Bases de données

Pierre Senellart, Michaël Thomazo, Paul Boniol, Serge Abiteboul
Notes par Antoine Groudiev

Version du 13 mars 2024

Table des matières

Introduction

Ce document est l'ensemble non officiel des notes du cours *Bases de données* du Département Informatique de l'ENS Ulm. Elles sont librement inspirées des notes de cours sous forme de présentation rédigées par Pierre Senellart.

1 Gestion de données, modèle relationnel, algèbre et SQL

1.1 Introduction

1.1.1 Gestion de données

Les bases de données ont de nombreuses applications (logiciels seuls, sites Web, ...), et servent à gérer des données. En particulier, elles sont utilisées pour :

— structurer des données utiles aux applications

— les conserver de manière persistante, même quand l'application ne tourne pas

— lancer des requêtes efficaces sur un large volume de données

— mettre à jour des données sans violer les contraintes structurelles de la base

— autoriser plusieurs utilisateurs à accéder et modifier les données, possiblement simultanément.

Il est par ailleurs souvent désirable d'accéder aux mêmes données depuis des applications distinctes, sur des ordinateurs différents.

On peut considérer comme exemple le système d'information d'un hôtel. Il est accessible depuis un logiciel interne (bureau d'accueil), un site web (hôtes), et une suite de logiciels de comptabilité. Il nécessite des données structurées pour les chambres, clients, réservations, ...Les données doivent rester stockées de manière pérenne, même en cas de coupure de courant. Les requêtes doivent être quasi instantanées, même avec un long historique de l'hôtel.

L'implémentation naïve nécessite un format de fichier pour stocker les données sur un disque, et un mécanisme de synchronisation, et de récupération en cas d'interruption de la synchronisation. Elle nécessite beaucoup de compétences différentes, de la programmation orientée objet, à des notions de réseau, de maîtrise de structures de données et algorithmes efficaces, de gestion en parallèle, ...Réimplémenter une telle structure à chaque fois qu'un tel outil est nécessaire est une perte de temps.

On utilise donc souvent un *Système de Gestion de Bases de Données* (SGBD), qui est un logiciel simplifiant le design d'une application manipulant des données, en fournissant un accès unifié aux fonctionnalités nécessaire pour la gestion des données, et ce peu importe l'application. Une *base de donnée* désigne une collection de données, gérée par un SGBD.

Un SGBD contient plusieurs fonctionnalités et caractéristiques :

— Indépendance physique

— Indépendance logique

— Facilité de l'accès aux données

— Optimisation des requêtes

— Intégrité logique

— Intégrité physique

— Partage de données

— Standardisation

1.1.2 Types de SGBD

De nombreux SGBD sont largement utilisées et ils ne fournissent pas tous toutes ces fonctionnalités. Les SGBDS peuvent être distingués selon le modèle de données utilisé, les choix techniques pris, la facilité d'utilisation, la possibilité de mise à l'échelle, ou encore l'architecture interne.

Les types de SGBD les plus courants sont les suivants :

— Relationnel : les données sont stockées sous forme de table ; ce modèle permet des requêtes complexes, utilisant le langage SQL

— XML : les données sont stockées sous forme d'arbre

— Graphes/Triplets

— Objets, inspiré de la POO

— Clé-valeur : très proche du fonctionnement d'une table de hachage

— Orienté colonne : modèle de donnée entre le Clé-valeur et Relationnel ; permet de parcourir très rapidement les données à l'intérieur de ce qui correspond à une colonne dans le modèle relationnel

Les SGBD ne suivant pas le modèle relationnel sont appelés `NoSQL`.

Dans le modèle relationnel, les données sont décomposées sous forme de *relation* (i.e. de tables), et utilisent un langage plus ou moins standardisé, le SQL. Les données sont stockées sur le disque, dans des tables de relation elles-mêmes stockées ligne par ligne. C'est un système centralisé, avec des propriétés de distributions limitées. Parmi les systèmes les plus répandus, on pourra citer `Oracle`, `Microsoft SQL Server`, `MySQL`, `SQLite`, et `PostgreSQL`. En TP, on utilisera la convention `PostgreSQL`.

