Chapitre $\underline{1111}_2$
Bases de données et langage SQL

17 juillet 2018

1 Le modèle logique (MLD)

On a représenté des données par des tables. Voici par exemple celle des films.

titre	date
Gran Torino	2008
The good, the Bad and the Ugly	1966
Study in Pink	2010
Schindler's List	1993
Dr Strangelove	1964
Invictus	2009

Au départ, on a identifié les films par leur titre. On dit que le champ titre de la table «films» est une *clé primaire* pour signifier que :

— tout film a un titre ;

— pour un titre donné, il y au plus un film ayant ce titre ;

— on fera référence à tout film par son titre.

Une clé primaire peut être constituée de plusieurs champs. Exemple pour la table «personnes» sans identifiants : le couple (nom, prénom). On dit qu'il s'agit d'une clé primaire *composée*.

nom	prénom	datenaissance
Kubrick	Stanley	1928
Spielberg	Steven	1946
Eastwood	Clint	1930
Cumberbatch	Benedict	1976
Freeman	Martin	1971
Leone	Sergio	1929
McGuigan	Paul	1963
Sellers	Peter	1925

On a vu qu'il est préférable d'utiliser des identifiants pour faire référence à une personne. La clef primaire est alors naturellement cet identifiant.

id	nom	prénom	datenaissance
1	Kubrick	Stanley	1928
2	Spielberg	Steven	1946
3	Eastwood	Clint	1930
4	Cumberbatch	Benedict	1976
5	Freeman	Martin	1971
6	Leone	Sergio	1929
7	McGuigan	Paul	1963
8	Sellers	Peter	1925

Dans la table «joue» (sans identifiants), on fait référence à ces clés. On dit que les champs (nom, prénom) de la table «joue» sont une *clé étrangère* car ils font référence à une clé primaire de la table «personnes».

nom	prenom	titre	nom (de personnage)
Eastwood	Clint	The good, the Bad and the Ugly	Blondie
Eastwood	Clint	Gran Torino	Walt Kowalski
Cumberbatch	Benedict	Study in Pink	Sherlock Holmes
Freeman	Martin	Study in Pink	Dr John Watson
Selers	Peters	Dr Strangelove	Dr Strangelove
Selers	Peters	Dr Strangelove	Group Capt. Lionel Mandrake
Selers	Peters	Dr Strangelove	President Merkin Muffley

Exercice 1.0.1. Voici la table de la relation «joue», avec les identifiants.

idacteur	idfilm	idpersonnage
3	2	2
3	1	1
4	3	3
5	3	4
8	5	5
8	5	6
8	5	7

Que peut-on considérer comme clef primaire ? Y a-t-il une ou des clef étrangères ?

2 Vers le modèle physique (MPD)

Pour implanter cette base de données on peut choisir n'importe quel langage de programmation.

Mais tous les langages ne sont pas aussi adaptés à la modélisation et l'interrogation des données.

2.1 SQL

Nous allons étudier le langage **SQL**[1], pour : *Standard Query Language*.
Notamment

— C'est un langage de définition de données (LDD ou DDL), pour définir le format des données.

— C'est un langage de manipulation de données (LMD ou DML) pour ajouter ou supprimer des données.

— Il possède une instruction SELECT pour effectuer des requêtes (recherches) sur les données.

C'est **le** standard actuel (décliné en différentes versions), implanté dans les systèmes de gestion de bases de données.

2.2 Définition de tables

Nous allons utiliser SQLite, un outil très dépouillé pour la manipulation d'une base de données.

En SQlite, une base de données est stockée dans un fichier (et tout fichier contient une unique base de données).

1. Prononcer comme le mot anglais *sequel*.

2.2.1 Lancement de SQLite

En ligne de commande, taper

`sqlite3` *f*

pour lancer l'interface (textuelle) de sqlite3.
Cette commande :

— ouvre la base de données contenue dans le fichier *f* ;

— crée cette base si ce fichier n'existe pas.

2.2.2 Table «personne»

On va créer une table «personne» avec 4 champs :

— `id`

— `nom`

— `prenom`

— `datenaissance`

Il reste à préciser le *type* des données que contiendront ces champs.

2.2.3 Types de données en SQL

SQL permet de nombreux types de données, dont :

VARCHAR(n**)** (une chaîne de caractères d'au plus n caractères) ;

INT ou INTEGER (entier) ;

REAL (nombre flottant) ;

DECIMAL(p**,** s**)** (décimal sur p chiffres, dont s après la virgule) ;

DATE (une date, par exemple : 2016-02-29)

TIME (une heure de la journée, par exemple 15 :43 :07, précision en général $10^{-7}s$) ;

DATETIME (couple des deux précédents, par exemple 2016-02-29 15 :43 :07).

