Poly de Cours 1.

Le modèle relationnel

Objectifs du Chapitre:

- ✎ comprendre le modèle relationnel, en connaître le vocabulaire
- ✎ savoir écrire des requêtes dans l'algèbre relationnelle.

1 Le modèle relationnel: introduction

1.1 Organiser et modéliser les données: le modèle relationnel

Les bases de données organisent en *collections* les données qu'elles contiennent. Une collection est un ensemble d'objets partageant à peu de chose près les mêmes attributs. Par ex: la collection des clients d'un magasin, où chaque client a une adresse, un nom, prénom, un numéro de sécurité sociale, etc.[1] Des données sont dites *structurées* lorsqu'elles sont organisées en collections. Le modèle relationnel représente les données sous forme de tables: chaque collection est modélisée par une table, que l'on appellera *relation*. Chaque ligne de la table est un tuple de la relation, et chaque colonne correspond à un attribut (une propriété de l'objet), comme on peut le voir en Figure 1. Ceci implique que tous les éléments de la collection ont exactement le même ensemble d'attributs.[2]... Ce modèle relationnel a été introduit en 1970, par un ingénieur d'IBM: E.F. Codd. Codd a obtenu un prix Turing (en 1981), principalement pour avoir proposé :

- un modèle de données: le modèle relationnel
- un langage pour manipuler ces données: l'algèbre relationnelle

Une base de données relationnelle est alors définie comme un ensemble de relations traitant d'un même sujet. La plupart des systèmes de gestion de bases de données (Oracle, PostgreSQL, MariaDB, MySQL, Microsoft SQL Server, SQLite...) s'inscrivent dans ce modèle relationnel. Pour évaluer les requêtes soumises par les utilisateurs, les systèmes de gestion de bases de données traduisent ces requêtes en opérations de l'algèbre relationnelle.[3] Nous commencerons par étudier le modèle relationnel, puis présenterons l'algèbre relationnelle.

1.2 Le modèle relationnel: définition formelle

Nous allons désormais donner une définition formelle (i.e., "mathématique") des relations. Pour définir une relation, il convient d'abord de définir son *schéma*. On va ainsi distinguer le schéma (description de l'organisation

[1] Dans certaines collections, les éléments n'ont pas forcément tous les mêmes attributs. Par exemple, si vous comparez les infobox sur la page des compositeurs Bach et Mozart, vous verrez que seul Bach a un attribut "surnom", tandis que Mozart a, lui, un attribut "conjoint", etc. Il reste raisonnable de regrouper dans une même collection "personnalités de musique classique" ces auteurs car ils partagent la plupart de leurs attributs. Ces légères variations d'attributs entre éléments d'une même collection sont plus fréquents dans les modèles de données récents comme les collections de document JSON (ou XML). Mais ce cours ne présentera que le modèle relationnel qui exclut ces "irrégularités" .

[2] En pratique, lorsqu'un attribut n'est pas pertinent pour un élément particulier, (ou que sa valeur est inconnue) on remplit la cellule correspondante avec la valeur NULL, mais cette valeur NULL est uniquement présente dans le langage SQL, elle n'existe pas dans le modèle relationnel.

[3] En réalité, le langage de requête utilisé en pratique par les utilisateurs, SQL, est plus riche que l'algèbre relationnelle, donc ce n'est pas exactement l'algèbre relationnelle qui est utilisée, mais un ensemble d'opérations un peu plus variées, dont l'algèbre relationnelle reste quand même le "cœur".

Client

NSS	NOM	PRENOM	ADRESSE
1111	GROZ	Benoit	75016
3333	BIDOIT	Nicole	75014
2222	COHEN	Sarah	75008

Achat

ID_CLIENT	NOM_PRODUIT	JOUR
1111	cidre	01/01/2016
1111	pain	01/01/2016
2222	portable	05/01/2016
2222	DVD	07/01/2016

Figure 1: Une base de données, constituée de 2 relations: Client et Achat.

des données) de l'instance (les données elles-mêmes), que ce soit au niveau des relations ou de la base de données dans son ensemble. C'est le schéma qui permet à l'utilisateur de formuler facilement ses requêtes sans avoir à consulter les données au préalable.[4].

