CH2 BASES DE DONNEES : LE MODELE RELATIONNEL

Dans la vie courante, les entreprises, l'état, les associations... sont amenés à gérer dans quantités de données qui sont difficilement exploitables sous *format CSV* soit de par leur taille trop conséquente soit par leur structure même qui comporte des relations entre différentes données.
Il existe un outil complètement adapté à ce type de données, les **bases de données relationnelles**.

exemple de base : une bibliothèque

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	Charmander	Fire		309	39	52	43	60	50	65	1	False
5	Charmeleon	Fire		405	58	64	58	80	65	80	1	False
6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False
6	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130	85	100	1	False
6	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159	115	100	1	False
7	Squirtle	Water		314	44	48	65	50	64	43	1	False
8	Wartortle	Water		405	59	63	80	65	80	58	1	False
9	Blastoise	Water		530	79	83	100	85	105	78	1	False
9	BlastoiseMega Blastoise	Water		630	79	103	120	135	115	78	1	False
10	Caterpie	Bug		195	45	30	35	20	20	45	1	False
11	Metapod	Bug		205	50	20	55	25	25	30	1	False

exemple de base : un classeur csv

Note : *il existe plusieurs types de bases de données : Hiérarchiques, relationnelles... Ce sont aussi les plus répandues. Dans les bases de données relationnelles, les données sont stockées sous forme de table.*

3. Les bases de données

Ces bases de données ont vraiment démarré avec l'émergence d'internet et l'apparition du big data, gigantesque collecte de données qu'il faut stocker, modifier, traiter.
D'où les deux aspects :

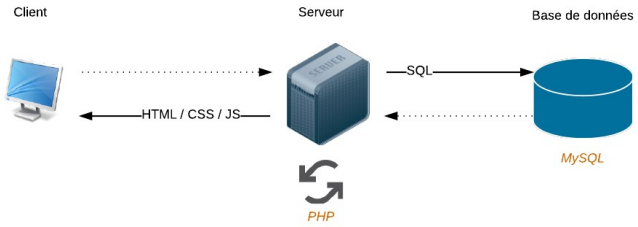
- la **base** en tant que telle, qui contient les données à stocker et répertorier.
- le **SGBD**, ou *Système de Gestion de Bases de Données*.

A retenir :

Une **base de données (BD)** est ensemble d'informations structurées mémorisées sur un support persistant.

Un **Système de Gestion de Base de Données (SGBD)** est un système informatique qui assure la gestion de l'ensemble des informations stockées dans une base de données. Il prend en charge, notamment, les deux grandes fonctionnalités :

- **Accès aux fichiers** de la base, garantissant leur intégrité, contrôlant les opérations concurrentes, optimisant les recherches et mises à jour.
- **Interactions** avec les applications et utilisateurs, grâce à des langages d'interrogation et de manipulation spécifiques. (LMD, ou *Langage de Manipulation de Données*)



SGBD en langage PHP sur un serveur, pour une base de données MySQL

Le SGBD doit permettre d'assurer les fonctions suivantes :

- l'écriture, la lecture et la modification des BD.
- l'indépendance entre le traitement des données et les données directement perceptibles.
- la non-redondance des données.

- le partage des données : elles assurent la possibilité à plusieurs utilisateurs l'accès simultané à la BD.
- la sécurisation de la BD et les niveaux d'accès des utilisateurs : de simples consultants à administrateur.
- la persistance du système en cas de panne. Stockage sur plusieurs supports, chaque modification devant être répercutée sur tous les supports.

Exemples :

