Ontology-Based Data Access

HAI824 – Traitement sémantique des données

ML Mugnier

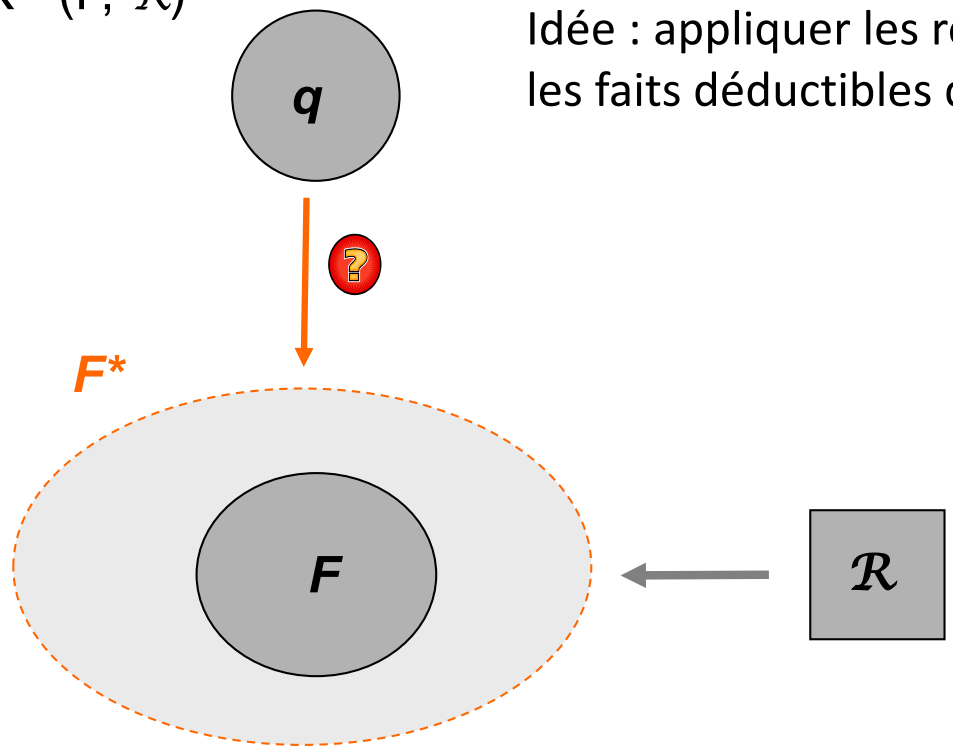
PLAN

- Deux grandes approches pour l'interrogation d'une base de connaissances
  - Saturation de la base de faits (chaînage avant)
  - Récriture de requête (forme de chaînage arrière)
- Si la base de faits provient d'une base de données
  - Matérialisation ou virtualisation de la base de faits ?
- Intégration classique de données : les mappings à la rescousse
  - Matérialisation (warehouse, Extract-Transform-Load) ou virtualisation (médiation) ?
- Ontology-Based Data Access

  ou le mariage de l'intégration de données et de la représentation de connaissances

SATURATION DE LA BASE DE FAITS (CHAÎNAGE AVANT) « bottom-up »

K= (F, $\mathcal{R}$)



Idée : appliquer les règles sur les faits pour obtenir tous les faits déductibles de la base de connaissance

F* (la saturation de F par R)
est contenue dans tous les modèles de K

Si q est une (U)CQ :

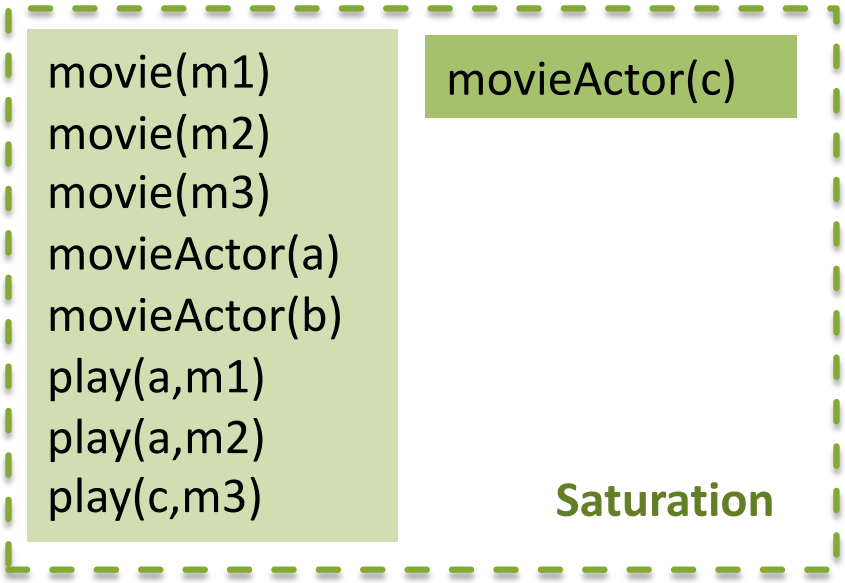
K ⊨ q ssi F* ⊨ q
ssi il y a un homomorphisme de q dans F*

