Fondements des Bases de Données

Frédéric Flouvat

Université de la Nouvelle-Calédonie

frederic.flouvat@univ-nc.nc

Présentation de l'UE

- Volume horaire :
  - 14h de cours (7 séances), 16h de TD (8 séances) et 10h de TP (5 séances)
  - répartition cours/TD en fonction de l'avancement
  - TP pour appliquer une fois les notions abordées en cours et TD

- Evaluation :
  - 4 contrôles continus (4 CC, 25% par CC)
    - Deux évaluations "papiers" (2h, documents autorisés)
    - Deux devoirs notés (TP à finir à la maison)
  - 2ème chance: plus mauvaise note de CC enlevée

- Objectif : interrogations avancées et normalisation des bases de données
  - approfondir les connaissances pratiques (contraintes, programmation procédurale, vues, …) et théoriques (algèbre relationnel, dépendances et normalisation)

Plan du cours

- Chapitre 1: Manipuler les données
  - Rappels: modèle relationnel, algèbre relationnel et SQL
  - Programmation procédurale
  - Contrôler les transactions

- Chapitre 2: Définir et structurer les bases de données
  - Contraindre les données
  - Définir des vues des données
  - Normaliser leur structure

UNIVERSITÉ de la NOUVELLE-CALÉDONIE

Chapitre 1: Manipuler les données

Rappels

Frédéric Flouvat

(en partie dérivé du cours du Pr. Jeffrey Ullman, Stanford University)