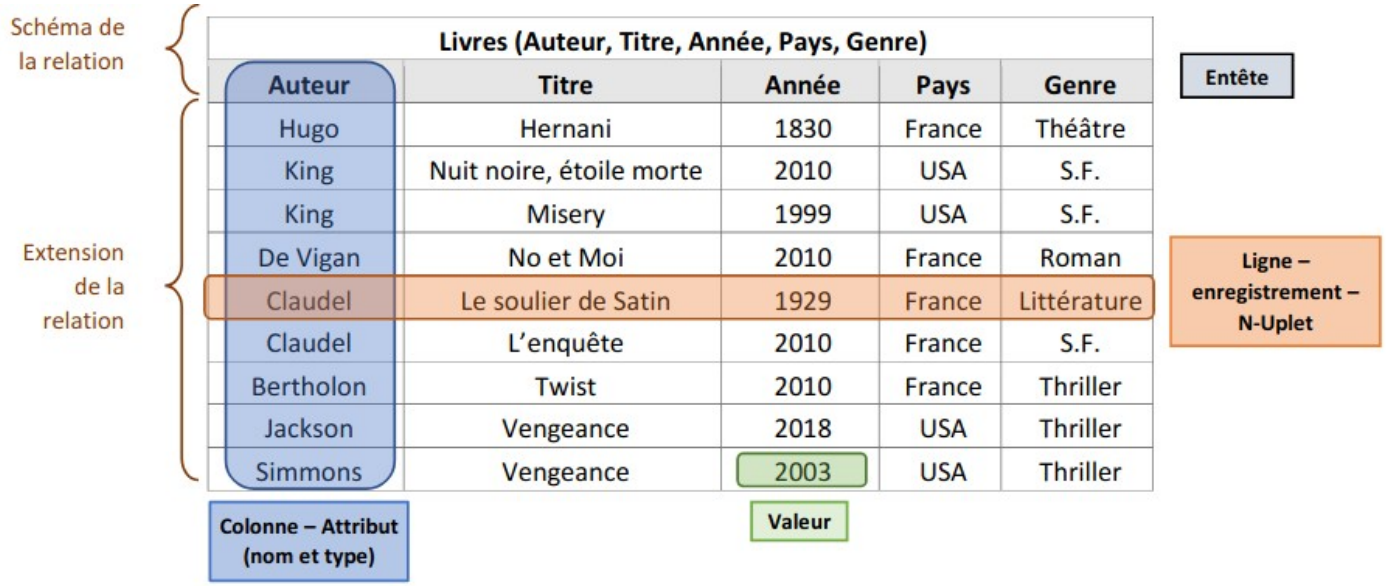
- Voir différents formats sur Pour les bases, on trouvera les formats usuels comme
  MySQL, PostgreSQL, SQLite, Oracle Database, Microsoft SQL Server, …
- Les SGBD sont également variés ; on s'intéressera au langage généralement utilisé, le *SQL*.
- Le format JSON

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

4. Structure d'une base de données

(a) Une façon de présenter les choses

Sur cet exemple, on voit la structure de la **table Livres** d'une base de données ainsi que le vocabulaire associé :



Les **attributs** sont donc, avec leurs types respectifs :

```
Auteur (str), Titre(str), Année (int), Pays(str), Genre(str)
```

Le **tuple** (ou *n-uplet*) surligné est donc :

```
('Claudel', 'Le soulier de satin', 1929, 'France', 'Littérature')
```

Pour cette donnée, la **valeur** de l'attribut *« Année »* est donc *1929*.

A retenir :

Une **table** est un tableau contenant un ensemble de valeurs réparties en colonnes (rubriques ou libellés)
L'**attribut** est le nom que l'on donne à une rubrique (Auteur, Année, ...)
Le **tuple** (ou *n-uplet*) est une ligne dans la table. On parle aussi de donnée.

Une **base de données relationnelle** est un ensemble de tables dépendantes entre elles.

Une **valeur** est l'information attachée à un attribut pour une donnée (une ligne).
L'ensemble des valeurs possibles pour un attribut (liste finie de valeurs, intervalle de valeurs, …) est un **domaine**.

Pour représenter de façon schématique la **relation**, on écrit : `NomTable(Attribut1, Attribut2, ...)`
On parle alors de **schéma relationnel**.

Exemples :

- La relation de la table ci-dessus s'écrit : `Livres(Auteur, Titre, Année, Pays, Genre)`
- La valeur de l'attribut ***« Titre »*** correspondant à l'auteur *« King »* vaut soit *« Nuit noire, étoile morte »*, soit *« Misery »*, suivant le tuple que l'on considère.
- L'attribut **« Genre »** possède 5 valeurs différentes sur cette table, mais en réalité son domaine est plus étendu car il existe d'autres genres

Remarque : Le schéma relationnel peut aussi être représenté comme

Nom de la table →

Attributs →

...

