COURS 6 : REVISIONS BASES DE DONNEES

Compétences attendues :

- Client/serveur → architecture trois-tiers
- Vocabulaire des bases de données : relation, attribut, domaine, schéma de relation ; notion de clé primaire
- Opérateurs usuels sur les ensembles dans un contexte de bases de données : union, intersection, différence.
- Opérateurs spécifiques de l'algèbre relationnelle : projection, sélection (ou restriction), renommage, Jointure symétriques ou simples, produit et division cartésiennes ; fonctions d'agrégation : min, max,somme, moyenne, comptage

1. INTRODUCTION

Les bases de données appartiennent aux systèmes informatiques qui nous aident à gérer des données très diverses. Nous avons donc, d'un côté, un serveur de données quelque part sur la Toile, avec des disques magnétiques et leurs pistes qui gardent précieusement des séquences de bits, des structures d'accès compliquées comme des index ou des arbres-B, des hiérarchies de mémoires avec leurs caches et, de l'autre, un utilisateur.

Un moteur de recherche de la Toile ne fait pas autre chose, seulement il le fait sur un système de fichiers à l'échelle de la planète. Les systèmes qui gèrent aussi des données mais qui sont bien plus sophistiqués que les systèmes de fichiers : les **systèmes de gestion de bases de données.** (SGBD)

Ce sont des logiciels complexes, résultats de dizaines d'années de recherche et de développement. Ils permettent à des individus ou des programmes d'exprimer des requêtes pour interroger des bases de données ou pour les modifier. Nous nous focaliserons ici sur les plus répandus d'entre ces systèmes, **les systèmes relationnels**, parmi lesquels nous trouvons :

- des logiciels commerciaux très répandus comme celui d'Oracle ;
- et des logiciels libres très utilisés comme **MySQL.**

1.1. Principe de l'architecture

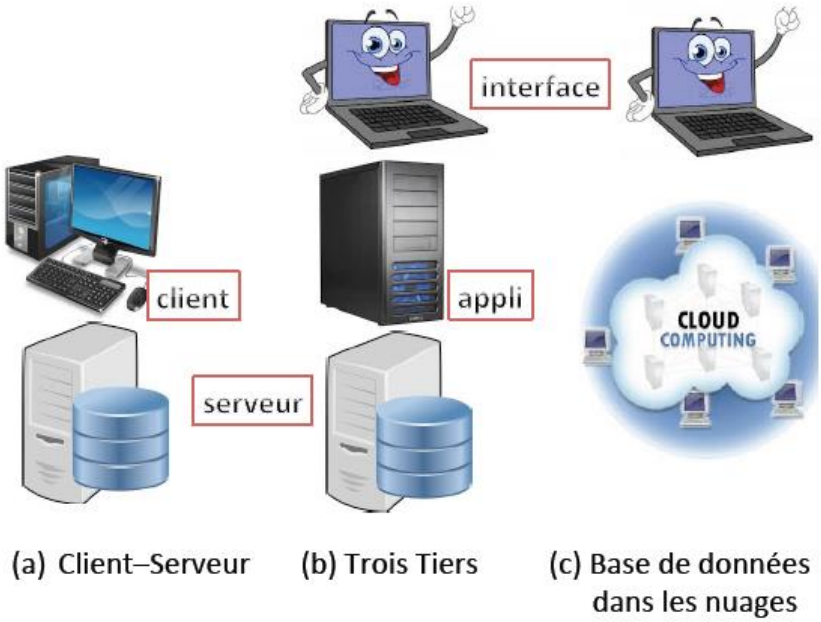
- *Abstraction :* le système doit organiser et présenter les données de façon intuitive et permettre de les manipuler en restant à un niveau abstrait sans avoir à considérer des détails d'implémentation.
- *Indépendance :* nous distinguons trois niveaux, physique, logique et externe. Le but est de pouvoir modifier un niveau (par exemple, ajouter un nouvel utilisateur externe avec de nouveaux besoins) sans modifier les autres niveaux.
- *Universalité :* capturer toutes les données d'une entreprise, d'un groupe, d'une organisation quelconque, pour tout type d'applications

Une première architecture est celle des systèmes client/serveur. La base de données est gérée sur un serveur. L'application tourne sur une autre machine, le client. De plus en plus, cette architecture se complique avec l'introduction d'un troisième "tiers" , une machine qui gère l'interface, typiquement un navigateur Web.

