Bases de Données
(SQL, NoSQL)

Amira Mouakher

Université de Perpignan Via Domitia

2023 - 2024

Plan du cours

1. Introduction
   - Présentation du cours
   - Motivations et premières définitions
2. Modèle relationnel
   - Relations et n-uplets
   - Clés, dépendances et normalisation
3. Exemples de schémas normalisés
   - La base des voyageurs
   - La base des films

Plan

Présentation du cours

- **Compétences :** Concevoir, implémenter et exploiter des bases de données.
- **Programme :**
  1. Modélisation : modèle Entité-Association, modèle relationnel, règles de passage entre les deux modèles.
  2. Algèbre relationnelle.
  3. Utilisation d'un serveur de gestion de base de données (MariaDB).
  4. Langage de description de données de SQL.
  5. Interrogation de la base pour la recherche d'information.
  6. Initiation aux déclencheurs.
  7. Bases de données NoSQL.
  8. Initiation à MongoDB, Neo4J.

Présentation du cours

- **Volume horaire :** 36h
  - 15h de CM (10×1h30)
  - 21h de TP (14×1h30)
- **Évaluation :** max $\{(CC + CT)/2, CT\}$
  - CC : 1 épreuve à mi-semestre + interrogations + rendus facultatifs.
  - CT : 1 épreuve en fin de semestre
- **Références :**
  - Bases de données, Jean-Luc Hainaut, 4ème édition.
  - Cours de bases de données - Modèles et langages, Philippe Rigaux `http://sql.bdpedia.fr/files/cbd-sql.pdf`.
  - Bases De Données Nosql Et Big Data - Concevoir Des Bases De Données Pour Le Big Data : Cours Et Travaux Pratiques - Lacomme Philippe

Introduction

La démarche classique consiste à réaliser pour un type d'information, un programme d'application.

- Une entreprise doit conserver un volume élevé d'information :
- noms, adresses, salaire, adresse des fournisseurs, quantités, prix des items, bilan financier, etc.
- Ces informations se retrouvent dans différents systèmes de traitement de fichiers.
- Système de gestion des stocks, système de facturation, système de préparation de paie, programme de gestion de personnel, etc.

Motivations et premières définitions

Exemple d'un système décentralisé :

- Pour obtenir une information, l'employé doit :
  1. Déterminer le système à consulter ;
  2. Trouver la bonne personne concernée.
- Perte de temps.
- De plus, certaines informations sont souvent conservées en plusieurs endroits.
- Duplication de données.
- Gaspillage au niveau du volume de fichiers.

Motivations et premières définitions

Résumons le système décentralisé...

- Principaux problèmes de ce système :
  1. Redondance de certaines informations ;
  2. Incapacité de répondre rapidement aux demandes d'information provenant de fichiers multiples ;
  3. Coûts élevés pour les modifications (plusieurs systèmes).
- Avec le temps, il y aura un accroissement inutile de :
  1. L'ensemble des fichiers ;
  2. La taille des fichiers ;
  3. Les temps d'accès.
- Code développé par différents programmeurs et écrits dans différents langages.
- Formats de fichiers différents.
- Inconsistance des données.

Motivations et premières définitions

Parmi les inconvénients des système de traitement de fichiers :

Redondance et inconsistance des données

- Informations identiques répliquées dans plusieurs fichiers.
  - Ex : Institution financière
  - Adresse et téléphone d'un employé
  - dans le fichier du système de paie
  - dans le fichier de gestion du personnel
- Accroissement inutile (taille des fichiers, temps d'accès)
- Risque d'inconsistance des données si le changement d'adresse ne s'effectue pas dans les deux fichiers.

Motivations et premières définitions

Difficulté d'accès aux données :

- Il faut un programme spécifique pour toute nouvelle demande d'information.
  - Ex : Estimation pour l'augmentation de 10% sur le prix des items vendus du mois dernier.
- Le temps d'accès à une requête non prévue peut être très long.

Multiplicité des remises à jour :

- Les traitements concurrents peuvent générer des erreurs.
  - Ex : Mises à jour d'un compte en même temps
- Le temps d'accès à une requête non prévue peut être très long.
- Solde de 400 $ (T1 : dépôt de 300 $, T2 : retrait de 500 $)
- Nécessite un programme superviseur pour gérer les transactions.

Motivations et premières définitions

Sécurité :

- La sécurité des données et les accès non-autorisés ne sont pas garanties.
  - Ex : Le personnel ne devrait pas avoir accès au programme de paie.

