BASES DE DONNÉES

Tiago de Lima

Cours magistraux
Faculté des Sciences Jean Perrin
Université d'Artois

2e semestre 2019–2020

Section 1

Présentation de l’unité

Objectif de l'unité

L'**objectif** de cette unité est de faire acquérir aux étudiants des notions et concepts basiques des bases de données.

Seront abordés :

- notions basiques (tables, identifiants, clés étrangères, contraintes d'intégrité) ;
- modèle entité-association ;
- modèle relationnel ;
- langage SQL (LDD, LMD) ;
- normalisation ;

Organisation

ECTS : 7

Cours magistraux (CM) : $12 \times 2\ \text{h} = 24\ \text{h}$

Travaux dirigés (TD) : $12 \times 2\ \text{h} = 24\ \text{h}$

Travaux pratiques (TP) : $9 \times 2\ \text{h} = 18\ \text{h}$

Plan (intention) :

Sem.	CM	TD	TP
1	Introduction ; Modèle relationnel	—	—
2	Algèbre relationnelle	Requêtes en AR	—
3	Algèbre relationnelle	Requêtes en AR	—
4	SQL : requêtes	Contraintes en AR	Requêtes SQL simp
5	SQL : créations et modifs.	Requêtes SQL simples	Requêtes SQL
6	**Devoir surveillé**	Mise à jour de la base	Modification de la b
7	Requêtes complexes	Requêtes complexes	Création de la base
8	Requêtes complexes	Requêtes complexes	
9	Modèle entité-association	Modèle entité-association	Projet
10	Normalisation	Normalisation	Projet (cont.)
11	Normalisation (cont.)	Normalisation (cont.)	Projet (cont.)
12		Normalisation (encore !)	Projet (cont.)
13			Soutenance du proj

Modalités de contrôle des connaissances

Session 1 :

- **Devoir surveillé :** semaine 6 ou 7
- **Projet :**
  - Réalisation : semaines 9–12
  - Dépôt : fin de la semaine 12
  - Soutenance : semaine 13
- **Examen :** semaines d'examens
- Calcul de la note :

$$Note = \frac{(DS + P + EX_1)}{3}$$

Session 2 :

- **Examen de rattrapage :** semaines d'examens
- Calcul de la note : $\max(Note, (2 \times DS + P + EX_2)/4)$

Site web de l'unité

Moodle : **`http://moodle.univ-artois.fr/`**

Clé : `vivesqlite2020`

Vous trouverez :

- support des cours (diaporama)
- liens
- annonces
- exercices
- dépôt du projet
- notes

Lecture supplémentaire

Site web :

SQLite :
`http://www.sqlite.org`

Livre en anglais :

Database Systems : The Complete Book
H. Garcia-Molina, J. D. Ullman et J. Windom
Pearson Prentice Hall, 2e édition.

Livre en français :

Bases de données : Concepts utilisation et développement
Jean-Luc Hainaut
Dunod, 3e édition.

Livre en français :

Bases de données et systèmes d'information
Saïd Benaïchou
Ellipses

Cours 1
Introduction

Sommaire

2. Les bases de données sont partout !

3. Qu’est-ce qu’une base de données ?

4. Un peu d’histoire

Section 2

Les bases de données sont partout !

Les bases de données sont partout !

À chaque fois que vous visitez un site web qui délivre de l'information, vous accédez à une base de données.

Des informations relatives à vous (votre nom, date de naissance, notes, etc.) sont stockées dans la base de données de l'université.

Quand vous compostez votre abonnement de transport, vous accédez à une base de données.

Votre assureur a stocké les informations relatives à votre contrat dans une base de données.

Votre téléphone portable contient les adresses et numéros de téléphone de vos collègues stockés dans une base de données.

...

Exemple d’une base de données

Imaginez une pile de bons de commandes comme ceci :

No. commande : 30188			Date : 02/01/2016	
No. client :	B512			
Nom :	GILLET			
Adresse :	14, r. de l’Été			
Localité :	Toulouse			
No. PRODUIT	**LIBELLE PROUDUIT**	**PRIX**	**QUANTITE**	**SOUS-TOT**
CS464	Chev. sapin 400x6x4	220,00	180	39 600,
PA45	Pointe acier 45 (20K)	105,00	22	2 310,
PA60	Pointe acier 60 (10K)	95,00	70	6 650,
PH222	Pl. hetre 200x20x2	230,00	92	21 160,
			TOTAL COMMANDE	69 720,

Exemple d’une base de données

Nous voulons les introduire dans l’ordinateur afin de pouvoir :

- Éditer les factures.
- Contrôler le stock.
- Faire la comptabilité.
- Etc.

Quelle procédure devons nous adopter ?

- Scanner les documents.
- Copier dans un document Word.
- Introduire dans une feuille Excel.

Exemple d’une base de données

En effet, l’utilisation d’une feuille Excel peut suffire, sauf si :

- Le nombre d’enregistrements est important.
- Vous désirez avoir un contrôle d’accès aux données.
- Vous avez besoin de vérifier l’intégrité des données.
- Vous désirez profiter des nombreuses fonctionnalités offertes par les systèmes de gestion de bases de données.

Si tel est le cas, nous allons préférer d’introduire les bons de commandes dans une **base de données**.

Exemple d’une base de données

La structure du bon de commande :

- Données du client :
  - Numéro
  - Nom
  - Adresse
  - Localité
- Données de la commande :
  - Numéro
  - Date
  - Total
- Détails de la commande :
  - Numéro du produit
  - Libellée
  - Prix
  - Quantité
  - Sous-total

Exemple d’une base de données

Nous décidons donc de ranger les données en trois fichiers :

Données de la commande :

ncom	**date**	**total**
30168	02/01/2016	69 720,00

Données du client :

ncli	**nom**	**adresse**	**localite**
B512	GILLET	14, r de l’Été	Toulouse

Données des détails :

npro	**libelle**	**prix**	**qcom**	**sous-total**
CS464	Chev. sapin 400x6x4	220,00	180	39 600,00
PA45	Pointe acier 45 (20K)	105,00	22	2 310,00
PA60	Pointe acier 60 (10K)	95,00	70	6 650,00
PH222	Pl. hetre 200x20x2	230,00	92	21 160,00

Exemple d'une base de données

Mais le résultat n'est pas tout à fait satisfaisant :

- Certaines données sont calculées. Donc, il n'est pas absolument nécessaire de les stocker.
- Il est impossible de reconstituer les documents initiaux !
  - Comment récupérer le client d'une commande ?
  - Comment identifier une commande de laquelle nous avons extrait un détail ?

Nous complétons donc la structure des données en introduisant dans le ficher des commandes un champ **ncli** indiquant le numéro du client.

Nous introduisons également dans le fichier des détails un champ **ncom** spécifiant la commande de chaque détail.

Exemple d’une base de données

Données des commandes :

ncom	**ncli**	**date**
30188	B512	02/01/2016
30179	C400	22/12/2015

Données des clients :

ncli	**nom**	**adresse**	**localite**
B512	GILLET	14, r. de l’Été	Toulouse
C400	FERARD	63, r. du Tertre	Poitiers

Données des détails :

ncom	**npro**	**libelle**	**prix**	**qcom**
30188	CS464	Chev. sapin 400x6x4	220,00	180
30188	PA45	Pointe acier 45 (20K)	105,00	22
30188	PA60	Pointe acier 60 (10K)	95,00	70
30188	PH222	Pl. hetre 200x20x2	230,00	92
30179	CS262	Chev. sapin 200x6x2	75,00	60
30179	PA60	Pointe acier 60 (10K)	95,00	20

Exemple d’une base de données

Mais il reste un malaise.

Pour tous les détails qui référencent un même produit, nous reprenons le libellé et le prix de ce produit.

Ceci crée des redondances inutiles mais aussi **dangereuses** : Quelle garantie que tous les détails d’un même produit indiquent exactement le même libellé et le même prix ?

Nous décidons donc de créer un quatrième fichier destiné à recueillir les données spécifiques aux produits.

Exemple d’une base de données

Données des commandes :

ncom	**ncli**	**date**
30188	B512	02/01/2016
30179	C400	22/12/2015

Données des clients :

ncli	**nom**	**adresse**	**localite**
B512	GILLET	14, r. de l’Été	Toulouse
C400	FERARD	63, r. du Tertre	Poitiers

Données des détails :

ncom	**npro**	**qcom**
30188	CS464	180
30188	PA45	22
30188	PA60	70
30188	PH222	92
30179	CS262	60
30179	PA60	20

Données des produits :

npro	**libelle**	**pr**
CS464	Chev. sapin 400x6x4	220,
PA45	Pointe acier 45 (20K)	105,
PA60	Pointe acier 60 (10K)	95,
PH222	Pl. hetre 200x20x2	230,
CS262	Chev. sapin 200x6x2	75,

Utilisation de la base de données

Les données organisées ainsi permenttent à d'innombrables applications, y compris certaines auxquelles nous ne pensons pas aujourd'hui.

Par exemple :

- Vérifier les commandes du jour.
- Produire des factures.
- Calculer le chiffre d'affaires hebdomadaire, mensuel, annuel.
- Préparer la déclaration de revenu.
- Étudier la répartition géographique du chiffre d'affaires.
- Identifier les produits que se vendent le mieux.
- Produire le chiffre d'affaires de chaque produit par localité et par quadrimestre.

Mais pour pouvoir faire cela, aurons besoin de connaître peu plus à propos des systèmes de gestion de bases de données.

Section 3

Qu’est-ce qu’une base de données ?

Qu’est-ce qu’une base de données ?

Donnée (« data ») : information représentée sous forme physique.

Base de données (« database ») : collection organisée de données persistantes (souvent plusieurs années).

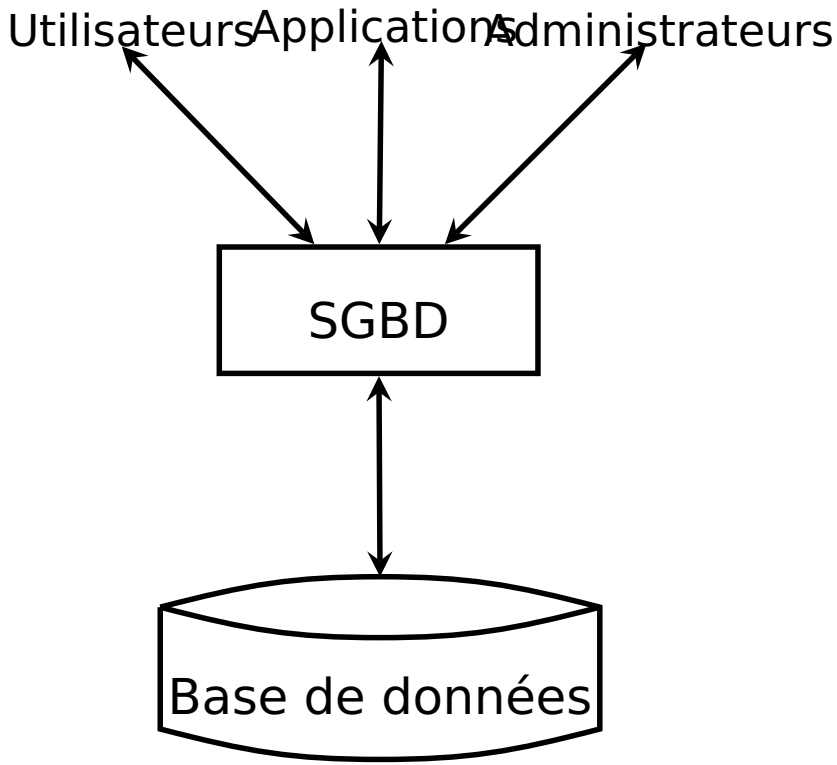
SGBD – Système de gestion base de données (« DBMS – database management system ») : outil logiciel assurant la structuration, stockage, maintenance, mise à jour et consultation des données d’une base de données.

Quelques exemples :

- MS Access
- MySQL
- OO Base
- Oracle
- PostgreSQL
- SQLite

BD vs. SGBD

L’utilisation d’une base de données est faite en utilisant un SGBD.



Fonctions d'un SGBD

Organisation :

- Permet la création des tables et colonnes.
- Permet la création des contraintes d'intégrité (unicité, référence...).

Gestion :

- Permet l'ajout, suppression et modification des tables, lignes et colonnes.
- Vérifie les contraintes d'intégrité.

Accès :

- Permet la consultation des tables, lignes et colonnes.
- Permet une recherche efficace des informations.
- Offre une interface qui facilite l'utilisation.

Autres :

- Protection contre les incidents (sauvegarde, journalisation, etc.).
- Gestion d'accès concurrents.
- Contrôle des droits d'accès (sécurité).

Section 4

Un peu d’histoire

Un peu d'histoire

Années 1950 :

- Fichier classiques.
- Programmes spécifiques.

Années 1960 :

- Modèle hiérarchique.
- Modèle réseaux.
- Langages navigationnels

Années 1970 :

- Modèle relationnel.
- Modèle entité-association.
- Langage SQL.

Années 1980 :

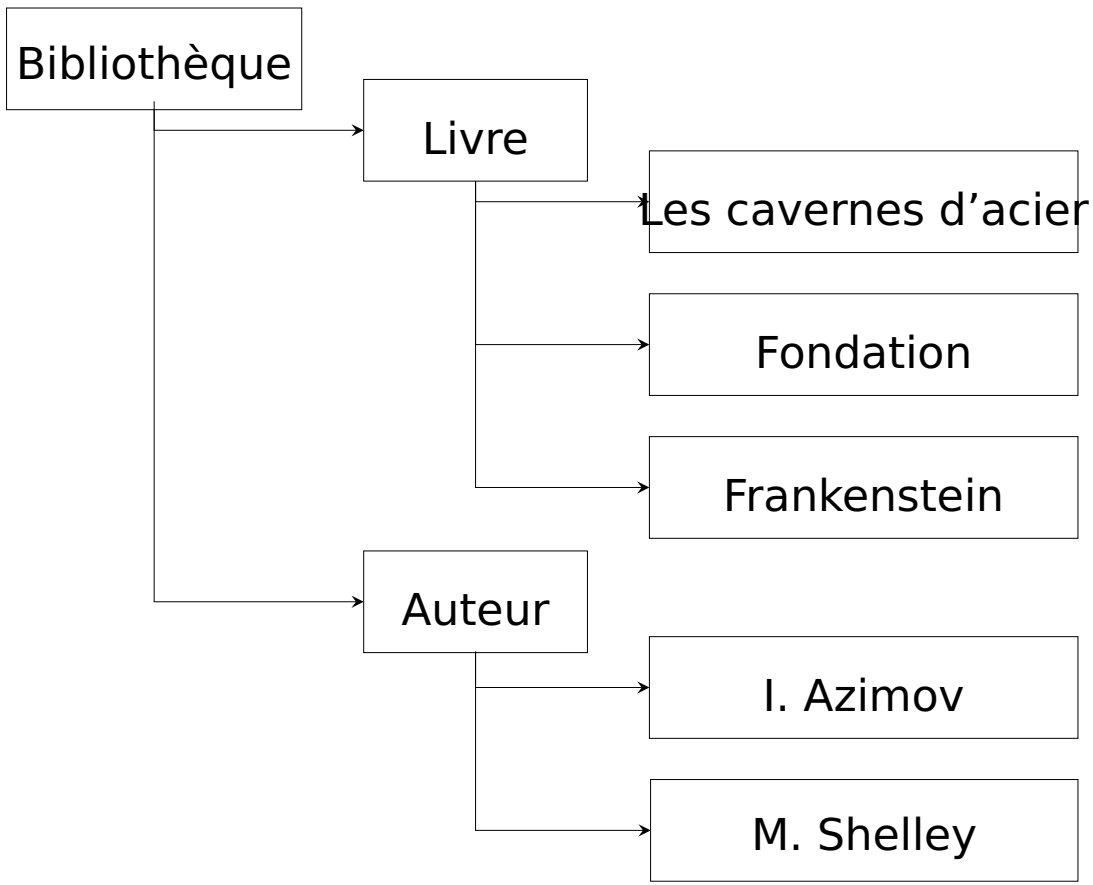
- Modèle orienté objet.

Modèle hiérarchique

Pour palier les faiblesses des fichiers traditionnels (redondance, dépendance de donnés, programmes).

Les données sont organisées sous forme arborescente.

Exemple : IMS (Integrated Managament System).

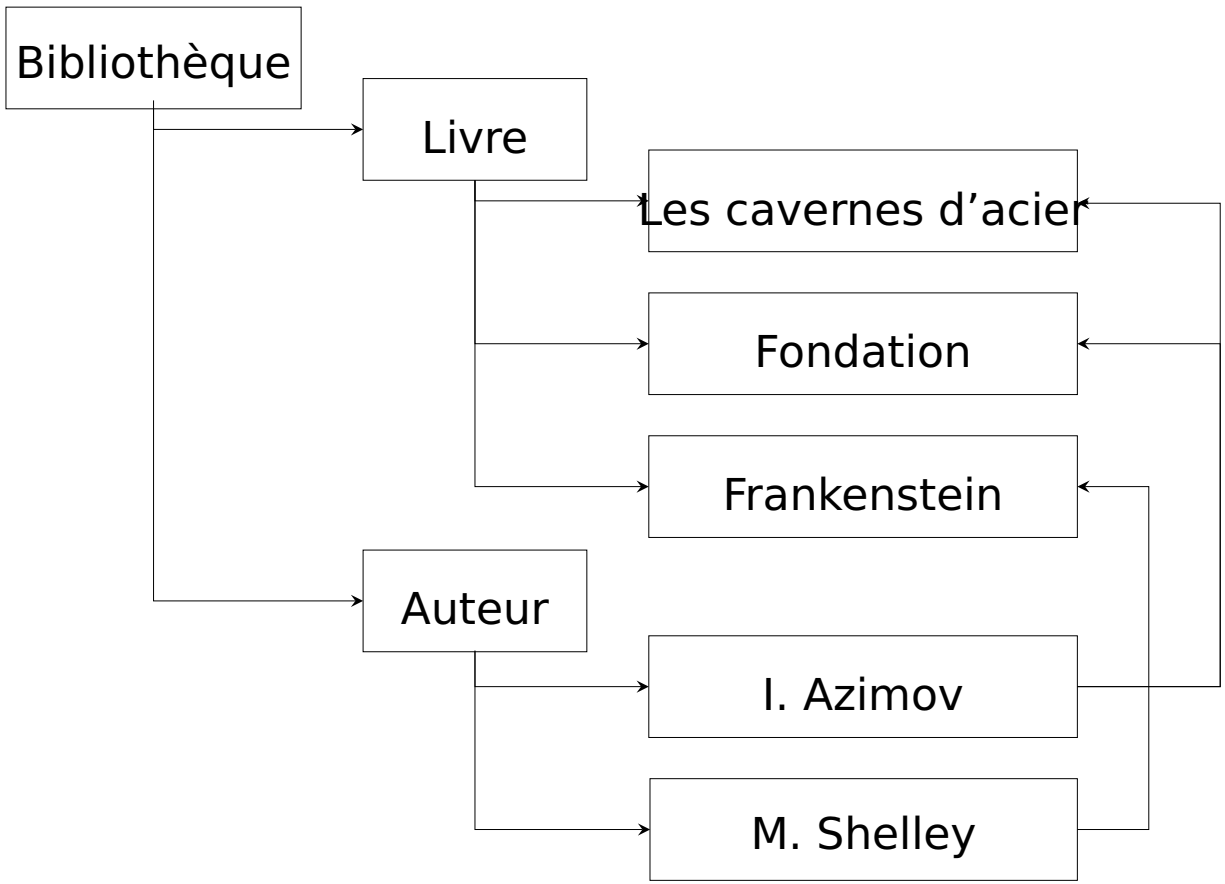


Modèle réseaux

Le modèle hiérarchique ne peut être parcouru que de façon descendante. Cela ne permet pas certaines requêtes.

Le modèle réseaux permet de lier les articles sous forme d'un réseaux.

Exemple : IDS (Integrated Data Store).

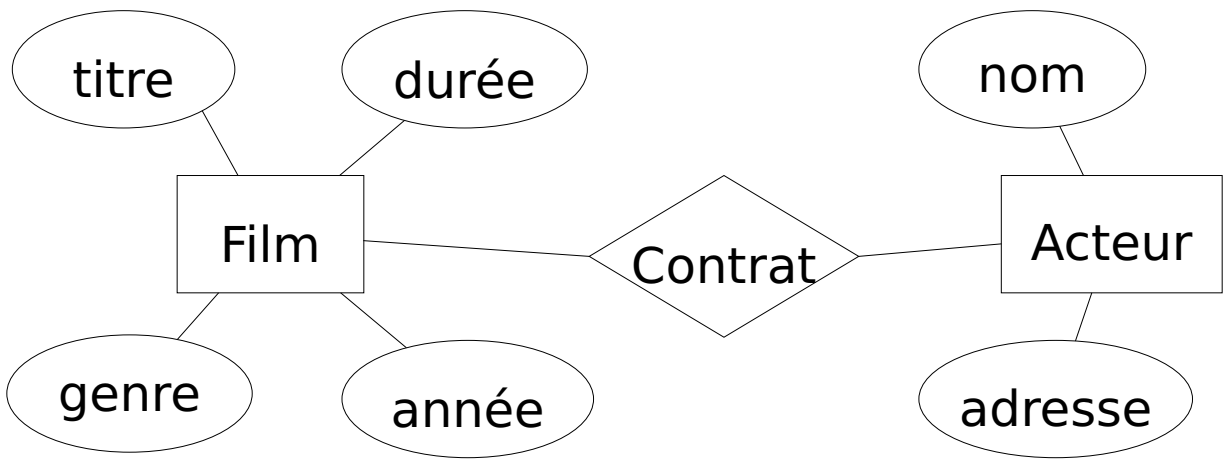


Modèle entité-association

Ce modèle propose de modéliser le domaine du discours au moyens de mécanismes de classification et agrégation.

Plus facilement compréhensible par les utilisateurs non-techniciens.

