Lycée Vauvenargues *PT*	**Bases de données** **Algèbre Relationnelle**	 ***Cours***

I. Bases de données

1) Un peu d'histoire

Une **base de données** (en anglais : *database*) est un conteneur informatique permettant de stocker l'intégralité des informations en rapport avec une activité.
Le dispositif comporte un système de gestion de base de données (abréviation SGBD) : un logiciel moteur qui manipule la base de données et dirige l'accès à son contenu.
En 1970, la thèse mathématique de Edgar F. Codd est à l'origine des bases de données dites « relationnelles ». Elle vise à définir un modèle logique mathématique qui permet de construire, modifier, comparer, agréger des bases de données. Edgar F. Codd a reçu le prix Turing en 1981.
Entre 1970 et 1980, IBM développe un langage qui permet de dialoguer avec les bases de données. Il est nommé SQL (*Structured Query Language =* Langage de requêtes structuré). Il devient la norme dès 1986.
SQL est un langage de définition de données (LDD), de manipulation de données (LMD), et de contrôle de données (LCD). On parle alors de DCL (Data Control Language).

2) Vocabulaire spécifique de l'Algèbre Relationnelle

L'**algèbre relationnelle** est une théorie mathématique proche de la théorie des ensembles qui définit des opérations qui peuvent être effectuées sur des tables contenant un ensemble de n-uplets

Vocabulaire : les termes utilisés dans ce chapitre sont souvent des faux-amis, prenez garde !

1. Une **relation** est une table. Une relation (au sens du modèle de Codd) est constituée de deux choses :
   -- Un **Schéma** : Le format de la table. Le schéma est fixé.
   -- Une **Extension** : Le contenu de la table.
2. Un ***n*-uplet (ou tuple** en anglais) est une collection ordonnée de *n* objets. Dans une relation, les tuples sont les lignes d'enregistrements.
3. Les noms des colonnes désignent les **attributs** de la relation, chacun ayant un **domaine** prédéfini (les types classiques : INTEGER, VARCHAR,REAL)

II. Opérateurs classiques sur l'ensemble des relations

Dans la suite, on étudie, de manière non exhaustive, les différents opérateurs sur l'ensemble des bases de données.

- Projection (π) : opérateur **SELECT**
- Sélection (ou restriction) (σ) : opérateur **WHERE**
- Opérateurs ensemblistes :

$$\left.\begin{array}{r}\text{Union } (\cup)\\ \text{Intersection } (\cap))\\ \text{Différence } (\setminus\end{array}\right\}\text{pour les relations de même schéma, traduits en SQL par : }\left\{\begin{array}{c}\textbf{OR}\\ \textbf{AND}\\ \textbf{EXCEPT}\end{array}\right.$$

- Produit cartésien (×)
- Jointure (⋈) : opérateur **JOIN**
- Fonctions d'agrégation

1) Opérateur Projection (π) :

Soit une relation R, la projection crée une nouvelle relation qui regroupe l'ensembles des tuples de R réduits aux seuls attributs de la liste spécifiée.
Cet opérateur ne porte que sur 1 relation.
Il permet de ne retenir que certains attributs spécifiés d'une relation.

Syntaxe : S = π [attribut_1, attribut_2,…](Relation)

Exemple :

Titre	Qualité	Genre	Support
Godzilla	Blu-Ray720p	Action	Disque Dur
Godzilla	Blu-ray3D	Action	DVD
Godzilla	Blu-ray1080p	Action	DVD
Batman	Blu-ray1080p	Comics	Disque Dur
Batman	Blu-Ray720p	Comics	DVD
L'aile ou la cuisse	Blu-Ray720p	Comédie	DVD

```
SELECT Titres
FROM Films
```

```
SELECT Titre,Support
FROM Films
```

S1 = π [Titre](Films)

Titre
Godzilla
Batman
L'aile ou la cuisse

S2= π [Titre,Support](Films)

Titre	Support
Godzilla	Disque Dur
Godzilla	DVD
Batman	Disque Dur
Batman	DVD
L'aile ou la cuisse	DVD

Remarque : On obtient tous les n-uplets de la relation **à l'exception des doublons.**

2) Opérateur Sélection (ou restriction) (σ) :

Soit une relation R, la sélection crée une nouvelle relation composée de tuples vérifiant une condition.