Intégrité des données :

- Difficulté d'imposer des contraintes.
  - Ex : Le solde ne doit jamais être inférieur à 0.

Solution

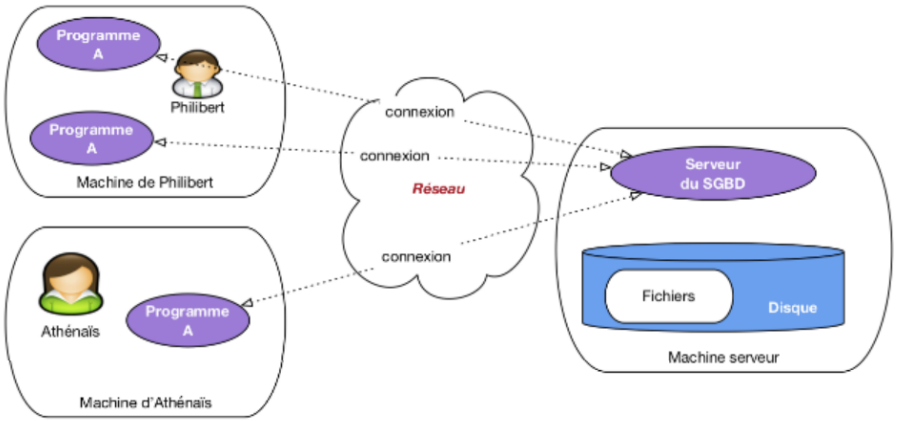
Une banque de données commune, entièrement centralisée.

Motivations et premières définitions

- Une **donnée** est une valeur numérisée décrivant un fait ou une mesure.
  - chaînes de caractères, nombres, dates
- Les données sont toujours associées à un contexte permettant de savoir quelle information elle représente.
  - nom d'un écrivain, nombre de livres vendus, date de naissance
- Une base de données (BDD) est un ensemble de données organisées par une structure pré-définie au moment de la conception.
  - les BDD sont stockées sur des supports "persistants" – i.e. qui préservent leur contenu même en l'absence d'alimentation électrique
- Un fichier de BDD a forcément une structure permettant de distinguer les différentes données et de décrire les liens entre celles-ci.

Motivations et premières définitions

- De manière générale, une BDD doit avoir les fonctionnalités suivantes :
  - Pouvoir accéder aux fichiers de la base et garantir leur intégrité
  - Gérer les accès concurentiels et donc les opérations concurrentes
  - Optimisation des recherches et des mises à jour
  - Pouvoir interagir avec des applications et des utilisateurs non spécialistes via un langage de haut niveau.
- En pratique on utilise un Système de Gestion de Bases de Données (SGBD) qui permettra d'obtenir toutes ces fonctionnalités
  - Seul le SGBD aura accès à la BDD
  - Les utilisateurs se connecteront seulement au SGBD



Motivations et premières définitions

- Grâce à cette architecture les programmes clients n'ont pas besoin de savoir sous quelle forme physique sont codées et stockées les données.
- Les programme clients ont besoin de
  - savoir comment sont organisées les données d'un point de vue logique
  - pouvoir modifier/interroger la BDD sans se préoccuper des structures de données ou du type de mémoire utilisée.
- En résumé :
  - Le client s'intéresse seulement aux aspects logiques de la BDD
  - Le serveur s'intéresse aux aspects physiques de la BDD
  - Le SGBD gère les interactions entre les clients et le serveur

Motivations et premières définitions

- Les données d'une BDD sont organisées sous la forme d'un modèle logique afin de :
  - structurer les différentes données
  - représenter les relations entre les différentes données
  - développer un langage afin d'interagir avec les données
- Les BDD sont généralement organisées dans un modèle **relationnel** qui est construit sur des bases formelles rigoureuses.
- Deux langages complémentaires basés sur ce modèle ont été définis :
  - un langage déclaratif $\longrightarrow$ permet de spécifier le résultat que l'on souhaite obtenir sans tenir compte des opérations nécessaires
  - un langage procédural $\longrightarrow$ un ensemble d'opérations permettant d'obtenir le résultat souhaité
- Le langage SQL, réunit les deux approches. Il est utilisé depuis les années 1970 dans tous les systèmes relationnels.