Utilisé plutôt dans la conception des bases de données.



Modèle relationnel

Le modèle relationnel est incontestablement le plus utilisé aujourd'hui.

Les données sont organisées sous forme de tables (matrices de lignes et colonnes) et manipulées par une algèbre (ensemble d'opérateurs) relationnelle.

Le langage SQL est le plus utilisé pour la définition, manipulation et consultation des bases de données.

Exemples : Oracle, MySQL, PostgreSQL, SQLite, etc.

```
Film(titre, annee, genre, duree)
Acteur(nom, adresse)
Contrat(titre_film*, nom_acteur*)
```

Modèle orienté objet

Les SGBD orientés objet sont basés sur le paradigme objet.

Modélisation plus riche des données : un objet comporte un arbre de valeurs et non une simple suite de valeurs atomiques.

En pratique, le modèle objet a connu un succès limité (plus de 75% des bases de données actuelles sont relationnelles).

Le langage SQL actuel incorpore quelques constructions objet.

Cours 2

Modèle relationnel

Sommaire

5. Modèle relationnel

6. Algèbre relationnelle

Section 5

Modèle relationnel

Qu’est-ce qu’un modèle de données ?

Modèle de données : Une notation pour décrire les données. Cette description est constituée de trois parties :

1. Structure de données.
2. Opérations.
3. Contraintes.

Modèle relationnel

La structure de données du modèle relationnel est la **relation**, souvent représentée sous la forme d'une table.

Exemple :

La relation `Film` :

titre	annee	duree	genre
Gone With the Wind	1939	231	drame
Star Wars	1977	124	fantastique
Dogma	1999	95	comedie

Les opérations associés aux relations sont les opérations de l'**algèbre relationnelle**. Il s'agit des opérations d'union, intersection, inclusion, etc.

Comme exemple de contrainte, nous pouvons imposer que la liste des films ne puisse pas contenir deux entrées avec le même titre.

Attribut

Les colonnes d'une relation sont appelées **attributs**.

Les attributs de notre exemple sont :

- `titre`
- `annee`
- `duree`
- `genre`

Schéma

Le nom de la relation et l'ensemble des ses attributs est appelée **schéma**.

Nous représentons le schéma d'une relation avec le nom de la relation et ses attributs entre parenthèses :

$$\texttt{Film}(\texttt{titre}, \texttt{annee}, \texttt{duree}, \texttt{genre})$$

Les attributs d'une relation sont un **ensemble** et non une liste.

Néanmoins, nous allons souvent spécifier un ordre « standard » pour la représentation des attributs d'une relation.

Nous utilisons $att(R)$ pour dénoter la **liste** d'attributs de R dans l'**ordre standard**. Par exemple :

$$att(\texttt{Film}) = (\texttt{titre}, \texttt{annee}, \texttt{duree}, \texttt{genre})$$

L'ensemble des schémas des relations d'une base de données est appelée **schéma de la base de données.**

Tuple (N-uplet)

Chaque ligne de la table (mise à part la ligne avec les noms des attributs) est appelée de **tuple** (ou bien **n-uplet**). Par exemple :

$$(\texttt{Gone With the Wind}, 1939, 231, \texttt{drame})$$

est un tuple de la relation `Film`.

Nous utilisons $tup(R)$ pour dénoter l'ensemble de tuples de R. Par exemple :

$$\begin{aligned} tup(\texttt{Film}) = \{&(\texttt{Gone With the Wind}, 1939, 231, \texttt{drame}),\\ &(\texttt{StarWars}, 1977, 124, \texttt{fantastique}),\\ &(\texttt{Dogma}, 1999, 95, \texttt{comedie})\} \end{aligned}$$

Notez aussi que les éléments de chaque tuple sont disposés dans l'**ordre standard**.

Domaine

Le modèle relationnel exige que chaque élément de chaque tuple soit d'un type élémentaire.

Par exemple, les types entier et string sont permis, alors que les types liste et array ne le sont pas.

Chaque attribut d'une relation est associé à un **domaine** qui correspond à un type élémentaire.

Chaque élément de chaque tuple d'une relation doit appartenir au domaine qui le correspond.

Nous pouvons inclure le domaine de chaque attribut dans le schéma de la relation :

`Film(titre : string, annee : integer, duree : integer, genre : s`

Instance

En base de données, les relations ne sont pas statiques.

Nous pouvons insérer, supprimer ou modifier les films dans la relation `Film`.

Nous pouvons aussi changer le schéma de la relation. C'est à dire, nous pouvons aussi ajouter et supprimer des attributs.

Nous appelons d'**instance** un ensemble de tuple d'une relation à un moment donné.

Par exemple, en bas nous avons une instance de la relation `Film` :

$$\{(\texttt{Gone With the Wind}, 1939, 231, \texttt{drame}),$$
$$(\texttt{Star Wars}, 1977, 124, \texttt{fantastique}),$$
$$(\texttt{Dogma}, 1999, 95, \texttt{comedie})\}$$

Clé primaire

Un ensemble d'attributs constitue une **clé primaire** d'une relation si nous décidons d'interdire que deux tuples d'une instance contiennent les mêmes valeurs pour tous les attributs de cet ensemble.

L'importance de la clé primaire est de permettre d'identifier chaque tuple de façon unique.

Exemple :

Nous pouvons décider que `Film` contient une clé primaire composé des attributs `titre` et annee.
Cela veut dire qu'il ne peut jamais y avoir deux films avec le même titre qui sortent dans une même année.

Les attributs qui forment une clé primaire sont soulignés dans le schéma de la relation :

$$\texttt{Film}(\underline{\texttt{titre}}, \underline{\texttt{annee}}, \texttt{duree}, \texttt{genre})$$

Clé étrangère

Un ensemble d'attributs dont le rôle est de référencer un tuple d'une autre relation est dénommé **clé étrangère**.

Les attributs qui forment une clé étrangère sont représentés avec une étoile dans le schéma de la relation.

Dans cet exemple, un film peut appartenir à un studio :

`Film`(<u>`titre`</u>, <u>`annee`</u>, `duree`, `genre`, `nom_studio`*)

`Studio`(<u>`nom`</u>, `adresse`)

Notez que, afin de pouvoir référencer un seul tuple, la clé étrangère de `Film` cible la clé primaire de `Sudio`.

Clé étrangère

Exemple :

Film :

titre	annee	duree	genre	nom_studi
Star Wars	1977	124	fantastique	Fox
Dogma	1999	95	comedie	View Aske
Blade Runner	1982	117	scifi	Warner Br

Studio :

nom	adresse
Warner Bros.	Burbank, CA, US
Fox	Los Angeles, CA, US
View Askew Prod.	Red Bank, NJ, US

Notez que les noms de studio qui apparaissent dans la table Film doivent être présents dans la table Studio.

La valeur NULL

Un attribut dont la valeur n'est pas renseigné a la valeur NULL (signifiant l'absence de valeur).

La valeur NULL est interdite aux clés primaires.

La valeur NULL n'est pas la valeur par défaut.

Faire attention dans les expressions arithmétiques et agrégations (à voir plus tard) si des valeurs NULL sont susceptibles d'être rencontrés.

Un schéma complet

```
Film(
     titre:string,
     annee:integer,
     duree:integer,
     genre:string,
     nom_studio*:string,
     num_prod*:integer
)
Acteur(
     nom:string,
     adresse:string,
     genre:char,
     date_naiss:date
)
Participe(
     titre_film*:string,
     annee_film*:integer,
     nom_acteur*:string
)
```

Un schéma complet

```
Producteur(
     nom:string,
     adresse:string,
     num:integer
)

Studio(
     nom:string,
     adresse:string,
     num_prod*:integer
)
```

Section 6

Algèbre relationnelle

Pourquoi l'algèbre relationnelle ?

Supposons que nous voulons interroger la base de données pour obtenir une information.

Par exemple : Quels sont les titres des filmes sorties avant 1990 ?

titre
Gone With the Wind
Star Wars

La réponse est donc l'instance d'une nouvelle relation.

Dans le SGBD, nous pouvons construire cette nouvelle relation en utilisant le langage SQL :

```
select titre
from Film
where annee < 1990
```

La sémantique de ce langage est donnée par les opérateurs de l'algèbre relationnelle.

Pourquoi l’algèbre relationnelle ?

Mais, nous pouvons faire cela en utilisant C, Python, Java, etc. ?

Oui. Pourtant, l’algèbre relationnelle est **moins puissante** que les langages de programmation conventionnels.

Cela veut dire que le compilateur est capable de générer un code beaucoup **plus efficace** pour SQL que pour C, par exemple.

Dans le cas de grandes bases de données, cela s’avère très important.

Qu’est-ce qu’une algèbre ?

De manière simplifiée, une **algèbre** est une collection d’opérateurs sur un ensemble.

Par exemple : $\langle \mathbb{Z}, \{+, *\} \rangle$ est une algèbre.

Nous pouvons créer des « expressions algébriques », comme par exemple, $((8 + 7) * (1 + (-3)) + x$.

L’expression porte sur des éléments de $\mathbb{Z}$ et son résultat est lui aussi un élément de $\mathbb{Z}$.

Qu’est-ce que l’algèbre relationnelle ?

L’algèbre relationnelle a été introduite par Edgar Frank Codd en 1969.

L’**algèbre relationnelle** est une collection d’opérateurs sur l’ensemble de toutes les relations finies.

Les opérateurs :

- Union, intersection et différence.
- Projection et restriction.
- Produit cartésien et jointure.
- Renommage.

Union

Soit R et S deux relations tel que $att(R) = att(S)$.

L'**union** de R et S, notée $R \cup S$, est une nouvelle relation tel que :

- $att(R \cup S) = att(R) = att(S)$
- $tup(R \cup S) = tup(R) \cup tup(S)$

Exemple :

R :

A	B
1	2
3	4

S :

A	B
3	4
5	6

$R \cup S$:

A	B
1	2
3	4
5	6

Attention : chaque tuple apparaît une seule fois dans le résultat.

Intersection

Soit R et S deux relations tel que $att(R) = att(S)$.

L'**intersection** de R et S, notée $R \cap S$, est une nouvelle relation tel que :

- $att(R \cap S) = att(R) = att(S)$
- $tup(R \cap S) = tup(R) \cap tup(S)$

Exemple :

R :

A	B
1	2
3	4

S :

A	B
3	4
5	6

$R \cap S$:

A	B
3	4

Différence

Soit R et S deux relations tel que $att(R) = att(S)$.

La **différence** de R et S, notée $R - S$, est une nouvelle relation tel que :

- $att(R - S) = att(R) = att(S)$
- $tup(R - S) = tup(R) - tup(S)$

Exemple :

R :

A	B
1	2
3	4

S :

A	B
3	4
5	6

$R - S$:

A	B
1	2

Diférence

Exercice : Montrez que l'intersection peut être définie en utilisant la différence.

Solution :

$$R \cap S = R - (R - S)$$

Projection

Soit R une relation. Soit $P = (A_1, \ldots, A_n)$ tel que chaque $A_i \in att(R)$.

La **projection** de R sur P, notée $\pi_P(R)$, est une nouvelle relation tel que :

- $att(\pi_P(R)) = P$
- $tup(\pi_P(R)) = \{\pi_P(t) \mid t \in tup(R)\}$

où :

- $\pi_P(t) = (e_1, \ldots, e_n) \subseteq t$ tel que chaque e_i correspond à la valeur de l'attribut A_i dans t.

Dit de manière plus simple, la projection sert à couper et/ou réarranger les colonnes d'une relation.

Projection

Exemple :

R :

A	B	C
1	2	3
4	5	6
1	7	3

$\pi_{A,C}(R)$:

A	C
1	3
4	6

$\pi_{C,B}(R)$:

C	B
3	2
6	5
3	7

Attention : chaque tuple apparaît une seule fois dans le résultat.

Restriction

Soit R une relation et C une contrainte.

La **restriction** (ou sélection) de R par C, notée $\sigma_C(R)$, est une nouvelle relation tel que :

- $att(\sigma_C(R)) = att(R)$
- $tup(\sigma_C(R)) = \{t \mid t \in tup(R) \text{ et } t \text{ satisfait } C\}$

La contrainte C est une expression construite avec des attributs, des constantes, des opérateurs de comparaison ($<, \leq, >, \geq, =, \neq$), et des opérateurs logiques ($\neg$, $\wedge$, $\vee$).

Dit de manière plus simple, la restriction sert à couper des lignes.

Restriction

Exemple :

R :

A	B	C
1	2	3
4	5	6
1	7	3

$\sigma_{B \leq 5}(R)$:

A	B	C
1	2	3
4	5	6

$\sigma_{B \leq 5 \wedge C=3}(R)$:

A	B	C
1	2	3

Exercice

Soit la table `Film` :

titre	annee	duree	genre	studio
Star Trek	1979	126	sciFi	Paramount
Star Wars	1977	124	fantastique	Fox
Galaxy Quest	1999	102	comedie	Fox
⋮	⋮	⋮	⋮	⋮

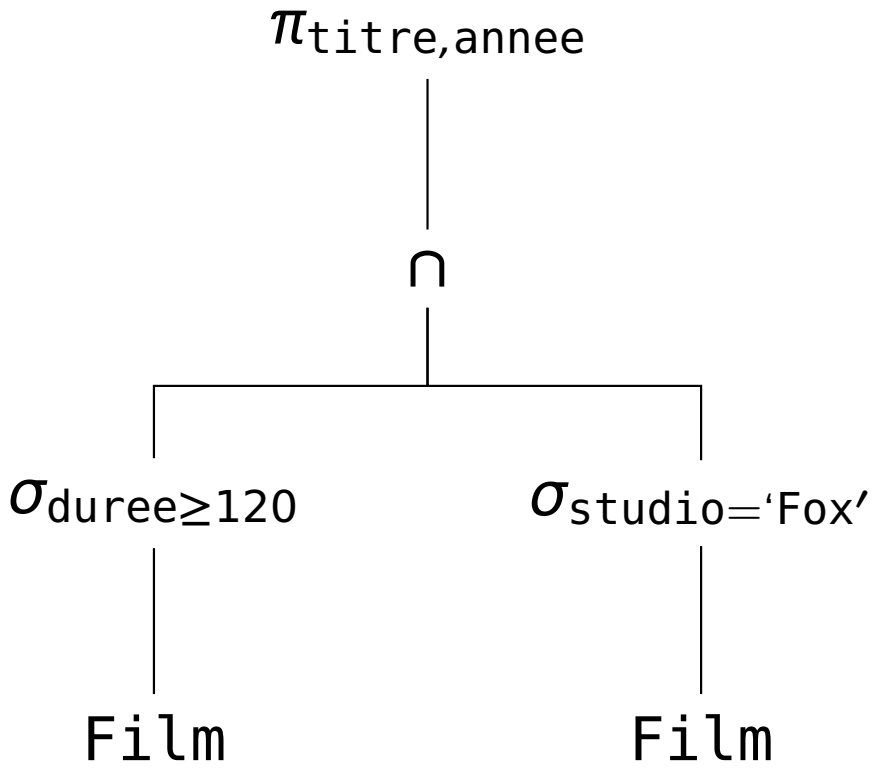
Écrivez une expression algébrique qui répond à la question suivante :

« Quels sont les titre et les années des films produits par Fox dont la durée est d'au moins 120 minutes ? »

Exercice

Voici une manière de la répondre :

1. Sélectionne les tuples de `Film` dont la `duree` est supérieur à 120.
2. Sélectionne les tuples de `Film` dont le `studio` est Fox.
3. Calcule l'intersection de 1 et 2.
4. Projette la relation calculé dans 3 sur `titre` et `annee`.



$$\pi_{\texttt{titre,annee}}(\sigma_{\texttt{duree}\geq 120}(\texttt{Film}) \cap \sigma_{\texttt{studio}=\text{'Fox'}}(\texttt{Film}))$$

Produit cartésien (mathématique)

En mathématique, le produit cartésien de R et S, noté $R \times S$ est l'ensemble de pairs (a, b) où $a \in R$ et $b \in S$.

Exemple :

Soit $R = \{(a, b), (c, d)\}$ et $S = \{(c, d), (e, f)\}$. Nous avons :

$$R \times S = \{((a, b), (c, d)), ((a, b), (e, f)), ((c, d), (c, d)), ((c, d), (e, f))\}$$

Notez que, définit ainsi, $R \times S$ est une relation de relations.

Rappelons-nous qu'en algèbre relationnelle, chaque élément de chaque tuple doit être d'un type élémentaire.

Donc, pour respecter cette contrainte, le produit cartésien en algèbre relationnelle est définit de manière un peu différente.

Produit cartésien (relationnel)

Soit R et S deux relations.

Le **produit cartésien** de R et S, noté $R \times S$, est une nouvelle relation tel que :

- $att(R \times S) = att(R) + att(S)$
- $tup(R \times S) = \{(r_1, \ldots, r_n, s_1, \ldots s_m) \mid ((r_1, \ldots, r_n), (s_1, \ldots s_m)) \in tup(R) \times tup(S)\}$.

Il est parfois nécessaire de renommer certains attributs de R et de S pour éviter des conflits.

Exemple :

R :

A	B
1	2
3	4

S :

B	C	D
2	5	6
4	7	8
9	10	11

$R \times S$:

A	$R.B$	$S.B$	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

θ-Jointure

Soit R et S deux relations.

La **θ-jointure** de R et S selon une contrainte C, notée $R \bowtie_C S$, est définie comme se suit :

$$R \bowtie_C S = \sigma_C(R \times S)$$

Exemple :

R :

A	B
1	2
3	4

S :

B	C	D
2	5	6
4	7	8
9	10	11

$R \bowtie_{C \leq 5} S$:

A	$R.B$	$S.B$	C	D
1	2	2	5	6
3	4	2	5	6

Jointure naturelle

Soit R et S deux relations tel que $J = att(R) \cap att(S) \neq \emptyset$.

La **jointure naturelle** de R et S, notée $R \bowtie S$, est définie comme se suit :

$$R \bowtie S = \pi_P(\sigma_C(R \times S))$$

où $C = \bigwedge_{A \in J}(R.A = S.A)$ et $P = att(R) + (att(S) - J)$

Dit de manière plus simple, la jointure naturelle est l'ensemble de tuples de deux relations dont les attributs en commun ont la même valeur. En plus, les attributs en commun ne sont pas répétés dans le résultat.

Jointure naturelle

Exemple :

R :

A	B	C
1	2	3
6	7	8
9	7	8

S :

B	C	D
2	3	4
2	3	5
7	8	10

$R \bowtie S$:

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Exercice

Soit le schéma relationnel suivant :

Client(<u>ncli</u>, nom, prenom, adresse, ntel)
Commande(<u>ncmd</u>, ncli*, date)

Écrivez une expression permettant d'extraire la liste de noms, prénoms et numéros de téléphones des clients qui ont passé des commandes avant le 1/1/2011.

$$\pi_{\texttt{nom,prenom,ntel}}(\sigma_{\texttt{date<1/1/2011}}(\texttt{Client} \bowtie \texttt{Commande}))$$

Jointure externe à gauche

Soit R et S deux relations.

La **jointure externe à gauche** de R et S, notée $R \ltimes S$, est une nouvelle relation tel que :

- $att(R \ltimes S) = att(R \bowtie S)$
- $tup(R \ltimes S) = (R \bowtie S) \cup E$

où E est l'ensemble de tuples de R qui ne participent pas à la jointure naturelle avec des valeurs NULL pour les attributs de S.

Jointure externe à gauche

Exemple :

R :

A	B	C
1	2	3
6	7	8
9	5	8
10	11	12

S :

B	C	D
2	3	4
2	3	5
7	8	10

$R \bowtie S$:

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10

$R ⟕ S$:

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	5	8	NULL
10	11	12	NULL

Jointure externe à droite

Soit R et S deux relations.

La **jointure externe à droite** de R et S, notée $R \rtimes S$ est une nouvelle relation tel que :

- $att(R \rtimes S) = att(R \bowtie S)$
- $tup(R \rtimes S) = (R \bowtie S) \cup E$

où E est l'ensemble de tuples de S qui ne participent pas à la jointure naturelle avec des valeurs NULL pour les attributs de R.

Dit de manière plus compliquée :

$$R \rtimes S = \pi_{att(R \bowtie S)}(S \rtimes R)$$

Jointure externe à droite

Exemple :

R :

A	B	C
1	2	3
6	7	8
9	7	8
10	11	12

S :

B	C	D
2	3	4
2	3	5
7	8	10
13	14	15
16	17	18

$R \bowtie S$:

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

$R ⟖ S$:

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10
NULL	13	14	15
NULL	16	17	18

Jointure externe complète

Soit R et S deux relations.

La **jointure externe complète** de R et S, notée $R \mathbin{⟗} S$ est une nouvelle relation tel que :

- $att(R \mathbin{⟗} S) = att(R \bowtie S)$
- $tup(R \mathbin{⟗} S) = (R \bowtie S) \cup E$

où E est l'ensemble de tuples de R et de S qui ne participent pas à la jointure naturelle avec des valeurs `NULL` pour les attributs de S et de R respectivement.

Dit de manière plus compliquée :

$$R \mathbin{⟗} S = (R \mathbin{⟕} S) \cup (R \mathbin{⟖} S)$$

Jointure externe complète

Exemple :

R :

A	B	C
1	2	3
6	7	8
9	7	8
10	11	12

S :

B	C	D
2	3	4
2	3	5
7	8	10
13	14	15
16	17	18

$R \bowtie S$:

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

$R \overset{\leftarrow}{\bowtie} S$:

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10
10	11	12	NULL
NULL	13	14	15
NULL	16	17	18

Exercice

Soit le schéma relationnel suivant :

Employee(<u>id</u>, nom, nomdep*)
Departement(<u>nomdep</u>, nomman)

Écrivez une expression permettant d'extraire la liste de **tous** les noms des employées avec le nom de leur département et le nom de leur manager, si applicable.

$$\pi_{\text{nom,nomdep,nomman}}(\text{Employee} \bowtie \text{Departement})$$

Renommage

Soit R une relation tel que $att(R) = (A_1, \ldots, A_n)$.

