Cours de bases de données, http://sql.bdpedia.fr

Environnements objets et bases relationnelles

Philippe Rigaux

le cnam

Le problème

Représentation relationnelle des entités : nuplets (entités), et mécanisme clés primaire / clé étrangère (liens entre entités).

Les applications objet (Java, C#, Python, PHP, ...) représentent les entités comme des objets directement liés les uns aux autres.

Problème : comment convertir d'une représentation à l'autre ?

Remarque: Bases objets

Une idée serait d'utiliser des bases objets et pas relationnelles. Elle n'a pas vraiment abouti.

Premier exemple de conversion

Pour lire les contacts comme des objets d'une classe `Contact`.

```
class Contact:
     def __init__(self,id,prenom,nom,email):
                self.id=id
                self.prenom=prenom
                self.nom=nom
                self.email=email

curseur.execute("select * from Contact")
for cdict in curseur.fetchall():
    # Conversion du dictionnaire en objet
    cobj = Contact(cdict["idContact"], cdict["prénom"],
                cdict["nom"], cdict["email"])
    print(cobj.prenom, cobj.nom)
```

Implique des instructions lourdes, répétitives, sujettes à erreur.

Plus compliqué : la classe Message

Composition d'objets, associations, méthodes.

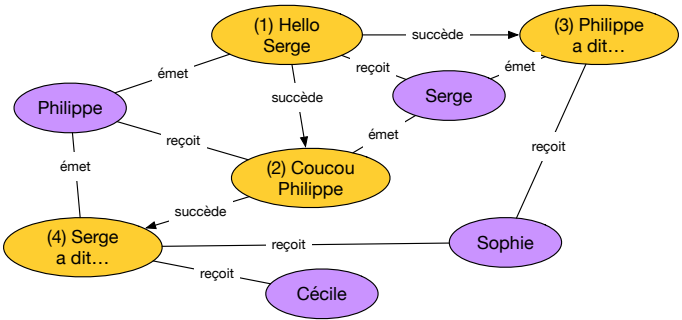
```
class Message:
    # Emetteur: un objet 'Contact'
    emetteur = Contact(0, "", "", "")
    # Prédecesseur: peut ne pas exister
    predecesseur = None
    # Liste des destinataires = des objets 'Contacts'
    destinataires = []

    # La méthode d'envoi de message
    def envoi(self):
        for dest in self.destinaires:
            ...
```

Conversion depuis la base ? Très pénible....

Comment une application objet “voit” les données

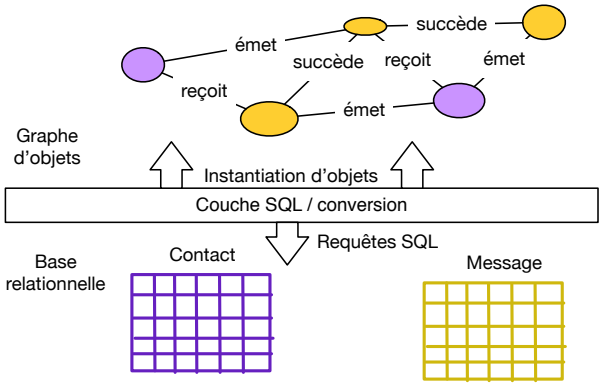
Comme un graphe !



Trop différent de la représentation relationnelle : il nous faut des outils.

Le *mapping* objet-relationnel (ORM)

Un système ORM convertit automatiquement, à la demande, la base relationnelle sous forme d'un graphe d'objet



Exemple avec le *framework* Hibernate (Java)

```
@Entity(table="Message")
public class Message {
  @Id
  private Integer id;

  @Column
  private String contenu;

  @ManyToOne
  private Contact emetteur;

  OneToMany
  private Set<Contact> destinataires ;
}
```

On indique au système ORM tout ce qui est nécessaire pour associer objets et nuplets de la base.

Parcours d'un graphe d'objets

Exemple (un peu simplifié) d'envoi des messages de Serge.

```
List<Message> resultat =
     session.execute("from Message as m "
                     + " where m.emetteur.prenom = 'Serge'");

for (Message m : resultat) {
        for (Contact c : m.destinataires) {
           mail (c.email, m.contenu);
        }
}
```

Le système engendre et exécute les requêtes SQL.

Tendance à produire beaucoup de requêtes élémentaires là où une seule jointure serait plus efficace.

À retenir

Les ORMs sont maintenant très utilisés pour les développements d'envergure

- Les accès à la base prennent la forme d'une navigation dans un graphe d'objets
- Le système engendre les requêtes SQL pour matérialiser le graphe
- Conversion entièrement automatique
- Gains de productivité, mais utilisation sous-optimale de SQL

Pour aller plus loin : la mini-application Django sur `http://sql.bdpedia.fr` ou le cours complet `http://orm.bdpedia.fr`