2.3 Définition des tables

La définition de tables est l'objet de la partie de SQL appelée DDL (*Data Definition Language*). Les deux opérations les plus importantes sont :

CREATE pour créer une table ;

DROP pour effacer une table («DROP TABLE *nomtable* ;»).

On les utilise de la manière suivante.

```
CREATE TABLE PERSONNE (
  id INTEGER,
  nom VARCHAR(50) NOT NULL,
  prenom VARCHAR(50) NOT NULL,
  datenaissance DATE,
  PRIMARY KEY(id));
```

On fera attention à

— ne pas oublier de terminer les commandes par des « ;»

— noter la déclaration de la clé primaire.

On crée de même la table «personnage» :

```
CREATE TABLE PERSONNAGE (
  id INTEGER,
  nom VARCHAR(50) NOT NULL,
  PRIMARY KEY(id)
);
```

et la table «joue»

```
CREATE TABLE JOUE (
  idacteur INTEGER,
  idfilm INTEGER,
  idpersonnage INTEGER,
  PRIMARY KEY(idacteur, idfilm, idpersonnage),
  FOREIGN KEY(idacteur) REFERENCES PERSONNE,
  FOREIGN KEY(idfilm) REFERENCES FILM,
  FOREIGN KEY(idpersonnage) REFERENCES PERSONNAGE
);
```

Notez bien les déclarations de clés étrangères. Enfin, on crée la table « film ».

```
CREATE TABLE FILM (
  id INTEGER,
  titre VARCHAR(50) NOT NULL,
  date DATE,
  idrealisateur INTEGER,
  PRIMARY KEY(id),
  FOREIGN KEY(idrealisateur) REFERENCES PERSONNE
);
```

2.4 Manipulation des tables

La manipulation de tables est l'objet de la partie de SQL appelée DML (*Data Manipulation Language*). Voici quelques actions possibles.

2.4.1 Insertions de valeurs

On peut insérer une donnée :

```
INSERT INTO PERSONNE (nom, prenom, datenaissance)
  VALUES('Kubrick', 'Stanley', date('1928-07-26'));
```

On remarquera que l'on n'a pas donné de valeur pour la variable `id`.

— Pour une clé primaire entière, la base de données donne une valeur si on ne la fournit pas.

— On peut indiquer lors de la création de la table une valeur par défaut pour les champs manquants, sinon la valeur `NULL` est mise par défaut, sauf si on l'a interdit lors de la création de la table.

Exemple 2.4.1. La commande

```
INSERT INTO PERSONNE (prenom, nom)
  VALUES ('Clint', 'Eastwood');
```

est accepté (pas de date de naissance renseignée), alors que

```
INSERT INTO PERSONNE (nom) VALUES ('Spielberg');
```

déclenche l'erreur suivante.

```
Error: PERSONNE.prenom may not be NULL
```

Remarque 2.4.2. Il n'y a pas besoin de mettre les champs dans le même ordre que dans la déclaration de la table.

2.4.2 Modification de valeurs

On peut modifier une valeur déjà renseignée.

```
UPDATE PERSONNE
SET datenaissance = date('1930-05-31')
WHERE prenom = 'Clint' AND nom = 'Eastwood';
```

2.4.3 Suppression de valeurs

On peut aussi supprimer des entrées de la base de données.

```
DELETE FROM PERSONNE WHERE prenom='Clint';
```

2.4.4 Vérification des contraintes de clés primaires

Lorsque l'on ajoute une valeur dans une nouvelle base et que l'on veut spécifier la clef primaire, cette dernière ne doit bien évidemment pas avoir déjà été affectée.

```
INSERT INTO PERSONNE (id, prenom, nom)
  VALUES (3, 'Steven', 'Spielberg');
INSERT INTO PERSONNE (id, prenom, nom)
  VALUES (3, 'Benedict', 'Cumberbatch');
```

La première insertion acceptée, la deuxième donne :

```
Error: PRIMARY KEY must be unique
```

2.5 Requêtes

Les requêtes SQL sont introduites par le mot-clé `SELECT`.

