Chapitre 6 - Bases de données

Table des matières

1 Étude d'un exemple

§ 1. On souhaite enregistrer des informations concernant un grand nombre d'étudiants en CPGE dans toute la France. On va donc créer un fichier contenant un certain nombre de caractéristiques, typiquement :

— nom de l'étudiant
— sexe
— date de naissance
— lycée
— ville de ce lycée
— filière
— classe
— professeurs
— matières suivies
— notes
— etc

§ 2. Une manière basique de représenter ces données est un *tableau* du type :

nom	sexe	naissance	lycée	ville	filière	matière	⋯
Astet	F	2000	Poincaré	Nancy	MPSI	mathématiques	
Astet	F	2000	Poincaré	Nancy	MPSI	physique	
Callot	M	2002	Saint-Louis	Paris	PCSI	chime	
Hervat	F	2000	Massena	Nice	PTSI	SI	
Pirane	M	2001	Dumont d'Urville	Toulon	MPSI	anglais	
Meunier	F	2000	Massena	Nice	PCSI	mathématiques	
Rais	M	2001	Dumont d'Urville	Toulon	MPSI	mathématiques	
⋮							

§ 3. Idéalement une fois le fichier créé :

— il faut le rendre accessible en consultation à un certain nombre de personnes (rectorats, établissements scolaires, parents, étudiants, etc) qui doivent pouvoir consulter le fichier et en extraire les informations qui leur sont utiles.
— Il faut sécuriser l'accès aux données : certaines informations ne sont pas accessibles aux parents, d'autres sont accessibles mais pas modifiables, etc
— l'administrateur du fichier doit pouvoir mettre à jour la table des données :

1. ajouter ou enlever des lignes (départ ou arrivée d'étudiants)
2. ajouter des colonnes supplémentaires (enrichissement des données)
3. modifier les données existantes (par exemple si erreur de saisie)

Exemple 1. Concrètement, on souhaite pouvoir interroger la base de donnée pour obtenir rapidement :

— la liste des élèves en PCSI au lycée Dumont d'Urville
— la liste des étudiants nés en 2000 inscrits en MPSI dans un lycée de Bordeaux
— la liste des lycées accueillant des CPGE, de leurs filières et des effectifs de leurs classes
— etc

§ 4. L'utilisation d'une table unique pour représenter les données pose un certain nombre de problèmes pratique :

1. des *redondances* : les lignes contenant « lycée Massena »seront toutes associées à la ville de Nice
2. des *entrées multiples* : le nom d'un lycée apparait autant de fois qu'il y a d'élèves dans ce lycée

La solution consiste à alléger la base en répartissant les données dans plusieurs tables.

Exemple 2. Nous pouvons par exemple subdiviser le tableau de départ en trois tables appelées **etudiants**, **classes** et **professeurs** :

etudiants				
id_etudiant	nom	sexe	naissance	id_classe
1	Astet	F	2000	1
2	Callot	M	2002	2
3	Hervat	F	2000	8
4	Pirane	M	2001	4
5	Meunier	F	2000	3
6	Rais	M	2001	4
⋮				

classes				
id_classe	lycée	ville	filière	effectif
1	Poincaré	Nancy	MPSI	45
2	Saint-Louis	Paris	PCSI	42
3	Massena	Nice	PCSI	35
4	Dumont d'Urville	Toulon	MPSI	46
5	Dumont d'Urville	Toulon	PCSI	38
⋮				

professeurs			
id_prof	nom	id_classe	matière
1	Legrand	2	mathématiques
2	Latour	8	mathématiques
3	Dupont	3	anglais
4	Henri	4	physique
⋮			

Remarque.

- Pour éviter des redondances et faciliter l'accès aux informations, nous avons numéroté les lignes en créant des colonnes particulières *id_etudiants, id_classe, id_prof* (clés primaires)
- les tables **etudiants** et **professeurs** sont liées à la table **classes** par une colonne id_classe (clés étrangères). Nous en auront besoin pour chercher des informations faisant appel à plusieurs tables.

2 client et serveur

§ 5. Quelques contraintes à la manipulation des bases de données :

- Elles sont souvent consultables par un grand nombre de personnes
- Ces utilisateurs peuvent certes accéder aux informations de la BDD, mais la plupart n'auront pas le droit de modifier la BDD
- Une interface logicielle et matérielle doit être mise en place pour assurer la communication entre l'utilisateur et la BDD

Pour toutes ces raisons, l'utilisation d'une BDD va s'appuyer sur un système clients / serveur :

Définition 1.
★ Un serveur est un dispositif informatique offrant des services à des clients.
★ Le client est un matériel logiciel ou informatique, permettant l'envoi de requêtes à un serveur donné et la réception des réponses.