\+ raisonnement effectué offline,
l'interrogation elle-même ne considère qu'une base de faits (saturée)

\- volume de la base de faits saturée
inadapté si la base de faits change fréquemment
(en particulier si on doit recalculer la saturation à chaque requête)

EXEMPLE (SATURATION)

play(x,y) ∧ movie(y) → movieActor(x)



q(x) = movieActor(x)

« find all movie actors »

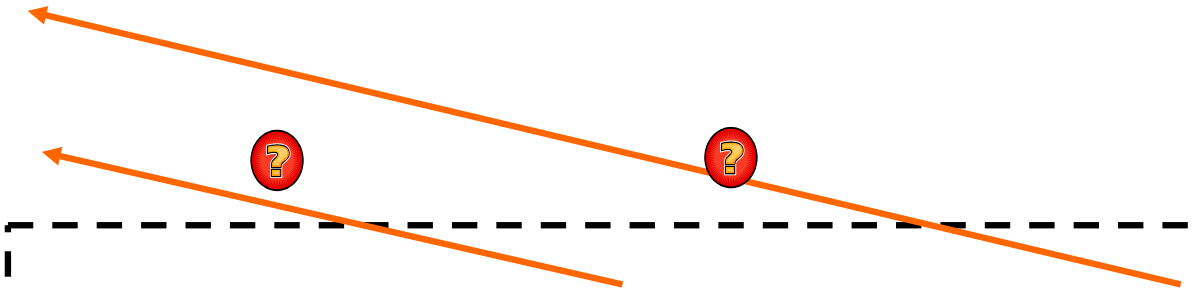
x = a
x = b
x = c

EXEMPLE (RÉÉCRITURE DE REQUÊTE)

play(x,y) ∧ movie(y) → movieActor(x)

movie(m1)
movie(m2)
movie(m3)
movieActor(a)
movieActor(b)
play(a,m1)
play(a,m2)
play(c,m3)

q(x) = movieActor(x) *« find all movie actors »*



Rewriting(q) = ∃y (movie(y) ∧ play(x, y)) ∨ movieActor(x)

