Bases de Données

Amélie Gheerbrant

Université Paris Diderot
UFR Informatique
Laboratoire d'Informatique Algorithmique : Fondements et Applications

amelie@liafa.univ-paris-diderot.fr

13 novembre 2014

Le Modèle Relationnel

- Les données sont organisées dans des relations (tables)
- Schéma de bases de données relationnel
  - ensemble de noms de tables
  - liste d'attributs pour chaque table
- Les tables sont spécifiées sous la forme :

  `<nom de la table> : <liste d'attributs>`
- Exemples :

  `Compte: numero, agence, clientId`

  `Film: titre, directeur, acteur`
- Dans une même table les attributs ont des noms différents
- Les tables ont des noms différents

Exemple de base de données relationnelle

Film	Titre	Réalisateur	Acteur
	Shining	Kubrick	Nicholson
	The Player	Altman	Robbins
	Chinatown	Polanski	Nicholson
	Chinatown	Polanski	Polanski
	Repulsion	Polanski	Deneuve

Projection	Cinéma	Titre
	Le Champo	Shining
	Le Champo	Chinatown
	Le Champo	The Player
	Odéon	Chinatown

Exemples de requêtes

- Trouver le nom des films projetés en ce moment :

réponse	titre
	Shining
	The Player
	Chinatown

- Trouver les cinémas qui passent des films de Polanski :

réponse	cinéma
	Le Champo
	Odéon

- Trouver les cinémas qui passent des films avec Nicholson :

réponse	cinéma
	Le Champo
	Odéon

- Trouver tous les réalisateurs qui se sont dirigés eux-mêmes :

réponse	director
	Polanski

- Trouver tous les réalisateurs dont les films sont joués dans tous les cinémas :

réponse	cinéma
	Polanski

- Trouver tous les cinémas qui ne passent que des films avec Nicholson :

réponse	cinéma

mais si le Champo cesse de passer 'The Player', la réponse devient :

réponse	cinéma
	Le Champo

Les Résultats des Requêtes

- Ce sont des tables construites à partir d'autres tables de la base de données

Comment formuler une requête ?

- Deux types de langages de requêtes :
  - Commercial : SQL
  - Théorique : le Calcul Relationnel, l'Algèbre Relationnelle, Datalog, etc

Déclaratif versus Procédural

Déclaratif :

$$\{\text{cinéma} \mid (\text{titre}, \text{réalisateur}, \text{acteur}) \in \text{film}, (\text{cinéma}, \text{titre}) \in \text{Projection}, \text{acteur}=\text{'Nicholson'} \}$$

Procédural :

```
for each tuple T1=(t1,r,a) in relation film do
    for each tuple T2=(c,t2) in relation projection do
        if t1=t2 and a='Nicholson' then output c
    end
end
```

Déclaratif versus Procédural

- **Langages théoriques :**
  - Déclaratif : le Calcul Relationnel, les requêtes basées sur des règles
  - Procédural : l'Algèbre Relationnelle

- **Langages utilisés en pratique :** mélange des deux, mais surtout déclaratif

On va voir quoi aujourd'hui ?

- exemple de requêtes dans différents langages

Exemples de requêtes

- Trouver le nom des films projetés en ce moment :

  `réponse(ti) :- film(ti, real, act)`

  i.e.,

  while tuple `(ti, real, act)` dans la relation film,
  output `ti` (valeur de l'attribut titre)

- Formulation des requêtes comme règles indiquant quand certains éléments appartiennent à la réponse.
- On appelle ça des requêtes conjonctives (pourquoi ? +tard)

Autre exemple

- Trouver les cinémas qui passent des films de Polanski :

`réponse(ci) :- film(ci, 'Polanski', act), projection(ci, ti)`

i.e.,

while `(ti, real, act)` dans la relation film,
tester : `real='Polanski'` ? ;
si non, considérer le tuple suivant,
si oui, considérer tous les tuples `(ci,ti)` dans Projection,
et pour chacun, *output* `ci` (valeur de l'attribut cinema)

= type de requête le plus répandu.

- Trouver les réalisateurs qui se sont dirigés eux-mêmes :

```
réponse(real) :- film(ti, real, act), real=act
```

i.e.,

while `(ti, real, act)` dans la relation film,
tester : `real=act?` ;
si non, considérer le tuple suivant,
si oui, `output real`.

Un exemple plus compliqué

Trouver les réalisateurs dont les films passent dans tous les cinémas...

- "Tous" : souvent problématique.
- Besoin du quantificateur universel $\forall$

$$\{\text{real} \mid \quad \forall\ (\text{ci}, \text{ti}) \in \text{Projection}, \exists\ (\text{ti'}, \text{act}) : \quad (\text{ti'}, \text{real}, \text{act}) \in \text{Film} \wedge (\text{ci}, \text{ti'}) \in \text{Projection}\ \}$$

pour tester si `real` $\in$ réponse, pour tout nom de cinéma `ci`, tester s'il existe un tuple `(ti', real, act)` dans Film et un tuple (ci, ti') dans Projection.

Notation de la logique mathématique :

- $\forall$ signifie "pour tout", $\exists$ signifie "il existe"
- $\wedge$ est une conjonction (ET logique)

SQL, les raisons du succès

- SQL = *Structured Query Language* (IBM fin 1970')
- Standards : SQL-86, SQL-92, SQL-99 / SQLL3 (+1000 pages)
- Requêtes basées sur le modèle relationnel : langage logique, simple et compréhensible.
- Une requête du calcul peut être facilement traduite en une expression de l'algèbre qui s'évalue simplement (Théorème de Codd)
- Algèbre relationnelle = modèle limité de calcul (n'autorise pas les fonctions arbitraires). Autorise l'optimisation de l'évaluation des expressions algébriques.
- Parallélisme facilité pour les très grandes bases de données.

Exemples de requêtes SQL

- Trouver le nom des films projetés en ce moment :

```
SELECT Titre
FROM FILM ;
```

- SELECT liste les attributs retournés par la requête
- FROM liste les relations prises en entrée

Plus d'exemples

- Trouver les cinémas qui passent des films de Polanski :

```
SELECT Projection.cinema
FROM Projection , Film
WHERE Film.titre = Projection.titre
                AND Film.realisateur = 'Polanski' ;
```

Différences :

- SELECT spécifie maintenant de quelle relation viennent les attributs - parce qu'on en utilise plus d'une
- FROM liste deux relations
- WHERE spécifie les conditions de sélection des tuples

Jointures de relations

- `WHERE` autorise à faire la jointure de plusieurs relations.

Requête : lister les réalisateurs avec les cinémas dans lesquels passent leurs films

- Requête conjonctive :

```
réponse(real, ci) :- Projection(ci, ti), Film(ti, real, act)
```

- Requête SQL :

```
SELECT Film.realisateur, Projection.cinema
FROM Projection, Film
WHERE Film.titre = Projection.titre ;
```

Jointures de relations

- `SELECT Cinema.realisateur, Projection.cinema`
  `FROM Projection, Film`
  `WHERE Film.titre = Projection.titre ;`

- Sémantique : boucles imbriquées sur les relations listées dans le `FROM`

```
for each tuple (titre1, réalisateur, acteur) in Film do
     for each tuple (cinema, titre2) in Projection do
           if titre1=titre2 then output (realisateur, cinema)
     end
end
```

- Cette opération s'appelle une jointure : une des opérations les plus fondamentales en BD.

Un langage procédural : l'algèbre relationnelle

- Commençons par un sous-ensemble de l'algèbre relationnelle qui suffit à capturer les requêtes simples basées sur les règles et les énoncés SQL simples de la forme `SELECT-FROM-WHERE`.

- Ce sous-ensemble a trois opérations :

  Projection π
  Sélection σ
  Produit Cartésien $\times$

- Parfois on utilise aussi le renommage ρ, mais on peut l'éviter sous certaines conditions.

La Projection

- Choisit des attributs dans une relation.
- $\pi_{A_1,\ldots,A_n}(R)$: conserve uniquement les attributs $A_1, \ldots, A_n$ dans la relation R
- Exemple

$\pi_{titre,realisateur}$

titre	réalisateur	acteur
Shining	Kubrick	Nicholson
The Player	Altman	Robins
Chinatown	Polanski	Nicholson
Chinatown	Polanski	Polanski
Repulsion	Polanski	Deneuve

=

titre	réalisateur
Shining	Kubrick
The Player	Altman
Chinatown	Polanski
Repulsion	Polanski

- Fournit à l'utilisateur une vue des données en omettant certains attributs

La sélection

- Choisit des tuples satisfaisant certaines conditions.
- $\sigma_{cond}(R)$: conserve uniquement les tuples t pour lesquels la condition $cond(t)$ est vraie.
- Conditions : conjonctions de
  - `R.A = R.A'` deux attributs ont la même valeur
  - `R.A = c` la valeur de l'attribut est la constante c
  - Idem mais avec $\neq$ à la place de $=$
- Exemples :
  - `Film.acteur = Film.realisateur`
  - `Film.acteur` $\neq$ `Nicolson`
  - `Film.acteur = Film.realisateur` $\wedge$ `Film.acteur = Nicolson`
- Fournit à l'utilisateur une vue des données en omettant les tuples qui ne satisfont pas certaines conditions voulues par l'utilisateur.

La sélection : exemple

$$\sigma_{acteur=realisateur \wedge realisateur='Polanski'}$$

titre	réalisateur	acteur
Shining	Kubrick	Nicholson
The Player	Altman	Robins
Chinatown	Polanski	Nicholson
Chinatown	Polanski	Polanski
Repulsion	Polanski	Deneuve

=

titre	réalisateur	acteur
Chinatown	Polanski	Polanski

Combiner sélection et projection

- Trouver les réalisateurs qui ont joué dans leurs propres films :
- `réponse(real) :- film(ti,real,act), act=real`
- ```
  SELECT realisateur
  FROM Film
  WHERE realisateur=acteur ;
  ```
- Requête de l'algèbre relationnelle :

$$Q = \pi_{realisateur}(\sigma_{realisateur=acteur}(\text{Film}))$$

- $\sigma_{realisateur=acteur}(\text{Film})$ donne

titre	réalisateur	acteur
Chinatown	Polanski	Polanski

- D'où $\pi_{realisateur}(\sigma_{realisateur=acteur}(\text{Film}))$ donne

réalisateur
Polanski
  ```

Combiner sélection et projection

- Il peut y avoir plusieurs façons d'écrire la même chose
- Exemple : trouver les films et les directeurs en excluant les films de Polanski
- `réponse(ti,real) :- film(ti,real,act), real` $\neq$ `'Polanski'`
- Requête de l'algèbre relationnelle :

$$Q_1 = \sigma_{realisateur \neq `Polanski'}(\pi_{titre,realisateur}(Film))$$

- Une autre requête, équivalente, de l'algèbre relationnelle :

$$Q_1 = \pi_{titre,realisateur}(\sigma_{realisateur \neq `Polanski'}((Film))$$

- La même requête déclarative peut avoir plusieurs traductions procédurales.

Combiner sélection et projection

- Q_1 et Q_2, est-ce que c'est *la même* requête ?
- Sémantiquement, oui : elles produisent le même résultat
- Mais elles différent en termes d'efficacité.
- Q_1 parcourt d'abord `Film`, projette deux attributs, et parcourt à nouveau le résultat.
- Q_2 parcourt `Film`, sélectionne certains tuples, et parcourt ensuite les tuples sélectionnés.
- Q_2 semble donc efficace dans le contexte présent.
- Les langages procéduraux peuvent être optimisés : il y a des manières sémantiquement équivalentes d'écrire la même requête, et certaines sont plus efficaces que d'autres.

Le Produit Cartésien

- Mets deux relations ensemble.
- $R_1 \times R_2$ associe chaque tuple t_1 de R_1 avec chaque tuple t_2 de R_2
- Exemple :

R_1	A	B
	a_1	b_1
	a_2	b_2

$\times$

R_2	A	C
	a_1	c_1
	a_2	c_2
	a_3	c_3

$=$

$R_1.A$	$R_1.B$	$R_2.A$	$R_2.C$
a_1	b_1	a_1	c_1
a_1	b_1	a_2	c_2
a_1	b_1	a_3	c_3
a_2	b_2	a_1	c_1
a_2	b_2	a_2	c_2
a_2	b_2	a_3	c_3

- Nous avons renommé les attributs pour inclure le nom des relations : dans la table résultante, tous les attributs doivent avoir des noms différents.

Le Produit Cartésien

- Si R_1 a n tuples et R_2 a m tuples, alors $R_1 \times R_2$ a $n \times m$ tuples
- Opération coûteuse : si R et S ont chacun 1000 tuples (petites relations), $R \times S$ a 1 000 000 tuples ($\neq$ petit).
- Les algorithmes d'optimisation de requêtes essaient d'éviter la construction des produits - à la place ils tentent d'en construire seulement des sous-ensembles ne contenant que les informations pertinentes.

Le Produit Cartésien : exemple

Trouver les cinémas qui jouent des films de Polanski :