Université de la Nouvelle-Calédonie

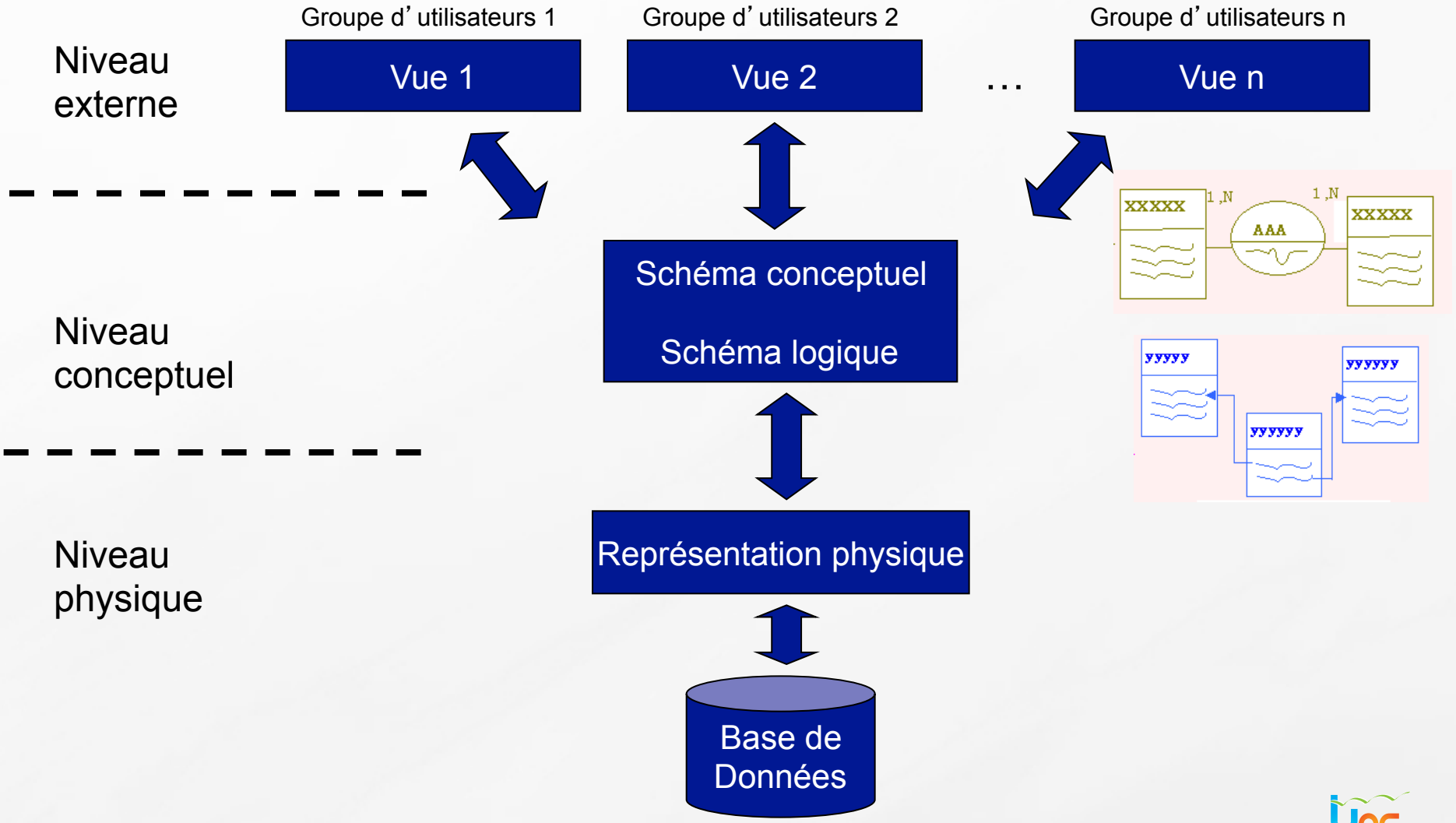
frederic.flouvat@univ-nc.nc

Généralités

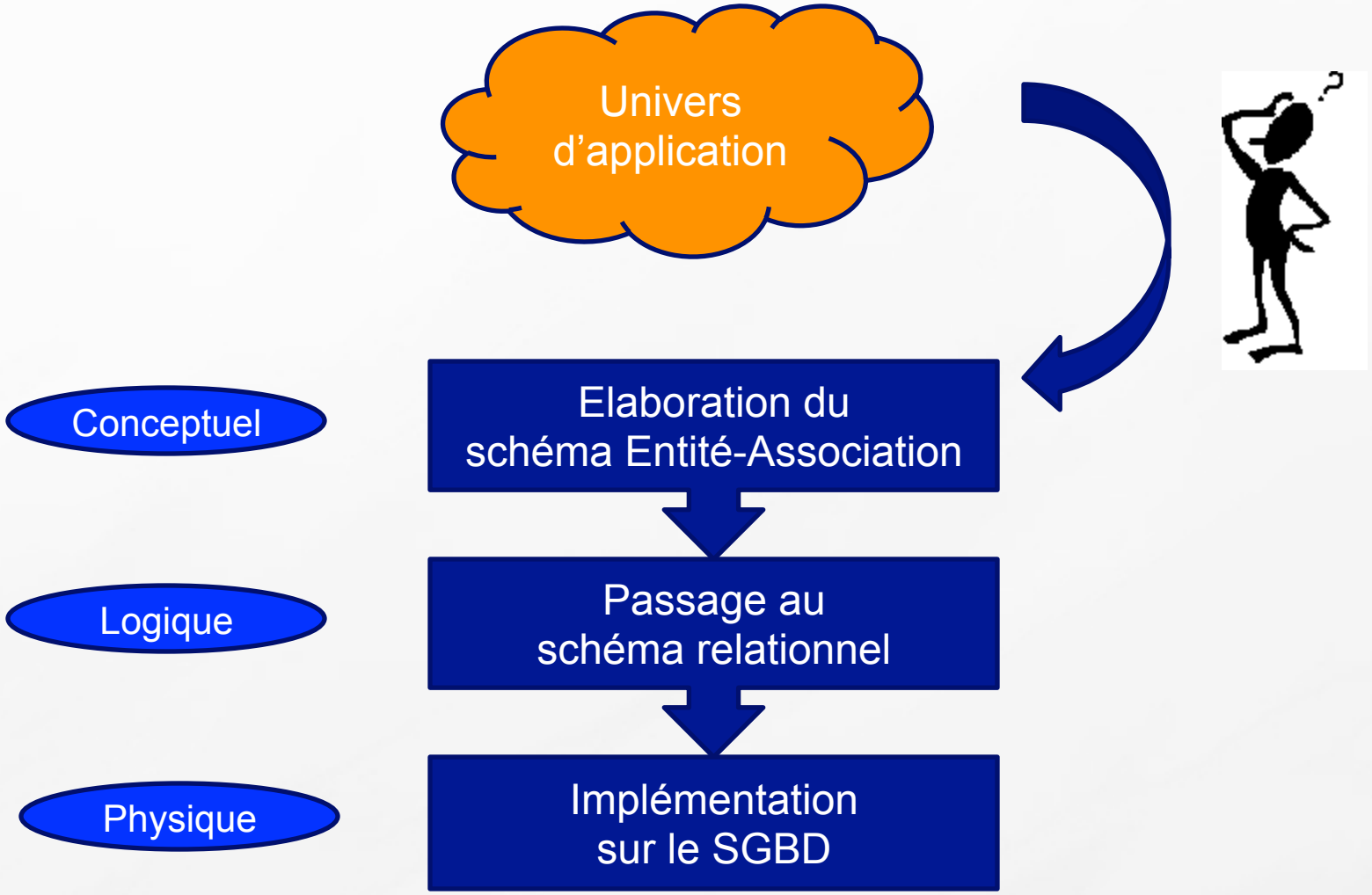
- Une ***Base de Données*** : ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation.

- Un ***Système de Gestion de Bases de Données*** : comprend un ensemble de données liées (la bd) et un ensemble de programmes permettant la définition, la saisie, le stockage, le traitement, la modification, la diffusion et la protection de ces données.
  - Liens entre les données
  - Cohérence des données
  - Souplesse d'accès aux données
  - Sécurité
  - Partage des données
  - Indépendance des données
  - Performances

Architecture ANSI/SPARC



La démarche de conception



Modèle relationnel

- Dans le modèle relationnel, les données sont structurées en ***relation***, d'ordinaire représentées sous forme tabulaire.
  - Ex :

Personnes	nss	nom	prenom	age
	12	Ijo	John	45
	45	Mayer	Solange	35

- Une relation est définie sur un ***schéma de relation***.
- Chaque ligne est appelée un ***tuple***.
- Les noms des colonnes sont appelés ***attributs***.
  - Ex : La relation Personnes est définie sur le schéma PERSONNES qui compte 4 attributs : schema(PERSONNES)={nss, nom, prenom, age}
    t = <12, Ijo, John, 45> est un tuple de cette relation (t∈Personnes)

Modèle relationnel

- Une ***base de données*** est un ensemble de relations r_i sur R_i.
- Un ***schéma de base de données*** est un ensemble $\{R_1, \ldots, R_n\}$ de schémas de relations.
  - Ex : Soit d={Personnes, Departements, Activites} une base de données.