LIVRES
Auteur
Titre
Année
Pays
Genre

(b) La clé, les clés

Prenons en exemple la table suivante, une playlist de radio :

2012-07-13	05H00	Katy Perry	Hot N Cold	2008	F	USA
2012-07-13	05H03	Dry	Ma Melodie	2012	H	France
2012-07-13	05H07	Gotye	Somebody That Use To Know	2012	H	Australie
2012-07-13	05H11	Julian Perretta	Generation X	2012	H	Angleterre
2012-07-13	05H17	Sexion D'assaut	Wati House18	2012	Groupe	France
2012-07-13	05H23	Lmfao	Party Rock Anthem	2011	Groupe	USA
2012-07-13	05H27	Jenifer	Sur Le Fil	2012	F	France
2012-07-13	05H29	Of Monsters And Men	Little Talks	2011	Groupe	Islande
2012-07-13	05H33	Carly Rae Jepsen	Call Me Maybe	2012	F	Canada
2012-07-13	05H40	Pitbull	Back In Time	2012	H	USA
2012-07-13	05H43	Katy Perry	Hot N Cold	2008	F	USA
2012-07-13	05H51	Pink	Blow Me (one Last Kiss)	2012	F	USA

La relation s'écrit : `Playlist(Date, Heure, Artiste, Titre, Année, Formation, Pays)`
Quels attributs permettent d'identifier précisément et sans ambiguïté chaque tuple ?

A retenir :

Chaque relation doit comporter une unique **clé primaire**. C'est une information qui permet d'identifier de manière unique une ligne dans une relation.

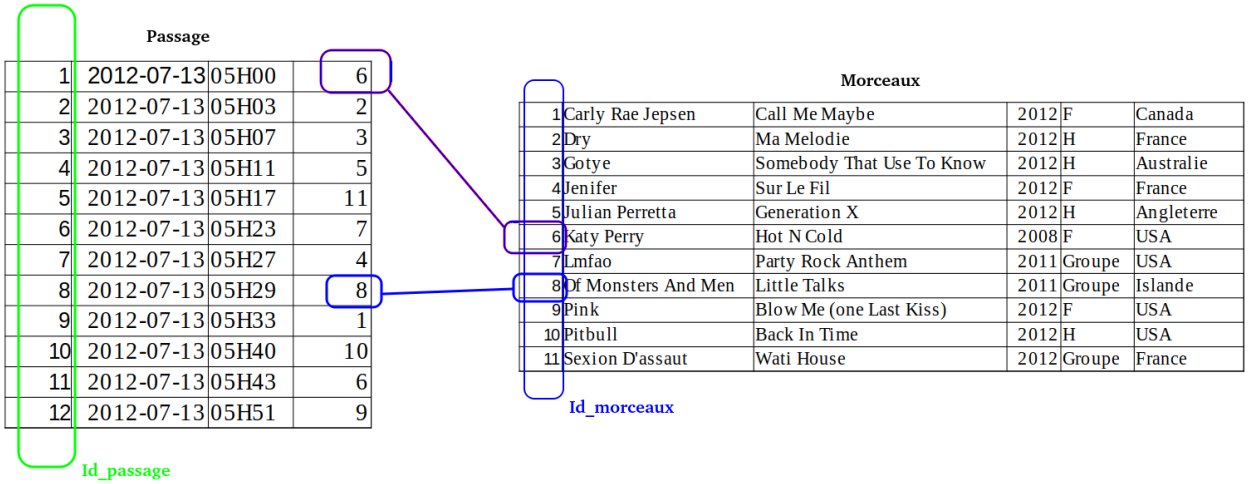
- Une clé primaire peut être composée de plusieurs attributs. Par exemple, dans une base de donnée regroupant les élèves du lycée, l'ensemble (nom, prénom, date de naissance) est une bonne clé primaire. La probabilité d'avoir deux homonymes parfaits nés le même jour étant très faible.
- Si l'ensemble des données d'une table ne présente pas de clé primaire, il est possible d'ajouter un attribut comprenant un **numéro d'identification** (*Id*), généré automatiquement

Exemples :

Dans une table regroupant des voitures, la plaque d'immatriculation est une bonne clé primaire.