Cet opérateur permet de ne retenir que les n-uplets répondant à une condition exprimée à l'aide des opérateurs arithmétiques (=, >, <, >=, <=, <>) ou logiques de base (ET, OU, NON).
Tous les attributs de la relation sont conservés.

Syntaxe: S= σ [test](relation)

Exemple :

```
SELECT *
FROM Films
WHERE Qualité ="Blu-Ray720p"
```

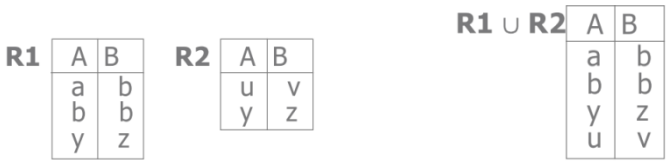
S= σ [Qualité=''Blu-Ray720p''](Films)

Titre	Qualité	Genre	Support
Godzilla	Blu-Ray720p	Action	Disque Dur
Batman	Blu-Ray720p	Comics	DVD
L'aile ou la cuisse	Blu-Ray720p	Comédie	DVD

3) Opérateur Union :

Soient R1 et R2 deux relations (de même schéma), l'union, **R1∪R2**, réunit dans une même relation les tuples de R1 et R2.

Exemple :



R1 : Garçons

N° Ens.	Nom	Prénom	Matière
12	Aire	Axel	Sciences Industrielles
14	Anchier	Yvon	Maths
15	Bambelle	Larry	Sciences Physiques
19	Moncostume	Jimmy	Maths

R2 : Filles

N° Ens.	Nom	Prénom	Matière
13	Anchier	Eva	Maths
16	Diote	Kelly	Sciences Industrielles
17	Kourcy	Sarah	Maths
18	Melmou	Kara	Sciences Physiques
20	Nenfant	Ludivine	Sciences Industrielles
21	Oduvillage	Lydie	Sciences Physiques

```
SELECT * FROM R1
UNION
SELECT * FROM R2
```

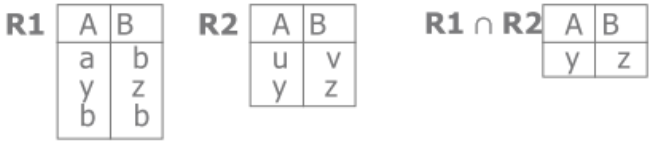
E1= R1∪R2

N° Ens.	Nom	Prénom	Matière
12	Aire	Axel	Sciences Industrielles
13	Anchier	Eva	Maths
14	Anchier	Yvon	Maths
15	Bambelle	Larry	Sciences Physiques
16	Diote	Kelly	Sciences Industrielles
17	Kourcy	Sarah	Maths
18	Melmou	Kara	Sciences Physiques
19	Moncostume	Jimmy	Maths
20	Nenfant	Ludivine	Sciences Industrielles
21	Oduvillage	Lydie	Sciences Physiques

4) Opérateur Intersection :

Soient R1 et R2 deux relations (de même schéma), l'intersection, **R1∩R2**, sélectionne les tuples qui sont à la fois dans R1 et R2.

Exemple :



R1 : Elèves pratiquant le foot

N°	Nom	Prénom
12	Aire	Axel
13	Anchier	Eva
19	Moncostume	Jimmy
15	Bambelle	Larry
20	Nenfant	Ludivine
21	Oduvillage	Lydie

R2 : Elèves pratiquant le Basket :

N°	Nom	Prénom
14	Anchier	Yvon
19	Moncostume	Jimmy
15	Bambelle	Larry
18	Melmou	Kara

On désire connaitre les élèves pratiquant du Foot et du Basket :

```
SELECT * FROM R1
INTERSECT
SELECT * FROM R2
```

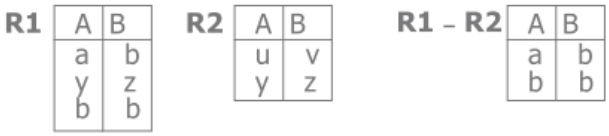
E2= **R1∩R2**

N°	Nom	Prénom
19	Moncostume	Jimmy
15	Bambelle	Larry

5) Opérateur Différence :

Soient R1 et R2 deux relations (de même schéma), la différence, **R1 \ R2** (ou R1-R2), sélectionne les tuples de R1 qui ne sont pas dans R2.