Une requête renvoie une liste de n-uplets, que l'on peut identifier à une table. De très nombreuses variantes de cette requête sont possibles, mais elles ont toutes en commun la structure suivante.

```
SELECT col
FROM t ;
```

qui renvoie les colonnes désignées par `col` de la table `t`. On remarquera que `t` peut être le résultat d'une autre requête, c'est-à-dire que l'on pourra avoir des constructions du type

```
SELECT col
FROM (SELECT colbis
      FROM [...]
);
```

Enfin, on pourra renommer les colonnes (on dit qu'on leur donne un *alias*) par la commande `AS` :

```
SELECT col AS alias
FROM t ;
```

2.5.1 Lister tous les éléments d'une table

On dispose d'un symbole *joker* `*`, qui désigne toutes les colonnes d'une table.

```
SELECT *
FROM personne ;
```

id	nom	prenom	datenaissance
1	Kubrick	Stanley	1928-07-26
2	Spielberg	Steven	1946-12-18
3	Eastwood	Clint	1930-05-31
4	Cumberbatch	Benedict	1976-07-19
5	Freeman	Martin	1971-09-08
6	Leone	Sergio	1929-01-03
7	McGuigan	Paul	1963-09-19
8	Sellers	Peter	1925-09-08

2.5.2 Lister certaines colonnes d'une table

```
SELECT nom, prenom
FROM PERSONNE ;
```

nom	prenom
Kubrick	Stanley
Spielberg	Steven
Eastwood	Clint
Cumberbatch	Benedict
Freeman	Martin
Leone	Sergio
McGuigan	Paul
Sellers	Peter

2.5.3 Lister certaines lignes d'une table.

Si on sait que Clint Eastwood est le réalisateur no 3 :

```
SELECT titre, date
FROM FILM
WHERE idrealisateur = 3;
```

titre	date
Gran Torino	2008
Invictus	2009

Et sinon, on utilise une requête *croisée*, sur les tables « film » et « personne » :

```
SELECT FILM.titre, FILM.date
FROM FILM, PERSONNE
WHERE idrealisateur = PERSONNE.id
      AND
      nom = 'Eastwood' ;
```

titre	date
Gran Torino	2008
Invictus	2009

2.5.4 Calcul de jointure.

Dans l'exemple précédent, on construit le *produit cartésien* des tables `FILM` et `PERSONNE`, écrit comme la table `FILM, PERSONNE`. Cela revient, pour chaque ligne de la table `FILM`, de lui concaténer chaque ligne de la table `PERSONNE`. Cela crée une table ayant autant de lignes que le produit des nombre de lignes des tables `FILM` et `PERSONNE`.

L'instruction précédente ne conserve dans cette table produit que les lignes pour lesquelles les identifiants du réalisateur du film et celui de la personne coïncident. On a donc concaténé à chaque ligne de la table `FILM` la ligne de la table `PERSONNE` du réalisateur de ce film. On dit que l'on a réalisé une *jointure* de ces deux tables, selon le critère `idrealisateur = PERSONNE.id`.

On peut reformuler la requête précédente en utilisant l'instruction spécifique `JOIN ON`.

```
SELECT titre, date
FROM FILM
JOIN PERSONNE ON idrealisateur = PERSONNE.id
WHERE nom = 'Eastwood' ;
```

titre	date
Gran Torino	2008
Invictus	2009

Remarque 2.5.1. On peut mettre tout type de clause booléenne à droite du `ON`, même si seul l'exemple présenté est au programme.

On peut aussi mettre plusieurs instructions `JOIN ON`, qui partent toutes d'une même table centrale.

2.5.5 Ensembles ou listes ?

Commençons par un exemple.

```
SELECT nom, prenom
FROM PERSONNE, JOUE
WHERE id = idacteur;
```

nom	prenom
Eastwood	Clint
Eastwood	Clint
Cumberbatch	Benedict
Freeman	Martin
Sellers	Peter
Sellers	Peter
Sellers	Peter

`SELECT` ne renvoie pas un ensemble de n-uplets mais en fait une liste de n-uplets : il peut y avoir des doublons. On peut les supprimer :

```
SELECT DISTINCT nom, prenom
FROM PERSONNE, JOUE
WHERE id = idacteur;
```

nom	prenom
Eastwood	Clint
Cumberbatch	Benedict
Freeman	Martin
Sellers	Peter

2.5.6 Tri des résultats.

On peut trier la liste renvoyée (par ordre alphabétique sur les chaînes de caractères, temporel sur les dates).

