Contraintes d'Intégrité dans le Modèle Relationnel

Marc Plantevit

marc.plantevit@liris.cnrs.fr

Objectifs

Structure du modèle relationnel (cours préc.)

- Permet de modéliser la réalité à un certain niveau de granularité :
  - les attributs vont décrire les objets ;
  - on peut séparer les données dans différentes relations avec des noms explicites ;
- Largement insuffisant pour représenter plus finement les différents aspects des données réelles.
- L'incapacité à représenter des *méta-données* conduit sans conteste à un certain nombre de problèmes.

Contraintes d'intégrité

- Un cadre pour ajouter une sémantique au modèle relationnel.
- Propriétés supposées être satisfaites par toutes les instances d'un schéma de bases de données.
- Ex. : numéro de sécurité sociale (clé).

Outline

Films	*Titre*	*Metteur*	*Acteur*
	Les oiseaux	Hitchcock	Hedren
	Les oiseaux	Hitchcock	Taylor
	Bladerunner	Scott	Hannah
	Apocalypse Now	Coppola	Brando

Programme	*Ciné*	*Salle*	*Titre*	*Friandise*
	Rex	1	Les oiseaux	café
	Rex	1	Les oiseaux	popcorn
	Rex	2	Bladerunner	café
	Rex	2	Bladerunner	popcorn
	ArtC	1	Les oiseaux	thé
	ArtC	1	Les oiseaux	popcorn
	Cinoche	1	Les oiseaux	Coca Cola
	Cinoche	1	Les oiseaux	vin
	Cinoche	2	Bladerunner	Coca Cola
	Cinoche	2	Bladerunner	vin

TABLE : Exemple BD

Introduction dépendances

- Le schéma en lui-même ne fait aucune restriction sur les données qui peuvent être stockées.
- Toutefois, l'application attendue de ce schéma peut impliquer plusieurs de ces restrictions.

Exemple : les dépendances fonctionnelles (df)

- Les valeurs de certains des attributs d'un tuple déterminent de façon unique, ou *fonctionnellement*, les valeurs des autres attributs du tuple.
- Un seul metteur en scène associé à chaque titre de film.

$$\mathit{Films} : \ \mathit{Titre} \rightarrow \mathit{Metteur}$$

- Dans *Programme*, un seul titre de film est associé à un couple cinéma-salle donné.

$$\mathit{Programme} : \ \mathit{Ciné}, \ \mathit{Salle} \rightarrow \mathit{Titre}.$$

Les dépendances fonctionnelles

Définition

- (syntaxe) une *Dépendance Fonctionnelle* (DF) est une expression formelle de la forme $X \rightarrow Y$
- (sémantique) une relation I satisfait la dépendance fonctionnelle $X \rightarrow Y$ *ssi* pour tout couple s, t de tuples de I avoir la même valeur sur X implique que s et t on la même valeur sur Y.

Implication logique de dépendances

Toute relation qui satisfait $Titre \rightarrow Metteur$ satisfait également $Titre, Acteur \rightarrow Metteur$

On dit que la première dépendance fonctionnelle implique logiquement (sémantiquement) la seconde.

Dépendance de clé

Définition

- Une dépendance de clé est une dépendance fonctionnelle où U est l'ensemble de tous les attributs de la relation

$$X \rightarrow U$$

- X est appelée clé de la relation
- c'est la contrepartie formelle de la notion de clé primaire.

Exemple

$$\mathit{Titre},\ \mathit{Acteur} \rightarrow \mathit{Titre},\ \mathit{Acteur},\ \mathit{Metteur}$$

est une dépendance de clé de la relation *Films*.

Dépendances de jointure (dj)

- (th, sc, ti, sn) est dans *Programme* si le cinéma *th* passe le film *ti* dans la salle *sc* et si le cinéma *th* propose la friandise *sn*.
- Intuitivement, on peut deviner une certaine **indépendance** entre les attributs *Salle*, *Titre* et l'attribut *Friandise*, pour une valeur donnée de *Ciné*.

