Outils Informatiques pour les Sciences

Module 5. Base de données

Partie 2 : interrogation

Dr. Etienne Rivière
etienne.riviere@unine.ch
http://iiun.unine.ch/

Assistants : Maria Carpen-Amarie

Institut d'informatique
Université de Neuchâtel

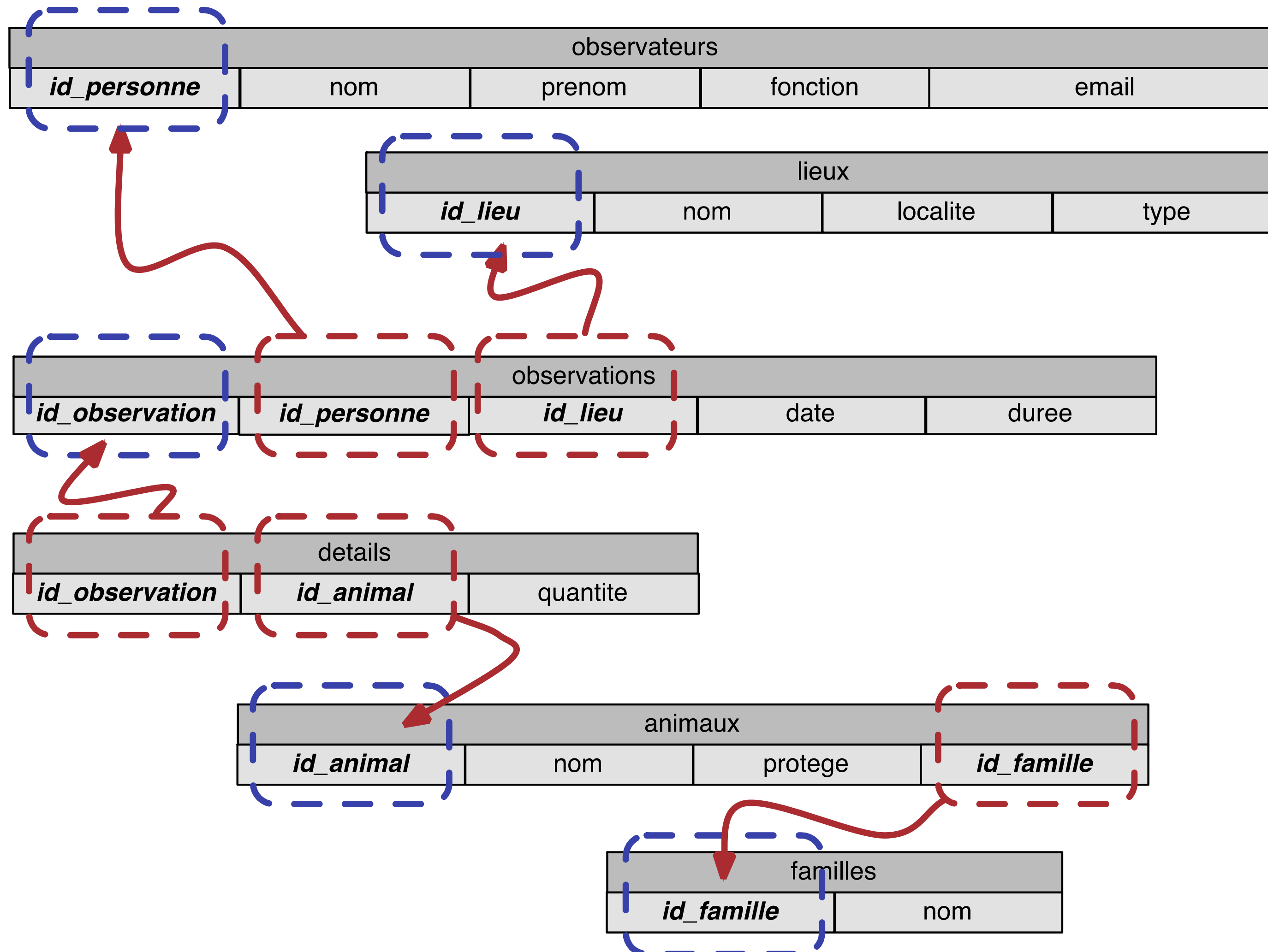
Objectifs de ce module

- Introduire les principes des bases de données relationnelles
- Apprendre à modéliser une représentation du réel sous forme de relations avec le modèle *entité-association*
- Apprendre à créer la base sous *Access* et à ajouter des données dans la base
- Apprendre à interroger une base de données avec les opérateurs relationnels du *langage SQL*