Première partie du cours

Conception et utilisation d'une BDD

- Modèle relationnel (représentation des données sous forme de table)
- Langage SQL sous ses deux formes (déclarative et procédurale)
- Modèle Entité-Association et conversion vers le modèle relationnel

Plan

Relations et n-uplets

- Le produit cartésien de deux ensembles A et B, noté $A \times B$, est l'ensemble de toutes les paires possibles (a, b) avec $a \in A$ et $b \in B$.
- Exemple $A = \{0, 1, 2\}$ et $B = \{\text{'a'}, \text{'b'}\}$

$$A \times B = \{(0, \text{'a'}), (0, \text{'b'}), (1, \text{'a'}), (1, \text{'b'}), (2, \text{'a'}), (2, \text{'b'})\}$$

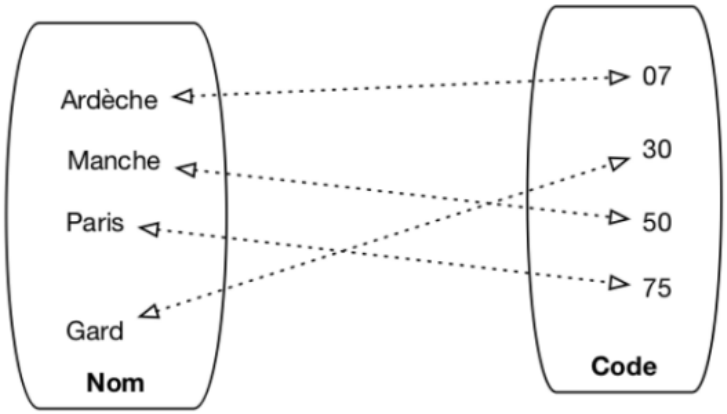
- On peut généraliser le produit cartésien à n ensembles en définissant $E_1 \times \cdots \times E_n$ comme l'ensemble des n-uplets $(e_1, \cdots, e_n)$ avec $e_i \in E_i$ pour tout $i \in [1, n] \cap \mathbb{Z}$

Définition (Relation)

- Soit $n \in \mathbb{N}^*$, une **relation** sur des ensembles D_1, D_2, ..., D_n est un sous ensemble **fini** de $D_1 \times D_2 \times \ldots D_n$.
- L'entier n est appelé le **degré** ou la **dimension** de la relation.

Relations et n-uplets

- Une relation peut être représentée de plusieurs manières
  - sous forme de graphe



  - sous forme de table

Nom	Code
Ardèche	07
Gard	30
Manche	50
Paris	75

- Dans une base relationnelle, on utilisera toujours la représentation d'une relation sous forme de table.

Relations et n-uplets

- Un élément d'une relation de dimension n est un n-uplet $(e_1,...e_n)$.
- Dans la représentation par table, un n-uplet correspond à une ligne.
- La définition d'une relation comme un ensemble (au sens mathématique) a quelques conséquences :
  - L'ordre des éléments n'est pas important car il n'y a pas forcément de relation d'ordre dans un ensemble.
    - ⋆ le résultat d'une requête ne dépend pas de l'ordre des lignes.
  - Il ne peut y avoir deux fois le même élément car il n'y a pas de doublons dans un ensemble
  - Il n'y a pas de champs vide
    - ⋆ dans une relation les valeurs de chaque attributs sont connues.
- Nous verrons qu'en pratique ces contraintes ne sont pas toujours respectées.

Relations et n-uplets

- Pour des raisons pratiques on associe à chaque dimension un nom que l'on appelle **attribut**.
- Une relation peut donc être décrite par :
  - Le nom de la relation $\mathcal{R}$
  - L'attribut de chaque dimension noté A_i.
    - ⋆ pour éviter les redondances les attributs doivent être distincts deux à deux
  - Le type de valeur de chaque dimension (int, string, etc. . .) noté D_i
- On écrit de manière concise $\mathcal{R}(A_1 : D_1, ..., A_n : D_n)$. Cette notation est appelée le schéma de la relation.
  - La relation précédente concernant les départements est schématisée par `Département (nom: string, code : string)`
  - En pratique on omet souvent de préciser le type des attributs

Relations et n-uplets

- Dans une BDD, les valeurs des attributs doivent être élémentaires
  $\rightarrow$ pas de type composé (liste, sous-relation, etc...)
- Une base bien formée suit des règles de normalisation.

Définition (première forme normale)

Une relation est en **première forme normale** si toutes les valeurs d'attributs sont connues, atomiques et si elle ne contient aucun doublon.

