420-C42

Langages d'exploitation des bases de données

Partie 5

DQL I

Introduction aux requêtes de consultation

DQL I

SELECT

- La clause SELECT permet de retrouver de l'information.
- Elle est la seule clause du DQL (souvent mise dans le groupe DML).
- Une requête SELECT retourne toujours l'un de ces résultats :
  - une erreur si la requête est erronée ou refusée
  - sinon, le résultat **toujours** sous la forme d'une **table**
    avec colonne(s) et ligne(s) (discussion sur les 5 formes de sortie)
- Sans la clause FROM :
  - nous verrons la clause FROM un peu plus loin
  - une requête SELECT retourne un seul tuple.
  - Plusieurs SGBD requièrent l'usage de la clause FROM avec elle et utilisent une table unitaire nommée DUAL lorsque requis (Oracle). Ce n'est pas le cas de PostgreSQL.

DQL I

SELECT

- Minimalement, la clause SELECT peut être utilisée avec une ou plusieurs expressions (séparées par des virgules) créant des colonnes pour chaque expression :
  - Littéraux (constantes) : numériques, textuelles, de date, …
  - Opérations : addition, concaténation, opération sur les dates, …
  - Fonctions : cosinus, mise en majuscule, manipulation sur les dates, …
  - Des expressions réalisant des amalgames des éléments précédents.

```
SELECT 0;                    SELECT 'Bonjour';              SELECT '2000-01-01';
SELECT 2 * 3, 3 * 2;         SELECT 'Hello' || ' world';    SELECT DATE('2000-01-01') + '1';
SELECT RANDOM();             SELECT UPPER('déjà');          SELECT NOW();
SELECT 100, 'Bonsoir', SQRT(81) + SIN(RANDOM()), EXTRACT(YEAR FROM NOW());
```

DQL I

FROM

- La clause FROM s'ajoute à la clause SELECT et permet la spécification de(s) table(s) où se trouvent les informations à manipuler.
- L'utilisation du SELECT avec la clause FROM utilise les données de la table spécifiée. Dès lors, n_t lignes sont retournées – une ligne de sortie par ligne de la table source.

n_t = le nombre de tuples présents dans la table

DQL I

FROM

- Dans ce cas, la clause SELECT permet l'identification des colonnes désirées.
- L'astérisque est utilisé pour spécifier toutes les colonnes de la table.
- Il est aussi possible de réaliser toutes les manipulations stipulées précédemment avec les données des tuples (opérateurs et fonctions).

```
SELECT * FROM employe;
SELECT nom, prenom, salaire FROM employe;
SELECT nom, NOW() - date_embauche FROM employe;
SELECT salaire, ROUND(salaire * 1.10, 2) FROM employe;
```

DQL I

alias

- Les titres des colonnes du résultat obtenu sont générés automatiquement.
- Il est possible de créer un alias pour chaque colonne de sortie avec le mot réservé AS ou directement avec un espace. L'usage du AS est recommandé.
- Cet alias devient le titre de la colonne de sortie. Un alias peut aussi servir de référent dans certaines parties du langage SQL (nous y reviendrons).
- L'alias peut s'écrire directement s'il n'y a pas d'espaces ou de caractères spéciaux, sinon les guillemets sont requis. Attention, les guillemets expriment une distinction nette avec les chaînes de caractères.

```
SELECT   salaire AS Salaire,
                ROUND(salaire * 1.10, 2) "Nouveau salaire"
       FROM employe;
```

DQL I

WHERE

- La clause WHERE s’ajoute aux clauses SELECT FROM et permet la spécification des lignes désirées.
- Elle utilise une expression logique (retournant une valeur booléenne) qui est calculée pour chaque ligne. La ligne est gardée si le résultat est vrai sinon la ligne est retirée du résultat de sortie.
- Il existe plusieurs façons de former cette expression :
  - Opérateurs de comparaison : =, <>, <, <=, >, >=
  - Opérateurs logiques : AND, OR, NOT
  - Recherche dans une liste : IN
  - Recherche dans un intervalle : BETWEEN
  - Recherche dans une chaîne de caractères : LIKE
  - Valeur nulle : IS NULL
  - ...