{ ans(x) :- movie(y), play(x,y)
ans(x) :- movieActor(x) }

Réécriture de requête

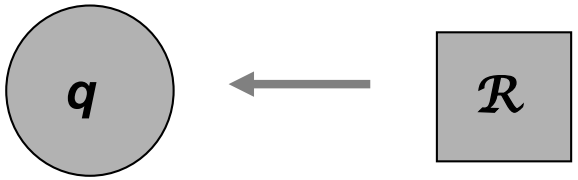
x = a	y = m1
x = a	y = m2
x = c	y = m3

x = a
x = b

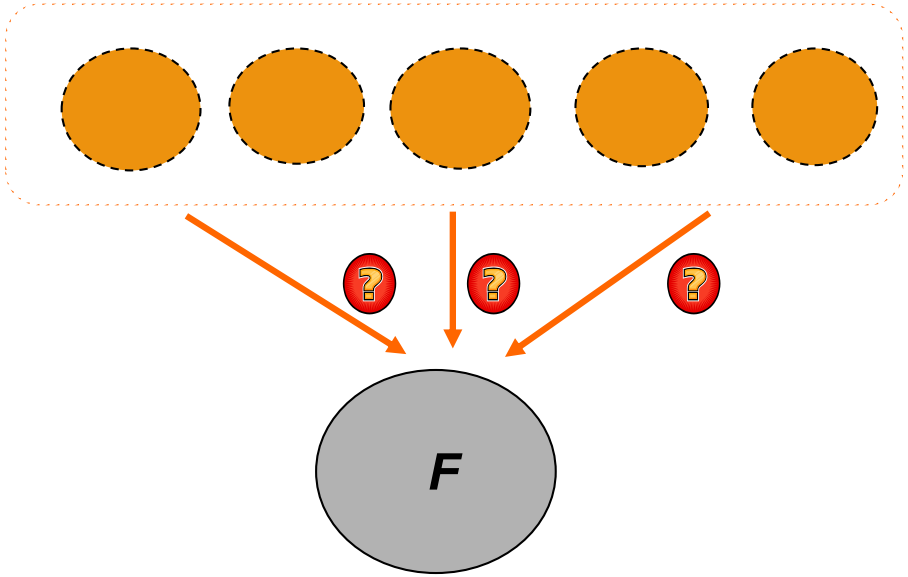
x = a
x = b
x = c

RÉÉCRITURE DE REQUÊTE (SEMI CHAINAGE ARRIÈRE) « top-down »

K= (F, $\mathcal{R}$)



Idée : chercher toutes les façons de répondre à la requête en « remontant » le long des règles



Technique essentiellement applicable aux CQ (et UCQ)

Réécriture en un ensemble de CQ vu comme une UCQ

Pour une CQ q :
K ⊨ q ssi F ⊨ rewriting(q,$\mathcal{R}$)

La réécriture de la requête est indépendante de toute base de faits

\+ processus indépendant des changements dans la base de faits

\- raisonnement pendant l'interrogation (→ plus lente)
la réécriture peut conduire à une requête complexe
(très grande et d'une forme inhabituelle pour les SGBD)

LIEN AVEC LE CHAÎNAGE ARRIÈRE ?

Rappels cours sur les règles positives en logique des propositions

Chaînage avant :

BF = {A, B, C}
BR = { R_1 : A ∧ B → E
R_2 : C ∧ E → D }

BF* = BF ∪ {E, D}

Chaînage arrière :

- principe : prouver un but (atome) en «remontant» le long des règles
- le but initial est prouvé lorsqu'on arrive à une liste de buts vide

But initial : D ?	liste de buts	{D}
Avec R_2 :		{C, E}
Avec le fait C (vu comme → C) :		{E}
Avec R_1 :		{A,B}
Avec le fait A		{B}
Avec le fait B		{}

LIENS AVEC LE CHAÎNAGE ARRIÈRE ?

Qu'est-ce qui change ?

1. Décomposition du chaînage arrière en deux étapes :
   - Réécriture avec les « vraies » règles
   - puis recherche dans la base de faits

BF = {A, B, C}

BR = { $R_1 : A \wedge B \rightarrow E$
$R_2 : C \wedge E \rightarrow D$ }

Partant de la « requête » $q_0 = D$:
- q_0 se réécrit avec R_2 : on obtient $q_1 = C \wedge E$
- q_1 se réécrit avec R_1 : on obtient $q_2 = C \wedge A \wedge B$

D'où la requête réécrite :
$q_0 \vee q_1 \vee q_2 = D \vee (C \wedge E) \vee (C \wedge A \wedge B)$

2. En logique du premier ordre, on n'a pas une simple égalité entre un atome d'une requête et une tête de règle, mais une unification de ces atomes

R: $p_2(x) \rightarrow p_1(x)$ $Q() = p_1(a)$? on réécrit : $Q'() = p_2(a)$

R: $p_2(x,y) \rightarrow p_1(x,x)$ $Q() = \exists u \exists v.\ p_1(a,u) \wedge p_3(u,v)$
on réécrit : $Q'() = p_2(a,y) \wedge p_3(a,v)$

Query Rewriting with Datalog Rules (1)

R1: p(x1,y1), p(y1,z1) → gp(x1,z1)
R2: mo(x2,y2) → p(x2,y2)
R3: fa(x3,y3) →p(x3,y3)

q(x) = gp(x,a)

A unifier *u* of atoms *A* and *B* is a substitution (of variables) such that *u(A) = u(B)*

A most general unifier (mgu) of *A* and *B* is a unifier *u* of *A* and *B* such that every other unifier of *A* and *B* can be written as ***s*** **o** ***u*** where ***s*** is a substitution

Basic step: computation of a **direct rewriting** of a CQ *q* with a rule *R*
[Important: we always assume that q and R have disjoint *sets of variables; if it is not the case, rename some variables with fresh ones]*

