# Chapitre 8

# Bases de données

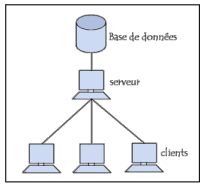
## 8.1 Introduction

### 8.1.1 Base de données. Architecture clients-serveur

• L'utilisation de fichiers pour stocker des données est trop limitée pour gérer un grand volume de données. On utilise plutôt une base de données qui manipulera des fichiers mais organisera les données de façon à les gérer de façon optimale.

• Par exemple, dans les sites de e-commerce, le catalogue des produits est stocké dans une base de donnée. Toutes les données administratives de la population française sont stockées dans des bases de données.

• Une base de donnée est le système permettant le stockage des données, de manière structurée, en général sur un serveur (ordinateur relié à un réseau, internet par exemple). Elle est conçue selon une architecture clients-serveurs :




• Le serveur reçoit des **requêtes** de clients pour :

– créer/modifier la base de donnée (seulement les administrateurs)

– Interroger la base de donnée (tous les utilisateurs autorisés).

### 8.1.2 Une base de donnée qu'est que c'est ?

• Un bon exemple de base de données est un carnet d'adresse. Dans un carnet d'adresse chaque entrée a un prénom, nom, numéro de téléphone, adresse, etc... Toutes ces données sont en relation : une adresse est reliée à un nom-prénom, etc...

• La recherche dans un carnet d'adresse se fait en général par le nom. Mais par exemple un commercial pourra rechercher plutôt par adresse afin de gérer ses déplacements ; une centrale d'appel par numéro de téléphone pour optimiser ses coûts de frais d'appel ; une personne dont le véhicule est en panne pourra y rechercher plutôt un garagiste, etc...

• Une base de donnée répond à des requêtes formulées dans un langage structuré de requêtes : ce langage permettra de créer/modifier ou d'accéder aux informations pertinentes à l'aide d'un langage approprié. Par exemple : "Quels sont les garagistes dans mon carnet d'adresse situés sur la commune de mon lieu de panne ?".

### 8.1.3 Exemple

• Illustrons ces propos en imaginant quelle pourrait être une base de donnée pour gérer tous ses contacts.

• On pourrait utiliser un carnet d'adresse pour stocker toutes les informations concernant ses amis : prénom, nom, numéro(s) de téléphone (1 ou plusieurs), adresse, date anniversaire.

• Et un autre carnet d'adresse pour stocker toutes les informations concernant ses contacts professionnels : prénom, nom, numéros(s) de téléphone, adresse, société, position.

• Ces deux carnets d'adresse constitueront deux tables dans la base de donnée ; la table est définie par son nom et son schéma de relation : c'est l'ensemble des champs (attributs) qui la constituent (nom, prénom, téléphone, etc...).

• Les deux tables et leurs schémas relationnels :

| Amis | | | | |
|---|---|---|---|---|
| Prénom | Nom | Téléphone | Adresse | Anniversaire |

| Contacts Pro. | | | | | |
|---|---|---|---|---|---|
| Prénom | Nom | Téléphone | Adresse | Société | Poste |

### 8.1.4 Vocabulaire

• Chaque table va regrouper des contacts, $n$-uplets de valeurs. Ils ont appelés enregistrement.
Par exemple on a ajoute un enregistrement. :

| Amis | | | | |
|---|---|---|---|---|
| Prénom | Nom | Téléphone | Adresse | Anniversaire |
| Paul | Dupond | 0102030405 | 13 av. de la lumière | 27 mars 1990 |

• Cet enregistrement est constitué des valeurs d'attributs : Paul, Dupond, 0102030405, etc...

• Chaque attribut a un domaine ; c'est essentiellement son type : prénom, nom et adresse seront stockés comme chaîne de caractère ; téléphone comme un entier et anniversaire comme une date.

### 8.1.5 Notion de clé

• Dans une table on doit pouvoir accéder facilement à un enregistrement à l'aide de la valeur de l'un de ses attributs.

• Un ou plusieurs attributs donnant accès à un unique élément dans une table s'appelle une clé.

