VI

Bases de données relationnelles - langage SQL

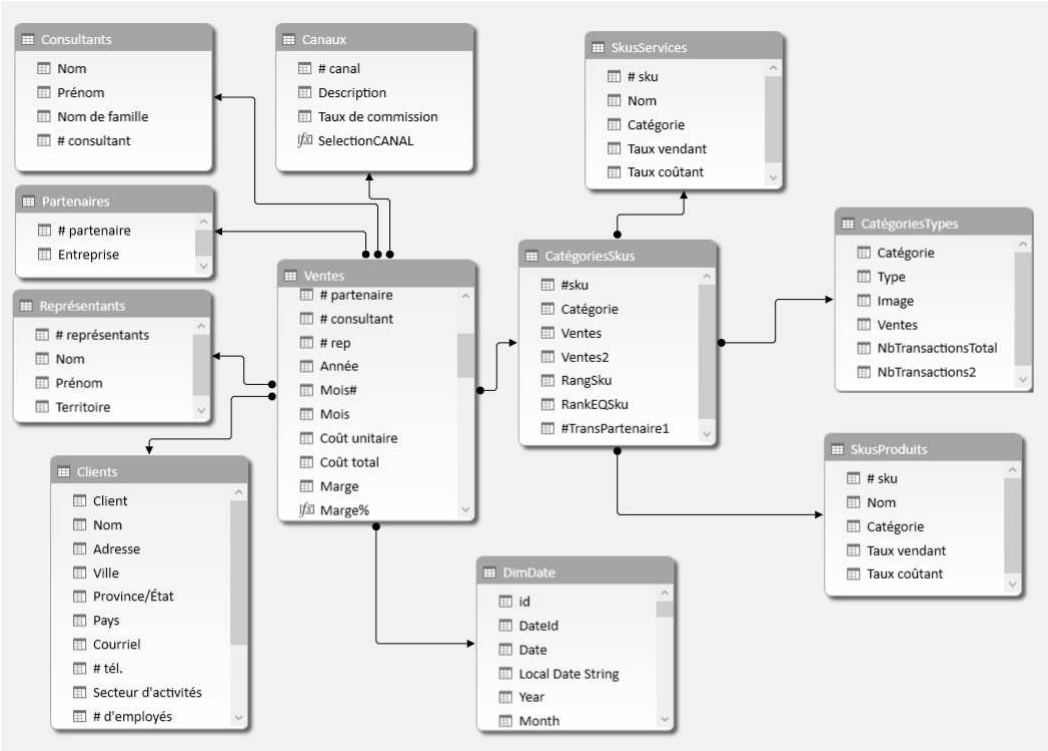


FIGURE VI.1 – Représentation schématique d'une base de données

PLAN DU CHAPITRE

1 Bases de données - Tables

1.1 Définition

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

Définition 1-1: Enregistrement

Un enregistrement de la table est l'un de ces **n-uplets**. Un enregistrement s'appelle aussi une **entité**. Si la table est bien conçue, il ne peut pas y avoir de redondance d'enregistrement.

Définition 1-2: Attribut

Un attribut de la table est le nom qui désigne les éléments d'un même sous-ensemble E_i permettant de constituer la table. Là-encore, un attribut ne peut être redondant dans une base de données.

Définition 1-3: Domaine

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

De manière très schématique, il est donc possible de représenter une table comme un tableau dans lequel chaque **colonne** correspond à un sous-ensemble E_i et chaque **ligne** a une entité :

Attribut 1 (ex. Dom(Attribut 1): int)	Attribut 2 ((ex. Dom(Attribut 2): str))	...	Attribut n ((ex. Dom(Attribut n): float))
1	bleu	...	123.562
...	...	...	...

Plus concrètement maintenant, on peut considérer par exemple une table `tableau` représentant la classification des éléments de Mendeleiev dont le schéma élémentaire 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

Table VI.1 - La table **tableau**

Un enregistrement ou **entité** de cette table `tableau` est par exemple :

$$(\text{Lithium}, \mathbf{3}, \mathbf{Li}, \mathbf{1}, \mathbf{2}, \mathbf{s})$$

Les attributs sont `nom, numéro_atomique`, `symbole`, `colonne`, `ligne` et `blocs` dont les domaines sont tous des chaînes de caractères sauf le numéro_atomique et la ligne qui sont des entiers (les numéros des colonnes ne sont pas que des entiers).

