L'ORM d'OpenERP

Toulibre

Mercredi 15 décembre 2010

Préambule

- Je ne suis pas spécialiste des ORMs
- Je ne suis qu'un utilisateur de l'ORM d'OpenERP

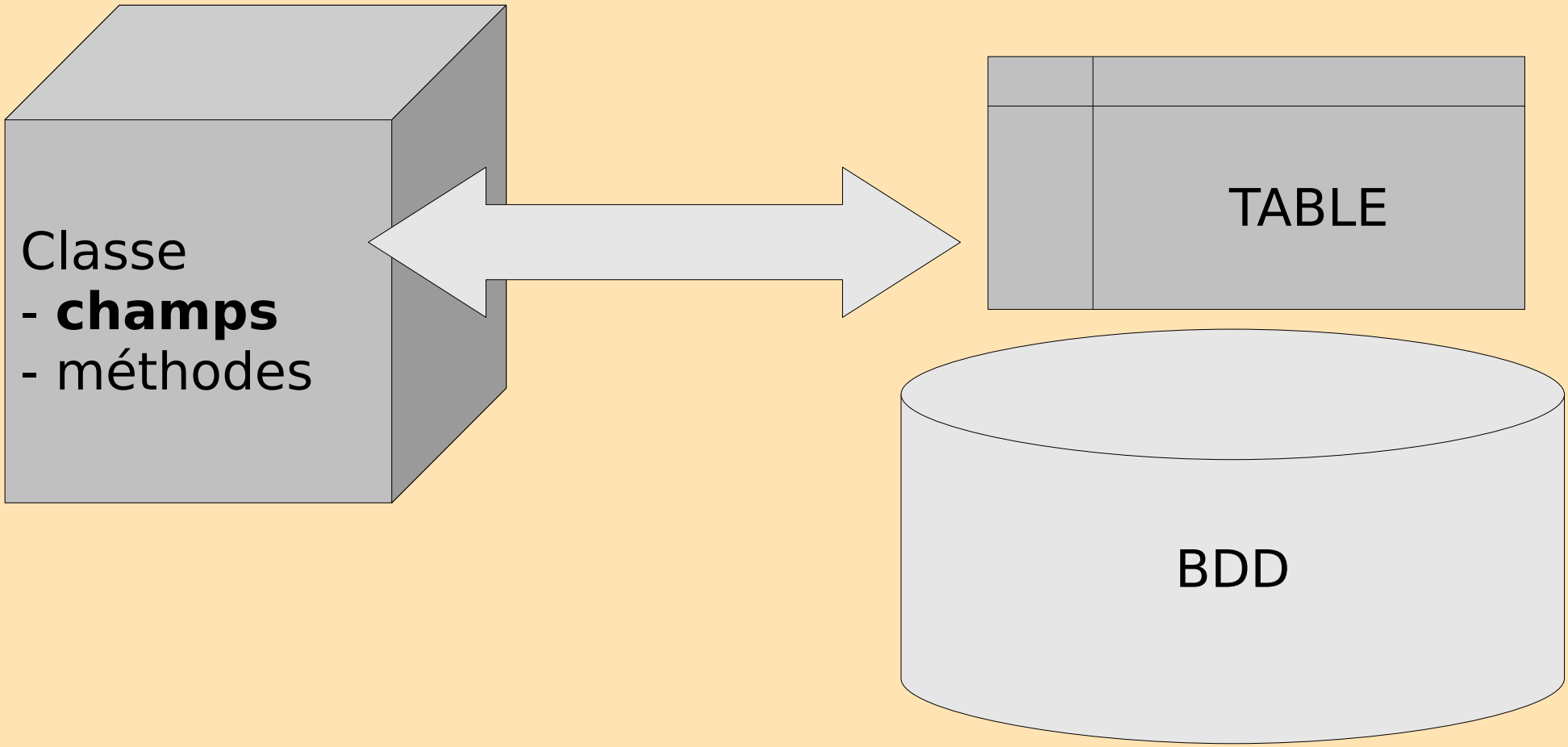
Introduction

- Logiciel généraliste pour l'ensemble de l'activité des entreprises
  - Devis
  - Commandes
  - Production
  - Facturation
  - Compta
- OpenERP = OpenObject + modules prédéfinis