Organisation de la séance 2

- Récupération de la base "observation de la nature" sur Moodle, copie sur votre espace personnel
- Cours partie 1 : Requêtes SQL sur *une* table
- Cours partie 2 : Requêtes sur *plusieurs* tables
- Exercices dirigés

La base "Observation de la Nature"



Le langage SQL

- « *Structured Query Language* » :
  - Langage d'interrogation structuré
- Le langage standard de gestion de base de données depuis les années 1970, standardisé dans les années 1990
  - Supporté par tous les SGBD, à quelques variantes de syntaxe prêt (Access, Oracle, MySQL, …)
- Le langage SQL permet
  - La manipulation des tables et des contraintes d'intégrité
  - La manipulation des données (insertion, suppression)
  - L'interrogation de la base

Format d’une requête SQL

- Une requête d'interrogation est toujours de la même forme
  - Succession de clauses indiquant quoi, où et comment sélectionner des lignes de données

```
SELECT col1, col2 FROM table;
```

La requête est toujours terminée par un point-virgule

Une requête commence toujours par SELECT

Les colonnes à extraire (* : toutes)

La clause FROM indique la table source (ici, table). Elle est obligatoire

La clause facultative WHERE permet de donner un critère de sélection des lignes sous la forme d’une *expression*

```
SELECT * FROM table WHERE expression;
```

Projection

```
SELECT prenom, nom FROM observateurs;
```

- *Projette* la table sur les colonnes sélectionnées
  = conserve seulement les colonnes indiquées

observateurs				
id_personne	nom	prenom	fonction	email
1	Sandoz	Christine	assistant-étudiant	christine.sandoz@unine.ch
2	L'Eplatennier	Marie	doctorant	marie.lp@gmail.com

SELECT prenom, nom FROM observateurs	
prenom	nom
Christine	Sandoz
Marie	L'Eplatennier

Sélection

```
SELECT * FROM lieux
WHERE type="Foret";
```

- Les lignes de `lieux` pour lesquelles la valeur dans la colonne `type` a la valeur « Foret » sont sélectionnées
- Wildcard `*` = on conserve toutes les colonnes

lieux			
id_lieu	nom	localite	type
1	Lac des taillères	La Brévine	Lac
2	Les Planchettes	La Chaux-de-Fond	Foret

SELECT * FROM lieux WHERE type = "Foret"			
id_lieu	nom	localite	type
2	Les Planchettes	La Chaux-de-Fond	Foret

Sélection et projection

```
SELECT nom FROM lieux
WHERE type="Foret";
```

lieux			
id_lieu	nom	localite	type
1	Lac des taillères	La Brévine	Lac
2	Les Planchettes	La Chaux-de-Fond	Foret

SELECT nom FROM lieux WHERE type = "Foret"
nom
Les Planchettes

Critères de sélection

- <> : différent de

```
SELECT prenom, nom, fonction
  FROM observateurs
  WHERE fonction<>"assistant-étudiant";
```

observateurs				
id_personne	nom	prenom	fonction	email
1	Sandoz	Christine	assistant-étudiant	christine.sandoz@unine.ch
2	L'Eplatennier	Marie	doctorant	marie.lp@gmail.com

SELECT prenom, nom, fonction FROM observateurs WHERE fonction<>"assistant-etudiant"		
prenom	nom	fonction
Marie	L'Eplatennier	doctorant

Critères de sélection

```sql
SELECT id_observation FROM observations
WHERE duree >= 100;
```

- Critères numériques : <, >, <=, >=, <>, =

observations				
id_observation	***id_personne***	***id_lieu***	date	duree
1	1	1	1/11/2013	135
2	2	2	22/10/2013	68

