

Rappels sur le modèle relationnel

Polytech Marseille, SN, 4A

Simon Vilmin

`simon.vilmin@univ-amu.fr`

2023 - 2024



Plan

Structure

- Base, relations, tuples et attributs
- Contraintes

Langages de requêtes

- Généralités
- L'algèbre relationnelle vite fait
- Un peu de SQL

Modélisation EA

- Modèle EA
- traduction vers un schéma relationnel

Exercice

Structure

Structure

Base, relations, tuples et attributs

Contraintes

Langages de requêtes

Modélisation EA

Exercice

i Remarque : (idée générale)

- *le modèle relationnel* est basé sur la *théorie des ensembles*
- *très structuré* et *contraint beaucoup* ce qu'on y stocke
- réduit beaucoup la *redondance* et renforce l'*intégrité* des données

Il est très structuré et contraint beaucoup ce qu'on y stocke. De ce fait, il réduit beaucoup la redondance et renforce l'intégrité des données.

La base de données BonTemps

schéma de la relation

Café	nomcafé	adresse	téléphone	avis
	les délices de l'abstraction	1 bd. de l'Impasse, 13007, Marseille	04 12 12 12 12	5/5
	Rakwé	18 rue du Moka, 13190, Allauch	08 36 65 65 65	4/5
	Le Maté-matique	3bis av. du rire, 13010, Marseille	04 36 30 36 30	2/5

relation

Personne	nom	prénom	adresse
	Nette	Marie	3 rue Lumière, 54500, langres
	Fenouil	Emile	5 place du Potiron, 63036, Ceyssat
	Taie	Clara	17 bd. du Flan, 17001, La Rochelle

tuple

Aime	nom	café	attribut
	Nette	Le Maté-matique	
	Fenouil	Rakwé	
	Taie	les délices de l'abstraction	
	Nette	Rakwé	



Rappel : le modèle relationnel est avant tout basé sur la *théorie des ensembles*. En pratique, les SGBDR offrent plus de libertés et de fonctionnalités.

Modélisation des données possibles dans une base de données

- *domaine* ensemble infini dénombrable de *valeurs constantes*. Par ex : entiers, booléens, réels, chaînes de caractères, ...
- \mathcal{D} l'ensemble des domaines possibles

Modélisation des attributs

- un *nom* choisi dans un univers de noms \mathcal{U}
- un *domaine*, soit l'ensemble des valeurs que peut prendre un attribut
- attribut A avec domaine $dom(A)$

Tuples

- Un *tuple* rassemble des valeurs de plusieurs attributs :
 - Soit $U \subseteq \mathcal{U}$. Un *tuple* sur U est une fonction $t: U \rightarrow \mathcal{D}$ satisfaisant $t(A) \in \text{dom}(A)$ pour tout $A \in U$.
 - Soit $X \subseteq U$. $t[X]$ est la *restriction* de t à X , i.e. le tuple u sur X tel que $u(A) = t(A)$ pour tout $A \in X$.



Astuce : cette modélisation compliquée pour dire que :

- un tuple t correspond à une ligne dans une des tables
- $t[X]$ veut dire qu'on regarde les colonnes X

Schémas de relations et de base de données

i Note : pour pouvoir créer des relations ou des bases de données, il faut prédéfinir un *schéma* que les données vont devoir respecter. Ca rajoute au côté *très structuré* des BD relationnelles.

Les *schémas* sont des modèles de construction de relations et de base :

- \mathcal{R} : ensemble de *noms de relations*
- chaque $R \in \mathcal{R}$ est associé à un ensemble fini d'attributs de \mathcal{U}
- R aussi appelé *schéma de relation*
- *schéma de base de données* \mathbf{R} : ensemble fini de schémas de relations sur \mathcal{U}

! Important : les schémas sont *indépendants* des données !

Relations et bases de données

Maintenant que l'on a des schémas, on peut définir relations et bases de données :

- étant donné un schéma de relation R , une *relation* r sur R est un ensemble fini de tuples sur R
- étant donné un schéma de bases de données $\mathbf{R} = \{R_1, R_2, \dots, R_n\}$ une *base de données* \mathbf{d} sur \mathbf{R} est un ensemble $\{r_1, \dots, r_n\}$ de relations tel que pour tout $1 \leq i \leq n$, r_i est une relation sur R_i

Exemple de BonTemps

Café	nomcafé	adresse	téléphone	avis
	les délices de l'abstraction	1 bd. de l'Impasse, 13007, Marseille	04 12 12 12 12	5/5
	Rakwé	18 rue du Moka, 13190, Allauch	08 36 65 65 65	4/5
	Le Maté-matique	3bis av. du rire, 13010, Marseille	04 36 30 36 30	2/5

Personne	nom	prénom	adresse
	Nette	Marie	3 rue Lumière, 54500, langres
	Fenouil	Emile	5 place du Potiron, 63036, Ceyssat
	Taie	Clara	17 bd. du Flan, 17001, La Rochelle

Aime	nom	café
	Nette	Le Maté-matique
	Fenouil	Rakwé
	Taie	les délices de l'abstraction
	Nette	Rakwé

Exemple de **BonTemps** : schémas

- Disons que *cafe* est le schéma de la relation *Café* :
 - les attributs du schéma *cafe* sont *nomcafé*, *adresse*, *téléphone*, et *avis*
 - le schéma se note parfois *cafe*(*nomcafé*, *adresse*, *téléphone*, *avis*)
- De même :
 - le schéma de *Personne* est *personne*(*nom*, *prénom*, *adresse*)
 - le schéma de *Aime* est *aime*(*nom*, *nomcafé*)
- Le schéma de la base de données **BonTemps** est **BonTemps** :
 - On a $\text{BonTemps} = \{\text{cafe}, \text{personne}, \text{aime}\}$
 - Ce qu'on note également **BonTemps**(*cafe*, *personne*, *aime*)


Éléments stockés

Dans une BD relationnelle, les éléments stockés dans les tuples doivent être « atomiques »

Un schéma de relation R est en *première forme normale (1FN)* si pour tout $A \in R$, $dom(A)$ ne contient que des valeurs constantes et *atomiques*.
Un schéma de base de données \mathbf{R} est en 1FN si tous ses schémas le sont.

i Remarque : autre point d'accroche avec les BD noSQL ! Dans une BD noSQL, on peut stocker des éléments composites (comme des documents)


Contraintes

 **Définition** : les *contraintes (d'intégrité)* sont des déclarations logiques qui visent à renforcer l'*intégrité et la non-redondance* de l'information dans une base de données relationnelle.