Le **renommage** de R, noté $\rho_{B_1,\ldots,B_n}(R)$, est une nouvelle relation tel que :

- $att(\rho_{B_1,\ldots,B_n}(R)) = (B_1, \ldots, B_n)$
- $tup((\rho_{B_1,\ldots,B_n}(R)) = tup(R)$

Dit de manière plus simple, le renommage de R change uniquement le nom de ces attributs.

Ceci est utile notamment pour gérer les conflits dans les relations résultant de produits cartésiens ou jointures où les deux opérandes contiennent des attributs de même nom.

Notation linéaire

Pour simplifier l'écriture des expressions algébriques, nous alons parfois utiliser la « notation lineaire ».

Exemple :

« Quels sont les titre et les années des films produits par Fox dont la durée est de 120 minutes minimum ? »

$$R := \sigma_{\texttt{duree} \geq 100}(\texttt{Film})$$
$$S := \sigma_{\texttt{studio}=\texttt{'Fox'}}(\texttt{Film})$$
$$\texttt{Resultat} := \pi_{\texttt{titre,annee}}(R \cap S)$$

Cours 3

Contraintes sur les relations

Sommaire

7. Un langage pour les contraintes

8. Contrainte référentielle

9. Contrainte d’unicité

10. Autres contraintes

Section 7

Un langage pour les contraintes

Un langage pour les contraintes

Rappelons-nous :

Modèle de données : Une notation pour décrire les données. Cette description est constituée de de trois parties :

1. Structure de données.
2. Opérations.
3. Contraintes.

Un langage pour les contraintes

Il y a deux manières d'utiliser l'algèbre relationnelle pour exprimer des contraintes :

1. Si E est une expression en algèbre relationnelle, alors $E = \emptyset$ est une contrainte qui exprime :
   « *Il n'y a pas de tuples dans le résultat de E.* »

2. Si E_1 et E_2 sont des expressions en algèbre relationnelle, alors $E_1 \subseteq E_2$ est une contrainte qui exprime :
   « *Chaque tuple dans le résultat de* E_1 *doit être présent dans le résultat de* E_2*.* »

En effet, nous avons :

$$E_1 \subseteq E_2 \text{ est équivalent à } E_1 - E_2 = \emptyset$$

Donc, le choix entre la première ou la deuxième se fait tout simplement par un souci de clarté.

Section 8

Contrainte référentielle

Contrainte référentielle

La **contrainte référentielle** stipule qu'une valeur dans un contexte doit apparaître dans un autre contexte.

Il s'agit de la contrainte utilisée pour les **clés étrangères**.

Soit les relations $R(a, b)$ et $S(c, d)$, tel que a est une clé étrangère qui fait référence à c.

La contrainte référentielle correspondante peut être exprimée sous la forme :

$$\pi_{R.a}(R) \subseteq \pi_{S.c}(S)$$

Ou, de forme équivalente :

$$\pi_{R.a}(R) - \pi_{S.c}(S) = \emptyset$$

Exemple

Soit les relations :

$\texttt{Film}(\underline{\texttt{titre}}, \underline{\texttt{annee}}, \texttt{duree}, \texttt{genre}, \texttt{nom\_studio}*, \texttt{num\_prod}*)$

$\texttt{Producteur}(\underline{\texttt{num}}, \texttt{nom}, \texttt{adresse})$

$\texttt{Studio}(\underline{\texttt{nom}}, \texttt{adresse}, \texttt{num\_prod}*)$

Les contraintes référentielles peuvent être exprimées comme suit :

$$\pi_{\texttt{nom\_studio}}(\texttt{Film}) \subseteq \pi_{\texttt{nom}}(\texttt{Sudio})$$
$$\pi_{\texttt{num\_prod}}(\texttt{Film}) \subseteq \pi_{\texttt{num}}(\texttt{Producteur})$$
$$\pi_{\texttt{num\_prod}}(\texttt{Studio}) \subseteq \pi_{\texttt{num}}(\texttt{Producteur})$$

Exemple

Soit les relations :

$\texttt{Film}(\underline{\texttt{titre}, \texttt{annee}}, \texttt{duree}, \texttt{genre}, \texttt{nom\_studio}*, \texttt{num\_prod}*)$

$\texttt{Acteur}(\underline{\texttt{nom}}, \texttt{adresse}, \texttt{genre}, \texttt{date\_naiss})$

$\texttt{Participe}(\underline{\texttt{titre\_film}*, \texttt{anee\_film}*, \texttt{nom\_acteur}*})$

Les contraintes référentielles peuvent être exprimées comme suit :

$$\pi_{\texttt{titre\_film,anne\_film}}(\texttt{Participe}) \subseteq \pi_{\texttt{titre,annee}}(\texttt{Flim})$$

$$\pi_{\texttt{nom\_acteur}}(\texttt{Participe}) \subseteq \pi_{\texttt{nom}}(\texttt{Acteur})$$

Section 9

Contrainte d’unicité

Contrainte d'unicité

La **contrainte d'unicité** stipule qu'une valeur doit apparaître une seule fois dans un contexte.

Il s'agit de la contrainte utilisée pour la **clé primaire**.

Soit la relation $R(a, b)$. La contrainte d'unicité sur l'attribut a peut être exprimée sous la forme :

$$R1(a, b) := R(a, b)$$
$$R2(a, b) := R(a, b)$$
$$\sigma_{(R1.a=R2.a)\wedge(R1.b\neq R2.b)}(R1 \times R2) = \emptyset$$

Contrainte d'unicité

Exercice : Écrivez une expression en algèbre relationnelle exprimant la contrainte qui stipule que les valeurs de l'attribut a dans la relation $R(a, b, c)$ sont uniques.

$$R1 := R$$
$$R2 := R$$
$$C := R1.a = R2.a \wedge ((R1.b \neq R2.b) \vee (R1.c \neq R2.c))$$
$$\sigma_C(R1 \times R2) = \emptyset$$

Section 10

Autres contraintes

Autres contraintes

En effet, l'algèbre relationnelle nous permet de spécifier un grand nombre de contraintes sur les données.

Par exemple, nous pouvons stipuler que les seules valeurs permis pour la colonne sexe sont 'F' et 'M'.

$$\sigma_{(\texttt{sexe} \neq \texttt{'F'}) \wedge (\texttt{sexe} \neq \texttt{'M'})}(\texttt{Acteur}) = \emptyset$$

Un autre exemple. Les seuls valeurs permis pour la colonne prix sont les valeurs positifs :

$$\sigma_{(\texttt{prix} < 0)}(\texttt{Produit}) = \emptyset$$

Cours 4

Le langage SQL

Sommaire

11. Le langage SQL

12. SQLite

13. Types de données

14. Création de tables

15. Modification du contenu des tables

16. Vérification du schéma d’une table

Section 11

Le langage SQL

Le langage SQL

SQL = Structured Query Language (« langage de requêtes structuré »).

SQL est composé de :

- langage de définition de données (LDD) ;
- langage d'interrogation et manipulation de données (LMD) ;
- langage de contrôle de données (LCD).

SQL est un langage pouvant être utilisé directement par des utilisateurs ou par des programmes afin de s'adresser à un SGBD.

SQL est le langage de base de données le plus utilisé actuellement. Utilisé par Oracle, PostgreSQL, MySQL, SQLite, etc.

Langage de définition de données

- Création de la base de données ;
- Suppression de la base de données ;
- Création de tables ;
- Suppression de tables ;
- Modification de tables.

Langage de manipulation de données

- Insertion de tuples dans la base ;
- Suppression de tuples de la base ;
- Mise à jour des tuples de la base ;
- Interrogation de la base de données.

Langage de contrôle de données

C’est l’ensemble de commandes servant à contrôler l’accès aux données.

- Accorder des privilèges aux utilisateurs ;
- Dénier des privilèges aux utilisateurs ;
- etc.

Un peu d'histoire

- 1969 : modèle relationnel
- 1975 : langage QUEL
- 1976 : langage SEQUEL (IBM)
- 1986 : SQL (ANSI)
- 1989 : SQL-89 (révision mineure)
- 1992 : SQL2 (révision majeure)
- 1999 : SQL3 (quelques concepts des langages orientés objet)
- 2003 : SQL:2003
- 2006 : SQL:2006
- 2008 : SQL:2008
- 2011 : SQL:2011

Langages associés

Langages associés à SQL :

- PL/SQL (Oracle)
- Transact/SQL (SQLServer)
- PL/pgSQL (PostgreSQL)

Ces langages servent à écrire des scripts et programmes compilés et exécutés par le SGBD.

Ce sont des langages de programmation procéduraux permettant d'inclure des requêtes SQL.

Section 12

SQLite

SQLite

SQLite est un petit (< 500 Mo) SGBD **sans serveur** et publique.

`http://www.sqlite.org`

Selon ses créateurs, SQLite est un des logiciels le plus déployés au monde.

Utilisateurs de SQLite :

- Firefox, Chrome, Safari ;
- Skype, iTunes, Dropbox ;
- Android, iPhone ;
- Mac OS, Windows 10 ;
- Bosch, General Electric, Toshiba, Airbus ;
- etc.

L’interface shell de SQLite

Exemple d’utilisation de l’interface shell de SQLite :

```
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> create table t1(a int);
sqlite> create table t2(b int);
sqlite> .tables
t1  t2
sqlite> drop table t2;
sqlite> .tables
t1
```

L’interface shell de SQLite

L’interface shell de SQLite accepte deux types d’instructions :

1. Instructions SQL : instructions en langage SQL communes à plusieurs SGBD différents.
2. Instructions commencent avec un point (« . ») : instructions spécifiques à l’interface shell SQLite.

Instructions spécifiques

Quelques instructions utiles spécifiques à l'interface shell de SQLite :

- `.exit` : quitte le shell SQLite.
- `.help` : affiche l'aide.
- `.open FICHIER` : ouvre une base de données sauvegardée.
- `.save FICHIER` : sauvegarde la base de données courante.
- `.read FICHIER` : exécute un script SQL dans un fichier.
- `.tables` : liste les tables crées dans la base de données courante.

Section 13

Types de données

Types de données

L'ensemble des types primitifs supportés par SQL inclue :

`null`	représente l'absence de valeur.
`char(n)`	chaîne de caractères de longueur *n*.
`varchar(n)`	chaîne de caractères de longueur au plus *n*.
`bit(n)`	chaîne de bits de longueur *n*.
`bit varying(n)`	chaîne de bits de longueur au plus *n*.
`boolean`	valeurs de vérité : `true`, `false` et unknown.

`int`
`smallint`
`bigint`
	nombres entiers.

`float` `real`	nombres en virgule flottante.
`numeric(p, s)` `decimal(p, s)`	nombres en décimal.
`date`	date.
`time`	heure.

Types de données

En pratique, certains types ne sont pas supportés par tous les SGBD.

Dans le cas spécifique de SQLite, nous avons seulement :

- `null`
- `integer`
- `real`
- `text` : chaîne de caractères avec longueur (presque) illimitée.

Les autres types sont acceptés. Mais les données sont automatiquement transformés vers un de quatre types en haut.

Section 14

Création de tables

Relations en SQL

Il y a trois types de relations en SQL :

1. Tables : relations sauvegardées dans la base de données.
2. Views : relations sauvegardées sous forme de requête, elles sont calculées en temps d'exécution.
3. Tables temporaires : relations crées par le SGBD pour sauvegarder le résultat des calculs intermédiaires. Ces tables sont détruites automatiquement par le SGBD quand le calcul termine.

Pour commencer, nous allons utiliser seulement le premier type. Les views seront traitées plus tard.

Création de tables

Syntaxe simplifiée :

```
CREATE TABLE nom_table (
    colonne type,
    colonne type,
    ...
)
```

Exemple :

```
1 create table film(
2     titre        char(100),
3     annee        int,
4     duree        int,
5     genre        char(10),
6     nom_studio   char(30),
7     num_prod     int
8 );
```

Création de tables

Résultat de la création de la table film :

film :

titre	annee	duree	genre	nom_studio	nom_prod

Section 15

Modification du contenu des tables

Insertion

Syntaxe simplifiée :

```
INSERT INTO nom_table (colonne, ...)
VALUES (val_1, val_2, ...),
       (val_1, val_2, ...),
       ...
```

Exemple :

```
1 insert into film (titre, annee, genre)
2 values ('Star Wars', 1977, 'science fiction'),
3        ('Dogma', 1999, 'science fiction');
```

Insertion

Résultat de l'insertion :

film :

titre	annee	duree	genre	nom_studio	nom
Star Wars	1977	NULL	science fiction	NULL	NUL
Dogma	1999	NULL	science fiction	NULL	NUL

Suppression

Syntaxe simplifiée :

```
DELETE FROM nom_table WHERE condition
```

Exemple :

```
1 delete from participe
2 where titre_film = 'The Maltese Falcon' and
3       annee = 1942 and
4       nom_acteur = 'Sydney Greenstreet';
5
6 delete from producteur
7 where numero < 100;
```

Mise à jour

Syntaxe simplifiée :

```
UPDATE nom_table SET col1 = val1, ... WHERE condition
```

Exemple :

```
1 update film
2 set genre = 'scifi'
3 where genre = 'science fiction';
```

Mise à jour

Résultat de la mise à jour :

film :

titre	annee	duree	genre	nom_studio	nom_prod
Star Wars	1977	NULL	scifi	NULL	NULL
Dogma	1999	NULL	scifi	NULL	NULL

Consultation de tables

Syntaxe (très) simplifiée :

```
SELECT colonne, ... FROM nom_table
```

Exemple :

```
sqlite> select titre, annee from film;
Star Wars|1977
Dogma|1999
```

Pour consulter toutes les colonnes d'une table, vous pouvez utiliser l'étoile (`*`) comme raccourci :

```
sqlite> select * from film;
Star Wars|1977||||
Dogma|1999||comedie||
```

Consultation de tables

Vous pouvez aussi demander à SQLite de montrer l'« entête »(les nom des attributs) de la table :

```
sqlite> .headers on
sqlite> select * from film;
titre|annee|duree|genre|nom_studio|num_prod
Star Wars|1977||||
Dogma|1999||comedie||
```

Un affichage encore plus amical :

```
sqlite> .headers on
sqlite> .mode columns
sqlite> select * from film;
titre       annee       duree       genre       ...
----------  ----------  ----------  ----------
Star Wars   1977
Dogma       1999                    comedie
```

Retour sur l'insertion

Comme l'instruction `insert` peut être utilisée pour l'insertion de plusieurs tuples, nous pouvons l'utiliser avec, par exemple, `select`.

Exemple :

Soit le schéma :

`studio(nom, adresse, num_prod)`
`film(titre, annee, duree, genre, nom_studio, num_prod)`

Insérer dans la table `studio` les noms des studios qui apparaissent dans la table `film` mais qui sont absents dans la table `studio` :

```
1 insert into studio(nom)
2     select distinct nom_studio
3     from film
4     where nom_studio not in (
5         select nom
6         from studio
```

Retour sur la mise à jour

Comme l'instruction `update` peut être utilisée pour la mise à jour de plusieurs tuples, nous pouvons l'utiliser avec, par exemple, `select`.

Exemple :

Ajouter le mot `Pres.` au nom de chaque producteur qui est président d'un studio :

```
1 update producteur
2 set nom = 'Pres. ' || nom
3 where numero in (
4     select num_prod from studio
5 );
```

Modification de tables

Syntaxes simplifiées :

```
DROP TABLE nom_table
```

```
ALTER TABLE nom_table ADD colonne type
```

```
ALTER TABLE nom_table DROP colonne
```

Exemple :

```
1 alter table acteur add telephone char(13);
2 alter table acteur drop date_naissance;
```

Note : SQLite ne supporte pas l'instruction :
`ALTER TABLE nom_table DROP ....`

Contraintes sur les colonnes

Il est possible de définir des contraintes sur les colonnes d'une table.

Par exemple, nous pouvons stipuler que si le numéro de téléphone n'est pas renseigné, alors la colonne `telephone` contiendra la valeur `'non-renseigne'` :

```
1 alter table acteur
2     add telephone char(13) default 'non-renseigne';
```

Il est également possible de définir la table avec telle restriction :

```
1 create table acteur(
2     nom       char(50),
3     telephone char(13) default 'non-renseigne'
4 );
```

Contraintes sur les colonnes

Types de contraintes de colonne :

- `PRIMARY KEY`
- `NOT NULL`
- `UNIQUE`
- `CHECK (expr)`
- `DEFAULT valeur`
- `REFERENCES nom_table (nom_colonne)`

Exemple :

```
1 create table acteur (
2     nom               char(30) primary key,
3     adresse           varchar(255),
4     telephone         char(13) default 'non-reseigne',
5     sexe              char(1) check (sexe in ('F', 'M')),
6     date_naissance    date check
7                           (date_naissance < current_date)
8 );
```

Clés étrangères

La contrainte REFERENCES sert à définir les clés étrangères :

Exemple :

```
1 create table participe (
2     nom_acteur   char(30) references acteur(nom),
3     titre_film   char(100) references film(titre),
4     annee_film   int references film(annee)
5 );
```

Voyez-vous un problème avec cette définition ?

Rappelons nous les contraintes associées à la table `Participe` :

$$\pi_{\texttt{titre\_film,annee\_film}}(\texttt{Participe}) \subseteq \pi_{\texttt{titre,annee}}(\texttt{Film})$$

$$\pi_{\texttt{nom\_acteur}}(\texttt{Participe}) \subseteq \pi_{\texttt{nom}}(\texttt{Acteur})$$

La clé étrangère doit porter sur les deux colonnes, `titre_film` et `annee_film`, en même temps.

Contraintes de table

Syntaxe (un peu plus) complète :

```
CREATE TABLE nom_table (
    colonne type contrainte,
    colonne type contrainte,
    ...,
    CONSTRAINT nom_contrainte contrainte_table,
    CONSTRAINT nom_contrainte contrainte_table,
    ...
)
```

Types de contraintes de table :

- `PRIMARY KEY (colonne, ...)`
- `UNIQUE (colonne , ...)`
- `CHECK (expr)`
- `FOREIGN KEY (col, ...) REFERENCES nom_table (col,`

Contraintes de table

Exemple :

```
1 create table participe (
2     nom_acteur   char(30) references acteur(nom),
3     titre_film   char(100),
4     annee_film   int,
5
6     constraint participe_fk
7         foreign key (titre_film, anne_film)
8         references film(titre, annee)
9 );
```

Contraintes de table

D'ailleurs, la clé primaire de la table `film` doit être crée en utilisant une contrainte de table, également :

```
1  create table film(
2      titre        char(100),
3      annee        int,
4      duree        int,
5      genre        char(10),
6      nom_studio   char(30) references studio(nom),
7      num_prod     int references producteur(numero),
8
9      constraint film_pk primary key (titre, anne)
10 );
```

Section 16

Vérification du schéma d’une table

Vérification du schéma d'une table

Il y a deux manières de vérifier le schéma d'une table en SQLite.

L'instruction `.schema` affiche l'instruction SQL utilisée pour créer la table.

```
sqlite> .schema film
CREATE TABLE film(
titre char(100),
annee int,
duree int,
genre char(10),
nom_studio char(30) references studio(nom),
num_prod int references producteur(numero),
constraint film_pk primary key (titre, anne)
);
```

Vérification du schéma d'une table

L'instruction :

```
PRAGMA table_info(nom_table)
```

affiche le schéma de la table enregistré dans le catalogue de la base de données :

```
sqlite> .headers on
sqlite> .mode column
sqlite> pragma table_info(film);
cid         name        type        notnull     ...
----------  ----------  ----------  ----------
0           titre       char(100)   0
1           annee       int         0
2           duree       int         0
3           genre       char(10)    0
4           nom_studio  char(30)    0
5           num_prod    int         0
```

Vérification du schéma d'une table

L'instruction :

```
PRAGMA foreign_keys_list(nom_table)
```

affiche la liste de clés étrangères d'une table :

```
sqlite> .headers on
sqlite> .mode column
sqlite> pragma foreign_key_list(participe);
id          seq         table       from        to
----------  ----------  ----------  ----------  --------
0           0           film        titre_film  titre
0           1           film        annee_film  annee
```

Très, très important ! ! !

Très important : La vérification des clés étrangères n'est pas activé par défaut en SQLite !

Pour que la vérification soit activé, il est nécessaire d'exécuter l'instruction :

```
PRAGMA foreign_keys = true;
```

Pour savoir si la vérification des clés étrangères est activée :

```
sqlite> pragma foreign_keys;
0
```

Activation :

```
sqlite> pragma foreign_keys = true;
sqlite> pragma foreign_keys;
1
```

Maintenant, la vérification des clés étrangères est activée.

Cours 5

Requêtes simples en SQL

Sommaire

17. Requêtes simples en SQL

18. Requêtes avec plus d’une relation

19. Sémantique des requêtes

20. Opérations entre requêtes

Section 17

Requêtes simples en SQL

Un premier exemple

Soit le schéma :

film(titre, annee, duree, genre, nom_studio*, num_prod*)

acteur(nom, adresse, genre, date_naissance)

participe(nom_acteur*, titre_film*, annee_film*)

studio(nom, adresse, nom_prod*)

producteur(numero, nom, adresse)

Nous avons déjà vu l'instruction `select` pour consulter une table. Voici un exemple un peu plus complexe :

```
1 select *
2 from film
3 where nom_studio = 'MGM' and annee > 1990;
```

Un premier exemple

- La clause `select` spécifie les colonnes à consulter. Dans l'exemple, nous avons utilisé `*`, qui signifie que toutes les colonnes seront présentes dans le résultat.
- La clause `from` spécifie les tables à consulter. Dans l'exemple, nous avons consulté une seule table, la table `film`.
- La clause `where` spécifie les lignes à consulter. Dans l'exemple, nous avons consulté uniquement les lignes dont la colonne `nom_studio` est égal à `MGM` et dont la colonne annee est supérieure à `1990`.