SELECT id_observation FROM observations WHERE duree >= 100;
id_observation
1

Critères de sélection

- `LIKE` : critère sur chaîne de caractères
  - Utilise les *wildcards* (caractères génériques)
  - `*` remplace n'importe quelle suite de symboles (même vide)
  - `?` remplace un seul symbole

```
SELECT prenom, nom
  FROM observateurs
  WHERE nom LIKE "Du%";

  SELECT prenom, nom
    FROM observateurs
    WHERE nom LIKE "Dupon?";
```

Critères de sélection

- Inverser un critère : mot-clé `NOT`
- Trouver les personnes dont l'email n'est pas du domaine `unine.ch` :

```
SELECT prenom, nom
  FROM observateurs
  WHERE email NOT LIKE "%@unine.ch";
```

Critères de sélection

- Pour tester une condition vrai/faux, la valeur de la colonne est une expression elle même

```
SELECT nom
FROM animaux
WHERE protege;
```

```
SELECT nom
  FROM animaux
  WHERE NOT protege;
```

L'absence de valeur et le mot-clé `null`

- Une colonne non-obligatoire peut n'avoir aucune valeur
  - L'absence de valeur est représentée par le mot clé `null`
  - **Attention :**
    la chaîne vide, si autorisée, est `""` qui n'est pas `null`
  - **Attention :** `null` ne peut être comparé à rien, pas même à lui même. On doit utiliser «`IS null`» et pas «`= null`»

```
SELECT prenom, nom
  FROM observateurs
  WHERE email IS null;
```

Sélection composée

```
SELECT * FROM observateurs
WHERE (nom LIKE "Dupon%")
  AND (email LIKE "%@unine.ch");
```

- Combinaison de critères avec des opérateurs logiques
- Les parenthèses sont importantes
- Inversion d'un critère avec `NOT`
  - `WHERE (A) AND NOT (B)`

A OR B (ou logique)

		condition A	
		VRAI	***FAUX***
condition B	***VRAI***	VRAI	VRAI
	FAUX	VRAI	FAUX

A AND B (et logique)

		condition A	
		VRAI	***FAUX***
condition B	***VRAI***	VRAI	FAUX
	FAUX	FAUX	FAUX

A XOR B (ou exclusif)

		condition A	
		VRAI	***FAUX***
condition B	***VRAI***	FAUX	VRAI
	FAUX	VRAI	FAUX

Sélection sans doublon

```
SELECT type
FROM lieux;
```

```
SELECT DISTINCT type
FROM lieux;
```

```
SELECT DISTINCT type, localite FROM lieux;
```

- Supprime les lignes du résultat qui apparaissent plusieurs fois

Sélection ordonnée

```
SELECT id_observation, duree
FROM observations
ORDER BY duree ASC;
```

- Les lignes sont triées selon la valeur croissante de la colonne `duree`

```
SELECT id_observation, duree
FROM observations
ORDER BY duree DESC;
```

- Tri dans l'ordre décroissant

Sélection partielle

```
SELECT TOP 1 id_observation, duree
FROM observations
ORDER BY duree DESC;
```

- Retourne la première ligne seulement
- Ici, retourne l'identifiant `id_observation` et la durée `duree` de la plus longue observation
- `TOP` n'a de sens que lorsqu'il est utilisé avec la clause `ORDER BY`

Calcul sur les colonnes

- La clause `SELECT` peut retourner une colonne ou le résultat d'une fonction sur cette colonne

```
SELECT SUM(duree)/60 FROM observations;
SELECT MAX(quantite) FROM details;
```

Fonction	Calcule, pour une colonne
`AVG(col)`	Moyenne des valeurs
`SUM(col)`	Somme des valeurs
`COUNT(col)`	Nombre de valeurs
`FIRST(col)` `LAST(col)`	Première/dernière valeur (à toujours utiliser avec `ORDER BY`)
`MAX(col)` `MIN(col)`	Max ou Min des valeurs

Sélection groupée

- Une fonction donne une seule valeur (pour toute la colonne)

```
SELECT id_animal, MAX(quantite)
  FROM details;
```

➽ cette requête n'est pas correcte !
(et sera refusée par Access)

- Que faire si on souhaite le max de `quantite` mais pour chaque valeur de `id_animal` présent dans la table ?

