Aspects Logiques

S. Lopes, complété par L.Yeh

March 18, 2021

Content

1. Introduction
2. Importance du schéma
3. Qu'est-ce qu'un bon schéma relationnel ?
4. Tuning du Schéma
5. Extraction de CI

Introduction

- Tuning d'un système relationnel
- Intervention de plus haut niveau: tuning du schéma
- Sujets abordés
  - normalisation/partitionnement vertical
  - dénormalisation
  - inférence de contraintes d'intégrité (CI)

SECTION Importance du schéma

- Anomalies
- Espace Occupé
- Performances
- Autres

Anomalies

Schema 1: Personne(Nom, CP, Ville)

- Anomalie d'insertion: l'ajout d'une entité impose également d'ajouter une autre entité

  Example 1 *Dans le schéma 1, ajouter l'information 63000, `Clermont-Ferrand` impose d'ajouter une personne.*

- Anomalie de suppression: la suppression d'une entité provoque la suppression d'une autre

  Example 2 *Dans le schéma 1, supprimer la dernière personne d'une ville provoque la suppression de la ville.*

- Anomalie de mise à jour: un changement nécessite de modifier plusieurs tuples

  Example 3 *Changer `Clermont-Ferrand` en `Clermont` dans le schéma 1 nécessite de modifier tous les tuples où la valeur apparaît.*

Espace Occupé

```
Schema 1:  Personne(Nom, CP, Ville)
Schema 2:  Personne(Nom, CP), Ville(CP, NomVille)
```

- Supposons qu'il y ai un million de personnes et 10000 villes.
- `Personne(Nom, CP)` est en commun sur les deux schéma
- Supposons que `CP` occupe 4 octets et `Nom`, `NomVille` 50 octets
- Le schéma 1 occupe donc en plus 50 millions d'octets à cause de Ville
- Le schéma 2 occupe en plus $10000 \times (4 + 50) = 540000$ octets soit environ 49Mo de moins

Performances

- Si on a souvent besoin de connaître l'adresse d'une personne, le schéma 1 peut être plus efficace (pas de jointures)
- Le choix se fait entre une jointure sur des tables plus petites et un accès à une grosse table

Autres

- Si l'on fait beaucoup d'insertions sur le schéma 1, il y a un risque d'erreur dû aux saisies multiples

$\Rightarrow$ Nécessité de disposer d'une méthode pour concevoir un schéma

SECTION Qu'est-ce qu'un bon schéma relationnel ?

- Informellement: le Bon Sens
- Formellement: la Normalisation
- Modèlisation E/A

Informellement: le Bon Sens

- Simplicité: chaque relation devrait représenter une unique entité
- Limiter la redondance: évite les anomalies et facilite les mises à jour
- Limiter le nombre de valeurs manquantes: pose des problèmes de sémantique
- Eviter les tuples parasites dans les jointures

Example 4 *Soit la relation*

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

A	B
a_1	b_1
a_2	b_1

B	C
b_1	c_1
b_1	c_2

A	B	C
a_1	b_1	c_1
a1	b1	c2
a2	b1	c1
a_2	b_1	c_2

Formellement: la Normalisation

- Définition de CI[1]: clés (DF), clés étrangères (DI), ...
- Définition de formes normales: 1NF, 2NF, 3NF, BCNF, ...
- Algorithme pour concevoir un schéma dans une forme normale données

1. Soit R un schéma et r une relation sur R
2. Une DF sur R est une expression $X \rightarrow Y$ où $X \subseteq R$ et $Y \subseteq R$
3. Une DF $X \rightarrow Y$ est satisfaite dans r si $\forall t, t' \in r, t[X] = t'[X] \Rightarrow t[Y] = t'[Y]$
4. Une DF $X \rightarrow Y$ est non triviale si $Y \not\subseteq X$
5. $X \subseteq R$ est une super-clé si $X \rightarrow R$
6. $X \subseteq R$ est une clé si c'est une super-clé et si elle est minimale par rapport à l'inclusion

[1] propriétés vérifiées par toutes les instances d'un schéma

Formellement: la Normalisation

- Un schéma de relation R est normalisé si chaque DF non triviale $X \rightarrow Y$ vérifie "X est une clé de R" (BCNF)