Exemple 3. Voici quelques services que peut assurer un serveur :
- partage de fichiers
- partage de données
- hébergement de site web
- courrier électronique
- cloud computing

Remarque. Quelques caractéristiques demandées à un serveur :
- communication (débit) et traitement rapide
- adaptation à la demande et au nombre souvent élevé de clients
- disponibilité 24 h sur 24

Définition 2.
★ Dans une architecture trois tiers (client-BDD-SGBD), le serveur est scindé en deux serveurs :
  - serveur 1 : stockage de l'information (base de donnée BDD)
  - serveur 2 : serveur métier, ie serveur effectuant la gestion logicielle (système de gestion de base de donnée SGBD)

§ 6. Quelques avantages de cette architecture :
- dissociation BDD-SGBD : il n'y a pas de communication directe entre le client et la base. Le obligatoire par le serveur de gestion est est une garantie supplémentaire de sécurité pour la base. L'administrateur de la BDD attribue des droites aux clients (consultation, modification, ajout)
- dissociation client-BDD : sécurité, et enrichissement de la base indépendante des utilisateurs.
- dissociation client-SGBD : le logiciel de gestion peut être actualisé indépendamment du client, directement au niveau du serveur. Par ailleurs, seule une interface légère est nécessaire du côté client

Exemple 4. Quelques logiciels populaires pour manipuler des BDD :
- Oracle
- MySQL
- Microsoft SQL Server
- PostgreSQL
- SQLite

Signalons aussi qu'il est possible de manipuler des BDD avec Python, et plus précisément son module sqlite3.

§ 7. La plupart des SGBD actuels utilisent le langage SQL (*structured query language*). Les instructions principales de ce langage sont :
- **CREATE** pour créer une base de données ;
- **INSERT** pour ajouter des données à une base existante ;
- **SELECT ... FROM ... WHERE ...** pour interroger la base (*requête*)

Seules les quelques requêtes SELECT que nous allons détailler dans ce cours sont au programme.

3 Attributs et relations

Définition 3.[attributs]
★ Les titres des colonnes d'une base de données sont appelés ses *attributs*.

§ 8. On adopte les conventions suivantes :
- Les attributs sont en nombre fini ;
- Les attributs doivent être deux à deux distincts ;
- L'ordre des attributs n'a pas d'importance.

On utilisera donc

$$\mathcal{A} = \{A_1, \cdots, A_p\}$$

l'*ensemble fini* formé par ces attributs

Exemple 5. nom , sexe , naissance , lycée , ville , filière , matière , etc sont quelques attributs de l'exemple vu partie I.

Définition 4.[domaine, schéma relationnel]
★ L'ensemble des valeurs que peut prendre un attribut A_i est appelé son *domaine*, noté $dom(A_i)$ ou $\mathcal{D}(A_i)$.
★ L'application $\mathcal{S}$ définie sur $\mathcal{A}$ par

$$\forall i \quad \mathcal{S} : A_i \mapsto dom(A_i)$$

qui à chaque attribut associe son domaine est appelée un *schéma relationnel*

Notation. Nous noterons abusivement

$$\mathcal{S} = (A_1, \cdots, A_p)$$

les domaines correspondants sont connus implicitement.
On écrira « $A \in \mathcal{S}$ » pour signifier que l'attribut A appartient au schéma relationnel $\mathcal{S}$.

Exemple 6. Le domaine de l'attribut « naissance » est un ensemble d'entiers. Le domaine de l'attribut « nom » est un ensemble de chaînes de caractères. Le schéma relationnel peut s'écrire :

$$\mathcal{S} = (\text{nom} : str, \text{sexe} : str, \text{naissance} : int, \text{moyenne} : float, \cdots)$$

Remarque. Le schéma relationnel est précisé lors de la création d'une BDD. Il interdit par exemple d'inscrire un nombre dans la colonne « nom ».

Définition 5.[relation, enregistrement]
★ Une *relation*, ou *table*, associée au schéma $\mathcal{S}$ est une partie finie notée $\mathcal{R}$ ou $\mathcal{R}(\mathcal{S})$ de l'ensemble

$$dom(A_1) \times dom(A_2) \times \cdots \times dom(A_n)$$

★ Un élément d'une relation $\mathcal{R}(\mathcal{S})$ est appelé un *enregistrement* ou une *valeur*.