OpenObject

- Plate-forme de dev client/serveur
- La gestion du stockage est faite par l'ORM

Principe d'un ORM



Les subtilités de l'ORM d'OERP

- Ce ne sont pas les champs python standards qui sont pris en compte !
- Les champs de la bdd sont définis dans un dictionnaire, qui est une variable de classe

```
class ma_classe (osv.osv)
    _name = 'ma_classe'
    _columns = {
          'nom' : fields.char('Nom'),
          'date_depart' : fields.date('Date de depart'),
    }
```

- cela permet de bien séparer les champs de la bdd avec les champs python
- il existe les types standards (booléen, char, date, entier, numérique, sélection)
- il existe aussi des types relationnels :
  - many2one
  - one2many
  - many2many
- ils sont très puissants !

exemple : partenaire/adresses

```
class res_partner(osv.osv):
    _name = "res.partner"
    _columns = {
        'name': fields.char('Name', size=128, required=True),
        'address': fields.one2many('res.partner.address', 'partner_id', 'Contacts'),
        'category_id': fields.many2many('res.partner.category', 'res_partner_category_rel',
'partner_id', 'category_id', 'Categories'),
    }


class res_partner_address(osv.osv):
    _name = 'res.partner.address'
    _columns = {
        'partner_id': fields.many2one('res.partner', 'Partner'),
        'name': fields.char('Contact Name', size=64),
        'street': fields.char('Street', size=128),
        'street2': fields.char('Street2', size=128),
        'zip': fields.char('Zip', size=24),
        'city': fields.char('City', size=128),
    }
```

Les APIs

- La grande subtilité est la différence entre l'objet et l'enregistrement !

```
#
#on demande une instance qui gère la table 'ma.class'
maclass_obj = pool.get('ma.class')
# on effectue une recherche sur cette table
# 'cr' est un curseur de la bdd
# 'uid' est le user id
ids = maclass_obj.search(cr, uid, [('nom', '=', 'test')] )

for id in ids:
    # on boucle sur les enregistrements
    maclass_rec = maclass_obj.browse(cr, uid, id)
    # on a un objet python qui represente un enregistrement
    # on peut acceder aux champs :
    print maclass_rec.nom
    print maclass_rec.date
    # et pour les champs relationnels :
    maclass_rec.autre_table.autre_champ
```

autre exemple

```
#on demande une instance qui gère la table 'ma.class'
maclass_obj = pool.get('ma.class')
# on effectue une recherche sur cette table
ids = maclass_obj.search(cr, uid, [('nom', '=', 'test')] )

res = maclass_obj.read(cr, uid, ['nom', 'date'])
# res est une liste de dictionnaires : on retrouve une structure classique
# de table (lignes, colonnes)
for d in res:
    # on boucle sur les resultats
    print d['nom']
    print d['date']
```

- browse()
  - objet, très puissant pour les champs relationnels
  - peut être lourd en mémoire (!)
- read()
  - structure classique, optimisation des données chargées

Méthodes de l'ORM

- browse()
- create(), unlink()
- read(), write()
- search()
- copy()
- export_data(), import_data()

http://doc.openerp.com/developer/2_5_Objects_Fields_Methods/methods.html

pour aller plus loin...

- l'ORM gère aussi la notion d'héritage...
  - on peut augmenter les fonctionnalités d'un objet existant (héritage de classe)
    - ajout de champs dans bdd
    - ajout de méthodes
  - on peut créer des nouvelles classes par prototypage...

http://doc.openerp.com/developer/2_5_Objects_Fields_Methods/object_inherit.html

Documentation

- http://doc.openerp.com/developer/

Merci !