  Example 5 *Soit l'ensemble de DF* $\{Nom \rightarrow CP, Nom \rightarrow NomVille, CP \rightarrow NomVille\}$. *Le schéma* `Personne(Nom, CP, NomVille)` *n'est pas normalisé car* $CP \rightarrow NomVille$ *et* `CP` *n'est pas clé de* `Personne`. *Par contre, le schéma* `Personne(Nom, CP)`, `Ville(CP, NomVille)` *est normalisé.*

- Plusieurs algorithmes existent pour obtenir un schéma normalisé à partir d'un ensemble d'attributs et d'un ensemble de CI
- Mais
  - ces algorithmes sont parfois difficiles à mettre en œuvre
  - **identifier les CI est difficile**

$\Rightarrow$ Nécessité de disposer d'une méthode plus pratique

Modèlisation E/A

- Démarche
  1. Modéliser l'application (EA, Merise, UML, ...)
  2. Transformer le modèle conceptuel en modèle logique (EA vers relationnel par exemple)
  3. Optimiser le modèle logique obtenu pour en déduire un modèle physique (dénormalisation, ...)
- Les points 1 et 2 sont assez systématiques alors que le 3ème point concerne le tuning

Modèlisation E/A

- La modélisation conceptuelle consiste à identifier les entités, leurs attributs et les liens qui les relient

Example 6

- Transformation en modèle logique
  - Application de règles (chaque entité devient une relation, . . .)

  Example 7 *Personne*(<u>*Nom*</u>, *CP*), *Ville*(<u>*CP*</u>, *Nom*), *Personne*[*CP*] $\subseteq$ *Ville*[*CP*]

- Généralement, le schéma obtenu est normalisé

SECTION Tuning du Schéma

- Normalisation et Partitionnement Vertical
- Anti-Partitionnement Vertical
- Dé-normalisation

$\Rightarrow$ Les choix de tuning vont se faire en fonction d'une charge de requêtes SQL

Normalisation et Partitionnement Vertical

- On peut avoir le choix entre plusieurs schémas normalisés

Example 8

- *Trois attributs:* `nocompte`, `adresse`, `solde`, *et une DF* *nocompte* $\rightarrow$ *adresse*, *solde*
- *Deux schémas normalisés:*
  1. (`nocompte`, `adresse`, `solde`)
  2. (`nocompte`, `adresse`), (`nocompte`, `solde`) *(partitionnement vertical)*
- *Deux types de requêtes*
  1. *accès à* `adresse` *lors de l'envoi mensuel du relevé*
  2. *accès à* `solde` *éventuellement plusieurs fois par jour*

Normalisation et Partitionnement Vertical

- Dans ce cas de figure, le deuxième schéma peut être préférable car les tuples sont plus petits
  - un index primaire non dense sur `nocompte` dans (`nocompte`, `solde`) peut avoir un niveau de moins (un pointeur par page et plus de tuples par page)
  - plus de tuples tiennent en mémoire et donc améliorent les accès aléatoires (plus de chance que le tuple soit déjà en mémoire)
  - un scan d'un nombre important de compte parcourra moins de pages
  - en contrepartie, cette solution occupe plus de place

Normalisation et Partitionnement Vertical

- On peut avoir le choix entre plusieurs schémas normalisés

Example 9

  - *Supposons que `adresse` se décompose en `CP` et `ville`*
  - *Est-ce que le schéma suivant est intéressant ? (`nocompte`, `CP`), (`nocompte`, `ville`), (`nocompte`, `solde`)*
  - *Non car `CP` et `ville` sont accédés en même temps*
  - *Les performances sont moins bonnes à cause de la jointure*
  - *Le schéma occupe plus de place*

- Un schéma à une seule relation (`X`, `Y`, `Z`) est intéressant si `X`, `Y` et `Z` sont accédés ensembles (évite une jointure et occupe moins de place)
- Un schéma à deux relation (`X`, `Y`) et (`X`, `Z`) est intéressant si les accès sont partitionnés de même et si les attributs `Y` ou `Z` (ou les deux) sont de taille importante (plus d'un tier de la taille d'une page)

Anti-Partitionnement Vertical

Example 10

1. *(nom, adresse, ...), (nom, type, tel)(type = perso, travail, ...)*
2. *(nom, adresse, tel_perso, ...), (nom, type, tel)*
   - *Le deuxième schéma a un léger surcoût en place (taille respective d'un numéro de téléphone par rapport à la taille d'un tuple)*
   - *L'accès au numéro personnel est plus performant (pas de jointure)*
   - *On peut éviter la redondance en ne mettant pas le numéro personnel dans le deuxième table (mais les requêtes peuvent être alors plus complexes)*

