# Le langage OQL





## Les objets persistants

- Persistance par référence :
  - un objet persiste s'il est relié à une racine de persistance
  - sinon c'est un objet temporaire
- Racines de persistance :
  - objets déclarés comme persistant dans le schéma,
  - par l'intermédiaire de la commande *name*
  - objets interrogeables par des requêtes,
  - à partir de ces objets, on peut atteindre toutes les données persistantes



## Schéma agence_s
## Définition des classes



```
class Pays inherit Object public type
  tuple(  nom: string,
          capitale: Ville,
          population: integer,
          villes: set(Ville))
end;

class Ville inherit Object public type
  tuple(  nom: string,
          population: integer,
          pays: Pays,
          lieux_a_visiter: set(Place),
          hotels: set(Hotel),
          tours: set(Tour))
end;

class Hotel inherit Object public type
  tuple(  nom: string,
          etoiles: integer,
          petit_dejeuner: integer,
          demi_pension: integer,
          pension_complete: integer,
          adresse: string,
          ville: Ville)
end;

class Tour inherit Object public type
  tuple(  nom: string,
          places: list(Place))
  method public tarif :integer
/* retourne le tarif cumulé des places*/
end;
```



## Schéma agence_s (2)



```
class Place inherit Object private
  type
  tuple(  public  nom: string,
          read    description: string,
          read    photo: Bitmap,
          public  adresse:
                  tuple( rue: string,
                         ville: Ville),
          public  tarif: integer)
end;

class Monument inherit Place public type
  tuple(siecle: integer,
        jours_fermeture : list(string))
end;

class Musee inherit Place public type
  tuple(  specialite: string,
          jours_visite: list(string))
end;
```

### Racines de persistance

```
name La_France :Pays;
name Les_Monuments :set(Monument);
name Les_Musees :set( Musee);
name Les_Pays :list(Pays);
name Les_Places :set(Place);
name Les_Villes :list(Ville);
name Monuments_Paris :set(Monument);
name Paris :Ville;
name Rome :Ville;
name Tour_Eiffel :Monument;
```





## Requêtes élémentaires

- Nom d'objet : renvoie l'objet associé
  - La_France ⇒ résultat de type Pays
- Requêtes sur les atomes
  - atome : renvoie cet atome
    - 2 ⇒ 2
  - opérations arithmétiques : **+,-,*,/,mod, abs**
    - 2 + 2 ⇒ 4
  - opérations sur les chaînes : +,x[i], x[i:j], **count**(x)

  **!!** la numérotation commence à 0
  - opérations booléennes : x **and** y, x **or** y, **not**(x)





## Requêtes élémentaires (2)

- Projection : accès à un attribut d'un n-uplet
  - attribut atomique : nom.attribut
    - La_France.nom ⇒ "France" (type **string**)
  - attribut non-atomique : possibilité de reprojeter
    - Tour_Eiffel.adresse ⇒ type **tuple(**rue:...**)**
    - Tour_Eiffel.adresse.ville ⇒ type Ville
    - Tour_Eiffel.adresse.ville.nom ⇒ "Paris"
    - La_France.villes ⇒ ensemble de villes

!!!! La_France.villes.nom est incorrect !!!

car nom n'est pas un attribut de la classe ensemble

# Requêtes élémentaires (3)

- application d'une méthode :
  - nom.méthode[(paramètres)]
- construire un n-uplet
  - struct() renvoie n-uplet vide
  - struct($a_1$:$q_1$, ..,$a_n$:$q_n$) $a_i$ =nom $q_i$=requête
  - ↳ tuple
  - ↳ attributs de noms $a_i$,
  - ↳ valeurs : résultat de $q_i$

ex : **struct**(nom: Paris.nom, pays:Paris.pays.nom)
=> tuple(nom:Paris, pays:France)





# Requêtes sur des collections

- les collections doivent être homogènes!
- opérations
  - **element**(C) : renvoie l'unique élément de C
  - **count, sum, max, avg, in, distinct**
  - + (concaténation)
  - **flatten** : élimination d'un niveau de décomposition

ex: flatten(list(list(1,2),list(3,4,5),list(6))) =>list(1,2,3,4,5,6)
(n'enlève qu'un seul niveau)





# Opérations sur les listes

soit a une liste

- ième élément : a**[i]**
  - Les_Villes[2] : 3ième ville
- sous-liste : a**[i:j]**
  - Les_Villes[0:10] : les 11 premières villes
- **first**(a), **last**(a)
- construction : **list**(x1,x2, ... xn)
- transformation en ensemble : **listtoset**(a)





# Opérations sur les ensembles et les tas

Soit a et b deux ensembles

- différence : a **except** b
- union : a **union** b
- intersection : a **intersect** b
- inclusion : <,<=,>,>=
- construction :
  - ensemble/tas vide : **set()** ou **bag()**
  - **set**(x1, x2, ..., xn) ou **bag**(x1, x2, …, xn)





# Prédicats et quantificateurs

- comparaison : **=, !=, <, >, <=, >=, in,like**
  ex : nom **like** "Du*"
- quantificateurs
  **exists** x **in** y **:** p(x)
  **forall** x **in** y **:** p(x)
  - x nom de variable,
  - y collection,
  - p prédicat = requête faisant référence à x





# Select-from-where

**Select** [**distinct**] R
**from** $v_1$ **in** $c_1$, $v_2$ **in** $c_2$, ... $v_n$ **in** $c_n$
[**where** e]

- $c_1$, $c_2$, ... expressions de type collection
- $v_1$, $v_2$, ... noms de variables
- e expression booléenne

↳ produit carthésien de tous les $c_i$, filtré par e
collection de n-uplets ($v_1$: type des él de $c_1$, $v_2$: ..., $v_n$: .. )
↳ requête R ensuite exécutée sur chaque n-uplet

# Partition

**select** R
**from** $v_1$ **in** $c_1$**,** $v_2$ **in** $c_2$**, ...,** $v_n$ **in** $c_n$
**where** e
**group by** $att_1$**:**$e_1$**,** $att_2$**:**$e_2$ **...** $att_m$**:**$e_m$
$att_i$ : noms, $e_i$: expressions
groupe les éléments de la collection select-from-where
selon même valeur de ($e_1$,... $e_m$)
- résultat : ensemble de m+1uplets
  ($att_1$:$val_1$, ... $att_m$:$val_m$, **partition**:collection de n-uplets)
- la partition est de la forme (v1:..., v2 : ..., ..., vn:...)
- la requête R est ensuite exécutée sur chaque m+1 uplet





# Partition : exemple

**select struct(**nompays : v.pays.nom, listeville :**partition**)
**from** v **in** Les_Villes
**group by** v.pays.nom
groupe les éléments de la collection Les_Villes
selon le nom du pays
- résultat : ensemble couples
  (nompays : ..., listeville :set(tuple(v:..),tuple(v:...),tuple(v:...)))

- **having** e'
  - résultat filtré selon e'





# Tri

**select-from-[where]-[group-by-[having]]**
**order by** e1 [**desc**], e2 [**desc**], ... en [**desc**]
ex :
**select** p **from** p **in** Les_Places
**order by** p.tarif **desc**

- collection obtenue : **liste**