On appelle SGBD un système de gestion de base de données, en anglais DBMS *(Data Base Management System*).

Nous pouvons noter différentes évolutions générées par des améliorations dans les matériels disponibles :

- l'accroissement des performances notamment fondées sur les mémoires vives de plus en plus massives, et des mémoires flash encore plus massives ;
- l'utilisation de plus en plus de parallélisme massif dans des grappes de machines pour traiter d'énormes volumes de données. On parle parfois de "big data" ;
- pour simplifier la gestion de données, on tend à la déporter dans les nuages (le cloud), c'est-à-dire à mettre ses données dans des grappes de machines gérées par des spécialistes comme Amazon.



1.2. Organisation d'une Base de données

Généralement, une base de données s'organise de la manière suivante :

BASE DE DONNEES/DATABASE
TABLE(S)/TABLE(S)
CHAMPS/FIELD(S)
VALEUR(S)/VALUE(S)

Les éléments d'une base de données sont rangés **en tables**, qui elles-mêmes contiennent des **champs** qui peuvent avoir plusieurs **valeur**s.

Le modèle le plus courant pour représenter les bases de données est le modèle relationnel, qui utilise l'algèbre relationnelle.

Dans le modèle relationnel, les données sont organisées en tableaux à deux dimensions que nous appellerons des *relations*. À la différence des mathématiciens, nous supposons les **relations de taille finie.**

Une table ou relation est donc un tableau fini à deux dimensions, dans lequel les éléments de chaque colonne sont tous de même type.

Une colonne est appelée **attribut.**

À chaque attribut est associé un **domaine** $Di = dom(Ai)$, généralement égal à un type usuel, comme des chaînes de caractères, où des adresses postales, des dates, des entiers positifs, des noms propres, etc

*Exemple des films en créant la base de données **CINEMA***

FILM			SEANCES		
Titre	**Réalisateur**	**Acteur**	**Titre**	**Salle**	**Heure**
Interstallar	C. Nolan	Matthew McConaughey	La famille Bélier	1	19 :00
Imitation Game	M.Tyldum	Matthew Goode	Imitation Game	2	20 :00
La famille Bélier	E. Lartigau	François Damiens	Interstellar	3	19 :15
Le Hobbit 3	P. Jackson	Martin Freeman	Interstellar	3	22 :30

Donner 3 attributs avec 3 domaines différents :

-
-
-

Plus généralement, on appelle schéma relationnel $S = (A_1, \dots, A_n)$ un ensemble fini d'attributs, et alors une table ou relation est un ensemble fini de n-uplets de $D_1 \times \dots \times D_n$.

On peut aussi noter $S = ((A_1, D_1), \dots, (A_n, D_n))$, et $R(S)$ pour une relation R de schéma relationnel S. Par exemple :

$$S = ((Acteur; Texte);\ (Réalisateur; Texte);\ (Salle; \mathbb{N});\ (Heure; \mathbb{R}))\ :$$

L'**arité** d'un schéma relationnel est son nombre d'attributs.

Donner le schéma relationnel et l'arité de FILM :

Une clef permet de différencier toutes les lignes de la table R.

On peut citer comme exemple le numéro d'abonné, ou le numéro de film mais pas le nom du réalisateur, ni du film. Dans le cas où la clef est constituée d'un unique attribut, on parle **d'identifiant.**

On choisit en général une clef particulière qu'on appelle **clef primaire.**

Exemple CINEMA : Ici il n'y a pas encore de clé ! Il faudra donner une clé primaire à chacun des films.

L'algèbre relationnelle consiste en un petit nombre d'opérations de base qui, appliquées à des relations, produisent de nouvelles relations. Ces opérations peuvent être composées pour construire des expressions algébriques de plus en plus complexes.