Contraintes communes

- les *clés*
 - contraintes *intra-relation*
 - sous-ensembles d'attributs qui permettent d'identifier de façon unique un tuple dans une relation
 - ex : « le numéro de sécurité sociale identifie un patient »
- les *clés étrangères*
 - contraintes *inter-relations*
 - séquences d'attributs qui permettent de référencer des tuples entre relations
 - ex : « un produit ne peut vendu que s'il existe dans le catalogue »

Contraintes : clés

 **Définition** : Soit R un schéma de relation, r une relation sur R et $K \subseteq R$. K est une *clé* de r si pour tout tuples $t_1, t_2 \in r$, $t_1[K] \neq t_2[K]$


Dans **BonTemps**

- *nomcafé* est une clé de cafe : il n'y a pas deux cafés avec le même nom
- *nom* est une clé de Personne
- *nom, nomcafé* est une clé de Aime

 **Remarque** : clé = cas particulier de *dépendance fonctionnelle* :

- expression $X \rightarrow A$, $X \cup \{A\} \subseteq R$
- vraie dans une relation r si pour tout $t_1, t_2 \in r$, $t_1[X] = t_2[X]$ implique $t_1[A] = t_2[A]$

Contraintes : clés étrangères

 **Définition** : soit \mathbf{R} un schéma de base de données et $\{R_1, R_2\} \subseteq \mathbf{R}$. Soit K_1, K_2 deux *séquences d'attributs distincts* de R_1 et R_2 (respectivement) telles que :

- K_1 et K_2 sont de même taille
- K_2 est incluse dans une clé de R_2

Soit \mathbf{d} une base de données sur \mathbf{R} . $R_1[K_1] \subseteq R_2[K_2]$ est une *clé étrangère* de \mathbf{d} si pour tout tuple t_1 de r_1 (la relation sur R_1), il doit exister un tuple t_2 de r_2 (sur R_2) tel que $t_1[K_1] = t_2[K_2]$.

Dans la relation *Aime* de **BonTemps**

- $\text{aime}[\text{nom}] \subseteq \text{personne}[\text{nom}]$: un nom de personne ne peut apparaître que s'il apparaît dans *Personne*
- $\text{aime}[\text{nomcafé}] \subseteq \text{café}[\text{nomcafé}]$: un nom de café ne peut apparaître que s'il apparaît dans *Café*

Résumé



Rappel : le modèle relationnel est très fortement structuré

- schéma de données *prédéfini*
- stockage de valeurs *atomiques*
- contraintes pour garantir l'*intégrité des données* et limiter la *redondance de l'information*



Note : les modèles noSQL cherchent généralement à *relâcher ces contraintes* pour faciliter la mise en place de systèmes distribués.

Accéder aux données avec SQL

Structure

Langages de requêtes

Généralités

L'algèbre relationnelle vite fait

Un peu de SQL

Modélisation EA

Exercice

i Remarque : (idée générale)

- SQL = langage de requête *requête déclaratif*
- dérivé du *calcul relationnel* et de l'*algèbre relationnelle*

Langages de requêtes

Langages *déclaratifs* : on *décrit* le résultat recherché sans spécifier *comment* l'obtenir.

Approche algébrique : *algèbre relationnelle*

- permet de représenter le *plan d'exécution d'une requête*
- *sémantique opérationnelle*

Approche logique :

- *calcul relationnel*
 - plus proche du langage naturel : on ne spécifie pas d'ordre
 - *sémantique déclarative*
- **Datalog** : langage déclaratif à base de règles, capacités d'inférence




Remarque : SQL émerge de ces trois approches

Algèbre relationnelle

Collection d'*opérateurs algébriques* s'appliquant sur des *relations*

- basée sur la théorie des ensembles (toujours)
- les opérateurs peuvent être *composés*
- le résultat d'une opération est une relation

Projection

 **Définition :** (*projection*) Soit R un schéma de relation, r une relation sur R et $X \subseteq R$. La *projection* de r sur X , notée $\pi_X(r)$ est définie par

$$\pi_X(r) = \{t[Y] \mid t \in r\}$$



Astuce : projection = sélection de *colonnes*

r_1	A	B	C		$\pi_{AB}(r_1)$	A	B
	0	1	1	$\xrightarrow{\pi_{AB}}$		0	1
	1	2	2			1	2
	0	0	0			0	0
	0	1	2				



Attention : il faut enlever les *doublons* pour avoir un *ensemble*.

Sélection : définir une formule

i Note : on voudrait *sélectionner* des tuples (des *lignes*) d'une relation r sur R répondant à une *propriété*, soit une *formule* F

? Question : Qu'est-ce qu'une (bonne) formule ?