Domaine (=type) d'un attribut

On associe à chaque attribut a, un ensemble de constantes, potentiellement infini, noté $\text{dom}(a)$.

Exemple: dans la table `Achat` en Figure 1: dom(Nom_produit) pourrait être l'ensemble des chaînes de caractères, et dom(jour) l'ensemble des dates.

Schéma relationnel

De façon générale, le schéma d'une collection décrit la structure des éléments au sein d'une collection. Dans le modèle relationnel, le *schéma* d'une relation est défini par :

- le nom de la relation (ex: `Achat`)
- un ensemble de noms d'attributs ((ex: $\{id_client, nom_produit, jour\}$)
- le domaine de chaque attribut (ex: $\text{dom}(id_client)$ =chaînes de caractères composées de 4 chiffres)

Notation: On notera parfois R(U) ou R[U] le schéma d'une relation R dont l'ensemble d'attributs est U (en omettant abusivement les types).

On se limite généralement à des types atomiques pour le domaine (ex: entiers, string, pas ~~vector<int>~~) ...)

Relation = Instance d'un schéma relationnel

Une instance du schéma relationnel R[U] (avec $U = \{u_1, \ldots, u_k\}$) est un ensemble fini de tuples (=n-uplets), où un tuple est une fonction qui associe à chaque attribut de U un élément de son domaine. Par abus, on ordonnera souvent arbitrairement les attributs de U et un tuple sera juste un élément de $\text{dom}(u_1) \times \text{dom}(u_2) \times \ldots \text{dom}(u_k)$.

Notation: Soit t un n-uplet. On notera $t(u_i)$ ou $t.u_i$ la valeur de associée par t à l'attribut u_i. Soit $V \subseteq U$ un ensemble d'attributs. On notera $t_{|V}$ la restriction du tuple t aux attributs dans V.

Exemple: Dans la relation `Client` en Figure 1, on a 3 tuples. Un des tuples est {`NSS`→ `2222`, `NOM`→ `COHEN`, `PRENOM`→ `Sarah`, `ADRESSE`→ `75008`}, que l'on notera plus souvement abusivement `(2222, COHEN, Sarah, 75008)`. Soit t ce tuple, alors $t(NSS) = t.NSS = 2222$, et $t_{|(NSS,ADRESSE)} = (2222, 75008)$.

Schéma et Instance d'une base de données

Le *schéma* d'une base de données est un ensemble de schémas de relation. Une instance de ce schéma est un ensemble de relation (une pour chaque schéma relationnel).

On peut voir une instance comme l'état courant (ou parfois un état hypothétique) de la relation/base de données, alors que le schéma fait abstraction du contenu détaillé. Exemple: la Figure 1 représente une instance d'une base de données. La Figure 3 représente de façon graphique le schéma d'une autre base de données (en omettant les domaines). La Figure 4 représente une instance d'une autre base de données dont le schéma est donné en Figure 5.

Cardinalité et arité d'une relation

L'arité d'une relation (ou de son schéma) R(U) est son nombre d'attributs $|U|$. La cardinalité d'une relation R est son nombre de tuples $|R|$.

Exemple: en Figure 1, l'arité de la relation `Achat` est 3, sa cardinalité est 4.

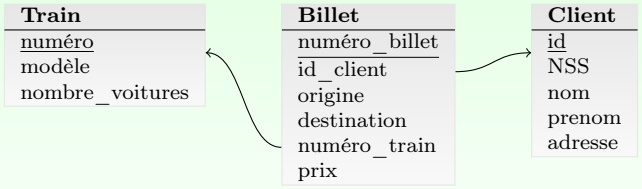
Remarques:

- Les schémas sont une (méta)donnée. Dans les principaux SGBDs, cette information est typiquement stockée directement à l'intérieur de la base de données dans des relations dédiées, qui vont lister les attributs, les tables, etc. avec leurs propriétés.
- Le généralement accessible aux utilisateurs: même si on n'autorise pas toujours l'utilisateur à accéder aux tables systèmes dans la base, on lui fournit généralement le schéma de la base de donnée (ou plus exactement de la partie de la base de données à laquelle il a accès), souvent sous une forme graphique.
- Le schéma d'une relation est généralement fixé une fois pour toutes lors de la création de la base de données. Ensuite, il ne varie généralement pas, ou très rarement. L'instance, elle, peut changer beaucoup plus souvent.