En creusant cet exemple, on pourrait écrire cette relation sous forme de 2 tables :

- Passage d'une part listant les horaires et le morceau
- Morceaux d'autre part listant les morceaux et les infos sur l'artiste



Passage

1	2012-07-13	05H00	6
2	2012-07-13	05H03	2
3	2012-07-13	05H07	3
4	2012-07-13	05H11	5
5	2012-07-13	05H17	11
6	2012-07-13	05H23	7
7	2012-07-13	05H27	4
8	2012-07-13	05H29	8
9	2012-07-13	05H33	1
10	2012-07-13	05H40	10
11	2012-07-13	05H43	6
12	2012-07-13	05H51	9

Id_passage

Morceaux

1	Carly Rae Jepsen	Call Me Maybe	2012	F	Canada
2	Dry	Ma Melodie	2012	H	France
3	Gotye	Somebody That Use To Know	2012	H	Australie
4	Jenifer	Sur Le Fil	2012	F	France
5	Julian Perretta	Generation X	2012	H	Angleterre
6	Katy Perry	Hot N Cold	2008	F	USA
7	Lmfao	Party Rock Anthem	2011	Groupe	USA
8	Of Monsters And Men	Little Talks	2011	Groupe	Islande
9	Pink	Blow Me (one Last Kiss)	2012	F	USA
10	Pitbull	Back In Time	2012	H	USA
11	Sexion D'assaut	Wati House	2012	Groupe	France

Id_morceaux

On constate que pour plus de clarté, nous avons ajouter des identifiants : **Id_passage** et **Id_morceaux**.
La relation pour chaque table s'écrit :

```
Passage(Id_passage, Date, Heure, Id_morceau)
Morceaux(Id_morceaux, Artiste, Titre, Année, Formation, Pays)
```

Id_passage est donc la clé primaire de **Passage**, et **Id_morceaux** la clé primaire de **Morceaux**.
On voit apparaître dans la relation le lien entre les 2 tables : **Id_Morceaux** est présent dans la table **Passage**.
Id_morceaux est donc une **clé étrangère** à la table **Passage**, et c'est elle qui relie ces tables.
On la visualise avec un # dans la notation de la relation :

```
Passage(Id_passage, Date, Heure, #Id_morceau)
```

A retenir :

Une **clé étrangère** est un attribut dans une relation dont les données sont prises parmi les données d'une clé primaire d'une autre relation.

```
Table(Id_Table, Attribut1, Attribut2,… , #Clé_Etrangère)
```

Remarque :

L'intérêt de faire plusieurs tables au lieu d'une seule réside dans la facilité de mise à jour, de correction, mais aussi d'accès à la base.

Exemples :

Dans la base d'une entreprise comportant deux tables, on a les relations suivantes :

```
Client(Num_client, Nom_client, Rue, Ville, Code_postal, Telephone)
Facture(Num_facture, Date_facture, #Num_client)
```

- *Num_client* et *Num_facture* sont des clés primaires, car l'identification de chaque tuple par ces clés est unique.
- *Num_client* est aussi utilisé dans la relation **Facture**, donc c'est une clé étrangère.

(c) Les contraintes d'intégrité

Par exemple, considérons cette version (très) simplifiée de la table `Transaction` décrivant des opérations bancaires :

Date	Id_Client_Emetteur	Id_Client_Destination	Montant	Type
25/12/2020	0001200263	0001200526	1000	virement
25/12/2020	0001200653	0001300598	250	cheque
25/12/2020	0001200263	0001200526	1000	virement
26/12/2020	0001200698	0001200584	632	virement

Comparer les **tuples** des transactions ligne 2 et ligne 3 : s'agit-il d'une erreur ou bien effectivement de deux transactions différentes ?? …

Une solution est donc d'ajouter un **nouvel attribut qui identifie de façon unique** la donnée. Le plus simple est en général l'ajout d'un Id.

Id_Transaction	Date	Id_Client_Emetteur	Id_Client_Destination	Montant	Type
1	25/12/2020	0001200263	0001200526	1000	virement
2	25/12/2020	0001200653	0001300598	250	cheque
3	25/12/2020	0001200263	0001200526	1000	virement
4	26/12/2020	0001200698	0001200584	632	virement

Maintenant, nous savons que les deux opérations sont bien distinctes !

A retenir :

La cohérence des données au sein d'une base est assurée par des **contraintes d'intégrité**. Il s'agit de propriétés que les données doivent vérifier à tout instant.