• Par exemple le couple (nom, prénom) est un assez bon choix de clé. On parle de clé composite quand elle est constituée de plusieurs attributs.

• La clé choisie s'appelle la clé primaire. Elle doit être aussi simple que possible.

• Les autres clés possibles sont les clés secondaires.

## 8.2 Tables, attributs et enregistrements

### 8.2.1 Vocabulaire des BDD

| | |
|---|---|
| Une **Base de donnée** | est constituée de tables, contenant les données organisées selon un schéma relationnel. |
| Une **Table** ou **Relation** | est un ensemble de données organisées vérifiant un même schéma relationnel. |
| Un **schéma relationnel** | est une suite finie d'attributs (colonnes d'une table). |
| Un **Attribut** | définit la colonne d'une table : nom (identifiant) et type. |
| Un **Enregistrement** | est un élément d'une table (une ligne). |
| Une **Clé** | est un ou plusieurs attributs donnant accès à un seul élément dans une table. |
| La **Clé primaire** | est la clé choisie parmi toutes les clés possibles. |
| Une **Clé secondaire** | est toute clé autre que la clé primaire |
| Une **Clé étrangère** | est un attribut dont les valeurs figurent dans le domaine d'une clé primaire (non composite) d'une autre table. |

### 8.2.2 Théorie : algèbre relationnelle

• Une BDD est constituée de tables (ou relations) chacune définie par un schéma relationnel.

• Un **schéma relationnel** est une suite finie d'attributs : $S = (A_1, A_2, \dots, A_p)$.

• Chaque **attribut** $A_i$ est doté d'un *type*; en SQL les types peuvent être : `NULL`, `BOOLEAN`, `INT`, `INT(N)`, `DECIMAL`, `DECIMAL(N)`, `DECIMAL(N,M)`, `FLOAT`, `FLOAT(N)`, `VARCHAR`, `VARCHAR(N)`, `DATE`, `YEAR`, `TIMES`, etc ... (SQLite comprend ces derniers mais ne stocke que sous les types suivants : `NULL, INTEGER, REAL, TEXT, BLOB`.)

• Chaque type définit le *Domaine* $DOM(A_i)$ de l'attribut $A_i$, c'est à dire l'ensemble des valeurs qu'il peut prendre.

• Une relation $R(S)$ de schéma relationnel $S = (A_1, A_2, \dots, A_p)$, est un sous-ensemble fini du produit cartésien :

$$DOM(A_1) \times DOM(A_2) \times \cdots \times DOM(A_p)$$

Une relation (ou table) est analogue à un ensemble mathématique à la différence près qu'il est possible d'avoir répétitions. C'est possible grâce l'usage d'un identifiant clé : $R(S) \subset \mathbb{N} \times DOM(A_1) \times \cdots \times DOM(A_p)$.

• On dit que deux relations *ont le même schéma* si ils ont même attributs (noms et types).

• $e \in DOM(A_1) \times DOM(A_2) \times \cdots \times DOM(A_p)$ est un *enregistrement* d'une relation $R(A_1, A_2, \dots, A_p)$.

• Si $A_k$ est un attribut, alors $e[A_k]$ désigne la valeur de l'attribut $A_k$ dans l'enregistrement $e$.

• Si $K = (A'_1, \dots A'_q)$ est une sous-suite d'attributs de $(A_1, \dots A_p)$, on note $e[K]$ le $q$-uplet $\big(e[A'_1], \dots, e[A'_q]\big)$ des valeurs dans $e$ des attributs $A'_1, \dots, A'_q$.