[4] L'idée est que le schéma est une représentation abstraite, de haut niveau, des données. Cette idée d'abstraction est omniprésente en informatique, et particulièrement dans la gestion de données: classes abstraites/interfaces dans les langages orientés objets, niveaux d'abstractions dans les architectures logicielles et en particulier l'architecture à 3 niveaux ANSI SPARC pour les SGBD...

Modèle Relationnel	SQL	
Relation	= Table	client
Attribut	= Colonne	adresse
Domaine	= Type	string
Clefs	= Clefs	nss : clef primaire de Client

Figure 2: Correspondances entre les concepts du modèle relationnel et SQL



Dans cet exemple, la base de données comporte 3 tables `Train`, `Billet` et `Client` d'arités respectives 3, 6, et 5. On n'a ici aucune information sur la cardinalité des relations/tables, bien entendu.

Figure 3: Représentation d'un schéma de base de données (mais sans le type de chaque attribut).

1.3 Différences entre le modèle relationnel et SQL

Le modèle relationnel étant un modèle purement théorique, les SGBDs utilisent en pratique souvent un vocabulaire légèrement différent, correspondant aux concepts de SQL, cf Figure 2. Parmi les principales différences entre le modèle relationnel et SQL: SQL autorise une table à contenir des doublons: c'est rare, car les tables contiennent souvent un identifiant unique, mais cela arrive (et c'est assez fréquent dans les résultats de requêtes). Il n'y a pas fondamentalement d'ordre entre les lignes d'une table en SQL, par contre SQL permet de spécifier un ordre dans lequel on souhaite afficher le contenu d'une table. En SQL comme dans le modèle relationnel, les relations d'une base de données ont des noms distincts, ainsi que les attributs d'une même relation, mais il est possible dans une requête de renvoyer une table dont plusieurs colonnes porteront le même nom. Le concept de clé, que nous détaillons ci-dessous, a été formalisé dans le cadre du modèle relationnel avant d'être implémenté en SQL. Mais SQL permet de définir d'autres contraintes d'intégrité (en particulier pour gérer les NULLs).

1.4 Représentation graphique des schémas

Nous n'étudierons pas en détail la conception de schéma cette année, donc nous n'aborderons pas les représentations standardisées de schémas. Toutefois on peut déjà remarquer que la représentation textuelle des schémas est généralement remplacée dans les outils par une représentation graphique, plus ou moins similaire à celle représentée en Figure 3 sauf que l'on y indiquera très souvent le type des données de chaque attribut.

1.5 Clés et autres contraintes d'intégrité

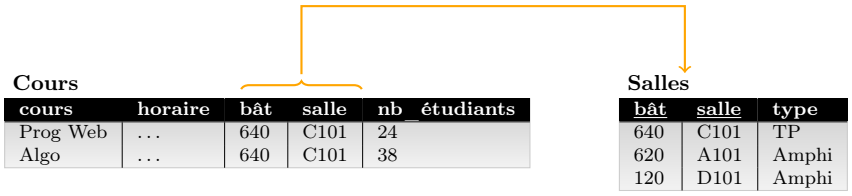
Le schéma de la base inclut en général des clés pour garantir la cohérence des données. L'idée est que le SGBD refuse toute insertion ou modification de la table si la mise à jour viole la contrainte. Sans entrer dans les détails de la conception de schéma, on peut définir une clé comme suit:

Clé d'une relation (plus exactement de son schéma)

Une *clé* de la relation $R(U)$ est groupe d'attributs $V \subseteq U$ minimum (pour l'inclusion) qui détermine chaque tuple de façon unique. Cela signifie que la relation R ne peut pas contenir 2 tuples dont la restriction à V soit identique.

C'est au niveau du schéma qu'on détermine/décide si un attribut ou groupe d'attributs forme une clé de R: il s'agit de garantir cette propriété pour toute instance possible de la table, et pas seulement pour l'instance que l'on observe à un instant particulier.

Exemple: `NSS` est probablement une clé de la table client. Jour n'est pas une clé dans Achat, mais `prénom` n'est pas une clé de Client non plus.