Un premier exemple

Sortie :

titre	annee	duree	genre	nom_studio	num_
The Hobbit	2014	114	fantastique	MGM	3982
RoboCop	2014	117	scifi	MGM	8364
⋮					

Projection en SQL

Il est possible d'indiquer les colonnes souhaitées dans la sortie de l'instruction `select` :

```
1 select titre, annee
2 from film
3 where nom_studio = 'MGM' and annee > 1990;
```

Sortie :

titre	annee
The Hobbit	2014
RoboCop	2014

Renommage en SQL

Il est également possible de renommer les colonnes dans la sortie de l'instruction `select` :

```
1 select titre as nom, annee as sortie
2 from film
3 where nom_studio = 'MGM' and annee > 1990
```

Sortie :

nom	sortie
The Hobbit	2014
RoboCop	2014

Renommage en SQL

L'instruction `select` accepte également des expressions à la place des colonnes :

```
1 select titre as nom, duree * 0.0167 as nombre heures
2 from film
3 where nom_studio = 'MGM' and annee > 1990
```

Sortie :

nom	nombre heures
The Hobbit	1.9038
RoboCop	1.9539

Restriction en SQL

La restriction en SQL est faite avec la clause `where` de l'instruction `select`.

Les expressions acceptées par la clause `where` sont celles crées avec (mais pas seulement) les opérateurs suivants :

Logiques	`and, or, not, case, like`
Comparaison	$=, <>, <, >, <=, >=$
Arithmétiques	$+, -, *, /, \%$
Concaténation	$\|\|$

Exemple « Tous les films de MGM après 1970 ou dont la durée ne dépasse pas 90 minutes » :

```
1 select title
2 from film
3 where (annee > 1970 or duree <= 90) and
4     nom_studio = 'MGM';
```

L’opérateur `like`

L’opérateur `like` sert à faire du « pattern matching » :

```
1 select title
2 from film
3 where titre like 'Star ____';
```

titre
Star Wars
Star Trek
⋮

```
1 select titre
2 from film
3 where titre like '%m%';
```

titre
Dogma
Terminator
⋮

La valeur `null`

Si un opérateur arithmétique rencontre la valeur `null`, le résultat de l'opération est `null`. Soit $x =$ `null` :

$1 + x$	résultat : `null`
$0 * x$	résultat : `null`
$x - x$	résultat : `null`

Si un opérateur de comparaison rencontre la valeur `null`, le résultat de l'opération est unknown. Soit $x =$ `null` :

$1 > x$	résultat : unknown
$(1 > x)$ `or` $(1 <= x)$	résultat : unknown
$x = x$	résultat : unknown

Pour tester si x est `null` :

`1 is null`	résultat : `false`
x `is null`	résultat : `true`

La valeur unknown

Deux requêtes surprenantes :

```
sqlite> .nullvalue '(null)'
sqlite> create table t(a int);
sqlite> insert into t values (1);
sqlite> insert into t values (2);
sqlite> insert into t values (null);
sqlite> select * from t;
1
2
(null)
sqlite> select * from t where a < 0 or a >= 0;
1
2
sqlite> select * from t where a = 0 or a <> 0;
1
2
```

La valeur unknown

x	y	x and y	x or y	not x
true	true	true	true	false
true	unknown	unknown	true	false
true	false	false	true	false
unknown	true	unknown	true	unknown
unknown	unknown	unknown	unknown	unknown
unknown	false	false	unknown	unknown
false	true	false	true	true
false	unknown	false	unknown	true
false	false	false	false	true

En effet, soit true $= 1$, unknown $= \frac{1}{2}$ et false $= 0$, nous avons :

- x and $y = min(x, y)$
- x or $y = max(x, y)$
- not $x = 1 - x$

Ordination

La clause `order by` sert à ordonner le résultat de l'instruction `select` :

```
1 select titre
2 from film
3 order by titre;
```

titre
Dogma
RoboCob
Star Trek
Star Wars
Terminator
The Hobbit

Requêtes SQL et l'algèbre relationnelle

En effet, les requêtes que nous avons fait jusqu'ici sont du type :

$$\texttt{select}\ P\ \texttt{from}\ R\ \texttt{where}\ C$$

Ceci équivaut à l'expression :

$$\pi_P(\sigma_C(R))$$

Section 18

Requêtes avec plus d’une relation

Produits et jointures

Il est possible de lister plus d'une relation dans la clause `from`. Dans ce cas, nous pouvons référencer n'importe quelle attribut de ces relations dans les clauses `select` et `where`.

Par exemple, soit le schéma :

`film(titre, annee, duree, genre, nom_studio*, num_prod*)`

`producteur(numero, nom, adresse)`

« Quel est le nom du producteur de Star Wars ? » :

```
1 select nom
2 from film, producteur
3 where titre = 'Star Wars' and num_prod = numero;
```

Désambigüisassion des attributs

Soit le schéma :

> `acteur`(`nom`, `adresse`, `genre`, `date_naissance`)
>
> `producteur`(`numero`, `nom`, `adresse`, `genre`)

« Quels sont les acteurs et producteurs résidant à la même adresse ? » :

```
1 select acteur.nom, producteur.nom
2 from acteur, producteur
3 where acteur.adresse = producteur.adresse
```

Utilisation des alias

« Quels sont les acteurs résidants à la même adresse ? » :

```
1 select a1.nom, a2.nom
2 from acteur a1, acteur a2
3 where a1.adresse = a2.adresse and
4       a1.nom < a2.nom;
```

Pourquoi nous avons utilisé ici < au lieu de <> ?

Section 19

Sémantique des requêtes

Sémantique sous forme de boucle

```
for t1 in R1:
  for t2 in R2:
    ...
      for tn in Rn:
        if t1, t2, ..., tn satisfait la clause where:
          evalue l'expression de la clause select
          produit le tuple de valeurs resultant
```

Un résultat non intuitif

Soi le schéma :

$$R(A)$$
$$S(A)$$
$$T(A)$$

« Quels sont les lignes de R qui sont également présentes dans S ou bien T ou bien les deux ? » :

```
1 select R.A
2 from R, S, T
3 where R.A = S.A or R.A = T.A;
```

En effet, ceci est incorrecte !

Si la relation T est vide, l'instruction `if` dans la dernière boucle `for` n'est jamais exécuté.

Nous pouvons néanmoins utiliser l'opérateur `union` pour résoudre ce problème.

Section 20

Opérations entre requêtes

Intersection

« Quelles sont les adresses des actrices qui sont aussi de productrices ? » :

```
(
    select nom, adresse
    from acteur
    where genre = 'F'
)
intersect
(
    select nom, adresse
    from producteur
);
```

Différence

»Quelles sont les acteurs qui ne sont pas de producteurs ? » :

```
(
    select nom, adresse
    from acteur
)
except
(
    select nom, adresse
    from acteur
);
```

Union

```
(
    select titre, annee
    from film
)
union
(
    select titre_film as titre, annee_film as annee
    from acteur
);
```

Cours 6

Requêtes complexes

Sommaire

21. Sous-requêtes

22. Opérations sur les relations

Section 21

Sous-requêtes

Sous-requêtes

En effet, nous avons déjà rencontré des sous-requêtes, quand nous avons vu les opérateurs d'union, intersection et différence.

Il y a d'autres manières d'utiliser les sous-requêtes :

1. Une sous-requête peut retourner une valeur, qui peut être utilisée dans une clause `where`.
2. Une sous-requête peut retourner une relation, qui peut aussi être utilisée dans une clause `where`.
3. Une sous-requête peut être utilisée dans la clause `from`, comme s'il s'agissait d'une table.

Sous-requêtes produisant une valeur

Quand une requête retourne un seul tuple avec un seul attribut, ceci peut être utilisé comme une constante dans une requête.

Exemple :

```
sqlite> select nom
   ...> from film, studio
   ...> where titre = 'Star Wars' and annee = 1977 and
   ...>     nom = nom_studio;
Fox
```

```
sqlite> select adresse
   ...> from studio
   ...> where nom = (
   ...>     select nom
   ...>     from film, producteur
   ...>     where titre = 'Star Wars' and annee = 1977 and
   ...>         nom = nom_studio
   ...>);
Los Angeles, CA, US
```

Sous-requêtes produisant une valeur

Notez que, utilisée de cette manière, la sous-requête doit retourner une et une seule valeur.

Si zéro valeurs, ou bien plus d'une valeur sont retournées par la sous-requête, l'instruction produit une erreur.

Sous-requêtes produisant plusieurs valeurs

Quand une sous-requête produit plus d'une valeur, ceci est vu comme une relation. Dans ce cas, nous pouvons utiliser une des opérations suivantes :

1. `exists` R :
   vrai si et seulement si R n'est pas vide.
2. `not exists` R :
   vrai si et seulement si R est vide.
3. t `in` R :
   vrai si et seulement si le tuple t appartient à R.
4. t `not in` R :
   vrai si et seulement si le tuple t n'appartient pas à R.

Sous-requêtes produisant plusieurs valeurs

Exemple :

```
sqlite> select nom, adresse
   ...> from studio
   ...> where nom in (
   ...>    select nom_studio
   ...>    from film
   ...>    where (titre, annee) in (
   ...>        select titre_film, annee_film
   ...>        from participe
   ...>        where nom_acteur = 'Ford'
   ...>    )
   ...> );
nom          adresse
----------   --------------------
Fox          Los Angeles, CA, US
Warner Bro   Burbank, CA, US
```

Sous-requêtes dans la clause `from`

Une sous-requête peut aussi être utilisée dans la clause `from`.

Dans ce cas, il est nécessaire de lui donner un alias.

Exemple :

```
sqlite> select nom, adresse
   ...> from studio, (
   ...>     select nom_studio
   ...>     from film, participe
   ...>     where titre = titre_film and
   ...>           annee = annee_film and
   ...>           nom_acteur = 'Ford'
   ...>     ) s
   ...> where nom = s.nom_studio;
nom              adresse
---------------  ---------------
Warner Brothers  Burbank, CA, US
Fox              Los Angeles, CA
```

Produit cartésien

Le produit cartésien en SQL est appelée « cross join »ou bien « join » :

```
1 select *
2 from film cross join participe;
```

Ceux-ci produisent exactement le même résultat :

```
1 select *
2 from film join participe;
```

```
1 select *
2 from film, participe;
```

θ-jointure

La θ-jointure en SQL peut-être faite de deux manière différentes :

La première nous avons déjà rencontré :

```
1 select *
2 from film, participe
3 where titre = titre_film and annee = annee_film;
```

La deuxième utilise l'opérateur de manière explicite :

```
1 select *
2 from film join participe on
3      titre = titre_film and annee = annee_film;
```

Jointure naturelle

Rappelons-nous des différences entre la jointure naturelle et la θ-jointure :

1. Dans la jointure naturelle, la condition porte sur l'ensemble des attributs en commun entre les deux relations.
2. Dans la jointure naturelle, les attributs redondant ne font pas partie du résultat.

Soit le schéma :

$$\texttt{film}(\texttt{titre}, \texttt{annee}, \texttt{genre}, \texttt{nom\_studio}, \texttt{num\_prod})$$

$$\texttt{participe}(\texttt{titre}, \texttt{annee}, \texttt{nom})$$

$$\texttt{acteur}(\texttt{nom}, \texttt{adresse}, \texttt{genre})$$

Nous pouvons faire :

```
1 select * from film natural join participe;
```

```
1 select * from participe natural join acteur;
```

Jointures externes

Les trois types de jointures externes sont présentes en SQL :

```
1 select * from film natural full outer join participe;
```

```
1 select * from film natural left outer join participe;
```

```
1 select * from film natural right outer join participe;
```

Note sur SQLite

Note : SQLite n'implémente pas le `natural full outer join` ni le `natural right outer join`.

Pourtant, il est possible de les simuler.

Comment peut-on faire la jointure externe à droite de `film` et `participe` ?

```
1 select * from participe natural left outer join film;
```

Comment peut-on faire la jointure externe complète ?

```
1 (select * from film natural left outer join participe)
2 union
3 (select * from participe natural left outer join film);
```

Section 22

Opérations sur les relations

Élimination des lignes dupliquées

En effet, SQL n'élimine pas les lignes dupliquées.

Ce comportement rend SQL beaucoup plus efficace.

Pour que les lignes dupliquées soient éliminées, nous devons utiliser `distinct` :

```
1 select distinct *
2 from film
3 where annee 1980
```

Élimination des lignes dupliquées

Néanmoins, les opérations d'union, intersection et différence éliminent toujours les lignes dupliqués.

Nous devons utiliser le mot clé `all` pour « réintroduire » les lignes dupliquées :

```
1 select * from film union all select * from film;
```

```
1 select * from t1 intersect all select * from t2;
```

```
1 select * from t1 except all from select * from t2;
```

Opérateurs d'agrégation

SQL contient quelques opérateurs d'agrégation :

- `count` : compte le nombre d'éléments.
- `avg` : calcule la moyenne.
- `min` : calcule le minimum.
- `max` : calcule le maximum.
- `sum` : calcule sa somme.

Compte le nombre de lignes dans la table `film` :

```
1 select count(*)
2 from film;
```

Calcule la moyenne des prix :

```
1 select avg(prix)
2 from pc;
```

Groupement

La clause `group by` peut être utilisée pour grouper les valeurs d'une table.

Exemple : « La somme des durées de tous les films de chaque studio. »

```
1 select nom_studio, sum(duree)
2 from film
3 group by nom_studio;
```

Quand il y a un opérateur d'agrégation, seulement deux types de termes peuvent apparaître dans la clause `select` :

1. Les agrégations. Celles-ci sont évaluées de manière groupées.
2. Les attributs (dans l'exemple, `nom_studio`). Ceux-ci doivent apparaître dans la clause `group by`.

Groupement

Notez que l'expression :

```
1 select nom_studio
2 from film
3 group by nom_studio;
```

est équivalente à :

```
1 select distinct nom_studio
2 from film;
```

Groupement

Il est également possible d'utiliser `group by` dans une requête avec plusieurs tables. Le comportement est le suivant :

1. Calcule relation R à partir des clauses `from` et `where`. C'est à dire, calcule le produit cartésien des relations dans la clause `from` et restreint par la condition dans la clause `where`.
2. Regroupe les tuples de R en accord avec la clause `group by`.
3. Produit le résultat des attributs de la clause `select`.

Exemple : « Quel est la somme des durées des films de chaque producteur ? »

```
1 select nom, sum(duree)
2 from producteur, film
3 where numero = num_prod
4 group by nom;
```

Les valeurs `null`

Les règles pour les valeurs `null` :

- La valeur `null` est ignorée dans les agrégations. Cela ne compte pas pour les sommes et les moyennes et cela ne peut pas être le minimum ou le maximum. Donc, `count(a)` compte seulement les valeurs non-nulles de la colonne a.
- Pourtant, la valeur `null` compte pour les groupements. C'est à dire, il est possible d'avoir un groupe `null`.
- À l'exception de `count`, toute agrégation d'une colonne vide retourne la valeur `null`. Le `count` d'une colonne vide retourne 0.

Les valeurs null

Soit la table :

R :

a	b
null	null

Nous avons :

```
sqlite> .nullvalue '(null)'
sqlite> select a, count(b) from R group by a;
(null)|0
sqlite> select a, sum(b) from R group by a;
(null)|(null)
```

La clause having

La clause having sert à restreindre une requête en se basant sur le résultat d'une agrégation.

Exemple : « La somme des durées de films de chaque producteur qui a produit au moins un film avant 1930. »

```
1 select nom, sum(duree)
2 from producteur, film
3 where numero = num_prod
4 group by nom
5 having min(annee) < 1930;
```

La clause having

Quelques règles à propos des clauses having :

- L'agrégation dans une clause having s'applique uniquement aux tuples du groupe en question.
- Tout attribut des relations dans la clause from peut être agrégé dans la clause having, mais seulement les attributs dans la clause group by peuvent apparaître non-agrégés dans la clause having.

Cours 7

Autres fonctionnalités d’un SGBD

Sommaire

23. Vues

24. Index

25. Transactions

26. Déclencheurs

Section 23

Vues

Vue

Une vue est comme une table sauf qu'elle ne contient pas de données.

Les données d'une vue sont sauvegardées dans d'autres tables et sélectionnées avec `select`.

Une vue sert à :

- Limiter l'accès à toutes les données d'une table.
- Accélérer certaines requêtes.

Création d'une vue

Syntaxe simplifiée :

```
CREATE VIEW nom_vue (col1, col2, ...) AS instr_select
```

Exemple :

```
1 create view films_cette_annee as (
2     select titre
3     from film
4     where annee = 2016
5 );
```

Une vue est stockée comme une requête.

Une modification dans les tables d'origine provoquera des modifications dans la vue.

Utilisation d'une vue

Une vue peut être utilisée dans de requêtes comme une table :

```
sqlite> create view films_cette_annee as (
  ... >     select titre
  ... >     from film
  ... >     where annee = 2016
  ... > );

sqlite> .tables
film       films_cette_annee

sqlite> select * from films_cette_annee;


...

sqlite> select nom
  ... > from producteur natural join film
  ... > where titre in films_cette_anne;


...
```

Suppression d’une vue

Syntaxe simplifiée :

```
DROP VIEW nom_vue
```

Exemple :

```
1 drop view films_cette_annee;
```

Lorsqu’une vue est supprimée, les données ne sont pas perdues.

Section 24

Index

Création d'un index

Comme pour un livre, un index sert à accéder aux données recherchées le plus rapidement possible.

Syntaxe simplifiée :

```
CREATE INDEX nom_index ON nom_table (col_1, ...)
```

Exemple :

```
sqlite> create index film_idx on film(titre, annee);
sqlite> .indexes
film_idx
```

Les index ne sont pas indispensables, ils servent uniquement à accéder plus rapidement aux tables volumineuses.

Suppression d’un index

La suppression d’un index n’entraine pas de parte de données.

Syntaxe simplifiée :

```
DROP INDEX nom_index
```

Exemple :

```
sqlite> drop index film_idx;
```

Avantages et inconvénients des index

Un index peut être créé sur un ou plusieurs attributs d'une table.

Un index n'améliore pas les accès sur de petites tables.

Un index améliore beaucoup les accès pour les tables volumineuses notamment lors des jointures et des requêtes portant sur les attributs indexés.

Les index ralentissent les insertions car cela implique des calculs supplémentaires.

Section 25

Transactions

Transactions

Une transaction regroupe un ensemble d'instructions SQL qui sont exécutées de façon atomique.

Une transaction fait passer une BD d'un état cohérent à un autre état cohérent.

Pourquoi les transactions ?

- La majorité des SGBD sont multi-utilisateurs : Plusieurs utilisateurs accèdent aux mêmes bases de donnée en même temps.
- Les SGBD sont des programmes qui peuvent tomber en panne.
- Dans certains cas, il est nécessaire qu'un ensemble d'instructions SQL soient exécutées toutes ensemble ou bien annulées si au moins l'une n'est pas exécutée correctement.

Transactions

Syntaxe simplifiée :

```
BEGIN TRANSACTION;
    instruction1;
    instruction2;
    ...
{ COMMIT | ROLLBACK }
```

```
sqlite> create table t1(a int);
sqlite> insert into t1 values(1);
sqlite> select * from t1;
1
sqlite> begin transaction;
sqlite> insert into t1 values (2);
sqlite> select * from t1;
1
2
sqlite> rollback;
sqlite> select * from t1;
1
```

Exemple de transaction

Nous voulons créer une nouvelle table `resp` qui contiendra tous les responsables parmi les employés.

Pour faire cela, il faut :

1. Créer la table.
2. Sélectionner les responsables de la table `emp` et les insérer dans la table `resp`.
3. Une fois la table `resp` remplie, il faut supprimer les responsables de la table `emp`.

```
sqlite> begin transaction;
sqlite> create table resp ( ... );
sqlite> insert into resp
   ... >     select * from emp
   ... >     where ...;
sqlite> delete from emp where ...;
sqlite> commit;
```

Exemple de transaction

Dans une application de site de commerce en ligne :

1. On débite le client.
2. On retire l'article acheté du stock.
3. On paye le fournisseur (et on empoche la commission).

```
1 begin transaction;
2     update client
3       set credit = credit - 100 where ...;
4     update stock
5       set qtd = qtd - 1 where ...;
6     update fournisseur
7       set credit = credit + 80 where ...;
8 commit;
```

Exercice

Soit le schéma de la base :

etudiant(num_etu, nom, prenom)
livre(cote, titre, dispo)
emprunt(num_etu, cote, date)

Prêter le livre ayant la cote VG10 à l'étudiant 12.

```
sqlite> begin transaction;
sqlite> insert into emprunt
  ... > values (12, 'VG10', current_date);
sqlite> update livre set dispo = 0 where cote = 'VG10';
sqlite> commit;
```

Section 26

Déclencheurs

Déclencheurs (triggers)

Un déclencheur (ou « trigger » en anglais) est un mécanisme permettant d'effectuer un traitement à chaque fois qu'un événement donné (`insert`, `delete`, ...) se produit dans la base.

Un déclencheur est utilisé généralement pour garantir la cohérence de la BD ou pour automatiser certains traitements à réaliser après l'occurrence d'un événement donné.