Personnes	nss	nom	prenom	age
	12	Ijo	John	45
	45	Mayer	Solange	35

Departements	dep	adresse
	Sciences	Nouville
	Lettres	Magenta

Activites	#nss	#dep	fonction
	12	Sciences	Prof
	45	Sciences	Vacataire
	45	Lettres	MdC
	12	Eco	Vacataire

Définitions

- Soit A un attribut. On note DOM(A), le ***domaine de A*** i.e. l'ensemble des valeurs que peux prendre l'attribut A.
  - Ex : DOM(dep) = {Sciences, Lettres, Eco, Droit, SHS …}
  - Ex : DOM(age) = ensemble des entiers naturels

- Le ***domaine actif de A dans r*** noté ADOM(A,r), est l'ensemble des valeurs constantes prises par A dans r.
  - Ex : ADOM(dep,Activites) = {Sciences, Lettres, Eco}

- La ***projection*** d'un tuple t sur l'attribut A (resp. {A,B}) est notée t[A] (resp t[A,B]).
  - Ex : Soit t = <12, Ijo, John, 45>
  - t[nom] = <Ijo> et t[nom, prenom] = <Ijo, John>

Différents types de langages

- LDD : Langage de Définition de Données
  - Schéma de la bd et des vues
- LMD : Langage de Manipulation de Données
  - Requêtes, mises à jour
- LCD : Langage de Contrôle des Données
  - Gestion des accès utilisateurs
- LCT : Langage de Contrôle des Transactions
  - Gestion des transactions

Langages de requêtes relationnels

- Les langages de requêtes sont généralement ***déclaratifs*** i.e. on spécifie ce que la sortie doit contenir et non comment l'obtenir.

- Les SGBD relationnels fonctionnent en amont avec des langages ***procéduraux*** qui spécifient comment obtenir les résultats aux requêtes.

  - Déclaratif : {nss ∈ DOM(Activites) | (nss,dep,fonction)∈Activites, (dep,adresse)∈Departements, adresse= 'Carnot'}
  - Procédural :

```
for each tuple t1=<n,d,f> in relation Activites do
        for each tuple t2=<d',a> in relation Departements do
                if d=d' and a= 'Carnot' then output n
        end
end
```

Langages de requêtes relationnels

- Le modèle relationnel supporte des langages de requêtes simples et puissants qui permettent beaucoup d'optimisation.

- Langages théoriques
  - ***Algèbre relationnelle*** : Langage procédural très utile pour représenter les plans d'exécution des requêtes.
  - ***Calcul relationnel*** : Langage déclaratif orienté utilisateur.
  - ***Datalog*** : Langage déclaratif à base de règles. Augmente le calcul relationnel avec des capacités d'inférence.

- Langages commerciaux
  - ***SQL*** : Les opérateurs s'inspirent des différents langages théoriques.

Syntaxe générale d'une requête SQL

```
 SELECT    <liste des attributs à projeter ou *>
 FROM      <liste des tables>
[WHERE     <critères de restriction>]
[GROUP BY  <liste des attributs d'agrégation>]
[HAVING    <critères de restriction sur les agrégats>]
[ORDER BY  <liste des attributs de tri>]
```

```
 SELECT    <liste des attributs à projeter ou *>
 FROM      <liste des tables>
[CONNECT BY PRIOR <critères de récursivité>]
[START WITH <condition de départ>]
```

Principes des langages de requêtes relationnels

- Elle est constituée d'un ensemble d'opérateurs algébriques :
  - π, σ, ×, -, ∪
  - Entrée : une ou deux relations (opérateurs unaires ou binaires)
  - Sortie : une relation

  - Requête relationnelle : Composition d'un nombre fini d'opérateurs algébriques. Le résultat d'une requête est une relation.

- L'ordre d'évaluation des opérateurs est spécifié dans la requête (requête procédurale).

- L'algèbre relationnelle manipule des ensembles.

Intérêts

- A la base de l'***optimisation de requêtes*** :
  - Les SGBDR traduisent les requêtes formulées avec SQL en une requête interne construite avec les opérateurs algébriques, requête qui sera par la suite optimisée pour en générer un plan d'exécution équivalent.

- Tous les LMD sont construits à partir de ces opérateurs de base.
  - + généralement mise à jour de la bd et emploi d'expressions arithmétiques et de fonctions d'agrégation telles que cardinalité, somme, minimum, maximum et moyenne.

Les opérateurs

- Opérateurs de base
  - Projection (π)
  - Sélection (σ)
  - Produit cartésien ($\times$)
  - Différence ($-$)
  - Union ($\cup$)

- Opérateurs supplémentaires (non essentiels, mais très utiles)
  - Intersection ($\cap$)
  - Jointure ($\bowtie$)
  - Division ($\div$)
  - Renommage ($\rho_{A \to B}$)

Projection

- Soient r une relation sur R et Y ⊆ schema(R).
- La ***projection*** de r sur Y notée $\pi_Y(r)$, est définie par :

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

- Soit S le schéma de relation associé à $\pi_Y(r)$. On a schema(S)=Y.
  - Ex : $\pi_{nom}(Personnes)$ =

nom
Ijo
Mayer

$\pi_{nom,prenom}(Personnes)$ =

nom	prenom
Ijo	John
Mayer	Solange

$\pi_{dep}(Activites)$ =

dep
Sciences
Lettres
Eco

Sélection

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

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

- Soit S le schéma de relation associé à $\sigma_F(r)$. On a schema(S) = schema(R).
  - Ex : $\sigma_{\text{fonction= 'Prof'}}(\text{Activites})$ =

nss	dep	fonction
12	Sciences	Prof

Sélection

- Une ***formule de sélection simple*** sur R est une expression de la forme : A = a ou A = B, où A, B ∈ schema(R) et a ∈ Dom(A).