DQL I

WHERE

```
SELECT * FROM employe WHERE salaire >= 25;
SELECT * FROM employe WHERE nom = 'Dupuis';
SELECT * FROM employe WHERE nom <> 'Dupuis' AND salaire >= 20;
SELECT nom, prenom FROM employe WHERE date_embauche < '2002-01-01';
```

DQL I SELECT DISTINCT

- Il est fréquent que le résultat d'une requête retourne plusieurs lignes identiques. Par exemple :

SELECT departement FROM employe;

- Il est possible de retirer tous les doublons avec SELECT DISTINCT. Par exemple :

SELECT DISTINCT departement FROM employe;

- Cette clause est très utilisée dans plusieurs contextes. Par exemple, pour générer une liste d'éléments uniques.

DQL I opérateurs et fonctions

- Comme tous les langages, SQL possède plusieurs opérateurs et fonctions pour les différents types.
- On présente ici les plus communs mais il est essentiel de consulter la documentation pour :
  - connaître les outils existant avec le SGBD utilisé
  - les différences avec d'autres SGBD (elles peuvent être significatives)
  - les paramètres et leur usage
  - on présente ici PostgreSQL
- On parle des opérateurs et fonctions logiques, de comparaison, mathématiques, de gestion des chaînes de caractères et des dates, etc.

DQL I

opérateurs de comparaison et opérateurs logiques

- Les opérateurs de comparaisons sont :
  - = égalité
  - <> ou != différent
  - < plus petit
  - <= plus petit ou égal
  - > plus grand
  - >= plus grand ou égal
- Les opérateurs logiques sont :
  - AND et logique
  - OR ou logique
  - NOT négation logique

DQL I opérateur BETWEEN

- L'opérateur BETWEEN permet de simplifier une expression utilisant un intervalle.
- Les bornes sont inclusives et il est impossible de faire autrement.
- Par exemple, l'exemple suivant :

```
SELECT nom FROM employe WHERE salaire BETWEEN 10.00 AND 20.00;
```

- Correspond en tout point à :

```
SELECT nom FROM employe WHERE salaire >= 10.00 AND salaire <= 20.00;
```

DQL I opérateurs sur les listes

- Une liste est définie par une suite de valeurs du même type.
- Il en existe quelques types :
  - une liste explicite :
    (2, 7, 72)
    ('Montréal', 'Toronto', 'Vancouver')
  - une table de valeur en ligne
    - produites explicitement
      (VALUES (2), (7), (72))
      (VALUES ('Montréal'), ('Toronto'), ('Vancouver'))
    - par une requête retournant une seule colonne:
      SELECT DISTINCT ville FROM employe;
- Il existe plusieurs opérateurs facilitant l'analyse de liste : ANY, ALL, IN

DQL I opérateurs sur les listes – ANY, SOME

- L'opérateur ANY simplifie l'analyse d'une condition logique sur un ensemble de valeurs.
- Retourne vrai si **au moins une valeur** respecte la condition logique.
- La syntaxe est la suivante :
  ... WHERE xyz >= ANY(VALUES (12), (5), (74));
- La requête précédente est équivalente à :
  ... WHERE xyz >= 12 OR xyz >= 5 OR xyz >= 74;
- Cet opérateur requiert :
  - un opérateur conditionnel à gauche : <, <=, >, >=, <>, =
  - une liste à droite (une liste explicite n'est pas possible ici)
- Il est possible d'utiliser SOME plutôt que ANY sans distinction.