Construction *inductive* des formules de sélections

- formule *simple* : expression de la forme $A = a$ où $A = B$ avec $A, B \in R$ et $a \in \text{dom}(A)$
- une formule de *sélection* est :
 - soit une formule simple,
 - soit une expression de la forme $(F_1 \vee F_2)$, $(F_1 \wedge F_2)$, $\neg(F_1)$, ou (F_1) , avec F_1, F_2 des formules de sélection

Sélection : satisfaire une formule

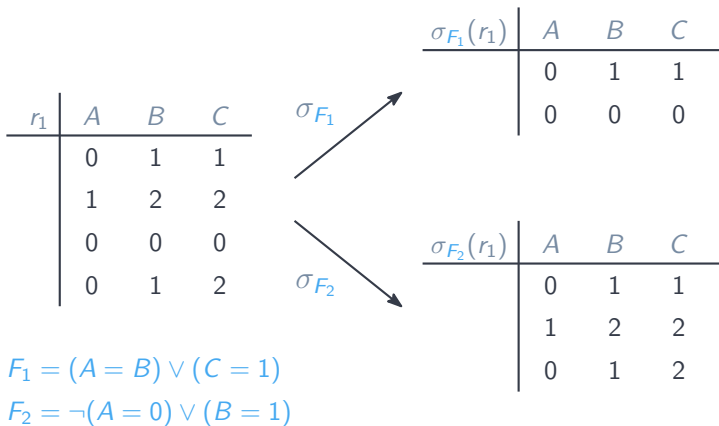
Définition *inductive*. Un tuple t satisfait F , noté $t \models F$, est donné par :

- $t \models A = a$ si $t[A] = a$
- $t \models A = B$ si $t[A] = t[B]$
- $t \models (F_1 \wedge F_2)$ si $t \models F_1$ *et* $t \models F_2$
- $t \models (F_1 \vee F_2)$ si $t \models F_1$ *ou* $t \models F_2$
- $t \models \neg(F)$ si $t \not\models F$
- $t \models (F)$ si $t \models F$

 **Définition :** Soit r une relation sur R et F une formule de sélection sur R . La *sélection* des tuples de r par rapport à F , notée $\sigma_F(r)$, est définie par :

$$\sigma_F(r) = \{t \in r \mid t \models F\}$$


Sélection : exemple



Opérations ensemblistes

 **Définition :** (*union*, *intersection*, *différence*) soient r_1 , r_2 deux relations sur R


- l'*union* de r_1 et r_2 est $r_1 \cup r_2 = \{t \mid t \in r_1 \text{ ou } t \in r_2\}$
- l'*intersection* de r_1 et r_2 est $r_1 \cap r_2 = \{t \mid t \in r_1 \text{ et } t \in r_2\}$
- la *différence* de r_1 et r_2 est $r_1 - r_2 = \{t \mid t \in r_1 \text{ et } t \notin r_2\}$

 **Exercice :** donner $r_1 \cup r_2$, $r_1 \cap r_2$ et $r_1 - r_2$

r_1	A	B	C
	1	2	3
	1	1	1
	1	2	2

r_2	A	B	C
	2	2	2
	1	1	1

Produit cartésien

 **Définition :** (*produit cartésien*) soient r_1 et r_2 deux relations sur R_1 et R_2 respectivement, avec R_1 et R_2 disjoints. Le *produit cartésien* $r_1 \times r_2$ de r_1 et r_2 est la relation sur $R = R_1 \cup R_2$ définie par :

$$r_1 \times r_2 = \{t \mid \exists t_1 \in r_1, \exists t_2 \in r_2 \text{ t.q. } t[R_1] = t_1 \text{ et } t[R_2] = t_2\}$$

 **Exercice :** quel est le produit cartésien de r_1 et r_2 ?

r_1	A	B	C
	1	2	3
	1	1	1
	1	2	2

r_2	A	B	C
	2	2	2
	1	1	1

 **Attention :** si $R_1 \cap R_2 \neq \emptyset$, *conflit* entre les *attributs de mêmes noms* !

Quelques propriétés

- la projection sur l'ensemble vide est possible ($\langle \rangle$ est le tuple vide)

$$\pi_{\emptyset}(r) = \begin{cases} \emptyset & \text{si } r = \emptyset \\ \langle \rangle & \text{sinon.} \end{cases}$$

- $r_1 \cap r_2 = r_1 - (r_1 - r_2)$
- $\sigma_{\neg F}(r) = r - \sigma_F(r)$
- $\sigma_{F_1 \vee F_2}(r) = \sigma_{F_1}(r) \cup \sigma_{F_2}(r)$
- $\sigma_{F_1 \wedge F_2}(r) = \sigma_{F_1}(r) \cap \sigma_{F_2}(r)$
- $r \times \emptyset = \emptyset$ et $r \times \{\langle \rangle\} = r$

La jointure naturelle

 **Définition :** Soient r_1 et r_2 deux relations sur R_1 et R_2 respectivement. La *jointure naturelle* $r_1 \bowtie r_2$ de r_1 et r_2 , est définie par :

$$r_1 \bowtie r_2 = \{t \mid \exists t_1 \in r_1, \exists t_2 \in r_2 \text{ t.q. } t[R_1] = t_1 \text{ et } t[R_2] = t_2\}$$

Le schéma de $r_1 \bowtie r_2$ est $R_1 \cup R_2$

Remarque :

- opération *très utile* : permet d'agréger/de rassembler des données
- *coûteuse* quand on doit joindre beaucoup de tables ...
- le stockage de données déjà structurées permet aux *modèles noSQL* de limiter ce surcoût !
- ... mais au détriment de la variété d'applications

Exercices

s_1	A	B	C
	2	2	2
	1	1	1

s_2	A	D	E
	1	2	3
	1	1	2
	2	3	1
	3	1	2

s_3	A	B	C
	1	1	1
	1	2	3


s_4	D	E	F
	1	1	1
	2	2	2

⚙ **Exercice** : donner $s_1 \bowtie s_2$, $s_1 \bowtie s_3$, $s_1 \bowtie s_4$.

⚙ **Exercice** : Soient r_1, r_2 deux relations sur R_1, R_2 (resp.)


- Si $R_1 = R_2$, que vaut $r_1 \bowtie r_2$?
- Si $R_1 \cap R_2 = \emptyset$, que vaut $r_1 \bowtie r_2$?

