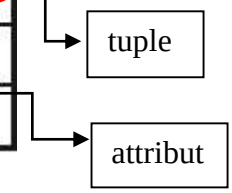
BASES DE DONNÉES

I. Modèle relationnel

Relation : tableau à 2 dimensions (tuples et attributs)

Ex :

Étudiant	num	nom	adresse	age
	1	Bélaïd	Maisel	20
	2	Millot	CROUS	20
	3	Silber	Maisel	21



Tuple (ou n-uplet) : ligne dans une relation

Attribut : colonne dans une relation

Clé [UNE SEULE CLE POSSIBLE PAR RELATION] : groupe minimum d'attributs (colonnes) qui différencie un tuple (une ligne) d'un autre. On souligne les clés.

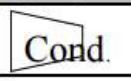
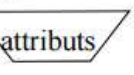
Ex : ici c'est num.

Clé étrangère dans une relation A : c'est une clé dans une autre relation (tableau) B mais qui apparaît dans la relation A. Permet de faire le lien entre 2 relations (ce qu'on appelle jointure).

Schéma d'une relation : c'est le nom de la relation suivi d'entre parenthèses de la liste des attributs

Ex : Etudiants (num, nom, adresse, age)

II. Algèbre relationnelle

Opérateur	Sémantique	Notation textuelle	Notation graphique
Restriction	« Sélectionner » des tuples	$T \leftarrow \sigma_{cond}(R)$	
Projection	« Sélectionner » des attributs	$T \leftarrow \Pi_{attributs}(R)$	
Union	Fusionner les extensions de 2 relations	$T \leftarrow R \cup S$	
Intersection	Obtenir l'ensemble des tuples communs à deux relations	$T \leftarrow R \cap S$	
Différence	Tuples d'une relation qui ne figurent pas dans une autre	$T \leftarrow R - S$	
Produit cartésien	Concaténer chaque tuple de R avec chaque tuple de S	$T \leftarrow R \times S$	
Jointure	Etablir le lien sémantique entre les relations	$T \leftarrow R \bowtie_{condition} S$	
Division	Répondre aux requêtes de type « tous les »	$T \leftarrow R \div S$	

III. SQL

Requête SQL = composition d'opérations de l'algèbre relationnelle

Syntaxe

```
SELECT  <liste d'attributs
                    projetés>

FROM   <liste de relations>

[WHERE        <liste des
  critères                  de
  restriction et de
       jointure>]
```

Remarques :

- à la fin d'une requête, mettre un « ; ».
- les jointures sont à faire dans le WHERE (on écrit l'égalité des clés
  ex : Etudiants (num,……..)
  Inscrits (**idetud, UV**, ….)

  Requête jointure :

  SELECT *

  FROM Etudiants, Inscrits

  WHERE num = idetud ;

Si jamais on avait eu 2 fois le même nom de clé (par exemple à la place de idetud on a aussi num), on aurait écrit (pour différencier) :

SELECT *

FROM Etudiants E, Inscrits I

WHERE E.num = I.num ;

- **les fonctions sont à mettre impérativement dans le SELECT jamais dans le WHERE (le where ne contient que des conditions sur les attributs)**

Mots-clés utiles :

- **DISTINCT** : enlève les doublons

Ex : SELECT DISTINCT habits (enlève les habits identiques)

- **AND/ET/IN/NOT** : « et » / « ou » / « dans » / « n'a pas » : à mettre dans un WHERE

Ex : WHERE couleur = 'rouge' AND texture = 'laine'

- **LIKE** : « contient … » , le % signifie que le reste est n'importe quoi

Ex : WHERE nom LIKE « j% » (nom commençant par j)

Ex : WHERE nom LIKE « %j% » (nom contenant j)

- * : sélection de tous les attributs

Ex : SELECT *

- **ORDER BY** : rangement par ordre croissant
- **ORDER BY DESC** : rangement par ordre décroissant
- **UNION** : union : à mettre entre 2 requêtes [REM : élimination automatique des doublons]
- **INTERSECT** : intersection : à mettre entre 2 requêtes
- **EXCEPT** : différence : à mettre entre 2 requêtes (à utiliser souvent quand la requête est une phrase négative)
- **COUNT(…)** : nombre de …
- **MIN(…)** : minimum de …
- **MAX(…)** : maximum de …
- **AVG(…)** : moyenne de …
- **SUM(…)** : somme de …

Ce sont les 5 fonctions (agrégats) prédéfinies de SQL, les « … » correspondent à des attributs.