1. look for a mgu *u* of an atom A in *q* and the head of *R*
2. the direct rewriting of *q* with *R* according to *u* is

$$rew(q,R,u) = u(q \setminus A) \cup u(body(R))$$

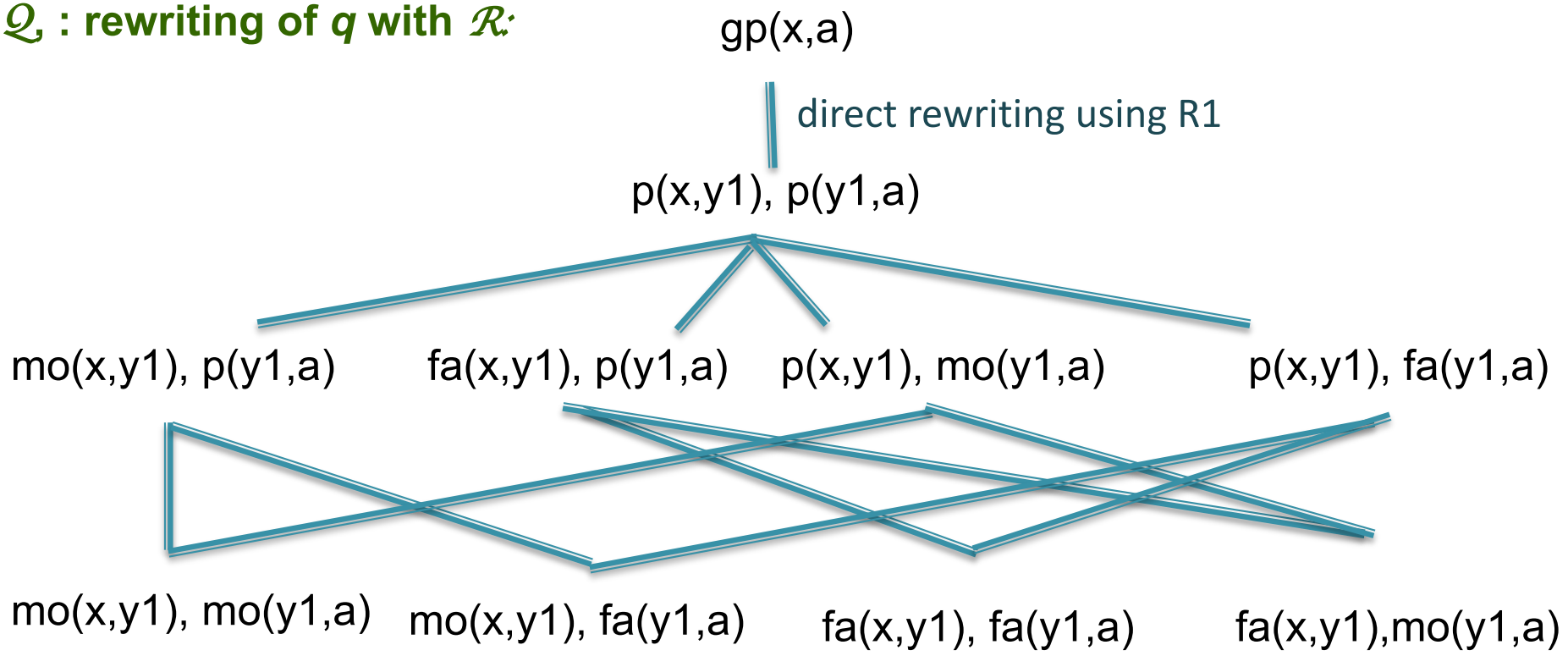
The **naive rewriting** of *q* with $\mathcal{R}$ is the set of all CQs obtained by a (possibly empty) sequence of direct rewritings starting from *q* and using the rules in $\mathcal{R}$
It is logically seen as the disjunction of all the CQs it contains

Query Rewriting with Datalog Rules (2)

R1: p(x1,y1), p(y1,z1) → gp(x1,z1)
R2: mo(x2,y2) → p(x2,y2)
R3: fa(x3,y3) →p(x3,y3)

q(x) = gp(x,a)

$\mathcal{Q}$: rewriting of *q* with $\mathcal{R}$:



Let q be a Boolean CQ and $\mathcal{Q}$ be a rewriting of q with $\mathcal{R}$
For any factbase *F*, $F, \mathcal{R} \models q$ iff $F \models \mathcal{Q}$ (where $\mathcal{Q}$ is seen as a disjonction of CQs)
iff there is q_i in $\mathcal{Q}$ such that $F \models q_i$

ALGORITHME (BASIQUE) DE RÉÉCRITURE

Récriture$(q, \mathcal{R})$: retourne l'ensemble des réécritures de q avec $\mathcal{R}$