On peut trier par ordre croissant avec le mot clef `ASC`.

```
SELECT DISTINCT nom, prenom
FROM PERSONNE, JOUE
WHERE id = idacteur
ORDER BY nom ASC, prenom ASC;
```

nom	prenom
Cumberbatch	Benedict
Eastwood	Clint
Freeman	Martin
Sellers	Peter

On peut trier par ordre décroissant avec le mot clef `DESC`.

```
SELECT *
FROM PERSONNE
ORDER BY datenaissance DESC;
```

id	nom	prenom	datenaissance
4	Cumberbatch	Benedict	1976-07-19
5	Freeman	Martin	1971-09-08
7	McGuigan	Paul	1963-09-19
2	Spielberg	Steven	1946-12-18
3	Eastwood	Clint	1930-05-31
6	Leone	Sergio	1929-01-03
1	Kubrick	Stanley	1928-07-26
8	Sellers	Peter	1925-09-08

2.5.7 Requêtes agrégats.

On peut compter le nombre de lignes d'une table par la fonction `COUNT`. Étant donné une colonne, on peut facilement calculer des moyennes (fonction `AVG`), obtenir le maximum (`MAX`) et le minimum (`MIN`), ou bien une somme (`SUM`).

```
SELECT COUNT(*) AS nbfilms
FROM FILM;
```

nbfilms
6

```
SELECT MIN(date) AS 'date premier film'
FROM FILM;
```

date premier film
1964

On peut calculer des cumuls. Par exemple, le nombre de films que les différents réalisateurs ont tournés :

```
SELECT nom, prenom, COUNT(*) as nbfilms
FROM PERSONNE, FILM
WHERE idrealisateur = PERSONNE.id
GROUP BY PERSONNE.id;
```

nom	prenom	nbfilms
Kubrick	Stanley	1
Spielberg	Steven	1
Eastwood	Clint	2
Leone	Sergio	1
McGuigan	Paul	1

Pour compter dans combien de films chaque acteur a joué :

```
SELECT nom, prenom, COUNT(*)
FROM PERSONNE, JOUE
WHERE PERSONNE.id=idacteur
GROUP BY PERSONNE.id;
```

nom	prenom	COUNT(*)
Eastwood	Clint	2
Cumberbatch	Benedict	1
Freeman	Martin	1
Sellers	Peter	3

Aïe, ce n'est clairement pas ce qu'on voulait (Peter Sellers n'a joué que dans un film de notre base de données). Pourquoi ?

Façon possible de corriger (compliquée)

```
SELECT nom, prenom, COUNT(*) AS nbfilms
FROM PERSONNE, FILM
WHERE EXISTS (SELECT *
              FROM joue
              WHERE PERSONNE.id=idacteur
                    AND
                    FILM.id=idfilm
)
GROUP BY PERSONNE.id ;
```

nom	prenom	nbfilms
Eastwood	Clint	2
Cumberbatch	Benedict	1
Freeman	Martin	1
Sellers	Peter	1

La condition `WHERE` porte sur les données qu'on va ensuite agréger. On peut aussi mettre des conditions sur les agrégats à retenir :

```
SELECT nom, prenom, COUNT(*) AS nbroles
FROM personne, joue
WHERE personne.id=idacteur
GROUP BY personne.id
HAVING nbroles >= 2;
```

nom	prenom	nbroles
Eastwood	Clint	2
Sellers	Peter	3

2.5.8 Opérations sur les tables.

On peut faire l'union de deux tables avec la commande `UNION`, l'intersection de deux tables avec la commande `INTERSECT` et la différence (au sens ensembliste) de deux tables avec la commande `EXCEPT`. Tout ceci permet d'écrire plus simplement des requêtes, qui demanderaient des écritures lourdes à base de `WHERE`, `AND`, `OR` *etc.*

Par exemple, de manière quelque peu factice, on peut vouloir obtenir la liste des personnes nées après 1950 ou dont l'identifiant est inférieur ou égal à 4.

```
SELECT *
FROM PERSONNE
WHERE datenaissance >= '1950-01-01'

UNION

SELECT *
FROM PERSONNE
WHERE id <=4
;
```

id	nom	prenom	datenaissance
1	Kubrick	Stanley	1928-07-26
2	Spielberg	Steven	1946-12-18
3	Eastwood	Clint	1930-05-31
4	Cumberbatch	Benedict	1976-07-19
5	Freeman	Martin	1971-09-08
7	McGuigan	Paul	1963-09-19