- **GROUP BY** : partitionnement (groupes de tuples où tous les tuples de chaque groupe a la même valeur pour l'attribut) : requête de type « pour chaque … »
- **HAVING (…)** : réduction du groupe de tuples => se place après le GROUP BY. Il est impérativement suivi d'une fonction (agrégat) jamais d'un attribut !

Ex : « Donner pour chaque cru, la moyenne des degrés des vins de ce cru par degré décroissant et uniquement si ce cru concerne + de 3 vins »

```
SELECT cru, AVG(degre)

FROM Vins

GROUP BY cru

HAVING COUNT ( * ) >= 3
```

ORDER BY degre DESC ;

- **EXISTS(requête)** : teste si la réponse à une sous-requête existe
- **NOT EXISTS(requête)** : teste si la réponse à une sous-requête est vide

Ex : « Producteurs ayant produits au moins 1 vin »

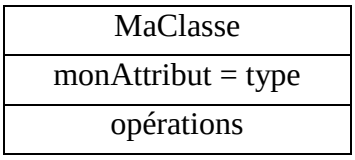
SELECT P.*

FROM Producteurs P

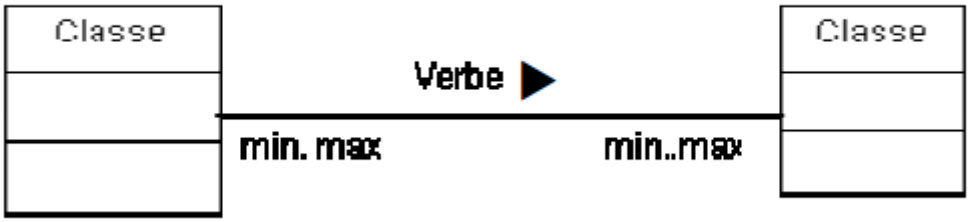
WHERE EXISTS (SELECT R.* FROM Recoltes R WHERE P.num = R.nprod) ;

IV. UML : Diagramme de classes

Construction d'une classe :



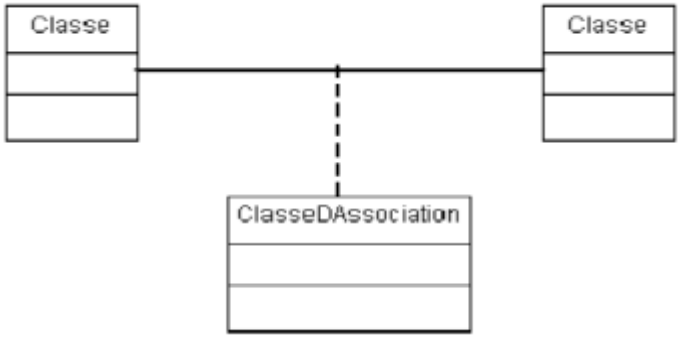
Lien entre 2 classes :



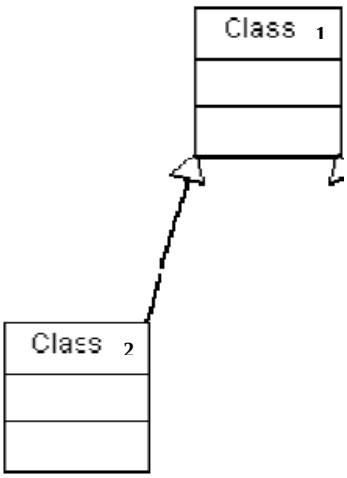
Remarque : min..max correspond aux valeurs minimales et maximales pour la classe correspondante.

- x..y : x à y
- 0..* : 0 à l'infini

Classe d'association :



Héritage : « est un(e) » [ex : si class2 = jeunes conducteurs et class1 = personne alors on a bien un héritage : class2 est une class1]



Composition / Agrégation :

Remarque : l'agrégation est moins forte que la composition. En effet, la classe agrégée peut être supprimée sans compromettre l'existence de la classe principale, contrairement aux classes composées.

Ex : pour une voiture, elle est composée d'une carrosserie (sans la carrosserie la voiture n'existe pas) et elle est agrégée de roues (la voiture peut exister sans roues).

Mais ces 2 outils sont rarement utilisés ou alors vraiment dans des cas explicites.