Le renommage

 **Définition** : soit r une relation sur R , $A \in R$ et $B \notin R$. Le *renommage* de A en B dans r , noté $\rho_{A \rightarrow B}(r)$, est la relation sur $(R \setminus \{A\}) \cup \{B\}$ définie par :

$$\rho_{A \rightarrow B}(r) = \{t \mid \exists u \in r \text{ t.q. } t[R - \{A\}] = u[R - \{A\}] \text{ et } t[B] = u[A]\}$$

r_1	A	B	$\xrightarrow{\rho_{B \rightarrow C}}$	$\rho_{B \rightarrow C}(r_1)$	A	C
	0	0			0	0
	0	1			0	1

 **Remarque** : le renommage permet de *forcer* ou d'*éviter* des jointures naturelles


Résumé sur l'algèbre relationnelle

- Opérations ensemblistes

- *Union* (\cup) : sélection des tuples d'une relation r_1 et de ceux d'une autre relation r_2
- *Intersection* (\cap)
- *Différence* ($-$) : sélection des tuples de r_1 qui n'appartiennent pas à r_2

- Autres opérations

- *Projection* (π) : suppression des attributs d'une relation
- *Sélection* (σ) : sélection d'un sous-ensemble de tuples d'une relation.
- *Jointure* (\bowtie) : combine deux relations
- *Renommage* ($\rho_{A \rightarrow B}$) : renomme un attribut
- *Produit* cartésien (\times) : cas particulier de jointure

 **Définition :** Un langage d'interrogation de bases de données relationnelles est *relationnellement complet* s'il peut exprimer toute requête exprimable dans l'algèbre relationnelle

Extensions de l'algèbre relationnelle

- prise en compte des valeurs nulles
- *requêtes récursives*
- fonctions d'agrégation sur les données

Requêtes récursives

Parents	Enfant
p_1	p_3
p_1	p_4
p_2	p_4
p_2	p_5
p_3	p_6
p_3	p_7
p_7	p_8

- « donner tous les descendants de p_1 »
- $\pi_{\text{Parent}, \text{Enfant}}(\rho_{\text{Enfant} \rightarrow AJ}(\text{Famille}) \bowtie \rho_{\text{Parent} \rightarrow AJ}(\text{Famille}))$
- *impossible* en algèbre relationnelle : il faudrait pouvoir tester *toutes* les distances possibles ...
- mais *possible* en SQL !

i **Remarque** : question *particulièrement adaptée* aux *BD graphes* !!!

SQL (Structured Query Language)

- SQL : Langage d'*interrogation des données* très populaire
- Langage construit à partir des langages formels
- Normalisé par l'ANSI en 1992 (SQL-92) puis en 1999 (SQL-99), avec différentes compatibilités :
 - Chaque SGBDR a son propre SQL (pour les aspects hors normes : built-in function, fermeture transitive, ...)

i Remarque : *Défaut d'impédance de SQL* : SQL et les SGBDR sont très puissants, mais utiliser SQL comme interface avec un autre langage pour une application est souvent compliqué !

La sémantique de SQL

! Attention : une relation SQL n'est pas un ensemble de tuples ! C'est un multi-ensemble !

- $\{1, 2, 2, 3\}$ est un multi-ensemble : 2 est présent plusieurs fois

⊗ Problème : les propriétés classiques des ensembles ne sont plus toujours vérifiées ...

- Par exemple : $R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$
- Si $R = S = T = \{1\}$, on a
 - $R \cap (S \cup T) = \{1\}$ mais
 - $(R \cap S) \cup (R \cap T) = \{1, 1\}$

Exemple de base de données


Personne	nom	prénom	ville	âge
	Micoton	Mylène	Nancy	27
	Groin	Philippe	Rennes	25

Aime	nom	isbn
	Micoton	2815934558
	Groin	2070468755
	Micoton	2070468755

Livre	auteur.e	isbn	titre	année
	Lem	2070468755	Solaris	1961
	Nelscott	2815934558	Quatre jours de rage	2019

- $R = \{\text{personne}, \text{aime}, \text{livre}\}$ avec $\text{personne}(\underline{\text{nom}}, \text{prénom}, \text{ville}, \text{âge})$, $\text{livre}(\text{auteur(e)}, \underline{\text{isbn}}, \text{titre}, \text{année})$, et $\text{aime}(\underline{\text{nom}}, \underline{\text{isbn}})$
- base de données **bibliotheque** = $\{\text{Personne}, \text{Aime}, \text{Livre}\}$
- attributs soulignés : clés (primaires)
- attributs en **gras** : clés étrangères

Créer une base de données

 **Syntaxe :** on utilise les mots clés `CREATE DATABASE`

```
CREATE DATABASE nom_bd;
```

Remarque :

- Le SGBD se débrouille pour créer et stocker les fichiers correspondants
- pour SQLite, il faut plutôt créer la table avant de lancer l'interface

```
CREATE DATABASE bibliotheque;
```

Créer des tables

 **Syntaxe** : on va utiliser le bloc `CREATE TABLE`

```
CREATE TABLE nom_relation (  
    attr1 type [contraintes],  
    attr2 type [contraintes],  
    ...  
    [contraintes]  
);
```

Remarque :

- [] indique des éléments facultatifs,
- type est la nature de l'attribut (`INTEGER`, `CHAR`, ...)

Quelques contraintes :

- `PRIMARY KEY` : clé primaire
- `FOREIGN KEY` (attr) `REFERENCES` autre_relation(attr) : clé étrangère
- `NOT NULL` : forcer l'utilisateur.ice à renseigner la valeur

Exemple

```
CREATE TABLE personne (  
    nom CHAR(50) PRIMARY KEY NOT NULL,  
    prenom CHAR(50),  
    ville CHAR(50),  
    age INTEGER  
);
```

```
CREATE TABLE livre (  
    isbn INTEGER PRIMARY KEY NOT NULL,  
    auteur CHAR(50),  
    titre CHAR(50),  
    annee INTEGER  
);
```

Exemple

```
CREATE TABLE aime (  
    nom CHAR(50) NOT NULL,  
    isbn INTEGER NOT NULL,  
    date_lecture TEXT,  
    PRIMARY KEY (nom, isbn),  
    FOREIGN KEY (nom) REFERENCES personne(nom),  
    FOREIGN KEY (isbn) REFERENCES livre(isbn)  
);
```

! **Attention** : quand la contrainte **PRIMARY KEY** porte sur plusieurs attributs, il faut la préciser *à la fin*