1.2 Principe de recherche et garantie d'unicité des enregistrements : la "clé primaire"

Une base de données bien construite implique l'absence de redondance d'enregistrement ; pour s'assurer de cela, chaque enregistrement dispose d'un attribut ou d'un ensemble d'attributs qui l'identifie de manière unique ; on appelle cela une **clé primaire**.
En l'absence de redondance d'enregistrement, nous pourrions très simplement décréter que la clé est par exemple **l'ensemble de tous les attributs** ; mais ce ne serait vraiment pas pratique de devoir tous les citer au moment de la formulation d'une requête pour extraire un enregistrement.

On a plutôt intérêt à choisir un attribut qui identifie de manière unique chaque enregistrement si la base de données le contient déjà, ou bien introduire par exemple avec un simple numéro d'identification.

Définition 1-4: CLÉ PRIMAIRE

On appelle **clé primaire** d'une table un (ou plusieurs) attribut(s) qui permet d'identifier de façon unique un enregistrement.

Dans notre exemple, le `nom`, le `numéro_atomique`, le `symbole` sont des clés primaires. Dans le cas où on ne peut pas utiliser un des attributs comme clé primaire, on rajoute un attribut (souvent appelé *id*) qui est un simple numéro (on numérote tout simplement les lignes).
Il est assez fréquent que les clés primaires soient soulignées dans la représentation concrète d'une table :

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

TABLE VI.2 - Les clés primaires de la table **tableau**

1.3 Modèle Entité-Association - clés étrangères

L'un des gros avantages des bases de données est qu'il n'est pas nécessaire d'organiser les informations dans un "conteneur structuré" unique comme ce serait par exemple le cas avec les données d'un tableur.
En contrepartie, si les données sont séparées, par exemple **sur plusieurs tables**, il est absolument nécessaire de les lier de manière intelligente pour concevoir une base de données.

Le modèle entités-associations constitue l'une des premières et des plus courantes techniques de construction d'une base de données. Ce modèle introduit par Peter Chen en 1976 assure une description facilement compréhensible d'une relation à l'aide de concepts un peu abstraits appelés "entités" et "associations", mais qui permettent, une fois mis en œvre, **de lier facilement des tables entre-elles** pour former une base de données complexe.

Ce modèle universel et reconnu assure notamment une totale indépendance entre les bases de données créées selon ce schéma et le logiciel exploité pour les consulter.

a - Entités et associations

La structure du modèle entités-associations s'appuie sur deux concepts essentiels :
- une **entité** et un **type-entité** :

Définition 1-5: ENTITÉ

Une entité est objet ayant une existence propre, par exemple, dans le cas de notre classification périodique, un élément chimique enregistré dans la table `tableau` et décrit par les attributs qui assurent son unicité.

Définition 1-6: TYPE ENTITÉ

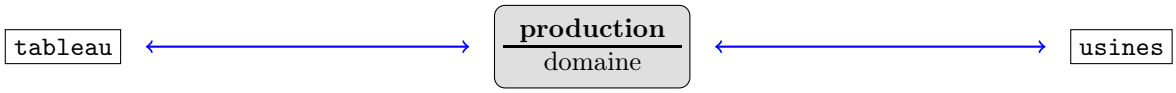
Le type-entité désigne un ensemble d'entités **de mêmes caractéristiques**, par exemple les différents enregistrements de la table `Tableau`.
Une autre table qui contiendrait par exemple les caractéristiques des usines produisant le corps pur simple associé à cet élément ; les enregistrements de cette nouvelle table constitue un autre type-entité.

tableau
nom
numéro atomique
symbole
colonne
ligne
bloc

usines
id-usine
pays
adresse
consommation énergétique
bilan carbone annuel

- une **association** est un lien entre entités, **en général de type-entités différents**.