- Une ***formule de sélection*** est une expression composée de formules de sélection simples connectées à l'aide des connecteurs logiques ∧, ∨, ¬ et des parenthèses.

- Soit r une relation sur R, t∈r et F une formule de sélection.
  t satisfait F, noté t ⊨ F, est défini récursivement par :
  1) t ⊨ A = a si t[A] = a
  2) t ⊨ A = B si t[A] = t[B]
  3) t ⊨ $F_1 \wedge F_2$ si t ⊨ F_1 et t ⊨ F_2
  4) t ⊨ $F_1 \vee F_2$ si t ⊨ F_1 ou t ⊨ F_2
  5) t ⊨ ¬F si t ⊭ F
  6) t ⊨ (F) si t ⊨ F

Opérateurs ensemblistes

- Soient r_1 et r_2 deux relations sur R_1 et R_2.
- ***Union*** : $r_1 \cup r_2 = \{ t \mid t \in r_1 \text{ ou } t \in r_2 \}$.
- ***Différence*** : $r_1 - r_2 = \{ t \mid t \in r_1 \text{ et } t \notin r_2 \}$.
- ***Intersection*** : $r_1 \cap r_2 = \{ t \mid t \in r_1 \text{ et } t \in r_2 \}$.
  - Ex : $r_1 \cup r_2 =$ $r_1 - r_2 =$ $r_1 \cap r_2 =$

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

A	B	C
1	2	3
1	2	2

A	B	C
1	1	1

r1	A	B	C
	1	2	3
	1	1	1
	1	2	2

r2	A	B	C
	2	2	2
	1	1	1

Opérateurs ensemblistes

- Propriétés :
  - $\sigma_{\neg F}(r) = r - \sigma_F(r)$
  - $\sigma_{F1 \vee F2}(r) = \sigma_{F1}(r) \cup \sigma_{F2}(r)$
  - $\sigma_{F1 \wedge F2}(r) = \sigma_{F1}(r) \cap \sigma_{F2}(r)$

- Toutes ces opérations prennent comme entrées deux relations qui doivent être ***compatibles vis-à-vis de l'union.***

- Soit S le schéma de relation associé à $r_1 \cup r_2$, $r_1 - r_2$ ou $r_1 \cap r_2$. On a par convention schema(S) = schema(R_1).

Produit cartésien

- Soient r_1 et r_2 deux relations sur R_1 et R_2 avec schema(R_1)∩schema(R_2) = ø.

- Le ***produit cartésien*** $r_1 \times r_2$ est définie par :

$$r_1 \times r_2 = \{ t \mid \exists t_1 \in r_1 \text{ et } \exists t_2 \in r_2 \text{ tel que } t[\text{schema}(R_1)] = t_1 \text{ et } t[\text{schema}(R_2)] = t_2 \}$$

- Soit S le schéma de relation associé à $r_1 \times r_2$. On a schema(S) = schema(R1)∪schema(R_2).

Produit cartésien

r1	A	B	C
	1	2	3
	1	1	1
	1	2	2

r2	D	E
	2	4
	1	3

- Ex : $r_1 \times r_2 =$

A	B	C	D	E
1	2	3	2	4
1	1	1	2	4
1	2	2	2	4
1	2	3	1	3
1	1	1	1	3
1	2	2	1	3

Jointure

- Soient r_1 et r_2 deux relations sur R_1 et R_2 et F une formule de sélection.
- La ***jointure*** de r_1 et r_2 par rapport à F, notée $r_1 \bowtie_F r_2$, est définie par :

$$r_1 \bowtie_F r_2 = \sigma_F(r_1 \times r_2)$$

- Soit S le schéma de relation associé à $r_1 \bowtie_F r_2$. On a schema(S) = schema(R1)∪schema(R_2).

  - ***Équi-jointure*** quand F est une égalité.
  - ***Thêta-jointure*** quand F n'est pas une égalité (>,<,>=,<=,<>).
  - ***Auto-jointure*** quand r1 = r2.
  - ***Jointure naturelle*** quand équi-jointure sur tous les attributs communs à R1 et R2 + projection (pas utile d'écrire F).

Jointure

r1	A	B	C
	1	2	3
	1	1	1
	1	2	2

r2	D	E
	2	4
	1	3

- Ex : $r_1 \times r_2$ =

A	B	C	D	E
1	2	3	2	4
1	1	1	2	4
1	2	2	2	4
1	2	3	1	3
1	1	1	1	3
1	2	2	1	3

$r_1 \bowtie_{A=D} r_2$ =

A	B	C	D	E
1	2	3	1	3
1	1	1	1	3
1	2	2	1	3

$r_1 \bowtie_{A<D} r_2$ =

A	B	C	D	E
1	2	3	2	4
1	1	1	2	4
1	2	2	2	4

Jointure naturelle

r1	A	B	C
	2	2	2
	1	1	1

r2	A	D	E
	1	2	3
	1	1	2
	2	3	1
	3	1	2

r3	A	B	C
	1	1	1
	1	2	3

r4	D	E	F
	1	1	1
	2	2	2

- Ex : $r_1 \bowtie r_2 =$

A	B	C	D	E
2	2	2	3	1
1	1	1	2	3
1	1	1	1	2

$r_1 \bowtie r_3 = r_1 \cap r_3$

A	B	C
1	1	1

$r_1 \bowtie r_4 = r_1 \times r_4$

A	B	C	D	E	F
2	2	2	1	1	1
2	2	2	2	2	2
1	1	1	1	1	1
1	1	1	2	2	2

Renommage

- Soient r une relation sur R, A ∈ schema(R) et B ∉ schema(R).
- Le ***renommage*** de A en B dans r noté $\rho_{A \to B}(r)$, est défini par :

$$\rho_{A \to B}(r) = \{ t \mid \exists u \in r,\ t[schema(R) - \{B\}] = u[schema(R) - \{A\}] \text{ et } t[B] = u[A] \}$$