- Utilisation d'une clause `GROUP BY`

Sélection groupée

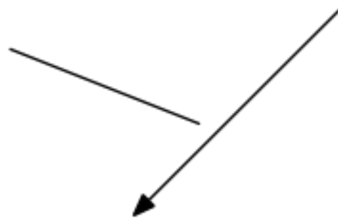
GROUP BY(joueur)

goals		
joueur	***match_id***	***buts***
Francois	23	2
Simon	9	1
Samantha	12	3
Francois	3	1
Samantha	4	2

goals GROUP BY joueur		
joueur	***match_id***	***buts***
Francois	23	2
	3	1
Samantha	12	3
	4	2
Simon	9	1

(cette table ne peut pas être résultat, elle est montrée pour illustrer le mécanisme)

SELECT joueur, SUM(buts)
FROM goals GROUP BY joueur;



resultat	
joueur	***expr0001***
Francois	3
Samantha	5
Simon	1

Sélection groupée

```
SELECT id_animal, MAX(quantite)
FROM details
GROUP BY id_animal;
```

- Groupe les lignes par valeur commune pour une colonne
  - Ici, les lignes avec la même valeur pour `id_animal` sont groupées ensemble
- La clause `GROUP BY` est toujours utilisée avec une fonction de calcul : à partir de plusieurs lignes pour un groupe, on doit obtenir une seule ligne résultat

- Access nomme `'expr001'` la colonne contenant `MAX(quantite)`
  - On peut lui donner un nom explicite avec "`AS`" :

```
SELECT id_animal, MAX(quantite) AS meilleure_obs
FROM details
GROUP BY id_animal;
```

Requêtes sur plusieurs tables

- Jusqu'ici, les requêtes SQL que nous avons vu s'appliquaient à *une seule table*
- L'intérêt d'une base de données est de faire des interrogations recoupant les données de *plusieurs tables* !
- Pour cela, deux mécanismes :
  - Jointure
  - Requête composée

Produit cartésien

- Exprimé par une virgule entre les noms des tables :

```
SELECT * FROM animaux, familles;
```

- Donne une table résultat du produit cartésien
  - Nombre de colonnes = somme du nombre de colonnes de toutes les tables utilisées dans le produit cartésien
  - Lignes = toutes les combinaisons de lignes possibles
    - Ici, tous les couples de lignes entre `animaux` et `familles`
  - Nombre de lignes = produit du nombre de lignes des tables utilisées dans le produit cartésien

Produit cartésien

empl		
nom	***prenom***	***classe***
Robert	Jean	2
Dubouc	Mireille	2
Cohen	Simon	1

paye	
classe	***salaire***
1	85.000 CHF
2	120.000 CHF

produit cartésien
entre tables *empl* et *paye*

SELECT * FROM empl, paye;				
empl.nom	***empl.prenom***	***empl.classe***	***paye.classe***	***paye.salaire***
Robert	Jean	2	1	85.000 CHF
Robert	Jean	2	2	120.000 CHF
Dubouc	Mireille	2	1	85.000 CHF
Dubouc	Mireille	2	2	120.000 CHF
Cohen	Simon	1	1	85.000 CHF
Cohen	Simon	1	2	120.000 CHF

Jointure

- Avec le produit cartésien, on obtient toutes les lignes possibles, mais seules certaines ont un sens par rapport à notre schéma et à nos données
- La jointure est un produit cartésien où on filtre les lignes qui n'ont pas une valeur en commun sur une colonne précise
  - Le plus souvent, la colonne est la clé primaire d'un côté et une clé étrangère de l'autre

```
SELECT * FROM animaux, familles
  WHERE
  animaux.id_famille = familles.id_famille;
```