Remarque. Dans une relation :
- Les titres des colonnes sont les attributs.
- Les lignes sont les enregistrements de la relations. On peut aussi les représenter par des n-uplets (ou tuple en anglais).

Notation. Le nombre de lignes d'une table est noté $\sharp\mathcal{R}$ ou $|\mathcal{R}|$.
Notation. Soit $t \in \mathcal{R}$ un enregistrement d'une relation $\mathcal{R}$. Étant donné un attribut A_i on note

$$t[A_i] \quad \text{ou} \quad t.A_i$$

la valeur de t correspondant à l'attribut A_i. On peut aussi étendre cette notation :

$$t[A_{i_1}, A_{i_2}, \cdots, A_{i_n}] = (t[A_{i_1}], t[A_{i_2}], \cdots, t[A_{i_n}])$$

Exemple 7.

— La table **classes** et son schéma relationnel :

classes :(id_classe :*int* , lycée :*str* , ville :*str* , filière :*str* , effectif :*int*)

— Le quintuplet t=(3, Massena , Nice , PCSI , 35) est un enregistrement de cette table.

— t[id_classe,ville]=(3,Nice)

Remarque. Une relation est un ensemble :

— Les lignes doivent être deux à deux distinctes : on veut donc éviter d'éventuelles redondances

— L'ordre des lignes n'a pas d'importance

4 identifiants (clés)

§ 9. Étant donnée une table il est commode de pouvoir repérer rapidement si deux enregistrement sont égaux ou non. On introduit pour cela les concepts d'identifiant et de clé primaire :

Définition 6.

★ Un *identifiant*, ou super-clé, d'une relation $\mathcal{R}$ est un ensemble $\mathcal{E}$ d'attributs tel que

$$\forall (t,t') \in \mathcal{R}^2 \quad t[\mathcal{E}] = t'[\mathcal{E}] \Rightarrow t = t'$$

★ Un *identifiant minimal*, ou clé-candidate, est un identifiant $\mathcal{E}$ tel qu'enlever un attribut de $\mathcal{E}$ lui fait perdre son caractère d'identifiant.

§ 10. En pratique il est intéressant dans une table d'avoir un identifiant constitué par un unique attribut.

Définition 7.

★ Une clé primaire d'une relation $\mathcal{R}$ est le choix d'un attribut A_{PK} (« Primary Key ») qui est un identifiant de la relation

Remarque. Par convention la clé primaire d'une table sera soulignée.

Exemple 8. La table **etudiants** a pour clé primaire « id_etudiant » :

etudiants : (id_etudiant , nom , sexe , année de naissance, id_classe)

de même pour les tables **classes** et **professeurs** :

classes : (id_classe , lycée , ville , filière , effectif)
professeurs : (id_prof , nom , id_classe , matière)

§ 11. L'attribut id_classes de la table **professeurs** (et aussi dans la table **etudiants**) n'est pas une clé primaire, mais sert de lien avec la table **classes**. C'est ce que l'on appelle une clé étrangère :

Définition 8.[clé étrangère]

■ Soient $\mathcal{R}(\mathcal{S})$ et $\mathcal{R}'(\mathcal{S}')$ deux relations, sur des schémas relationnels $\mathcal{S}$ et $\mathcal{S}'$.

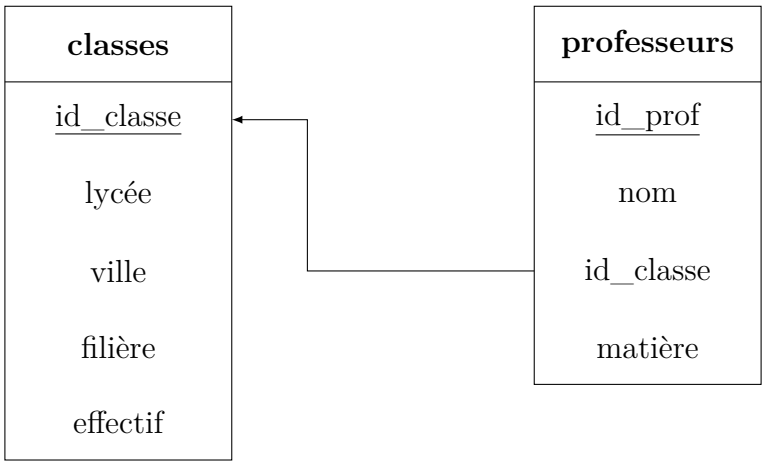
♦ Soit A'_{PK} une clé primaire sur $\mathcal{R}'$.