Insérer des tuples

 Syntaxe : bloc `INSERT INTO`

```
INSERT INTO nom_relation(attribut1, attribut2, ...)  
VALUES (val1, val2, ...);
```

Dans notre exemple

```
INSERT INTO personne(nom, prenom, ville, age)  
VALUES ('micoton', 'mylene', 'nancy', 27);
```

```
INSERT INTO livre(isbn, auteur, titre, annee)  
VALUES (2070468755, 'lem', 'solaris', 1961);
```

```
INSERT INTO aime(nom, isbn, date_lecture)  
VALUES ('micoton', 2070468755, date('2023-10-25'));
```

Syntaxe d'une requête SQL

- Une requête SQL comporte trois clauses :

```
SELECT <liste attributs>  
FROM <listes relations>  
[WHERE <conditions sur les lignes>]
```

- requête SFW :
 - **SELECT** code la *projection*
 - **WHERE** code la *sélection*
- le résultat est une relation sur le schéma défini par la clause **SELECT**
- Une requête SQL peut être nommée \Rightarrow on parle de *vue*.

Passerelles entre ensembles et multi-ensembles

! Attention : les requêtes SFW sont *par défaut* définies sur des multi-ensembles, sauf si des opérations ensemblistes (**UNION**, **INTERSECTION**, **MINUS**) sont utilisées. Dans ce cas, le résultat de la requête sera un ensemble

- Pour avoir des ensembles : utiliser **DISTINCT** après **SELECT**
- pour avoir des multi-ensembles : utiliser **ALL** après des opérateurs ensemblistes

! Attention : se méfier du surcout engendré par **DISTINCT** pour filtrer les doublons

Requête SFW

- « Donner le *nom* et *prénom* des personnes de 25 ans »

```
SELECT DISTINCT nom, prenom  
FROM Personne  
WHERE age = 25;
```

i Remarque : les mots-clés du langage (**SELECT**, ...) ne sont pas sensibles à la casse

Le renommage en SQL

- Possible sur les attributs de la clause **SELECT** et sur les relations de la clause **FROM**
- mot-clé réservé : **AS**

```
SELECT DISTINCT nom [AS] 'Mes collègues'  
FROM Personne [AS] P  
WHERE [P.]age = 27;
```

i Remarque : **AS** est optionnel. Pour résoudre les ambiguïtés, on préfixe l'attribut par le nom de sa relation, i.e. `relation.nom`

Conditions complexes pour la sélection

- mots-clés **AND**, **OR**, **NOT** et parenthèses
- permettent de créer des formules de sélection complexes
- « Donner le *nom* et *prénom* des personnes de plus de 20 ans ou qui habite à Nancy »

```
SELECT DISTINCT nom, prenom
FROM   Personnes
WHERE  (ville = 'Nancy') OR (age > 20);
```

Douceurs syntaxiques : les caractères jokers

- ne s'exprime pas en algèbre relationnelle
- dans la clause **SELECT** :
 - caractère ***** pour lister tous les attributs
 - expressions arithmétiques usuelles sur les valeurs retournées

```
SELECT DISTINCT nom, age - 10  
FROM personne;
```

- dans la clause **WHERE** :
 - mot clé **LIKE** avec les caractères jokers : %, _

```
SELECT DISTINCT nom  
FROM personne  
WHERE prenom LIKE 'M%';
```

Jointure naturelle en SQL

- **NATURAL JOIN** qui va joindre des tables sur les attributs avec le même nom

```
SELECT attributs  
FROM R1 NATURAL JOIN R2
```

- en précisant les attributs sur lesquels joindre des relations

```
SELECT attributs  
FROM R1, R2  
WHERE R1.A1 = R2.A2;
```

« Donner les *noms*, *prénoms* des personnes et les *isbn* des livres associés »

```
SELECT DISTINCT nom, prenom, isbn  
FROM Personne NATURAL JOIN Aime;
```

```
SELECT DISTINCT nom, prenom, isbn  
FROM Personne, Aime  
WHERE Personne.nom = Aime.nom;
```

Produit cartésien

- Cas particulier de jointure
- Suffit de ne pas expliciter les attributs (2 solutions) :

```
SELECT *  
FROM  Personne CROSS JOIN Aime
```

```
SELECT *  
FROM  Personne, Aime
```

i Remarque : Si un attribut apparaît dans les 2 tables, il est renommé.

Les opérateurs ensemblistes en SQL

- Union : **UNION**
- Intersection : **INTERSECT**
- Différence : **MINUS**

```
SELECT A, B  
FROM R  
UNION  
SELECT A, B  
FROM S
```



Remarque : forcent la sémantique ensembliste par défaut

Douceurs syntaxiques : composition de requêtes

- Le résultat d'une requête SFW peut être utilisé dans la clause **FROM** ou **WHERE** d'une autre requête
- Les sous-requêtes de la clause **WHERE** sont introduites par les mots-clés **IN**, **EXISTS**, **ANY**, **ALL**

 **Remarque :** *beaucoup de sucre syntaxique* possible avec SQL

Modélisation EA

Structure

Langages de requêtes

Modélisation EA

Modèle EA

traduction vers un schéma relationnel

Exercice

i Remarque : (idée générale) vu la structure du modèle relationnel, implémenter une base de données demande une modélisation minutieuse.