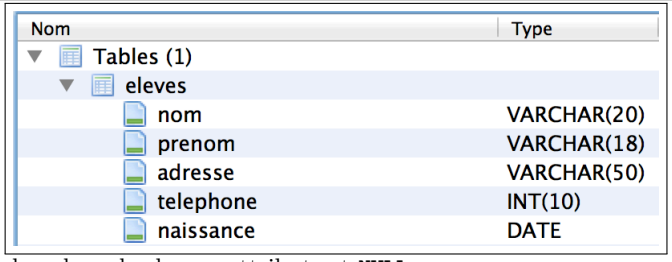
• Pour une relation $T \subset DOM(A_1) \times DOM(A_2) \times \cdots \times DOM(A_p)$, une *clé* est un ensemble $C$ d'attributs, tels que pour tout couple $e, e'$ dans $T$ :

$$e[C] = e'[C] \implies e = e'$$

### 8.2.3 Exemple

• Dans une base de donnée nommée `classe` : créons une table `eleves` ayant pour attributs : nom, prénom, adresse, telephone, date de naissance :

```
CREATE TABLE eleves (
    nom VARCHAR(20),
    prenom VARCHAR(18),
    adresse VARCHAR(50),
    telephone INT(10),
    naissance DATE )
```




Par défaut la valeur de chaque attribut est `NULL`.

• Nous avons saisi manuellement les enregistrements suivants dans la table `eleves` :

| | nom | prenom | adresse | telephone | naissance | langue |
|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | Dupond | Tom | 3 place de la gare | 609080706 | 1995-03-15 | esp |
| 2 | Charron | Ella | 10 rue de la gare | 0607080908 | 1994-12-10 | ita |
| 3 | Cuvelier | Paul | 11 rue Paul Bert | 0665544332 | 1995-09-12 | all |
| 4 | Dumas | Anne | 5 rue de l'oiseau | 0667778778 | 1994-10-01 | rus |
| 5 | Henri | Thomas | 3 quai du port | 0607677878 | 1995-01-24 | all |
| 6 | Laurin | Theo | 4 place de l'Ormeau | 0605040321 | 1995-06-07 | esp |

• On peut alors afficher la table grâce à la requête :

```
SELECT * FROM eleves
```

qui produit pour résultat :

| | nom | prenom | adresse | telephone | naissance | langue |
|---|---|---|---|---|---|---|
| 1 | Dupond | Tom | 3 place de la gare | 609080706 | 1995-03-15 | esp |
| 2 | Charron | Ella | 10 rue de la gare | 607080908 | 1994-12-10 | ita |
| 3 | Cuvelier | Paul | 11 rue Paul Bert | 665544332 | 1995-09-12 | all |
| 4 | Dumas | Anne | 5 rue de l'oiseau | 667778778 | 1994-10-01 | rus |
| 5 | Henri | Thomas | 3 quai du port | 607677878 | 1995-01-24 | all |
| 6 | Laurin | Theo | 4 place de l'Ormeau | 605040321 | 1995-06-07 | esp |

• On peut ne sélectionner qu'un enregistrement :

```
SELECT * FROM eleves WHERE nom = 'Charron'
```

| | nom | prenom | adresse | telephone | naissance | langue |
|---|---|---|---|---|---|---|
| 1 | Charron | Ella | 10 rue de la gare | 607080908 | 1994-12-10 | ita |

• ou ne sélectionner que certains attributs :

```
SELECT nom, prénom FROM eleves
```

| | nom | prenom |
|---|---|---|
| 1 | Dupond | Tom |
| 2 | Charron | Ella |
| 3 | Cuvelier | Paul |
| 4 | Dumas | Anne |
| 5 | Henri | Thomas |
| 6 | Laurin | Theo |

• `... ORDER BY...` permet de les ordonner selon une valeur d'attribut :

```
SELECT nom, prénom, naissance FROM eleves ORDER BY nom
```

| | nom | prenom | naissance |
|---|---|---|---|
| 1 | Charron | Ella | 1994-12-10 |
| 2 | Cuvelier | Paul | 1995-09-12 |
| 3 | Dumas | Anne | 1994-10-01 |
| 4 | Dupond | Tom | 1995-03-15 |
| 5 | Henri | Thomas | 1995-01-24 |
| 6 | Laurin | Theo | 1995-06-07 |