- Soit S le schéma de relation associé à $\rho_{A \to B}(r)$. On a schema(S) = (schema(R)-{A})∪{B}.
  - Ex : $r_1 \bowtie \rho_{B \to B', C \to C'}(r_3)$ =

A	B	C	B'	C'
1	1	1	1	1
1	1	1	2	3

Division

- La division permet d'exprimer le ***quantificateur universel*** (∀).
- Exemple de requête : "Donner la liste des étudiants qui sont inscrits à ***tous*** les cours".

Inscriptions	etud	cours
	1	BD
	1	RO
	1	BI
	2	RO

Enseignements	cours
	BD
	RO
	BI

Division

- Soient r une relation sur R, avec schema(R) = XY, et s une relation sur S, avec schema(S) = Y.

- La ***division*** de r par s notée r ÷ s, est défini par :

$$r \div s = \{ t[X] \mid t \in r \text{ et } s \subseteq \pi_Y(\sigma_F(r)) \}$$

$$\text{avec } X=\{A_1,\ldots, A_q\} \text{ et } F =(A_1=t[A_1]) \wedge \ldots \wedge (A_q=t[A_q])$$

- Soit schema(T) le schéma de relation associé à r ÷ s.
  On a schema(T) = X.

Division

- La division n'est pas une opération essentielle car elle peut être exprimée à l'aide des opérateurs de projection, différence et produit cartésien.

- Pour calculer r ÷ s, il faut calculer toutes les valeurs de X dans r qui ne sont pas disqualifiées par des valeurs de Y dans s.

- Une valeur de X est ***disqualifiée*** si en lui rattachant une valeur de Y de s, le tuple obtenu sur XY n'appartient pas à r.
  - Valeurs disqualifiées de X : $\pi_x((\pi_x(r) \times s) - r)$
  - $r \div s = \pi_x(r)$ - valeurs disqualifiées de X
  - ➢ $r \div s = \pi_x(r) - \pi_x((\pi_x(r) \times s) - r)$

Division

Mod1	cours
	c2

F3	etud
	2
	4

Mod2	cours
	c1
	c3
	c4

UNC	etud	cours
	1	c1
	1	c2
	1	c3
	1	c4
	2	c1
	2	c2
	3	c2
	4	c2
	4	c4

- Ex : UNC ÷ Mod1 =

etud
1
2
3
4

UNC ÷ F3 =

cours
c2

UNC ÷ Mod2 =

etud
1

Propriétés

- $\sigma_F(\sigma_{F'}(r)) = \sigma_{F'}(\sigma_F(r))$
- $\pi_X(\pi_Y(r)) = \pi_{X \cap Y}(r)$
- $\sigma_F(\pi_X(r)) = \pi_X(\sigma_F(r))$
- $\sigma_F(r \bowtie s) = \sigma_F(r) \bowtie s$ si F porte sur schema(R)
- $\pi_X(r \bowtie s) = \pi_X(r) \bowtie \pi_X(s)$ si schema(R) $\cap$ schema(S) = X
- $(r1 \cup r2) \bowtie r3 = (r1 \bowtie r3) \cup (r2 \bowtie r3)$

Ex : Si schema(R1)=XY et schema(R2) =XZ, exprimer $r_1 \bowtie r_2$ avec $\times$, ρ, σ, π

$$r_1 \bowtie r_2 = \pi_{XYZ}(\sigma_{X=X'}(r_1 \times \rho_{X \rightarrow X'}(r_2)))$$

Ex : Si schema(R1)=schema(R2), exprimer $r_1 \cap r_2$ avec -

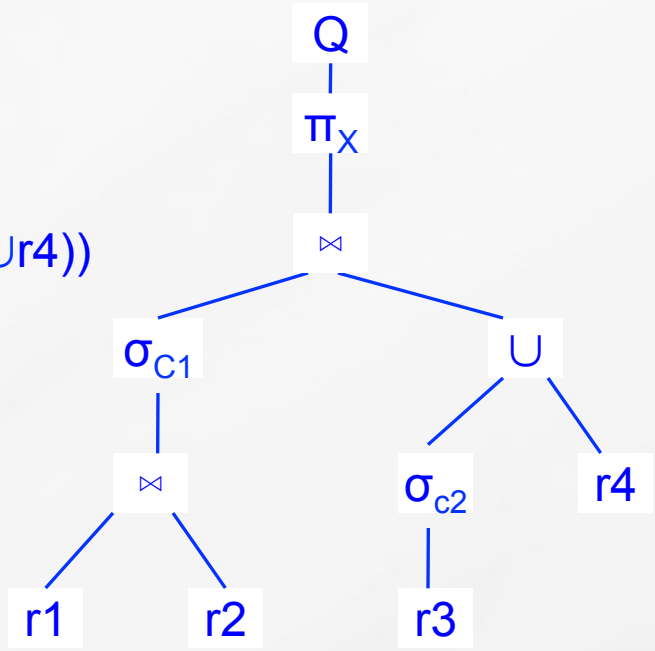
$$r_1 \cap r_2 = r_1 - (r_1 - r_2)$$

Représentation en arbre

- Une expression algébrique peut se représenter sous forme d'arbre :
  - La racine de l'arbre correspond à la requête
  - Les feuilles de l'arbre correspondent aux relations
  - Les nœuds de l'arbre correspondent aux opérateurs algébriques