```
réponse(ci) :- Film(ti,real,act), Projection(ci,ti), real='Polanski'
```

- Étape 1 : Soit $R_1 = \mathit{Film} \times \mathit{Projection}$
- On ne veut que les tuples dans lesquels les titres sont identiques, d'où :
- Étape 2 : Soit $R_2 = \sigma_{\mathit{Film.titre}=\mathit{Projection.titre}}(R_1)$
- Étape 3 : On ne veut que les films de Polanski, d'où :

$$R_3 = \sigma_{\mathit{realisateur}=\text{`}\mathit{Polanski}\text{'}}(R_2)$$

- Étape 4 : Dans la réponse, on ne veut que des cinémas, donc :

$$\text{Réponse} = \pi_{\mathit{cinema}}(R_3)$$

- En résumé, la réponse est :

$$\pi_{\mathit{cinema}}(\sigma_{\mathit{realisateur}=\text{`}\mathit{Polanski}\text{'}}(\sigma_{\mathit{Film.titre}=\mathit{Projection.titre}}(\mathit{Film}\times\mathit{Projection})))$$

Le Produit Cartésien : exemple

- La réponse est :

$$\pi_{cinema}(\sigma_{realisateur='Polanski'}(\sigma_{Film.titre=Projection.titre}(Film \times Projection)))$$

- Mais plusieurs sélections peuvent être combinées en une seule :

$$\sigma_{cond_1}(\sigma_{cond_2}(R)) = \sigma_{cond_1 \wedge cond_2}(R)$$

(avec $cond_1$, $cond_2$ des conditions Booléennes sur les tuples)

- Au final la réponse à la requête est donc :

$$\pi_{cinema}(\sigma_{realisateur='Polanski' \wedge Film.titre=Projection.titre}(Film \times Projection))$$

SQL et l'Algèbre Relationnelle

Il nous faut maintenant traduire d'un langage déclaratif à un langage procédural.

- Idée :
  `SELECT` correspond à la projection π
  `FROM` au produit Cartésien $\times$
  `WHERE` à la sélection σ

- Cas simple : juste une relation dans le `FROM`

```
SELECT A,B, · · ·
FROM R
WHERE condition c ;
```

  sera traduit

$$\pi_{A,B,\ldots}(\sigma_c(R))$$

Traduction des requêtes déclaratives dans l'algèbre relationnelle

- Trouver le titre de tous les films

  `réponse(ti) :- Film(ti,real,act)`
- `SELECT Titre`
  `FROM Film ;`
- C'est juste une projection :

$$\pi_{titre}(Film)$$

Exemples de Traductions

- Trouver tous les cinémas qui jouent des films de Polanski :

```
SELECT Projection.cinema
FROM Projection, Film
WHERE Film.titre = Projection.titre
AND Film.realisateur='Polanski' ;
```

- D'abord, traduire sous forme de règle :

```
réponse(ci) :- Projection(ci,ti), Film(ti,'Polanski',act)
```

- Ensuite, convertir en une règle dans laquelle :
  - les constantes n'apparaissent que dans les conditions
  - toutes les variables sont distinctes
- ce qui nous donne :

```
réponse(ci) :- Projection(ci,ti), Film(ti',real,act), real = 'Polanski', ti=ti'
```

Exemples de Traductions

```
réponse(ci) :- Projection(ci,ti), Film(ti',real,act), real = 'Polanski', ti=ti'
```

- Deux relations $\Rightarrow$ produit Cartésien
- Conditions $\Rightarrow$ sélection
- Sous-ensemble des attributs dans la réponse $\Rightarrow$ projection

Étapes :

1. $R1 = Projection \times Film$
2. On ne veut parler que d'un seul et même film :
$$R_2 = \sigma_{Projection.titre=Film.titre}(R_1)$$
3. On ne veut que les films de Polanski :
$$R_3 = \sigma_{Film.realisateur} = Polanski(R_2)$$
4. Dans la réponse on ne veut que les cinémas :
réponse $= \pi_{Projection.cinema}(R_3)$

Exemples de Traductions

En résumé, la réponse est :

$$\pi_{Projection.cinema}(\sigma_{Films.realisateur='Polanski'}(\sigma_{Projection.titre=Film.titre}(Projection \times Film)))$$

ou, de par la règle $\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c1 \wedge c_2}(R)$:

$$\pi_{Projection.cinema}(\sigma_{Films.realisateur='Polanski' \wedge Projection.titre=Film.titre}(Projection \times Film))$$

Traduction formelle : de SQL aux règles

```
SELECT liste d'attributs <Ri.Aj>
FROM R1, ..., Rn
WHERE condition c ;
```

se traduit :

```
                                  R1(<attributs>)
réponse(<Ri.Aj>)    :-            ...,
                                  R1(<attributs>)
                                  c
```

Des règles à l'algèbre relationnelle

- Comment les règles sont-elles traduites dans l'algèbre ?

  `réponse(a1,...,ak) :- R1(`$\bar{A}_1$`), ...,Rn(`$\bar{A}_n$`),conditions`

- D'abord, s'assurer que les attributs sont deux à deux distincts : si on a $R_i(\ldots, A, \ldots)$ et $R_j(\ldots, A, \ldots)$ avec $i \neq j$, alors, traduire en $R_i(\ldots, A', \ldots)$ et $R_j(\ldots, A'', \ldots)$ et ajouter $A' \neq A''$ aux conditions.

- Exemple : `réponse(ti,real) :- Film(ti,real,act), Projection(ci,ti)`

  devient

  `réponse(ti,real) :- Film(ti',real,act), Projection(ci,ti'', act), ti'=ti''`

- Ces règles sont ensuite traduites :

$$\pi_{a_1,\ldots,a_k}(\sigma_{conditions}(R_1 \times \ldots \times R_n))$$

Enfin, de SQL à l'algèbre relationnelle

- En combinant les traductions :
  de SQL vers les règles, puis des règles vers l'algèbre
  on obtient la traduction suivante, de SQL vers l'algèbre :

```
SELECT liste d'attributs <Ri.Aj>
FROM R1, ..., Rn
WHERE condition c ;
```

devient

$$\pi_{<R_i.A_j>}(\sigma_c(R_1 \times \ldots \times R_n))$$

La jointure naturelle

- Étapes communes pour la traduction des deux requêtes précédentes :
  1. $R_1 = \mathit{Projection} \times \mathit{Film}$
  2. S'assurer que l'on parle bien du même film :

$$R_2 = \sigma_{\mathit{Projection.titre}=\mathit{Film.titre}}(R_1)$$

- Attributs de R_2 : Projection.cinema, Projection.titre, Film.titre, Film.realisateur, Film.acteur
- Mais l'un des attributs est redondant :
  Film.titre et Projection.titre sont toujours identiques dans R_2.
- On réduit donc R_2 à une relation plus simple avec les attributs :
  Projection.cinema, titre, Film.realisateur, Film.acteur
- Il s'agit de la jointure naturelle : Projection $\bowtie$ Film

La jointure naturelle : un exemple

Titre	Réalisateur	Acteur
Shining	Kubrick	Nicholson
The Player	Altman	Robbins
Chinatown	Polanski	Nicholson
Chinatown	Polanski	Polanski
Repulsion	Polanski	Deneuve

$\bowtie$

Cinéma	Titre
Le Champo	Shining
Le Champo	Chinatown
Le Champo	The Player
Odéon	Chinatown

=

Titre	Realisateur	Acteur	Cinema
Shining	Kubrick	Nicholson	Le Champo
The Player	Altman	Robbins	Le Champo
Chinatown	Polanski	Nicholson	Le Champo
Chinatown	Polanski	Nicholson	Odéon
Chinatown	Polanski	Polanski	Le Champo
Chinatown	Polanski	Polanski	Odéon

La jointure naturelle

- La jointure n'est pas une nouvelle opération de l'algèbre relationnelle.
- Elle est définissable à partir de $\pi, \sigma, \times$
- Soit R une relation sur des attributs $A_1, \ldots, A_n, B_1, \ldots, B_k$
- S une relation sur des attributs $A_1, \ldots, A_n, C_1, \ldots, C_m$
- $R \bowtie S$ a pour attributs $A_1, \ldots, A_n, B_1, \ldots, B_k, C_1, \ldots, C_m$

$$R \bowtie S$$
$$=$$
$$\pi_{A_1,\ldots,A_n,B_1,\ldots,B_k,C_1,\ldots,C_m}(\sigma_{R.A_1=S.A_1 \wedge \ldots \wedge R.A_n=S.A_n}(R \times S))$$

Propriétés de la jointure

- Commutativité : $R \bowtie S = S \bowtie R$
- Associativité : $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T)$
- On peut donc écrire $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$

Commutativité et associativité de la jointure

R	employee	department
	Jones	D1
	Brown	D2
	Smith	D3

S	department	office
	D1	USA
	D2	UK

T	office	head
	USA	Andrews
	UK	Morrison

$R \bowtie S \bowtie T$	employee	department	office	head
	Jones	D1	USA	Andrews
	Brown	D2	UK	Morrison

Requêtes Select-Project-Join (SPJ)

- Il s'agit des requêtes les plus courantes.
- Règles simples, ou requêtes SELECT-FROM-WHERE simples.
- Trouver les cinémas qui passent des films de Polanski :