  Par exemple, on peut établir une association dite **production** entre la table `tableau` contenant les éléments de la classification (et leur place dans le tableau de Mendeleiev) et la table `usines`.



b - Associations - cas particuliers des cardinalités 1 - 1, 1 - $\star$, et $\star$ - $\star$

Chaque entité d'une table peut apparaître **une fois, plusieurs fois, ou jamais** dans une association. Pour en rendre compte, on fait figurer sur le diagramme sagittal précédent le nombre de fois minimal et maximal que peut figurer une entité dans l'association ; on appelle cela la cardinalité ; le format est :

$$m \ - \ M \qquad \text{avec } m \in \{0,1\} \text{ le minimum, et } M \geq 1 \text{ le maximum que l'on notera } \star \text{ si M>1}$$

Les associations intéressantes au programme sont $1 \ - \ 1$, $1 \ - \ \star$ et $\star \ - \ \star$.

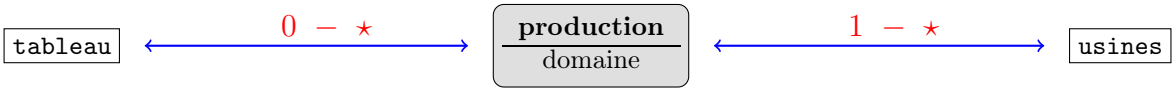
Par exemple : pour l'association `production` vue côté **tableau-production** :

- $1 \ - \ 1$: signifie que chaque élément de la classification possèderait exactement une production (et donc dans un domaine unique) ; peu probable ! ! ! !

- $1 \ - \ \star$: signifie que chaque élément de la classification possède au moins une production (donc dans au moins un domaine).

- $\star \ - \ \star$ signifie que chaque élément de la classification possède toujours plusieurs productions.

Par exemple, dans notre cas, il est évident qu'un corps purs simple d'un élément connu peut faire l'objet de production pour plusieurs domaines (médecine, armement, recherche, etc...), et à minima ne faire l'objet d'aucune production : la cardinalité de cette association est alors $0 \ - \ \star$; en outre, une usine peut n'assurer une production que dans un seul domaine, à l'inverse une usine peut produire à destination de divers domaines, et la cardinalité de l'association est alors $1 \ - \ \star$.



c - Séparation d'une association ⋆ - ⋆ ; clés étrangères

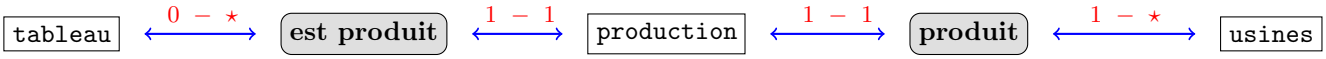
Pour l'instant, les notions que nous évoquons restent théoriques, mais le but est bien d'implémenter ces informations dans une véritable base de données. Les cardinalités maximales ⋆ des associations `tableau` ↔ `production` et `production` ↔ `usines` sont par essence indéterminées et interdisent d'établir facilement des liens entre tables (nécessaires dans une base de données complexe) :

Dans l'exemple de notre association **production**, comme chaque corps pur est produit pour **plusieurs domaines** (⋆) et chaque usine produit pour **plusieurs domaines** (⋆), il est impossible d'établir une correspondance entre **un** élément de la table `tableau` et **un** élément de la table `usines` qui produit son corps pur.

L'idée consiste à transformer ces associations `production` en une **relation à part entière** c'est à dire **une table** dans laquelle on introduit deux clés qui résultent de la concaténation des clés primaires des relations connectées ; cela revient à introduire un nouveau type-entité **production** avec par exemple :

production
id-production
nom-production
id-usine-production
domaine

Dans ces conditions, le diagramme sagittal de cette association devient :



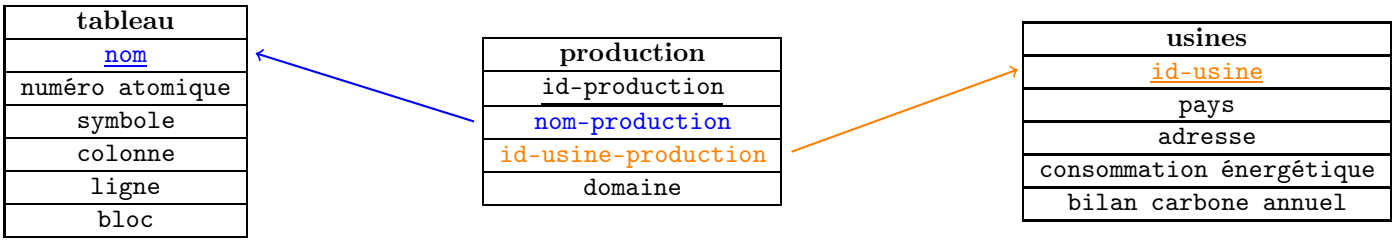
Interprétation :
- une production (identifiée par `id-production`) produit le corps pur d'un seul élément de **tableau**.
- une production (identifiée par `id-production`) n'est réalisée que par une seule usine.