- **Les contraintes d'unicité** (ou d'entité)

Chaque tuple (ou entité) est identifié de manière unique grâce à la clé primaire.

- **Les contraintes de référence**

Elles garantissent l'association de deux relations : la clé étrangère (attribut de la relation B) utilisée dans la relation A existe.

- **Les contraintes de domaines**

Les valeurs sont restreintes à un domaine. En dehors de ce domaine, la valeur est illégale et le SGBD retourne une erreur.

Exemples :

Dans la base d'entreprise de l'exemple précédent :

```
Client(Num_client, Nom_client, Rue, Ville, Code_postal, Telephone)
Facture(Num_facture, Date_facture, #Num_client)
```

- La contrainte d'unicité assure qu'un client n'est pas en doublon dans la table **Client.**
- La contrainte de référence assure qu'un client existe bien (*#Num_client*) dans la table **Client** pour pouvoir saisir une donnée de la table **Facture**.
- La contrainte de domaine rejette une saisie de *Date_facture* sous la forme « 20_octobre_2021 », mais accepte une saisie sous la forme « 20-10-2021 »

3. Gestion d'une base de données

Comme nous l'avons vu plus haut, il est impératif de mettre en place un SGBD lié à une base afin de gérer l'ensemble des événements : lecture/écriture d'une valeur, vérification des contraintes, mises à jour, etc.

(a) Le SQL : communiquer avec le serveur de données

L'algèbre relationnelle est la base du développement du **langage SQL** (*Structured Query Langage*). Ce langage textuel permet de communiquer avec une base de données relationnelle. Il a été réalisé par IBM puis est devenu un standard ANSI (*American National Standards Institute*) approuvé par l'ISO (*International Standards Organization*) en 1987.
Le dernier standard validé est SQL3 appelé bien souvent SQL/99. Toutefois, pour avoir un langage plus complet, les concepteurs de SGBD/R ajoutent certaines fonctionnalités au SQL standard.
Seules l'interrogation et la manipulation des données seront abordées ici.

Un des avantages de ce langage est qu'il est assez proche du langage naturel (en anglais...), et qu'il permet ainsi d'interroger et de maintenir des bases de données assez simplement. C'est donc un langage de **haut niveau** contrairement aux assembleurs qui sont plus "proches" de la machine.
Une autre spécificité, est qu'on peut classer SQL parmi les langages **déclaratifs** : les commandes SQL se basent sur le résultat que l'on veut obtenir, nous n'avons pas à décrire, ni même connaître la façon dont on va obtenir ce résultat, toute la partie algorithmique d'une recherche, du parcours d'une relation est « caché », optimisé et traité par SQL.

(b) Structure d'un requête SQL

Le langage SQL est basé sur une succession de requêtes. Ces requêtes se présentent comme des instructions plus ou moins complexes.

Voici la structure des requêtes les plus courantes, reliées aux tables suivantes :

Auteurs

id	nom	prenom	ann_naissance	langue_ecriture
1	Orwell	George	1903	anglais
2	Herbert	Frank	1920	anglais
3	Asimov	Isaac	1920	anglais
4	Huxley	Aldous	1894	anglais
5	Bradbury	Ray	1920	anglais
6	K.Dick	Philip	1928	anglais
7	Barjavel	René	1911	français
8	Boulle	Pierre	1912	français
9	Van Vogt	Alfred Elton	1912	anglais
10	Verne	Jules	1828	français

et **Livres**

id	titre	id_auteur	ann_publi	note
1	1984	1	1949	10
2	Dune	2	1965	8
3	Fondation	3	1951	9
4	Le meilleur des mondes	4	1931	7
5	Fahrenheit 451	5	1953	7
6	Ubik	6	1969	9
7	Chroniques martiennes	5	1950	8
8	La nuit des temps	7	1968	7
9	Blade Runner	6	1968	8

id	titre	id_auteur	ann_publi	note
10	Les Robots	3	1950	9
11	La Planète des singes	8	1963	8
12	Ravage	7	1943	8
13	Le Maître du Haut Château	6	1962	8
14	Le monde des Ā	9	1945	7
15	La Fin de l'éternité	3	1955	8
16	De la Terre à la Lune	10	1865	10

Les requêtes de base à connaître sont les suivantes, classées par fonctionnalité :

INTERROGER `SELECT attribut FROM relation WHERE condition ;`

C'est la requête de essentielle pour interroger et afficher un ensemble de valeurs en utilisant la combinaison SELECT … FROM … En plus, on peut ajouter une condition avec WHERE.