Jointure

empl		
nom	***prenom***	***classe***
Robert	Jean	2
Dubouc	Mireille	2
Cohen	Simon	1

paye	
classe	***salaire***
1	85.000 CHF
2	120.000 CHF

produit cartésien
entre tables *empl* et *paye*

SELECT * FROM empl, paye;				
empl.nom	***empl.prenom***	***empl.classe***	***paye.classe***	***paye.salaire***
Robert	Jean	2	1	85.000 CHF
Robert	Jean	2	2	120.000 CHF
Dubouc	Mireille	2	1	85.000 CHF
Dubouc	Mireille	2	2	120.000 CHF
Cohen	Simon	1	1	85.000 CHF
Cohen	Simon	1	2	120.000 CHF

filtrage sur
empl.classe = paye.classe

SELECT * FROM empl, paye WHERE empl.classe=paye.classe;				
empl.nom	***empl.prenom***	***empl.classe***	***paye.classe***	***paye.salaire***
Robert	Jean	2	2	120.000 CHF
Dubouc	Mireille	2	2	120.000 CHF
Cohen	Simon	1	1	85.000 CHF

Jointure multiple

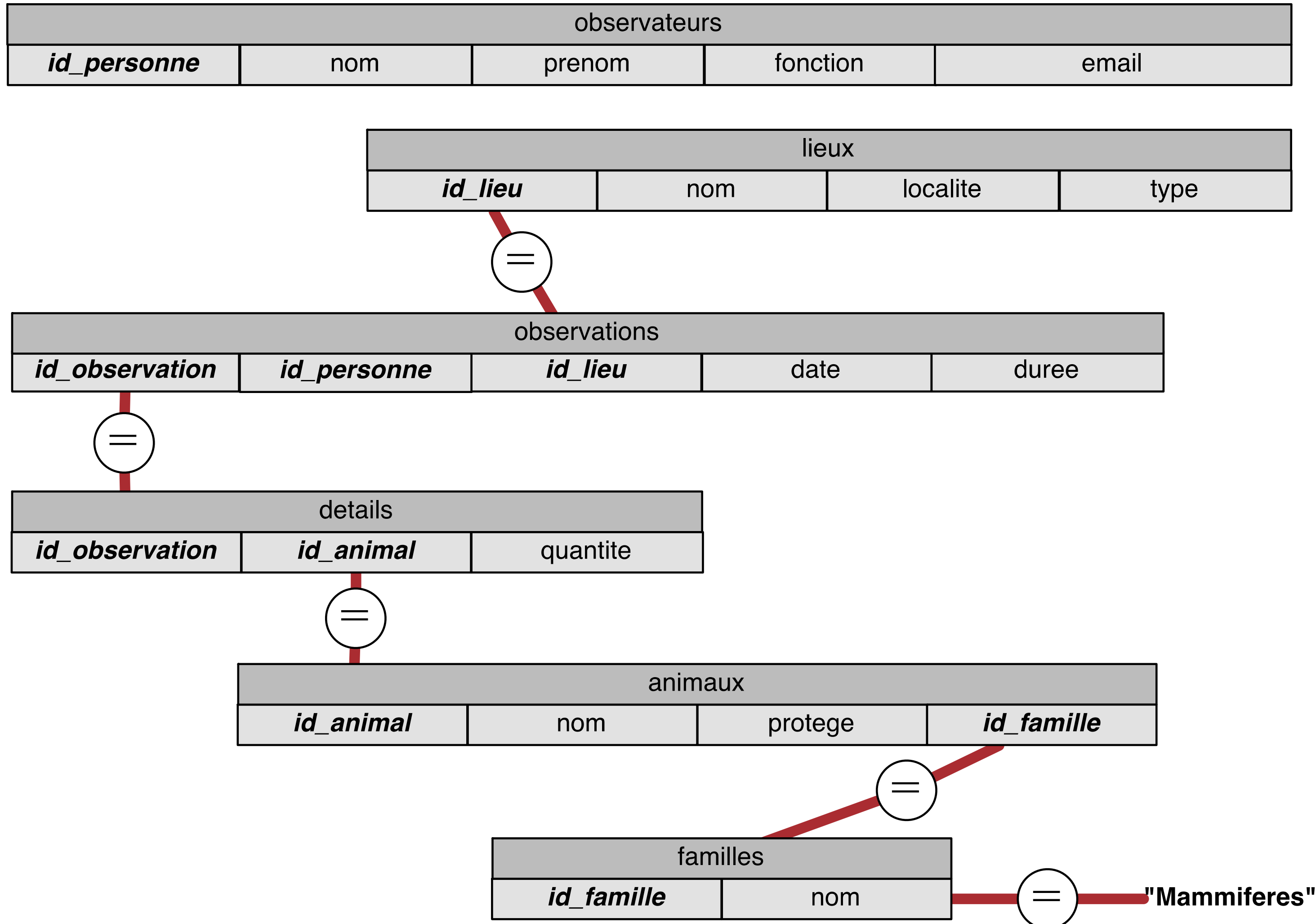
- On peut faire une jointure sur de multiples tables
  - Le nombre de critères pour la clause `WHERE` est (nombre de tables jointes - 1)

```
SELECT * FROM A, B, C, D
   WHERE A.id1=B.id1 AND B.id2=C.id2 AND C.id3=D.id3;
```