```
réponse(ci) :- Projection(ci,ti), Film(ti,'Polanski',act)
```

- Comme requête SPJ :

$$\pi_{Projection.cinema}(\sigma_{realisateur='Polanski'}(Film \bowtie Projection))$$

- En quoi est-ce une simplification de la version précédente ?

$$\pi_{Projection.cinema}(\sigma_{Film.realisateur='Polanski'}(\sigma_{Film.titre=Projection.titre}(Film \bowtie Projection)))$$

- $\sigma_{Film.titre=Projection.titre}$ est éliminé ; car impliqué par la jointure.

Propriétés des opérateurs de l'algèbre relationnelle

Taille des résultats

- Projection : $taille(\pi(R)) \leq taille(R)$
  (taille = nombre de tuples)
- Parfois, $taille(\pi(R)) < taille(R)$
- Se produit lorsque deux attributs ont les mêmes valeurs

$$\pi_{\mathsf{A}} \left(\begin{array}{cc} \mathsf{A} & \mathsf{B} \\ \hline \mathsf{a} & \mathsf{b1} \\ \mathsf{a} & \mathsf{b2} \end{array} \right) = \begin{array}{c} \mathsf{A} \\ \hline \mathsf{a} \end{array}$$

- Sélection : $0 \leq taille(\sigma(R)) \leq taille(R)$
- Dépend du nombre de tuples satisfaisant les conditions.

Taille des jointures et des produits Cartésiens

- $taille(R \times S) = taille(R) \times taille(S)$, mais :

$$0 \leq taille(R \bowtie S) \leq taile(R) \times taille(S)$$

- Certains tuples peuvent ne pas participer à la jointure :

R	employee	department
	Jones	D1
	Brown	D2
	Smith	D3

$\bowtie$

S	department	office
	D1	USA
	D2	UK

$=$

$R \bowtie S$	employee	department	office
	Jones	D1	USA
	Brown	D2	UK

- $(Smith, D3)$ n'est joint avec aucun tuple de S : S ne contient pas d'information sur le département D3.

Jointure vides

Les jointures peuvent être vides :

R	employee	department
	Jones	D1
	Brown	D2
	Smith	D3

$\bowtie$

S'	department	office
	D4	France
	D5	Italy

$=$

$R \bowtie S'$	employee	department	office

Traduction : de SPJ vers les règles vers SQL

- $Q = \pi A(\sigma_c(R \bowtie S))$
- Soit $B_1, \ldots, B_m$ les attributs communs de R et S
- Requête SQL équivalente :

```
SELECT A
FROM R, S
WHERE c, R.B1 = S.B1 AND . . . AND R.Bm = S.Bm ;
```

- Règle équivalente :

```
réponse(Ā)   :-  R(<attributs de R>), S(<attributs de S>),
                 R.B1 = S.B1, ..., R.Bm = S.Bm, c
```

De SPJ vers SQL : exemple

- Trouver les réalisateurs des films joués en ce moment et dans lesquels joue Ford :

$$\pi_{realisateur}(\sigma_{acteur='Ford'}(Film \bowtie Projection))$$

- Avec SQL :

```
SELECT Film.realisateur
FROM Film, Projection
WHERE Film.titre=Projection.titre AND Film.acteur='Ford' ;
```

Vu aujourd'hui

- Requêtes SQL simples `SELECT-FROM-WHERE`
- Mêmes requêtes données sous forme de règles
- Correspondent aux requêtes définissables dans l'algèbre relationnelle avec $\pi, \sigma, \times$

Sauver de l'espace

- Répéter les noms de relations plusieurs fois est un peu lourd
- SQL nous laisse donc utiliser des noms de relations temporaires pour les relations
- ```
  SELECT P.cinema
  FROM Projection P, Film F
  WHERE P.titre=F.titre AND F.realisateur='Polanski'
  ```
- Utiliser une variable après le nom d'une relation indique que la relation est temporairement renommée
  ```

Les requêtes imbriquées : un exemple simple

- Jusqu'à présent dans la clause `WHERE` nous avons utilisé des comparaisons d'attributs
- En général, une clause `WHERE` peut contenir une autre requête, et tester une relation entre un attribut et le résultat d'une autre requête.
- On parle de requêtes imbriquées, car elles utilisent des sous-requêtes
- Exemple : trouver les cinémas qui passent des films de Polanski :

```
SELECT Projection.cinema
FROM Projection
WHERE Projection.titre IN
                    (SELECT Film.titre
                    FROM Film
                    WHERE Film.realisateur='Polanski');
```

Requêtes imbriquées : comparaison

```
SELECT Projection.cinema
FROM Projection
WHERE Projection.titre IN
      (SELECT Film.titre
      FROM Film
      WHERE Film.realisateur='Polanski');
```

```
SELECT P.cinema
FROM Projection P, Film F
WHERE P.titre=F.titre
AND F.realisateur='Polanski';
```

- Sémantique = même requête
- A gauche, chaque sous-requête réfère à une relation
- Avantage de l'imbrication : on peut utiliser des prédicats plus complexes que IN

Disjonction dans les requêtes

- Trouver des acteurs qui ont joué dans des films de Kubrick OU de Polanski
- ```
  SELECT acteur
  FROM Film
  WHERE realisateur='Kubrik' OR realisateur='Polanski';
  ```
- Est-ce qu'on pourrait définir ça avec une seule règle ?
- Non !
  ```

Disjonction dans les requêtes

- Solution : les disjonctions peuvent être représentées par un ensemble de règles :

```
réponse(act) :- Film(ti,real,act), real='Kubrick'
réponse(act) :- Film(ti,real,act), real='Polanski'
```

- Sémantique : calculer la réponse à chacune des règles, puis prendre leur union
- Syntaxe alternative en SQL :

```
        SELECT acteur
        FROM Film
        WHERE realisateur='Kubrick'
UNION
        SELECT acteur
        FROM Film
        WHERE realisateur='Polanski';
```

Disjonction dans les requêtes

- Comment traduire une requête avec des disjonctions dans l'algèbre relationnel ?
- `réponse(act) :- Film(ti,real,act), real='Kubrick'` est traduit par

$$Q_1 = \pi_{acteur}(\sigma_{realisateur='Kubrick'}(Film))$$

- `réponse(act) :- Film(ti,real,act), real='Kubrick'` est traduit par

$$Q_2 = \pi_{acteur}(\sigma_{realisateur='Polanski'}(Film))$$

- On traduit la requête entière par $Q_1 \cup Q_2$:

$$\pi_{acteur}(\sigma_{realisateur='Kubrick'}(Film)) \bigcup \pi_{acteur}(\sigma_{realisateur='Polanski'}(Film))$$

L'union dans l'algèbre relationnelle

- Une autre opération de l'algèbre relationnelle : l'union
  $R \cup S$ est l'union des relations R et S

- R et S doivent avoir les mêmes attributs.
  On a maintenant quatre opérations de l'algèbre relationnelle :

  $$\pi, \sigma, \times, \cup$$

  (et bien sûr, $\bowtie$, qui est définissable à partir de $\pi, \sigma, \times$)

- Ce fragment, est appelé algèbre relationnelle positive, ou requêtes SPJU (select-project-join-union)

Interaction des opérateurs de l'algèbre relationnelle

- $\pi_{\bar{A}}(R \cup S) = \pi_{\bar{A}}(R) \cup \pi_{\bar{A}}(S)$
- $\sigma_{cond}(R \cup S) = \sigma_{cond}(R) \cup \sigma_{cond}(S)$
- $(R \cup S) \times T = (R \times T) \cup (S \times T)$
- $T \times (R \cup S) = (T \times R) \cup (T \times S)$

où $\bar{A} = A_1, \ldots, A_n$ est une séquence d'attributs et *cond* est une condition

Requêtes SPJU

Toute requête SPJU est équivalente à une union de requêtes SPJ.

- Il suffit de propager l'opérateur d'union
- Exemple :

$$\pi_A(\sigma_{cond}((R \times (S \cup T)) \cup W)$$

$$=$$

$$\pi_A(\sigma_{cond}((R \times S) \cup (R \times T) \cup W))$$

$$=$$

$$\pi_A(\sigma_{cond}(R \times S) \cup \sigma_{cond}(R \times T) \cup \sigma_{cond} W))$$

$$=$$

$$\pi_A(\sigma_{cond}(R \times S)) \cup \pi_A(\sigma_{cond}(R \times T)) \cup \pi_A((\sigma_{cond} W))$$

Équivalences

Algèbre positive relationnelle

=

Union de requêtes SPJ

=

requêtes définies par un ensemble de règles

=

requêtes SQL `SELECT-FROM-WHERE-UNION`

=

unions de requêtes conjonctives

=

requêtes définies avec $\exists, \wedge, \vee$

- Question : est-ce que l'intersection est une requête SPJU ?
- i.e., étant donné R, S, avec les mêmes ensemble d'attributs, peut-on définir $R \cap S$?

Plus sur l'union

- Relation R_1 : *pere*, *enfant*

R_1	pere	enfant
	George	Elizabeth
	Philip	Charles
	Charles	William

- Relation R_2 : *mere*, *enfant*

R_2	mere	enfant
	Elizabeth	Charles
	Elizabeth	Andrews

- Nous voulons leur union, qui devrait être la relation "parent-enfant"
- Mais nous ne pouvons pas utiliser $R_1 \cup R_2$, parce que R_1 et R_2 ont des attributs différents !
- Nous devons donc renommer les attributs

Le renommage

- Soit R une relation qui a pour attribut A, mais pas B
- $\rho_{B \leftarrow A}(R)$ est la relation qui est comme R à ceci près que A est renommé B

$$\rho_{\text{parent} \leftarrow \text{pere}} \left(\begin{array}{cc} \text{pere} & \text{enfant} \\ \hline \text{George} & \text{Elizabeth} \\ \text{Philip} & \text{Charles} \\ \text{Charles} & \text{William} \end{array} \right) = \left(\begin{array}{cc} \text{parent} & \text{enfant} \\ \hline \text{George} & \text{Elizabeth} \\ \text{Philip} & \text{Charles} \\ \text{Charles} & \text{William} \end{array} \right)$$

$$\rho_{\text{parent} \leftarrow \text{mere}} \left(\begin{array}{cc} \text{mere} & \text{enfant} \\ \hline \text{Elizabeth} & \text{Charles} \\ \text{Elizabeth} & \text{Andrew} \end{array} \right) = \left(\begin{array}{cc} \text{parent} & \text{enfant} \\ \hline \text{Elizabeth} & \text{Charles} \\ \text{Elizabeth} & \text{Andrew} \end{array} \right)$$

Le renommage

L'union désirée est :

$$\rho_{parent \leftarrow pere}(R_1) \bigcup \rho_{parent \leftarrow mere}(R_1)$$

ce qui donne

parent	enfant
George	Elizabeth
Philip	Charles
Elizabeth	Charles
Elizabeth	Andrew

SQL et le renommage

- De nouveaux attributs peuvent être introduits dans les clauses SELECT en utilisant le mot AS

```
SELECT pere AS parent, enfant
FROM R1;

SELECT mere AS parent, enfant
FROM R2;
```

- On peut prendre l'union des deux requêtes, car elles ont le même ensemble d'attributs

```
      SELECT pere AS parent, enfant
      FROM R1
UNION
      SELECT mere AS parent, enfant
      FROM R2;
```

Requêtes avec "Tous"

- Trouver les réalisateurs dont les films sont joués dans tous les cinémas

$$\{real \mid \forall (ci, ti') \in Projection\ \exists ti, act\ (Projection(ci, ti) \wedge Film(ti, real, act))\}$$

- Qu'est-ce que ça veut dire ?
- Pour comprendre ça il faut revenir aux requêtes basées sur les règles, et les écrire en notation logique.

Les règles revisitées

- Reprenons un exemple familier :

```
réponse(ci) :- Film(ti, 'Polanski', act), Projection(ci, ti)
```

- Qu'est-ce que ça veut dire ?
- On demande, pour chaque cinéma `ci` : "Est-ce qu'il existe un film `ti` et un acteur `act` tels que (ci, ti) appartient à `Projection` et `(ti, 'Polanski', act)` appartient à `Film` ?"
- Dans le langage de la logique mathématique, ça s'écrit comme ça :

$$Q(ci) = \exists ti \exists act\ (Film(ti, 'Polanski', act) \wedge Projection(ci, ti))$$

Autres requêtes en notation logique

