# 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**

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

**(b) TRC**

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

**(c) DRC**

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

**2 - Requête 3 (SQL)**

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

**(a) Algèbre relationnelle**

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

**(b) TRC**

$\{\, t \mid \exists r, d\ (r \in \text{regions} \wedge d \in \text{departements} \wedge r.rid = d.rid \wedge t.code = d.code \wedge$

$t.nomD = d.nom \wedge t.prefecture = d.prefecture \wedge t.nomR = r.nom) \,\}$

**(c) DRC**

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

**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)**

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

**(b) Algèbre relationnelle (formulation 2)**

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

**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**

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

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

$\gamma_{r.nom;\ \text{count}(v.Region2) \to nb\_voisins}\Big(\text{regions } r \overset{LOJ}{\bowtie}_{r.nom=v.Region1} \text{voisinsSymNoms } v\Big)$

**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 $d$.

$\{\, r \mid r \in \text{regions} \wedge \neg\exists d\ (d \in \text{departements} \wedge d.rid = r.rid \wedge \neg\exists z\ (z \in \text{zus} \wedge z.departement = d.nom))\,\}$

**(b) Transformations logiques**

"Tous les départements ont au moins un ZUS"

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

équivaut à

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

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