Cours

cours	horaire	bât	salle	nb_étudiants
Prog Web	...	640	C101	24
Algo	...	640	C101	38

Salles

bât	salle	type
640	C101	TP
620	A101	Amphi
120	D101	Amphi

Cours ne peut faire référence qu'à des salles (avec leur bâtiment) répertoriées dans **Salles**.
Par contre bâtiment et salle peuvent être nuls dans **Cours**.

Figure 4: Un exemple de clé étrangère

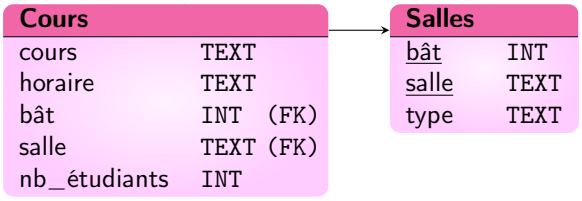


Figure 5: Schéma d'une base de données, avec types et clés.

Clé primaire (🇬🇧 Primary Key) d'une relation

Il s'agit la d'un concept SQL et non pas d'un concept du modèle relationnel. SQL permet si on le souhaite de définir une *clé primaire* par table. Il s'agit d'une clé que l'utilisateur va identifier comme le moyen "par défaut" d'identifier une ligne de la table. On ne peut donc définir qu'*une seule clé primaire* par table (au plus). Selon les SGBDs, cette clé pourra alors jouer un rôle plus important que les autres clés: par exemple, SQL Server ou MySQL/MariaDB vont trier les tuples de la tables selon cette clé, ce qui permet d'accélérer certaines requêtes.

Exemple: si on est sûr que deux clients différents ne peuvent pas avoir simultanément le même nom et le même prénom (tout en permettant qu'ils aient le même nom si les prénoms diffèrent, etc.), il serait alors possible de définir (nom, prénom) comme clé primaire de Client. En pratique, cela ne peut pas être garantit (sauf si le patron décide de bouter hors du magasin tout homonyme d'un de ses clients), donc il faudra définir NSS comme clé primaire, ce qui est justifié si NSS est vraiment unique, comme un numéro de sécurité sociale par exemple.

Les clés étrangères définissent des contraintes d'intégrité référentielle. Cela signifie qu'on garantit par exemple que tout client mentionné dans un billet fait bien référence à un client enregistré dans la base.

Clé étrangère (🇬🇧 Foreign Key) d'une relation

Une clé étrangère est un groupe d'attributs d'une relation R1 devant apparaître comme clé primaire (ou au moins comme clé) dans une autre relation R2. Elle sert à relier les 2 tables en imposant que, si la valeur des attributs est renseignée pour un tuple t de R1, alors il doit y avoir un tuple t' dans R2 qui a la même valeur sur ces attributs : chaque tuple t de R1 fait ainsi "référence à" un tuple t' de R2.

Exemple: dans la table `Achat` en Figure 1, if_client pourrait être défini comme clé étrangère faisant référence à Client.NSS. Les flèches en Figure 3 représentent deux clés étrangères. Comme pour les clés (primaires), une clé étrangère peut être formée de plusieurs attributs, comme illustré en Figure 4.

Une convention graphique typique est de souligner les éléments formant la clé primaire d'une relation (presque toutes les tables ont une clé primaire en pratique). Les clés étrangères étant souvent représentées par des flèches. Il existe d'autres représentations (ex: indiquer (PK) à côté du nom de l'attribut).

2 L'algèbre relationnelle

Par algèbre, nous entendons ici simplement un ensemble avec des opérations fermées sur cet ensemble. On trouve parfois de petites différences de notation entre les sources, donc il ne faudra pas s'étonner si vous tombez un jour sur des conventions (très légèrement) différentes de celles utilisées dans ce poly.

2.1 Propriétés générales

L'algèbre relationnelle a été introduite par Codd pour modéliser les principales opérations permettant de manipuler des relations. Les opérateurs de l'algèbre relationnelle prennent en entrée une ou plusieurs relations, et renvoient une relation. Ceci permet donc de composer plusieurs opérations pour former une requête. Une requête en algèbre relationnelle est donc une expression, qui peut être vue comme une fonction associant une relation

(le résultat) à toute instance de la base. Quand on formule une requête en algèbre relationnelle, le résultat est obtenu en "évaluant" l'expression sur l'instance courante de la base de données.