### 8.2.4 Commandes SQL de sélection

| | |
|---|---|
| `SELECT * FROM ma_table` | Afficher tous les enregistrements de la table `ma_table` |
| `SELECT A1, ..., Ap FROM ma_table` | Projection : afficher certaines colonnes de la table `ma_table` |
| `SELECT DISTINCT A1 FROM ma_table` | Projection avec élimination des doublons de l'attribut `A1` de `ma_table` |
| `SELECT ... FROM ... WHERE condition` | Sélection : sélectionne les enregistre--ments vérifiant une condition |
| `SELECT ... FROM ... AS nom` | Sélection avec renommage |
| `SELECT ... ORDER BY Ak [DESC]` | Ordonner l'affichage selon un attribut |
| `SELECT ... LIMIT n` | Limitation aux `n` premiers éléments |
| `SELECT ... LIMIT n OFFSET m` | Limitation aux `n` éléments à partir du `(m+1)`-ième |

## 8.3 Opérateurs unaires : projection, sélection, agrégation

### 8.3.1 Définition formelle

• Soit une relation (table) $r(A_1, A_2, \ldots, A_p)$.

**Projection suivant un sous-ensemble d'attribut** :

• Soit $X = (A'_1, A'_2, \ldots, A'_q)$ une suite finie d'attributs dans $\{A_1, A_2, \ldots, A_p\}$. La **projection** de la relation $r$ suivant $X$ est :

$$\pi_X(r) = \{e[X]; e \in r\}$$

On demande aussi à la projection d'effacer les doublons.

**Sélection par une propriété :**

• Si on se donne une propriété (ou prédicat) $P$, la **sélection** (ou **restriction**) de la relation $r$ par la propriété $P$ est :

$$\sigma_P(r) = \{e \in r\,;\, e \text{ satisfait } P\}$$

• **Projection et sélection retournent une relation !**
Ce qui autorise les requêtes composées :

```
SELECT ... FROM (SELECT ... FROM ...) ...
```

• **Renommage** : changer le nom d'un attribut ou d'une table : avec `AS`.

### 8.3.2 Projection

**Projection :**

• Pour projeter : `SELECT DISTINCT attributs FROM`

```
SELECT DISTINCT nom, prénom FROM eleves
```

projette la table `eleves` suivant les attributs `nom, prénom`.

| | nom | prenom |
|---|---|---|
| 1 | Dupond | Tom |
| 2 | Charron | Ella |
| 3 | Cuvelier | Paul |
| 4 | Dumas | Anne |
| 5 | Henri | Thomas |
| 6 | Laurin | Theo |

### 8.3.3 Sélection

**Sélection :**

• Pour sélectionner : `SELECT ... FROM ... WHERE ...`

```
SELECT * FROM eleves WHERE naissance > '1995-01-01'
```

Sélection de toutes les lignes des élèves nés au plus tôt en 1995.

| | nom | prenom | adresse | telephone | naissance | langue |
|---|---|---|---|---|---|---|
| 1 | Dupond | Tom | 3 place de la gare | 609080706 | 1995-03-15 | esp |
| 2 | Cuvelier | Paul | 11 rue Paul Bert | 665544332 | 1995-09-12 | all |
| 3 | Henri | Thomas | 3 quai du port | 607677878 | 1995-01-24 | all |
| 4 | Laurin | Theo | 4 place de l'Ormeau | 605040321 | 1995-06-07 | esp |

• Pour sélectionner : `SELECT attributs FROM table WHERE conditions`

Les attributs peuvent être constitués :

1. De un ou plusieurs attributs, séparés par des virgules
2. un attribut étant soit un attribut de la table, soit obtenu à l'aide d'opérations arithmétiques sur des attributs numériques de la table ou des nombres, à l'aide des opérateurs `+`, `-`, `*`, `/` (addition, soustraction, multiplication, division) et des parenthèses `( . )`.
   Exemple : `SELECT (A1+A2)/2, (A1-A2)/2 FROM table WHERE ...`

Les conditions peuvent être constituées :

1. Opérateurs de comparaison : `=`, `<`, `>`, `<=`, etc...
2. Connecteurs logiques : `AND`, `OR`, `NOT`, `XOR` et parenthèses.
3. Prédicat ensemblistes : `EXISTS()`, `IN()` (HORS PROGRAMME)
   (a) `EXISTS(table)` est vérifié lorsque `table` est non vide. Le plus souvent `table` est `SELECT .... FROM ...` : le résultat d'une sélection.
   (b) `attribut IN (liste)` est vérifié lorsque `attribut` apparait dans `liste`. Le plus souvent `liste` est une liste de valeurs `(1,2,3,...)` ou une liste résultat d'une sélection : `SELECT attribut FROM ....`