DQL I opérateurs sur les listes - ALL

- L'opérateur ALL simplifie l'analyse d'une condition logique sur un ensemble de valeurs.
- Retourne vrai si **toutes les valeurs** respectent la condition logique.
- La syntaxe est la suivante :
  ... WHERE xyz >= ALL(VALUES (12), (5), (74));
- La requête précédente est équivalente à :
  ... WHERE xyz >= 12 AND xyz >= 5 AND xyz >= 74;
- Cet opérateur requiert :
  - un opérateur conditionnel à gauche : <, <=, >, >=, <>, =
  - une liste à droite (une liste explicite n'est pas possible ici)

DQL I opérateurs sur les listes - IN

- L'opérateur IN correspond à l'opérateur = ANY
  - permet de valider qu'une valeur existe dans la liste
- L'opérateur NOT IN correspond à l'opérateur <> ALL
  - permet de valider qu'une valeur n'existe pas dans la liste
- L'opérateur IN est de loin le plus utilisé.
- Il s'utilise avec les listes explicites.

```
SELECT nom
   FROM employe
   WHERE departement IN ('ventes', 'r&d');
```

Exercices 1 & 2

DQL I opérateurs mathématiques

- Les opérateurs mathématiques principaux sont :
  - - négation (-5)
  - +, -, *, / addition, soustraction, multiplication, division
  - % modulo
  - ^ exponentiel (5^2 => 25)
  - |/, ||/ racine carrée et cubique (|/25 => 5)
  - @ valeur absolue (@-5 => 5)
- Les opérateurs bit à bit (« *bitwise* »)
  - &, |, #, ~ et, ou, ou exclusif, négation
  - <<, >> décalage à gauche et à droite

DQL I fonctions mathématiques

- Quelques fonctions mathématiques :
  - abs, sign — valeur absolue et valeur signe
  - floor, ceil, round, trunc — fonctions d'arrondissement
  - gcd, lcm — plus grand diviseur, plus petit multiple
  - power, exp — puissance et e^x
  - log, log10, ln — logarithme (avec base 10 et *e*)
  - sqrt, cbrt — racine carrée et cubique
  - degrees, radians — conversion degrés et radians
  - random, setseed — génération de nombre pseudo aléatoire
  - pi, sin, cos, tan, atan2, … — fonctions trigonométriques

Exercices 3

DQL I

opérateur et fonctions liés aux chaînes de caractères lien

- Opérateurs pour les chaînes de caractères :
  - || concaténation de 2 chaînes de caractères
- Fonctions pour les chaînes de caractères :
  - ascii, chr retourne le code ascii d'un caractères et inversement
  - lower, upper, initcap changement de casse (minuscule et majuscule)
  - concat concaténation de plusieurs chaînes
  - char, char_length, length retourne le nombre de caractères
  - overlay, translate, replace substitution de texte dans une chaîne existante
  - position, strpos recherche d'une sous chaîne
  - substr, left, right extraction d'une sous chaîne
  - trim, ltrim, rtrim efface des caractères (espace et autres)
  - lpad, rpad ajoute des caractères aux extrémités
  - substring, regexp_... utilisation d'expressions régulières

DQL I

LIKE

- L'opérateur LIKE (ou ~~) retourne vrai si la chaîne de caractères correspond au patron donné.
- L'opérateur ILIKE (ou ~~*) est le même mais insensible à la casse
- Le patron est une chaîne de caractères ayant ces caractéristiques :
  - si la chaîne ne contient aucun caractère de soulignement ou de pourcentage, alors la chaîne est prise tel quel et correspond à un =
  - le caractère de soulignement correspond à la substitution d'un caractère quelconque
  - le caractère de pourcentage correspond à la substitution de 0 à *n* caractères quelconques.
- L'usage de la barre oblique inverse permet d'utiliser un soulignement ou un pourcentage

```
SELECT 'Gaston' LIKE 'Gaston';          vrai
SELECT 'Gaston' LIKE 'G_ston';          vrai
SELECT 'Gaston' LIKE 'ga%on';           faux
SELECT 'Gaston' ~~* 'g%';               vrai
```

DQL I

expressions régulières

- Les expressions régulières ne font pas parti de la norme ANSI/ISO SQL. Cependant, la plupart des SGBD supporte les expressions régulières.
- PostgreSQL offre un excellent support pour la version POSIX.
- Les opérateurs sont :
  - ~, !~ correspondance et non correspondance
  - ~*, !~* correspondance et non corr. insensible à la casse
- Les fonctions sont :
  - regexp_replace substitution de patron
  - regexp_match(es) extraction de patron
- Pour faciliter la construction et la compréhension des expressions régulières : regex101

