: первый параметр (comodel_name) — имя привязываемой модели,
Второй параметр (string) необязательный – наименование поля.

Если мы хотим сделать строки документа, то пишем:

```
quiestion_ids = fields.One2many('snt.question', 'meeting_id',
                                'Вопросы',
                                required=True)
```

Обратите внимание, тут уже два параметра связности. Первый (comodel_name) — имя модели, а второй (inverse_name) — столбец в той, другой модели, который указывает на идентификатор нашей модели. Это обязательное условие — сделать такой столбец.

Например в sale.order имеем:
line_ids — поле с идентификаторами строк
А в строках в модели sale.order.line есть поле order_id.

Если хотим просто привязать много чего-то, что не является только лишь строками нашего документа, например, пользователи, или же теги-классификаторы, то мы пишем:

member_ids = fields.Many2many('res.users', 'Список участников')

Особенно следует отметить тип данных Datetime.
По историческим причинам, как пишут в книгах, это поле хранится в строковом формате.
Для того, чтобы применить всю мощь модуля Datetime, Вам нужно конвертировать данные из этого типа в Datetime

К счастью, есть встроенные методы:
date_datetime=fields.Datetime.from_string(<Значение поля типа Datetime в Odoo>)
И обратно:
date_string=fields.Datetime.to_string(<значение типа Datetime в Python>)

Важная ремарка: значение в базе хранится в UTC, поэтому при конвертации Вы получите данные без учета часового пояса.
Я рекомендую всегда очищать данные от влияния часового пояса перед операциями сравнения.

Остальные типы данных не вызовут никаких затруднений, это обычные типы, кроме Float.
Этот тип может быть использован **для хранения времени**.
Вы, быть может, видели поля, в которых можно вводить только час и минута. Это на самом деле поле float. Есть виджет time, он автоматически переводит поле float в удобный для пользовательского ввода.

Это поле хранит в часах смещение от начала суток.
Например, 12.30 будет равно 12.5, 7.45 будет равно 7.75.
А если проще, то timedelta(hours=<float значению>).

Готовим формы

Модели мы создали, теперь нужно сделать формы и списки, чтобы odoo смогла красиво отобразить наш модуль и создать для него графический интерфейс.

Но сначала давайте посмотрим, что будет, если мы поленимся и не будем делать формы.

Давайте сделаем пункт меню, в него добавим одно единственное действие — отображение списка нашей модели snt.meeting.

Идем в файл

views/views.xml

И вносим вот такие данные:

MK.Lab

views.xml

```xml
<odoo>
  <data>

    <record model="ir.actions.act_window" id="snt-rassvet.action_window">
      <field name="name">Собрания</field>
      <field name="res_model">snt.meeting</field>
      <field name="view_mode">list,form</field>
    </record>


    <menuitem name="СНТ" id="snt-rassvet.menu_root"/>
    <menuitem name="Собрания"
              id="snt-rassvet.menu_1"
              parent="snt-rassvet.menu_root"
              action="snt-rassvet.action_window"
    />

  </data>
</odoo>

```

Давайте разберем.

Сначала мы создали действие, связанное с нашей моделью snt.meeting, которое говорит системе, что для этой модели есть представления форма(form) и список(list, до 18 версии tree).

Далее мы создали в главном меню пункт, потом добавили подпункт, который вызывает наше действие. А именно отображение списка собраний.

Важное правило. Идентификаторы желательно, а порой и обязательно начинать с названия модуля. Обратите внимание, как это сделано у меня.

Настройки доступа

Казалось бы, модель прописана, формы нарисованы, устанавливай и пользуйся! Однако, если установить модуль сейчас, мы не сможем даже найти его в меню. Почему так? Потому что у нас нет доступа к свежесозданным моделям. Мы их просто не видим.

Файл с параметрами доступа лежит в папке security и называется ir.model.access.csv. Откройте файл и посмотрите, что находится внутри.

Файл представляет собой таблицу в формате csv, в которой указывается какие права, какой группе пользователей и для какой модели присваиваются.