Début

Résultat ← ø

AExplorer ← {q} *// requêtes pas encore réécrites*

Tant que AExplorer ≠ ø

Retirer q_i de AExplorer

Ajouter q_i dans Résultat

Pour toute règle $R_j \in R$

et tout unificateur le plus général u de tête(R_j) avec un atome de q_i

Calculer q_k = rewriting(q_i,R_j, u)

Ajouter q_k dans AExplorer si q_k n'est pas déjà dans AExplorer U Résultat (à un isomorphisme près)

Fin Pour

Fin Tant que

Retourner Résultat

Fin

HOMOMORPHISME / ISOMORPHISME

Soient S1 et S2 deux ensembles d'atomes

- Un homomorphisme h de S1 dans S2 est une application des variables de S1 dans les termes de S2 telle que h(S1) $\subseteq$ S2
- Un isomorphisme h de S1 dans S2 est une **bijection** des variables de S1 dans les **variables** de S2 telle que h(S1) = S2. On dit que S1 et S2 sont isomorphes

Un isomorphisme est donc un homomorphisme, mais l'inverse est faux

Propriété : un homomorphisme h de S1 dans S2 est un isomorphisme
ssi h^{-1} est un homomorphisme de S2 dans S1

Remarque : on peut avoir un homomorphisme h1 de S1 dans S2 et un homomorphisme h2 de S2 dans S1 sans que S1 et S2 soient isomorphes

S1= { p(x,y), p(x,z) }
S2= { p(u,v) }

REMOVING REDUNDANCES IN THE REWRITING (« MINIMIZATION »)

Let $\mathcal{Q} = q_1 \vee q_2$, where q_1 maps by « query-homomorphism » to q_2

i.e.: there is a homomorphism h from $q_1(x_1...x_n)$ to $q_2(y_1...y_n)$ such that $h(x_i) = y_i$ for all i

Then q_2 is useless because every answer to q_2 is also an answer to q_1 (notation : $q_2 \sqsubseteq q_1$)

Example:

$q_1(x) = \exists y.\ parent(x,y)$

$q_2(x) = \exists y.\ parent(y,x)$

$q_3(x) = \exists y.\ parent(x,y) \wedge girl(y)$

$\mathcal{Q} = q_1 \vee q_2 \vee q_3$

q_3 is useless, but not q_2

$\mathcal{Q}$ is equivalent to $q_1 \vee q_2$

REMOVING REDUNDANCES IN THE REWRITING (« MINIMIZATION »)

We can keep only a ``**cover**'' of the rewriting $\mathcal{Q}$

A **cover** of $\mathcal{Q}$ is a subset Q_c of $\mathcal{Q}$ such that:

1. for any q in $\mathcal{Q}$, there is q' in Q_c such that $q \sqsubseteq q'$

2. elements of Q_c are pairwise incomparable with respect to $\sqsubseteq$

We can also supress redundancies **inside** each conjunctive query q
(computation of the **core** of q: the minimal subset of q that is equivalent to q by query homomorphism)

Minimal UCQ : no CQ can be made smaller or removed (without losing equivalence)

If the naive rewriting of q with $\mathcal{R}$ is equivalent to a finite set of CQs
we can find it by taking one of its covers
(then minimizing each CQ if we want a minimal UCQ)

Query Rewriting can be infinite !

R = friend(u,v) ∧ friend(v,w) → friend(u,w)

No finite minimal rewriting

q = friend(Giorgos,Maria)

q_1 = friend(Giorgos, v0) ∧ friend (v0,Maria)

q_2 = friend(Giorgos, v1) ∧ friend(v1, v0) ∧ friend (v0,Maria)