Dé-normalisation vs Normalisation

- **La normalisation**
  - est le processus consistant à placer chaque fait à la place la plus appropriée (un fait est stocké à un seul endroit)
  - peut limiter les performances lors des accès aux données (jointures) par contre les mises à jour sont très rapides (un seul endroit)
- **La dé-normalisation**
  - est le processus visant à introduire délibérément de la redondance dans les données (placer un fait à plusieurs endroit)
  - peut améliorer les performances en lecture au prix des performances des mises à jour
  - a beaucoup de désavantages et **une seule justification**: les performances

Faut il dénormaliser ?

- Trois questions avant de dénormaliser
  - Est-ce que les performances du système sont acceptables sans dénormaliser ?
  - Est-ce que les performances du système seront encore inacceptables après la dénormalisation ?
  - Est-ce que le système sera moins fiable à cause de la dénormalisation ?
- Si une des réponses est oui alors il ne faut pas dénormaliser
- Si on décide tout de même de dénormaliser
  1. s'il y a suffisamment de place disque, créer deux ensembles de tables (l'un normalisé, l'autre dénormalisé). Les tables dénormalisées sont peuplées à partir des tables normalisées (il faut maintenir la synchronisation). Les modifications se font sur la version normalisée alors que les accès en lecture se font sur les tables dénormalisées.
  2. sinon, conserver juste la version dénormalisée mais la maintenir par programme (utiliser des triggers par exemple)
  3. l'application doit être conçue pour pouvoir revenir à un schéma normalisé

En résumé

- La décision de dénormaliser ne doit pas être prise à la légère car elle peut provoquer des problèmes d'intégrité et induit un surplus d'administration
  - Documenter chaque dénormalisation
  - Assurer la validité et la précision des données
  - Planifier la migration des données et les propagations
  - Analyser périodiquement la base de données pour décider si la dénormalisation est toujours nécessaire

Dé-normalisation

Example 11

- *Soit le schéma normalisé suivant:* $emp(\underline{eno}, nom, dno)$, $tel(\underline{eno, type}, tel)$, $dept(\underline{dno}, nom, mgr)$
- Suite à des problèmes de performances, *on peut envisager les schémas dénormalisés suivants:*

1. *Table redondante "préjointe":* $empdept(\underline{eno}, nom, nomdept)$
2. *Colonne redondante:* $emp(\underline{eno}, nom, dno, nomdept)$
3. *Attribut multivalué: supprimer* tel, $emp(\underline{eno}, nom, dno, teltravail, teldomicile, telportable, fax)$
4. *Attribut calculé: supposons que emp possède les attributs* $sal, comm, bonus$ *alors* $emp(\underline{eno}, nom, dno, sal, comm, bonus, salcalc)$

SECTION Extraction de CI

- Introduction
- Sources d'Information
- Découverte des DI intéressantes
- Découverte des DI

Introduction

- Situation idéale
  - Dossier de conception à jour
  - CI spécifiées dans le SGBD (issus de la normalisation ou de la transformation du modèle conceptuel)
  - Données "propres"
- Cependant, en pratique
  - Tous les SGBD ne permettent pas de spécifier les CI (anciennes versions, ...)
  - Dossier de conception obsoléte
  - Evolution mal maîtrisée de la BD
  - Données erronées

Introduction

- **Objectif**: résoudre les problèmes de performance, dus à des données erronées, . . . d'une BD existante
- Comment comprendre un schéma existant ?
- On va essayer de retrouver le schéma conceptuel de la BD (rétro-conception) donc on ve devoir retrouver les CI de la BD
- A partir de quoi ?
  - programmes d'application, i.e. une charge de requêtes SQL
  - les extensions des relations
- Que cherche-t'on ?
  - Les clés (DF et DF interessantes)
  - Les clés étrangères (DI et DI interessantes)

Charge de Requêtes SQL

- Reflète ce que les utilisateurs du système accèdent

$\Rightarrow$ seules les CI *utilisées* apparaîtront

$\Rightarrow$ on extrait plutôt de cette façon les CI *interessantes*

- Avantages
  - les CI découvertes doivent être interessantes
  - moins de CI donc plus facilement manipulables
- Inconvénient
  - non exhaustif donc certaines CI peuvent manquer