```
réponse(ci) :- Film(ti, real, 'Nicholson'), Projection(ci, ti)
```

est équivalente à

$$Q(ci) \;=\; \exists ti\ \exists real\ (Film(ti, real, \text{`}Nicholson\text{'}) \wedge Projection(ci, ti))$$

- En général, toute requête sous forme de règle unique peut être réécrite en utilisant seulement :
  - la quantification existentielle $\exists$
  - la conjonction logique $\wedge$ (AND)

Requêtes SPJU sous forme logique

- Trouver les acteurs qui ont joué dans des films de Polanski OU de Kubrick
- Requêtes sous forme de règles :

```
réponse(act) :- Film(ti, real, act'), real='Polanski'
réponse(act) :- Film(ti, real, act'), real='Kubrick'
```

- Notation logique :

$$Q(act) = \exists ti\ \exists real(Film(ti, real, act) \wedge$$
$$(real = `Kubrick' \vee real = `Polanski'))$$

- Nouvel élément : la disjonction logique $\vee$ (OR)
- Les requêtes SPJU s'écrivent en notation logique en utilisant :
  - la quantification existentielle $\exists$
  - la conjonction $\wedge$ et la disjonction $\vee$

Les requêtes avec "pour tout"

$$\{real \mid \forall (ci, ti') \in Projection\,, \exists ti\ \exists act(Projection(ci, ti) \wedge Film(ti, real, act))\}$$

- Nouvel élément ici : la quantification universelle $\forall$ "pour tout"
- $\forall x\ F(x) \ = \ \neg \exists x\ \neg F(x)$
- Le nouvel élément est donc en fait la négation $\neg$
- Il faut faire attention avec la négation : quelle est la signification de

$$\{x \mid \neg R(x)\}$$

- Ça a l'air de nous dire : donne moi tout ce qui n'est pas dans la base de données. Mais il s'agit d'un ensemble infini !

Les requêtes avec "pour tout" et la négation

- Sûreté : une requête écrite en notation logique est sûre si elle retourne nécessairement des résultats finis sur toutes les bases de données.
- Cette propriété doit être imposée pour tous les langages pratiques.
- Mauvaise nouvelle : il n'existe pas d'algorithme pour vérifier qu'une requête quelconque est sûre.
- Bonne nouvelle : toutes les requêtes SPJ et SPJU sont sûres. Raison : tout les attributs qui occurrent dans la réponse doivent avoir une occurrence dans l'entrée, pas de création de nouvel élément.
- Problème = la négation, comment la gérer ?

Le calcul relationnel

- Calcul relationnel : requêtes écrites en notation logique au moyen de :
  - noms de relations (e.g., Film)
  - constantes (e.g., 'Nicholson')
  - conjonction $\wedge$, disjonction $\vee$
  - négation $\neg$
  - quantificateurs existentiels $\exists$
  - quantificateurs universels $\forall$
- $\wedge, \exists, \neg$ suffisent à définir tous les autres opérateurs :
  - $\forall x\ F(x) = \neg\exists x \neg F(x)$
  - $F \vee G = \neg(\neg F \wedge \neg G)$
- Autre nom : la logique du premier ordre

Le calcul relationnel

- Variable liée : une variable x qui ocurre dans $\forall x$ ou $\exists x$
- Variable libre : une variable qui n'est pas liée.
- Les variables libres sont celles qui vont dans la réponse de la requête
- Deux façons d'écrire une requête :
  $Q(\bar{x}) = F$, où $\bar{x}$ est un tuple de variables libres
  $\{\bar{x} \mid F\}$
- Exemples :
  $\{x, y \mid \exists z\ (R(x, z) \wedge S(z, y))\}$
  $\{x \mid \exists y\ R(x, y)\}$
- Les requêtes sans variables libres sont appelées requêtes Booléennes.
- Leur réponse est toujours le *vrai* ou le *faux*. Exemples :
  $\forall x\ R(x, x)$
  $\forall x \exists y\ R(x, y)$

Le calcul relationnel sûr

- Une requête du calcul relationnel $Q(\bar{x})$ est sûre si elle retourne toujours un résultat fini.
- Exemple de requêtes sûres :
  - toute requête Booléenne
  - toute requête SPJ ou SPJU
- Exemple de requêtes non sûres :
  - $\{x \mid \neg R(x)\}$
  - $\{x, y \mid \mathit{Film}(x, \mathit{Polanski}, \mathit{Nicholson}) \vee \mathit{Film}(\mathit{Chinatown}, \mathit{Polanski}, y)$
- Calcul relationnel sûr = ensemble des requêtes sûres du calcul relationnel
- Mais la sûreté ne peut pas être vérifiée par un algorithme !
- On peut tout de même décrire ce langage.

La différence

- Si R et S sont deux relations avec le même ensemble d'attributs, alors $R - S$ est leur différence : l'ensemble des tuples qui occurrent dans R mais pas dans S

- Exemple :

A	B
a1	b1
a2	b2
a3	b3

$-$

A	B
a2	b2
a3	b3
a4	b4

$=$

A	B
a1	b1

L'algèbre relationnelle

- Inclut les opérateurs $\pi, \sigma, \times, \cup, -, \rho$

  Théorème fondamental de la théorie des bases de données relationnelles :
  Le calcul relationnel sûr = l'algèbre relationnelle

- On ne va pas donner une preuve formelle ici, mais essayer d'expliquer pourquoi c'est vrai.

Du l'algèbre relationnelle au calcul relationnel sûr

- Montrer que l'algèbre relationnelle (qui est sûre) peut être exprimée dans le calcul relationnel
- Chaque expression e produisant une relation à n attributs est traduite par une formule $F_e(x_1, \ldots, x_n)$
- R est traduit par $R(x_1, \ldots, x_n)$
- $\sigma_{cond}(R)$ est traduit par $R(x_1, \ldots, x_n) \wedge cond$

  Exemple : si R a pour attributs A et B, $\sigma_{A=B}(R)$ est traduit par $(R(x_1, x_2) \wedge x_1 = x_2)$

Du l'algèbre relationnelle au calcul relationnel sûr

- Si R a pour attributs $A_1, \ldots, A_n, B_1, \ldots, B_m$, alors $\pi_{A_1,\ldots,A_n}(R)$ se traduit

$$\exists y_1, \ldots, y_m \; R(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

  Important : ce sont les attributs qui ne sont *pas* projetés qui sont quantifiés.

  Exemple : si R a pour attributs A, B, $\pi_A(R)$ se traduit $\exists x_2 \; R(x_1, x_2)$

- $R \times S$ se traduit $R(x_1, \ldots, x_n) \wedge S(y_1, \ldots, y_m)$ toutes les variables sont distinctes et le résultat aura donc $n + m$ attributs

Du l'algèbre relationnelle au calcul relationnel sûr

- Si R et S ont même ensemble d'attributs, alors $R \cup S$ se traduit $R(x_1, \ldots, x_n) \vee S(x_1, \ldots, x_n)$
  (toutes les variables sont les mêmes et le résultat aura donc n attributs)

- Si R et S ont même ensemble d'attributs, alors $R - S$ se traduit $R(x_1, \ldots, x_n) \wedge \neg S(x_1, \ldots, x_n)$
  (toutes les variables sont les mêmes et le résultat aura donc n attributs)

Préliminaires à la traduction du calcul relationnel sûr dans l'algèbre relationnelle

- Domaine actif d'une relation : l'ensemble des constantes qui y occurrent.

  Exemple : le domaine actif de

R_1	A	B
	a_1	b_1
	a_2	b_2

  est $\{a_1, a_2, b_1, b_2\}$

- Calcul du domaine actif de R
  Soit R avec des attributs $A_1, \ldots, A_n$

$$ADOM(R) = \rho_{B \leftarrow A_1}(\pi_{A_1}(R)) \cup \ldots \cup \rho_{B \leftarrow A_n}(\pi_{A_1}(R))$$

- On construit une relation sur un unique attribut B
- De même on peut calculer

$$ADOM(R_1, \ldots, R_k) = ADOM(R_1) \cup \ldots \cup ADOM(R_k)$$

Du calcul relationnel sûr à l'algèbre relationnelle

- Une requête sûre sur les relations $R_1, \ldots, R_n$ ne peut produire d'élément hors de $ADOM(R_1, \ldots, R_n)$.
- I.e., pour une requête sûre Q,

$$ADOM(Q(R_1, \ldots, R_n)) \subseteq ADOM(R_1, \ldots, R_n)$$

- Raison : tous les éléments hors de $ADOM(R_1, \ldots, R_n)$ se valent, si l'un est dans la réponse alors tous le sont, et donc la requête n'est pas sûre.
- On traduit donc les requêtes du calcul relationnel évaluées dans $ADOM(R_1, \ldots, R_n)$ en requêtes de l'algèbre relationnelle
- Chaque formule du calcul relationnel $F(x_1, \ldots, x_n)$ est traduite en une expression E_F qui produit une relation avec n attributs.

Du calcul relationnel sûr à l'algèbre relationnelle : traduction

- Cas faciles (pour R avec attributs $A_1, \ldots, A_n$) :

$$R(x_1, \ldots, x_n) \quad \text{se traduit} \quad R$$

$$\exists x_1 R(x_1, \ldots, x_n) \quad \text{se traduit} \quad \pi_{A_2,\ldots,A_n} R$$

- Cas moins faciles :
  - Une condition $c(x_1, \ldots, x_n)$ se traduit

$$\sigma_c(ADOM \times \ldots \times ADOM)$$

    e.g., $x_1 = x_2$ se traduit $\sigma_{x_1=x_2}(ADOM \times ADOM)$
  - Une négation $\neg R(\bar{x})$ se traduit

$$(ADOM \times \ldots \times ADOM) - R$$

    i.e., on ne calcule que les tuples d'éléments *de la base de données* qui n'appartiennent pas à R

Du calcul relationnel sûr à l'algèbre relationnelle : traduction

- Le cas le plus difficile : la disjonction
- Soit R et S avec deux attributs

$$Q(x, y, z) = R(x, y) \vee S(x, z)$$

- Son résultat a trois attributs et consiste en tous les tuples (x, y, z) tels que :
  - soit $(x, y) \in R$ et $z \in ADOM$,
  - ou bien $(x, z) \in S$ et $y \in ADOM$
- Le premier ensemble de tuples est simplement $R \times ADOM$
- Le second est plus complexe à définir :

$$\pi_{\#1,\#3,\#5}(\sigma_{\#1=\#4 \wedge \#2=\#5}(S \times ADOM \times S))$$

- Q est donc traduite comme suit :

$$R \times ADOM \cup \pi_{\#1,\#3,\#5}(\sigma_{\#1=\#4 \wedge \#2=\#5}(S \times ADOM \times S))$$

($\#i$ = i-ème élément du produit cartésien $S \times ADOM \times S$, relation 5-aire.)

Les requêtes avec "pour tout" dans l'algèbre relationnelle

- Trouver les réalisateurs dont les films sont joués dans tous les cinémas.

$$\{real \mid \forall (ci, ti') \in Projection\ \exists ti, act(Projection(ci, ti) \wedge Film(ti, real, act))\}$$

- On définit :
  - $C_1 = \pi_{cinema}(P)$
  - $C_2 = \pi_{cinema,realisateur}(F \bowtie P)$

  (pour sauver de l'espace on va utiliser P pour projection et F pour Film)
- C_1 contient tous les cinémas, C_2 contient tous les réalisateurs avec les cinémas où leurs films passent.
- Notre requête est :

$$\{real \mid \forall ci \in C_1\ (ci, real) \in C_2)\}$$

Requêtes avec "pour tout"

$$\{real \mid \forall ci \in C_1 \ (ci, real) \in C_2)\}$$

se réécrit

$$\{real \mid \neg(\exists ci \in C_1 \ (ci, real) \notin C_2)\}$$

La réponse à la requête est donc

$$\pi_{realisateur}(F) - V$$

où $V = \{real \mid (\exists ci \in C_1 \ (ci, real) \notin C_2)\}$

Les paires $(ci, real)$ qui ne sont pas dans C_2 sont

$$(C_1 \times \pi_{realisateur}(F)) - C_2$$

D'où :

$$V = \pi_{realisateur}((C_1 \times \pi_{realisateur}(F)) - C_2)$$

Requêtes avec "pour tout"

- Requête : trouver les réalisateurs dont les films passent dans tous les cinémas.
- On obtient donc :

$$\pi_{realisateur}(F) - \pi_{realisateur}((\pi_{cinema}(P) \times \pi_{realisateur}(F)) - \pi_{cinema,realisateur}(F \bowtie P))$$

- Beaucoup moins intuitif que la description logique de la requête.
- Les langages procéduraux sont loin d'être aussi compréhensibles que les langages déclaratifs...

Pour tout et la négation dans SQL

- Trouver les réalisateurs dont les films passent dans tous les cinémas.
- La façon SQL de dire ça : trouver tous les réalisateurs tels qu'il n'existe pas de cinéma où leurs films ne passent pas.