Таблица состоит из следующих столбцов:

Id – уникальный идентификатор разрешения.

name – уникальное имя разрешения.

model_id:id – уникальное имя модели класса, к которому мы хотим применить разрешение.

group_id:id – идентификатор группы пользователь, для которой применяются разрешения (можно создать свою группу или указать существующую, например base.group_user – это все внутренние пользователи)

perm_read, perm_write, perm_create, perm_unlink – 4 значения для относительного разрешения на чтение, запись, создание, удаление записи в определенном классе. 1 – это True (вы можете выполнить это действие), 0 – False (вы не можете).

Права доступа являются аддитивными, доступ пользователя представляет собой объединение доступов, которые он получает через все свои группы, например, если пользователь, который является частью группы A, предоставляет доступ для чтения и создания, а группа B предоставляет доступ для обновления, пользователь будет иметь все три из создавать, читать и обновлять.

Укажем права доступа для нашего модуля, как показано на скриншоте ниже.

ir.model.access.csv

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_snt_meeting,snt.meeting,model_snt_meeting,base.group_user,1,1,1,1
access_snt_protocol,snt.protocol,model_snt_protocol,base.group_user,1,1,1,1
access_snt_question,snt.question,model_snt_question,base.group_user,1,1,1,1
```

При создании структуры модуля с помощью scaffold, файл security/ir.model.access.csv оказывается заранее прописанным в манифесте, однако он закомментирован. Найдите файл внутри раздела data и раскомментируйте.

Всё, модуль можно пробовать в работе.

Установите модуль и посмотрите, как он работает. Напомню, что формы odoo сделает сам из всех полей, которые есть.

MK.Lab

Доработка форм

Посмотрим, какие формы создал сам odoo для нашего модуля.

В целом, работать можно, но хочется, чтобы использовать модель было еще и удобно.

Начнем со списка наших собраний. Сейчас там только наименование собрания, добавим к этому дату и время проведения собрания.

В файл views/views.xml добавим следующее представление:

```
<record model="ir.ui.view" id="snt-rassvet.meeting_list">
  <field name="name">snt-rassvet meeting_list</field>
  <field name="model">snt.meeting</field>
  <field name="arch" type="xml">
    <list>
      <field name="name"/>
      <field name="date"/>
    </list>
  </field>
</record>
```

model="ir.ui.view" – показывает, что создается представление для какой-либо модели

id="snt-rassvet.meeting_list" – идентификатор, по которому можно обратиться к конкретному представлению

name – наименование представления

model – указание модели, к которому относится наше представление

arch – описание макета представления

тег **list** (до 18 версии тег **tree**) показывает, что представление будет иметь структуру списка, внутри него указываются поля (field) для отображения в указанном порядке.

Обновите модуль и посмотрите, что получилось.

Теперь займемся непосредственно формой собрания.

```xml
<record model="ir.ui.view" id="snt-rassvet.meeting_form">
  <field name="name">snt-rassvet meeting_form</field>
  <field name="model">snt.meeting</field>
  <field name="arch" type="xml">
    <form>
      <sheet string="Собрание">
        <h1>
          <field name="name" nolabel="True"/>
        </h1>
        <h2>
          <field name="date" nolabel="True"/>
        </h2>
        <group>
          <field name="protocol_id"/>
        </group>
        <notebook>
          <page name="questions" string="Вопросы">
            <field name="question_ids">
              <list editable="bottom">
                <field name="name"/>
                <field name="yes"/>
                <field name="state"/>
                <field name="meeting_id"/>
              </list>
            </field>
          </page>
          <page name="members" string="Участники">
            <field name="member_ids">
              <list editable="bottom">
                <field name="name"/>
              </list>
            </field>
          </page>
        </notebook>
      </sheet>
    </form>
  </field>
</record>
```

Основным отличием от предыдущего представления является тег **form**, вместо тега **list**.

В данном примере теги **h1** и **h2** пришли из html разметки и означают, что название собрания и время его проведения будут напечатаны как заголовки, а атрибут **no_label** выведет поля без подписи к ним

Теги **notebook** и **page** позволяют создавать страницы с вкладками.

Тег **list** (до 18 версии **tree**), указанный внутри полей типа One2many или Many2many позволяет отображать поля связанной с ним модели.

Параметр **editable** указывает на режим редактирования полей в списке.

Обновите модуль и посмотрите, как изменилось отображение модели.

В следующем уроке мы добавим логику.

Задание:

Наберите руками этот модуль, добейтесь чтобы он работал. Пришлите тренеру скриншоты.