Ex. : *(Cinoche, 1, Les oiseaux, Coca Cola)* et *(Cinoche, 2, Bladerunner, vin)* sont dans *Programme*,

$\Rightarrow$ *(Cinoche, 1, Les oiseaux, vin)* et *(Cinoche, 2, Bladerunner, Coca Cola)* devraient également être présents

Plus précisément si une relation I possède cette propriété, alors on a :

$$I = \pi_{Ciné,Salle,Titre}(I) \bowtie \pi_{Ciné,Friandise}(I).$$

C'est un exemple simple de *dépendances de jointure* (dj), exprimée formellement par :

$$Programme : \bowtie [\{Ciné, Salle, Titre\}, \{Ciné, Friandise\}].$$

- Une *dj* peut faire intervenir plus de deux ensembles d'attributs !
- Une *dépendance multivaluée (dmv)* est un cas particulier de dj qui possède au plus deux ensembles d'attributs.

Dépendances d'inclusion (di)

- Un troisième type de dépendances, appelées *dépendances d'inclusion (di)* ou « contraintes référentielles ».
- Entre deux relations.

Ex. : *Tout titre projeté actuellement (présent dans la relation* Programme*) est le titre d'un film (c'est-à-dire apparaissant dans la relation* Films*).*

$$Programme[Titre] \subseteq Films[Titre].$$

- Les di peuvent faire intervenir des *séquences* d'attributs de chaque côté.

Un mécanisme formel pour exprimer des propriétés attendues des données stockées

Si on sait que la BD satisfait un ensemble de dépendances, alors cette information peut être utilisée :

- pour améliorer la conception d'un schéma ;
- pour protéger les données en se prémunissant contre certaines mise à jour erronées ;
- pour améliorer les performances.

Conception du schéma et anomalies de mise à jour

Les dépendances sont utilisées pour fournir des informations sur la sémantique de l'application afin que le système puisse aider l'utilisateur à choisir, parmi tous les schémas possibles, le plus approprié.

Il existe plusieurs façons, pour un schéma, de ne pas être approprié.

Informations incomplètes :

Supposons qu'il faille insérer le titre d'un nouveau film et son metteur en scène sans encore connaître les acteurs.

- Impossible avec le schéma décrit précédemment. Anomalie d'insertion.
- Phénomène analogue pour les anomalies de suppression.

Redondance

- Le fait que *Coca Cola* puisse être trouvé au *Cinoche* est enregistré **plusieurs fois**.
- Supposons de plus que le directeur du *Cinoche* décide de vendre du *Pepsi Cola* au lieu du *Coca Cola*.
- Il faut modifier plusieurs tuples sinon violation de la dépendance de jointure. Anomalie de modification possible.
- Des anomalies d'insertion et de suppression sont également causées par la redondance.

Schéma plus approprié :

- La relation *Films* pourrait être « décomposée » en deux relations *M-Metteur[Titre, Metteur] et M-Acteur[Titre, Acteur]* avec
  - la df *M-Metteur* : $Titre \rightarrow Metteur$.
- *Programme* pourrait être décomposée en deux relations *ST-Programme[Ciné, Salle, Titre]* et *S-Programme[Ciné, Friandise]*, avec
  - La df *ST-Programme* : $Ciné, Salle \rightarrow Titre$.

Intégrité des données

Dépendances des données servent également de filtres aux m.a.j

- Une proposition de m.a.j conduisant à une violation d'une dépendance σ, alors la m.a.j est rejetée.
- Le système supporte les transactions.
- Durant une transaction, la BD peut être dans un état inconsistant mais, à la fin de celle-ci, le système doit vérifier l'intégrité de la BD.
- Si les dépendances sont violées, alors la transaction entière est rejetée (rollback), sinon elle est acceptée (validée – commit).

Implémentation efficace et optimisation de requêtes

- La connaissance des propriétés structurelles des données stockées est utile à l'amélioration des performances d'un système pour une application particulière.
- La satisfaction des dépendances conduit à une grande variété d'alternatives pour les structures de stockage et d'accès.

Ex. : La satisfaction d'une df ou d'une dj implique qu'une relation peut être stockée physiquement sous forme décomposée.

Ex. ++ : La satisfaction d'une dépendance de clé peut être utilisée pour réduire l'espace d'indexation.

Exemple d'optimisation

- $ans(d, a) \leftarrow Films(t, d, a'), Films(t, d', a)$ renvoie les couples (d, a), où un acteur a joue dans un film mis en scène par d. Une implémentation naïve de cette requête exige une jointure ;
- puisque *Films* satisfait $Titre \rightarrow Metteur$, cette requête peut être simplifiée en $ans(d, a) \leftarrow Films(t, d, a)$ qui peut être évaluée sans jointure ;
- en effet, lorsque $\{(t, d, a'), (t, d', a)\}$ est trouvé dans la relation *Films*, on doit avoir $d = d'$, aussi peut-on aussi bien n'utiliser que $\{(t, d, a)\}$, ce qui conduit à la requête simplifiée.