On définit alors le concept de **clé étrangère** ; dans la nouvelle relation créée, on a introduit deux nouvelles clés dites **clé étrangères**, d'une part par concaténation du nom de la clé primaire `nom` de la table `tableau` et de `production` (arbitraire !), et d'autre part `production` et `id-usine` clé primaire de la table `usines`. Cela assure un lien unique, s'il existe entre chaque entité de `tableau` et une entité de `usines`.

Définition 1-7: CLÉ ÉTRANGÈRE

Une clé étrangère est un attribut d'une table construite à partir d'une association, et dont **les valeurs sont exactement celles prises par la clé primaire de la table à laquelle elle est liée.**

Il n'y a alors plus de difficulté à lier les enregistrements des tables `tableau` et `usines`, et le schéma relationnel devient :



2 Consultation des bases de données : le langage SQL

2.1 Utilité

Une fois le concept de **base de données** défini, il est indispensable de disposer d'un outil de création et consultation de celles-ci ; l'un d'entre eux, appelé **SQL** pour **S**tructured **Q**uery **L**anguage s'est aujourd'hui imposé comme un véritable standard.

Il est une application concrète de **l'algèbre relationnelle**, théorie mathématique inventée dans les années 70, proche de la théorie des ensembles, et qui définit complètement le cadre mathématique des **relations**, donc **des bases de données** ; ces notions sont hors programme, mais on en donnera quelques éléments en fin de chapitre ; on pourra en particulier constater la proximité de la syntaxe du langage SQL avec le vocabulaire de l'algèbre relationnelle.

Remarque 2-1: PROGRAMME ITC

Le programme d'ITC se limite à la manipulation des données d'une base de données **existante** en vue de **simples consultations organisées de ses contenus**.

2.2 Requêtes de base

Une requête SQL se formule en utilisant les commandes suivantes :

```
SELECT attributs                # attributs à chercher (* pour tous les attributs)
SELECT DISTINCT                 # Même chose sans les doublons
FROM tables                     # Lieu de la recherche
WHERE conditions                # Critère de sélection
ORDER BY expression             # Trier les résultats
LIMIT n                         # Limiter à n enregistrements
OFFSET n                        # Débuter à partir de n enregistrements.
```

Exemple(s) :

La requête :

```
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).

2.3 Les fonctions d'agrégation

Les **fonctions d'agrégation** permettent d'exécuter une requête sur un groupe d'enregistrement. Il existe différentes fonctions d'agrégation, les suivantes sont celles qui permettent de faire des statistiques sur une table.

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

Exemples :

- La requête :

```
SELECT COUNT(*) FROM tableau
```

renvoie le nombre d'enregistrements du tableau, c'est à dire le nombre total de lignes.

- La requête :

```
SELECT SUM(numéro_atomique) FROM tableau WHERE ligne=2
```

renvoie 52, somme des numéros atomiques des éléments de la deuxième ligne.

On peut également regrouper les attributs selon certains critères en utilisant `GROUP BY`, et rajouter des conditions sur les groupes avec `HAVING` (placé après `GROUP BY`).

Exercice de cours: **(2.3) - n° 1.** *Que renvoient les requêtes suivantes ?*

- *La requête :*

```
SELECT SUM(numéro_atomique) FROM tableau GROUP BY ligne
```

- *La requête :*

```
SELECT SUM(numéro_atomique) FROM tableau GROUP BY ligne HAVING COUNT(*) > 8
```

2.4 La fonction de filtrage spécifique IS (NULL ou NOT NULL)

Les conditions de filtrage dans la commande `WHERE` sont en général formulées à l'aide des opérateurs de comparaison `=,<,>,<=,>=`, et des opérateurs booléens `AND` et `OR`. On peut les compléter par l'opérateur `IS`, indispensable pour filtrer les résultats avec des données enregistrées sous la valeur `NULL` car cette dernière possède une valeur inconnue, et ne peut donc pas être comparée.

Exemple :

La requête :