★ Une clé étrangère est un attribut A_{FK} (« foreign key ») sur $\mathcal{R}$ tel que

$$\mathcal{R}(A_{FK}) \subset \mathcal{R}'(A'_{PK})$$

Remarque. Ainsi, les entrées de la table $\mathcal{R}$ correspondant à l'attribut A_{FK} renvoient aux entrées de la table $\mathcal{R}'$ correspondant à l'attribut A'_{PK} , et toute valeur prise par A_{FK} doit aussi être prise par A'_{PK}.

Exemple 9. Représentation schématique des tables **classes** et **professeurs** :

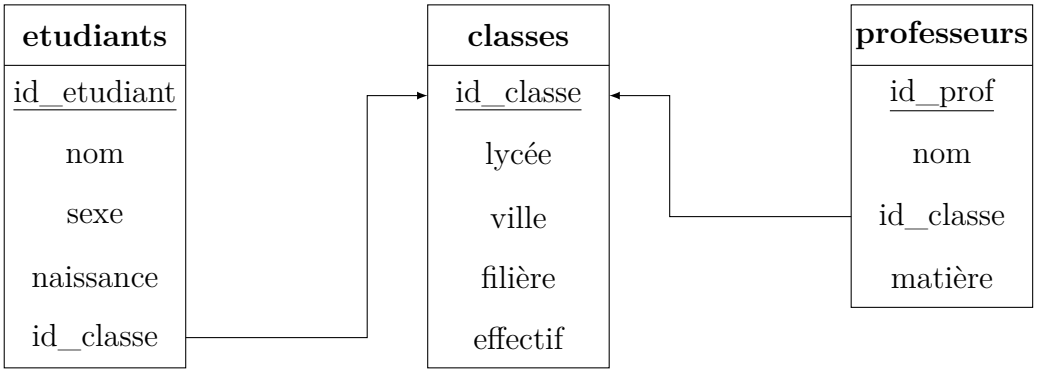


5 Bases de données

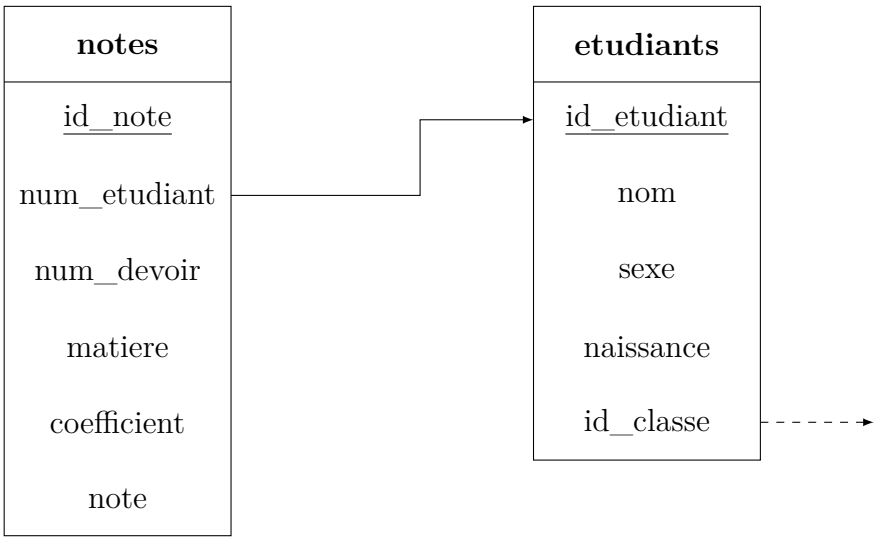
Définition 9.

★ Une base de données est constituée d'un ensemble E de relations (tables) chacune possédant
- une clé primaire ;
- des clés étrangères pointant vers les clés primaires d'autres relations de E

Exemple 10. Le schéma de BDD de la partie I :



Remarque. On peut bien entendu enrichir cette base de données. Par exemple en ajoutant une table **notes**, contenant toutes les notes obtenues par les étudiants pendant l'année, selon le schéma :
notes : (id_note :*int*, num_etudiant :*int*, num_devoir :*int*, matière :*str*, coefficient :*int*, note :$float$).
L'attribut « num_etudiant » est ici une clé étrangère qui pointe vers l'attribut « id_etudiant » de la table **etudiants** :



6 Requêtes simples

§ 12. Le langage SQL, reconnu par la plupart des SGBD, permet de créer, modifier, et consulter une base de données. Afin de rendre le code plus lisible, on convient généralement :