On rappelle ici que l'utilisateur formule typiquement sa requête sans consulter l'instance, en se basant exclusivement sur le schéma. On rappelle aussi qu'il n'y a pas de doublons, etc. dans une relation, donc a fortiori quand on évalue une expression de l'algèbre relationnelle sur une instance de la base de données.

L'algèbre relationnelle est un langage de requête purement théorique. Il n'existe pas d'outil (du moins d'outil populaire) permettant d'évaluer une expression de l'algèbre relationnelle sur des tables d'une base de données. Mais l'algèbre sert à modéliser les opérations effectuées par le langage SQL, et est au coeur des techniques utilisées par les SGBD pour optimiser l'évaluation des requêtes. En réalité, les opérations que nous allons discuter: jointure, sélection, projection sont très générales et se retrouvent sous une forme ou une autre au delà des SGBD relationnels dans beaucoup d'outils manipulant des données.

Les opérateurs de l'algèbre relationnelle peuvent être classifiés en fonction de leur arité (en tant qu'opérateur on peut parler d'arité = nombre d'arguments, à ne pas confondre avec l'arité d'une relation ci-dessus)

- *2 opérateurs unaires* : projection, sélection
- *3 opérateurs binaires* : jointure, union, différence
- *1 opérateur de renommage*
- *opérateurs dérivés* : produit cartésien, intersection, division, ...
- Typage : *schéma cible* déterminé par le schéma source et l'opérateur.

Les opérateurs "dérivés" peuvent être déduits des opérateurs précédents.

Typage : chaque *opérateur* définit une *application* de l'ensemble des instances d'un schéma de base de données (source) dans l'ensemble des instances d'un schéma de relation (cible).

2.2 La sélection σ

Cette opération permet d'indiquer quelles sont les n-uplets (lignes) de la relation d'entrée que l'on souhaite conserver dans le résultat.

Si l'on veut en donner une définition formelle (qui n'est pas forcément si utile en soi, l'essentiel étant de comprendre l'opération et d'être capable d'écrire et comprendre une telle expression):

$\sigma_c(\mathbf{r})$

Soit $R(V)$ schéma d'une relation $\mathbf{r}$, et C condition sur V.
$\sigma_C(\mathbf{r})$ retourne les lignes de $\mathbf{r}$ qui satisfont la condition C.

Une condition sur V est une combinaison booléenne de termes de la forme A `op` a ou A `op` B .
Formellement la grammaire des conditions est donnée par:

$C ::= A$ `op` a	(comparaison d'un attribut avec une constante)
$\mid A$ `op` B	(comparaison de deux attributs)
$\mid C_1 \wedge C_2$	(ET logique) (A,B$\in V$, a constante, op$\in \{=, \neq, <, >, \leq, \geq\}$)
$\mid C_1 \vee C_2$	(OU)
$\mid \neg C_1$	(Négation)

La sémantique de l'opération est:

$$\sigma_{A=B}(\mathbf{r}) = \{u \in \mathbf{r} \mid u_{|A} = u_{|B}\}$$
$$\sigma_{A=a}(\mathbf{r}) = \{u \in \mathbf{r} \mid u_{|A} = a\}$$

Un exemple est probablement plus simple à comprendre que la définition formelle:

Pour extraire les informations du film “Speed2” on utilisera l'expression: $\sigma_{Titre='Speed2'}(\mathbf{film})$

film

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier
Marion	M. Poirier	M. Poirier

$\sigma_{Titre='Speed2'}(\mathbf{film})$

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe

Vous noterez les guillemets (simples) autour de la chaîne de caractère constante 'Speed2': sans guillemets on comprendrait qu'on cherche à comparer les attributs Titre et Speed2, ce qui n'aurait pas de sens.

2.3 La projection π

La projection permet de sélectionner les colonnes que l'on souhaite conserver dans le résultat:

$\pi_{A_1,\ldots,A_k}(\mathbf{r})$

Retourne les colonnes $A_1, \ldots, A_k$ de $\mathbf{r}$. Plus précisément:

$$\pi_W(\mathbf{r}) = \{u_{|W} \mid u \in \mathbf{r}\}$$

Donc schéma $(A_1, \ldots, A_k)$ du résultat diffère en général de celui de $\mathbf{r}$.

