Requêtes sur une table

I Vocabulaire des bases de données

⋆ Étant donnés des ensembles E_1 , E_2 , .. E_n, on appelle **table** (ou **relation**) tout sous ensemble de $E_1 \times E_2 \times .. \times E_n$. Ainsi, une **table** est un ensemble de n-uplets. Une **base de données relationnelle** peut être vue comme un ensemble de tables.

⋆ Un **enregistrement** (ou une **ligne**) de la table est l'un de ces n-uplets.

⋆ Un **attribut** (ou une **colonne**) de la table est le nom qui désigne les éléments d'un même ensemble E_i permettant de constituer la table

⋆ Le **domaine** d'un attribut est le type d'éléments de l'ensemble qu'il désigne.

Exemple(s) :

(*I.1*) On considère une table, nommée `tableau`, représentant la classification des éléments de Mendeleiev dont le schéma est le suivant :

nom	numéro_atomique	symbole	colonne	ligne	bloc
Hydrogène	1	H	1	1	s
Hélium	2	He	18	1	s
Lithium	3	Li	1	2	s
Béryllium	4	Be	2(IIA)	2	s

⋆ Un enregistrement de cette table est (Hydrogène,1,H,1,1,s)

⋆ Les attributs sont `nom`, `numéro_atomique`, `symbole`, `colonne`, `ligne` et `blocs`.

⋆ Le domaine des attributs `nom`, `symbole`, `colonne` et `blocs` dont des chaînes de caractères. Le domaine des attributs `numéro_atomique` et `ligne` sont des entiers.

Remarque(s) :

(*I.2*) Le programme de PSI limite les domaines aux entiers, flottants et chaînes de caractères. Il existe d'autres domaines comme les dates mais qui ne sont pas au programme ; on pourra quand même considérer des dates comme des chaînes de caractères sous la forme `'aaaa/mm/jj'` et des heures sous la forme `'hh:mm:ss'` (dans cet ordre de façon à pouvoir les classer chronologiquement avec l'ordre lexicographique).

(*I.3*) La structure d'une table (nombre de colonnes, attributs, domaines des attributs,...) est définie lors de la création d'une table. Il est possible de modifier une table mais les commandes permettant de créer ou de modifier une table ne sont pas au programme ; on se contentera de présenter certaines commandes permettant d'utiliser une table déjà existante.

⋆ Une **clé primaire** d'une table est un attribut (ou un ensemble d'attributs) qui permet d'identifier de façon unique un enregistrement.

Exemple(s) :

(*I.4*) Dans notre exemple, le `numéro_atomique` ou le `symbole` peuvent servir à définir une clé primaire.

(*I.5*) Dans un hôpital, si on considère deux tables : la première `patients` qui contient les identités des patients et la seconde `operations` qui contient certaines informations sur les opérations effectuées que l'on peut schématiser de la façon suivante

prenom	nom	numero_secu
Jean	Dupont	1221233052555
Marie	Dupond	2221275265875

Table `patients`

id	nom	numero_secu	date_operation	chirurgien
1	Dupont	1221233052555	2022/12/05	Dr Durand
2	Dupond	2221275265875	2022/12/06	Dr Durand

Table `operations`

— Dans cet exemple, le `numero_secu` peut être une clé primaire dans la table `patients` mais pas le coupe `(prenom,nom)` (on pourrait avoir deux homonymes)

— Par contre le `numero_secu` ne peut pas être une clé primaire dans la table `operations` (un même patient pourrait subir deux opérations donc apparaître dans deux enregistrements différents). Comme il n'y a pas de clé primaire que l'on peut définir à partir des attributs, on peut rajouter une colonne `id` qui devient une clé primaire (mais qui n'a pas de signification).

Remarque(s) :

(*I.6*) Là encore, la définition d'une clé primaire doit être faite au moment de la création de la table.

(*I.7*) Une table est un ensemble d'enregistrements donc ils ne sont pas ordonnés (même si on est obligé de les ordonner pour représenter une table), il est donc indispensable que les enregistrements d'une tables soient deux à deux distincts ; une clé primaire permet donc de repérer de façon unique un enregistrement de la table.

II Bases du langage SQL

L'intérêt des bases de données est de pouvoir rechercher des informations qui sont contenues dans les tables. Pour cela, on utilise le langage SQL (*Structured Query Langage*) qui permet de faire de telle recherches par l'intermédiaire de **requêtes**.

1. Projection

La commande `SELECT` permet de sélectionner certains des attributs d'une table :

`SELECT` *attribut 1,...,attribut n* `FROM` *table*

Cette commande crée en fait une nouvelle table extraite de la table initiale qui ne comporte plus que les colonnes `attribut 1,...,attribut n` qui doivent donc être des attributs de la table initiale.