Les SGBD relationnels classiques possèdent de nombreuses forces. Ils introduisent une indépendance entre le modèle de données et les structures de stockage, et les requêtes déclaratives et la façon dont elles sont exécutées. Ils sont capables d'exécuter des requêtes optimisées, finement optimisées, permettant de traiter jusqu'à des `To` de données. C'est une technologie mature, stable, et efficace, riche en fonctionnalités et interfaces. Le modèle relationnel permet de maintenir des contraintes d'intégrité et des transactions qui garantissent le contrôle de la concurrence des processus, l'isolation entre les utilisateurs, et la récupération en cas d'échec.

Les SGBD relationnels satisfont les propriétés suivantes, dites `ACID` :

— *Atomisme* : une transaction est soit exécutée en entier, soit pas exécutée du tout

— *Consistance* : les contraintes d'intégrité sont respectées

— *Isolation* : deux transactions concurrentes sont exécutées en série

— *Durabilité* : les données peuvent être récupérées même après un échec système

Ces propriétés viennent néanmoins également avec des faiblesses, comme leur incapacité à gérer des volumes de données extrêmement larges, de certains types, et impliquent un coût élevé des requêtes.

Les modèles `NoSQL` forment un écosystème très divers, abandonnant généralement les propriétés `ACID`, et parfois les requêtes complexes. Des nouveaux systèmes, dits `NewSQL`, essayent d'établir un compromis entre propriétés `ACID` et performances.

1.2 Modèle relationnel

On se fixe les ensembles infinis dénombrables suivants :

— $\mathcal{L}$, les étiquettes

— $\mathcal{V}$, les valeurs

— $\mathcal{T}$, les types, tel que $\forall \tau \in \mathcal{T}, \tau \subseteq \mathcal{V}$

Définition (Schéma relationnel d'arité n)**.** Un schéma relationnel d'arité n est un n-uplet $(A_1, \ldots, A_n)$ où chaque A_i est un attribut, c'est-à-dire une paire $(L_i, \tau_i) \in \mathcal{L} \times \mathcal{T}$ où les L_i sont tous distincts.

Définition (Schéma de base de données)**.** Un schéma de base de données est défini par un ensemble fini d'étiquettes $L \subseteq \mathcal{L}$ (les noms des relations), où chaque étiquette L est associée à un schéma de relation.

Exemple. *On peut par exemple considérer que $\mathcal{L}$ est l'ensemble des chaînes de caractères alphanumériques commençant par une lettre. $\mathcal{V}$ est l'ensemble des suites de bits, et $\mathcal{T}$ est formé des types* `INTEGER`*,* `REAL`*,* `TEXT`*, et* `DATA`*, ...Un schéma de base de données est formé de deux noms de relations* `Client` *et* `Reservation`*. Un* `Client` *est de la forme :*

((`id, INTEGER`*), (*`name, TEXT`*), (*`email, TEXT`*))*

et une réservation est de la forme :

((`id, INTEGER`*), (*`guest, INTEGER`*), (*`room, INTEGER`*), (*`arrival, DATE`*), (*`nights, INT.`*))*

Définition (Instance d'un schéma de relation)**.** Une *instance d'un schéma de relation* (aussi appelé une *relation sur ce schéma*) $((L_1, \tau_1), \ldots, (L_n, \tau_n))$ est un ensemble fini $\{t_1, \ldots, t_k\}$ de tuples de la forme $t_j = (v_{j1}, \ldots, v_{jn})$, où $\forall j, \forall i, v_{ij \in \tau_i}$.

Définition (Instance d'un schéma de base de données)**.** Une *instance d'un schéma de base de données* (ou plus simplement *base de donnée sur ce schéma*) est une fonction qui associe à chaque nom de relation une instance du schéma de relation correspondant.

Remarque. *Le terme* relation *est utilisé de manière ambiguë pour parler aussi bien d'un schéma de relation ou d'une instance d'un schéma de relation.*

Si $A = (L, \tau)$ est le i-ème attribut d'une relation R, et que t est le n-ième tuple d'une instance de R, on note $t[A]$ (ou $t[L]$) la valeur de la i-ème composante de t. De manière similaire, si $\mathcal{A}$ est un k-uplet d'attributs parmi les n attributs de R, $t[\mathcal{A}]$ est le k-uplet formé à partir de t en concaténant les $t[A]$ pour $A \in \mathcal{A}$.