- d'écrire les mots-clés SQL en majuscules ;
- d'écrire les noms des tables, des attributs de la BDD en minuscule

Nous allons nous limiter à la consultation des BDD (requêtes). Pour les exemples nous nous appuierons

- sur la BDD présentée dans les parties précédentes, et ses tables **professeurs**,**classes**,**etudiants**, **notes**
- Sur une table **geographie**, qui donne des informations sur les pays du monde, selon le schéma :

geographie : (pays , continent, superficie, population , PIB)

Définition 10.[projection]
■ Soit $\mathcal{R}(\mathcal{S})$ une relation (table) et $X \subset \mathcal{S}$ un ensemble d'attributs (colonnes)
★ La projection de $\mathcal{R}$ sur X est la relation :

$$\pi_X(\mathcal{R}) = \left\{ \; t[X] \quad / \quad t \in \mathcal{R} \; \right\}$$

obtenue à partir de $\mathcal{R}$ en ne gardant que les attributs qui appartiennent à X

Remarque. Une projection consiste donc à oublier une partie des colonnes d'une table. Voici comment se traduit cette opération en SQL :

[SELECT]
★ L'instruction

```
SELECT * FROM ma_table
```

affiche tous les enregistrements de la relation « ma_table »
★ Pour afficher uniquement les attributs « attr1 » et « attr2 » de ma_table, on utilisera

```
SELECT attr1,attr2 FROM ma_table
```

Exemple 11.

— Pour afficher toutes les caractéristiques de nos étudiants, on applique la requête :
SELECT * FROM etudiants

— Pour afficher le nom des professeurs et la matière enseignée, on utilisera
SELECT nom,matiere FROM professeurs

§ 13. Il peut être utile de renommer un attribut, ou bien d'éviter des redondances dans les résultats affichés :

[AS, DISTINCT]
★ L'instruction

SELECT attr1 AS attr2 FROM ma_table

renomme l'attribut attr1 en attr2
★ L'instruction

SELECT DISTINCT attr1 FROM ma_table

affiche les valeurs de attr1 sans répétition

Exemple 12.

— Pour afficher uniquement les pays du monde et leur population on appliquera la requête :
SELECT pays, population FROM geographie

— Il est possible d'effectuer les opérations arithmétiques usuelles lors d'une requête. Pour obtenir des densités de population on écrira :
SELECT pays, population/superficie FROM geographie

— En passant il peut être utile de renommer le nouvel attribut « population/superficie » :
SELECT pays, population/superficie AS densite FROM geographie

— Pour afficher la liste des filières (MPSI,PCSI,...) de cpge à partir de la table « classes »on utilise :
SELECT DISTINCT filiere FROM classes
Sans l'option DISTINCT une même filière apparaitrait plusieurs fois dans la réponse.

Définition 11.[sélection]
★ Soit $\mathcal{R}(\mathcal{S})$ une relation (table) et soit $\mathcal{P}$ un prédicat (énomcé vrai ou faux contenant des variables)
★ La sélection de $\mathcal{R}$ avec la condition $\mathcal{P}$ est la relation

$$\sigma_{\mathcal{P}}(\mathcal{R}) = \left\{ \ t \in \mathcal{R} \quad / \quad \mathcal{P}(t) \text{ est vrai } \right\}$$

§ 14. Concrètement une sélection consiste à ne garder dans une table que les lignes vérifiant la condition $\mathcal{P}$. Voici sa traduction en SQL :

[WHERE]
★ L'instruction

```
SELECT * FROM ma_table
    WHERE contrainte
```

effectue la sélection des enregistrements de la table « ma_table »vérifiant la condition *contrainte*.
★ On peut composer une projection et une sélection avec l'instruction

```
SELECT attr1,attr2 FROM ma_table
    WHERE contrainte
```

Remarque. En SQL, contrairement aux apparences, c'est bien l'opération de projection qui correspond à l'instruction SELECT ; la sélection correspond à l'instruction WHERE.
Exemple 13. Pour afficher les noms et classes des étudiants nés en 2000 :
SELECT nom, id_classe FROM etudiants WHERE naissance=2000

§ 15. Voici quelques tests utilisables avec la condition WHERE :