Exemple :



R1 : Elèves pratiquant le foot

N°	Nom	Prénom
12	Aire	Axel
13	Anchier	Eva
19	Moncostume	Jimmy
15	Bambelle	Larry
20	Nenfant	Ludivine
21	Oduvillage	Lydie

R2 : Elèves pratiquant le Basket :

N°	Nom	Prénom
14	Anchier	Yvon
19	Moncostume	Jimmy
15	Bambelle	Larry
18	Melmou	Kara

On désire connaitre la liste des élèves pratiquant du Foot qui ne font pas du Basket :

```
SELECT * FROM R1
EXCEPT
SELECT * FROM R2
```

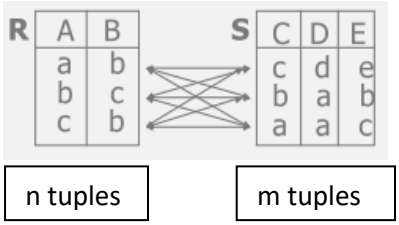
E2= **R1 \ R2**

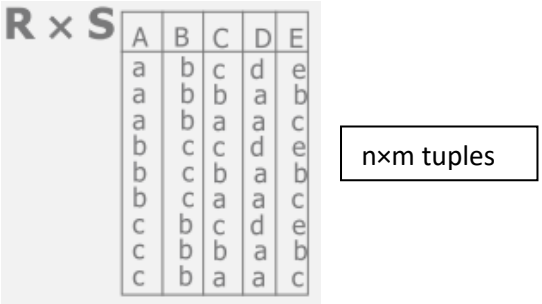
N°	Nom	Prénom
12	Aire	Axel
13	Anchier	Eva
20	Nenfant	Ludivine
21	Oduvillage	Lydie

6) Opérateur Produit cartésien :

Soient R1 et R2 deux relations, le produit cartésien, **R1×R2**, construit toutes les combinaisons de tuples des deux relations. Chaque tuple de R1 est combiné avec chaque tuple de R2.

Exemple :





R1 : Liste d'étudiants

N° Ens.	Nom	Prénom
12	Aire	Axel
13	Anchier	Eva
14	Anchier	Yvon
15	Bambelle	Larry

R2 : Epreuves

Epreuve	Coefficient
Repassage	3
Couture	2
cuisine	4

La requête SQL :

```
SELECT *
FROM R1,R2
```

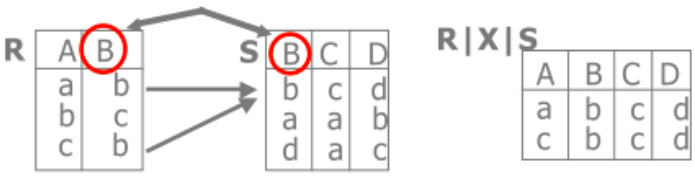
renvoie **R1×R2**

N° Ens.	Nom	Prénom	Epreuve	Coefficient
12	Aire	Axel	Repassage	3
12	Aire	Axel	Couture	2
12	Aire	Axel	cuisine	4
13	Anchier	Eva	Repassage	3
13	Anchier	Eva	Couture	2
13	Anchier	Eva	cuisine	4
14	Anchier	Yvon	Repassage	3
14	Anchier	Yvon	Couture	2
14	Anchier	Yvon	cuisine	4
15	Bambelle	Larry	Repassage	3
15	Bambelle	Larry	Couture	2
15	Bambelle	Larry	cuisine	4

7) Opérateur Jointure symétrique (ou équi-jointure) (⋈):

Cet opérateur porte sur 2 relations qui doivent avoir au moins un attribut défini dans le même domaine (ensemble des valeurs permises pour un attribut).
La condition de jointure peut porter sur l'égalité d'un ou de plusieurs attributs définis dans le même domaine (mais n'ayant pas forcément le même nom).
Les n-uplets de la relation résultat sont formés par la concaténation des n-uplets des relations d'origine qui vérifient la condition de jointure.