$q_{2'}$ = *friend(*Giorgos*, v0) ∧ friend(v0, v1) ∧ friend (v1,Maria)*

q_2 and $q_{2'}$ are isomorphic

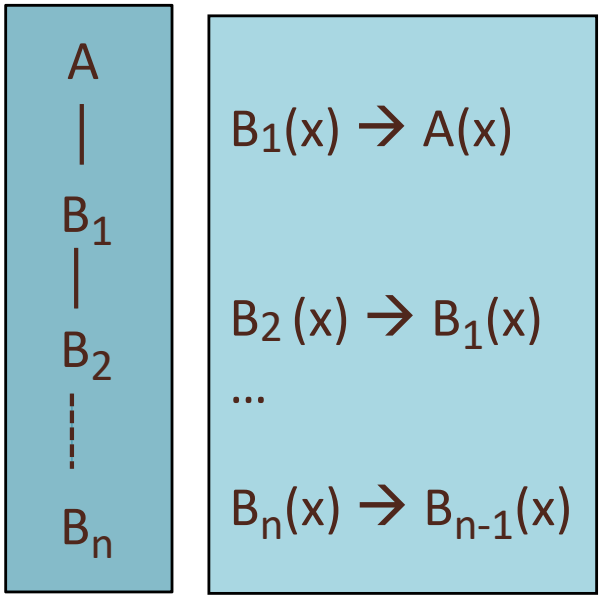
q_3 = friend(Giorgos, v2) ∧ friend(v2, v1) ∧ friend(v1, v0) ∧ friend (v1,Maria)

Etc.

- If we know the number of facts we can bound the size of a maximal « path of friends », hence number of atoms in a rewriting,
  (however, this will result in very large rewritings!)

MÊME QUAND LA RÉÉCRITURE EST FINIE, ELLE PEUT ÊTRE (TRÈS) GRANDE !

- Intérêt de l'approche : indépendance vis à vis de la base de faits
- Cependant, la taille de la réécriture peut être prohibitive en pratique



$q = A(x_1) \wedge \ldots \wedge A(x_k)$

UCQ produite : $(n+1)^k$ CQ, $k \times (n+1)^k$ atomes

Ce n'est pas un « pire des cas » théorique : se produit souvent en pratique

→ Réécriture en des formes de requêtes plus compactes (mais pas toujours plus facile à évaluer !)

$(A(x_1) \vee B_1(x_1) \ldots \vee B_n(x_1)) \wedge \ldots \wedge (A(x_k) \vee B_1(x_k) \ldots \vee B_n(x_k))$

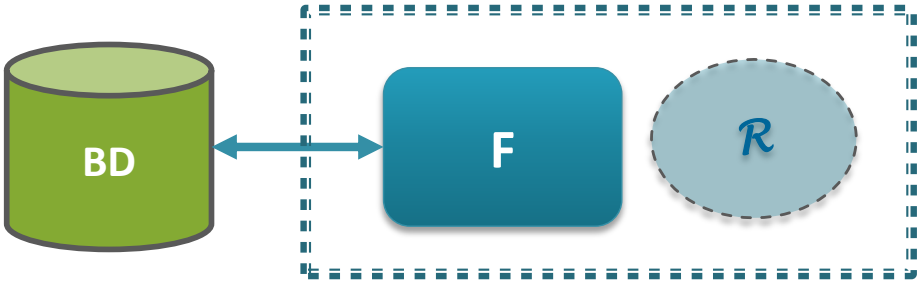
$k \times (n+1)$ atomes

→ Utilisation combinée de saturation et de réécriture de requête

QUAND LA BASE DE FAITS EST LA TRADUCTION D'UNE BASE DE DONNÉES

BD relationnelle, triplets RDFS

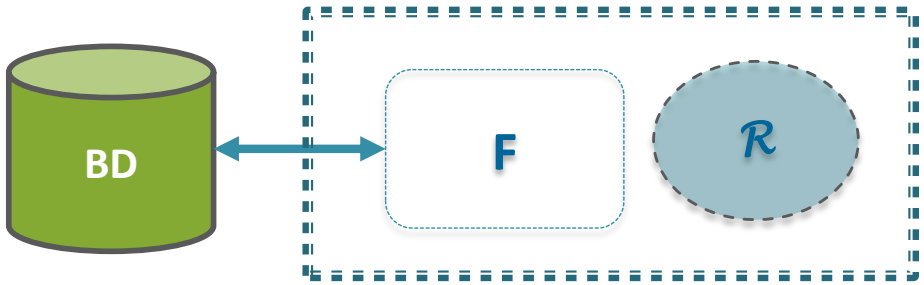
La base de faits peut être matérialisée



Interrogation avec une requête q :

- soit :
  - calcul de la saturation de F avec $\mathcal{R}$
  - évaluation de q sur F*
- soit :
  - calcul de la réécriture de q avec $\mathcal{R}$
  - évaluation de cette réécriture sur F

ou elle peut rester virtuelle



Interrogation avec une requête q :

- calcul de la réécriture de q avec $\mathcal{R}$
- traduction de cette réécriture en une requête sur la BD source
- évaluation de cette requête sur la BD

[Ceci permet de tirer parti des optimisations du SGBD]