Création de déclencheurs

Syntaxe simplifiée :

```
CREATE TRIGGER nom_trigger
temps evenement ON nom_table
[ WHEN expression ]
BEGIN
    instr_1;
    instr_2;
    ...
END
```

temps: { BEFORE | AFTER | INSTEAD OF }

evenement: { DELETE | INSERT | UPDATE [OF col1, ...] }

instr_n: { DELETE... | INSERT... | UPDATE... | SELECT... }

Déclencheurs

Suppression de déclencheurs :

Syntaxe simplifiée :

```
DROP TRIGGER nom_trigger
```

Visualisation :

```
1 sqlite> create trigger tr_1 ...
2 sqlite> select name from sqlite_master
3   ... > where type = 'trigger';
4 tr_1
```

Exemple de déclencheur

Soit le schéma :

client(num_cli, nom, prenom, adresse)
commande(num_comm, num_cli*, adresse)

Le déclencheur qui actualise les adresse des commandes effectuées par le client :

```
1 create trigger maj_adresse
2 after update of adresse on client
3 begin
4     update commande set adresse = new.adresse
5     where num_cli = old.num_cli;
6 end;
```

Exemple de déclencheur

Un déclencheur sur une vue :

```
1  create table client(
2      num_cli int primary key,
3      nom varchar,
4      adresse varchar
5  );
6
7  create view client_adresse as (
8      select num_cli, adresse from client
9  );
10
11 create trigger maj_client
12 instead of update of adresse on client_adresse
13 begin
14     update client set adresse = new.adresse
15     where num_cli = new.num_cli
16 end;
```

Exercice

Soit le schéma :

emp(nss, num_dept*, nom, prenom)
dept(num_dept, nom)

Créez un déclencheur capable de supprimer tous les employés d'un département quand celui ci est supprimé.

```
1 create trigger supp_emp
2 after delete on dept
3 begin
4     delete from emp where num_dept = old.num_dept;
5 end;
```

Notez que la même chose peut être obtenu avec l'option ON DELETE CASCADE de la clé étrangère.

Et plus encore...

SQL – ce que nous n’avons pas vu :

- Utilisateurs et droits
- Optimisation
- Procédures stockées et fonctions
- Journalisation
- Gestion de la concurrence
- Sécurité
- …

Cours 8

Modèle entité-associations

Sommaire

27. Concepts de modélisation

28. Concepts de base du modèle EA

29. Démarche générale de la modélisation EA

30. Traduction vers le modèle relationnel

Section 27

Concepts de modélisation

Modélisation

Lors de la conception d'une base de donnée, nous posons souvent quelques questions :

- Comment modéliser les données d'un problème particulier ?
- Comment concevoir une base de données pour gérer les données de ce problème ?

Quelques éléments de réponse peuvent être trouvés :

- Analyse et conception des systèmes d'informations (Merise, etc.)
- Modélisation : première étape de la solution !

Niveaux de modélisation

Les données d'une BD peuvent être décrites à plusieurs niveaux d'abstraction.

La norme ANSI/SPARC préconise trois principaux niveaux d'abstraction :

- Niveau interne.
- Niveau conceptuel.
- Niveau externe.

Niveau interne

Le **niveau interne (physique)** est propre au SGBD utilisé pour la mise en oeuvre de la BD.

Ce niveau concerne :

- Le codage et la représentation machine des données.
- Les fichiers utilisés, index, chemins d'accès, les disques.
- Les méthodes d'organisation et d'accès aux données.

Niveau conceptuel

Au **niveau conceptuel**, toutes les données du domaine sont représentées de façon globale mais de manière abstraite et dans un formalisme de modélisation particulier.

On décrit ici la structure logique des données, leurs sémantiques, les relations qu'elles entretiennent.

Dans la méthode Merise, ce niveau est représenté par le MCD (modèle conceptuel des données) et le MLD (modèle logique des données).

À ce niveau, les données sont modélisées indépendamment du SGBD sur lequel sera construite la future BD.

Niveau externe

Le **niveau externe** représente les différentes vues que les différents utilisateurs (utilisateurs finaux, développeurs, programmeurs...) ont sur la BD.

Une vue concerne la partie de la BD visible par un utilisateur ou un programme. Les vues dérivent toutes du schéma conceptuel et constituent une partie de ce dernier.

Modélisation entité-association

Modéliser un phénomène ou une réalité exige :

- Un langage de modélisation qui précise les symboles à utiliser, leurs sémantiques et règles de composition.
- Un travail d'abstraction qui réduit le phénomène à modéliser à une représentation abstraite permettant d'atteindre l'objectif fixé.

Section 28

Concepts de base du modèle EA

Concepts du modèle EA

Un schéma EA permet de mettre en évidence les entités, les associations entre les entités du domaine d'étude et certaines contraintes imposées sur ces associations.

Concepts du modèle EA :

- Entité
- Attribut
- Identifiant
- Association
- Propriété
- Dimension
- Cardinalité

Entité

Une **entité** est une abstraction des objets ou individus du monde réel qui sont :

- pourvus d'existence propre et
- revêtant un intérêt pour le système

Une entité peut être :

- Concrète (étudiant, salle, voiture, etc.)
- Abstraite (module, séjour, vol, etc.)

Convention :

- Les entités sont dénotées par des **noms** et représentés graphiquement par des **rectangles**.

Entité

Exemple d’entité :

Livre
Cote Titre Éditeur Année d’édition ISBN

Entité

Par abus de langage on confond entité et instance ou occurrence.

Pour éviter cela, certains auteurs conviennent d'utiliser les concepts suivants :

- Une entité-type (ou classe d'entités) désigne l'ensemble des entités partageant les mêmes propriétés.
- Ainsi, l'entité-type « Étudiant » regroupe l'ensemble des étudiants.
- Une occurrence (ou instance) d'une entité s'obtient en affectant une valeur à chacun de ses attributs.
- Par exemple, l'étudiant « Toto » représente une occurrence de l'entité-type Étudiant.

Attribut

Un **attribut** est une propriété caractérisant les occurrences d'une entité.

- Un attribut doit revêtir un intérêt pour le système d'information.
- Exemple : « Nom » et « Prénom » sont des attributs de l'entité Étudiant.
- Un attribut peut être atomique (accepte une seule valeur) ou composé (date : jour/mois/année).
- Un attribut peut être mono-valué (ex. Nom) ou multi-valué (ex. Prénoms)

Identifiant

Un **identifiant** est un attribut (ou ensemble d'attributs) qui caractérise chaque instance d'une entité de manière unique.

- Un identifiant est parfois appelé attribut clé.
- Il correspond au concept de clé primaire du modèle relationnel.

Convention :

- Les identifiants sont les premiers dans la liste des attributs et sont **soulignés**.

Attributs

Combien et quels attributs choisir ?

Règle 1 :
Ne retenir que les attributs présentant un intérêt pour le système considéré.

Règle 2 :
Ne pas retenir des attributs qui peuvent être déduits à partir des autres attributs, pour éviter les redondances (ex. : Âge et Date de naissance).

Toutefois, il faut tenir compte des possibles évolutions du système !

Association

Une **association** est une abstraction des relations existantes entre les entités et revêtant un intérêt pour le système d'information.

Une association se caractérise par des propriétés, cardinalités et une dimension.

Convention :

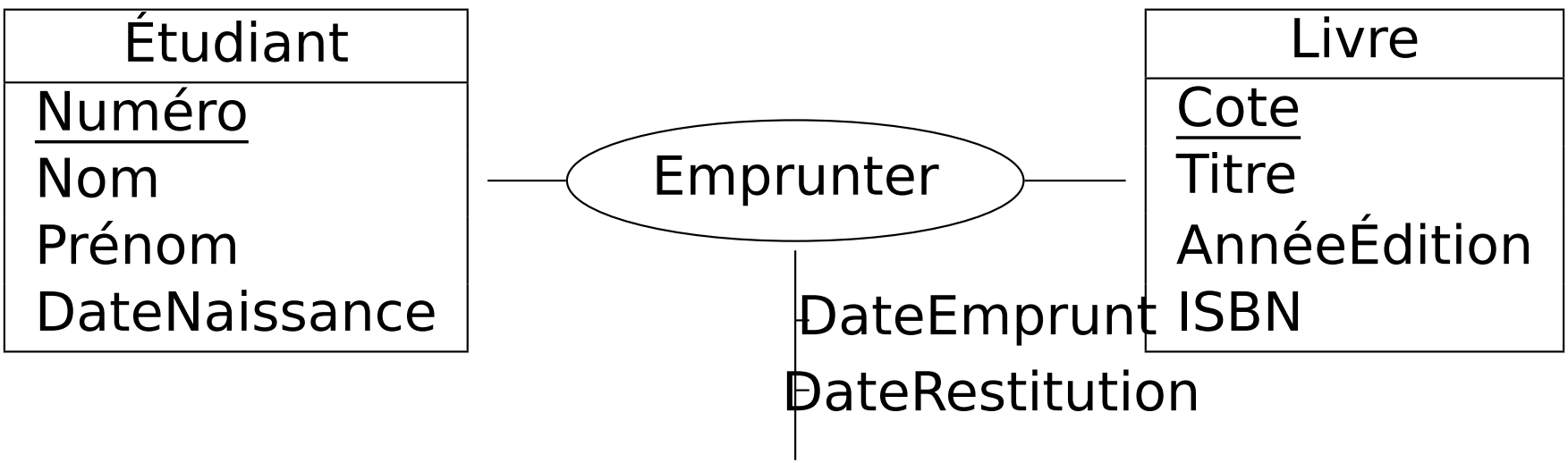
- Les associations sont dénotées par des **verbes** (généralement à l'infinitif) et représentées graphiquement avec des **ellipses**.

Propriétés d’une association

Les **propriétés d’une association** sont des informations importantes mais qui n’existent qu’avec l’existence conjointe des entités impliquées dans cette association.

Convention :

- Une propriété qualifiant deux entités différentes devrait avoir des noms différents (ex. NomEnseignant, NomÉtudiant).



Dimension

Une **dimension (ou degré)** d'une association est le nombre d'entités impliquées par cette association.

Ainsi, il existe des associations de dimension 1 (association unaire).

Une association binaire fait intervenir deux entités différentes.

Une association *n*-aire fait intervenir *n* entités différentes.

Exercice

Donner un exemple d'association unaire, binaire et ternaire.

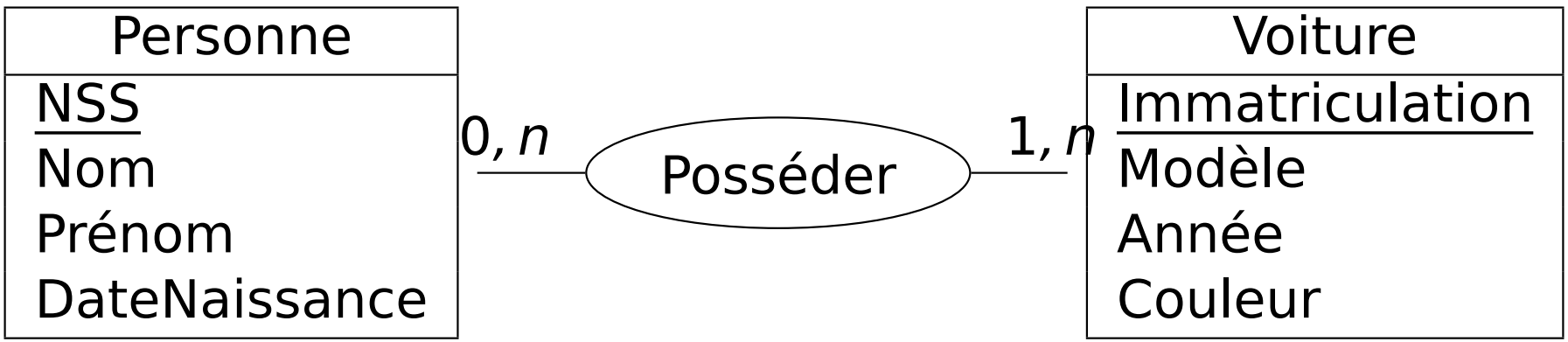
- Unaire :
  - Entité : Personne
  - Association : Épouser
- Binaire :
  - Entités : Étudiant et Promotion
  - Association : Appartenir
- Ternaire :
  - Entités : Enseignant, Promotion, Cours
  - Association : Enseigner

Cardinalités

Les **cardinalités** permettent d'exprimer des contraintes sur le nombre d'occurrences des entités dans une association.

- La **cardinalité minimale** exprime le nombre de fois au minimum qu'une instance d'une entité peut être impliquée dans une association.
- La **cardinalité maximale** exprime le nombre de fois au maximum qu'une instance d'une entité peut être impliquée dans une association.

Cardinalités



Une voiture est la propriété d'une ou plusieurs personnes.

Une personne peut ne pas avoir de voiture comme elle peut en avoir plusieurs.

Cardinalités

Les combinaisons de cardinalités admises sont :

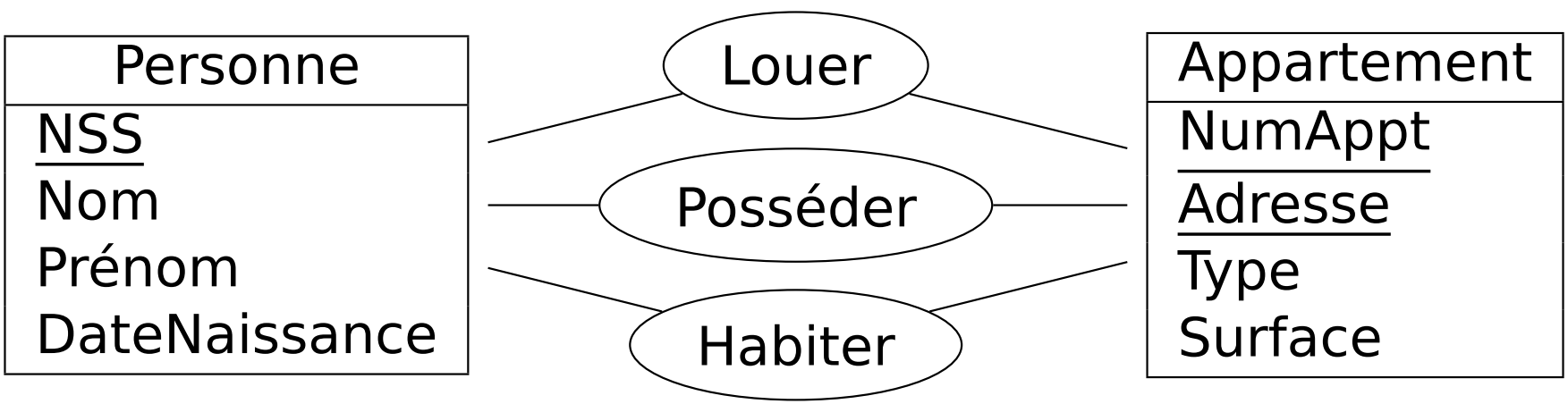
- $0, n$: l'entité peut ne pas apparaître dans l'association comme elle peut apparaître plusieurs fois (un nombre connu ou inconnu). Même si n est connu, il ne faut pas le remplacer par sa valeur.
- $1, n$: l'entité doit apparaître au moins une fois dans l'association mais peut apparaître plusieurs fois.
- $0, 1$: l'entité peut ne pas apparaître dans l'association mais ne peut pas apparaître plus d'une fois dans l'association.
- $1, 1$: l'entité doit apparaître au moins et au plus une seule fois dans l'association.

Remarques

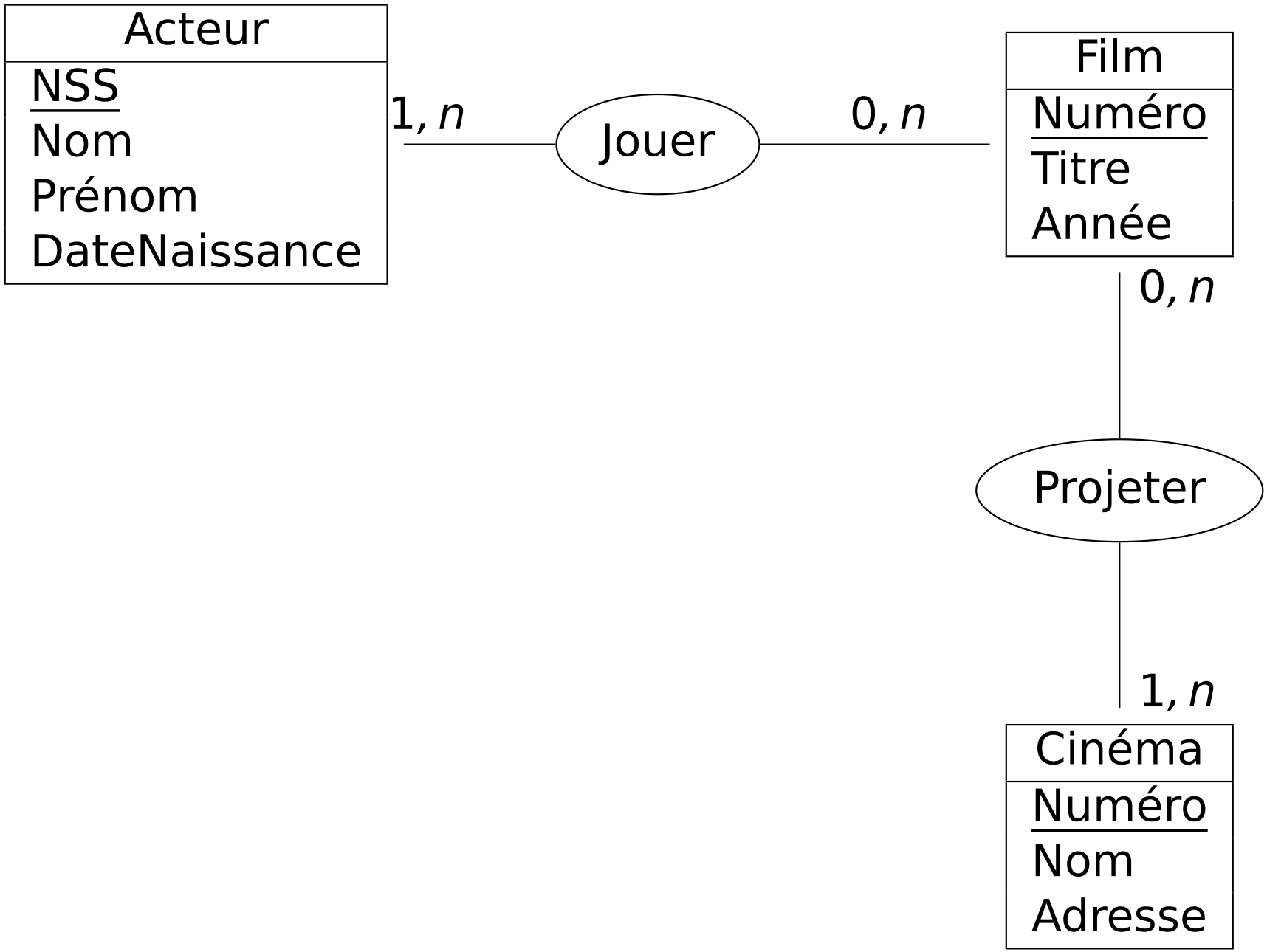
Remarque 1 : L'identifiant d'une association est implicite et peut être formé par juxtaposition des identifiants des entités impliquées auxquels on ajoute parfois d'autres propriétés de l'association.

Remarque 2 : Une association peut ne pas avoir de propriétés.

Remarque 3 : Entre les mêmes entités, il peut exister plusieurs associations
Exemple :



Exercice


Acteur
NSS
Nom
Prénom
DateNaissance
1, n
Jouer
0, n
Film
Numéro
Titre
Année
0, n
Projeter
1, n
Cinéma
Numéro
Nom
Adresse

Exercice

Soit le shcéma ci-dessus :

1. Peut-il y avoir des films où il ne joue aucun acteur ?
2. Peut-il y avoir un acteur qui n'a joué dans aucun film ?
3. Comment peut-on ajouter au schéma ci-dessus les informations suivantes :
   - 3.1 La nationalité de l'acteur.
   - 3.2 Le rôle d'un acteur dans un film.
   - 3.3 La date de projection d'un film dans une salle de cinéma donnée.
4. Comment peut-on ajouter le « réalisateur » d'un film, sachant qu'un réalisateur peut être lui-même un acteur et qu'un film n'a qu'un seul réalisateur ?

Entité faible et entité forte

Une entité est qualifiée de **faible** si l'existence de chacune de ces occurrences est conditionnée par l'existence d'une occurrence d'une autre entité.

Cela correspond aux entités participant à des associations avec des cardinalités $(1, 1)$.

Dans le modèle relationnel, l'identifiant de l'entité faible inclut celui de l'entité dont elle dépend.

Une entité faible n'a pas d'identifiant propre.

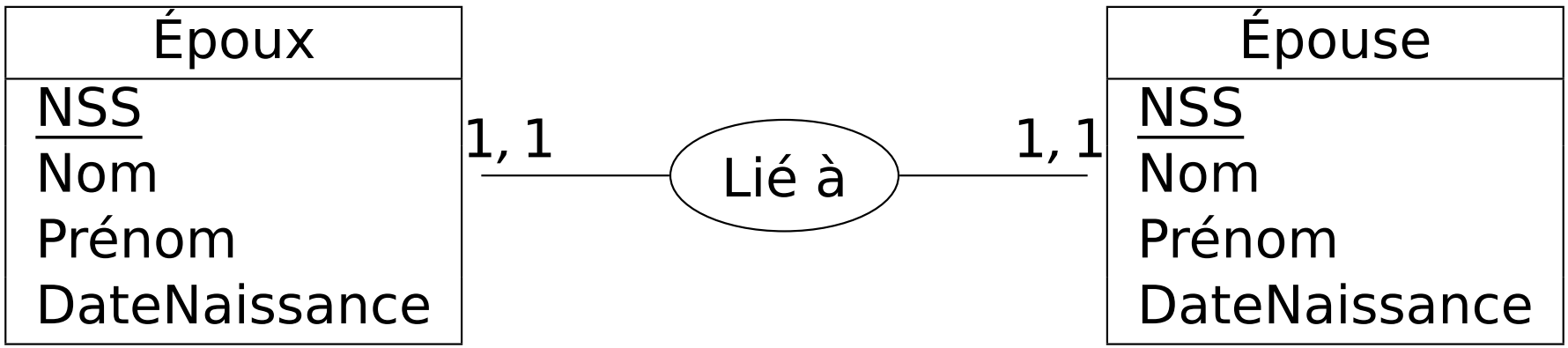
Exemples :

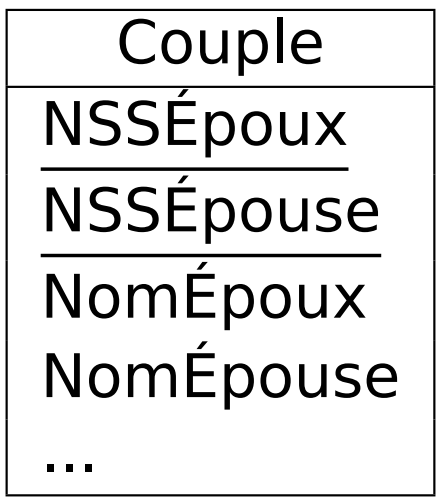
- Chapitre dépend de Livre.
- Appartement dépend de Bâtiment.