- Ex : Représenter

$Q = \pi_X(\sigma_{C1}(r1 \bowtie r2) \bowtie (\sigma_{c2}(r3) \cup r4))$

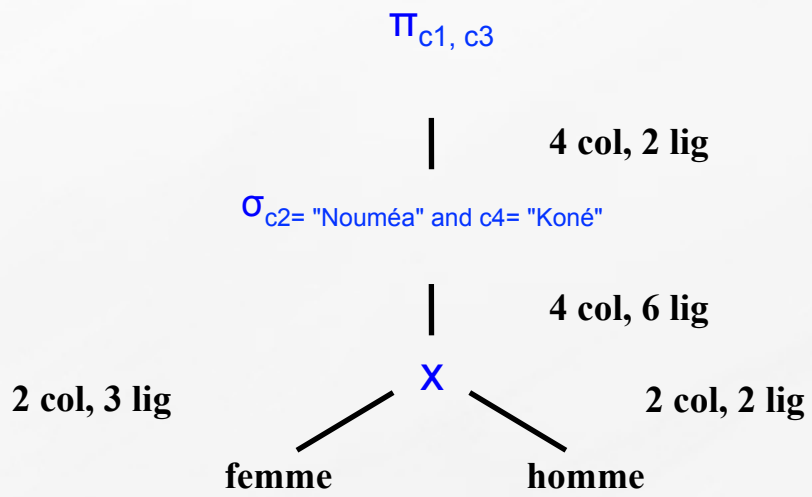


Arbres et optimisation de requêtes

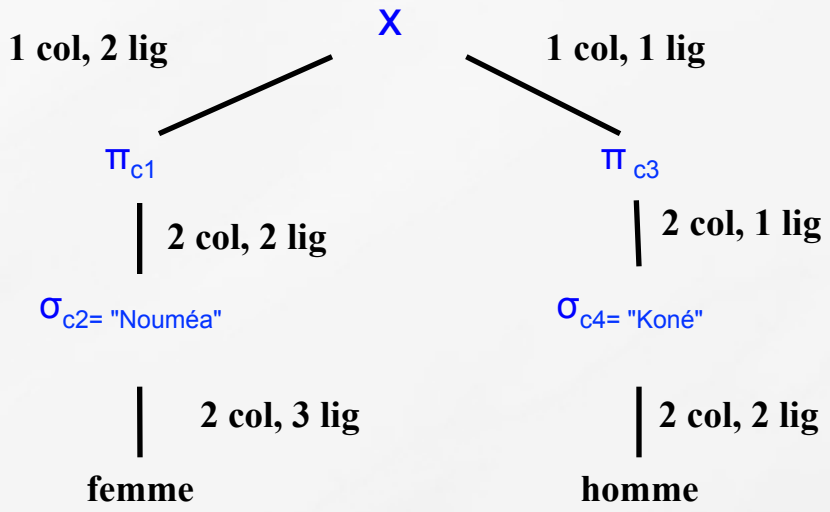
Femme	c1 (nom)	c2 (adr)
	Cathy	Wé
	Julia	Nouméa
	Linda	Nouméa

Homme	c3 (nom)	c4 (adr)
	Bob	Nouméa
	Sam	Koné

- Quels sont les couples possibles avec une femme de Nouméa et un homme de Koné ?



42 cases manipulées



19 cases manipulées

Extension de l'algèbre relationnelle

Quatre nouveaux opérateurs:

- **δ** (delta) : élimine de la relation résultat les tuples en double.
  - ➢ **Extension de la théorie aux multi-ensembles**
- **τ** (tau) : trie les tuples.
- **γ** (gamma) : groupe et agrège.
- **Outerjoin** : évite les "tuples incomplets" = les tuples qui ne sont liés/ joins à aucun autre tuple.

Elimination des doublons

- R1 := δ(R2).
- R1 est une copie de chaque tuple qui apparaissent dans R2 une ou plusieurs fois (une copie sans doublons donc).
- Exemple:

R =

A	B
1	2
3	4
1	2

δ(R) =

A	B
1	2
3	4

Trier

- R1 := τ_L (R2).
  - *L* est une liste d'attributs de R2.
- R1 est la liste des tuples de R2 triés par rapport au premier attribut de *L*, puis par rapport au deuxième attribut, …
  - si plusieurs solutions sont possibles, en prend une arbitrairement
- τ est le seul opérateur n'ayant pas pour résultat un ensemble ou un multi-ensemble.
  - le résultat est une séquence de tuples

- Exemple:

R =

(A	B)
1	2
3	4
5	2

τ_B (R) = [(5,2), (1,2), (3,4)]

Opérateurs d'agrégation

- Les opérateurs d'agrégation ne sont pas des opérateurs de l'algèbre relationnelle.
- Ils sont appliqués sur une colonne d'une table et ont pour résultat une seule valeur.
- Exemple d'opérateurs: SUM, AVG, COUNT, MIN, et MAX.

- Exemple:

R = (A | B)

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
AVG(B) = 3

Opérateur de regroupement

R1 := γ_L (R2). *L* est une liste composées d'éléments de type:

1. attributs (*regroupement*).
2. AGG(*A*), où AGG est un des opérateurs d'agrégation et A est un attribut.
   - Une flèche et un nouveau nom d'attribut renomme la composante.