• On peut aussi ajouter en fin de commande un ordre (croissant/décroissant) :

```
ORDER BY attribut [DESC]
```

pour ordonner les résultats.
Ainsi qu'une limitation :

```
LIMIT n [OFFSET m]
```

pour limiter l'affichage aux $n$ premiers à partir du $(m+1)$-ième ; le plus souvent après `ORDER BY attribut`.

• Exemple :

```
SELECT nom, prenom FROM eleves ORDER BY naissance DESC LIMIT 1
```

pour obtenir nom et prénom de l'élève le plus agé, ou :

```
SELECT nom, prenom FROM eleves ORDER BY naissance DESC LIMIT 1 OFFSET 1
```

pour le second plus agé.

### 8.3.4 Fonctions d'agrégations

• Une fonction d'agrégation est une application qui a une table associe un nombre. Le plus souvent pour compter, faire la moyenne d'un attribut, etc..., selon une colonne.

• Les principales fonctions d'agrégation :

| | |
|---|---|
| `COUNT` | pour compter le nombre d'enregistrements d'une table |
| `AVG` | valeur moyenne d'une colonne de type numérique d'une table |
| `MIN` | valeur minimale d'une colonne de type numérique d'une table |
| `MAX` | valeur maximale d'une colonne de type numérique d'une table |
| `SUM` | somme d'une colonne de type numérique d'une table |

• Exemple : Compter les élèves (en paramètre les attributs à compter, les `NULL` ne sont pas comptabilisés)

```
SELECT COUNT(*) FROM eleves
```

| | COUNT(*) |
|---|---|
| 1 | 6 |

La table contient 6 enregistrements (= élèves).

• Une fonction d'agrégation retourne une seule ligne. Avec la commande `GROUP BY` on retourne plusieurs lignes en regroupant les mêmes valeurs d'un attribut et en effectuant l'agrégation sur chaque groupe ; c'est une agrégation :

```
SELECT COUNT(*) FROM eleves GROUP BY langue
```

On compte le nombre d'élèves par groupe de langue.

| | COUNT(*) | langue |
|---|---|---|
| 1 | 2 | all |
| 2 | 2 | esp |
| 3 | 1 | ita |
| 4 | 1 | rus |

• `GROUP BY` peut être suivi de `HAVING condition` pour requérir une condition dans la constitution des groupes :

```
SELECT COUNT(*) FROM eleves GROUP BY langue HAVING langue = 'esp'
```

ne comptera que le groupe des élèves faisant espagnol. Même résultat avec :

```
SELECT COUNT(*) FROM (SELECT * FROM eleves WHERE langue = 'esp')
```

•**Différence avec `WHERE` :** `HAVING` doit suivre une agrégation `GROUP BY...` et peut être suivie d'une condition invoquant une fonction d'agrégation : `MAX()`, `MIN()`, `SUM()`, `COUNT()`. Son emploi est préférable à `WHERE` car optimisé.

## 8.4 Opérateurs binaires ensemblistes

• On considère deux relations $r_1$ et $r_2$ ayant le même schéma $(A_1, \ldots, A_p)$.

• La **Réunion** des relations $r_1$ et $r_2$ est la relation :

$$r_1 \cup r_2 = \{e\,;\, e \in r_1 \text{ ou } e \in r_2\}$$

```
SELECT * FROM r1 UNION SELECT * FROM r2
```

• L'**Intersection** des relations $r_1$ et $r_2$ est la relation :

$$r_1 \cap r_2 = \{e\,;\, e \in r_1 \text{ et } e \in r_2\}$$

```
SELECT * FROM r1 INTERSECT SELECT * FROM r2
```

• La **Différence** des relations $r_1$ et $r_2$ est la relation :

$$r_1 \setminus r_2 = \{e\,;\, e \in r_1 \text{ et } e \notin r_2\}$$

```
SELECT * FROM r1 EXCEPT SELECT * FROM r2
```

### 8.4.1 Produit cartésien, jointure et différence

• On considère deux relations $r_1$ et $r_2$ (n'ayant pas forcément le même schéma) ayant pour ensembles d'attributs $E_1$ et $E_2$.