Nom	Code
Ardèche	07
Pyrénées Orientales	
Gard	30
Manche	50
Paris	75

Nom	Code
Ardèche	07
Gard	30
Manche	50
Paris	75
Gard	30

Exemples de relation qui ne sont pas en première forme normale

Clés, dépendances et normalisation

- Considérons une base de données, formée d'une seule relation, et qui contient des informations sur des films,

titre	année	prénom du réalisateur	nom du réalisateur	année de naissance
Alien	1979	Ridley	Scott	1943
Psychose	1960	Alfred	Hitchcock	1899
Spider-Man	2002	Sam	Raimi	1959
Volte-face	1997	John	Woo	1946
Vertigo	1958	Alfred	Hitchcock	1899
Titanic	1997	James	Cameron	1954

- Ce format de BDD entraine plusieurs problème potentiels
  - l'information sur les réalisateurs est redondante
    - ★ problème potentiel de mise à jour (année de naissance)
  - un réalisateur est supprimé dès que l'on supprime un film
  - impossible d'ajouter un film si l'on ne connait pas son réalisateur

Clés, dépendances et normalisation

- Le schéma de relation précédent contient plusieurs défaut qui vont compliquer la gestion de la base de données
  - nécessité de faire plus que juste énumérer les attributs
- En pratique une base de données sera composée de plusieurs relations et donc de plusieurs tables
- Le modèle relationnel va nous permettre d'évaluer la "qualité" d'un schéma de relation normalisé
  - nous verrons plus tard comment normaliser un schéma
- Afin d'élaborer les différentes contraintes qui doivent régir un schéma il faut se poser la question de la dépendance des différents attributs.

Clés, dépendances et normalisation

Définition (dépendance fonctionnelle)

Soit $\mathcal{R}$ un schéma de relation, $\mathcal{S}$ un sous-ensemble d'attributs de $\mathcal{R}$ et A un attribut de $\mathcal{R}$.

- A **dépend fonctionnellement** de $\mathcal{S}$ lorsque pour toute paire (e_1, e_2) d'éléments de $\mathcal{R}$ si e_1 et e_2 sont égaux sur $\mathcal{S}$ alors ils ont la même valeur pour l'attribut A.
- Dans ce cas on note $\mathcal{S} \to A$ et on utilise la notation $\mathcal{S} \to A, B$ pour abbréger $\mathcal{S} \to A$ et $\mathcal{S} \to B$.

- Informellement on peut dire que $\mathcal{S} \to A$ si "la valeur de $\mathcal{S}$ détermine la valeur de A".
- Pour `Personne(nom, prénom, noSS, dateNaissance, email)` on a :
  - `noSS` $\to$ `nom, prénom, dateNaissance, email`
  - `email` $\to$ `nom, prénom, dateNaissance, noSS`

Clés, dépendances et normalisation

- Donnez la liste des dépendances fonctionnelles de la relation $\mathcal{R}(A_1, A_2, A_3, A_4)$ décrite par la table ci-dessous

A_1	A_2	A_3	A_4
1	2	3	4
1	2	3	5
6	7	8	2
2	1	3	4

  - $A_1 \rightarrow A_2, A_3$
  - $A_2 \rightarrow A_1, A_3$
  - $A_2, A_3 \rightarrow A_1$
  - $A_4 \rightarrow A_3$

Clés, dépendances et normalisation

Propriétés

- Réflexivité : si $A \subseteq X$ alors $X \rightarrow A$
- Transitivité : Si $X \rightarrow Y$ et $Y \rightarrow Z$ alors $X \rightarrow Z$
- Augmentation : si $X \rightarrow Y$ alors $X, Z \rightarrow Y$ pour tout ensemble Z

Définition

- Une dépendance fonctionnelle $A \rightarrow X$ est **minimale** s'il n'existe pas d'ensemble d'attributs $B \subsetneq A$ tel que $B \rightarrow X$.
- Une dépendance fonctionelle $A \rightarrow X$ est **directe** si elle n'est pas obtenue par transitivité.

Clés, dépendances et normalisation

- Les dépendances fonctionnelles vont nous permettre de définir des contraintes que la base de données devra respecter.
  - définir des dépendances fonctionnelles minimales et directes pour chaque relation
- Malheureusement le contrôle de ces contraintes deviendra vite coûteux si l'on ne prend pas de précaution.
  - faire en sorte que toutes les informations d'une relation dépendent d'un nombre minimal d'attributs

Définition (clé)

Une **clé** d'une relation $\mathcal{R}$ est un sous-ensemble minimal C des attributs de $\mathcal{R}$ tel que tout attribut de $\mathcal{R}$ dépend fonctionnellement de C.