De même, on peut obtenir la liste des personnes nées après 1950 et dont l'identifiant est inférieur ou égal à 4.

```
SELECT *
FROM PERSONNE
WHERE datenaissance >= '1950-01-01'

INTERSECT

SELECT *
FROM PERSONNE
WHERE id <=4
;
```

id	nom	prenom	datenaissance
4	Cumberbatch	Benedict	1976-07-19

Enfin, on peut obtenir la liste des personnes nées après 1950, exceptées celles dont l'identifiant est inférieur ou égal à 4.

```
SELECT *
FROM PERSONNE
WHERE datenaissance >= '1950-01-01'

EXCEPT

SELECT *
FROM PERSONNE
WHERE id <=4
;
```

id	nom	prenom	datenaissance
5	Freeman	Martin	1971-09-08
7	McGuigan	Paul	1963-09-19

2.6 Annexes.

Cette partie ne sera pas traitée en cours.

2.6.1 Calcul du lieu d'un maximum, d'un minimum.

C'est une question assez fréquente. On peut par exemple chercher à déterminer le ou les acteurs les plus âgés (*i.e.* qui possède la date de naissance minimale.

Une première requête imbriquée naïve.

```
SELECT *
FROM PERSONNE
WHERE datenaissance = (SELECT min(datenaissance)
                       FROM PERSONNE
                       )
;
```

id	nom	prenom	datenaissance
8	Sellers	Peter	1925-09-08

Sous la condition qu'il n'y a qu'un enregistrement réalisant ce minimum, on pourra aussi utiliser l'idée suivante : il suffit de trier la table par dates de naissances, puis de conserver uniquement le premier enregistrement (fonction `LIMIT`, on pourra aussi utiliser `OFFSET k` pour passer les `k` premières lignes).

```
SELECT *
FROM PERSONNE
ORDER BY datenaissance ASC
LIMIT 1
;
```

id	nom	prenom	datenaissance
8	Sellers	Peter	1925-09-08

2.6.2 Compter des enregistrements différents.

On peut essayer de compter le nombre d'enregistrements ayant des valeurs distinctes. Par exemple : combien d'acteurs ont joué dans les films de notre base ?

Commençons part une requête imbriquée naïve.

```
SELECT COUNT(*) AS nbacteurs
FROM (SELECT DISTINCT idacteur
      FROM JOUE
      )
;
```

nbacteurs
4

On peut aussi écrire cela comme suit.

```
SELECT COUNT(DISTINCT idacteur) AS nbacteurs
FROM JOUE
;
```

nbacteurs
4

2.6.3 Opérations sur les nombres et les chaînes de caractères.

On peut réaliser des opérations arithmétiques sur les nombres : + pour l'addition, - pour la soustraction *etc.*

Par exemple, pour l'addition.

```
SELECT idacteur + idfilm + 42 AS total
FROM JOUE
;
```

total
47
46
49
50
55
55
55

Attention, comme en Python, les opérations sur les entiers donnent des résultats entiers.

```
SELECT idacteur /2 AS quotient, idacteur/2 AS reste
FROM JOUE
;
```

quotient	reste
1	1
1	1
2	2
2	2
4	4
4	4
4	4

Pour effectuer des divisions flottantes (par exemple, pour calculer ici un pourcentage), on pourra utiliser une fonction de conversion ou bien ruser avec la division entière.

```
SELECT CAST(idacteur AS FLOAT)/100 , (1.0*idacteur)/100
FROM JOUE
;
```

CAST(idacteur AS FLOAT)/100	(1.0*idacteur)/100
0.03	0.03
0.03	0.03
0.04	0.04
0.05	0.05
0.08	0.08
0.08	0.08
0.08	0.08

On pourra mettre des chaînes de caractères en minuscules ou en majuscules.

```
SELECT UPPER(prenom), LOWER(prenom)
FROM PERSONNE
;
```

UPPER(prenom)	LOWER(prenom)
STANLEY	stanley
STEVEN	steven
CLINT	clint
BENEDICT	benedict
MARTIN	martin
SERGIO	sergio
PAUL	paul
PETER	peter

Cette liste n'est bien entendu pas exhaustive.

3 Exercices

Donner des requêtes pour :

— trouver le nombre d'acteurs qui ont joué au moins deux rôles ;

— trouver les acteurs qui sont aussi des réalisateurs ;

— trouver les acteurs nés le 1er janvier 1930 ou après ;

— compter les acteurs nés avant le 1er janvier 1940 ;

— trouver les acteurs qui ont joué au moins dans un film où jouait Martin Freeman ;

— trouver les réalisateurs qui ont dirigé Clint Eastwood.