Exemple(s) :

(*II.8*) `SELECT * FROM tableau` va renvoyer tous les enregistrements de la table.

(*II.9*) `SELECT nom,symbole FROM tableau` va renvoyer la « liste » de tous les noms des éléments chimiques et de leur symbole.

(*II.10*) Même si les enregistrements de la table initiale `tableau` sont deux à deux distincts, il se peut que la table produite par `SELECT` comporte des lignes identiques : `SELECT ligne,bloc FROM tableau` produira par exemple plusieurs lignes identiques. Pour éliminer ces répétitions, on peut rajouter l'option `DISTINCT` : `SELECT DISTINCT ligne,bloc FROM tableau`

2. Sélection

Pour ne sélectionner que certains enregistrements (ie certaines lignes) on effectue une sélection avec une condition :

`SELECT (DISTINCT)` *attribut 1,...,attribut n* `FROM` *table* `WHERE` *condition*

Cette commande crée une table extraite de la table initiale qui ne comporte que les colonnes `attribut 1,...,attribut n` et les lignes qui satisfont la condition donnée. Cette *condition* doit donc pouvoir être évaluée afin de produire un booléen.

Les tests réalisables (au programme) :

`=` tester l'égalité de deux attributs (y compris des chaînes de caractères).
`<>` symbole « différent de ».
`+`, `-`, `*`, `/` opérations sur les entiers/flottants.
`<=`, `<`, `>=`, `>` comparaison de deux attributs (y compris sur les chaînes de caractères avec l'ordre lexicographique).
`AND`, `OR`, `NOT` opérateurs logiques permettant de construire des conditions plus complexes.

Exemple(s) :

(*II.11*) `SELECT nom,symbole FROM tableau WHERE ligne=2` va renvoyer la liste de tous les noms des éléments chimiques et de leur symbole, qui sont sur la deuxième ligne.

(*II.12*) `SELECT symbole FROM tableau WHERE numero_atomique < 20 AND symbole <= 'D'` renvoie les symboles des éléments de numéro atomique < 20 qui commencent par les lettres A, B, C ou D.

3. Renommage

Lorsque le nom d'un attribut ou d'une table est compliqué, on peut le renommer avec un alias :
`SELECT symbole, numero_atomique AS n FROM tableau WHERE n < 20` renvoie les symboles et numéros atomiques des éléments de numéro atomique < 20.

Remarque(s) :

(*II.13*) La commande `AS` peut même être omise : `SELECT symbole, numero_atomique n FROM tableau WHERE n < 20` produit le même effet.

(*II.14*) On pourra être amené à renommer une table dans le cas où on utilisera plusieurs tables (cf cours suivant) dont les attributs portent le même nom.

4. Options d'affichage

— Il est possible d'ordonner les résultats avec la commande `ORDER BY` *attribut* : les résultats seront affichés dans l'ordre croissant de l'attribut (ordre lexicographique si c'est une chaîne de caractères)
— On peut limiter le nombre de résultats : `LIMIT n` ne renvoie que les n premiers résultats et `OFFSET m` (à utiliser obligatoirement avec `LIMIT`) renvoie les résultats à partir du $m+1^{\text{ème}}$: `LIMIT n OFFSET m` renvoie les résultats du $m+1^{\text{ème}}$ au $m+n^{\text{ème}}$. Le résultat produit peut être incertain car l'ordre des enregistrements dépend de la façon dont a été créée la table.

Exemple(s) :

(*II.15*) `SELECT symbole,numéro_atomique FROM tableau WHERE ligne = 3 ORDER BY bloc` renvoie la table contenant les symboles et les numéros atomiques des éléments de la ligne 3 en les triant par bloc (dans l'ordre alphabétique).

(*II.16*) `SELECT symbole FROM tableau LIMIT 4 OFFSET 1` renvoie les symboles des éléments « compris » entre 2 et 2+4=6 (dans l'ordre où ils ont été rentrés au moment de la création de la table).

III Fonctions d'agrégation

Les **fonctions d'agrégation** permettent d'exécuter une opération sur un groupe d'enregistrements (à priori sur tous les enregistrements sélectionnés par `WHERE`). Il existe différentes fonctions d'agrégation, les suivantes sont les seules qui sont au programme :

nom	Action
`COUNT()`	Compte le nombre d'enregistrements d'un attribut
`AVG()`	Calcule la moyenne d'un attribut d'un ensemble d'enregistrements de type numérique
`MIN()`	Calcule le minimum d'un attribut d'un ensemble d'enregistrements de type numérique
`MAX()`	Calcule le maximum d'un attribut d'un ensemble d'enregistrements de type numérique
`SUM()`	Calcule la somme d'un attribut d'un ensemble d'enregistrements de type numérique

1. Utilisations simples

Pour appliquer une fonction d'agrégation `f` à un tous les attributs `a` d'une table, on exécute :

`SELECT` f(a) `FROM` *table*

On peut aussi appliquer `f` aux enregistrements d'une table sélectionnés par une condition : la fonction `f` est alors appliquée aux attributs `a` qui satisfont la condition de sélection

`SELECT` f(a) `FROM` *table* `WHERE` *condition*

Exemple(s) :

(*III.17*) La requête `SELECT COUNT(*) FROM tableau` renvoie le nombre d'enregistrements de la table, ie le nombre de lignes.