Extension des Relations

- Appliquer des algorithmes de data mining pour extraire les CI
- Avantages
  - approches exhaustives
- Inconvénients
  - complexité (beaucoup de candidats)
  - nombre de résultats importants (potentiellement au moins) donc il faut pouvoir sélectionner les CI interessantes
  - **CI accidentelles: les CI sur une instance ne sont pas forcément les CI du schéma**
- On peut envisager de coupler les deux approches

Découverte des DI intéressantes

A PARTIR DI

Extraction de CI à partir d'une charge de requêtes SQL

- Préliminaires
  - Une DI sur un schéma de BD est de la forme $R_i[\overline{X}] \subseteq R_j[\overline{Y}]$ où R_i, R_j sont des schémas de relation, $\overline{X}$, $\overline{Y}$ sont des **séquences** d'attributs, $X \subseteq R_i$, $Y \subseteq R_j$ et $\overline{X}$ et $\overline{Y}$ sont compatibles ($|X| = |Y|$ et le i-ème attribut de X a le même domaine que le i-ème attribut de Y)
  - Une DI $R_i[\overline{X}] \subseteq R_j[\overline{Y}]$ est satisfaite dans une instance de BD ssi $\forall u \in r_i, \exists v \in r_j \mid u[\overline{X}] = v[\overline{Y}]$ ou $\pi_{\overline{X}}(r_i) = \pi_{\overline{Y}}(r_j)$[2]

[2] $\pi_{\overline{X}}(r)$ est un abus de notation car la projection n'est normalement pas définie pour des séquences d'attributs.

Navigation Logique

- La navigation logique est définie par rapport à une charge de requêtes SQL
- Informellement, c'est une relation entre séquences d'attributs: deux séquences d'attributs sont en relation si elles apparaissent dans une condition de jointure.
- La navigation logique est une relation binaire définie par $nav(R_i[\overline{X}], R_j[\overline{Y}]) \stackrel{def}{=} R_i[\overline{X}] \bowtie R_j[\overline{Y}] \in W$ où W est l'ensemble des conditions de jointure de la charge de requêtes
- nav est symétrique mais ni transitive, ni réflexive: on définit donc nav^* comme la fermeture transitive de nav et π_{nav^*} une partition de W selon nav^*.

Navigation Logique

Example 12

- $W = \{Ins[ssn] \bowtie Teach[ssn], Dept[dnum] \bowtie Teach[dnum], Ins[ssn] \bowtie Dept[mgr]\}$
- $\pi_{nav^*} = \{\{Ins[ssn], Teach[ssn], Dept[mgr]\}, \{Dept[dnum], Teach[dnum]\}\}$

- π_{nav^*} nous donne les DI candidates (à tester)

Example 13 $(Ins[ssn], Teach[ssn]), (Ins[ssn], Dept[mgr]), (Teach[ssn], Dept[m$
$(Dept[dnum], Teach[dnum])$ *(à tester dans les deux sens)*

Test d'une DI et Algorithme

- $d \models R_i[\overline{X}] \subseteq R_j[\overline{Y}] \Leftrightarrow |\pi_{\overline{X}}(r_i)| = |\pi_{\overline{X}}(r_i) \bowtie_{\overline{X}=\overline{Y}} \pi_{\overline{Y}}(r_j)|$
- Algorithme

Pour tout $c \in \pi_{nav^*}$
  Pour tout $R_i[\overline{X}] \in c$
    $c = c - R_i[\overline{X}]$
    Pour tout $R_j[\overline{Y}] \in c$
      $p = |\pi_{\overline{X}}(r_i) \bowtie_{\overline{X}=\overline{Y}} \pi_{\overline{Y}}(r_j)|$
      Si $|\pi_{\overline{X}}(r_i)| = p$ **alors** $I = I \cup \{R_i[\overline{X}] \subseteq R_j[\overline{Y}]\}$
      Si $|\pi_{\overline{Y}}(r_j)| = p$ **alors** $I = I \cup \{R_j[\overline{Y}] \subseteq R_i[\overline{X}]\}$
Retourner I

Application au Tuning Logique

- Si $R_i[\overline{X}] \subseteq R_j[\overline{Y}]$ et que Y est une clé de R_j alors il faut définir une clé étrangère sur R_i (bon candidat pour créer un index)
- Si $R_i[\overline{X}] \subseteq R_j[\overline{Y}]$ alors il existe peut être un problème de normalisation