⚠ Les doublons sont éliminés (sémantique *ensembliste* de l'algèbre relationnelle.).

Extraire les titres de tous les films:

film

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier
Marion	M. Poirier	M. Poirier

$\pi_{Titre}(\mathbf{film})$

Titre
Speed 2
Marion

Les opérateurs renvoient une relation; on peut donc les composer pourvu que les schémas soient compatibles:

$\sigma_{Titre \neq' Marion'}(\pi_{Titre}(\mathbf{film}))$

Titre
Speed 2

Par contre:
$\sigma_{Acteur \neq' J.Patrick'}(\pi_{Titre}(\mathbf{film}))$
est indéfini.

2.4 Produit cartésien $\times$

Cette opération permet de "mettre en regard/apparier" des éléments provenant de deux relations:

$\mathbf{r} \times \mathbf{s}$

Soient $\mathbf{r}$ et $\mathbf{s}$ deux relations de schéma respectifs $R(A_1, \ldots, A_n)$ et $S(B_1, \ldots, B_m)$.

$$\mathbf{r} \times \mathbf{s} = \{uv \mid u \in \mathbf{r} \text{ et } v \in \mathbf{s}\}$$

Schéma du résultat: concaténation des schémas de R et S (attr. de R puis S, dans l'ordre). Il faut que les schémas de $\mathbf{r}$ et $\mathbf{s}$ soient disjoints. Sinon, on renomme les attributs (ex: $A \to \mathbf{r}.A$) avant d'effectuer l'opération (d'autres conventions pour traiter les schémas non-disjoints existent).

La cardinalité du résultat est $|\mathbf{r}| \times |\mathbf{s}|$.

Exemple:

r

a	b	c
1	2	10
4	4	10

s

d	e
10	'aa'
15	'cc'

$\mathbf{r} \times \mathbf{s}$

a	b	c	d	e
1	2	10	10	'aa'
1	2	10	15	'cc'
4	4	10	10	'aa'
4	4	10	15	'cc'

Le produit cartésien est une opération coûteuse! La taille du résultat (et a fortiori le coût) est quadratique lorsque les deux relations sont de taille similaire: si 1000 tuples dans $\mathbf{r}$ et $\mathbf{s}$ alors 1M de tuples dans résultat. En pratique, le produit cartésien est assez rarement utilisé et les requêtes raisonnables utilisent presque toujours des jointures.

2.5 Jointure $\bowtie$

Par rapport au produit cartésien, il s'agit de n'apparier qu'une partie des tuples: ceux qui "vont bien ensemble", généralement pour croiser des tuples comportant des informations complémentaires sur un même objet. Assez souvent, il s'agira de croiser des informations entre deux tables liées par une clé étrangère, en utilisant cette clé étrangère (R1$\to$ R2) pour compléter les informations de la table $R1$ avec les informations complémentaires dans la table $R2$ à laquelle elle fait référence (ex: ajouter à un achat les informations de son client).

On distingue deux types de jointures:

- La *jointure naturelle* est opération binaire qui combine deux relations: produit cartésien des n-uplets avec même valeur sur attributs communs.
- La $\theta-$jointure est juste le produit cartésien auquel on rajoute une condition de sélection: on a donc plus de souplesse pour préciser les conditions de jointures: on peut donc ainsi exprimer une jointure naturelle, mais on peut aussi l'utiliser pour joindre les deux relations sur des attributs de noms différents, voire remplacer la condition d'égalité par un autre critère de comparaison.