On peut ajouter à un schéma relationnel des *contraintes d'intégrité* de natures différentes, pour définir une notion de *validité* d'une instance.

Définition (Clé)**.** Un uplet d'attributs $\mathcal{A}$ sur un schéma de relation R est une clé s'il ne peut pas exister deux uplets distincts t_1 et t_2 dans une instance de R tels que $t_1[\mathcal{A}] = t_2[\mathcal{A}]$.

Définition (Clé étrangère)**.** Un k-uplet d'attributs $\mathcal{A}$ d'un schéma de relation R est une *clé étrangère* référençant un k-uplet d'attributs $\mathcal{B}$ d'un schéma de relation S si pour toutes les instances I^R et I^S de R et S, et pour tout uplet t de I^R, il existe un *unique* tuple t' de I^S avec $t[\mathcal{A}] = t'[\mathcal{B}]$.

On peut également définir des contraintes vérifiées, c'est-à-dire une condition arbitraire sur les valeurs des attributs d'une relation (en appliquant à chaque tuple de l'instance cette relation).

Exemple. *En reprenant l'exemple précédent, on peut donner les contraintes suivantes :*

Opérateur	Arité	Description	Condition
R	0	Nom de relation	$R \in \mathcal{L}$
$\rho_{A \to B}$	1	Renommage	$A, B \in \mathcal{L}$
$\Pi_{A_1 \dots A_n}$	1	Projection	$A_1 \dots A_n \in \mathcal{L}$
σ_φ	1	Sélection	φ est une formule
$\times$	2	Produit en croix	
$\cup$	2	Union	
$\setminus$	2	Différence	
$\bowtie_\varphi$	2	Jointure	φ est une formule

FIGURE 1 – Opérateurs principaux

1.3 Langages de requêtes

1.3.1 Algèbre relationnelle

On utilise des *langages algébriques* pour exprimer des requêtes. Un expression d'algèbre relationnelle produit un nouvelle relation à partir des relations de la base de données. Chaque opérateur prend 0, 1 ou 2 sous-expressions. Les opérateurs principaux sont les suivants :

Nom de relation R Retourne la relation (table) spécifiée en argument.

Renommage $\rho_{A \to B}$ Prend une sous-expression, et renomme une colonne.

Projection $\Pi_{A_1 \dots A_n}$ Prend en argument une séquence d'attributs, et conserve en sortie simplement les attributs spécifiés de la relation donnée.

Sélection σ_φ Prend une formule de sélection pouvant être n'importe quelle combinaison booléenne de comparaisons d'attributs et de constantes, et conserve uniquement les tuples où la condition booléenne est satisfaite.

Produit cartésien $\times$ Retourne le produit cartésien ensembliste des deux ensembles obtenus par les sous-expressions.

Union $\cup$ Retourne l'union des ensembles obtenus par les sous-expressions.

Différence $\setminus$ Retourne l'ensemble obtenu par la première sous-expression, privé de l'ensemble obtenu par la seconde sous-expression. Cet opérateur ajoute de l'expressivité aux requêtes, par exemple avec des requêtes non monotones.

Jointure $\bowtie_\varphi$ Prends deux tables et une condition de jointure, et retourne les tuples où la condition de jointure est vérifiée. C'est un opérateur crucial dans la perception et l'usage du modèle relationnel, mais est en fait redondant. Il peut être vu comme une combinaison des opérateurs de renommage, produit cartésien, sélection et projection. Si R et S ont pour attributs $\mathcal{A}$ et $\mathcal{B}$, on note $R \bowtie S$ la *jointure naturelle* de R et S, où la formule de jointure est $\bigwedge_{A \in \mathcal{A} \cap \mathcal{B}} A = A$.

Toutes les expressions de l'algèbre relationnelle ne sont pas valides, et la validité d'une requête dépend en général du schéma de la base de données. Par exemple, si un nom de relation n'existe pas, si l'on essaye d'unifier deux relations avec des attributs différents, ou encore si l'on

essaye de produire une table par produit, jointure, ou renommage qui contient deux attributs avec le même nom. L'implémentation de la gestion de ces erreurs par les SGBD est très variable.

En sémantique "sac", qui est utilisée en pratique par les SGBD, toutes les opérations retournent des multiensembles, avec répétitions ; en particulier, la projection et l'union peuvent introduire des multiensembles quand des relations initiales sont définies.