Pour répondre à la question *Où et à quelle heure puis-je voir un film avec Francois Damiens ?*, et pour faciliter la lisibilité de ces formules logiques, on peut exprimer une requête en définissant précisément la forme du n-uplet (nommée *res*) en lui affectant les variables libres que l'on cherche à obtenir :

$$res(s,h), \exists t, r\ ; \left(FILM\ (t, r, "\ Francois\ Damiens") \wedge SEANCES(t, s, h)\right)$$

L'intérêt de ce procédé est qu'on peut ensuite réutiliser les n-uplets résultats exactement comme s'il s'agissait d'une relation. C'est ce qu'on appelle le mécanisme des *vues*.

Une vue est une relation, qui au lieu d'être stockée dans la base de données, est définie intentionnellement.

Un utilisateur de la base de données peut l'utiliser comme n'importe quelle autre relation.

En pratique, les machines utilisent le langage **SQL** (*pour Structured Query Language*) qui exprime différemment les mêmes questions. Par exemple la question précédente s'exprime en SQL comme une **REQUETE** :

```
SELECT salle, heure
FROM Film, Seance
WHERE Film.titre = Seance.titre AND acteur= "Francois Damiens"
```

Par la suite, nous allons donc étudier comment écrire en algèbre relationnelle la question qui nous sert d'exemple. Il nous faudra trois opérations, la jointure, la sélection et la projection, que nous composerons dans l'expression suivante de l'algèbre relationnelle :

$$E_{HB} = \pi_{salle,heure}\ (\pi_{titre}(\sigma_{acteur} = "FrancoisDamiens")(FILM)\ ) \bowtie SEANCES)$$

L'opération de sélection, dénotée σ, filtre une relation, ne gardant que les n-uplets satisfaisant une condition, ici acteur = ,$"\ FrancoisDamiens"$. L'opération de projection, dénotée π, permet aussi de filtrer de l'information d'une relation mais cette fois en éliminant des colonnes. L'opération peut-être la plus exotique de l'algèbre, la *jointure*, dénotée $\bowtie$, combine des n-uplets de deux relations.

2. OPERATEURS

2.1. Opérateurs simples

2.1.1. Union, intersection, différence

Soit $R_1(S)$ et $R_2(S)$ deux relations de même schéma relationnel S.

Alors les relations $R_1 \cup R_2$, $R_1 \cap R_2$ et $R_1 - R_2$, appelé respectivement union, intersection, différence de R_1 et de R_2, sont les relations de schéma relationnel S qui contiennent les lignes qui appartiennent respectivement à R_1 ou à R_2, à R_1 et à R_2, à R_1 mais pas à R_2.

2.1.2. Projection

Soit R une table de schéma relationnel $S = (A_1, \dots, A_n)$ et soit $S' = (A_{i1};, \dots, A_{ik})$ une partie de S, c'est-à-dire qu'on ne garde que certains attributs. La projection de R sur S' est la table :

$$\pi_{S'}(\mathrm{R}) = \{(eA_{i1}, \dots, eA_{ik}), e \in R\}$$

2.1.3. Sélection

Si A est un attribut de $R, et\ a$ un élément du domaine de A alors on appelle sélection ou restriction de R selon la condition $A = a$, la relation contenant les lignes de R satisfaisant la condition $A = a$.

$$\sigma_{A=a}(R) = \{e \in R, e.A = a\}$$

2.1.4. Renommage

Soit A_i un attribut de R. On peut changer son nom, on dit qu'on renomme son nom en B : $\rho_{B \leftarrow A}(R)$ est la relation R dans laquelle on a juste changé le nom de A_i en B.

2.2. Opérateurs complexes

2.2.1. Produit et division cartésienne

Soit $R(S)$ et $R'(S')$ deux relations. On note $S \cup S'$ l'union de S et de S' s'ils sont disjoints ($S'' = S \uplus S'$ équivaut à $S'' = S + S'$ et $\cap\ S' = \emptyset$).

On définit alors <u>le produit cartésien</u> $R \times R'$ comme étant la relation de schéma $S \uplus S'$ contenant les couples (e, e') tels que $e \in R$ et $e' \in R'$.

