SQL3

Schéma et modélisation

- **/!\ Ordre important, faire des headers (déclaration vide) en cas de besoin.** `create type T_Piece;`
- `;` à la fin
- `create type X as object ()`
- `varchar(20)`
- `number(5)`
- `Propriété ref Type` quand on veut une ref ou non
- - `not instantiable member function masse return number` pour dire qu'on définie que la signature

Nested table

```
create type Consommateur as object(
    nom varchar(20),
    age number(2),
    achete ref EnsBierreMag,
    consomme EnsBierreBar
);

create type MagBiere as object(
    quoi ref Bierre,
    lieu ref Magasin,
);
create type EnsMagBierre as table of MagBierre;

create type BarBiere as object(
    quoi ref Bierre,
    lieu ref Bar,
);
create type EnsBarBierre as table of BarBierre;
```

- Lorsque qu'on doit faire une table dans une table on utilise la syntaxe précédente : on créer un type `as table of Type` ou `as table of ref Type`
- On peut ensuite utiliser `ref` si on pense réutiliser plusieurs fois
- **/!\ lors de l'instantiation** `... nested table X store as t_i`

Héritage

```
create type Etablissement as object (
    num Varchar2(20),
    horaire varchar2(20),
    localisation ref Ville,
) not final; -- Indique que l'objet à des descendants (héritage)

create type magasin under Etablissement(
    surface number(5)
);
```

- `not final` si l'objet a des descendants (héritage)
- `create type Child under Parent` Pour faire héritage
- `overriding member function volume return number` pour override les méthodes lors d'un héritage

Instanciation

```
create table LesBars of Bar;
create table LesBierre of Biere;
create table LesConsomateur of Consommateur
    nested table achete store as t1,
    nested table consomme store as t2;
```

Ou en faisant attention aux noms **uniques**

```
create table lesUnité of Unité
    nested table contenu store as t1 (
        nested table presents store as t3
    )
    nested table noteInscrits store as t4
```

Requête

Diapo 19 :

- `ref(*)` : Utilisé dans les insertions, lorsque qu'on veux insérer une ref
- `deref(*)` : Utilisé dans les requêtes, pour deref l'objet pour l'afficher. Lorsque qu'on veut utiliser l'objet, on peut appliquer le `.` directement, ça deref automatiquement l'objet. Attention PL/SQL force sont utilisation (voir diapo 26)
- `table(*)` : Utilisé dans le `from` des requêtes lorsque doit accéder à un type ensembliste
- `value(*)` : Utilisé dans les requêtes lorsqu'on doit accéder à une valeur d'un type ensembliste (ref ou non). Car dans un type ensembliste, `table(e.ens_type)` est un nuplet du genre `(id, Object)` donc faut au moins le décomposer avec `value()`.

Insertion

Deux syntaxes en fonction de si on insère dans un type ensembliste ou non.

- `insert into LaTable Value Type()`
- `insert into table (LaTable) values (Valeur)` On peut mettre des instances vides :

```
insert into lesUnité VALUES (
    Unité('MLBDA',
        '4I801',
        6,
        Séances(
            Séance(1, 'SQL', ?), -- On met une instance vide d'étudiant
            Séance(2, 'SQL3', Etudiant()),
            Séance(3, 'XML', Etudiant())
        ),
        EnsC2() -- Pareil ici on met une instance vide
);
```

On peut mettre des requette pour query les ref()

```
INSERT INTO LesEtudiants VALUES Etudiant(
    1,
    2022
    Alice
    28
    Unités(
        SELECT ref(u) from LesUnités WHERE u.code = '4I808',
        SELECT ref(u) from LesUnités WHERE u.code = '4I101',
    )
);
```

On peut mettre des `null` aussi

```
INSERT INTO Les_Pieces_Composites VALUES (T_Piece_composite(
    'billard', -- nom
    null,  -- entre dans
    10, -- cout
    T_contient_plusieurs(
        T_contient((SELECT ref(p) FROM Les_Pieces_Bases p WHERE p.nom = 'boule'), 3),
        T_contient((SELECT ref(p) FROM Les_Pieces_Bases p WHERE p.nom = 'canne'), 2)
    )
));
```

Méthode

```
member function notes return EnsC AS
    res EnsC1;
    BEGIN
        SELECT C1(value(u).code, n.note) BULK COLLECT INTO res
```

```
        FROM table(self.contrat) u, table(value(u).notesInscrits) n
        WHERE n.etu = ref(self);
    RETURN res;
END;
```

- ``BULD COLLECT INTOpour affecter à une variable l'ensemble des objets retournés par une requête SQL. Entre leSELECTet leFROM`.
- `overriding member function volume return number` lors de la déclaration de type en cas d'héritage !
- `not instantiable member function masse return number` pour dire qu'on définie que la signature
- On peut faire des **récursions**, notamment en utlisant une `UNION`

```
member function prerequisTousNiveaux return EnsU
    res EnsU;
    BEGIN
        SELECT p BULK COLLECT INTO res
        FROM table(self.prerequis) p
        UNION
        SELECT r FROM table(self.prerequis()) p, table(value(p).prerequisTousNiv()) r;
    RETURN res;
END;
```

XML

DTD

- Faire un arbre et checker que tout vas bien. **Ordre important**
- La balise après le DOCTYPE doit indiquer la balise à la racine du document.
- ~~(#PCDATA, balise1, balise2)~~ FAUX => (#PCDATA | balise1 | balise2) OK
- + = [1-n]
- * = [0-n]
- ? = [0-1]
- On peut utiliser `EMPTY` pour un élément vide, mais en général avec des attributs

```
<!ELEMENT menu EMPTY>
<!ATTLIST menu
    nom CDATA #REQUIRED
    prix CDATA #REQUIRED
>
```

Attribut

```
<!ATTLIST Balise
    nom type mode
    nom type mode
>
```

Liste des types :

- **CDATA** : la valeur de l 'attribut est une chaîne de caractères
- **ID** : identificateur d 'élément, **IDREF(S)** : renvoi vers un (des) ID. Ici c'est ez, **tous les ID sont uniques** donc on idref juste vers un ID existant.
- NMTOKEN(S) : un ou des noms symboliques (sans blanc)
- `(value1 | value2 | value3)` liste
- ENTITY(IES) : entités externes non XML Liste des mode :
- Une valeurs par default
- `#FIXED` une constante
- `#REQUIRED` Attribut requis
- `#IMPLIED` Attribut facultatif
-

XSchema

- Au vu des annales, il faut voir le cours.

- Les key et keyref la racine du parent commun. @ lorsqu'on pointe sur un attribut

```
<!-- Ville : Key -->
<xs:key name="idVille">
    <xs:selector xpath="ville"></xs:selector>
    <xs:field xpath="@nom"></xs:field>
</xs:key>

<!-- Restaurant Ref ville -->
<xs:keyref refer="idVille" name="refVille">
    <xs:selector xpath="restaurant"></xs:selector>
    <xs:field xpath="@ville"></xs:field>
</xs:keyref>
```