Exercices 4

DQL I

opérateurs et fonctions liés aux dates et aux heures lien

- Opérateurs pour les dates et heures :
  - +, - addition et soustraction entre dates et heures
  - *, / multiplication et division de dates et heures avec un nombre ou un intervalle
- Fonctions pour les dates et heures :
  - current_date — date courante
    current_time — heure courante (avec *tz*)
    current_timestamp — heure et date courante (avec *tz*)
    local_time — heure courante (sans *tz*)
    local_timestamp — heure et date courante (sans *tz*)
    transaction_timestamp — heure et date de la transaction courante = current_timestamp
    statement_timestamp — heure et date de la réception de l'expression courante
    clock_timestamp — heure et date courante (change pendant transaction)
    now — = transaction_timestamp
  - date_part, extract — extraction d'une partie de la date et de l'heure
  - make_date — construit une date avec des nombres
  - age — retourne l'intervalle entre deux dates

DQL I formatage (entrée et sortie) lien

- Les fonctions suivantes permettent la manipulation d'un type de données vers une chaîne de caractères et vice versa.
- Il est important de se référer à la documentation car les variantes syntaxiques liées au formatage sont nombreuses.
- Conversion d'un type spécifique vers une chaîne de caractères :
  - to_char un nombre, une date, une heure
un horodatage, un intervalle, …
- Conversion d'une chaîne de caractères vers le type approprié :
  - to_number vers un nombre
  - to_date vers une date
  - to_timestamp vers un horodatage

DQL I formatage (entrée et sortie) lien

```
SELECT to_char(1234567.89, '9,999,999.99');                    -- Résultat : 1,234,567.89
SELECT to_char(CURRENT_DATE, 'DD Mon YYYY');                   -- Exemple : 28 Oct 2025
SELECT to_char(CURRENT_TIMESTAMP, 'HH24:MI:SS');               -- Exemple : 10:56:27

SELECT to_number(' $1,234.56', 'L9,999.99');                   -- Résultat : 1234.56
SELECT to_date('2023-10-27', 'YYYY-MM-DD');                    -- Résultat : 2023-10-27
SELECT to_date('October 27, 2023', 'Month DD, YYYY');          -- Résultat : 2023-10-27
SELECT to_timestamp('2023-10-27 14:35:22', 'YYYY-MM-DD HH24:MI:SS’);

SELECT to_char(to_date('2023-10-27', 'YYYY-MM-DD'), 'Month');
                                                               -- Résultat : October
```

DQL I

conversion de types lien

- SQL est un langage typé fort (comme les langages C++ et Java).
- Ainsi, la conversion de type est un sujet important et parfois essentiel à l'écriture de certaines requêtes.
- La conversion de type peut être réalisée ainsi :
  - conversion implicite
  - conversion explicite par :
    - l'opérateur CAST (standard SQL)
    - l'opérateur :: (style PostgreSQL)
    - une fonction
- Il est important de savoir que PostgreSQL supporte la notion de surcharge de fonctions. Par exemple, il y a 3 fonctions ROUND.

DQL I conversion de types

```
-- Appel de la fonction appropriée (surcharge vers le dbl. précision)
SELECT ROUND(0.75);

-- Conversion implicite de la chaîne de car. vers un double précision
SELECT ROUND('0.75');

-- Conversion explicite :
SELECT ROUND(CAST('0.75' AS NUMERIC))    -- syntaxe SQL SELECT
ROUND('0.75'::NUMERIC);                  -- syntaxe PostgreSQL
SELECT ROUND(NUMERIC '0.75');            -- autre syntaxe possible
```

DQL I

valeurs nulles

- La manipulation de la valeur nulle est délicate en SQL car elle peut avoir plusieurs sens :
  - une valeur inconnue (date de naissance);
  - un sens spécifique au contexte (date de mariage)
- De plus, les opérateurs et fonctions peuvent donner des résultats inattendus. Par exemple :
  SELECT 5 + NULL; => NULL
  SELECT TRUE AND NULL; => NULL
  SELECT TRUE OR NULL; => TRUE
- Il est donc essentiel de considérer les valeurs nulles partout où elles s'appliquent.