Exemples :

`SELECT id, nom FROM Auteurs ;`	*→ Affiche les valeurs d'Id et noms de la table Auteurs.*
`SELECT * FROM Auteurs ;`	*→ Affiche les valeurs de tous les attributs de la table Auteurs.*
`SELECT nom FROM Auteurs` `WHERE ann_naissance > 1900 ;`	*→ Affiche les valeurs de nom pour les auteurs dont la date de naissance est au XX^{ème} siècle.*

`SELECT nom FROM Auteurs`
`WHERE langue_ecriture = 'français'`
`OR ann_naissance > 1920 ;`
	→ Affiche les valeurs de nom pour les auteurs nés après 1920 ou écrivant en langue française.

CLASSER `SELECT attribut FROM relation ORDER BY attribut ;`

Il est possible de classer les résultats d'une requête par ordre croissant grâce à la clause ORDER BY.

Exemples :

`SELECT nom FROM Auteurs ORDER BY ann_naissance ;`
→ Affiche les noms des auteurs classés par année de naissance croissante.
`SELECT nom FROM Auteurs WHERE langue_ecriture = 'français' ORDER BY ann_naissance ;`
→ Même chose, mais pour les auteurs de langue française.
`SELECT nom FROM Auteurs WHERE langue_ecriture = 'français' ORDER BY ann_naissance DESC ;`
→ Même chose, mais dans l'ordre décroissant

DIFFERENCIER `SELECT DISTINCT attribut FROM relation ;`

Il est possible d'éviter les doublons dans une réponse grâce à la clause DISTINCT.

Exemples :

`SELECT lang_ecriture FROM Auteurs ;`

→ *Affiche toutes les valeurs de l'attribut lang_ecriture ...*
`SELECT DISTINCT lang_ecriture FROM Auteurs ;`
→ *… N'affichera pas les doublons.*

INSERER **`INSERT INTO relation (att1, att2, ...) VALUES (val1, val2, …) ;`**

Il est possible d'ajouter une entrée à une table grâce à une requête d'insertion INSERT INTO.

Exemples :

`INSERT INTO Livres (id,titre,id_auteur,ann_publi,note) VALUES (17,'Hypérion',11,1989,8) ;`
→ *Ajoute un nouveau tuple à la table Livres.*

SUPPRIMER **`DELETE FROM relation WHERE condition ;`**

De même, on peut supprimer une donnée par DELETE FROM

Exemples :

`DELETE FROM Livres WHERE titre = 'Blade Runner' ;`
→ *Supprime l'ensemble du tuple correspondant au titre « Blade Runner ».*
`DELETE FROM Livres ;`
→ *Supprime l'ensemble de la table !!*

METTRE A JOUR **`UPDATE relation SET nouvelle_valeur WHERE condition ;`**

UPDATE va permettre de modifier une ou des entrées. Nous utiliserons WHERE, comme dans le cas d'un "SELECT", pour spécifier les entrées à modifier.

Exemples :

`UPDATE Livres SET note = 7 WHERE titre = 'Hyperion' ;`
→ *Passe la note de 8 à 7 pour le tuple correspondant au titre « Hyperion ».*

(c) Jointure

Nous avons 2 tables, grâce aux **jointures** nous allons pouvoir associer ces 2 tables dans une même requête.

En général, les jointures consistent à associer des lignes de 2 tables. Elles permettent d'établir un lien entre 2 tables. Qui dit lien entre 2 tables dit clé étrangère et clé primaire.

```
SELECT * FROM Livres
INNER JOIN Auteurs ON Livres.id_auteur = Auteurs.id ;
```

- Le FROM Livres INNER JOIN Auteurs permet de créer une **jointure** entre les tables Livres et Auteurs (= rassembler les 2 tables en une seule grande table).
- Le ON Livres.id_auteur = Auteurs.id signifie qu'une ligne quelconque A de la table Livres devra être fusionnée avec la ligne B de la table Auteurs à condition que l'attribut id_auteur de la ligne A soit égal à l'attribut id de la ligne B.