Jointure naturelle $\mathbf{r} \bowtie \mathbf{s}$, et $\theta-$jointure $\mathbf{r} \bowtie_c \mathbf{s}$

Soient $\mathbf{r}$ et $\mathbf{s}$ deux relations de schéma respectifs $R(V)$ et $S(W)$, avec $V \cap W = A_1, \ldots, A_k$.

$$\mathbf{r} \bowtie_c \mathbf{s} = \sigma_c(\mathbf{r} \times \mathbf{s})$$

$$\mathbf{r} \bowtie \mathbf{s} = \{u \mid u_{|V} \in \mathbf{r} \text{ et } u_{|W} \in \mathbf{s}\}$$

Remarque:

$$\mathbf{r} \bowtie \mathbf{s} = \mathbf{r} \underset{\mathbf{r}.A_1=\mathbf{s}.A_1 \wedge \cdots \wedge \mathbf{r}.A_k=\mathbf{s}.A_k}{\bowtie} \mathbf{s}$$

Exemple: jointure naturelle

r

a	b	c
1	2	10
4	4	10
5	6	20
7	7	30

s

c	e
10	'aa'
10	'bb'
15	'cc'
20	'dd'

$\mathbf{r} \bowtie \mathbf{s} = \mathbf{r} \bowtie_{\mathbf{r}.c=\mathbf{s}.c} \mathbf{s}$

a	b	c	e
1	2	10	'aa'
1	2	10	'bb'
4	4	10	'aa'
4	4	10	'bb'
5	6	20	'dd'

Exemple: θ-jointure

r

c	d	e
1	2	'b'
1	2	'dd'
5	6	'dd'

s

a	b
'x'	'aa'
'y'	'dd'

$\mathbf{r} \bowtie \mathbf{s} = \mathbf{r} \bowtie_{\mathbf{r}.c=\mathbf{s}.b} \mathbf{s}$

c	d	e	a	b
1	2	'dd'	'y'	'dd'
5	6	'dd'	'y'	'dd'

La jointure peut être une opération coûteuse: dans le pire cas elle est quadratique, comme le produit cartésien. Mais la plupart du temps, les jointures se font entre des attributs liés par des contraintes d'intégrité référentielle (clés étrangères). Les jointures sont donc presque toujours beaucoup moins coûteuses que ce pire cas, mais restent suffisamment coûteuses (surtout lorsqu'il y en a beaucoup) pour constituer une grande partie du coût d'exécution des requêtes. Une des principales tâches des optimiseurs de requêtes est d'organiser le calcul d'une requête de façon à évaluer efficacement les jointures utilisées.

2.6 Renommage ρ

Le renommage permet de renommer un ou plusieurs attributs (il n'y a naturellement pas de raison de renommer les tables en algèbre relationnelle):

$\rho_{A_1 \to B_1, \ldots, A_k \to B_k}(\mathbf{r})$

$R(A_1, \ldots, A_k)$. On suppose la fonction de renommage ρ injective de $A_1, \ldots, A_k$ dans $B_1, \ldots, B_k$. Retourne une relation similaire à $\mathbf{r}$ dans laquelle chaque colonne A_i est renommée en B_i:

$$\rho_{A_1 \to B_1, \ldots, A_k \to B_k}(\mathbf{r}) = \{v \mid \exists u \in R, \forall i : u_{|A_i} = v_{|B_i}\}$$

Note: on omettra en indice les $A_i \to B_i$ tels que $A_i = B_i$.

Intérêt: donner des noms distincts à des attributs homonyme (jointure...), ou inversement permettre certaines opérations ensemblistes (union...) en donnant le même nom à des attributs distincts.

Exemple: renommage

film

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier
Marion	M. Poirier	M. Poirier

$\rho_{\text{Acteur} \to \text{Acteur_Actrice}}(\mathbf{film})$

Titre	M-en-S	Acteur_Actrice
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier
Marion	M. Poirier	M. Poirier

2.7 Opérateurs ensemblistes: Union $\cup$, Intersection $\cap$, Différence $-$

Opérations ensemblistes supportées par l'algèbre: $\cup, \cap, -, \times$. Il s'agit des opérations ensemblistes habituelles, appliquées aux ensembles de tuples puisque les relations sont des ensembles de tuples. Il est essentiel que les relations aient le même schéma pour toutes ces opérations ensemblistes, sinon l'opération n'est pas définie. Le schéma du résultat est naturellement le même que celui des relations en entrée.

r ∪ s

Soient R et S deux relations *de même schéma*.