- **Exemple** : `noSS` et `email` sont des clés de la relation `Personne(nom, prénom, noSS, dateNaissance, email)`.
  - il peut y avoir plusieurs clés par relation

Clés, dépendances et normalisation

Définition (schéma en troisième forme normale)

Un schéma de relation $\mathcal{R}$ est **normalisé** quand, dans toute dépendance fonctionnelle $S \rightarrow A$ sur les attributs de $\mathcal{R}$, S est une clé.

- Nous verrons par la suite qu'il est toujours possible de se ramener à des relations normalisées.
- La présence de relation normalisée est fondamentale pour la maintenance d'une base de données
  - évite les anomalies au moment des mises à jour
- Les clés seront uniques de sorte à éviter facilement les redondances dans les tables de relation
  - utilisation de dictionnaire et de tables de hachage par le SGBD
- En pratique un bon schéma relationnel sera divisé en plusieurs relations normalisées.

Clés, dépendances et normalisation

- **Exemple** : on considère des experts lisant des manuscrits pour un éditeur
  - `Manuscrit(idManuscrit, auteur, titre, idEpxert, nom)`

idManuscrit	auteur	titre	idExpert	nom
10	Serge	L'arpète	2	Philippe
20	Cécile	Louis XIV	2	Sophie
10	Serge	L'arpète	2	Philippe
10	Philippe	SQL	1	Sophie

    - ★ La relation `Manuscrit` n'est pas normalisé car dans `idExpert` $\rightarrow$ `nom`, l'attribut `idExpert` n'est pas une clé.
    - ★ il y a des lignes redondantes et incohérentes
  - `Manuscrit(idManuscrit, auteur, titre, idExpert)` et `Expert(idEpxert, nom)`
    - ★ On a mis les informations sur l'expert dans une seconde relation
    - ★ On se réfère à l'expert par l'attribut `idExpert` qui doit être le même dans les deux tables.

idManuscrit	auteur	titre	idExpert
10	Serge	L'arpète	2
20	Cécile	Louis XIV	1
10	Philippe	SQL	1

idExpert	nom
1	Sophie
2	Philippe

Clés, dépendances et normalisation

Définition (clé étrangère)

Soient $\mathcal{R}$ et $\mathcal{S}$ deux relations de clés respectives `idR` et `idS`. Une clé **étrangère** de $\mathcal{S}$ dans $\mathcal{R}$ est un attribut de $\mathcal{R}$ dont la valeur est toujours identique à exactement une valeur de `idS`.

- Lorsqu'une relation contient plusieurs clés on en choisit une qui servira d'identifiant et que l'on nommera clé **primaire**.
- Une clé étrangère permet, par transitivité, de tout savoir sur le n-uplet identifié par sa valeur, ce n-uplet étant en général (pas toujours) placé dans une autre table.
- Dans l'exemple précédent `idExpert` est une clé étrangère de `Expert` dans `Manuscrit`.

Clés, dépendances et normalisation

- Dans un schéma normalisé, un système doit donc gérer deux types de contraintes, toutes deux liées aux clés.

Définition (contraintes)

- Contrainte d'unicité : une valeur de clé ne peut apparaître qu'une fois dans une relation.
- Contrainte d'intégrité référentielle : la valeur d'une clé étrangère doit toujours être également une des valeurs de la clé référencée.

- Ces deux contraintes garantissent l'absence totale de redondances et d'incohérences.

Plan

La base des voyageurs

- Le premier exemple décrit la vie de quelques voyageurs ayant occupé occasionnellement des logements pendant des périodes plus ou moins longues, en y exercant (ou pas) quelques activités.
- Voici le schéma de la base. Les clés primaires sont en bleu, les clés étrangères en rouge
  - `Voyageur(idVoyageur, nom, prénom, ville, région)`
  - `Séjour(idSéjour, idVoyageur, codeLogement, début, fin)`
  - `Logement(code, nom, capacité, type, lieu)`
  - `Activité(codeLogement, codeActivité, description)`