De multiples extensions ont été proposées à l'algèbre relationnelle en ajoutant des fonctionnalités supplémentaires. En particulier, l'extension *agrégation et groupement* des résultats (Klug, 1982 et Libkin, 2004) est beaucoup utilisée en pratique (voir `GROUP BY` en SQL).

1.3.2 SQL

Le *Structured Query Language* (SQL), est un langage plus ou moins standardisé pour interagir avec un SGBD. Seules quelques différences syntaxiques sont remarquables d'une version à une autre.

C'est un langage *déclaratif*, c'est-à-dire qu'il n'est pas nécessaire de penser à la complexité de l'opération, simplement à ce que l'on veut. C'est le SGBD qui prend en charge l'optimisation de la requête.

Les mots-clés et identifiants sont généralement insensibles à la casse. En pratique, les mots clés sont écrits en majuscule, et les identifiants commencent par une majuscule. Les commentaires sont introduits par `--`. Dans certains contextes, on termine les commandes par `;`, mais le point-virgule ne fait pas partie de la commande, il est simplement demandé par l'environnement (par exemple, pour un client en ligne de commande).

NULL La valeur spéciale `NULL` signifie l'absence de valeur, et introduit une logique tri-valuée étrange : `True`, `False`, et `NULL`. La comparaison normale avec `NULL` renvoie toujours `NULL`. On peut vérifier la présence de valeur avec `IS NULL` ou `IS NOT NULL`.

Langage de définition de données

Contraintes

Mises à jour

Requêtes

2 Calcul relationnel, théorème de Codd, décidabilité, complexité

2.1 Calcul relationnel

Le calcul relationnel est un langage logique utilisé pour exprimer des requêtes. Il est constitué de formules logiques du premier ordre, sans symboles de fonctions, mais avec des symboles de relation sur le schéma de base de donnée (plus des prédicats de comparaison). Il approche les requêtes avec une perspective non-nommée et non typée.

2.1.1 Syntaxe

Soient $\mathcal{X}$ un ensemble de variables, un ensemble $\mathcal{V}$ de valeurs et un schéma de base de données S.

— Pour chaque relation $R \in S$ d'arité n, et pour tout $(\alpha_1, \ldots, \alpha_n) \in (\mathcal{X} \cup \mathcal{V})^n$, alors $R(\alpha_1, \ldots, \alpha_n) \in \text{FO}$.

— Prédicat d'égalité : $\alpha = \alpha' \in \text{FO}$, pour $(\alpha, \alpha') \in (\mathcal{X} \cup \mathcal{V})^2$

— Pour toutes $(\varphi_1, \varphi_2) \in \text{FO}$:

— — $\varphi_1 \wedge \varphi_2 \in \text{FO}$

— $\varphi_1 \vee \varphi_2 \in \text{FO}$

— $\not\varphi_1 \in \text{FO}$

— $\forall x \varphi_1 \in \text{FO}$

— $\exists x \varphi_1 \in \text{FO}$

Les variables libres de $\varphi \in \text{FO}$ sont les variables x apparaissant dans φ et qui ne sont pas quantifiées par $\forall x$ ou $\exists x$.

2.1.2 Sémantique

2.1.3 Sous-classes de requêtes

— Requête conjonctive (CQ, *Conjunctive query*) : requête de calcul relationnel sans $\vee, \not\forall$.

— Requête positive (PQ, *Positive query*) : requête sans $\not\forall$. Elle ne conserve que les requêtes monotones, c'est-à-dire que lorsque la base de donnée augmente en taille, la requête ne peut retourner que plus d'éléments.

— Union de requêtes conjonctives (UCQ, *Union of conjunctive queries*) : cas spécifique des requêtes positives lorsque les $\vee$ et $\wedge$ forment une DNF (Forme Normale Disjonctive).

— Requête booléenne : requête sans variable libre.

2.1.4 Théorème de Codd

Théorème (Codd, 1972)**.** *L'algèbre relationnelle et le calcul relationnel sont équivalents :*

— *Pour toute requête d'algèbre relationnelle q sur un schéma S, il existe une requête de calcul relationnel Q sur S telle que pour toute base de donnée D sur S, $q(D) = Q(D)$.*

— *Pour toute requête de calcul relationnel Q sur un schéma S, il existe une requête d'algèbre relationnelle q sur S telle que pour toute base de donnée D sur S, $q(D) = Q(D)$.*

De plus, la conversion d'un formalisme à un autre peut être fait en temps polynomial.