Découverte des DI

Extraction de CI à partir des données

- **Problème**: Étant donné une instance de BD, trouver toutes les DI non triviales satisfaites dans cette instance

Axiomatisation

1. $R[\overline{X}] \subseteq R[\overline{X}]$ (réflexivité)
2. Si $R[\overline{A_1, \ldots, A_n}] \subseteq S[\overline{B_1, \ldots, B_n}]$ alors $R[\overline{A_{\sigma_1}, \ldots, A_{\sigma_m}}] \subseteq S[\overline{B_{\sigma_1}, \ldots, B_{\sigma_m}}]$ pour toute séquence $\sigma_1, \ldots, \sigma_m$ d'entiers distincts de $\{1, \ldots, n\}$ (projection et permutation)
3. Si $R[\overline{A_1, \ldots, A_n}] \subseteq S[\overline{B_1, \ldots, B_n}]$ et $S[\overline{B_1, \ldots, B_n}] \subseteq T[\overline{C_1, \ldots, C_n}]$ alors $R[\overline{A_1, \ldots, A_n}] \subseteq T[\overline{C_1, \ldots, C_n}]$ (transitivité)

Approche par Niveau

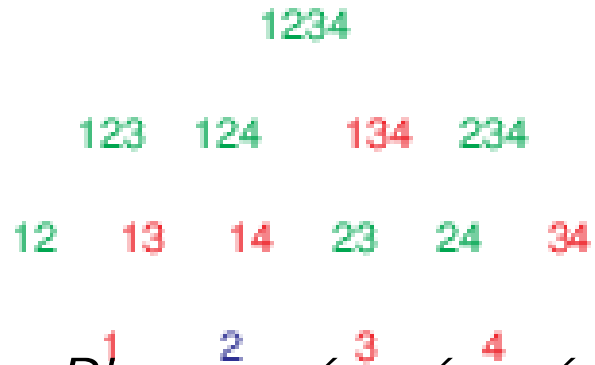
- Principe
  1. Générer les DI satisfaites de taille 1
  2. A partir de ces DI, générer les candidats de taille 2
  3. Calculer les DI satisfaites de taille 2
  4. A partir de ces DI, générer les candidats de taille 3
  5. ...
- **Intérêt**: optimisation de la recherche des motifs vérifiant un prédicat anti-monotone[3] (ou monotone[4])

[3] Si $X \subseteq Y$ et $p(Y)$ alors p(X).
[4] Si $X \subseteq Y$ et $p(X)$ alors p(Y).

Approche par Niveau

Example 14

- *On cherche les sous-ensembles de* $\{1, 2, 3, 4\}$ *qui ne contiennent pas* 2
- *Le prédicat ne pas contenir 2 est anti-monotone par rapport à l'inclusion, i.e. si* X *ne contient pas 2, tout sous-ensemble de* X *ne contient pas 2 non plus*



- *Rouge: résultats, Bleu: testé et écarté, Vert: non testé mais écarté comme sur-ensemble de 2*

Approche par Niveau

- Correction pour les DI: basée sur le deuxième axiome
- Algorithme

$i = 2$
$C_i = GenNext(DI_1)$
Tant que $C_i \neq \emptyset$ **faire**
  Pour tout $di \in C_i$ **faire**
    Si $d \models di$ **alors** $DI_i = DI_i \cup \{di\}$
  $C_{i+1} = GenNext(DI_i)$
  $i = i + 1$
Retourner $\bigcup_{j<i} DI_j$

Extraction des DI Unaires

Approche par candidat

- Généré puis tester tous les candidats de taille 1 est très couteux (si n attribut alors $O(n^2)$ *candidats* et une requête sur le BD par candidat)
- Donc peu utilisable en pratique

Approche data mining

- Idée: construire une relation binaire associant chaque valeur de la BD avec les attributs contenant cette valeur
- Contexte d'extraction (pour un type de données)
  - triplet $(V, U, \mathcal{B})$ où
  - $U = \{R.A \mid A \in R, R \text{ schéma de relation de la BD}\}$
  - $V = \{v \in \pi_A(r) \mid R.A \in U, r \in d, r \text{ définit sur } R\}$
  - $\mathcal{B} \subseteq V \times U$ telle que $(v, R.A) \in \mathcal{B} \Leftrightarrow v \in \pi_A(r)$