Retour sur les dépendances

- *Les dépendances fonctionnelles*
- *Les dépendances d'inclusion*
- Les dépendances de jointure
- Les dépendances multi-valuées

Outline

DF

- La forme la plus fréquemment rencontrée de dépendances.
- Sont à l'origine de l'approche par décomposition des schémas.

Classe de dépendances

On doit définir la syntaxe et la sémantique des dépendances concernées.

- La **syntaxe** : c'est le langage logique autorisé pour définir une contrainte. C'est la "*forme*" de la contrainte.
- La **sémantique** : ensemble de conditions devant être remplies pour que la contrainte soit *satisfaite* (c'est à dire juste) dans les données. C'est le *sens* de la contrainte.

Définition des dépendances fonctionnelles

Syntaxe des dépendances fonctionnelles

Une *Dépendance Fonctionnelle (DF)* sur un schéma de relation R est une expression formelle de la forme $R : X \to Y$ (ou simplement $X \to Y$ lorsque R est implicite), avec $X, Y \subseteq R$.

- Une DF $X \to Y$ est dite *triviale* si $Y \subseteq X$
- Une DF *standard* si $X \neq \emptyset$.

- A ce point, une dépendance $X \to Y$ est simplement une certaine écriture formelle : une syntaxe ;
- On pourrait aussi bien la noter $X \Rightarrow Y$, $X \spadesuit Y$ ou $X \rightsquigarrow Y$;
- On va donner donner le sens de cette écriture, sa sémantique.

Sémantique des dépendances fonctionnelles

Soit r une relation sur R. Une DF $R : X \rightarrow Y$ est *satisfaite* dans r, noté $r \models X \rightarrow Y$, *ssi*

$$\forall t_1, t_2 \in r . t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$

- Si $r \models X \rightarrow Y$, on dit aussi que X *détermine (fonctionnellement)* Y dans r.
- intuitivement, on définit une DF $X \rightarrow Y$ pour exprimer le fait que lorsqu'on connaît la valeur de X, alors on peut déterminer (en parcourant la relation) de façon certaine la valeur de Y.
- C'est l'expression du *caractère fonctionnel* de $\pi_{X,Y}(R)$.

Quel est le sens d'une dépendance *non-standard* ?
Pourquoi une dépendance *triviale* est-elle dite *triviale* ?

Si $X \rightarrow Y$ est satisfaite dans une relation r, on dit aussi que X *détermine* Y dans r. De façon intuitive, on définit une DF $X \rightarrow Y$ pour exprimer le fait que lorsqu'on connaît la valeur de X, alors on peut déterminer (en parcourant la relation) de façon certaine la valeur de Y.

La notion de *clé d'une relation* est très connue par les utilisateurs du modèle relationnel.

Clé

Une clé peut-être définie de deux manières :

- une clé est un ensemble d'attributs qui ne prend jamais deux fois la même valeur (pas de doublons dans les relations) ;
- A l'aide des DF : une clé est un ensemble d'attributs qui détermine (au sens des DF) tous les autres attributs de la relation.

Ces deux définitions sont rigoureusement équivalentes.

Une clé primaire est simplement une clé parmi les autres, choisie par le concepteur pour sa simplicité ou son aspect naturel.

Outline

Les DI se différencient des DF sur plusieurs points.

1. Peuvent être définies entre attributs de **relations différentes** et possèdent un caractère plus **global**.
2. Les DI sont définies non pas entre deux ensembles quelconques d'attributs, mais **entre deux séquences d'attributs de même taille**.

⋆ L'ordre des attributs est donc très important pour les DI ! ! !

Syntaxe

La syntaxe (forme) des DI est la suivante.

Dépendance d'inclusion

Soit **R** un schéma de base de données. Une dépendance d'inclusion sur **R** est une expression de la forme

$$R[X] \subseteq S[Y],$$

où $R, S \in \mathbf{R}$, X et Y sont des séquences d'attributs distincts respectivement de R et de S, et $|X| = |Y|$.

Sémantique

Une DI est satisfaite dans une base de données si toutes les valeurs prises par la partie gauche apparaissent dans la partie droite.