```
SELECT SUM(numéro_atomique) FROM tableau WHERE colonne IS NOT NULL GROUP BY ligne
```

renvoie la somme des numéros atomiques de chaque ligne pour les éléments dont la colonne n'est pas à la valeur `NULL`.

(**NB :** ce filtrage est donné en guise d'illustration simple car aucun des champs de cette base de données n'est à la valeur `NULL`)

2.5 Les jointures

L'intérêt principal des bases de données est de pouvoir manipuler plusieurs tables en même temps et de pouvoir en croiser les résultats. On utilisera les deux autres tables `découverte` et `grandeurs` dont les premières lignes sont les suivantes :

nom	pays	symbole	date
Henry Cavendish	Grande-Bretagne	H	1766
Jules Janssen	Grande-Bretagne	He	1895
Joseph Norman Lockyer	Grande-Bretagne	He	1895
Johan August Arfwedson	Suède	Li	1817

TABLE VI.1 – Table `découverte`

symbole	remplissage	masse_atomique	temp_fusion	temp_ébullition
H	1	1,00794		
He	2	4,002602		
Li	2|1	6,94100	180,5	1342
Be	2|2	9,012182	1287	2471

TABLE VI.2 – Table `grandeurs`

La jointure de deux tables se fait par l'utilisation de la commande :

```
table1 JOIN table2 ON condition
```

la condition permettant de faire la liaison entre les deux tables.

Exemple(s) :

- On considère la requête suivante :

```
SELECT DISTINCT pays FROM découverte JOIN tableau ON découverte.symbole = tableau.symbole WHERE ligne = 2
```

La jointure se fait par l'identification `découverte.symbole=tableau.symbole` ; l'utilisation des préfixes `découverte` et `tableau` est nécessaire car `symbole` est un attribut dans les deux tables. Par contre comme `ligne` et `pays` ne sont pas des attributs des deux tables, le préfixe est inutile.

Exercice de cours: **(2.5) - n° 2.** *Que renvoie cette requête ?*

- Pour alléger les notations des préfixes, on peut utiliser des alias :

```
SELECT DISTINCT pays FROM découverte d JOIN tableau t ON d.symbole = t.symbole WHERE ligne = 2
```

a le même effet ; on utilise l'alias `d` pour `découverte` et `t` pour `tableau`.

- On peut alors exécuter des jointures multiples :

```
SELECT DISTINCT d.nom FROM (découverte d JOIN tableau t ON d.symbole = t.symbole)
JOIN grandeurs g ON d.symbole=g.symbole WHERE numéro_atomique<50 AND temp_fusion <0
```

permet de récupérer les noms des personnes qui ont découvert les éléments dont le numéro atomique est < 50 et qui ne sont pas solides à une température nulle. Le préfixe `d` devant `nom` est nécessaire pour distinguer l'attribut `nom` de la table `découverte` (le nom de la personne qui a découvert l'élément) de l'attribut nom de la table `tableau` (le nom de l'élément).

Exercice de cours: **(2.5) - n° 3.** *Ecrire une requête renvoyant le nom, le symbole, la date de découverte, et la température de fusion de tous les éléments découverts avant la moitié du XVIII^{ième} siècle et dont la température de fusion est supérieure ou égale à 100°C.*

Exercice de cours: **(2.5) - n° 4.** *Ecrire une requête renvoyant le nom, le symbole, et la date de découverte de l'élément appartenant au bloc d et qui a été découvert le plus tard dans le courant du XIX$^{ième\ siècle}$.*

2.6 L'autojointure

Il est également possible de joindre une table à elle même. Cet approche est particulièrement adaptée lorsque des enregistrements de la table possèdent des liens entre eux et que l'on souhaite les associer dans une sélection ; par exemple :

- extraire d'une table certaines caractéristiques d'enfants d'une même fratrie à partir d'un critère de sélection
- plus proche de notre base de données `classification périodique` : extraire des caractéristiques d'éléments qui ont un critère commun par exemple le même découvreur

La syntaxe typique de l'autojointure est :