L'entité dont dépend l'entité faible est une entité **forte** !

Associations inutiles

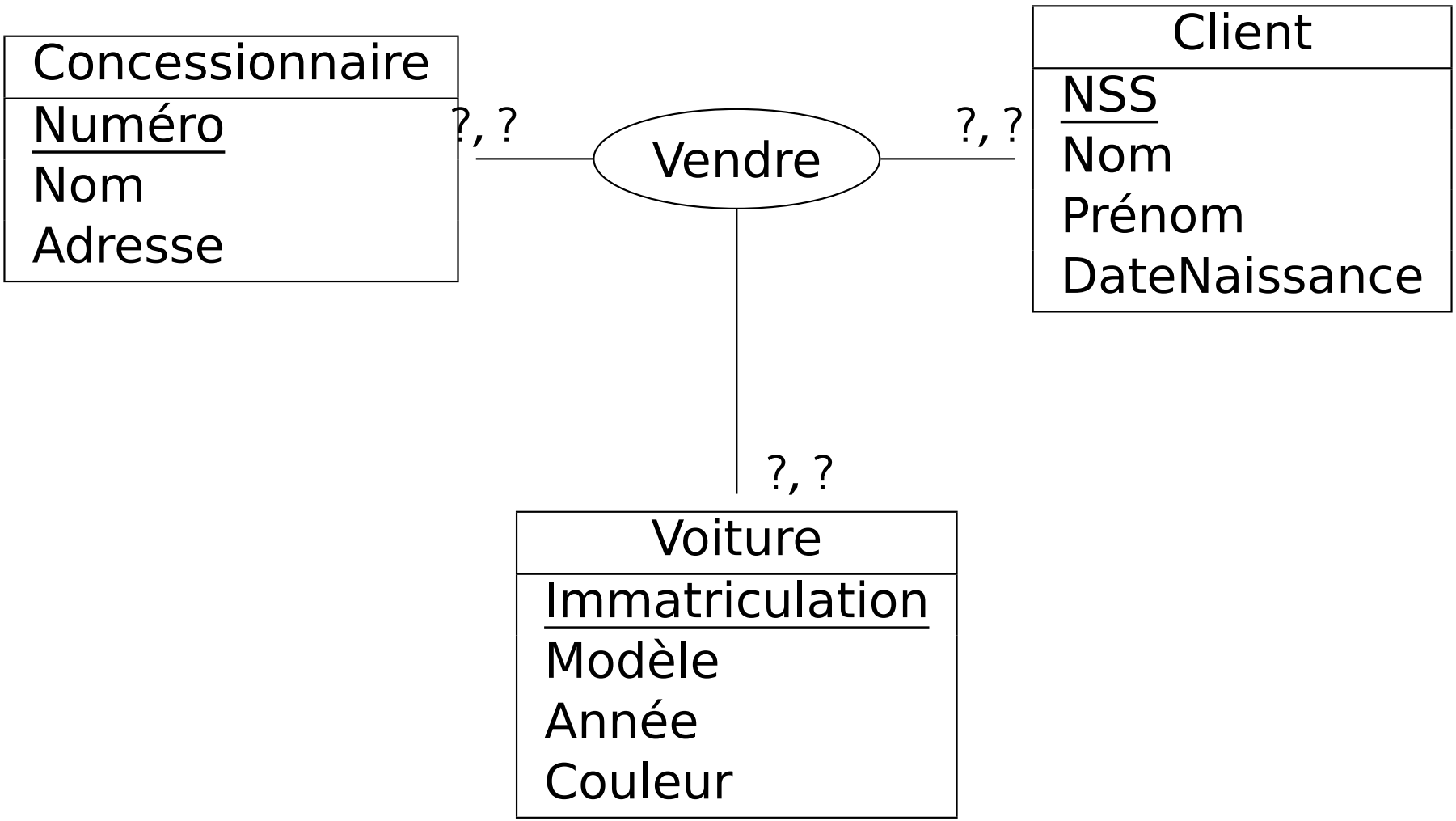
Dans certains cas, une association binaire avec des cardinalité $(1, 1)$ pour les deux entité est **inutile**.



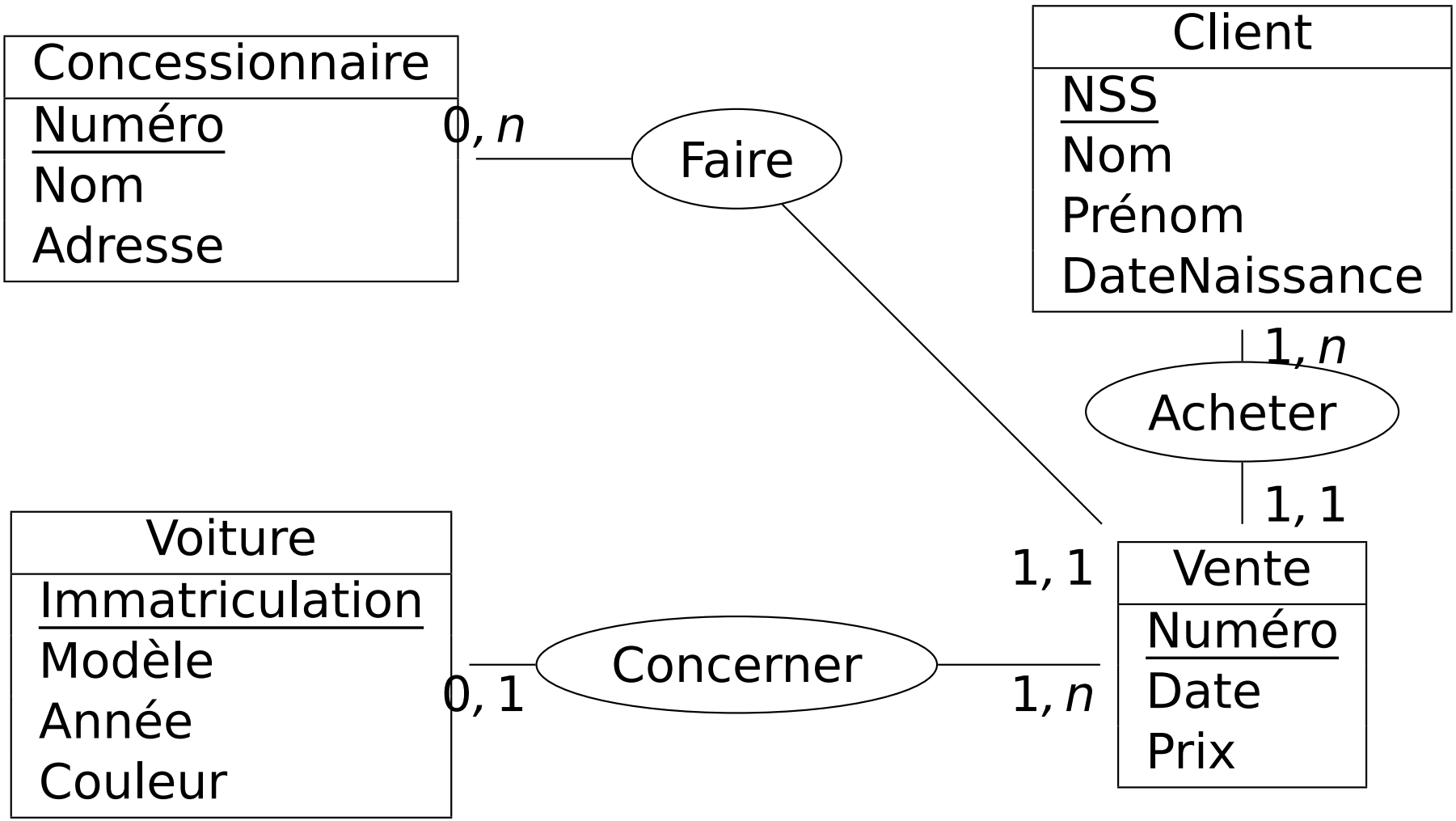


Associations ternaires et plus

Pour faciliter l'expression des cardinalités, l'analyse et la compréhension d'un schéma conceptuel, les associations *n*-aires (avec $n > 2$) sont souvent transformées sous forme d'associations binaires.



Associations ternaires et plus



Section 29

Démarche générale de la modélisation EA

Entités

Démarche pour la conceptions des entités :

1. Repérer les entités d'**intérêt**.
2. Lister les attributs d'**intérêt**.
3. Dégager l'**identifiant naturel** s'il existe ou ajouter un identifiant artificiel (ex. : numéro séquentiel).

Associations

Démarche pour la conception des associations :

1. Repérer les associations d'**intérêt** entre entités.
2. Lister les propriétés de chaque association.
3. Déterminer et mentionner les cardinalités de chaque patte d'une association.
4. Éliminer les synonymes (deux propriétés ne doivent pas porter le même nom dans un même schéma EA).
5. Vérifier que le schéma est **exhaustif** et **non redondant**.

Exercice 1

Soit le système d'informations d'une agence immobilière (location et vente d'appartements). Cette agence identifie chaque appartement par :

- un numéro séquentiel unique,
- son type (T1, T2, ...),
- sa superficie,
- l'étage où se trouve l'appartement et
- le numéro du bâtiment qui abrite cet appartement.

Chaque appartement est la propriété d'un seul propriétaire identifié par :

- un numéro,
- nom,
- prénom,
- adresse et
- numéro de téléphone.

Un bâtiment est caractérisé par un numéro unique et une adresse. Créez le modèle EA de cette agence.

Exercice 2

Modélisation d'un cabinet médical en EA :

- Un médecin est identifié par un numéro de SS, et ayant un nom, un prénom, une adresse. Il prescrit des ordonnances à des patients.
- Chaque ordonnance est datée et peut demander des tests identifiés par un numéro.
- Un patient est identifié par un numéro de SS, et a lui aussi un nom, un prénom et une adresse.
- Un test effectué est identifié par un numéro et a un libellé, un résultat et d'autres caractéristiques.
- Un test se déroule pour un patient donné à une date donnée.
- Lorsque le médecin reçoit les résultats d'un test, il établit un diagnostic qu'il conserve.

Pour conclure sur la modélisation EA

Un modèle EA est un modèle conceptuel décrivant la structure et ne dit rien sur comment les informations sont traitées et par qui.

Les schémas évoluent peu, ce sont les extensions (occurrences) qui évoluent.

La modélisation EA n’est ni évidente ni unique : C’est un ensemble de choix relevant de la perception, l’abstraction et l’intérêt pour le problème considéré et l’objectif final recherché.

Pour conclure sur la modélisation EA

Avantages :

- Peu de concepts : entités, associations, attributs, propriétés, cardinalités, dimension.
- Langage graphique.
- Modélisation et représentation simple, intuitive et rapide.

Inconvénients :

- Difficulté (et parfois impossibilité) d'exprimer certaines contraintes.
- Difficulté de modéliser des problèmes complexes.
- Un modèle décrit les données mais pas comment, quand et qui traite les données.

Section 30

Traduction vers le modèle relationnel

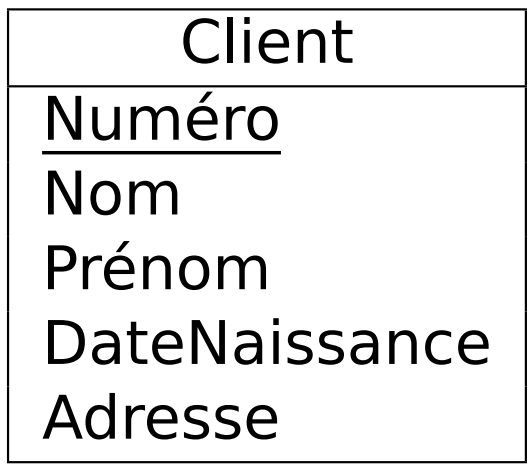
Traduction des entités

Chaque entité du modèle EA devient une relation portant le même nom.

L'identifiant de l'entité sera la clé primaire de la relation.

Chaque attribut de l'entité devient un attribut de la relation.

Exemple :



Devient : `client(numero, nom, prenom, date_naissance, adresse)`

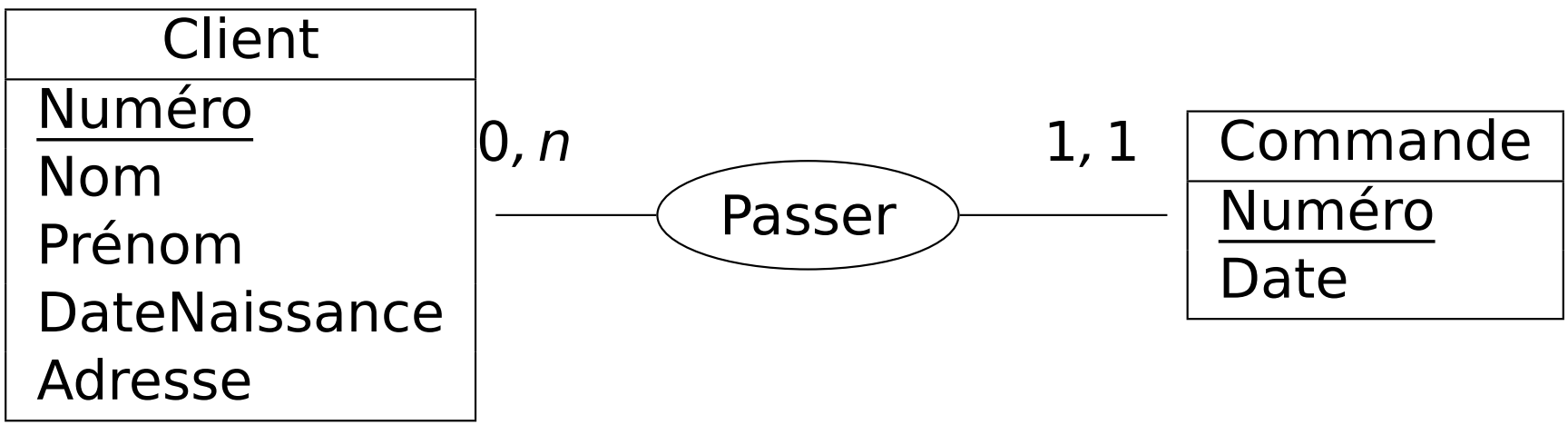
Traduction des associations père-fils

Les associations père-fils (c'est à dire, les associations binaires ayant sur l'une des pattes des cardinalités $0,1$ ou $1,1$) sont supprimées.

La clé de l'entité père deviendra une clé étrangère dans l'entité fils.

Les propriétés de l'association seront des attributs de la relation correspondant à l'entité fils.

Exemple :



`client(`<u>`numero`</u>`, nom, prenom, date_naissance, adresse)`

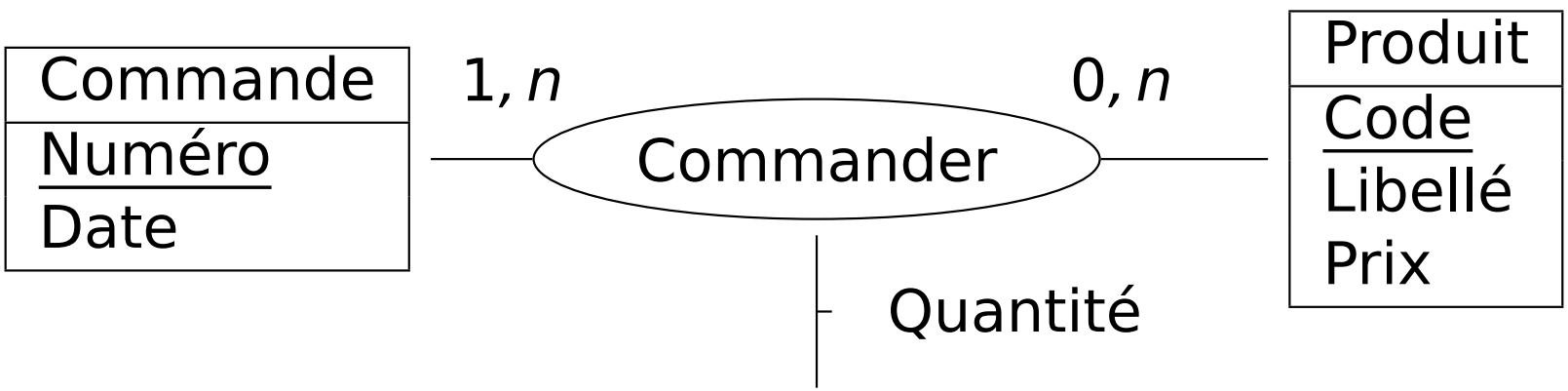
Traduction des associations plusieurs-plusieurs

Pour chaque association plusieurs à plusieurs ($0, n$ ou $1, n$), créer une relation spécialement pour cette association.

La clé primaire de cette relation sera formée par juxtaposition des identifiants des entités concernées par l'association en question.

Les propriétés de cette association seront des attributs de la relation créée.

Exemple :




```
commande(numero, date)
produit(code, libelle, prix)
commander(numero_commande*, code_produit*,
```

Exercice

Un assureur automobile désire construire une base de données pour gérer ses contrats d'assurances et le suivi des accidents automobiles de ses clients. Les clients de cet assureur souscrivent des contrats d'assurances caractérisés par un numéro unique, un montant, une date de début, une durée, le type de contrat. Un contrat concerne une seule voiture caractérisée par son numéro d'immatriculation, sa marque, type, couleur, puissance. Un client peut posséder plusieurs voitures. Un accident a lieu à une date et heure données et concerne une ou plusieurs voitures. Chaque accident est suivi par un seul agent. Chaque véhicule accidenté peut être expertisé par un ou plusieurs experts indépendants.

1. Donner le schéma EA représentant cette agence d'assurance.
2. Déduire le modèle relationnel correspondant.

Cours 9

Dépendances

Sommaire

31. Introduction

32. Dépendances fonctionnelles

33. Dépendances multivaluées

Section 31

Introduction

Introduction

Les dépendances sont à la base du processus de construction des bases de données relationnelles.

La satisfaction des dépendances engendre la redondance des donnés.

Si cette redondance n'est pas éliminé ou parfaitement maîtrisé, diverses anomalies peuvent se produire :

1. Anomalie de l'insertion.
2. Anomalie de la suppression.
3. Anomalie de la mise à jour.

Anomalie de l'insertion

L'insertion d'un type d'objet oblige l'insertion d'un autre type d'objet.

Exemple :

modèle	catégorie	constructeur	pays
Twingo	citadine	Renault	France
C3	citadine	Citroën	France
Mégane	berline	Renault	France
206	citadine	Peugeot	France
Corsa	citadine	Opel	Allemagne
Defender	SUV	Land Rover	Grand Bretagne
NULL	**NULL**	**Mercedez Benz**	**Allemagne**

L'insertion d'un constructeur oblige l'insertion d'une voiture.

Anomalie de la suppression

La suppression d'un type d'objet peut causer la suppression d'un autre type d'objet.

Exemple :

modèle	catégorie	constructeur	pays
Twingo	citadine	Renault	France
C3	citadine	Citroën	France
Mégane	berline	Renault	France
206	citadine	Peugeot	France
Corsa	citadine	Opel	Allemagne
Defender	**SUV**	**Land Rover**	**Grande Bretagne**
NULL	NULL	Mercedez Benz	Allemagne

Il n'est pas possible de supprimer la Defender sans supprimer le constructeur Land Rover de la base.

Anomalie de la mise à jour

La mise à jour d'un tuple peut rendre la base incohérente.

Exemple :

modèle	catégorie	constructeur	pays
Twingo	**citadine**	**Renault**	**France**
C3	citadine	Citroën	France
Mégane	**berline**	**Renault**	**France**
206	citadine	Peugeot	France
Corsa	citadine	Opel	Allemagne
Defender	SUV	Land Rover	Grand Bretagne
NULL	NULL	Mercedez Benz	Allemagne

Si l'on met à jours la Twingo, il faut penser à vérifier tous les autres véhicules du constructeur Renault.

Section 32

Dépendances fonctionnelles

Définition

Soient X et Y deux sous ensembles d'attributs d'une même relation. Y **dépend fonctionnellement** de X (noté $X \rightarrow Y$) si et seulement si, une valeur de X correspond à une et une seule valeur de Y.

Une dépendance fonctionnelle (DF) décrit le fait qu'un sous ensemble d'attributs d'une relation **détermine** un autre sous ensemble d'attributs de cette même relation.

Lorsque Y ne dépend pas fonctionnellement de X, nous utilisons $X \not\rightarrow Y$.

Le schéma de relation $R(U)$, où U est un ensemble d'attributs, muni de l'ensemble F de DF sera noté $(R(U), F)$,

Exemple

Voiture:

immat	marque	modèle	catégorie	couleur
888 HH 62	Citroën	C3	citadine	Noire
777 BH 62	Peugeot	306	berline	Bleue
666 KJ 62	Toyota	Yaris	citadine	Blanche
555 DV 59	Toyota	Hilux	utilitaire	Blanche
222 CS 62	Renault	Mégane	berline	Noire
123 OP 59	Opel	Corsa	citadine	Noire

- L'immatriculation détermine la marque, le modèle, la catégorie et la couleur :

  $$\text{immat} \rightarrow \text{marque, modèle, catégorie, couleur}$$

- Le modèle détermine la marque et la catégorie :

  $$\text{modèle} \rightarrow \text{marque, catégorie}$$

- La marque ne détermine pas le modèle, ni la catégorie ni la couleur :

  $$\text{marque} \not\rightarrow \text{modèle}$$
  $$\text{marque} \not\rightarrow \text{catégorie}$$
  $$\text{marque} \not\rightarrow \text{couleur}$$

Remarques

Les DF d'une relation R sont des contraintes qui doivent être valables quelque soit l'extension de R.