Exemple :



Coureurs

N°	Nom	Code_équipe	Code_pays
8	ULLRICH Jan	TEL	All
31	JALABERT Laurent	ONC	Fra
61	ROMINGER Tony	COF	Sui
91	BOARDMAN Chris	GAN	G-B

Pays

Code_pays	Nom_pays
All	Allemagne
Fra	France
Sui	Suisse
G-B	Grande-Bretagne

La requête SQL :

```
SELECT *
FROM Coureurs JOIN Pays ON Coureurs.Code_pays = Pays.Code_pays
```

renvoie : Coureurs⋈Pays

N°	Nom	Code équipe	Code pays	Nom pays
8	ULLRICH Jan	TEL	All	Allemagne
31	JALABERT Laurent	ONC	Fra	France
61	ROMINGER Tony	COF	Sui	Suisse
91	BOARDMAN Chris	GAN	G-B	Grande-Bretagne

8) Fonctions d'agrégation

Les fonctions d'agrégation permettent d'effectuer des opérations statistiques sur un ensemble d'enregistrement.

a) Liste des fonctions d'agrégation statistiques

Les fonctions d'agrégation sont des fonctions idéales pour effectuer quelques statistiques de bases sur des tables. Les principales fonctions sont les suivantes :

- AVG() pour calculer la moyenne sur un ensemble d'enregistrement
- COUNT() pour compter le nombre d'enregistrement sur une table ou une colonne distincte
- MAX() pour récupérer la valeur maximum d'une colonne sur un ensemble de ligne. Cela s'applique à la fois pour des données numériques ou alphanumérique
- MIN() pour récupérer la valeur minimum de la même manière que MAX()
- SUM() pour calculer la somme sur un ensemble d'enregistrement

b) Utilisation simple

Soit f une des six fonctions d'agrégation. L'utilisation la plus générale consiste à utiliser la syntaxe suivante :

```
SELECT f(colonne) FROM table
```

La fonction COUNT() possède une subtilité. Pour compter le nombre total de ligne d'une table, il convient d'utiliser l'étoile « * » qui signifie que l'ont cherche à compter le nombre d'enregistrement sur toutes les colonnes. La syntaxe serait alors la suivante :

```
SELECT COUNT(*) FROM table
```

c) Utilisation avec GROUP BY

Toutes ces fonctions prennent tout leur sens lorsqu'elles sont utilisées avec la commande GROUP BY qui permet de filtrer les données sur une ou plusieurs colonnes. Imaginons une table qui contient tout les achats sur un site avec le montant de chaque achat pour chaque enregistrement. Pour obtenir le total des ventes par clients, il est possible d'exécuter la requête suivante :

```
SELECT client, SUM(tarif)
 FROM achat
 GROUP BY client
```

d) Utilisation évoluée : GROUP BY … HAVING …

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions d'agrégation *f*.

L'utilisation de HAVING s'utilise de la manière suivante :

```
SELECT colonne1, f(colonne2)
FROM nom_table
GROUP BY colonne1
HAVING condition       # ligne de sélection utilisant la fonction f
```

Cela permet donc de SÉLECTIONNER les colonnes de la table « nom_table » qui vérifient la condition après HAVING, mais en veillant à GROUPER les lignes qui ont des valeurs identiques sur la « colonne1 ».

Important : HAVING est très souvent utilisé en même temps que GROUP BY.

Exemple :

Pour utiliser un exemple concret, imaginons une table « achat » qui contient les achats de différents clients avec le coût du panier pour chaque achat.

id	client	tarif	date_achat
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Marie	18	2012-11-05
4	Marie	20	2012-11-14
5	Pierre	160	2012-12-03

Si dans cette table on souhaite récupérer la liste des clients qui ont commandé plus de 40€, toutes commandes confondues, alors il est possible d'utiliser la requête suivante :

```
SELECT client, SUM(tarif)
FROM achat
GROUP BY client
HAVING SUM(tarif) > 40
```

qui renvoie :

client	SUM(tarif)
Pierre	262
Simon	47