=, >, <, <>, <=, >=,	comparaisons sur des nombres ou des chaînes de caractères
IS NULL	teste si la valeur n'est pas attribuée
IN (A1,A2,...)	teste l'appartenance à une liste
BETWEEN a AND b	teste l'appartenance à un intervalle (nombre ou chaîne)
LIKE '_1'	compare à une chaîne de caractères, _ représente un caractère quelconque
LIKE '%1'	même chose, mais % représente une chaîne quelconque
AND, OR, NOT	opérations booléennes usuelles

§ 16. Voici quelques opérations possibles sur les attributs :

+ * - /	Opérations arithmétiques usuelles
CHAR_LENGTH(ch)	longueur de la chaîne de caractères ch
LOWER(ch)	met la chaîne ch en minuscule
UPPER(ch	met en majuscule
ch1 || ch2	concaténation de deux chaînes

Exemple 14.
— Pour afficher la liste des pays du monde dont le nom contient 5 lettres :
SELECT pays FROM geographie WHERE CHAR_LENGTH(pays)=5
— Pour afficher la liste des pays du monde dont la population (comptée en millions d'habitants) est comprise entre 10 et 20 :
SELECT pays FROM geographie WHERE population BETWEEN 10 AND 20

7 Opérations et jointures

§ 17. Certaines informations ne peuvent être obtenues qu'en consultant plusieurs tables. Les requêtes correspondantes sont inspirées des opérations ensemblistes usuelles :
— intersection, union, différence pour des tables ayant le même schéma relationnel ;
— produit cartésien et jointures pour des tables de schémas distincts

Définition 12.[intersection, union, différence]
★ Soient $\mathcal{R}(\mathcal{S})$ et $\mathcal{R}'(\mathcal{S})$ des relations de même schéma relationnel $\mathcal{S}$.
★ On définit alors les relations $\mathcal{R} \cap \mathcal{R}'$, $\mathcal{R} \cup \mathcal{R}'$ et $\mathcal{R} \setminus \mathcal{R}'$:
— $\mathcal{R} \cap \mathcal{R}'$ est l'ensemble des enregistrements communs à $\mathcal{R}$ et $\mathcal{R}'$
— $\mathcal{R} \cup \mathcal{R}'$ est l'ensemble des enregistrements se trouvant dans $\mathcal{R}$ ou dans $\mathcal{R}'$ (ou inclusif)
— $\mathcal{R} \setminus \mathcal{R}'$ est l'ensemble des enregistrements se trouvant dans $\mathcal{R}$ mais pas dans $\mathcal{R}'$

§ 18. Voici la traduction de ces opérations en SQL :

[INTERSECT, UNION, EXCEPT]
■ Soient deux relations ma_table1 et ma_table2 de même schéma relationnel.
★ L'intersection de ces tables est donnée par

```
SELECT * FROM ma_table1
INTERSECT
SELECT * FROM ma_table2
```

★ L'union de ces tables est donnée par

```
SELECT * FROM ma_table1
UNION
SELECT * FROM ma_table2
```

★ La différence de ces tables est donnée par

```
SELECT * FROM ma_table1
EXCEPT
SELECT * FROM ma_table2
```

Exemple 15. Imaginons deux boutiques (1 et 2) ayant chacune une table qui recense leurs clients, selon un schéma identique :
clients1 : (nom, date,achat,depense) et clients2 : (nom, date,achat,depense)
— pour obtenir les noms des clients ayant fait un achat dans les deux boutiques :
SELECT nom FROM clients1
INTERSECT
SELECT nom FROM clients2
— pour obtenir les noms de tous ceux qui ont fait une dépense supérieure à 1000 euros dans l'une de ces boutiques :
SELECT nom FROM clients1 WHERE depense>1000
UNION
SELECT nom FROM clients2 WHERE depense>1000

Définition 13.[produit cartésien]
■ Soient $\mathcal{R}(\mathcal{S})$ et $\mathcal{R}'(\mathcal{S}')$ des relations dont les schémas sont disjoints : $\mathcal{S} \cap \mathcal{S}' = \varnothing$.
★ Le produit cartésien $\mathcal{R} \times \mathcal{R}'$ est la relation de schéma $\mathcal{S}'' = \mathcal{S} \cup \mathcal{S}'$ définie par

$$\mathcal{R} \times \mathcal{R}' = \left\{ \; t \quad / \quad t[\mathcal{S}] \in \mathcal{R} \text{ et } t[\mathcal{S}'] \in \mathcal{R}' \; \right\}$$

Remarque. Si $\mathcal{S} \cap \mathcal{S}' \neq \varnothing$, il suffit de renommer les attributs communs pour obtenir des ensemble disjoints ; le produit cartésien reste donc possible dans dans cas.