(*III.18*) La requête `SELECT COUNT(ligne) FROM tableau` renvoie le nombre de valeurs de la colonne `ligne` (donc 118) alors que `SELECT COUNT(DISTINCT ligne) FROM tableau` renvoie le nombre de valeurs différentes de la colonne `ligne` (donc 7).

(*III.19*) `SELECT SUM(numéro_atomique) FROM tableau WHERE ligne = 2` renvoie 52, la somme des numéros atomiques des éléments de la deuxième ligne.

2. Regroupements

Il est possible d'appliquer une fonction d'agrégation à tous les éléments de la table mais en les regroupant selon une condition : la requête suivante va appliquer la fonction d'agrégation `f` aux attributs `a` sur chaque « paquet » constitué par les valeurs de l'attribut `b`. On va donc obtenir autant de résultats qu'il y a de tels paquets. La commande `GROUP BY` ne peut pas être utilisée sans fonction d'agrégation.

`SELECT` f(a) `FROM` *table* `GROUP BY` b

Exemple(s) :

(*III.20*) `SELECT AVG(numéro_atomique) FROM tableau GROUP BY ligne` renvoie les 7 moyennes des numéros atomiques des 7 lignes.

Il est possible de commencer par limiter les enregistrements avec `WHERE` avant l'agrégation : dans la requête suivante, on commence par sélectionner les enregistrements qui vérifient *cond* avant de les regrouper en paquets selon les valeurs de `b` puis on applique la fonction `f` aux attributs `a` de chaque paquet.

`SELECT` f(a) `FROM` *table* `WHERE` *cond* `GROUP BY` b

Exemple(s) :

(*III.21*) `SELECT MAX(numéro_atomique AS n),MIN(n) FROM tableau WHERE ligne > 3 GROUP BY bloc` renvoie les couples plus grand/plus petit numéro atomique pour chacun des 4 blocs, en ne prenant en compte que les lignes à partir de la $4^{\text{ème}}$.

Il est enfin possible de ne sélectionner que les enregistrements dont les résultats de la fonction d'agrégation vérifie une condition ; c'est un filtrage après application de la fonction d'agrégation : la requête suivante commence par sélectionner les enregistrements qui vérifient *cond1* avant l'application de la fonction `f`, puis les regroupe selon les valeurs de l'attribut `b` et applique `f` à chacun des paquets, et termine par sélectionner seulement les paquets pour lesquels *cond2* est satisfaite (*cond2* est donc vérifiée après la création des paquets, elle peut donc elle aussi être définie par l'intermédiaire d'une fonction d'agrégation).

`SELECT` f(a) `FROM` *table* `WHERE` *cond1* `GROUP BY` b `HAVING` *cond2*

Exemple(s) :

(*III.22*) `SELECT SUM(numéro_atomique) FROM tableau GROUP BY ligne HAVING COUNT(*) > 8` renvoie la somme des numéros atomiques de chaque ligne qui possède au moins 9 éléments (donc à partir de la ligne 4).

(*III.23*) `SELECT bloc,COUNT(DISTINCT colonne),MAX(numéro_atomique) AS M FROM tableau WHERE ligne > 2 GROUP BY bloc HAVING M < 100` renvoie le bloc, le nombre de colonnes distinctes et le plus grand numéro atomique des enregistrements regroupés par bloc, à partir de la $3^{\text{ème}}$ ligne et dont le plus grand numéro atomique reste < 100.

(*III.24*) `SELECT bloc,COUNT(DISTINCT colonne) FROM tableau WHERE ligne > 2 GROUP BY bloc HAVING MAX(numéro_atomique) < 100` produit le même résultat sans renvoyer la valeur du plus grand numéro atomique.

Remarque(s) :

(*III.25*) **Ne pas confondre `WHERE` et `HAVING` :**

- `WHERE` s'utilise avec ou sans agrégation et, dans le cas de l'utilisation d'une fonction d'agrégation `f`, la condition est vérifiée avant l'application de `f` (et l'éventuelle création des paquets par `GROUP BY`).
- `HAVING` ne peut s'utiliser qu'après `GROUP BY`, avec une fonction d'agrégation `f`, et la condition est vérifiée après la création des paquets et l'application de `f`.