Nous ne pouvons pas déduire des DF à partir d'une extension d'une relation.

Mais nous pouvons invalider des DF sur une extension d'une relation (trouver des contres exemples).

Implication

Soit un schéma de relation $(R(U), F)$ et $X, Y \subseteq U$. Nous avons que F **implique** $X \rightarrow Y$ (noté $F \models X \rightarrow Y$) si et seulement si toute relation R vérifiant F vérifie aussi $X \rightarrow Y$.

Évidement, toute DF de F est impliquée par F. Mais il a d'autres dépendances qui peuvent être inférées.

Exemple :

Soit le schéma de relation $(R(A_1, A_2, A_3), \{A_1 \rightarrow A_2, A_2 \rightarrow A_3\})$

Nous avons que pour chaque valeur de A_1 il y a une unique valeur de A_2.
Nous avons aussi que pour chaque valeur de A_2 il y a une unique valeur de A_3.
Donc, pour chaque valeur de A_1 il y a une unique valeur de A_3.
Donc, nous avons $A_1 \rightarrow A_3$.

Fermeture d'un ensemble de DF

L'ensemble de toutes les DF impliquées par F (noté F^+) est appelée de **fermeture** de F.

Donc, par définition, toute relation vérifiant F vérifie aussi F^+.

Notez aussi que $F^+ = (F^+)^+$.

Équivalence

Soit deux ensembles F et G de DF sur un même un schéma $R(U)$.

G **implique** F (noté $G \models F$) si et seulement si, pour toute DF $f \in F$, nous avons $G \models f$.

F et G sont **équivalents** si et seulement si $F \models G$ et $G \models F$.

Notez que F et F^+ sont équivalents.

Notez aussi que F et G sont équivalents si et seulement si $F^+ = G^+$.

Règles d'inférence

Les règles d'inférence (ou dérivation) ont pour objectif engendrer de nouvelles DF à partir d'un ensemble de DF donné.

Soit le schéma de relation $(R(U), F)$ et soient X, Y, W et Z des sous ensembles non vides de U :

- Réflexivité : si $X \subseteq Y$ alors $F \models Y \rightarrow X$
- Augmentation : $X \rightarrow Y \models XW \rightarrow YW$
- Transitivité : $X \rightarrow Y, Y \rightarrow Z \models X \rightarrow Z$
- Pseudo-transitivité : $X \rightarrow Y, WY \rightarrow Z \models WX \rightarrow Z$
- Union : $X \rightarrow Y, X \rightarrow Z \models X \rightarrow YZ$
- Décomposition : si $Z \subseteq Y$ alors $X \rightarrow Y \models X \rightarrow Z$

Règles d'inférence

Exemple :

Soit le schéma de relation $(R(U), F)$ où :

- $U = \{A, B, C, D, E, G, H, I\}$
- $F = \{AC \rightarrow DE, C \rightarrow ABE, BD \rightarrow C, GBE \rightarrow I, IB \rightarrow H, E \rightarrow A, E \rightarrow B, AB \rightarrow G, ADE \rightarrow H\}$

Réflexivité : $A \rightarrow A$, $B \rightarrow B$, $AB \rightarrow A$, $ABC \rightarrow B$, etc., sont tous des éléments de F^+.

Augmentation : $EB \rightarrow AB \in F^+$ parce que $E \rightarrow A \in F$.

Transitivité : $BD \rightarrow ABE \in F^+$ parce que $BD \rightarrow C \in F$ et $C \rightarrow ABE \in F$.

Décomposition : $C \rightarrow A \in F^+$, $C \rightarrow B \in F^+$ et $C \rightarrow E \in F^+$ parce que $C \rightarrow ABE \in F$.

Union : $E \rightarrow AB \in F^+$ parce que $E \rightarrow A \in F$ et $E \rightarrow B \in F$.

Axiomes d'Armstrong

Les trois propriétés vue en premier :

- Réflexivité
- Augmentation
- Transitivité

sont appelées **axiomes d'Armstrong**.

Les axiomes d'Armstrong forment un ensemble de règles **valide** et **complet**, c.-à-d. :

- Valide : toute DF obtenue par l'application des ces règles à d'autres DF de F^+ appartiennent aussi à F^+.
- Complet : l'application répété de ces règles engendre la totalité de F^+.

Cela veut dire que F^+ peut être calculé à partir de F en utilisant les axiomes d'Armstrong.

Fermeture d'un sous-ensemble d'attributs

Soit le schéma de relation $(R(U), F)$, où $X \subseteq U$.

La **fermeture** de X (notée X^+) est l'ensemble de tous les attributs déterminés par X.

Notez donc que $X \rightarrow X^+ \in F^+$, pour tout $X \subseteq U$.

Calcul de la fermeture

L'ensemble X^+ peut être calculé avec l'algorithme suivant :

1. $X^+ = X$
2. S'il existe $Z \rightarrow Y \in F$ tel que $Z \subseteq X^+$ et $Y - X^+ \neq \emptyset$, alors :
   2.1 $X^+ = X^+ \cup Y$
3. Sinon :
   3.1 Retourne X^+.
4. Répéter 2.

Calcul de la fermeture

Exemple :

Soit le schéma de relation $(R(U), F)$, où :

- $U = \{A, B, C, D, E, G, H, I\}$
- $F = \{AC \rightarrow DE, C \rightarrow ABE, BD \rightarrow C, GBE \rightarrow I, IB \rightarrow H, E \rightarrow A, E \rightarrow B, AB \rightarrow G, ADE \rightarrow H\}$

Nous avons :

$$\begin{aligned} E^+ &= \{E\} \\ &= \{E, A\} \\ &= \{E, A, B\} \\ &= \{E, A, B, G\} \\ &= \{E, A, B, G, I\} \\ &= \{E, A, B, G, I, H\} \end{aligned}$$

Nous avons donc $E^+ = \{E, A, B, G, I, H\}$.

Calcul de la fermeture

L'algorithme vue précédemment peut être utilisé pour vérifier si une DF appartient à F^+.

Exemple :

Soit le même schéma de relation $(R(U), F)$, où :

- $U = \{A, B, C, D, E, G, H, I\}$
- $F = \{AC \rightarrow DE, C \rightarrow ABE, BD \rightarrow C, GBE \rightarrow I, IB \rightarrow H, E \rightarrow A, E \rightarrow B, AB \rightarrow G, ADE \rightarrow H\}$

Nous voulons vérifier si $IE \rightarrow H \in F$:

$$\begin{aligned} \{I, E\}^+ &= \{I, E\} \\ &= \{I, E, A\} \\ &= \{I, E, A, B\} \\ &= \{I, E, A, B, H\} \end{aligned}$$

Comme $H \in \{I, E\}^+$, donc nous pouvons affirmer que $IE \rightarrow H \in F^+$.

Clé candidate

Une **clé candidate** d'une relation R est un sous-ensemble d'attributs X de R tel que :

1. les tuples de n'importe quelle table de R sont tous distincts dans X.
2. si on supprime un seul attribut de X, la condition précédente n'est plus vraie.

Une clé candidate est donc capable de distinguer tous les tuples d'une table.

Exemple :

Soit le schéma de relation :
`Employé(matricule, nom, prénom, date_naissance, nss)`

Les attributs `matricule` et `nss` représentent deux clés candidates différentes.

En revanche, `nom` n'est pas une clé candidate.

Clé candidate

En d'autres mots :

Soit le schéma de relation $R(U)$. X est une **clé candidate** de R si et seulement si $X^+ = U$ et, pour tout $A \in X$, $(X - \{A\})^+ \neq U$.

Clé candidate

Exemple :

Soit le même schéma de relation $(R(U), F)$, où :

- $U = \{A, B, C, D, E, G, H, I\}$
- $F = \{AC \rightarrow DE, C \rightarrow ABE, BD \rightarrow C, GBE \rightarrow I, IB \rightarrow H, E \rightarrow A, E \rightarrow B, AB \rightarrow G, ADE \rightarrow H\}$

Nous avons :

$$\{I, B\}^+ = \{I, B, H\}$$
$$\{B, D\}^+ = U$$
$$\{A, B, C\}^+ = U$$

Donc :

- $\{I, B\}$ n'est pas une clé candidate.
- Comme $B^+ \neq U$ et $D^+ \neq U$, alors $\{B, D\}$ est une clé candidate.
- Comme $C^+ = U$, alors $\{A, B, C\}$ n'est pas une clé candidate.

Super clé

Une **super clé** est un ensemble d'attributs d'une relation qui contient une clé candidate.

Exemple :

Soit le schéma de relation :
`Employé(matricule, nom, prénom, date_naissance, nss)`

L'ensemble `{matricule, nom}` est une super clé.

Clé primaire

Lors qu'un schéma de relation comporte plusieurs clés candidates, il faut choisir une qui sera utilisée pour la distinction des tuples ; elle est nommée **clé primaire**.

Recherche d'une clé candidate

Soit le schéma de relation $(R(U), F)$. L'algorithme suivant sert à chercher une clé candidate pour R :

1. $C = U$
2. Pour chaque $A \in C$:
   2.1 Si $(C - \{A\})^+ = U$ alors :
      2.1.1 $C = C - \{A\}$
3. Retourne C.

Un autre algorithme consiste à commencer par un attribut X et calculer X^+. Si $X^+ \neq U$, alors ajouter un attribut dans X et continuer jusqu'à trouver une clé candidate.

Recherche de toues les clés candidates

Soit le schéma de relation $(R(U), F)$. L'algorithme suivant permet de déterminer toutes les clés candidates :

1. $\mathcal{P} = \{U\}$
2. $L = \emptyset$
3. Pour chaque $S \in \mathcal{P}$:
   - 3.1 Si S est inclus dans un autre élément de $\mathcal{P}$ ou $S^+ \neq U$ alors :
     - 3.1.1 Supprimer S de $\mathcal{P}$.
   - 3.2 Sinon :
     - 3.2.1 Chercher une clé candidate de U dans S. Soit $\{A_1, \ldots, A_n\}$ cette clé.
     - 3.2.2 Ajouter cette clé à L.
     - 3.2.3 Remplacer dans $\mathcal{P}$, S par $S - \{A_1\}, \ldots, S - \{A_n\}$.
4. Retourne L.

Recherche de toutes les clés candidates

Exemple :

Soit le schéma $(R(U), F)$ d'un exemple précédent :

- $\mathcal{P} = \{U\}$
  $L = \emptyset$
- $\{D, E\}$ est clé candidate de U.
  $L = \{\{D, E\}\}$
  $\mathcal{P} = \{\{A, B, C, D, G, H, I\}, \{A, B, C, E, G, H, I\}\}$
- $\{B, D\}$ est clé candidate de U dans $\{A, B, C, D, G, H, I\}$.
  $L = \{\{D, E\}, \{B, D\}\}$
  $\mathcal{P} = \{\{A, B, C, G, H, I\}, \{A, C, D, G, H, I\}, \{A, B, C, E, G, H, I\}\}$
- $\{A, B, C, G, H, I\} \subseteq \{A, B, C, E, G, H, I\}$
  $\mathcal{P} = \{\{A, C, D, G, H, I\}, \{A, B, C, E, G, H, I\}\}$
- $\{C\}$ est clé candidate de U dans $\{A, C, D, G, H, I\}$.
  $L = \{\{D, E\}, \{B, D\}, \{C\}\}$
  $\mathcal{P} = \{\{A, D, G, H, I\}, \{A, B, C, E, G, H, I\}\}$

Recherche de toutes les clés candidates

- $\{A, D, G, H, I\}^+ \neq U$
  $L = \{\{D, E\}, \{B, D\}, \{C\}\}$
  $\mathcal{P} = \{\{A, B, C, E, G, H, I\}\}$
- $\{C\}$ est clé candidate de U dans $\{A, B, C, E, G, H, I\}$
  Mais cette clé est déjà dans L.
  $L = \{\{D, E\}, \{B, D\}, \{C\}\}$
  $\mathcal{P} = \{\{A, B, E, G, H, I\}\}$
- $\{A, B, E, G, H, I\}^+ \neq U$
  $L = \{\{D, E\}, \{B, D\}, \{C\}\}$
  $\mathcal{P} = \emptyset$
  Fin.

Les clés candidates sont donc DE, BD et C.

Dépendance fonctionnelle élémentaire

Une **dépendance fonctionnelle élémentaire** est une dépendance fonctionnelle $X \rightarrow A$ tel que :

- A est un seul attribut ; et
- $A \notin X$; et
- Il n'existe pas $X' \subseteq X$ tel que $X' \rightarrow A$.

Exemple :

Soit le schéma relationnel suivant :

`Produit(`<u>`ref`</u>`, nom, prix)`

Les dépendance suivante sont élémentaires :

1. `ref` $\rightarrow$ `nom`
2. `ref` $\rightarrow$ `prix`

Les dépendance suivantes ne sont pas élémentaires :

1. `ref, nom` $\rightarrow$ `prix` (`{ref}` $\subseteq$ `{ref, nom}` et `ref` $\rightarrow$ `prix`)
2. `ref` $\rightarrow$ `nom, prix` (`{nom, pris}` n'est pas singleton)

Couverture minimale

Puisque plusieurs ensembles de DF différents peuvent être équivalents, il existe parmi ces ensembles un qui contient le plus petit nombre de DF et qui est équivalent aux autres.

Cet ensemble est appelé **couverture minimale** (CM).

La couverture minimale est composée des DF qui permettent de déduire les autres DF redondantes par l'application des règles d'inférence telle que la règle de transitivité.

Couverture minimale

Une DF $X \rightarrow Y$ de F est **redondante** si et seulement si $F - \{X \rightarrow Y\}$ implique $X \rightarrow Y$.

Soit un ensemble F de DF. F_{min} constitue une **couverture minimale** si et seulement si :

1. F_{min} est équivalent à F ; et
2. Les éléments de F_{min} sont tous de DF élémentaires ; et
3. Aucune DF dans F_{min} n'est redondante.

Couverture minimale

Exemple :

Montrez que l'ensemble de DF suivant contient une DF redondante.

$$F = \{A \rightarrow B, BC \rightarrow D, AC \rightarrow BDE, D \rightarrow E\}$$

Réponse :

1.	$A \rightarrow B$	
2.	$BC \rightarrow D$	
3.	$D \rightarrow E$	
4.	$BC \rightarrow B$	(Réflexivité)
5.	$BC \rightarrow E$	(Transitivité : 2, 3)
6.	$BC \rightarrow BDE$	(Union : 2, 4, 5)
7.	$AC \rightarrow BDE$	(Pseudo-transitivité : 1, 6)

La DF $AC \rightarrow BDE$ peut être inférée des autres trois dans F. Donc, elle est redondante.

Obtention de la couverture minimale

Le calcul de la couverture minimale F_{min} d'un ensemble de DF F se fait comme suit :

1. Initialiser F_{min} à F.
2. Réduire à droite les DF :
   Substituer $X \rightarrow A_1, A_2, \ldots, A_n$ par $X \rightarrow A_1$, $X \rightarrow A_2$, …, $X \rightarrow A_n$.
3. Éliminer les DF triviales :
   Supprimer toutes les DF $X \rightarrow Y$ tel que $Y \subseteq X$.
   Substituer toutes les DF $X \rightarrow Y$ par $W \rightarrow Y$, tel que $W \subseteq X^+$.
4. Réduire à gauche les DF :
   Pour toute $X \rightarrow A$, s'il existe $Y \subseteq X$ tel que $Y \rightarrow A$ dans F_{min} alors supprimer $X \rightarrow A$ de F_{min}.
5. Éliminer les DF redondantes :
   Pour toute $X \rightarrow A$ qui peut être inférée de $F_{\text{min}} - \{X \rightarrow A\}$, supprimer $X \rightarrow A$.

Obtention de la couverture minimale

Exemple :

Soit l'ensemble de dépendances fonctionnelles :

$$F = \{A \rightarrow B, BC \rightarrow D, AC \rightarrow BDE, D \rightarrow E\}$$

Calculer une couverture minimale pour F en utilisant l'algorithme.

Résponse :

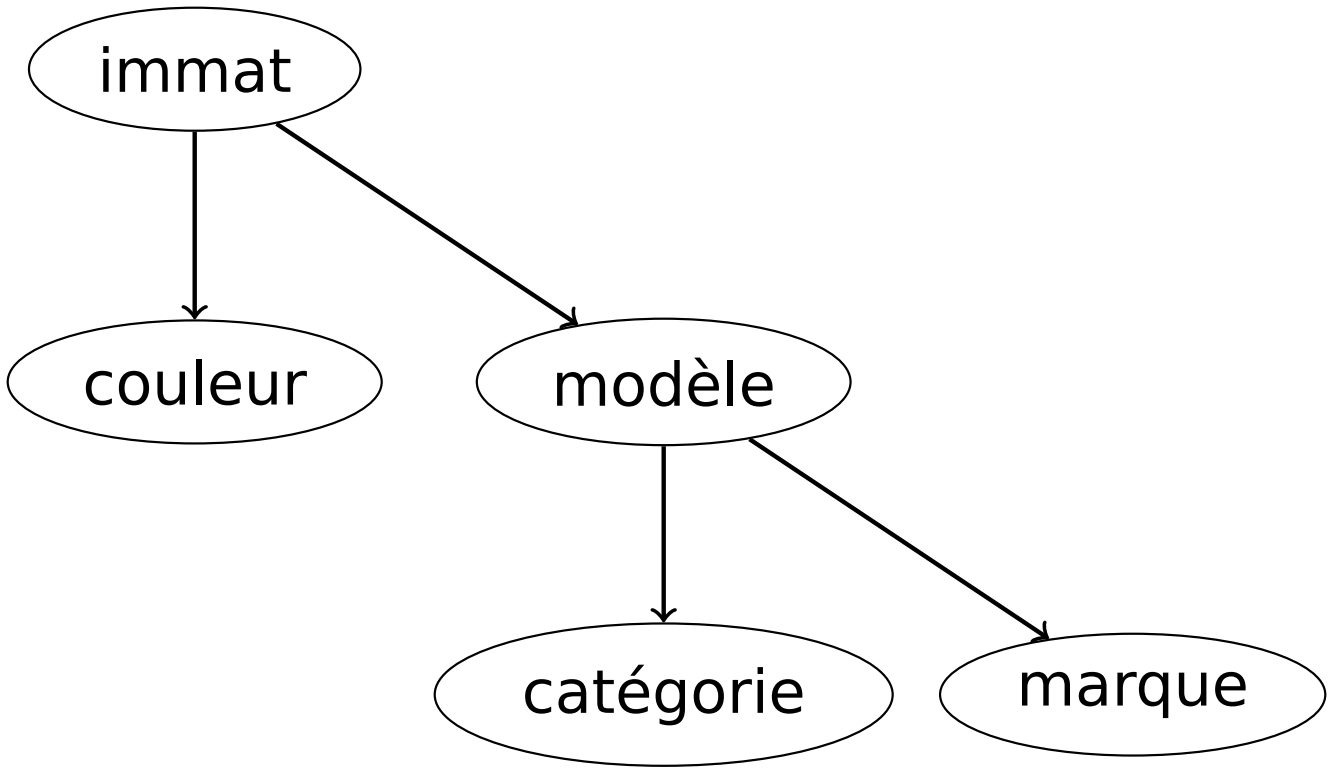
1. $F_{\text{min}} = \{A \rightarrow B, BC \rightarrow D, AC \rightarrow BDE, D \rightarrow E)\}$
2. $F_{\text{min}} = \{A \rightarrow B, BC \rightarrow D, AC \rightarrow B, AC \rightarrow D, AC \rightarrow E, D \rightarrow E\}$
3. $F_{\text{min}} = \{A \rightarrow B, BC \rightarrow D, AC \rightarrow D, AC \rightarrow E, D \rightarrow E\}$
4. $F_{\text{min}} = \{A \rightarrow B, BC \rightarrow D, AC \rightarrow D, D \rightarrow E\}$
5. $F_{\text{min}} = \{A \rightarrow B, BC \rightarrow D, D \rightarrow E\}$

Graphe des dépendances

Un ensemble de DF peut être représenté sous forme graphique.

Exemple :

`Voiture(immat, marque, modèle, catégorie, couleur)`



Section 33

Dépendances multivaluées

Dépendances multivaluées

Les dépendances multivaluées (DM) sont une généralisation des DF.

Les DM sont moins fréquentes que les DF, mais peuvent tout de même apparaître en pratique.

Définition :
Soit une relation $R(U)$, et X, Y, Z trois sous-ensembles de U.

Soit x une valeur de X, y une valeur de Y et z une valeur de Z.

X **multidétermine** Y (noté $X \twoheadrightarrow Y$) si et seulement si, $(x, y, z), (x, y', z') \in R$ implique $(x, y', z), (x, y, z') \in R$.

Dépendances multivaluées

Exemple :

Soit le schéma :

$$\texttt{Acheter(num\_classe, num\_élève, code\_livre).}$$

En imposant à tous les élèves de la même classe d'acheter les mêmes livres, nous avons :

$$\texttt{num\_classe} \twoheadrightarrow \texttt{num\_élève}$$

Par exemple :

- tous les élèves de la classe numéro 1 doivent acheter les livres 10X, 11Z et 12Y.
- tous les élèves de la classe numéro 2 doivent acheter les livres 20T et 21H.

Dépendances multivaluées

Nous avons :

num_classe	num_ élève	code_livre
1	100	10X
1	100	11Z
1	100	12Y
1	101	10X
1	101	11Z
1	101	12Y
2	160	20T
2	160	21H
2	161	20T
2	161	21H

Notez que l'insertion du tuple `(1, 102, 10X)` rend la table incohérente.

La suppression d'une ligne rend la table incohérente aussi.

Même la mise à jour d'une ligne de la table peut la rendre incohérente.