Extraction des DI Unaires

Example 15 *Soient les deux relations de la table:*

r	A	B
	1	X
	2	Y

s	C	D
	2	Y
	3	Z
	1	X

$\rightarrow$

v		v	
1	$AC\ (l_1)$	X	BD
2	$AC\ (l_2)$	Y	BD
3	$C\ (l_3)$	Z	D

Extraction des DI Unaires

- Propriété: $d \models C \subseteq D \Leftrightarrow D \in \bigcap_{(v,c)\in\mathcal{B}}\{E\in U|(v,E)\in\mathcal{B}\}$
- Algorithme

Pour tout $C \in U$ **faire** $rhs(C) = U$
Pour tout $v \in V$ **faire**
Pour tout $C \mid (v, C) \in \mathcal{B}$ **faire**
$rhs(C) = rhs(C) \cap \{D \mid (v, D) \in \mathcal{B}\}$
Pour tout $C \in U$ **faire**
Pour tout $D \in rhs(C), D \neq C$ **faire** $DI_1 = DI_1 \cup \{C \subseteq D\}$
Retourner DI_1

Example 16

- $rhs(A) = rhs(C) = AC$
- l_1: $rhs(A) = rhs(A) \cap \{AC\}$, *idem pour* $rhs(C)$
- l_2: *idem*
- l_3: $rhs(C) = rhs(C) \cap \{C\} = C$
- $DI_1 = \{A \subseteq C\}$

Génération des Candidats

- Basée sur l'axiome 2
- Relation de spécialisation $\prec$: $R_i[\overline{X}] \subseteq R_j[\overline{Y}] \prec R'_i[\overline{X'}] \subseteq R'_j[\overline{Y'}]$ ssi
  - $R_i = R'_i$ et $R_j = R'_j$ et,
  - $X' =< A_1, \ldots, A_k >$, $Y' =< B_1, \ldots, B_k >$ et il existe un ensemble d'indices $i_1, \ldots, i_h \in \{1, \ldots, k\}, h \leq k \mid X' =< A_{i_1}, \ldots, A_{i_h} >$, $Y' =< B_{i_1}, \ldots, B_{i_h} >$

Example 17

- $R_i[AC] \subseteq R_j[EG] \prec R_i[ABC] \subseteq R_j[EFG]$
- $R_i[AC] \subseteq R_j[GE] \not\prec R_i[ABC] \subseteq R_j[EFG]$

- Propriétés (antimonotonie de la satisfaction ou monotonie de la non satisfaction): soient I_1 et I_2 tels que $I_1 \prec I_2$
  - si $d \models I_2$ alors $d \models I_1$
  - si $d \not\models I_1$ alors $d \not\models I_2$
- Les candidats de niveau $i+1$ sont donc générés à partir des DI satisfaites de niveau i

Génération des Candidats

- Algorithme

// Phase de génération
insert into C_{i+1}
select $p.lhs.rel[p.lhs[1], ..., p.lhs[i], q.lhs[i]] \subseteq p.rhs.rel[p.rhs[1], ..., p.rhs[i], q.rhs[i]]$
from l_i p, l_i q
where $p.lhs.rel = q.lhs.rel$ and $p.rhs.rel = q.rhs.rel$
and $p.lhs[1] = q.lhs[1]$ and $p.rhs[1] = q.rhs[1]$
and ...
and $p.lhs[i] < q.lhs[i]$
// Phase d'élagage
Pour tout $l \in C_{i+1}$ **faire**
Pour tout $l' \prec l$ et l' de taille i **faire**
Si $l' \not\subseteq l_i$ **alors** $C_{i+1} = C_{i+1} \setminus \{l\}$
Retourner C_{i+1}

Génération des Candidats

Example 18

- $R_i[XA] \subseteq R_j[YC]$ *et* $R_i[XB] \subseteq R_j[YD]$ *génèrent* $R_i[XAB] \subseteq R_j[YCD]$
- $DI_2 = \{R[AB] \subseteq S[EF], R[AC] \subseteq S[EG]\}$
  1. $C_3 = \{R[ABC] \subseteq S[EFG]\}$ *(phase de génération)*
  2. $R[ABC] \subseteq S[EFG]$ *est supprimé de* C_3 *car* $R[BC] \subseteq S[FG] \notin DI_2$ *(phase d'élagage)*