```
....table t1 INNER JOIN table t2 ON t1.element=t2.element WHERE.....
```

Remarque 2-2: SYNTAXE DE L'AUTOJOITURE

- le mot clé `JOIN` seul fonctionne aussi pour l'autojointure.
- les alias sont ici indispensables pour que SQL distingue virtuellement deux tables pour effectuer la jointure.

Exercice de cours: **(2.6) - n° 5.** *Ecrire une requête renvoyant le symbole des éléments ayant été découverts par exactement 2 chimistes différents. On renverra également le nom de ces deux chimistes.*

3 L'algèbre relationnelle

3.1 Opérations ensemblistes

Les commandes UNION, INTERSECT et EXCEPT permettent d'exécuter les opérations de réunion, d'intersection ou de différence sur les tables que l'on crée :

```
SELECT * FROM t1 UNION SELECT * FROM t2        # enregistrements présents dans t1 ou t2 (sans répétition)
SELECT * FROM t1 INTERSECT SELECT * FROM t2    # enregistrements présents dans t1 et t2
```

Pour pouvoir utiliser ces opérations, les tables doivent avoir le même schéma relationnel (mêmes attributs)

EXEMPLE(S) :

- SELECT * FROM tableau WHERE ligne = 2 UNION SELECT * FROM tableau WHERE symbole < 'D' renvoie la sous-table de tableau des éléments qui sont sur la deuxième ligne ou dont le symbole commence par une lettre avant D.
- SELECT * FROM tableau WHERE ligne = 2 EXCEPT SELECT * FROM tableau WHERE colonne =1 renvoie la sous-table des éléments de la ligne 2 sauf celui de la colonne 1.

Le complémentaire s'obtient en utilisant `NOT IN` : la requête `SELECT nom FROM tableau WHERE nom NOT IN (SELECT nom FROM tableau WHERE bloc = 'f')` renvoie le nom des éléments qui ne sont pas dans le bloc f.

3.2 Opérations de l'algèbre relationnelle

■ **Projection** :
On note $\pi_{A_1,A_2,\ldots}(R)$ la projection d'une relation R suivant les attributs $A_1, \ldots, A_p$: cela revient à ne conserver que certain attributs de la table sans répétition. La requête SQL correspondante est :

```
SELECT DISTINCT A1,... A2 FROM table
```

■ **Sélection** :
On note $\sigma_E(R)$ la sélection d'une relation R suivant une condition logique E. La requête SQL correspondante est :

```
SELECT * FROM table WHERE condition E
```

■ **Renommage**
On note $\rho_{a \leftarrow b}(R)$ le renommage de l'attribut a en b dans la relation R. La requête SQL correspondante est :

```
ALTER TABLE table RENAME COLUMN a TO b
```

Cette opération n'est pas possible dans SQLite, on peut seulement renommer une table entière par `ALTER TABLE table1 RENAME TO table2`

■ **Jointure**
On note $R_1 \bowtie_E R_2$, la jointure (symétrique) des relations R_1 et R_2 avec la condition E. La requête SQL correspondante est :

```
SELECT * FROM table1 JOIN table2 ON cond E
```

■ **Produit cartésien**
On note $R_1 \times R_2$ le produit cartésien des relations R_1 et R_2 : on obtient une table qui contient les colonnes des deux tables (avec éventuelles répétitions, les colonnes de la deuxième table à droite). La requête SQL correspondante est :