Exemple 16. produit cartésien d'une partie des tables classes et professeurs de la partie I ; l'attribut commun « id_classes » est renommé :

classes× professeurs					
classes.id_classe	lycee	effectif	id_prof	nom	professeurs.id_classe
1	Poincaré	45	1	Legrand	2
1	Poincaré	45	2	Latour	8
1	Poincaré	45	3	Dupont	3
1	Poincaré	45	4	Henri	4
2	Saint-Louis	42	1	Legrand	2
2	Saint-Louis	42	2	Latour	8
⋮					

[produit cartésien en SQL]
★ Le produit cartésien de deux tables ma_table1 et ma_table2 est donné par la requête

```
SELECT * FROM ma_table1,ma_table2
```

§ 19. Sur l'exemple précédent, on constate qu'un grand nombre de lignes de la table obtenue par produit cartésien ne sont pas réalistes. Il serait raisonnable de ne garder que les lignes vérifiant la condition

$$classes.id\_classe = professeur.id\_classe$$

Cette opération sera appelée une jointure :

Définition 14.
■ Soient $\mathcal{R}(\mathcal{S})$ et $\mathcal{R}'(\mathcal{S}')$ des relations tels que $\mathcal{S} \cap \mathcal{S}' = \varnothing$, soit $\mathcal{S}'' = \mathcal{S} \cup \mathcal{S}'$
■ Soit $\mathcal{C}$ un prédicat sur $\mathcal{S}''$
★ La jointure notée $\mathcal{R} \bowtie_{\mathcal{C}} \mathcal{R}'$ est la relation de schéma $\mathcal{S}''$ définie par

$$\mathcal{R} \bowtie_{\mathcal{C}} \mathcal{R}' = \sigma_{\mathcal{C}}(\mathcal{R} \times \mathcal{R}') = \left\{ t \quad / \quad t[\mathcal{S}] \in \mathcal{R} \text{ et } t[\mathcal{S}'] \in \mathcal{R}' \text{ et } \mathcal{C} \text{ est vrai } \right\}$$

★ Lorsque $\mathcal{C}$ est une égalité on parle de jointure symétrique (seule au programme)

[JOIN...ON]
■ Soient attr1 et attr2 des attributs respectifs de deux tables ma_table1 et ma_table2, et de même domaine (par exemple : un clé étrangère de ma_table1 et la clé primaire de ma_table2)
★ La jointure symétrique des deux tables ma_table1 et ma_table2 sous la contrainte « attr1=attr2 » est donnée par la requête

```
SELECT * FROM
    ma_table1 JOIN ma_table2 ON ma_table1.attr1=ma_table2.attr2
```

Exemple 17.
— Pour effectuer la jointure des tables classes et professeurs en ne gardant que les valeurs communes de l'attribut « id_classe », on effectuera la requête :

```
SELECT * FROM classes
        JOIN professeurs ON classes.id_classe=professeurs.id_classe
```

— On peut bien sur effectuer des projections ou sélections supplémentaires. Pour obtenir la liste des professeurs du lycée Dumont d'Urville

```
SELECT nom FROM classes
    JOIN professeurs ON classes.id_classe=professeurs.id_classe
    WHERE lycee="Dumont d'Urville"
```

8 Fonctions d'agrégation

§ 20. Il est possible d'effectuer des calculs utilisant plusieurs lignes d'une table : sommes, moyennes, maximum, nombre de valeurs, etc.

Définition 15.[fonction d'agrégation]
■ Soient E, F des ensembles
★ Une fonction d'agrégation $E \to F$ est une suite d'applications

$$f = (f_n : E^n \to F)_{n\in\mathbb{N}^*}$$

qui sont toutes symétriques : pour tous $(x_1, \cdots x_n) \in E^n$, la valeur de $f_n(x_1, \cdots, x_n)$ ne change pas si on permute les éléments $x_1, \cdots, x_n$.

