

Partie 2 : Langages relationnels

Nom et Prénom : BEN REJEB Nour

1 - Requête 2 (SQL)

```
SELECT code, nom FROM departements WHERE prefecture = 'Bourges';
```

(a) Algèbre relationnelle

$$\pi_{code, nom} (\sigma_{prefecture='Bourges'}(departements))$$

(b) TRC

$$\{ t \mid t \in departements \wedge t.prefecture = 'Bourges' \}$$

(c) DRC

$$\{ \langle c, n \rangle \mid \exists p, r (departements(c, n, p, r) \wedge p = 'Bourges') \}$$

2 - Requête 3 (SQL)

```
SELECT code, d.nom, prefecture, r.nom FROM regions r, departements  
d WHERE r.rid = d.rid;
```

(a) Algèbre relationnelle

$$\pi_{code, d.nom, prefecture, r.nom} (regions \bowtie_{regions.rid=departements.rid} departements)$$

(b) TRC

$$\{ t \mid \exists r, d (r \in regions \wedge d \in departements \wedge r.rid = d.rid \wedge t.code = d.code \wedge$$
$$t.nomD = d.nom \wedge t.prefecture = d.prefecture \wedge t.nomR = r.nom) \}$$

(c) DRC

$$\{ \langle c, nD, p, nR \rangle \mid \exists rid, chefLieu (departements(c, nD, p, rid) \wedge regions(rid, nR, chefLieu)) \}$$

3 - Requête 5 (SQL)

```
SELECT code, d.nom, prefecture FROM regions r, departements d WHERE  
r.rid = d.rid AND r.nom = 'Centre-Val de Loire';
```

(a) Algèbre relationnelle (formulation 1)

$$\pi_{code, d.nom, prefecture} (\sigma_{r.nom='Centre-Val de Loire'}(regions) \bowtie_{regions.rid=departements.rid} departements)$$

(b) Algèbre relationnelle (formulation 2)

$\pi_{code, d.nom, prefecture}(\sigma_{r.nom='Centre-Val de Loire'}(regions \bowtie_{regions.rid=departements.rid} departements))$

Formulation la plus efficace et pourquoi

La formulation 1 est plus efficace car on filtre d'abord *regions* (sélection sur une seule région), puis on fait la jointure avec *departements*. Cela réduit la taille de la relation avant la jointure, donc moins de données à traiter.

4 - Requête 8 (algèbre relationnelle, sans tri)

On utilise l'affectation pour représenter les vues :

(a) Vue voisinsSymNoms

$$\text{voisinsSymNoms} \leftarrow \pi_{r1.nom \rightarrow Region1, r2.nom \rightarrow Region2} \left(\sigma_{r1.rid=v.rid1 \wedge r2.rid=v.rid2} (regions\ r1 \times regions\ r2 \times \text{voisinsSym}\ v) \right)$$

(b) Requête finale (compte des voisins par région)

$$\gamma_{r.nom; \text{count}(v.Region2) \rightarrow nb_voisins} \left(regions\ r \stackrel{LOJ}{\bowtie}_{r.nom=v.Region1} \text{voisinsSymNoms}\ v \right)$$

5 - Requête 14

Rappel SQL (Requête 14)

```
SELECT r.nom FROM regions r WHERE NOT EXISTS (  
  SELECT * FROM departements d WHERE d.rid = r.rid AND NOT EXISTS (  
    SELECT * FROM zus z WHERE z.departement = d.nom  
  )  
);
```

(a) TRC

L'idée : retourner les régions *r* telles que *pour tout* département *d* de *r*, il existe au moins un ZUS dans *d*.

On peut aussi dire "Les regions où il n'existe pas de départements qui n'ont aucun ZUS"

$\{ r \mid r \in regions \wedge \neg \exists d (d \in departements \wedge d.rid = r.rid \wedge \neg \exists z (z \in zus \wedge$

$z.departement = d.nom)) \}$

(b) Transformations logiques

“Tous les départements ont au moins un ZUS”

$\forall d (d.rid = r.rid \Rightarrow \exists z (z.departement = d.nom))$

équivalent à dire "Il n'existe pas de département qui n'a aucun ZUS" donc:

$\neg \exists d (d.rid = r.rid \wedge \neg \exists z (z.departement = d.nom))$

ce qui donne directement une écriture SQL avec **NOT EXISTS** imbriqués.