Par exemple, la ligne 1 (id=1) de la table Livres (que l'on nommera dans la suite ligne A) sera fusionnée avec la ligne 1 (id=1) de la table Auteurs (que l'on nommera dans la suite B) car l'attribut id_auteur de la ligne A est égal à 1 et l'attribut id de la ligne B est aussi égal à 1.

Autre exemple, la ligne 1 (id=1) de la table Livres (que l'on nommera dans la suite ligne A) ne sera pas fusionnée avec la ligne 2 (id=2) de la table Auteurs (que l'on nommera dans la suite B') car l'attribut id_auteur de la ligne A est égal à 1 alors que l'attribut id de la ligne B' est égal à 2.
Dans notre exemple l'attribut id_auteur de la tables Livres est bien une clé étrangère puisque cet attribut correspond à l'attribut id de la table Auteurs.
La requête ci-dessus permettra d'obtenir le résultat suivant :

id	titre	id_auteur	ann_publi	note	id	nom	prenom	ann_naissance	langue_ecriture
1	1984	1	1949	10	1	Orwell	George	1903	anglais
2	Dune	2	1965	8	2	Herbert	Frank	1920	anglais
3	Fondation	3	1951	9	3	Asimov	Isaac	1920	anglais
4	Le meilleur des mondes	4	1931	7	4	Huxley	Aldous	1894	anglais
5	Fahrenheit 451	5	1953	7	5	Bradbury	Ray	1920	anglais
6	Ubik	6	1969	9	6	K.Dick	Philip	1928	anglais
7	Chroniques martiennes	5	1950	8	5	Bradbury	Ray	1920	anglais
8	La nuit des temps	7	1968	7	7	Barjavel	René	1911	français
9	Blade Runner	6	1968	8	6	K.Dick	Philip	1928	anglais
10	Les Robots	3	1950	9	3	Asimov	Isaac	1920	anglais
11	La Planète des singes	8	1963	8	8	Boulle	Pierre	1912	français
12	Ravage	7	1943	8	7	Barjavel	René	1911	français
13	Le Maître du Haut Château	6	1962	8	6	K.Dick	Philip	1928	anglais
14	Le monde des Ā	9	1945	7	9	Van Vogt	Alfred Elton	1912	anglais
15	La Fin de l’éternité	3	1955	8	3	Asimov	Isaac	1920	anglais
16	De la Terre à la Lune	10	1865	10	10	Verne	Jules	1828	français

Dans le cas d'une jointure, il est tout à fait possible de sélectionner certains attributs et pas d'autres (aucune obligation de sélectionner tous les attributs des 2 tables, par exemple :

```
SELECT Livres.titre, Auteurs.nom, Auteurs.prenom FROM Auteurs
INNER JOIN Livres ON Livres.id_auteur = Auteurs.id ;
```

On obtiendra alors une jointure uniquement avec l'attribut titre de la table Livres et les attributs nom, prenom de la table Auteurs.

titre	nom	prenom
1984	Orwell	George
Dune	Herbert	Frank
Fondation	Asimov	Isaac
Le meilleur des mondes	Huxley	Aldous
Fahrenheit 451	Bradbury	Ray

titre	nom	prenom
Ubik	K.Dick	Philip
Chroniques martiennes	Bradbury	Ray
La nuit des temps	Barjavel	René
Blade Runner	K.Dick	Philip
Les Robots	Asimov	Isaac
La Planète des singes	Boulle	Pierre
Ravage	Barjavel	René
Le Maître du Haut Château	K.Dick	Philip
Le monde des Ā	Van Vogt	Alfred Elton
La Fin de l'éternité	Asimov	Isaac
De la Terre à la Lune	Verne	Jules

Et en ajoutant une condition WHERE, on peut ne sélectionner que certaines lignes :

```
SELECT Livres.titre, Auteurs.nom, Auteurs.prenom FROM Auteurs
INNER JOIN Livres ON Livres.id_auteur = Auteurs.id

WHERE Livres.ann_publi > 1965 ;
```

On obtient :

titre	nom	prenom
Ubik	K.Dick	Philip
La nuit des temps	Barjavel	René
Blade Runner	K.Dick	Philip