Exemple 18. Voici quelques fonctions d'agrégation reconnues par SQL :

1. $COUNT(x_1, \cdots, x_n) = n$ (nombre d'éléments)
2. $SUM(x_1, \cdots, x_n) = x_1 + x_2 + \cdots + x_n$ (somme des éléments)
3. $AVG(x_1, \cdots, x_n) = \frac{x_1+x_2+\cdots+x_n}{n}$ (moyenne)
4. $MIN(x_1, \cdots, x_n)$ (minimum)
5. $MAX(x_1, \cdots, x_n)$ (maximum)

[agrégation et requête]
■ Soit « attr » un attribut d'une table « ma_table » et f une fonction d'agrégation définie sur dom(attr)
★ La valeur de f sur l'ensemble des enregistrements de la table est obtenue par :

```
SELECT f(attr) FROM ma_table
```

Exemple 19.
— Pour calculer la population moyenne des pays du monde :
SELECT AVG(population) FROM geoographie
— Pour obtenir le nombre total d'élèves en cpge en France (exemple partie I) :
SELECT SUM(effectif) FROM classes

§ 21. Il peut être utile
— d'appliquer une fonction d'agrégation à une partie seulement des enregistrements
— et/ou d'effectuer une sélection sur les résultats à afficher

C'est possible en SQL avec les clauses GROUP BY et HAVING :

[GROUP BY, HAVING]
■ Soient attr1 et attr2 des attributs d'une table « ma_table » et f une fonction d'agrégation définie sur dom(attr1)
★ On obtient la table des valeurs de f, regroupées par valeurs de attr2 identiques, par la requête :

```
SELECT f(attr1) FROM ma_table
    GROUP BY attr2
```

★ Pour n'afficher que les résultats vérifions une condition *contrainte* :

```
SELECT f(attr1) FROM ma_table
    GROUP BY attr2
    HAVING contrainte
```

Exemple 20.
— Pour afficher la liste des filières de CPGE, et le nombre d'élèves dans chacune de ces filières :
SELECT filiere, SUM(effectif) FROM classes
GROUP BY filiere

— Pour afficher uniquement les filières ayant au moins 1000 élèves :
SELECT filiere, SUM(effectif) AS nombre FROM classes
GROUP BY filiere
HAVING nombre>=1000

Remarque. La clause WHERE permet de sélectionner avant d'appliquer une fonction d'agrégation. La clause HAVING s'applique après l'agrégation

9 Options supplémentaires

§ 22. Théoriquement l'ordre des enregistrements d'une table n'a pas d'importance. Cependant en pratique il peut être utile d'utiliser un ordre particulier ou de n'afficher qu'une partie des résultats

[ORDER BY ... ASC/DESC]
■ Soit *attr* un attribut d'une table ma_table.
★ On affiche les lignes de cette table par valeurs croissantes de *attr* avec la requête :

```
SELECT * FROM ma_table
    ORDER BY attr ASC
```

★ On affiche les lignes de cette table par valeurs décroissantes de *attr* avec la requête :

```
SELECT * FROM ma_table
    ORDER BY attr DESC
```

Exemple 21.
— Pour obtenir la liste des classes de cpge classées par effectif croissant :
SELECT id_classe,lycée,filière,effectif FROM classes ORDER BY effectif ASC
— Pour obtenir la liste des pays du monde par PIB décroissant :
SELECT pays, pib FROM demographie ORDER BY pib DESC

[LIMIT, OFFSET]

■ Soit une table ma_table et deux entiers n et p.

★ Pour n'afficher que les n premiers enregistrements de ma_table on utilise la requête :

```
SELECT * FROM ma_table
      LIMIT n
```

★ Pour ne pas afficher les p premiers enregistrements de ma_table on utilise la requête :

```
SELECT * FROM ma_table
      OFFSET p
```

Remarque. En pratique les clauses LIMIT et OFFSET n'ont d'intérêt que si les enregistrements ont d'abord été ordonnés avec ORDER BY.

Exemple 22. Pour afficher les 10 pays les plus peuplés du monde :

```
SELECT pays,population FROM geographie
      ORDER BY population DESC
      LIMIT 10
```

10 Résumé

§ 23. On retiendra l'ordre des requêtes en SQL :

```
SELECT attribut(s)
   FROM table1
   JOIN table2 ON ... JOIN table3 ON ...      (* jointures *)
   WHERE condition      (* sélection *)
   GROUP BY attribut(s)      (* agrégation *)
   HAVING condition      (*séléction post-agrégation *)
   ORDER BY attribut(s) (ASC | DESC)      (* ordonnancement *)
   LIMIT n      (* limitation du nombre de résultats affichés à n *)
   OFFSET p      (* ignorer les p premiers résultats *)
```

§ 24. Pour s'entraîner :

— TP 11 : utilisation du logiciel SQLiteStudio (https ://sqlitestudio.pl/) et d'une base de donnée sur les communes de France

— documentation et exemples supplémentaires sur SQL : voir le site https ://sql.sh/ et en particulier le cours en pdf : https ://sql.sh/ressources/cours-sql-sh-.pdf