Si

$$S = (A_1, \dots, A_n)\ et\ S' = (B_1, \dots, B_m)$$

alors $R \times R'$ admet pour schéma $S \uplus S' = (A_1, \dots, A_n, B_1, \dots, B_m)$, et

$$R \times R' = \{(x_1, \dots, x_n, y_1, \dots, y_m) \mid (x_1, \dots, x_n) \in R\ et\ (y_1, \dots, y_m) \in R'\}$$

2.2.2. Jointure symétrique

Cette opération, **dénoté ⋈,** s'applique à deux relations quelconques I et J d'attributs V et W, respectivement, et produit une relation d'attributs $V \cup W$:

$$I \bowtie J = \{t\ sur\ V \cup W \mid il\ existe\ v \in I\ et\ w \in J,\ t|_V = v\ et\ t|_W = w\}$$

Quand $dom(V) = dom(W)$, $I \bowtie J = I \cap J$ (c'est l'intersection ensembliste classique),

Quand $dom(V) \cap dom(W) = \emptyset$; alors $I \bowtie J$ est le produit cartésien de I et J, dénoté $I \times J$.

➔ Ici on ne considère que les <u>jointures symétriques $dom(V) = dom(W)$</u> donc la relation est

$$I \bowtie J = \{(e, e') \in I \times J \mid e.V = e'W\} = \sigma_{V=W}(I \times J)$$

2.2.3. Fonctions d'agrégation : min, max, somme, moyenne, comptage

On regroupe les lignes d'une relation selon un critère puis on applique la fonction. Ceci produit une nouvelle table.

Soit R une relation de schéma S. Soit A et B deux attributs de R, c'est à dire deux éléments de S. On note $A\gamma f(B)(R)$ la table obtenue :

- On regroupe les lignes qui ont le même attribut A, ce qui forme des agrégats,
- puis on applique la fonction f sur les agrégats, ce qui crée une table dont les attributs sont A et $f(B)$.

Plus généralement, soit $A1; \dots ; An$ et $B1; \dots ; Bm$ des attributs de R, et $f1 ; \dots ; fm$ des fonctions d'agrégation sur les Bj. On note $A1; \dots ; An\gamma f1(B1); \dots ; fm(Bm)(R)$ la table obtenue en agrégeant selon les Ai (tous les Ai doivent être égaux), et en appliquant les fonction fj sur les agrégats.

3. SYNTAXE SQL

L'algèbre relationnelle fait partie des fondements des bases de données relationnelles, utilisé en interne par le SGBD pour évaluer les requêtes. Le langage *SQL* quant à lui est destiné aux utilisateurs du SGBD et, à l'heure actuelle, est **le langage standard d'interrogation**. Il marie d'une certaine façon l'algèbre relationnelle et le calcul relationnel de n-uplets.

Le langage de définition de données permet la modification du schéma d'une base de données. Il propose trois opérations : la création (CREATE), la suppression (DROP) et la modification (ALTER). Par exemple, la création d'une table a pour syntaxe :

```
CREATE TABLE (table) ((définitions de colonnes) [( contraintes de table)]).
```

Le langage DDL permet de spécifier qu'un ensemble d'attributs est clé primaire (PRIMARY KEY), qu'une séquence d'attributs est clé étrangère (FOREIGN KEY/REFERENCES). Il permet aussi de contraindre le domaine d'un attribut (CHECK) ou qu'un ensemble d'attributs ne contient pas de valeur manquante (NOT NULL). Enfin, il permet, à la manière de PRIMARY KEY, de préciser qu'un ensemble d'attributs ne contient pas de doublons (UNIQUE) ; à la différence de PRIMARY KEY, il est possible de spécifier plusieurs UNIQUE sur la même table.