Ce théorème est très important, car il permet d'utiliser un formalisme déclaratif pour spécifier des requêtes logiques ou en SQL. Ces requêtes sont ensuite compilées grâce à la transformation de Codd vers un formalisme algébrique. Ces requêtes algébriques sont ensuite optimisées, en utilisant les propriétés de l'algèbre relationnelle. Les requêtes optimisées peuvent ensuite être évaluées en exploitant le fait que chaque opérateur de l'algèbre relationnelle peut facilement être implémenté.

2.1.5 Sous-classes de FO

Les requêtes conjonctives (CQ) sont équivalents aux requêtes d'algèbre relationnelle sans $\cup$ et $\setminus$, et dans lesquelles σ n'a pas de disjonction.

Les unions de requêtes conjonctives (UCQ) sont équivalentes aux requêtes positives (PQ), avec une sur-croissance exponentielle, et sont équivalentes aux requêtes algébriques sans $\setminus$.

2.2 Complexité de l'évaluation d'une requête

2.3 Analyse statique de requêtes

Conclusion

Le calcul relationnel a pour but de développer un langage de requête logique *équivalent à l'algèbre relationnelle*, ou au cœur de SQL. Grâce à celui-ci, l'évaluation de requêtes est très efficace ; cependant, il ne traite pas toutes les requêtes en temps polynomial, d'où la nécessité des langages de requêtes récursifs. L'analyse statique, souvent indécidable, est traitée en restreignant l'ensemble des requêtes aux requêtes conjonctives, à des fins d'optimisation.

3 Requêtes conjonctives

3.1 Requêtes conjonctives

3.2 Évaluation de requêtes

3.3 Hypergraphes

3.4 Analyse statique

Conclusion

4 Conception de schéma, normalisation

4.1 Le modèle entité-associations

4.2 Normalisation

5 Contraintes, dépendances fonctionnelles et multivariées

5.1 Premières définitions

On se place dans un cadre relationnel, dans une perspective nommée (par exemple, $R(A, B, C, D)$).

Définition (Dépendance fonctionnelle (FD))**.** Soit une table R. Une dépendance fonctionnelle sur R est de la forme

$$A \to B$$

où $A \subseteq \text{attr}(R)$ et $B \subseteq \text{attr}(R)$. On dit que σ est vérifiée par une instance de R lorsque

$$\forall t, t' \in R, \quad t[A] = t'[A] \implies t[B] = t'[B]$$

Intuitivement, si deux lignes ont la même valeur selon A, alors elles ont la même valeur selon B.

Remarque. *Les clés sont un exemple de dépendance fonctionnelle.*

Exemple. *Sur une table $R(A, B, C, D)$, on considère la dépendance fonctionnelle $\{A\} \to \{C\}$. Alors $R(1, 2, 3, 4)$ est correcte, mais $R(1, 2, 4, 3)$ ne peut pas être rajouté à la table tout en respectant la dépendance fonctionnelle.*

Définition (Dépendance multivariée (MVD))**.** Soit une table R, $A \subseteq \text{attr}(R)$ et $B \subseteq \text{attr}(R)$. Une dépendance fonctionnelle multivariée sur R est de la forme :

$$A \twoheadrightarrow B$$

On dit alors que σ est vérifiée par une instance de R lorsque :

$$\forall t, t', \quad t[A] = t'[A] \implies \exists t'', \begin{cases} t''[B] = t[B] \\ t''[A] = t[A] \\ t''[C] = t'[C] \end{cases}$$

où $C := \text{attr}(R) \setminus (A \cup B)$. Intuitivement, le reste des attributs est indépendant de A et B, donc toutes les autres combinaisons doivent exister.

Exemple. *La table suivante respecte la relation $A \twoheadrightarrow B$:*

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 5 & 6 \\ 1 & 2 & 5 & 6 \\ 1 & 3 & 3 & 4 \end{bmatrix}$$

5.2 Problème d'implication de contraintes

Soit Σ un ensemble de FD et MVD, et σ une FD ou une MVD. On s'intéresse au problème suivant : est-il vrai que si une instance I de R vérifie Σ, alors I vérifie σ ? On notera dans ce cas $\Sigma \vdash \sigma$.