```
SELECT F1.realisateur
FROM Film F1
WHERE NOT EXISTS (SELECT P.cinema
                  FROM Projection P
                  WHERE NOT EXISTS (SELECT F2.realisateur
                                    FROM Film F2
                                    WHERE F2.titre=P.titre
                                        AND
                                    F1.realisateur=F2.realisateur))
```

Pour tout et la négation dans SQL

Même requête avec EXCEPT

```
SELECT F.realisateur
FROM Film F
WHERE NOT EXISTS (SELECT P.cinema
                  FROM Projection P
                    EXCEPT
                  SELECT P1.cinema
                  FROM Projection P1, Film F1
                  WHERE P1.titre=F1.titre
                  AND F1.realisateur=F.realisateur)
```

- Autres conditions : IN, NOT IN, EXISTS...

Pour tout et la négation dans SQL

- Deux mécanismes principaux : les sous requêtes et les expressions ensemblistes
- Sous-requêtes souvent plus naturelles
- Syntaxe de SQL pour $R \cap S$:

  `R INTERSECT S`
- Syntaxe de SQL pour $R - S$:

  `R EXCEPT S`
- Trouver tous les acteurs qui

- ne sont pas réalisateurs :

```
SELECT acteur AS personne
FROM Film
 EXCEPT
SELECT realisateur AS personne
FROM Film;
```

- sont aussi réalisateurs :

```
SELECT acteur AS personne
FROM Film
 INTERSECT
SELECT realisateur AS personne
FROM Film;
```

Requêtes sans intersect et except

- Trouver tous les acteurs qui

- ne sont pas réalisateurs :

```
SELECT acteur  AS personne
FROM Film
 EXCEPT
SELECT realisateur AS personne
FROM Film;
```

- sont aussi réalisateurs :

```
SELECT acteur AS personne
FROM Film
 INTERSECT
SELECT realisateur AS personne
FROM Film;
```

- Requêtes alternatives (possiblement avec duplicats) :

- ne sont pas réalisateurs :

```
SELECT acteur
FROM Film
WHERE acteur NOT IN
 (SELECT realisateur
 FROM Film);
```

- sont aussi réalisateurs :

```
SELECT acteur
FROM Film
WHERE
acteur=realisateur;
```

Plus d'exemples de requêtes imbriquées : avec EXISTS et IN

Trouver les réalisateurs dont on joue les films au Champo.

```
SELECT F.realisateur
FROM Film F
WHERE EXISTS (SELECT *
              FROM Projection P
              WHERE F.titre=P.titre
              AND P.cinema='Le Champo');
```

```
SELECT F.realisateur
FROM Film F
WHERE F.titre IN (SELECT P.titre
              FROM Projection P
              WHERE P.cinema='Le Champo');
```

Plus d'exemples de requêtes imbriquées : avec NOT IN

Trouver les acteurs qui n'ont pas joué dans un film de Kubrick

```
SELECT F.acteur
FROM Film F
WHERE F.acteur NOT IN
          (SELECT F1.acteur
           FROM Film F1
           WHERE F1.realisateur='Kubrick');
```

La sous requête trouve les acteurs qui jouent dans des films de Kubrick, les trois lignes du haut prennent le complément de cet ensemble.

Trouver les acteurs qui n'ont joué que dans un seul film :

```
SELECT F1.acteur
FROM Film F1
WHERE F1.acteur NOT IN
                (SELECT F2.acteur
                FROM Film F2
                WHERE F.1=titre <> F2.titre);
```

Solution alternative :

```
SELECT F1.acteur
FROM Film F1
WHERE NOT EXISTS
      (SELECT *
      FROM Film F2
      WHERE F1.acteur=F2.acteur and
      F1.titre <> F2.titre);
```

Trouver les acteurs qui ont joué dans un film de Polanski mais pas dans un film de Kubrick :

```
SELECT F1.acteur
FROM Film F1
WHERE F1.acteur IN
      (SELECT F2.acteur
      FROM Film F2
      WHERE F2.realisateur='Polanski')
            AND
      F1.acteur NOT IN
      (SELECT F3.acteur
      FROM Film F2
      WHERE F3.realisateur='Kubrick');
```

Trouver les acteurs qui ont joué dans un film de Polanski mais pas dans un film de Kubrick :

```
SELECT F1.acteur
FROM Film F1
WHERE F1.acteur IN
      (SELECT F2.acteur
      FROM Film F2
      WHERE F2.realisateur='Polanski')
            AND
      F1.acteur NOT IN
      (SELECT F3.acteur
      FROM Film F2
      WHERE F3.realisateur='Kubrick');
```

L'aggrégation

- Dans l'algèbre relationnelle, les conditions sont évaluées pour un tuple à la fois
- Or, parfois on s'intéresse à des propriétés dépendant d'ensembles de tuples
- Exemple : trouver le nombre de films projetés en ce moment
- Le nombre de films ou le nombre de projections de films ?
- Dans ce contexte, la question des doublons est importante.

Doublons

```
SELECT * FROM T1

A1   A2
---- ----
1    2
2    1
1    1
2    2

SELECT A1 FROM T1

A1
--
1
2
1
2
```

Doublons

- SELECT ne correspond pas exactement à l'opérateur de projection de l'algèbre relationnelle.
- La projection retourne l'ensemble $\{1, 2\}$
- SELECT conserve les doublons
- Comment omettre les doublons ? Utiliser SELECT DISTINCT

```
SELECT DISTINCT A1 FROM T1

A1
--
1
2
```

Gérer les doublons

- Jusqu'à présent dans l'algèbre relationnelle, on a opéré sur des ensembles. SQL opère en fait sur des multi-ensembles, i.e., des ensembles pouvant contenir des doublons.
- Requière de petits ajustements
- La projection retourne l'ensemble $\{1, 2\}$
- `SELECT` conserve les doublons
- Comment se débarrasser des doublons ?
- Utiliser `SELECT DISTINCT`

```
SELECT DISTINCT A1 FROM T1;
```

```
A1
--
1
2
```

Gérer les doublons

- Jusqu'à présent dans l'algèbre relationnelle, on a opéré sur des ensembles. SQL, opère en fait sur des multi-ensembles, i.e., des ensemble pouvant contenir des doublons.
- Requière de petits ajustements
- La projection π ne retire plus les doublons :

$$\pi_A \left(\begin{array}{c|c} \text{A} & \text{B} \\ \hline a_1 & b_1 \\ \hline a_2 & b_2 \\ \hline a_1 & b_2 \end{array} \right) = \{a_1, a_2, a_1\}$$

  Ici a_1 apparaît deux fois.
- Il y a une opération spéciale d'élimination des doublons : elimination doublons$(\{a1, a2, a1\}) = \{a1, a2\}$

Gérer les doublons : l'union

- L'opération d'union groupe deux multi-ensembles :

$$S = \{1, 1, 2, 2, 3, 3\}$$

$$T = \{1, 2, 2, 2, 3\}$$

$$S \cup T = \{1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3\}$$

  i.e., si *a* occure k fois dans S, et m fois dans T, alors *a* occure k + m fois dans $S \cup T$.
- Ceci ne correspond pas à l'opération `UNION` de SQL, qui élimine les doublons.
- Pour conserver les doublons, utiliser `UNION ALL` :

```
SELECT * FROM S
   UNION ALL
SELECT * FROM T;
```

Gérer les doublons : l'intersection

- L'opération d'intersection conserve le nombre d'occurrences minimal d'un élément :

$$S = \{1, 1, 2, 2, 3, 3\}$$

$$T = \{1, 2, 2, 2, 3\}$$

$$S \cap T = \{1, 2, 2, 3\}$$

  i.e., si *a* occure k fois dans S, et m fois dans T, alors *a* occure min(k + m) fois dans $S \cap T$.
- Ceci ne correspond pas à l'opération `INTERSECT` de SQL, qui élimine les doublons.
- Pour conserver les doublons, utiliser `INTERSECT ALL` :

```
SELECT * FROM S
   INTERSECT ALL
SELECT * FROM T;
```

Gérer les doublons : la différence

- L'opération de différence fonctionne comme suit :

$$S = \{1, 1, 2, 2, 3, 3\}$$

$$T = \{1, 2, 2, 2, 3\}$$

$$S - T = \{1, 3\}$$

  i.e., si *a* occure k fois dans S, et m fois dans T, alors *a* occure k - m fois dans $S - T$.
- Ceci ne correspond pas à l'opération `INTERSECT` de SQL, qui élimine les doublons.
- Pour conserver les doublons, utiliser `EXCEPT ALL` :

```
SELECT * FROM S
   EXCEPT ALL
SELECT * FROM T;
```

SQL n'est pas un langage de programmation

- Calculer 2+2 en SQL
- Etape 1 : il nous faut une table sur laquelle opérer :

```
CREATE TABLE Arbitraire (a int);
```

- 2+2 doit aller dans une clause SELECT. Il faut aussi lui donner un nom d'attribut.
- Essai :

```
SELECT 2+2 as X
FROM Arbitraire;
```

```
 X
-----------
   0 record(s) selected.
```

SQL n'est pas un langage de programmation

- Problème : il n'y avait pas de tuple dans `Arbitraire`
- Peuplons notre table :

```
INSERT INTO Arbitraire VALUES 1;
INSERT INTO Arbitraire VALUES 5;
SELECT 2+2 as X
FROM Arbitraire;
```

```
 X
 -----------
           4
           4

  2 record(s) selected.
```

SQL n'est pas un langage de programmation

- Il faut aussi éliminer les doublons...
- Et finalement :

```
SELECT DISTINCT 2+2 as X
FROM Arbitraire;
```

```
 X
-----------
           4

  1 record(s) selected.
```

Les pièges de l'ensemble vide

- Soit trois relations, S, T, R, sur le même attribut A.
- Requête : calculer $Q = R \cap (S \cup T)$
- La requête suivante a l'air d'exprimer ça correctement :

```
SELECT R.A
FROM R, S, T
WHERE R.A=S.A OR R.A=T.A;
```

- Soit $R = S = \{1\}$, $T = \emptyset$. Alors $Q = \{1\}$, mais la requête SQL produit la table vide...
- Pourquoi ?

Les pièges de l'ensemble vide

- Soit trois relations, S, T, R, sur le même attribut A.
- Requête : calculer $Q = R \cap (S \cup T)$
- La requête suivante a l'air d'exprimer ça correctement :

```
SELECT R.A
FROM R, S, T
WHERE R.A=S.A OR R.A=T.A;
```

- Soit $R = S = \{1\}$, $T = \emptyset$. Alors $Q = \{1\}$, mais la requête SQL produit la table vide...
- Pourquoi ?
- Si T est vide, alors $R \times S \times T$ est vide aussi !

Plus sur la clause WHERE

- Une fois que nous avons des types (numériques, chaînes de caractères, etc), nous avons des opérations spécifiques aux types et donc des conditions de sélection spécifiques à ces types.