Fonctionnement de γ_L (R)

- Groupe R par rapport à tous les attributs de regroupement de *L*
  - forme un groupe pour chaque liste de valeur distincte pour les attributs de R
- Pour chaque groupe, calcule AGG(A)
- Chaque groupe est associé à un tuple dans la relation résultat
  - avec pour attributs, les attributs de regroupement
  - et les résultats des opérations d'agrégation

Opérateur de regroupement

- Exemple:

R = (

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

)

$\gamma_{A,B,AVG(C)->X}(R)$ = ??

Premièrement, regrouper *R* par rapport *A et B* :

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

Puis, calculer la moyenne de *C* pour chaque groupe:

A	B	X
1	2	4
4	5	6

Jointure externe (*Outerjoin*)

- Supposons que l'on fasse $R \bowtie_C S$.
- Un tuple de *R* qui n'est associé à aucun tuple de *S* par jointure est dit *incomplet*.
  - De la même manière pour un tuple de *S*.
- La jointure externe conserve les tuples incomplets dans la solution en leur associant la valeur NULL.

Jointure externe (*Outerjoin*)

- Exemple:

R =

A	B
1	2
4	5

S =

B	C
2	3
6	7

(1,2) est associé par jointure à (2,3), mais les autres tuples sont incomplets

R *OUTERJOIN* S =

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

MAINTENANT --- RETOUR AU SQL

Chaque opération a un équivalent en SQL

Jointures externe

- R OUTER JOIN S est l'instruction principale pour faire une jointure externe. Elle est complétée par:
  1. NATURAL devant OUTER (Optionnel).
  2. ON <condition> après JOIN (Optionnel).
  3. LEFT, RIGHT, ou FULL devant OUTER (Optionnel).
     - LEFT = conserve les tuples incomplets de R seulement.
     - RIGHT = conserve les tuples incomplets de S seulement.
     - FULL = conserve les tuples incomplets des deux; valeur par défaut.

Uniquement une des deux

- Exemple:

A partir de Sells(bar, beer, price) et Frequents(drinker ,bar), afficher tous les clients et les bières qu'ils peuvent consommer

```
SELECT drinker, beer
FROM Sell RIGHT OUTER JOIN Frequents
ON Sells.bar = Frequents.bar ;
```

Agrégations

- SUM, AVG, COUNT, MIN, et MAX peuvent être appliqués à une colonne dans une clause SELECT afin d'appliquer l'agrégation sur cette colonne.
- Egalement, COUNT(*) pour compter le nombre de tuples.

- Exemple:

A partir de Sells(bar, beer, price), trouver le prix moyen d'une Bud:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud';
```

Eliminer les doublons dans une Agrégation

- Utiliser DISTINCT à l'intérieur de l'agrégation.

- Exemple:

Trouver le nombre de prix différents associés à la Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud';
```

Les NULL ignorés dans les Agrégations

- Les valeurs NULL ne contribuent jamais à une somme, une moyenne, ou un comptage, et ne peuvent jamais être le minimum ou le maximum d'une colonne.
- Mais s'il n'y a pas de valeurs non-NULL dans une colonne, alors le résultat de l'agrégation est NULL.
  - Exception: COUNT d'un ensemble vide retourne 0.
- Exemple:

```
SELECT count(*)
FROM Sells
WHERE beer = 'Bud';
```

Le nombre de bars qui vendent de la Bud

```
SELECT count(price)
FROM Sells
WHERE beer = 'Bud';
```

Le nombre de bars qui vendent de la Bud à un prix connu

Groupement

- Après l'expression SELECT-FROM-WHERE, ajouter GROUP BY et une liste d'attributs.
- La relation qui résulte du SELECT-FROM-WHERE est groupées en accord avec les valeurs de tout ces attributs, et un opérateur d'agrégation est appliqué sur chaque groupe.

- Exemple:

  A partir de Sells(bar, beer, price), trouver ler prix moyen de chaque bière:

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer;
```

beer	AVG(price)
Bud	2.33
…	…

Groupement

- Exemple:

  A partir de Sells(bar, beer, price) et Frequents(drinker,bar), trouver pour chaque client le prix moyen de la Bud au bar qu'ils fréquentent:

```
SELECT drinker, AVG(price)
FROM Frequent,Sells
WHERE beer = 'Bud'
      AND Frequents.bar = Sells.bar
GROUP BY drinker;
```

Construit tous les tuples drinker-bar-price associés à la Bud

Puis, les regroupent par clients

Restriction sur le SELECT avec une Agrégation

- Si une agrégation est effectuée, alors chacun des éléments du SELECT doit être soit:
  1. Une opération d'agrégation, ou
  2. Un attribut utilisé au niveau d'un GROUP BY

- Exemple de requête interdite:
  - Il semble correcte de rechercher le bar qui vend les bières Bud les moins chères par la requête:

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = 'Bud';
```

  - Mais cette requête est interdite en SQL.

Clauses HAVING

- HAVING <condition> peut suivre une clause GROUP B.
- Si tel est le cas, la condition est appliquée à chaque groupe, et les groupes ne satisfaisant pas la condition sont ignorés.

- Exemple:

A partir de Sells(bar, beer, price) et Beers(name, manf), trouver le prix moyen des bières qui sont soient servies dans au moins trois bars ou sont fabriquées par Pete.

???