```
SELECT * FROM table1,table2
```

Une jointure est en fait obtenue en réalisant un produit cartésien puis une sélection : $R_1 \bowtie_E R_2 = \sigma_E(R_1 \times R_2)$.

■ **Division cartésienne**
La division cartésienne de R_1 par $R_2 \subset R_1$ (la deuxiéme table est donc une sous-table de la première) est la relation $R = R_1 \div R_2$, c'est la relation contenant les attributs de R_1 qui ne sont pas dans R_2 : elle est caractérisée par $x \in R_1 \div R_2$ si et seulement si $\forall y \in R_2 (x, y) \in R_1$. Il n'y a pas de requête SQL simple permettant de reproduire cette opération.

TABLE VI.3 – R_1

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2
a_2	b_2	c_3	d_3
a_3	b_3	c_1	d_1
a_3	b_3	c_2	d_2

TABLE VI.4 – R_2

C	D
c_1	d_1
c_2	d_2

TABLE VI.5 – $R_1 \div R_2$

C	D
a_1	b_1
a_3	b_3

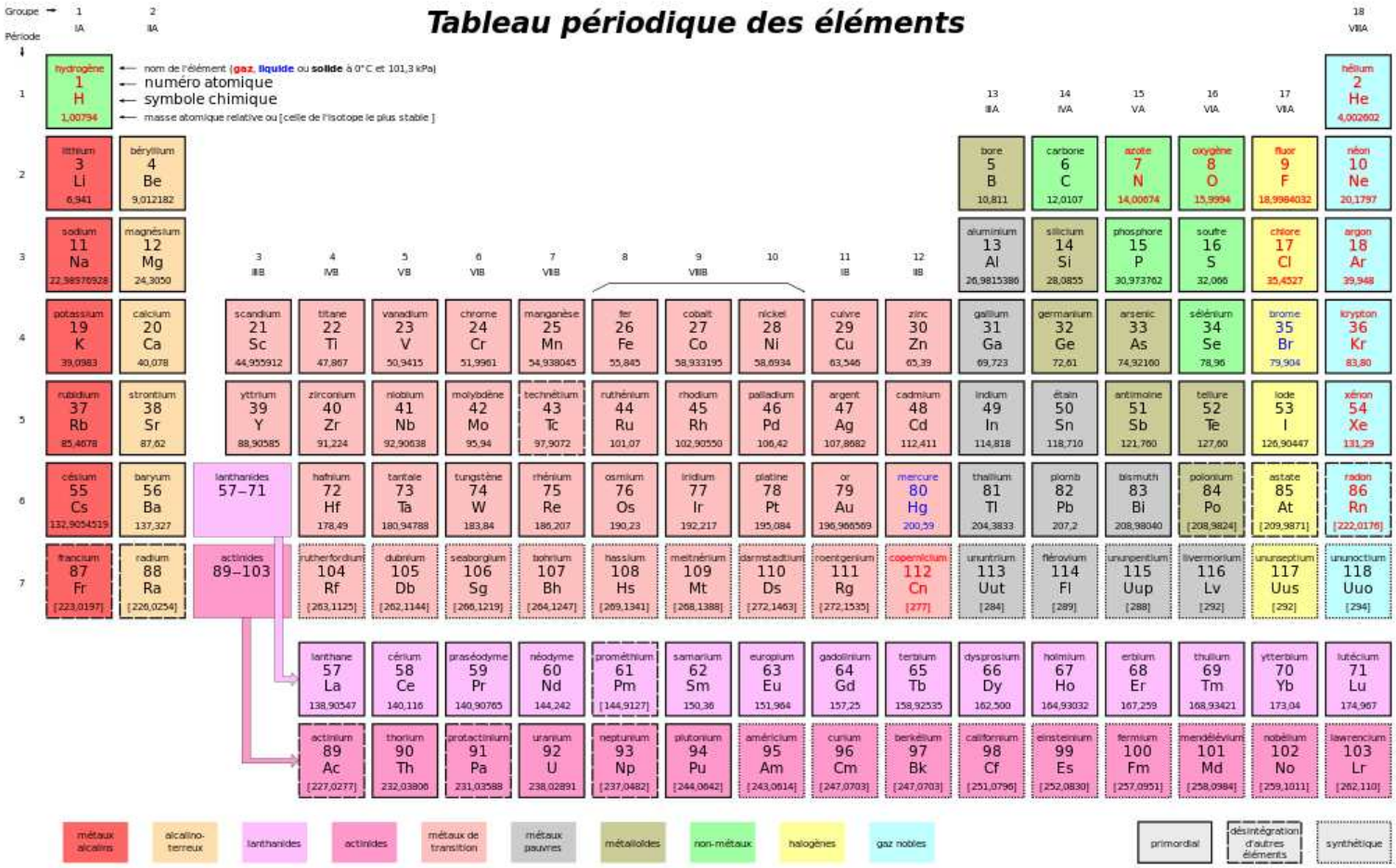


Tableau périodique des éléments
Groupe
Période
nom de l'élément (gaz, liquide ou solide à 0°C et 101,3 kPa)
numéro atomique
symbole chimique
masse atomique relative ou [celle de l'isotope le plus stable]
hydrogène 1 H 1,00794
hélium 2 He 4,002602
lithium 3 Li 6,941
béryllium 4 Be 9,012182
bore 5 B 10,811
carbone 6 C 12,0107
azote 7 N 14,00674
oxygène 8 O 15,9994
fluor 9 F 18,9984032
néon 10 Ne 20,1797
sodium 11 Na 22,98976928
magnésium 12 Mg 24,3050
aluminium 13 Al 26,9815386
silicium 14 Si 28,0855
phosphore 15 P 30,973762
soufre 16 S 32,066
chlore 17 Cl 35,4527
argon 18 Ar 39,948
potassium 19 K 39,0983
calcium 20 Ca 40,078
scandium 21 Sc 44,955912
titane 22 Ti 47,867
vanadium 23 V 50,9415
chrome 24 Cr 51,9961
manganèse 25 Mn 54,938045
fer 26 Fe 55,845
cobalt 27 Co 58,933195
nickel 28 Ni 58,6934
cuivre 29 Cu 63,546
zinc 30 Zn 65,39
gallium 31 Ga 69,723
germanium 32 Ge 72,61
arsenic 33 As 74,92160
sélénium 34 Se 78,96
brome 35 Br 79,904
krypton 36 Kr 83,80
rubidium 37 Rb 85,4678
strontium 38 Sr 87,62
yttrium 39 Y 88,90585
zirconium 40 Zr 91,224
niobium 41 Nb 92,90638
molybdène 42 Mo 95,94
technétium 43 Tc 97,9072
ruthénium 44 Ru 101,07