$$\mathbf{r} \cup \mathbf{s} = \{u \mid u \in \mathbf{r} \cup \mathbf{s}\}$$

$$\mathbf{r} \cap \mathbf{s} = \{u \mid u \in \mathbf{r} \cap \mathbf{s}\}$$

$$\mathbf{r} - \mathbf{s} = \{u \in \mathbf{r} \mid u \notin \mathbf{s}\}$$

⚠ Les doublons sont éliminés (dans le cas de l'union, puisque les autres ne génèrent pas de doublons)!

r

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard

s

Titre	M-en-S	Acteur
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier
Marion	M. Poirier	M. Poirier

r ∪ s

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier
Marion	M. Poirier	M. Poirier

r ∩ s

Marion	M. Poirier	C. Tetard

r − s

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe

Pour aller plus loin:

- Un autre opérateur de l'algèbre relationnelle que l'on rencontre parfois: la division, n'a pas été abordée dans ce poly car elle est plus complexe, peut être exprimées par les autres opérateurs, et n'est pas particulièrement utile pour analyserles manipulation de données effectuées par les langages manipulant des tables comme SQL ou la librairie Pandas.
- On étend souvent l'algèbre relationnelle avec les opérateurs de jointures externes, et plus rarement avec des opérations d'agrégation. Comme ces opérations ne font pas parties de l'algèbre "de base", nous nous contenterons de les étudier en SQL.
- Nous avons présenté l'algèbre relationnelle, en raison de son importance pour l'analyse, évaluation et optimisation des requêtes. Toutefois, la théorie des bases de données s'appuie plus souvent sur un autre formalisme, le *calcul relationnel*, pour étudier les propriétés de requêtes. Le calcul relationnel est grosso modo une adaptation de la logique du premier ordre aux données relationnelles. Il est particulièrement adapté pour étudier l'expressivité d'un langage de requête (car il permet en particulier d'appliquer aux bases de données les techniques et théorèmes développés dans les domaines de la logique et théorie des jeux).
- Nous n'étudierons pas cette année comment concevoir un schéma dans les règles de l'art. L'idée générale est simple: il s'agit d'éviter les informations redondantes. Éliminer les informations redondantes permet de gagner de la place (un peu), mais surtout d'éviter que des mises à jours incomplètes n'aboutissent à créer des données incohérentes. De façon plus générale, éviter les redondances est un principe courant en informatique: quand on écrit un programme on essaie aussi d'éviter la redondance en factorisant le code, et pour les mêmes raisons à savoir avant tout faciliter la maintenance du code, et accessoirement faciliter l'optimisation. Exemple: au lieu d'une table `Client(nom_client, ville_client, nbre_habitant_ville)`, on va plutôt créer deux tables: `Client(nom_client, ville_client)` et `Ville(nom_ville, nbre_habitant_ville)`. Le nouveau schéma contient les mêmes informations, car on peut retrouver les informations originales en joignant les deux tables sur le nom de la ville. Mais si la base de données comporte plusieurs clients Orcéens, en séparant ainsi les données en deux relations, on élimine le risque qu'un utilisateur mette à jour le nombre d'habitant de la ville 'Orsay' sur un seul client en oubliant les autres Orcéens. Ce principe se généralise et formalise avec la *théorie de la normalisation*, et en particulier les *dépendances fonctionnelles*.

Quelques références:

- `http://sql.bdpedia.fr/alg.html` un cours du CNAM. Contient diapos, vidéos, quizzs. (P. Rigaux)
- Les ressources pour lycéens sur Eduscol, et les manuels de NSI de terminale (ex: manuel Hachette, chap4).
- `http://abiteboul.com/Lili/bdrelationnelles.pdf` un poly de cours conçu pour les prépas. Vous trouverez quelques vidéos (que je ne recommande pas plus que ça, question de goût) en ligne car le cours est aussi conçu comme un MOOC `https://flot.sillages.info/?portfolio=bases-de-donnees-relationnelles` (S. Abiteboul, B. Nguyen, Y. Le Bras). S. Abiteboul est membre de l'académie des sciences, et est connu entre autre pour avoir écrit un livre sur les fondements logiques des bases de données.
- `https://en.wikipedia.org/wiki/Relational_algebra` la page anglaise est plutôt plus complète que la française.
- `https://openclassrooms.com/fr/courses/4449026-initiez-vous-a-lalgebre-relationnelle-avec-le-langage-sql`