Dépendances multivaluées

Exemple :

Soit le schéma :

`Posséder(num_équipe, num_ouvrier, code_matériel).`

En imposant à tous les ouvrier d'une même équipe de posséder le même matériel, nous avons :

$$\texttt{num\_équipe} \twoheadrightarrow \texttt{num\_ouvrier}$$

Par exemple : tous les ouvrier de l'équipe numéro 1 doivent posséder les matériaux 12T, 20H et 15Z.

Dépendances multivaluées

Nous avons :

num_équipe	num_ouvrier	code_matériel
1	40	12T
1	40	20H
1	40	15Z
1	41	12T
1	41	20H
1	41	15Z
2	60	12T

Encore une fois, notez que l'insertion, suppression ou mise à jour d'une ligne peut rendre la table incohérente.

Dépendances multivaluées

Théorème :

Soit le schéma $R(U)$ et X, Y deux sous-ensembles de U.

Soit $S = \pi_{X,Y}(R) \bowtie \pi_{X,U-(XY)}(R)$.

Nous avons que, si $X \twoheadrightarrow Y$, alors $R = S$.

Preuve :

En effet, parce que, $(x, y, z), (x, y', z') \in R$ implique $(x, y', z), (x, y, z') \in R$ (par la définition de $X \twoheadrightarrow Y$).

Nous avons aussi :

- $(x, y), (x, y') \in \pi_{X,Y}(R)$; et
- $(x, z), (x, z') \in \pi_{X,U-(XY)}(R)$.

Donc, nous avons :

- $(x, y, z), (x, y, z'), (x, y', z), (x, y', z') \in S$.

Comme ceci est vrai pour tous les tuples (x, y, z) et (x, y', z'), alors $R = S$. □

Les DF sont des cas particuliers des DM.

Si $X \rightarrow Y$ alors nous avons :

$$\{(x, y, z), (x, y', z') \in R \text{ implique } y = y'\}$$

Donc, nous avons aussi :

$$\{(x, y, z), (x, y', z') \in R \text{ implique } (x, y, z'), (x, y', z) \in R\}$$

Donc, nous avons $X \twoheadrightarrow Y$.

Autrement dit :

$$X \rightarrow Y \models X \twoheadrightarrow Y$$

Règles d'inférence

Soit le schéma $(R(U), D)$:

- **Complémentation :** $X \twoheadrightarrow Y \models X \twoheadrightarrow U - XY$
- **Augmentation :** si $S \subseteq T$ alors $X \twoheadrightarrow Y \models XT \twoheadrightarrow YS$
- **Transitivité :** $X \twoheadrightarrow Y, Y \twoheadrightarrow Z \models X \twoheadrightarrow Z - Y$
- $X \rightarrow Y \models X \twoheadrightarrow Y$
- si $Z \subseteq Y$ et $Y \cap W = \emptyset$ alors $X \twoheadrightarrow Y, W \rightarrow Z \models X \rightarrow Z$

Ces règles plus les axiomes d'Armstrong forment un système valide et complet pour les DM.

Dépendance multivaluée élémentaire

Définition :

Une **dépendance multivaluée élémentaire (DME)** est une DM $X \twoheadrightarrow Y$ où :

- $Y \neq \emptyset$
- $X \cap Y = \emptyset$
- Il n'existe pas de DM $X' \twoheadrightarrow Y'$ tel que $X' \subset X$ et $Y' \subset Y$.

Définition :

Soit le schéma de relation $(R(U)$. Une **dépendance multivaluée triviale** est une DM $X \twoheadrightarrow Y$ tel que :

- $Y \subseteq X$; ou
- $XY = U$.

Cours 10
Normalisation

Sommaire

34. Introduction

35. Formes normales

36. Décomposition

Section 34

Introduction

Section 35

Formes normales

Formes normales

Les formes normales ont été introduites afin de pouvoir évaluer la qualité d'un schéma de relation.

Il y a six formes normales :

- Première forme normale (1NF)
- Deuxième forme normale (2NF)
- Troisième forme normale (3NF)
- Forme normale de Boyce Codd (BCNF)
- Quatrième forme normale (4NF)
- Cinquième forme normale (5NF)

Première forme normale (1NF)

Définition :
Un schéma de relation est en **première forme normale (1NF)** si chacun des ses attributs est de type de donnée élémentaire.

Cela veut dire que la valeur prise par un attribut dans un tuple est à la fois unique (pas d'ensemble) et atomique (non décomposable).

En effet, cette forme normale exige seulement une certaine présentation pour les schémas.

Elle ne traite pas du problème de la redondance.

Première forme normale (1NF)

Exemple :

`Livre(cote, titre, auteurs)`

Le schéma `Livre` n'est pas en 1NF car l'attribut auteur n'est pas de type élémentaire (un livre peut être écrit par plusieurs auteurs).

Solution 1 : Créer plusieurs d'attributs :

`Livre(cote, titre, auteur_1, auteur_2, auteur_3)`

Solution 2 : Décomposer en plusieurs schémas :

`Livre(cote, titre)`
`Auteur(id_auteur, nom)`
`Écrire(cote*, id_auteur*)`

Remarque : tout schéma doit d'abord être transformé en 1NF avant de procéder à une quelconque opération de décomposition. Dans la suite, un schéma est toujours supposé être en 1NF.

Deuxième forme normale (2NF)

Définition :

Un schéma de relation est en **deuxième forme normale (2NF)** si tout attribut non clé ne peut être déterminé par une partie stricte d'une clé candidate.

Il revient au même de dire que pour tout attribut non clé A et toute clé candidate X, la dépendance fonctionnelle $X \rightarrow A$ est élémentaire.

En d'autres mots, nous pouvons dire aussi que il n'y a pas de dépendance d'une partie d'une clé candidate.

Deuxième forme normale (2NF)

Exemple :

vol	id_comp	compagnie	avion	capacité	départ	arriv
0022	AF	Air France	B767	375	CDC	JFK
0157	AZ	Alitalia	MD80	288	BRU	CIA
0022	LH	Lufthansa	A319	280	FRF	YHU
7887	AF	Air France	A320	180	LIL	AJA

$\texttt{vol} \rightarrow \texttt{avion, départ, arrivée}$
$\texttt{avion} \rightarrow \texttt{capacité}$
$\texttt{id\_comp} \rightarrow \texttt{compagnie}$
$\texttt{vol} \not\rightarrow \texttt{compagnie}$

Ce schéma n'est pas en 2NF, parce que compagnie ne dépend pas de la totalité de la clé candidate {vol, id_comp}.

Solution :

Compagnie(<u>id_comp</u>, compagnie)
Vol(<u>vol, id_comp*</u>, avion, capacité, départ,

Troisième forme normale (3NF)

Définition :
Une DF $X \rightarrow Y$ est **triviale** si $Y \subseteq X$.

Définition :
Un schéma de relation est en **troisième forme normale (3NF)** si tout attribut non clé ne peut être déterminé que par les clés candidates (en écartant les DF triviales).

Autrement dit, tout attribut n'appartenant pas à une clé ne dépend pas d'un autre attribut non clé.

Ou encore, pas de dépendence transitive.

Un schéma en 3NF est un schéma en 2NF auquel on a écarté en plus les DF entre attributs non clés.

La 3NF implique la 2NF.

La 3NF est donc de meilleure qualité.

Troisième forme normale (3NF)

Exemple :

Le schéma dans la solution de l'exemple précédent :

`Vol(vol, id_comp*, avion, capacité, départ, arrivé)`

n'est pas en 3NF, parce que :

`avion → capacité`

et `avion` n'est pas clé candidate.

La relation `Vol` est soumise à un problème de redondance :

vol	id_comp	avion	capacité	départ	arrivée
0022	AF	**B767**	**375**	CDC	JFK
0033	AF	**B767**	**375**	JFK	CDC

Solution :

`Companie(id_comp, compagnie)`
`Avion(id_avion, capacité)`
`Vol(id_comp*, vol, id_avion*, départ, arrivée)`

Forme normale de Boyce Codd (BCNF)

Un schéma en 3NF peut encore présenter des redondances.

Le schéma :

`Livraison(num_labo, num_produit, num_fournisseur)`
`num_labo, num_produit → num_fournisseur`
`num_fournisseur → num_produit`

est en 3NF. Pourtant, il y en a des redondances :

num_lab	num_produit	num_fournisseur
1	20	100
2	20	100
3	20	100
1	21	101
2	21	101

Forme normale de Boyce Codd (BCNF)

Définition :

Un schéma de relation est en **forme normale de Boyce Codd (BCNF)** si tout attribut ne peut être déterminé que par les clés candidates (en écartant les DF triviales).

Autrement dit, il n'y a pas de dépendance d'attribut non clé.

La BCNF implique la 3NF.

La BCNF est donc de meilleure qualité.

Forme normale de Boyce Codd (BCNF)

Solution de l'exemple :

Fournisseur_Produit(num_fournisseur, num_produit)
Labo_Fournisseur(num_labo, num_fournisseeur)

La table précédente devient :

Fournisseur_Produit:

num_fournisseur	num_produit
100	20
101	21

Labo_Fournisseur:

num_labo	num_fournisseur
1	100
2	100
3	100
1	101
2	101

Quatrième forme normale (4NF)

En présence des DM, un shcéma de relation en BCNF peut encore être soumis au problème de la redondance de données.

Exemple :

`Posséder(num_équipe, num_ouvrier, code_matériel)`

`num_équipe ↠ num_ouvrier`

Ce schéma est en BCNF puisqu'il n'existe aucune DF non triviale.

Pourtant, la table comporte beaucoup de redondances :

num_équipe	num_ouvrier	code_matériel
1	40	12T
1	40	20H
1	40	15Z
1	41	12T
1	41	20H
1	41	15Z
2	60	12T

Quatrième forme normale (4NF)

Définition :
Un schéma de relation est en **quatrième forme normale (4NF)** si et seulement si toute DM $X \twoheadrightarrow Y$ non triviale contient une clé candidate.

Autrement dit, il n'y a pas de DME d'attribut non clé.

Comme nous avons que DME implique DF, alors 4NF implique BCNF.

Quatrième forme normale (4NF)

Solution de l'exemple :

Équipe(num_équipe, num_ouvrier)
Matériel(num_équipe, code_matériel)

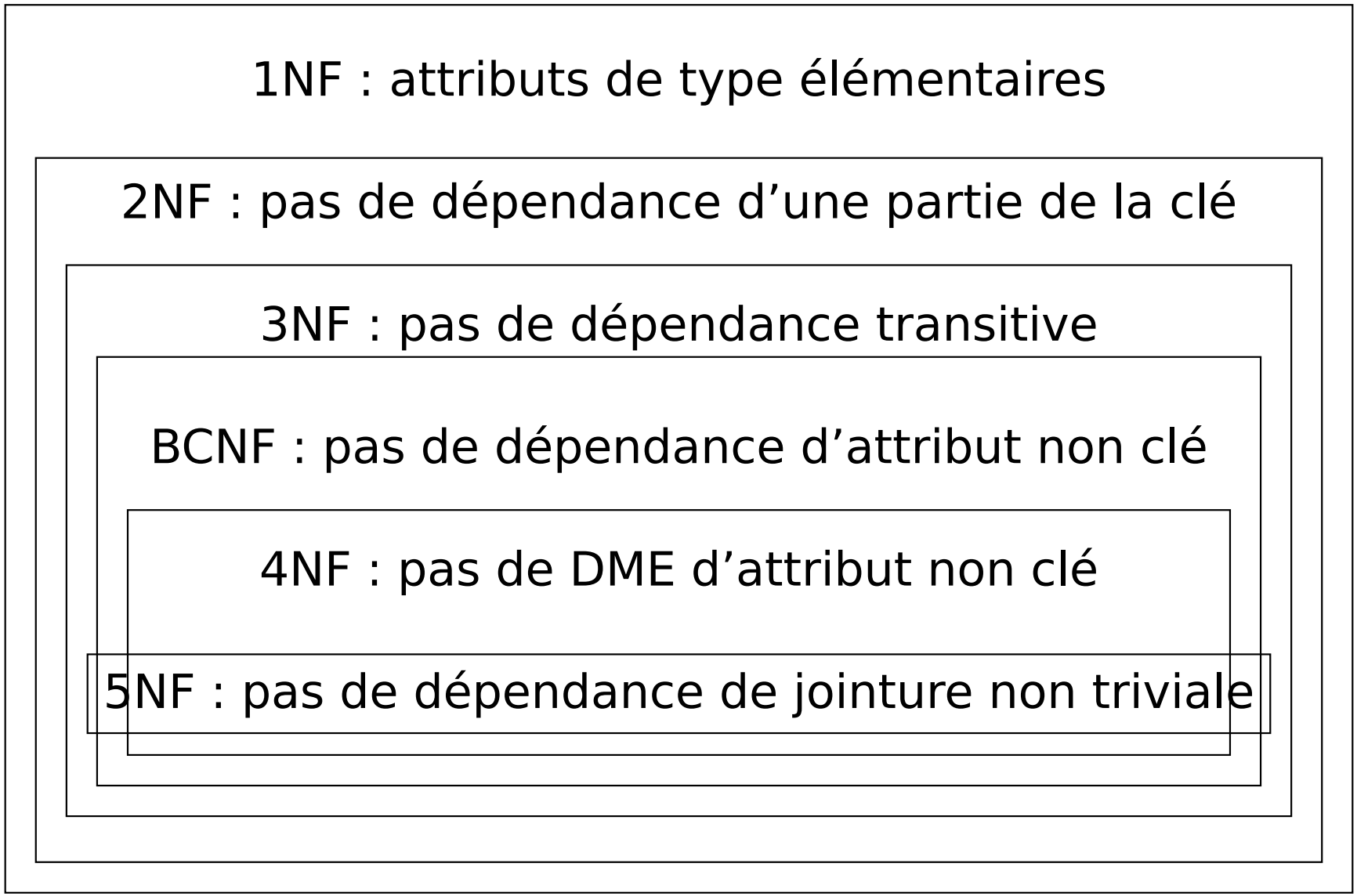
Équipe:

num_équipe	num_ouvrier
1	40
1	41
2	60

Matériel:

num_équipe	code_matériel
1	12T
1	20H
1	15Z
2	12T

Formes normales



Section 36

Décomposition

Décomposition

Principe de l'approche par décomposition :

- Démarrer avec une relation universelle (contenant touts les attributs du domaine).
- Parvenir à l'aide de raffinements successifs à un ensemble de relations qui souffrent de moins en moins des problèmes d'incohérences, redondances et autres inconvénients.
- Ces transformations s'appuient sur des opérations basées sur les propriétés des DF.

Décomposition

Les DF d'une relation doivent être vérifiées à chaque étape.

- Elles traduisent des contraintes sur les données.

Pour normaliser une relation, nous la décomposons :

- Il est donc nécessaire que ce processus de décomposition préserve les DF des relations décomposées.
- Il faut pouvoir retrouver ces DF dans les relations créées par la décomposition.

Préservation des dépendances fonctionnelles

Comment décomposer une relation R pour augmenter son niveau de normalisation en préservant ses DF ?

Définition :
La décomposition d'un schéma R en sous schémas $R_1, R_2, \ldots, R_n$ **préserve les DF** de R si la fermeture des DF de R est équivalente à la fermeture de l'union des DF des schémas résultants $R_1, R_2, \ldots, R_n$. En d'autres termes :

$$F^+ = (F_1 \cup F_2 \cup \cdots \cup F_n)^+$$

où chaque F_i est l'ensemble des DF de R_i.

Préservation des dépendances fonctionnelles

Exemple :

Voiture(immat, marque, modèle, catégorie)

immat → marque, modèle, catégorie
modèle → marque, catégorie

Décomposition :

Voiture(immat, modèle)
Modèle_voiture(modèle, marque, catégorie)

immat → modèle
modèle → marque, catégorie

Notez que les deux ensembles de DF sont équivalents.

Décomposition sans perte d'information

Nous pouvons décomposer un schéma $R(A, B, C, D)$ ayant la DF $B \rightarrow C$ en deux relations :

$$R_1(A, B, D)$$
$$R_2(B, C)$$

Décomposition sans perte d'information

Exemple :

id_comp	vol	avion	constructeur	départ	arrivée
AF	0022	B767	Boeing	CDC	JFK
AZ	0157	MD80	McDonnell Douglas	BRU	CIA
LH	1004	A319	Airbus	FRF	YHU
AF	7887	A320	Airbus	LIL	AJA
LH	1019	B767	Boeing	FRF	JOH

Décomposer la relation selon la DF `avion` → `constructeur`.

Décomposition sans perte d'information

Solution :

id_comp	vol	compagnie	avion	départ	arrivée
AF	0022	Air France	B767	CDC	JFK
AZ	0157	Alitalia	MD80	BRU	CIA
LH	1004	Lufthansa	A319	FRF	YHU
AF	7887	Air France	A320	LIL	AJA
LH	1019	Lufthansa	B767	FRF	JOH

avion	constructeur
B767	Boeing
MD80	McDonnell Douglas
A319	Airbus

Décomposition sans perte d'information

Définition :

Décomposer une relation $R(A_1, \ldots, A_n)$, c'est la remplacer par un ensemble de relations $R_1, R_2, \ldots, R_m$ où :

- les R_i sont obtenues par projections sur R et
- la jointure naturelle de toutes les R_i reconstitue la relation initiale R.

Théorème :

Toute relation admet au moins une décomposition en 3NF tel que :

- La décomposition préserve les DF.
- La décomposition est sans perte de données.

Algorithme de décomposition en 3NF

Entrée : un schéma $R(A_1, \ldots, A_m)$ et l'ensemble F de DF associés à R.
Sortie : un ensemble de schémas $\{R_1, \ldots, R_n\}$ avec chaque R_i en 3NF.

1. Calcule une couverture minimale C de F.
2. Ajoute à la clé primaire l'ensemble des attributs isolés dans une même relation.
3. Tant que $C \neq \emptyset$:
   - 3.1 Cherche le plus grand ensemble X d'attributs qui détermine d'autres attributs $A_1, \ldots, A_k$.
   - 3.2 Créé une nouvelle relation $R_i(\underline{X}, A_1, \ldots, A_k)$.
   - 3.3 Supprime les DF $X \rightarrow A_1, \ldots, X \rightarrow A_k$ de C.
   - 3.4 Supprime les attributs isolés (non source ou cible de DF) de C.

Algorithme de décomposition en 3NF

Exemple :

Décomposer en 3NF la relation :

voiture(<u>immat</u>, modèle, puissance, marque, couleur)

$$F = \{\ \texttt{immat} \rightarrow \texttt{modèle},\ \texttt{immat} \rightarrow \texttt{couleur},\ \texttt{modèle} \rightarrow \texttt{puissance},\ \texttt{modèle} \rightarrow \texttt{marque}\ \}$$

Étape 1 : F est déjà une CM. Donc, $C := F$.
Étape 2 : Pas d'attribut isolé.
Étapes 3.1 et 3.2 : création de R_1(<u>immat</u>, modèle, couleur).
Étape 3.3 : $C = \{\texttt{modèle} \rightarrow \texttt{puissance}, \texttt{modèle} \rightarrow \texttt{marque}\}$.
Étapes 3.1 et 3.2 : création de R_2(<u>modèle</u>, puissance, marque).
Étape 3.3 : $C = \emptyset$.

Algorithme de décomposition en BCNF

Entrée : un schéma de relation $(R(U), F)$.

Sortie : B décomposition en BCNF sans perte de $R(U)$.

1. $B = \{(R(U), F)\}$
2. Si $(S(W), F[W])$ est un schéma dans B non en BCNF alors :
   - 2.1 Choisir une DF $X \rightarrow Y$ non triviale de $F[W]$ où X n'est pas une super-clé de $S(W)$.
   - 2.2 Remplacer dans B $(S(W), F[W])$ par les deux schémas : $(S_1(W_1), F_1[W_1])$ et $(S_2(W_2), F_2[W_2])$ où $W_1 = XY$ et $W_2 = X(W - XY)$.
3. Répéter 2 tant qu'il existe un schéma dans B non en BCNF.

Note : $X \rightarrow Y \in F[W]$ signifie $X \rightarrow Y \in F^+$ et $XY \subseteq W$.

Remarques :

- Il peut y exister plus d'une solution.
- La décomposition obtenue est sans perte d'information, mais elle ne préserver pas forcement les dépendances.

Algorithme de décomposition en BCNF

Exemple :

Bibliothèque(num_étudiant, nom, prénom,
département, code_livre, titre, auteur,
date_retour, num_éditeur, désignation, ville)

num_etudiant → nom
num_étudiant → département
code_livre → auteur
num_éditeur → désignation
num_étudiant, code_livre → date_retour
num_étudiant → pré
code_livre → titre
code_livre → num_é
num_éditeur → ville

La seule clé candidate est donc
{num_étudiant, code_livre}.

Algorithme de décomposition en BCNF

Solution :

Étudiant(num_étudiant, nom, prénom, département)
Éditeur(num_éditeur, désignation, ville)
Livre(code_livre, titre, auteur, num_éditeur*)
Emprunter(num_étudiant*, code_livre*, date_retour)

Algorithme pour la 4NF

Il s’agit du même algorithme que pour la décomposition en BCNF, où on remplace les DF par DM.

Dans ce cas, les DF sont considérés implicitement car toute DF est une DM.

Encore une fois, cet algorithme préserve les information mais pas les dépendances.

Normalisation

Deux principales approches :

1. Modélisation, decomposition et vérification :
   - Lister les DF du domaine étudié.
   - Élaborer un modèle EA.
   - Traduire en relationnel.
   - Vérifier les relations obtenues et normaliser si besoin.
2. Décomposition algorithmique :
   - Recenser les informations (attributs) do domaine étudié (construire une reltaion universelle).
   - Lister les DF existantes.
   - Utiliser l'algorithme de décomposition.

Bilan

Plus le niveau de normalisation est élevé, meilleur est le schéma en terme de cohérence, redondance et anomalies.
Mais :

1. Les schémas ont tendance à avoir beaucoup de tables.
2. Les schémas sont difficiles à interroger (car beaucoup de jointures).
3. Les interrogations ont tendance à être lentes.