Satisfaction d'une DI

Soit $d = \{r_1, r_2, \ldots, r_n\}$ une base de données sur un schéma $\mathbf{R} = \{R_1, \ldots, R_n\}$. Une dépendance d'inclusion $R_i[X] \subseteq R_j[Y]$ sur $\mathbf{R}$ est *satisfaite* dans d, noté $d \models R_i[X] \subseteq R_j[Y]$, si $\forall t_i \in r_i, \exists t_j \in r_j \mid t_i[X] = t_j[Y]$ (de manière équivalente, $\pi_X(r_i) \subseteq \pi_Y(r_j)$).

Exemple (1)

Supposons des schémas de relation pour décrire les modules :

$$MODULE = \{\underline{NUMMODULE}; INTITULE; DESC\}$$

et un schéma de relation pour décrire les séances de cours :

$$SEANCE = \{\underline{DATE; NUMMODULE}; NUMSALLE\}$$

Pour forcer que les numéros de modules dans les séances soient bien des modules qui existent, on devra alors définir la contrainte :

Exemple (1)

Supposons des schémas de relation pour décrire les modules :

$$MODULE = \{\underline{NUMMODULE}; INTITULE; DESC\}$$

et un schéma de relation pour décrire les séances de cours :

$$SEANCE = \{\underline{DATE; NUMMODULE}; NUMSALLE\}$$

Pour forcer que les numéros de modules dans les séances soient bien des modules qui existent, on devra alors définir la contrainte :

$$SEANCE[NUMMODULE] \subseteq MODULE[NUMMODULE]$$

Exemple (2)

Supposons maintenant un schéma de relation EMPRUNTVIDEO modélisant les réservations de vidéos pour les séances, sous la forme :

$$EMPRUNTVIDEO = \{DATE, NUMMODULE; NUMPROF; NUMVIDEO\}$$

Pour assurer la cohérence de la base, on doit préciser que les vidéos doivent être réservés pour des séances existantes dans la base ; on définira alors la DI :

Exemple (2)

Supposons maintenant un schéma de relation EMPRUNTVIDEO modélisant les réservations de vidéos pour les séances, sous la forme :

$$EMPRUNTVIDEO = \{DATE, NUMMODULE; NUMPROF; NUMVIDEO\}$$

Pour assurer la cohérence de la base, on doit préciser que les vidéos doivent être réservés pour des séances existantes dans la base ; on définira alors la DI :

$$EMPRUNTVIDEO[DATE, NUMMODULE] \subseteq SEANCE[DATE, NUMMODULE]$$

Si on inverse les attributs à gauche par exemple, la DI change complètement de signification !

Une **contrainte d'intégrité référentielle** est une DI dont la partie droite est une clé (Un attribut (ou ens. d'attributs) d'une relation apparait comme clé d'une autre relation). ;

- La partie gauche d'une contrainte d'intégrité référentielle est appelée *clé étrangère*

Exemple

les DI ne définissent pas toujours des clés étrangères ! ! !

- Il suffit d'imaginer qu'on souhaite imposer que tous les cours possèdent au moins une séance dans l'année : on définira alors une DI :

$$COURS[NUMCOURS] \subseteq SEANCE[NUMCOURS]$$

- Tous les cours apparaîtront au moins une fois dans la relation des séances ;
- *NUMCOURS* n'est pas une clé de *SEANCE* (on imagine difficilement que tous les cours n'aient qu'une seule séance !)
- Donc ce n'est pas une contrainte d'intégrité référentielle.

Exemple

les DI ne définissent pas toujours des clés étrangères ! ! !

- Il suffit d'imaginer qu'on souhaite imposer que tous les cours possèdent au moins une séance dans l'année : on définira alors une DI :

$$COURS[NUMCOURS] \subseteq SEANCE[NUMCOURS]$$

- Tous les cours apparaîtront au moins une fois dans la relation des séances ;
- *NUMCOURS* n'est pas une clé de *SEANCE* (on imagine difficilement que tous les cours n'aient qu'une seule séance !)
- Donc ce n'est pas une contrainte d'intégrité référentielle.

Ce qu'il faut retenir

- Les dépendances apportent de la sémantique au modèle relationnel.
- Les différents types de dépendances.
- Les dépendances fonctionnelles :
  - définition (syntaxe),
  - satisfaction (sémantique)

  $\Rightarrow$ Etant donnée une instance r de BD, être capable de dire si r vérifie une df $X \rightarrow Y$.

  $++$ Etant donnée une instance r de BD, exhiber toutes les df valides.
- clé & cie
- dépendances d'inclusion
  - définition (syntaxe),
  - satisfaction (sémantique)

Fin de la séance.