• Le **Produit cartésien** des relations $r_1$ et $r_2$ est la relation ayant pour ensemble d'attributs $E_1 \cup E_2$ :

$$r_1 \times r_2 = \{e\,;\, e[E_1] \in r_1 \text{ et } e[E_2] \in r_2\}$$

• La **Jointure** des relations $r_1$ et $r_2$ par une condition $P$ est la relation, sous-ensemble de leur produit cartésien :

$$r_1 \bowtie_P r_2 = \{e\,;\, e[E_1] \in r_1 \text{ et } e[E_2] \in r_2 \text{ et } e \text{ vérifie } P\}$$

C'est le produit cartésien suivie de la sélection : $r_1 \bowtie_P r_2 = \sigma_P(r_1 \times r_2)$.

• Pour effectuer un produit cartésien : `SELECT * FROM table1, table2`

```
SELECT * FROM eleves, notes
```

• Un produit cartésien est rarement utile ; une jointure l'est beaucoup plus. Même si une jointure est théoriquement la composée d'un produit cartésien suivi d'une sélection, l'implémentation d'une jointure est beaucoup plus rapide qu'un produit cartésien suivi d'une sélection.

• Pour effectuer une jointure symétrique simple :

(Condition : égalité de certains attributs) :

```
SELECT * FROM table1 JOIN table2
    ON table1.attribut = table2.attribut
```

• Pour effectuer une jointure multiple :

```
SELECT * FROM table1 JOIN table2 JOIN table3 etc...
    ON conditions...
```

• Lorsque la jointure se fait sur les attributs de mêmes noms : (HORS-PROGRAMME)

```
SELECT * FROM table1 NATURAL JOIN table2
```

## 8.5 Exemples

### 8.5.1 Base de donnée `nobel.sqlite`

La base de donnée `nobel.sqlite` concerne les lauréats du prix nobel. Elle contient une seule table `nobel` ayant pour schéma de relation :

`nobel (année INT, sujet VARCHAR(15), laureat VARCHAR(50))`

**Question 1 :** Que pourrait-on choisir pour clé primaire ?

```
On ne pourrait choisir que le triplet : (annee,sujet,laureat).
```

**Question 2 :** Comment obtenir tous les lauréats du nobel de physique au XXe siècle et leur année d'obtention ?

**Réponse :** (Sélection)

```
SELECT laureat, annee FROM nobel WHERE annee > 1900 AND annee <= 2000
AND sujet = 'Physique'
```

**Question 3 :** Comment obtenir toutes les disciplines du nobel ?

**Réponse :** (Projection)

```
SELECT DISTINCT sujet FROM nobel
```

ou

```
SELECT sujet FROM nobel GROUP BY sujet
```

**Question 4 :** Comment de lauréats y-a-t-il eu dans chaque discipline ?

**Réponse :** (agrégation)

```
SELECT COUNT(*), sujet FROM nobel GROUP BY sujet
```

**Question 5 :** Sur quelle période s'étend la table ?

**Réponse :** (agrégation)

```
SELECT MIN(annee), MAX(annee) FROM nobel
```

**Question 6 :** Sur cette période combien d'année n'a été décerné aucun nobel ?

**Réponse :** (agrégation)

```
SELECT MAX(annee) - MIN(annee) + 1 - COUNT(DISTINCT annee)
        FROM nobel
```

**Question 7 :** Afficher tous les lauréats et pour chacun le nombre de nobels obtenus

**Réponse :** (agrégation)

```
SELECT COUNT(*) AS NbrePrix, laureat FROM nobel GROUP BY laureat
```

**Question 8 :** Afficher tous les lauréats ayant obtenus plusieurs nobels

**Réponse :** (requête composé et renommage)

```
SELECT Nbre, laureat FROM
(SELECT COUNT(*) AS Nbre, laureat FROM nobel GROUP BY laureat)
WHERE Nbre > 1
```

ou

```
SELECT laureat FROM nobel GROUP BY laureat HAVING COUNT(*) > 1
```

**Question 9 :** Afficher tous les lauréats ayant obtenus plusieurs nobels et leur discipline et année d'obtention, dans l'ordre d'année d'obtention.