Exemple. *Posons :*

$$\Sigma := \{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\}$$
$$\sigma := \{A\} \to \{C\}$$

On cherche à résoudre le problème d'implication de contraintes en construisant un potentiel contre-exemple. On part donc de :

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \end{bmatrix}$$

On unifie ensuite les attributs correspondant à la partie gauche de la flèche (a_2 devient a_1). En appliquant les contraintes, on transforme la table en :

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_1 & b_2 & c_1 & d_2 \\ a_1 & b_1 & c_1 & d_2 \\ a_1 & b_2 & c_1 & d_1 \end{bmatrix}$$

Cette table respecte la contrainte multivariée, donc tout Σ. Elle respecte de plus σ, donc on a bien $\Sigma \vDash \sigma$, car le processus de construction d'un contre-exemple n'a pas réussi.

Exemple. *Posons :*

$$\Sigma := \{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\}$$
$$\sigma := \{A\} \to \{C\}$$

Similairement, on part de :

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \end{bmatrix}$$

En appliquant les contraintes, on arrive à la table :

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_1 & b_2 & c_2 & d_2 \\ a_1 & b_1 & c_2 & d_2 \\ a_1 & b_2 & c_1 & d_1 \\ a_1 & b_1 & c_1 & d_2 \\ a_1 & b_1 & c_2 & d_1 \\ a_1 & b_2 & c_2 & d_1 \\ a_1 & b_2 & c_1 & d_2 \end{bmatrix}$$

Remarque. *Un ensemble Σ de dépendances exclusivement multivariées ne peut pas impliquer une dépendance simple.*

Exemple. *Posons :*

$$\Sigma := \{\{A\} \twoheadrightarrow \{B, C\}, \{C, D\} \twoheadrightarrow \{B\}\}$$
$$\sigma := \{A\} \to \{B\}$$

Similairement, on part de :

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \end{bmatrix}$$

et on arrive à :

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_1 & b_2 & c_2 & d_2 \\ a_1 & b_1 & c_1 & d_1 \\ a_1 & b_2 & c_2 & d_1 \end{bmatrix}$$

qui respecte Σ (le domaine pour A est $\{a_1\}$), mais pas σ.

5.3 EGD et TGD

Définition (Tuple Generating Dependancy (TGD))**.**

$$\forall x, \forall y, \quad \Big(B(X, y) \to \exists z, H(y, z)\Big)$$

avec B et H des conjonctions d'atomes.

Exemple. *La formule logique suivante est une TGD :*

$$\forall x_1, x_2, x_3, x_4, \quad \Big(R(x_1, x_2, x_3, x_4) \to \exists z_1, R(x_1, x_2, z_1, z_1)\Big)$$

Exemple. *On cherche une TGD représentant sur $R(A, B, C, D)$ la dépendance multivariée suivante :*

$$\{A\} \twoheadrightarrow \{C, D\}$$

On propose alors :

$$\forall x_1, x_2, x_3, x_4, x'_2, x'_3, x'_4, \quad \Big(R(x_1, x_2, x_3, x_4) \wedge R(x_1, x'_2, x'_3, x'_4) \to R(x_1, x_2, x'_3, x'_4)\Big)$$

Notons que nous n'avons pas utilisé de quantificateur existentiel. Ces derniers sont néanmoins nécessaires, par exemple pour les relations des clés étrangères.

Définition (Equality Generating Dependancy (EGD))**.**

$$\forall x, y_1, y_2, B(x, y_1, y_2) \to y_1 = y_2$$

avec B une conjonction d'atomes.

Exemple. *Sur $R(A, B, C, D)$, on cherche une EGD pour :*

$$\{A, B\} \to \{C, D\}$$

On propose :

$$\forall x, y, c_1, c_2, d_1, d_2, \quad R(x, y, c_1, d_1) \wedge R(x, y, c_2, d_2) \to c_1 = c_1 \wedge d_1 = d_2$$

Définition (Chase)**.** On considère Σ un ensemble de TGD, et σ une TGD.

Théorème. *Le problème d'implication de contraintes est indécidable.*