La base des voyageurs

- Table des voyageurs
  - on indique la ville et la région de résidence
  - les voyageurs sont identifiés par `idVoyageur`, incrémenté de 10 en 10

id-Voya-geur	nom	prénom	ville	région
10	Fogg	Phileas	Ajaccio	Corse
20	Bouvier	Nicolas	Aurillac	Auvergne
30	David-Néel	Alexandra	Lhassa	Tibet
40	Stevenson	Robert Louis	Vannes	Bretagne

La base des voyageurs

- Table des logements
  - l'attribut `lieu` correspond à l'attribut `région` de la table voyageurs
    - ★ il arrive qu'une même information soit présente sous des noms différents

code	nom	capa-cité	type	lieu
pi	U Pinzutu	10	Gîte	Corse
ta	Tabriz	34	Hôtel	Bretagne
ca	Causses	45	Auberge	Cévennes
ge	Génépi	134	Hôtel	Alpes

La base des voyageurs

- Table des séjours
  - les voyageurs sont identifiés par `idSéjour`, incrémenté de 1 en 1
  - `début` et `fin` correspondent aux numéro de semaine dans l'année
  - il y a deux clés étrangères référencant le logement et le voyageur

idSéjour	idVoyageur	codeLogement	début	fin
1	10	pi	20	20
2	20	ta	21	22
3	30	ge	2	3
4	20	pi	19	23
5	20	ge	22	24
6	10	pi	10	12
7	30	ca	13	18
8	20	ca	21	22

La base des voyageurs

- Table des activités
  - la clé de la relation `Activité` est la paire (`codeLogement`, `codeActivité`)
  - la clé étrangère `codeLogement` fait partie de la clé (primaire) de la relation `Activité`.

codeLoge-ment	codeActivité	description
pi	Voile	Pratique du dériveur et du catamaran
pi	Plongée	Baptèmes et préparation des brevets
ca	Randonnée	Sorties d'une journée en groupe
ge	Ski	Sur piste uniquement
ge	Piscine	Nage loisir non encadrée

La base des films

- Le deuxième exemple représente des films, leur metteur en scène, leurs acteurs. Les films sont produit dans un pays et des internautes peuvent noter des films.
- Voici le schéma de la base. Les clés primaires sont en bleu, les clés étrangères en rouge
  - `Films(idFilm, titre, année, genre, résumé, idRéalisateur, codePays)`
  - `Pays(code, nom, langue)`
  - `Artiste(idArtiste, nom, prénom, annéeNaissance)`
  - `Rôle(idFilm, idActeur, nomRôle)`
  - `Internaute(email, nom, prénom, région)`
  - `Notation(email, idFilm, note)`
- Le schéma est volontairement simplifié (pas plus d'un réalisateur par film, pas plus d'un rôle par acteur dans un film)

La base des films

- On considère 3 tables de la base de données :
  - La table des films

id	titre	année	genre	idRéali-sateur	code-Pays
20	Impitoyable	1992	Western	130	USA
21	Ennemi d'état	1998	Action	132	USA

  - La table des artistes

id	nom	prénom	annéeNais-sance
130	Eastwood	Clint	1930
131	Hackman	Gene	1930
132	Scott	Tony	1930
133	Smith	Will	1968

  - La table des rôles

idFilm	idArtiste	nomRôle
20	130	William Munny
20	131	Little Bill
21	131	Bril
21	133	Robert Dean

La base des films

- On considère 3 tables de la base de données :
  - La table des films

id	titre	année	genre	idRéali-sateur	code-Pays
20	Impitoyable	1992	Western	130	USA
21	Ennemi d'état	1998	Action	132	USA

  - La table des artistes

id	nom	prénom	annéeNais-sance
130	Eastwood	Clint	1930
131	Hackman	Gene	1930
132	Scott	Tony	1930
133	Smith	Will	1968

  - La table des rôles

idFilm	idArtiste	nomRôle
20	130	William Munny
20	131	Little Bill
21	131	Bril
21	133	Robert Dean

La base des films

- La compréhension du schéma relationnel de la base sur laquelle nous travaillons est indispensable car elle sert de support à l'expression des requêtes SQL.
- Il est impossible d'interroger correctement une base si l'on ne sait pas comment elle est conçue, et notamment si l'on n'a pas en tête les liens définis par les clés étrangères.
  - possibilité de représenter les différents liens sous forme de graphe
- **Exercice** :
  - Que peut-on dire de l'artiste 130 ?
  - Dans quel film joue Gene Hackman ?
  - Pourquoi regrouper acteur et artiste dans la même table ?
  - Représenter les liens entre les 3 tables par un graphe.

La base des films