DIVE INTO GENROPY

ORM GENROPY

CARATTERISTICHE PRINCIPALI

- Descrizione del modello in Python (Structures!)
- Creazione/Allineamento automatico del db dal modello
- Supporta differenti DBMS in modo trasparente
- Query con *path_relazionale*
- *aliasColumn, formulaColumn, pyColumn*
- Supporto multidatabase
- Trigger a livello Python
- Tabelle customizzabili

DEFINIZIONE TABLE – PROVINCIA

```
class Table(object):
    def config_db(self, pkg):
        #definizione della table
        tbl = pkg.table('provincia', pkey='sigla',caption_field='sigla')

        tbl.column('sigla' ,size='2',name_long='Sigla')
        tbl.column('nome', size= ':128',name_long = 'Descrizione')
        tbl.column('abitanti', dtype='L', name_long= 'Abitanti')
        tbl.column('regione', size='3').relation('glbl.regione.sigla',
                                                 relation_name='province')
```

- pkey indica la *primary key* della table
- caption_field è la colonna identificativa del record a livello utente
- dtype indica il tipo della colonna, se non specificato è **text**
- size se presente il tipo della colonna è **char** (valore fisso) oppure **varchar** (range)

DEFINIZIONE TABLE - CLIENTE

```
class Table(object):
    def config_db(self, pkg):
        tbl = pkg.table('cliente', pkey='id',caption_field='ragione_sociale')
        tbl.column('id' ,size='22', name_long='Id')
        tbl.column('ragione_sociale', size=':40', name_long='Ragione sociale')
        tbl.column('indirizzo', name_long='Indirizzo')
        prov_col = tbl.column('provincia',size='2',name_long='Provincia')
        prov_col.relation('glbl.provincia.sigla',
                          relation_name='clienti',
                          mode='foreignkey')
```

- Introduciamo l'elemento relation che si attacca alla colonna
- Nel primo parametro indichiamo la colonna collegata: **package.table.column**
- relation_name è il nome con il quale lo sviluppatore si può riferire alla relazione one-many, indicata dal punto di vista della table **one**
- mode se foreignkey significa che viene definita anche a livello di **db** come chiave esterna

QUERY

```
def myQuery(self):
    tbl = self.db.table('fatt.cliente')
    query = tbl.query(columns='$ragione_sociale, $indirizzo, @provincia.nome',
              where='@provincia.abitanti > :n_abitanti', n_abitanti = 500000)
    f = query.fetch()
```

- Partiamo sempre dal punto di vista di una table che otteniamo come oggetto usando il metodo table
- Usiamo il metodo query
- Ci riferiamo alle colonne della table con il prefisso $
- Ci riferiamo alle colonne di altre tabelle in relazione con il prefisso @
- Questo ci introduce al concetto di path relazionale come alternativa alle **JOIN**
- Dall'elemento query usiamo fetch che ci restituisce i record in un iteratore

PATH RELAZIONALE

- Ci permette di percorrere le relazioni tra le table come se stessimo attraversando una Bag
- Dalla table fattura voglio la regione del cliente:
  - rpath='@cliente_id.@provincia.@regione.nome'
  - Attraversando quindi le tabelle:
  - fattura->cliente->provincia->regione
- L'ORM Genropy traduce questo nelle opportune JOIN da usare nella query **SQL**

IL METODO RECORD – IL RITORNO DEL RESOLVER

```
def recordExample(self):
    tbl = self.db.table('glbl.regione')
    cliente = tbl.record('mf_6EJbWN1unRWkVjRRTgA', mode='bag')
    print cliente['@provincia.@regione.nome']
```

- Nel primo parametro passo l'identificativo del record (primary key)
- Il metodo record usato con mode='bag' mi restituisce il record come bag
- Questo significa che posso accedere alle colonne con []
- E usando il path_relazionale navigare lungo le relazioni
- Questo è reso possibile dai resolver
- Ogni volta che il path percorre un @ un resolver scatta, esegue una query sul DB e mi restituisce il record collegato

IL METODO RECORD – SECONDA PARTE

```
def myResolverMadness(self):
    tblobj = self.db.table('glbl.regione')
    regione = tblobj.record('LOM', mode='bag')
    for provincia in regione['@province'].values():
        print (provincia['nome'])
        for cliente in provincia['@clienti'].values():
            print (cliente['ragione_sociale'])
```

- Qui possiamo osservare che il path relazionale può percorrere la relazione anche in senso inverso, cioè da **one** a **many**
- E per fare questo usiamo l'attributo relation_name (ve lo ricordate?)
- In questo caso il resolver che scatta ogni volta che leggiamo @ va a fare una query che restituisce
  - Le province della regione
  - I clienti della provincia

SCRITTURE

- L'oggetto table offre ovviamente metodi per scrivere
- insert(record)
- update(record, old_record)
- delete(record)
- Come parametro record può accettare dizionari o Bag

TRIGGER PYTHONICI

- L'oggetto table ha anche dei metodi *hook* detti trigger
- Questi vengono innescati dalle scritture sul database
  - trigger_onInserting / trigger_onInserted
  - trigger_onUpdating / trigger_onUpdating
  - trigger_onDeleting / trigger_onDeleted
- Ricevono sempre il record inserito/modificato/cancellato
- Vengono eseguiti prima che avvenga il *commit*

COLONNE VIRTUALI

- Nella definizione della table possiamo definire colonne virtuale, che non diventeranno cioè vere colonne del DB
- Possono però essere lette in Genropy come tutte le altre
  - tbl.aliasColumn
  - tbl.formulaColumn
  - tbl.pyColumn

ALIASCOLUMN

```
tbl.aliasColumn('regione', '@provincia.@regione.nome', name_long='Regione')
```

- Si riferisce ad una colonna di una table, raggiungibile tramite relazioni con un path relazionale
- Rende trasparente all'esterno l'uso della relazione

FORMULACOLUMN

```
tbl.formulaColumn('indirizzo_provincia',
                  '$indirizzo||" - "||$provincia',
                  name_long='Indirizzo provincia')
```

```
tbl.formulaColumn('tot_fatturato',
                  select=dict(table='fatt.fattura',
                              columns='SUM($totale_fattura)',
                              where='$cliente_id=#THIS.id'),
                  dtype='N', name_long='Tot.Fatturato')
```

- Ritorna il risultato di un'operazione di SQL passata direttamente
- Oppure il risultato di una subquery definita nel parametro select

PYCOLUMN

```
tbl.pyColumn('life_universe_everything', dtype='L', py_method='deepThought')

def deepThough(self, record, field):
    return 42
```

- E’ analoga alla formulaColumn, ma restituisce il risultato calcolato da un metodo Python