```
CREATE TABLE Finance (titre char(20), budget int,
recette int);
INSERT INTO Finance VALUES ('Shining', 19, 100);
INSERT INTO Finance VALUES ('Star wars', 11, 513);
INSERT INTO Finance VALUES ('Wild wild west', 170, 80);
```

Plus sur la clause WHERE

- Trouver les films qui ont perdu de l'argent

```
SELECT titre
FROM Finance
WHERE recette < budget;
```

- Trouver les films qui ont généré au moins 10 fois plus de recette que ce qu'ils ont coûté

```
SELECT titre
FROM Finance
WHERE recette > 10 * budget;
```

- Trouver le bénéfice généré par chaque film :

```
SELECT titre, recette - budget as profit
FROM Finance
WHERE recette - budget > 0;
```

Plus sur la clause WHERE

- Est-ce que Kubrick s'écrit avec "k" ou "ck" à la fin ?
- Pas besoin de se souvenir.

```
SELECT titre, realisateur
FROM Film
WHERE realisateur LIKE 'Kubr%';
```

- Est-ce que Polanski s'écrit avec "y" ou "i" à la fin ?
- Pas besoin de se souvenir.

```
SELECT titre, realisateur
FROM film
WHERE realisateur LIKE 'Polansk_';
```

Les comparaisons avec LIKE

- Motifs d'attributs avec LIKE
- Les motifs sont construit à partir de :
  lettres
  _, qui représente n'importe quelle lettre
  %, qui représente n'importe quelle sous-chaîne, dont l'ensemble vide
- Exemples :
  addresse LIKE '%Paris%'
  le motif '_a_b_' représente cacbc, aabba, etc
  le motif %a%b_' représente ccaccbc, aaaabcbcbbd, aba, etc

Les comparaisons avec LIKE

```
SELECT titre, realisateur
FROM film
WHERE realisateur LIKE 'Polansk_';
```

retourne l'ensemble vide.

- Parce que parfois x=y est vrai, alors que x `LIKE` y est faux !
- Raison : les espaces
- `'Polanski '` = `'Polanski '` est vrai, mais `'Polanski '` `LIKE` `'Polanski '` est faux.
- Si realisateur défini comme char(10), alors `'Polanski'` est vraiment `'Polanski '` et ne correspond donc pas à `'Polanski_'`.

LIKE et les espaces

- Solution 1 : utiliser des déclaration de type varchar (ou char varying)
- Solution 2 : use 'Polansk%' comme motif
- Solution 3 : utiliser la fonction TRIM :

```
SELECT titre, realisateur
FROM Film
WHERE TRIM(TRAILING FROM realisateur) LIKE 'Polansk_';
```

- TRIM TRAILING élimine les espaces de fin (LEADING élimine les espaces de début, BOTH élimine les deux)
- Attention : tous les systèmes n'aiment pas ça...

Ajouter des attributs... vers les requêtes avec agrégation

```
ALTER TABLE Film ADD COLUMN Duree int DEFAULT 0;

UPDATE Film
SET Duree = 131
WHERE titre='Chinatown';

UPDATE Film
SET Duree = 146
WHERE titre=Shining';
```

ajoute l'attribut duree, et insère des valeurs pour cet attribut.

Ajouter des attributs... vers les requêtes avec agrégation

```
ALTER TABLE Projection ADD COLUMN heure int DEFAULT 0;

UPDATE Projection
SET heure = 18
WHERE cinema='Le Champo' AND titre='Chinatown';

INSERT INTO Film VALUES ('Le Champo', 'Chinatown', 21);
```

ajoute l'attribut heure, et insère des valeurs pour cet attribut.

Plus d'une projection par film : utiliser d'abord `UPDATE`, puis `INSERT`.

Plus d'exemples avec de l'arithmétique

Requête : je veux voir un film de Lucas. Je ne peux pas y aller avant 19h, et je veux être sortie avant 23h.

Je veux voir : les cinémas et l'heure exacte à laquelle je sortirai, si mes conditions sont satisfaites.

```
SELECT P.cinema, P.heure + (F.duree/60.0) AS heurefin
FROM Projection P, Film F
WHERE F.titre=P.titre
    AND F.realisateur='Lucas'
    AND P.heure >= 19
    AND P.heure + (F.duree/60.0) < 23;
```

Requêtes d'agrégat simple

- Compter le nombre de tuples dans Film

```
SELECT COUNT(*)
FROM Film;
```

- Additionner la durée de tous les films

```
SELECT SUM(durée)
FROM Film;
```

Les doublons et l'agrégation

- Trouver le nombre de réalisateurs, approche naïve :

```
SELECT COUNT(realisateur)
FROM Film;
```

retourne le nombre de tuples dans Film.
Raison : `SELECT` ne supprime pas les doublons.

- Requête correcte :

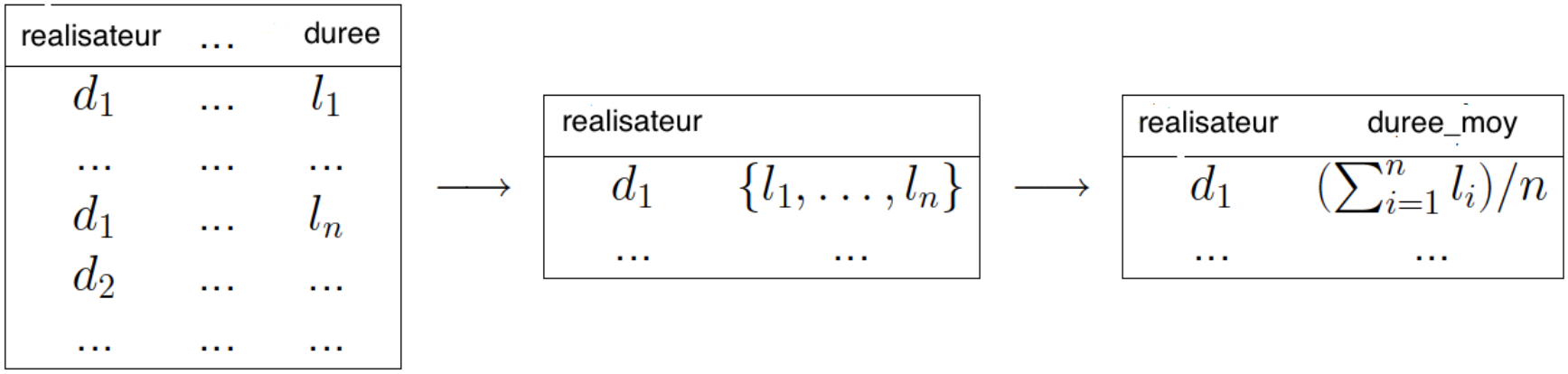
```
SELECT COUNT(DISTINCT realisateur)
FROM Film;
```

Agrégation et GROUP BY

Pour chaque réalisateur, retourner le temps moyen de ses films.

```
SELECT realisateur, AVG(duree) AS duree_moy
FROM Film
GROUP BY realisateur;
```

Comment GROUP BY fonctionne-t-il ?



(Attention : tous les attributs qui ne sont pas des agrégats et qui figurent dans la clause SELECT doivent figurer aussi dans la clause GROUP BY.)

Agrégation et doublons

Table	A1	A2	A3
	a	1	5
	a	1	2
	a	2	2
	a	2	3

```
SELECT A1, AVG(A3) as A4
FROM Table
GROUP BY A1;
```

A1	A4
a	?

Aggrégation et doublons

Une approche : prendre toutes les valeurs de A3 et calculer leur moyenne

$$\frac{5+2+2+3}{4} = 3$$

Une autre approche : seuls les attributs A1 et A3 sont pertinents

$$\pi_{\mathsf{A1,A3}} \left(\begin{array}{ccc} \mathsf{A1} & \mathsf{A2} & \mathsf{A3} \\ \hline \mathsf{a} & 1 & 5 \\ \mathsf{a} & 1 & 2 \\ \mathsf{a} & 2 & 2 \\ \mathsf{a} & 2 & 3 \end{array} \right) = \left(\begin{array}{cc} \mathsf{A1} & \mathsf{A3} \\ \hline \mathsf{a} & 5 \\ \mathsf{a} & 2 \\ \mathsf{a} & 3 \end{array} \right)$$

$$\frac{5+2+3}{3} = \frac{10}{3}$$

Agrégation et doublons

- Approche SQL : toujours garder les doublons.
- La bonne réponse est donc 3.
- Attention, cependant :

  `SELECT AVG(A2) FROM Table;`

  retourne 1
- Raison : l'arrondi
- Solution : convertir comme nombre réel :

  `SELECT AVG(CAST (A2 AS REAL)) FROM Table;`

  retourne 1.5
- Syntaxe de `CAST`

  CAST (<attribut> AS <type>)

Plus sur les doublons

- Et si on veut éliminer les doublons avant de calculer l'agrégat ?
- Utiliser `DISTINCT`

  ```
  SELECT AVG(DISTINCT A3) FROM Table;
  ```

  donne 3, à cause de l'arrondi, mais

  ```
  SELECT AVG(DISTINCT CAST (A3 AS REAL)) FROM Table;
  ```

  produit 3.3333..., comme prévu
- Un truc pour convertir les entiers en réels :

  ```
  SELECT AVG(A3 + 0.0) FROM Table;
  ```

Autres fonctions d'agrégation

- `MIN` calcule la valeur minimum d'une colonne
- `MAX` calcule la valeur maximum d'une colonne
- `SUM` additionne tous les éléments d'une colonne
- `COUNT` compte le nombre de valeurs d'une colonne
- `MIN` et `MAX` produisent le même résultat peu importe les doublons
- `SUM` additionne tous les éléments d'une colonne donnée ; `SUM DISTINCT` additionne tous les éléments distincts d'une colonne donnée
- `COUNT` compte les éléments d'une colonne donnée ;
- `COUNT DISTINCT` compte les éléments distincts d'une colonne donnée

SUM, COUNT et doublons

- `SELECT COUNT(A3) FROM Table;` donne 4
- `SELECT COUNT(DISTINCT A3) FROM Table;` donne 3
- `SELECT SUM(A3) FROM Table;` donne 12
- `SELECT SUM(DISTINCT A3) FROM Table;` donne 10
- `SELECT MIN(A3) FROM Table;` et `SELECT MIN(DISTINCT A3) FROM Table;` donnent le même résultat.
- Idem pour `MAX`.

Sélections basées sur des résultats d'agrégation

- Trouver les réalisateurs et le temps moyen de leurs films, à condition qu'ils aient réalisé au moins un film de plus de 2 heures.
- Idée : calculer deux agrégats : AVG(duree) et MAX(duree) et choisir seulement les réalisateurs pour lesquels MAX(duree) > 120.
- Syntaxe de SQL pour ça : HAVING

```
SELECT realisateur, AVG(duree+0.0)
FROM Film
GROUP BY realisateur
HAVING MAX(duree) > 120;
```

(Attention : seuls des prédicats contenant des opérateurs d'agrégation peuvent apparaître dans la clause `HAVING`)

Agrégation et jointures

- Les requêtes d'agrégation peuvent utiliser plus d'une relation.
- Pour tout cinéma montrant au moins un film de plus de 2 heures, trouver la durée moyenne des films qui y sont projetés.