Modèle Entité-Association (EA)

- Le plus connu en France est le *Modèle Entité-Association* (ou Entité-Relation), défini en 1974 !
 - *modèle conceptuel des données* (MCD) de la méthode de conception des systèmes d'information MERISE
- Dans le modèle EA :
 - Les aspects *dynamiques* ne sont pas pris en compte
 - Du fait du haut niveau d'abstraction, les *langages* n'ont pas été développés
 - Les *contraintes* restent anecdotiques de part leur puissance structurelle

Intérêts des modèles conceptuels

- Rapprocher le domaine d'application de sa représentation informatique
 - modélisation du monde réel plus naturelle, plus directe
 - dialogue plus aisé entre informaticien.nes et utilisateur.ices
 - aucune considération d'implémentation
 - produit souvent des schémas de relation déjà normalisés
- Représentation *graphique* des constructeurs
 - intuitive, facile, et « *conviviale* »
 - mais risque de syndrome « *flèches/patates* »
- *Mécanismes d'abstraction* pour représenter les données :
 - *Classification* : regrouper les objets similaires dans une *entité* ou un attribut (en gommant les différences entre les objets)
 - *Agrégation* : obtenir une nouvelle entité à partir d'objets/attributs existants


Modèles conceptuels/logiques/physiques


- Modèle EA \Rightarrow modèle *conceptuel*
- Modèle relationnel \Rightarrow modèle *logique*
- Éléments du C++ pour coder les données \Rightarrow modèle *physique*
- Les différences entre les modèles impliquent une étape de *traduction de schémas*

Limites des modèles conceptuels

1. *Subjectivité* : *un* problème, *plein* de solutions possibles
 - on peut interpréter la *même information* de façon différente
 - critères de choix d'une solution objectifs (dépendances fonctionnelles) ou subjectifs (élégance du schéma)
2. Représentation graphique des constructeurs : syndrome des « *flèches/patates* » \Rightarrow on peut faire n'importe quoi !
3. Plusieurs modèles conceptuels
 - Quel(le)s modèles/extensions sont pris en compte ?
 - Quelles conventions pour les cardinalités ? (UML ? MERISE ?)
 - Tous les modèles conceptuels sont différents ...

Les types d'entités

 **Définition** : un *type d'entités* est une abstraction d'un ensemble d'*entités* du monde réel qui ont des caractéristiques communes, appelées *attributs*.

 **Astuce** : en clair, on regroupe dans un seul « *concept* » des objets qui ont plus de points communs que de différences.

élève
nom : <code>string</code>
prénom : <code>string</code>
âge : <code>int</code>

école
nom : <code>string</code>
adresse : <code>string</code>
direction : <code>string</code>
capacité : <code>int</code>

Les clés dans les types d'entités

- Directement récupérées du *modèle relationnel*!
- Permet de spécifier un sous-ensemble d'attributs du type d'entités comme étant un *identifiant unique* des objets représentés par celui-ci.
- Deux cas :
 - il existe un identifiant « *naturel* » (ISBN, numéro de sécu, ...)
 - il n'en existe pas : on ajoute un attribut (*surrogate key*) qui fera office de clé.

! **Attention** : chaque type d'entités doit avoir un *identifiant*.

élève
<u>code</u> : int
nom : string
prénom : string
âge : int

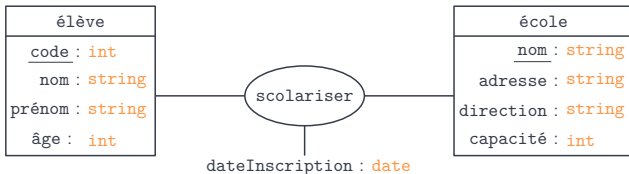
école
<u>nom</u> : string
adresse : string
direction : string
capacité : int

Les types d'associations

Un *type d'associations* est une abstraction d'un ensemble d'associations existantes entre des objets du monde réel qui sont devenus des types d'entités.

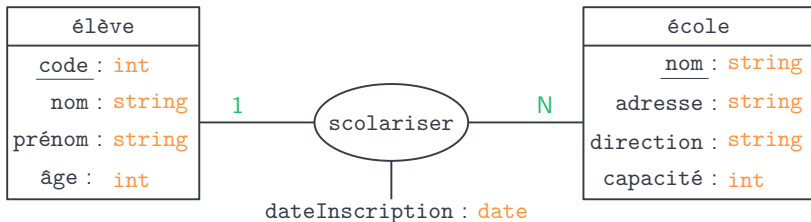
i Remarque :

- nécessite au moins deux types d'entités
- ne nécessite pas de définition de clés, même si c'est possible



Cardinalité des types d'association

- *cardinalités* = *contraintes de participation* des entités à l'association
- deux types de cardinalités :
 - *cardinalité maximale* : nombre maximal d'associations auxquelles un objet donné dans E_i peut participer. Vaut 1 (*une seule*) où N (*plusieurs*).
 - *cardinalité minimale* : nombre minimal d'associations auxquelles un objet donné dans E_i peut participer. Vaut 0 ou 1.



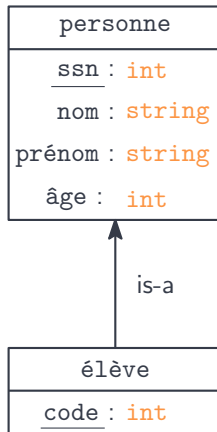
Quelques extensions possibles

- *Spécialisation* : à partir d'une entité donnée, consiste à représenter une nouvelle entité qui est plus spécifique, i.e. qui dispose d'attributs supplémentaires.
- *Généralisation* : à partir de plusieurs entités similaires, consiste à représenter une nouvelle entité qui factorise les propriétés communes aux différentes entités.
- *Associations généralisées* : on permet à une association d'agréger indifféremment des types d'entités et des types d'associations

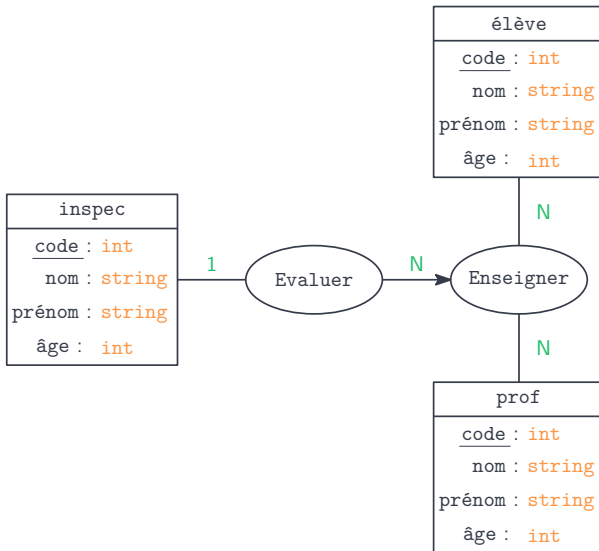
i Remarque : ça n'est pas sans rappeler la notion d'*héritage* des langages objets ...