Exemple CINEMA : table SALLES

```
CREATE TABLE SALLES(
Nom VARCHAR(30) PRIMARY KEY,
Adresse VARCHAR(255) UNIQUE NOT NULL,
Telephone VARCHAR(10)
);
```

L'attribut Nom est

Maintenant que les tables sont créées on peut insérer des données :

```
INSERT INTO salles
VALUES('ABC', '51 rue des Ecoles', '0561545160');
```

Le dernier aspect du langage SQL que nous étudierons est l'interrogation des données grâce à l'instruction SELECT :

```
SELECT (liste d'expressions)
FROM (liste de tables)
WHERE (conditions)
GROUP BY (liste d'attributs)
HAVING (conditions)
ORDER BY (liste d'attributs)
```

Les rôles des trois premières lignes sont les suivants :

- ➔ La clause SELECT spécifie le schéma de sortie (**projection)**.
- ➔ La clause FROM précise les tables impliquées et leurs liens (**produit cartésien et jointures**).
- ➔ La clause WHERE fixe les conditions que doivent remplir les n-uplets résultats (**sélection)**.

Les trois dernières clauses nous font sortir du calcul et de l'algèbre relationnelle :

- ➔ La clause GROUP BY comment regrouper des n-uplets (**agrégation**) ?
- ➔ La clause HAVING impose une condition sur les groupes (i.e. permet **d'éliminer l'intégralité** d'un groupe, en se basant sur la valeur d'un agrégat calculé sur ce groupe).
- ➔ Enfin ORDER BY définit les critères de tris des résultats.

La clause SELECT est suivie d'une liste d'expressions qui reprennent les opérations de l'algèbre relationnel. Le tableau suivant regroupe ces expressions les plus courantes :

Opérateur	Algèbre	SQL
Table entière	R	
Projection	$\pi_{A_1,...,A_n}(R)$	
Sélection	$\sigma_{A=a}(R)$	
Intersection	$R_1 \cap R_2$	
Union	$R_1 \cup R_2$	
Différence	$R_1 - R_2$	
Renommage	$\rho_{A_1,...,A_n \leftarrow B_1,...,B_n}(R)$	
Produit cartésien	$R_1 \times R_2$	
Jointure	$R_1 \bowtie_{A=B} R_2$	
Fonction d'agrégation	$\gamma_{f(A)}(R)$	
Agrégation	${}_A\gamma_{f(B)}(R)$	
Sélection en amont puis agrégation	${}_A\gamma_{f(B)} \circ \sigma_P(R)$	
Agrégation puis sélection en aval	$\sigma_P \circ {}_A\gamma_{f(B)}(R)$	

- ***AVG()*** *pour calculer la moyenne d'un set de valeur.*
- ***COUNT()*** *pour compter le nombre de lignes concernées.*
- ***MAX()*** *pour récupérer la plus haute valeur.*
- ***MIN()*** *pour récupérer la plus petite valeur.*
- ***SUM()*** *pour calculer la somme de plusieurs lignes.*

4. CONSULTATION D'UNE BASE DE DONNEES EN PYTHON

Il est possible d'interroger la base de données en utilisant Python. Pour cela, on utilise le formalisme suivant :

```
import os # Import des commandes permettant de gérer les répertoires de travail
os.chdir('C:/Users/Revisions Bdd') # on se place dans le répertoire ou se trouve la base de  donnée

import sqlite3 # Import des commandes permettant de manipuler la base de données
basesql = u"aeroports.db3" # Base de données initiale
cnx = sqlite3.connect(basesql )
curseur = cnx.cursor ()
requete = "SELECT * FROM airports"
curseur .execute(requete)
```

Le préfixe *u* permet d'importer les fichiers encodés en UTF-8. Dans une certaine mesure, les caractères spéciaux sont alors pris en compte.

Le curseur est un objet contenant le résultat de la requête. Pour visualiser la première entité de la requête, la syntaxe est la suivante :

```
data = curseur.fetchone()
print (data)
```

Attention, à chaque utilisation de fetchone(), l'entité est supprimée du curseur. Pour parcourir chacune des entités, on peut utiliser la syntaxe suivante :

```
for cur in curseur :
    print (cur)
```

Attention, cette opération peut s'avérer maladroite si la requête a un grand nombre d'entités.

Donner la requête SQL qui détermine le nombre de bases d'hydravion existantes. En utilisant la fonction d'agrégation COUNT pour obtenir un résultat équivalent

```
requete2 =
```

Donner la requête SQL ainsi que son expression en algèbre relationnelle qui permet de trouver liste des villes françaises (iso_country='FR') hébergeant de telles bases.

```
requete3 =
```