```
SELECT F.cinema, AVG(CAST(F.duree AS REAL))
FROM Projection P, Film F
WHERE P.titre=F.titre
GROUP BY F.cinema
HAVING MAX(F.duree) > 120;
```

- Ce que ça veut dire : générer la jointure Film $\bowtie$ Projection, et sur cette jointure exécuter la requête d'agrégation qui calcule la moyenne.

Agrégation, jointures et doublons

- Les doublons peuvent donner lieu à des résultats inattendus.
- Deux tables :

R	A1	A2
	'a'	1
	'b'	2

S	A1	A3
	'a'	5
	'a'	7
	'b'	3

- Requête :

```
SELECT R.A1, SUM(R.A2)
FROM R, S
WHERE R.A1=S.A1 AND R.A1='a'
GROUP BY R.A1
HAVING MIN(S.A3) > 0;
```

- Résultat ?

Agrégation, jointures et doublons

- La table S n'est pas pertinente, et le résultat devrait être le même que celui de :

```
SELECT A1, SUM(A2)
FROM R
WHERE A1='a'
GROUP BY A1;
```

- Retourne ('a', 1)
- alors que la première requête retourne ('a', 2).

Agrégation, jointures et doublons

- Que se passe-t-il ?
- La requête construit d'abord la jointure $R \bowtie S$

R ⋈ S	A1	A2	A3
	'a'	1	5
	'a'	1	7
	'b'	2	3

- et exécute ensuite la partie agrégation sur la jointure :

```
SELECT A1, SUM(A2)
FROM R ⋈ S
WHERE A1='a'
GROUP BY A1
HAVING MIN(A3) > 0
```

- la réponse est donc ('a',2)

Agrégation, jointures et doublons

- Morale : attention aux doublons, même quand il n'y en a apparemment pas.

- Pour retourner ('a',1), utiliser DISTINCT :

```
SELECT R.A1, SUM(DISTINCT R.A2)
FROM R, S
WHERE R.A1=S.A1 AND R.A1='a'
GROUP BY R.A1
HAVING MIN(S.A3) > 0;
```

Agrégats dans le WHERE

- Les résultats d'un agrégat peuvent être utilisés pour faire des comparaisons hors de la clause HAVING.
- Trouver les films plus longs que le film le plus long joué actuellement :

```
SELECT F.titre
FROM Film F
WHERE F.duree > (SELECT MAX(F1.duree)
                 FROM Film F1, Projection P
                 WHERE F1.titre=P.titre);
```

Agrégats dans le WHERE

- Attention à ne pas écrire :

```
SELECT F.titre
FROM Film F
WHERE F.duree > MAX(SELECT F1.duree
                    FROM Film F1, Projection P
                    WHERE F1.titre=P.titre);
```

qui est incorrect

- A la place on peut écrire :

```
SELECT F.titre
FROM Film F
WHERE F.duree > ALL(SELECT F1.duree
                    FROM Film F1, Projection P
                    WHERE F1.titre=P.titre);
```

Agrégats dans le WHERE

- De même :
- Trouver les films plus courts qu'un film joué actuellement :

```
SELECT F.titre
FROM Film F
WHERE F.duree < (SELECT MAX(F1.duree)
                 FROM Film F1, Projection P
                 WHERE F1.titre=P.titre);
```

ou

```
SELECT F.titre
FROM Film F
WHERE F.duree < ANY(SELECT F1.duree
                    FROM Film F1, Projection P
                    WHERE F1.titre=P.titre);
```

- Ici on utilise `ANY` et non `ALL`

ALL versus ANY

- <valeur> <condition> ALL (<requête>) est vrai si :
  - le résultat de <requête> est l'ensemble vide, ou
  - pour toute <valeur1> dans le résultat de <requête>, <valeur> <condition> <valeur1> est vrai.
- Par exemple,
  $5 > ALL(\emptyset)$ est vrai ;
  $5 > ALL(\{1, 2, 3\}$ est vrai ;
  $5 > ALL(\{1, 2, 3, 4, 5, 6\}$ est faux.
- Remarque : `NOT IN` signifie la même chose que `<>ALL`.

ALL versus ANY

- <valeur> <condition> ANY (<requête>) est vrai s'il existe une <valeur1> dans le résultat de <requête>, telle que <valeur> <condition> <valeur1> est vrai.
- Par exemple,
  $5 < ANY(\emptyset)$ est faux ;
  $5 < ANY(\{1, 2, 3\}$ est faux ;
  $5 < ANY(\{1, 2, 3, 4, 5, 6\}$ est vrai.
- Remarque : `IN` signifie la même chose que `=ANY`

Agrégats dans le WHERE

- Toutes les comparaisons avec des résultats d'agrégat ne peuvent pas être remplacées par des comparaisons avec ANY et ALL.

- Est-ce qu'il y a un film dont la durée correspond au moins à 10% de la longueur totale de tous les autres films combinés ?

```
SELECT F.titre
FROM Film F
WHERE F.duree >= 0.1 * (SELECT SUM(F1.duree)
                        FROM Film F1
                        WHERE F1.titre <> F.titre);
```

GROUP BY et HAVING : principe

- `WHERE` filtre les lignes individuellement, alors que `HAVING` filtre les groupes (donc après regroupement)
- Conséquence :
  - dans la partie `HAVING` on ne met que des conditions à base d'agrégations ou à base d'attributs situés dans le `GROUP BY`
  - dans le `WHERE` les agrégations sont interdites

Jointures dans les requêtes

- Lorsqu'on a présenté la sémantique des requêtes agrégées, on a utilisé cette "requête" :

```
SELECT A1, SUM(A2)
FROM R ⋈ S
WHERE A1='a'
GROUP BY A1
HAVING MIN(A3) > 0;
```

- Ce n'est pas une requête SQL - elle utilise le $\bowtie$ de l'algèbre relationnelle, mais on peut l'écrire en SQL :

```
SELECT A1, SUM(A2)
FROM R NATURAL JOIN S
WHERE A1='a'
GROUP BY A1
HAVING MIN(A3) > 0;
```

Jointures dans les requêtes

- Tous les systèmes n'acceptent pas le `NATURAL JOIN`
- Il y a une syntaxe plus générale :

```
SELECT A1, SUM(A2)
FROM R JOIN S ON R.A1=S.A1
WHERE A1='a'
GROUP BY A1
HAVING MIN(A3) > 0;
```

- `R JOIN S ON c` calcule

$$\sigma_c(R \times S)$$

- c peut être une condition plus compliquée qu'une simple égalité entre attributs, e.g., `R.A2 > S.A3 - 4`

Jointures dans les requêtes

- Exemple : tous les couples de films différents ayant le même réalisateur

```
SELECT F1.réalisateur, F1.titre, F2.titre
FROM Film F1 JOIN Film F2 ON
(F1.réalisateur=F2.réalisateur
    AND
F1.titre <> F2.titre);
```

- Formulation alternative avec `USING` (suivi d'une liste d'attributs) :

```
SELECT réalisateur, F1.titre, F2.titre
FROM Film F1 JOIN Film F2
USING(réalisateur)
WHERE F1.titre <> F2.titre;
```

Thêta-Jointures

- Les expressions de type `R join S on c` sont souvent appelées Thêta-jointures et sont souvent inclues dans l'algèbre relationnelle :

$$R \underset{\theta}{\bowtie} S$$

- Il ne s'agit pas d'une nouvelle opération de l'algèbre relationnelle mais simplement d'une abréviation pour $\sigma_\theta(R \times S)$
- Raison pour le nom : les conditions étaient traditionnellement dénotées par θ

Jointures dans les requêtes

- Attention : la relation dont provient un attribut n'est plus claire :

```
SELECT A1, SUM(A2)
FROM R JOIN S ON R.A1=S.A1
GROUP BY R.A1;
```

- SQL proteste : la référence à la colonne "A1" est ambiguë
- `SELECT * FROM R JOIN S ON R.A1=S.A1;`

```
A1 A2          A1 A3
-- ----------- -- -----------
a            1 a            5
a            1 a            7
b            2 b            3
```

Jointures dans les requêtes

- Pour utiliser l'agrégation, il faut spécifier d'où les attributs viennent :

```
SELECT F.cinema, MAX(F.duree)
FROM Film F JOIN Projection P ON F.titre=P.titre
GROUP BY F.cinema;
```

  trouve les cinémas et la durée des films qui y sont joués
- Notez l'utilisation d'un alias à l'intérieur du JOIN
- On peut aussi donner des noms spécifiques aux jointures :

```
SELECT JC.cinema, MAX(JC.duree)
FROM (Film NATURAL JOIN Projection) AS JC;
GROUP BY JC.cinema;
```

Jointures dans les requêtes

- Les jointures peuvent vite devenir compliquées :

```
( ( R JOIN S ON <cond1> )  AS  Table1
      JOIN
      ( U JOIN V ON <cond2> )  AS  Table2
     ON <cond3> )
```

- Il faut faire attention lorsqu'on référence les tables dans les conditions, e.g., :
  - `<cond1>` peut faire référence à `R, S`, mais pas à `U, V, Table1, Table2`
  - `<cond2>` peut faire référence à `U, V`, mais pas à `R, S, Table1, Table2`
  - `<cond3>` peut faire référence à `Table1, Table2`, mais pas à `R, S, U, V`

Retour sur les sous requêtes

- Jusqu'à présent nous avons vu les sous requêtes dans le `WHERE`, et de façon limitée, dans le `FROM`.
- Mais elles peuvent apparaître partout !
- Exemple : éviter `GROUP BY`

```
SELECT DISTINCT P.cinema,
          (SELECT MAX(F.duree)
          FROM Film F
          WHERE F.titre=P.titre)
FROM Projection P;
```

Les sous requêtes

- Éviter HAVING : sous requêtes dans le WHERE

```
SELECT DISTINCT P.cinema,
          (SELECT MAX(F.duree)
          FROM Film F
          WHERE F.titre=P.titre)
FROM Projection P
WHERE (SELECT COUNT(DISTINCT titre)
       FROM Film F1
       WHERE F1.titre IN (SELECT P1.titre
                          FROM Projection P1
                          WHERE P1.cinema=P.cinema))>5;
```

  restreint la requête précédente aux cinémas montrant 6 films ou plus.
- En général le nouveau standard est très libéral quant à l'usage des sous requêtes, bien qu'il y ait des variations d'un système à l'autre.

Quelques exemples

Pour trouver le nombre de films maximum projeté par cinéma, on utilise une sous requête qui calcule le nombre de films projetés par chaque cinéma :

```
SELECT COUNT(DISTINCT titre)
FROM Film NATURAL JOIN Projection
GROUP BY cinema;
```

Requête complète :

```
SELECT MAX(nbre)
FROM (SELECT COUNT(DISTINCT titre) as nbre
  FROM Film NATURAL JOIN Projection
  GROUP BY cinema) AS S;
```

Remarque : il faut donner un nom à la sous requête lorsqu'elle est dans le FROM, même si on ne s'en sert pas.

Une fonctionnalité utile : ordonner la réponse

- `SELECT * FROM S;`

```
A1 A3
-- -----------
a             5
a             7
b             3
```

- `SELECT * FROM S ORDER BY A3;`

```
A1 A3
-- -----------
b             3
a             5
a             7
```

Une fonctionnalité utile : ordonner la réponse

- Ordre décroissant :