Exemple

- On peut voir les élèves comme des cas particuliers de personne.
- d'entités `personne` qui a comme attributs `ssn` (numéro de sécu), `nom`, `âge` et `prénom` (récupéré de `élève`). Le lien qui unit `élève` à `personne` est annoté *is-a*.



Représentation graphique




Construction d'un schéma conceptuel

 **Question :** quelles sont les caractéristiques du monde réel à modéliser ?


 **Réponse :**

- nécessite des entretiens avec les « *client.es* »
 - lecture de documents de spécification (quand ils existent)
-
- Conseils à partir d'un énoncé en langage naturel :
 1. Décomposition du texte en propositions élémentaires : *sujet - verbe - complément*
 2. Premières structures EA
 - ▷ sujet/complément ⇒ *type d'entités* ou *attribut*
 - ▷ verbe ⇒ *type d'association*

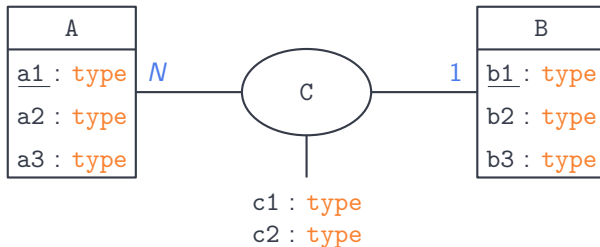
Traduction entre modèles

 **Question** : Comment obtenir un schéma de bases de données à partir d'un schéma Entité-Association ?

- Chaque type d'entités E va devenir une relation, dont le schéma contient *au moins* les attributs de E .
- les associations se traduisent en clés étrangères ou en relations *en fonction des cardinalités*

 **Idée** : Les contraintes implicites du modèle conceptuel (MCD) vont devenir des *clés* et des *clés étrangères* du modèle relationnel (MLD) !

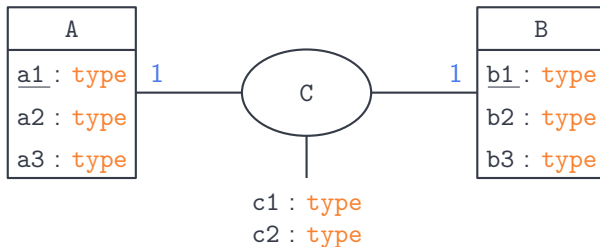
Associations 1 – N



Traduction en relationnel :

- B caractérise l'association : les propriétés de l'association sont déterminées par B
- B doit permettre d'identifier le A auquel il est associé : clé étrangère dans B (marquée en gras) !
- A(a1, a2, a3), B(b1, b2, b3, c1, c2, a1)

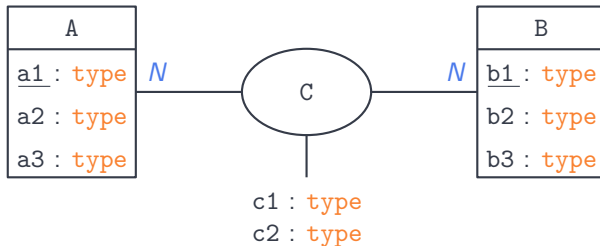
Associations 1 – 1



Traduction en relationnel :

- A et B en correspondance : ils se caractérisent l'un l'autre
- on peut ne garder qu'une seule entité, mettons A, avec deux clés
- A(a1, b1, a2, a3, b2, b3, c1, c2)

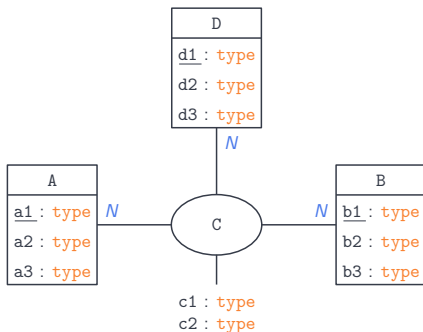
Associations $N - N$



Traduction en relationnel :

- A et B n'ont aucune forme de dépendance
- on crée une relation pour C qui centralise l'information de l'association, en référençant A et B.
- les clés de A et B forment la clé de C
- $A(\underline{a1}, a2, a3), B(\underline{b1}, b2, b3), C(\underline{a1}, \underline{b1}, c1, c2)$

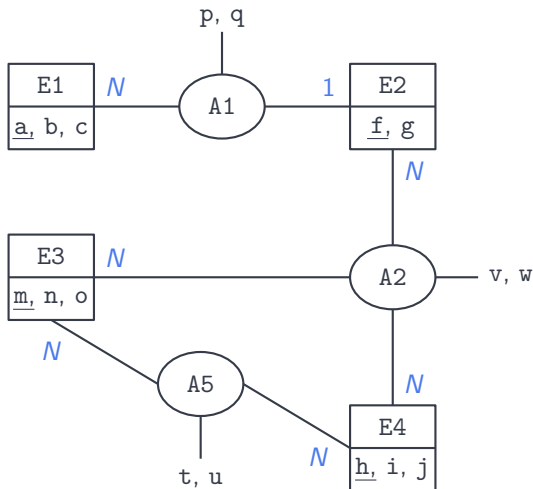
Associations ternaires



Traduction en relationnel :

- l'association devient une entité
- puis traitement comme une relation binaire N - N
- $A(\underline{a1}, a2, a3), B(\underline{b1}, b2, b3), D(\underline{d1}, d2, d3), C(\underline{a1}, \underline{b1}, \underline{d1}, c1, c2)$

Exercice



⚙️ **Exercice :** extraire un schéma de bases de données du modèle conceptuel ci-dessus, en déduire les contraintes (clés, clés étrangères)