- Exemple : trouver les types de lieux où ont été observés des mammifères

```
SELECT DISTINCT lieux.type
   FROM observations, lieux, details, animaux, familles
WHERE observations.id_lieu = lieux.id_lieu
  AND observations.id_observation = details.id_observation
  AND details.id_animal = animaux.id_animal
  AND animaux.id_famille = familles.id_famille
  AND familles.nom = "Mammiferes";
```

Jointure multiple : illustration



Alias de tables

- On peut simplifier l'écriture de la requête précédente
  - Pas besoin d'écrire le nom de la table en entier à chaque fois
  - Alias = nom court spécifié dans la clause FROM

```
SELECT *
   FROM observations O, lieux L, details D,
        animaux A, familles F
WHERE O.id_lieu = L.id_lieu
   AND O.id_observation = D.id_observation
   AND D.id_animal = A.id_animal
   AND A.id_famille = F.id_famille
   AND F.nom = "Mammiferes";
```

Requêtes composées : inclusion

- Le mot clé `IN` permet d'évaluer si une valeur est présente dans une colonne d'une autre table, résultat d'une sous-requête

- Exemple :
  déterminer la liste des animaux observés au moins une fois

```
SELECT nom FROM animaux
WHERE id_animal IN
  (SELECT id_animal FROM details);
```

- La requête `(SELECT id_animal FROM details)`
  est une sous-requête : les parenthèses sont importantes

Requêtes composées

- Exemple : lister les animaux observés en forêt mais jamais en plaine
- On décompose la requête en **deux requêtes plus simples**
- Les animaux observés en plaine

```
SELECT D.id_animal
FROM observations O, lieux L, details D, animaux A
WHERE O.id_lieu = L.id_lieu
  AND O.id_observation = D.id_observation
  AND D.id_animal = A.id_animal
  AND L.type = "Plaine";
```

- Les animaux observés en forêt

```
SELECT D.id_animal
FROM observations O, lieux L, details D, animaux A
WHERE O.id_lieu = L.id_lieu
  AND O.id_observation = D.id_observation
  AND D.id_animal = A.id_animal
  AND L.type = "Foret";
```

- On enregistre ces deux requêtes comme "*aid_plaine*" et "*aid_foret*"

Requêtes composées

- On compose notre requête finale en appelant nos deux sous-requêtes

- Version 1, avec deux IN/NOT IN

```
SELECT nom FROM animaux
WHERE (id_animal IN (SELECT id_animal FROM aid_foret))
  AND (id_animal NOT IN (SELECT id_animal FROM aid_plaine));
```

  - Les parenthèses sont importantes !
  - On ne peut pas lire le résultat de la requête aid_foret directement depuis une évaluation de critère, il faut faire un SELECT FROM

- Version 2, avec une jointure

```
SELECT DISTINCT animaux.nom FROM aid_foret, animaux
WHERE aid_foret.id_animal=animaux.id_animal
  AND (id_animal NOT IN (SELECT id_animal FROM aid_plaine));
```

Exemples dirigés

- Lister les localités où l'on trouve au moins un site d'observation de type "foret"
- Lister les lieux où des animaux protégés ont été observés