rhodium 45 Rh 102,90550
palladium 46 Pd 106,42
argent 47 Ag 107,8682
cadmium 48 Cd 112,411
indium 49 In 114,818
étain 50 Sn 118,710
antimoine 51 Sb 121,760
tellure 52 Te 127,60
iode 53 I 126,90447
xénon 54 Xe 131,29
césium 55 Cs 132,9054519
baryum 56 Ba 137,327
lanthanides 57–71
hafnium 72 Hf 178,49
tantale 73 Ta 180,94788
tungstène 74 W 183,84
rhénium 75 Re 186,207
osmium 76 Os 190,23
iridium 77 Ir 192,217
platine 78 Pt 195,084
or 79 Au 196,966569
mercure 80 Hg 200,59
thallium 81 Tl 204,3833
plomb 82 Pb 207,2
bismuth 83 Bi 208,98040
polonium 84 Po [208,9824]
astate 85 At [209,9871]
radon 86 Rn [222,0176]
francium 87 Fr [223,0197]
radium 88 Ra [226,0254]
actinides 89–103
rutherfordium 104 Rf [263,1125]
dubnium 105 Db [262,1144]
seaborgium 106 Sg [266,1219]
bohrium 107 Bh [264,1247]
hassium 108 Hs [269,1341]
meitnérium 109 Mt [268,1388]
darmstadtium 110 Ds [272,1463]
roentgenium 111 Rg [272,1535]
copernicium 112 Cn [277]
ununtrium 113 Uut [284]
flérovium 114 Fl [289]
ununpentium 115 Uup [288]
livermorium 116 Lv [292]
ununseptium 117 Uus [292]
ununoctium 118 Uuo [294]
lanthane 57 La 138,90547
cérium 58 Ce 140,116
praséodyme 59 Pr 140,90765
néodyme 60 Nd 144,242
prométhium 61 Pm [144,9127]
samarium 62 Sm 150,36
europium 63 Eu 151,964
gadolinium 64 Gd 157,25
terbium 65 Tb 158,92535
dysprosium 66 Dy 162,500
holmium 67 Ho 164,93032
erbium 68 Er 167,259
thulium 69 Tm 168,93421
ytterbium 70 Yb 173,04
lutécium 71 Lu 174,967
actinium 89 Ac [227,0277]
thorium 90 Th 232,03806
protactinium 91 Pa 231,03588
uranium 92 U 238,02891
neptunium 93 Np [237,0482]
plutonium 94 Pu [244,0642]
américium 95 Am [243,0614]
curium 96 Cm [247,0703]
berkélium 97 Bk [247,0703]
californium 98 Cf [251,0796]
einsteinium 99 Es [252,0830]
fermium 100 Fm [257,0951]
mendélévium 101 Md [258,0984]
nobélium 102 No [259,1011]
lawrencium 103 Lr [262,110]
métaux alcalins
alcalino-terreux
lanthanides
actinides
métaux de transition
métaux pauvres
métalloïdes
non-métaux
halogènes
gaz nobles
primordial
désintégration d'autres éléments
synthétique