Du relationnel vers EA


- Connus sous le nom de *retro-ingénierie* (ou « *reverse engineering* »)
- Hypothèses requises pour obtenir un résultat intelligible
 - soit toutes les contraintes sont connues \Rightarrow les schémas pouvant être quelconques
 - soit aucune contrainte n'est connue \Rightarrow il faut des hypothèses de nommage *fortes*

La boucle est bouclée

Café	nomcafé	adresse	téléphone	avis
	les délices de l'abstraction	1 bd. de l'Impasse, 13007, Marseille	04 12 12 12 12	5/5
	Rakwé	18 rue du Moka, 13190, Allauch	08 36 65 65 65	4/5
	Le Maté-matique	3bis av. du rire, 13010, Marseille	04 36 30 36 30	2/5

Personne	nom	prénom	adresse
	Nette	Marie	3 rue Lumière, 54500, Langres
	Fenouil	Emile	5 place du Potiron, 63036, Ceyssat
	Taie	Clara	17 bd. du Flan, 17001, La Rochelle

Aime	nom	café
	Nette	Le Maté-matique
	Fenouil	Rakwé
	Taie	les délices de l'abstraction
	Nette	Rakwé

 **Exercice** : retrouver un modèle conceptuel qui aurait pu donner cette base de données

Un dernier exo pour la route

Commandes	num	cnom	pnom	qte
	1534	Lise	crocs	6
	1854	Lise	klaxon	20
	1254	Arthur	espadrilles	20
	1259	Arthur	espadrilles	25
	1596	Arthur	crocs	12

Produits	pnom	fnom	prix
	crocs	Le Campeur âgé	20
	crocs	navelle	18
	klaxon	Vintude	8
	savon	Vintude	4
	klaxon	navelle	19
	navelle	navelle	5
	espadrilles	navelle	5

Fournisseurs	fnom	statut	ville
	Le Campeur âgé	SARL	Allauch
	navelle	SA	Allauch
	Vintude	SA	Le Redon
	Nimylu	Assoc	Aix

Et les questions

Pour chacune des questions suivantes, proposer une requête en algèbre relationnelle et en SQL :

1. donner les informations sur toutes les commandes
2. donner le nom de tous les produits commandés
3. donner le nom des produits commandés par Arthur
4. donner le nom des fournisseurs de crocs ou d'espadrilles à un prix égal à 5 euros
5. quels sont les produits dont le nom est le même que le nom des fournisseurs ?
6. donne le nom, le prix et le fournisseur des produits commandés par Lise
7. quels sont les fournisseurs qui se trouvent dans la même ville ?
8. quels sont les produits qui coûtent moins de 15€ ou qui sont commandés par Arthur ?
9. quels sont les produits commandés en quantité supérieure à 10 et dont le prix est au-dessus de 8 euros ?

Correction

1. donner les informations sur toutes les commandes :

- algèbre relationnelle : *Commandes*
- SQL :

```
SELECT *  
FROM Commandes;
```

2. donner le nom de tous les produits commandés :

- algèbre relationnelle : $\pi_{pnom}(\textit{Commandes})$
- SQL :

```
SELECT DISTINCT pnom  
FROM Commandes;
```

Correction

3. donner le nom des produits commandés par Arthur

- algèbre relationnelle : $\pi_{pnom}(\sigma_{cnom='Arthur'}(Commandes))$
- SQL :

```
SELECT DISTINCT pnom
FROM Commandes
WHERE cnom = 'Arthur';
```

4. donner le nom des fournisseurs de crocs ou d'espadrilles à un prix égal à 5 euros

- algèbre relationnelle : $\pi_{fnom}(\sigma_{prix=5 \wedge (pnom='crocs' \vee pnom='espadrilles')}(Produits))$
- SQL :

```
SELECT DISTINCT fnom
FROM Produits
WHERE prix = 5 AND (pnom = 'crocs' OR pnom = 'espadrilles');
```

Correction

5. quels sont les produits dont le nom est le même que le nom des fournisseurs ?

- algèbre relationnelle : $\sigma_{pnom=fnom}(Produits)$
- SQL :

```
SELECT *  
FROM Produits  
WHERE pnom = fnom;
```

6. donne le nom, le prix et le fournisseur des produits commandés par Lise

- algèbre relationnelle : $\pi_{pnom,fnom,prix}(Produits \bowtie \sigma_{cnom='Lise'}(Commandes))$
- SQL :

```
SELECT pnom, fnom  
FROM Produits NATURAL JOIN (  
    SELECT *  
    FROM Commandes  
    WHERE cnom = 'Lise') AS C2;
```

Correction

7. quels sont les fournisseurs qui se trouvent dans la même ville ?

- algèbre relationnelle :

$$\pi_{fnom, fnom1}(\sigma_{fnom \neq fnom1}(Fournisseurs)) \bowtie \rho_{fnom \rightarrow fnom1, statut \rightarrow statut1}(Fournisseurs))$$

- SQL :

```
SELECT DISTINCT fnom, fnom1
FROM Fournisseurs NATURAL JOIN (
  SELECT ville, fnom AS fnom1, statut AS statut1
  FROM Fournisseurs ) as F2
WHERE fnom != fnom1;
```

Correction

8. quels sont les produits qui coûtent moins de 15€ ou qui sont commandés par Arthur ?

- algèbre relationnelle : $\pi_{pnom}(\sigma_{prix \leq 15 \vee cnom = 'Arthur'}(Commandes \bowtie Produits))$
- SQL :

```
SELECT pnom
FROM Produits NATURAL JOIN Commandes
WHERE prix <= 15 OR cnom = 'Arthur';
```

Correction

9. quels sont les produits commandés en quantité supérieure à 10 et dont le prix est au-dessus de 8 euros ?

- algèbre relationnelle : $\pi_{pnom}(\sigma_{prix \geq 8}(Commandes) \bowtie (\sigma_{qte \geq 10}(Produits)))$
- SQL :

```
SELECT DISTINCT pnom
FROM
  (SELECT * FROM Produits WHERE prix >= 8) AS P1
  NATURAL JOIN
  (SELECT * FROM Commandes WHERE qte >= 10) AS C2;
```