Clauses HAVING

- Exemple:

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3 OR
    beer IN (SELECT name
             FROM Beers
             WHERE manf = 'Pete''s');
```

Les groupes de bières avec au moins 3 bars non-NULL et aussi les groupes de bières avec pour fabricant Pete

Les bières fabriquées par Pete

Restrictions sur les conditions du HAVING

- Tout doit être dans une sous-requête.

- A part les sous-requêtes, les conditions peuvent être sur des attributs uniquement si:
  1. ce sont les attributs utilisés pour le regroupement, ou
  2. s'ils sont agrégés

  (même condition que pour la clause SELECT lorsqu'elle est utilisée pour une agrégation).

Modifications des relations d'une base de données

- Une commande de *modification* ne retourne pas de résultat (comme le fait une requête), mais change le contenu des relations.

- Trois types de modifications:
  1. *Insert* : insertion d'un ou plusieurs tuples.
  2. *Delete* : suppression d'un ou plusieurs tuples.
  3. *Update* : mise à jours des valeurs d'un ou plusieurs tuples existants.

Insertion

- Pour insérer un unique tuple:

  INSERT INTO <relation>
  VALUES (<list of values>);

Attention: les valeurs doivent être dans le même ordre que les attributs de la relation et les types doivent correspondre.

- Exemple:

  Ajouter à Likes(drinker, beer) le fait que Sally aime les bières Bud.

```
INSERT INTO Likes
VALUES('Sally', 'Bud');
```

Spécifier les attributs de l'INSERT

- Il est possible d'ajouter au nom de la relation une liste d'attributs
- Deux raisons de faire ainsi:
  1. l'ordre des attributs de la relation a été oublié.
  2. il n'y a pas de valeurs pour tous les attributs, et nous voulons que le SGBD remplace ces valeurs manquantes par NULL ou par une valeur par défaut.

- Exemple:

Une autre approche pour ajouter à Likes(drinker, beer) le fait que Sally aime les Bud :

```
INSERT INTO Likes(beer, drinker)
VALUES('Bud', 'Sally');
```

Ajouter des valeurs par défaut

- Dans la commande CREATE TABLE permettant de créer une table, il est possible de faire suivre la définition d'un attribut par DEFAULT et une valeur.
- Lorsqu'un tuple à insérer n'a pas de valeur pour cet attribut, la valeur par défaut est utilisée.

- Exemple:

```
CREATE TABLE Drinkers (
        name CHAR(30) PRIMARY KEY,
        addr CHAR(50)
              DEFAULT '123 Sesame St.',
        phone CHAR(16)
);
```

Ajouter des valeurs par défaut

- Exemple :

```
INSERT INTO Drinkers(name)
VALUES('Sally');
```

- relation résultat:

name	address	phone
Sally	123 Sesame St	NULL

Insertion de plusieurs tuples

- Il est possible d'insérer la totalité du résultat d'une requête dans une relation, en faisant:

  INSERT INTO <relation>
  (<subquery>);

- Exemple :

  En utilisant la relation Frequents(drinker, bar), insérer dans une nouvelle relation PotBuddies(name) contenant tous les amis potentiels de Sally, i.e. les clients qui fréquente au moins un des bars que fréquente Sally.

Insertion de plusieurs tuples

- Exemple :

```
INSERT INTO PotBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = 'Sally' AND
    d2.drinker <> 'Sally' AND
    d1.bar = d2.bar
);
```

les autres clients

les pairs de tuples de clients où le premier est Sally et le second est quelqu'un d'autre, et les bars sont les mêmes

Suppression de tuples

- Pour supprimer les tuples d'une relation en fonction de conditions:

  DELETE FROM <relation>
  WHERE <condition>;

- Exemple:

  Supprimer de Likes(drinker, beer) le fait que Sally aime la Bud:

```
DELETE FROM Likes
WHERE drinker = 'Sally' AND
      beer = 'Bud';
```

- Supprimer tous les tuples d'une relation:

```
DELETE FROM Likes;
```

  - Remarque: pas de clause WHERE.

Problème de suppression

- Exemple:

  Supprimer de Beers(name,manf) toutes les bières pour lesquelles il y a une autre bière fournie par le même fabricant.

```
DELETE FROM Beers b
WHERE EXISTS (
    SELECT name FROM Beers
    WHERE manf = b.manf AND
          name <> b.name);
```

Bières faite par le même fabricant et ayant un nom différent du nom de la bière représenté par le tuple b.

name	manf
Bud	Anheuser-Busch
Bud lite	Anheuser-Busch
Man	Peterson

?

name	manf
Man	Peterson

name	manf
Bud lite	Anheuser-Busch
Man	Peterson

Problème de suppression

- Exercice: exécution de la suppression
  - Supposons que Anheuser-Busch fait uniquement de la Bud et de la Bud Lite.
  - Supposons que le SGBD arrive au tuple *b* de la première Bud.
  - La sous-requête est non vide, à cause du tuple avec Bud Lite, donc le tuple *b* de la Bud est supprimé.
  - Maintenant, quand le tuple *b* devient le tuple avec Bud Lite, est-il aussi supprimé ?

➡ *Réponse*: le tuple avec Bud Lite est aussi supprimé

Ceci est dû au fait que la suppression s'effectue en deux étapes:

1. marquer les tuples pour lesquels la condition du WHERE est satisfaite
2. supprimer les tuples marqués

Mise à jour

- Pour changer la valeur de certains attributs dans certains tuples d'une relation:

```
UPDATE <relation>
SET <list of attribute assignments>
WHERE <condition on tuples>;
```

- Exemple:

Modifier le numéro de téléphone du client Fred à la valeur 555-1212:

```
UPDATE Drinkers
SET phone = '555-1212'
WHERE name = 'Fred';
```

Fixer à $4 le prix maximum d'une bière:

```
UPDATE Sells
SET price = 4.00
WHERE price > 4.00;
```