```
SELECT * FROM S ORDER BY A3 DESC;
```

```
A1 A3
-- -----------
a            7
a            5
b            3
```

- Ordre sur plusieurs attributs :

```
SELECT * FROM S ORDER BY A1, A3;
```

```
A1 A3
-- -----------
a            5
a            7
b            3
```

Une fonctionnalité utile : ordonner la réponse

On peut également trier au moyen d'opérations

R	A1	A2	A3
	5	1	b
	1	4	g
	2	1	e

```
SELECT * FROM R ORDER BY A1+A2;
```

A1	A2	A3
2	1	e
1	4	g
5	1	b

Une fonctionnalité utile : ordonner la réponse

On peut également trier au moyen d'opérations

R	A1	A2	A3
	5	1	b
	1	4	g
	2	1	e

Même si les attributs sur lesquelles sont effectuées les opérations ne font pas partie de la réponse

```
SELECT A3 FROM R ORDER BY A1+A2;
```

A3
e
g
b

Une fonctionnalité utile : ordonner la réponse

On peut également tronquer la réponse

R	A1	A2	A3
	5	1	b
	1	4	g
	2	1	e

```
SELECT A3 FROM R ORDER BY A1+A2 LIMIT 2;
```

A3
e
g

(Attention aux cas où le résultat n'est pas ordonné de manière unique par `ORDER BY`.)

Résultats intermédiaires

- Il existe un moyen de sauver des résultats intermédiaires afin d'y faire référence plus tard.
- On appelle ces résultats intermédiaires des vues
- Utile lorsque l'on a souvent besoin du résultat d'une certaine requête
- Syntaxe : `CREATE VIEW <nom> (<attributs>) AS <requête>`
- Exemple : besoin des cinémas, des réalisateurs dont un film est projeté là, ainsi que de la durée de ces films

```
CREATE VIEW CRD (ci, real, dur) AS
SELECT P.cinema, F.realisateur, F.duree
FROM Film F, Projection P
WHERE P.titre=F.titre;
```

Utilisation des vues

- Une fois créée, une vue peut être utilisée dans des requêtes
- Trouver les cinémas montrant des films longs (> 2 heures) d'un réalisateur dont le nom commence par "K"

```
SELECT ci
FROM CRD
WHERE dur > 120 AND real LIKE 'K%';
```

- Avantage : si la vue a été créée, ce n'est plus la peine de faire une jointure.

Un autre exemple

Trouver la somme totale des recettes générées par chaque réalisateur :

```
CREATE VIEW RealRecet (real, recettes) AS
SELECT réalisateur, SUM(recette)
FROM Film F, Finance Fi
WHERE F.titre=Fi.titre
GROUP BY réalisateur;
```

Trouver le réalisateur ayant généré le plus de recettes :

```
SELECT réalisateur, recettes
FROM RealRecet
WHERE recettes=(SELECT MAX(recettes)
                FROM RealRecet);
```

Un autre exemple

Trouver le réalisateur ayant généré le plus de recettes (formulation alternative, sans vue) :

```
SELECT réalisateur, SUM(recette)
FROM Film, Finance
WHERE F.titre=Fi.titre
GROUP BY réalisateur
HAVING SUM(recette) >= ALL(SELECT total FROM
                           (SELECT réalisateur,
                           SUM(recette) as total,
                           FROM Film, Finance
                           WHERE F.titre=Fi.titre
                           GROUP BY réalisateur) AS X);
```

Attention : toujours nommer les sous requêtes dans le `FROM`.

Un autre exemple

Attention ! Cette requête est incorrecte :

```
SELECT réalisateur, SUM(recette)
FROM Film, Finance
WHERE F.titre=Fi.titre
GROUP BY réalisateur
HAVING SUM(recette) >= ALL(SELECT SUM(recette),
                           FROM Film, Finance
                           WHERE F.titre=Fi.titre
                           GROUP BY réalisateur);
```

Si réalisateur se trouve dans le `GROUP BY`, alors il devrait se trouver également dans le `SELECT`.

Un exemple un peu compliqué

Schéma : Film(titre, année, réalisateur, pays, classement, genre, budget, producteur), Distinctions(titre, année, prix, résultat) (résultat $\in$ {gagné, nominé})

Requête : Pour chaque décennie à partir de 1950-59, calculer le pourcentage des prix gagnés par des films US.

On créé d'abord une vue pour stocker tous les films postérieurs à 1949 ayant obtenu un prix, avec leur décennie, titre, année et pays.

```
CREATE VIEW DécennieFilm AS
SELECT (CAST(année/10) AS INTEGER) AS décennie, titre, année, pays
FROM Film
WHERE (titre, année) in
          (SELECT titre, année
           FROM Distinctions
           WHERE résultat='gagné' AND année >= 1950);
```

Un exemple un peu compliqué

Maintenant que l'on dispose de la vue
DécennieFilm(décennie, titre, année, pays) on écrit :

```
SELECT décennie, (US.No*100)/Tous.No
FROM (SELECT décennie, COUNT(titre,année) as No
     FROM DécennieFilm WHERE pays='US' GROUP by décennie) US,
     (SELECT décennie, COUNT(titre, année) as No FROM DécennieFilm
     GROUPE BY décennie) Tous
WHERE Tous.décennie=US.décennie
          UNION
SELECT décennie, 0
FROM  ( (SELECT décennie FROM DécennieFilm) EXCEPT
        (SELECT décennie FROM DécennieFilm WHERE pays='US'));
```

Modifications de la base de données

- On a vu comment insérer des tuples dans la base de données : `INSERT INTO Table VALUES (...);`
- On peut aussi insérer les résultats de requêtes, tant que les attributs coincident.
- Exemple : On veut s'assurer que tout film listé dans la table Projection est bien listé dans la table Film :

```
INSERT INTO Film(titre)
        SELECT DISTINCT P.titre
                FROM Projection P
                WHERE F.titre NOT IN (SELECT titre
                                      FROM Film);
```

- Valeurs des attributs `réalisateur` et `acteur` lorsqu'un nouveau titre est inséré = valeurs par défaut, le plus souvent nulles (on verra ça plus tard).

Modifications de la base de données : suppressions

- Supposons que l'on veuille supprimer les films qui ne sont pas joués actuellement, sauf ceux de Kubrick :

```
DELETE FROM Film
WHERE titre NOT IN (Select titre FROM Projection) AND
réalisateur <> 'Kubrick';
```

- Forme générale :

```
DELETE FROM <nom de la relation>
WHERE <condition>;
```

- Les conditions s'appliquent aux tuples individuels ; tous les tuples satisfaisant la condition sont supprimés.

Modifications de la base de données : mises à jour

- Supposons que l'on ait une table `Personnel` avec deux attributs `nom` et `genre`
- On veut remplacer dans la table `Film` chaque nom X d'un réalisateur homme par 'Mr. X' :

```
UPDATE Film
SET réalisateur = 'Mr.' || réalisateur
WHERE réalisateur IN
(SELECT nom FROM Personnel WHERE genre=masculin');
```

- Ici || est la notation SQL pour la concaténation.
- Forme générale :

```
UPDATE <table> SET <valeur-assignée>
WHERE <conditions>;
```

- Les tables sont mises à jour un tuple à la fois.

Intégrité référentielle et mises à jour

- Les mises à jour peuvent créer des problèmes avec les clefs et clefs étrangères
- On a vu que les insertions peuvent violer des contraintes de clef
- La situation est plus complexe avec les clefs étrangères

```
CREATE TABLE R (a int not null, b int, primary key (a));
CREATE TABLE S (a int not null, foreign key (a) references R);
INSERT INTO R VALUES (1,1);
INSERT INTO S VALUES 1;
```

Jusqu'ici, tout va bien.

Intégrité référentielle et mises à jour

- Les mises à jour peuvent créer des problèmes avec les clefs et clefs étrangères
- On a vu que les insertions peuvent violer des contraintes de clef
- La situation est plus complexe avec les clefs étrangères

```
CREATE TABLE R (a int not null, b int, primary key (a));
CREATE TABLE S (a int not null, foreign key (a) references R);
INSERT INTO R VALUES (1,1);
INSERT INTO S VALUES 1;
```

Jusqu'ici, tout va bien. Mais l'insertion suivante provoque une erreur :

```
INSERT INTO S VALUES 2;
```

Intégrité référentielle et mises à jour

Un problème plus sérieux : les suppressions

- Tables : R(a,b), a clef primaire ; S(a,c)
- S.a clef étrangère pour R.a

S	a	c
	1	2
	2	2

R	a	c
	1	2
	2	3

- On supprime maintenant (1,2) de R, que se passe-t-il ? Possibilités :
  1. rejeter l'opération de suppression
  2. la propager à S et supprimer (1,2) de S
  3. conserver le tuple, mais sans valeur pour l'attribut a

Intégrité référentielle et mises à jour

SQL permet chacune des trois approches

1. Rejeter l'opération de suppression

```
CREATE TABLE R1 (a int not null primary key, b int);
CREATE TABLE S1 (a int, c int, foreign key (a) references r1);
INSERT INTO R1 VALUES (1,2);
INSERT INTO R1 VALUES (2,3);
INSERT INTO S1 VALUES  (1,2);
INSERT INTO S1 VALUES (2,2);
DELETE FROM R1 WHERE a=1 and b=2;
```

Erreur.. (à cause de la contrainte de clef étrangère)

Intégrité référentielle et mises à jour

2. Propager l'opération de suppression à S

```
CREATE TABLE R1 (a int not null primary key, b int);
CREATE TABLE S1 (a int, c int, foreign key (a) references r1
      on delete cascade);
INSERT INTO R1 VALUES (1,2);
INSERT INTO R1 VALUES (2,3);
INSERT INTO S1 VALUES  (1,2);
INSERT INTO S1 VALUES (2,2);
DELETE FROM R1 WHERE a=1 and b=2;
SELECT * FROM S2;
```

A	C
2	2

Intégrité référentielle et mises à jour

3. Conserver le tuple, mais sans valeur pour l'attribut a

```
CREATE TABLE R1 (a int not null primary key, b int);
CREATE TABLE S1 (a int, c int, foreign key (a) references r1
      on delete set null);
INSERT INTO R1 VALUES (1,2);
INSERT INTO R1 VALUES (2,3);
INSERT INTO S1 VALUES  (1,2);
INSERT INTO S1 VALUES (2,2);
DELETE FROM R1 WHERE a=1 and b=2;
SELECT * FROM S2;
```

A	C
-	2
2	2

Encore quelques exemples

Sous requête corrélée

- Les cinémas qui passent tous les films.

```
SELECT P.cinema
FROM Projection P
WHERE NOT EXISTS
     (SELECT F1.titre FROM FILM F1
      WHERE F1.titre NOT IN
            (SELECT F2. titre FROM Film F2
                  WHERE F2.titre=P.titre));
```

Encore quelques exemples

Jointure d'une table sur elle même (de l'importance des alias)

- Les couples de films différents qui ont le même réalisateur :

```
SELECT F1.titre, F2.titre
FROM Film F1, Film F2
WHERE F1.réalisateur=F2.réalisateur
  AND
F1.titre <> F2.titre;
```

Schéma :

- Film(titre, année, réalisateur, pays, note, genre, budget, producteur)
- Acteurs(titre, année, nom_perso, acteur)
- Distinctions(titre, année, prix, résultat)
  (résultat $\in$ {gagné, nominé})

(titre, année) clef étrangère de Acteurs et de Distinctions, référence Film

(Exemples de requêtes au tableau)