**Réponse :** (jointure et requête composée et renommage)

```
SELECT nobel.laureat, sujet, annee FROM
(SELECT laureat AS supernobel FROM
(SELECT COUNT(*) AS Nbre, laureat FROM nobel GROUP BY laureat)
    WHERE Nbre > 1)
JOIN nobel on supernobel = nobel.laureat
    ORDER BY nobel.laureat
```

### 8.5.2 Base de donnée `movie.sqlite`

La base de donnée `movie.sqlite` contient 3 tables vérifiant les schémas suivants :

```
actor ( id INTEGER, name VARCHAR(35))
casting (movieid INTEGER, actorid INTEGER, ord INTEGER)
movie (id INTEGER, title VARCHAR(70), yr DECIMAL(4), score FLOAT,
        votes INTEGER, director INTEGER)
```

Elle porte sur tous les films de cinema, leur casting, acteurs et réalisateurs jusqu'à l'année 2000. La table `actor` contient les acteurs mais aussi les réalisateurs.

**Question 1.** Comment obtenir les titres des 10 films les mieux notés de l'année 1990 ?

**Réponse :** (sélection avec `ORDER BY` et `LIMIT`)

```
SELECT title FROM movie WHERE yr = 1990 ORDER BY score DESC LIMIT 10
```

**Question 2.** Comment obtenir les titres films ayant un score supérieur au score moyen ?

**Réponse :** (sélection avec agrégation et requête composée)

```
SELECT title FROM movie
    WHERE score >= (SELECT AVG(score) FROM movie)
```

ou sélection avec agrégation et `HAVING`

```
SELECT title FROM movie HAVING score >= AVG(score)
```

**Question 3.** Quels sont les (identifiants des) 10 réalisateurs ayant obtenu la meilleure note moyenne pour leurs films ?

**Réponse :** (sélection avec agrégation et requête composée)

```
SELECT director FROM
    (SELECT director, AVG(score) AS moyenne FROM movie
    GROUP BY director ORDER BY moyenne DESC LIMIT 10)
```

**Question 4.** Quels sont les noms des 10 réalisateurs ayant obtenu la meilleure note moyenne pour leurs films ?

**Réponse :** (jointure avec actor et la table précédente)

```
SELECT name FROM actor JOIN
    (SELECT director, AVG(score) AS moyenne FROM movie
    GROUP BY director ORDER BY moyenne DESC LIMIT 10)
ON actor.id = director
```

**Question 5.** Comment obtenir le casting du film Star Wars ?

**Réponse :** (jointure à 3 tables)

```
SELECT actor.name FROM actor JOIN casting JOIN movie
    ON actor.id = casting.actorid
    AND movie.id=casting.movieid
    AND movie.title="Star Wars"
```

**Question 6.** Comment obtenir le nom des acteurs ayant joué dans Star Wars 2 mais pas dans Star Wars ?

**Réponse :** (Différence sur 2 jointure à 3 tables)

```
SELECT actor.name FROM actor JOIN casting JOIN movie
    ON actor.id = casting.actorid
    AND movie.id=casting.movieid
    AND movie.title="Star Wars2"
EXCEPT
SELECT actor.name FROM actor JOIN casting JOIN movie
    ON actor.id = casting.actorid
    AND movie.id=casting.movieid
    AND movie.title="Star Wars"
```

**Question 7.** Comment obtenir le nom des acteurs ayant joué dans au moins un Star Wars ?

**Réponse :**

```
SELECT actor.name FROM actor JOIN casting JOIN movie
    ON actor.id = casting.actorid
    AND movie.id=casting.movieid
    AND movie.title IN ("Star Wars","Star Wars 2","Star Wars 3")
    GROUP BY actor.name
```