DQL I

valeurs nulles

- Plusieurs opérateurs et fonctions retournent systématiquement NULL si l'une des opérandes ou l'un des paramètres est NULL.
- Ce n'est pas toujours le cas. C'est pourquoi il est important de lire la documentation.
- Par exemples :
  - SELECT 'bob' = NULL; -- retourne NULL
  - SELECT NULL = NULL; -- retourne NULL
  - SELECT 123 <> NULL; -- retourne NULL
  - SELECT 'Allo' || NULL; -- retourne NULL

DQL I

valeurs nulles

- Une valeur nulle n’est ni vraie ni fausse.
- Tables de vérité :

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	NULL

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

DQL I

valeurs nulles

- L'opérateurs IS permet, entre autres, la gestion de la valeur nulle :
  - IS : questionne si une valeur est nulle :
    - exp IS NULL
    - exp IS NOT NULL
  - IS DISTINCT FROM : compare deux valeurs et considère les valeurs nulles :
    - exp1 IS DISTINCT FROM exp2 -- similaire à exp1 <> exp2 tout en considérant la valeur nulle
    - exp1 IS NOT DISTINCT FROM exp2 -- similaire à exp1 = exp2 tout en considérant la valeur nulle

a	b	a = b	a <> b	a IS b	a IS NOT b	a IS NOT DISTINCT FROM b	a IS DISTINCT FROM b
123	123	TRUE	FALSE	X	X	TRUE	FALSE
123	NULL	NULL	NULL	FALSE	TRUE	FALSE	TRUE
NULL	NULL	NULL	NULL	TRUE	FALSE	TRUE	FALSE

DQL I

valeurs nulles

- Fonction conditionnelle retournant NULL :
  - NULLIF(arg1, arg2) -- retourne NULL si arg1 = arg2, sinon arg1
- Fonction réalisant la substitution d'une éventuelle valeur nulle :
  - COALESCE(arg1, arg2, …) -- retourne la première valeur non nulle
-- sinon retourne NULL
-- les arguments doivent être du même type
  - Dans certain SGBD, la fonction NVL est plutôt utilisée (Oracle)
- Exemples

```
SELECT 1.0 / NULLIF(value, 0.0) AS Inverse FROM data;

SELECT prenom || ' ' || nom FROM employe WHERE departement IS NULL;

SELECT prenom || ' ' || nom, COALESCE(departement, '-') FROM employe;
```

Exercices 6

DQL I

ORDER BY

- Il est important de savoir que l'ordre de sortie d'une requête SELECT est arbitraire et aucun ordre spécifique ne peut être garantie
- L'ordre dans lequel les lignes sont retournées peut dépendre de nombreux facteurs, tels que :
  - la manière dont les données sont stockées physiquement
  - l'indexation si elle existe
  - le plan d'exécution choisi par l'optimiseur de requête
  - la version du système de gestion de base de données
- L'ordre peut changer entre différentes exécutions de la même requête!
- La clause ORDER BY s'ajoute aux clauses SELECT FROM (WHERE étant optionnel) et permet de trier les lignes selon le(s) critère(s) désiré(s).

DQL I

ORDER BY

- On détermine l'ordre de tri par l'identification des colonnes (séparées par des virgules en ordre de priorité). Ainsi, lorsque deux valeurs sont identiques pour la première colonne, le tri est précisé par la 2^{e} colonne et ainsi de suite.
- Les mots réservés ASC et DESC permettent de spécifier un tri croissant ou décroissant. ASC est l'ordre de tri par défaut.
- Les mots réservés NULLS FIRST et NULLS LAST permettent de préciser la position des valeurs nulles. Par défaut, suit l'ordre ASC (LAST) et DESC (FIRST).
- Il est possible d'utiliser l'alias d'une colonne dans la clause ORDER BY.

DQL I

ORDER BY

```
SELECT nom, prenom, departement
        FROM employe
        ORDER BY nom;


SELECT nom, prenom, departement AS dep
        FROM employe
        ORDER BY dep ASC NULLS LAST,
                        nom DESC NULLS LAST,
                        prenom;                         -- par défaut ASC NULLS FIRST
```

DQL I

LIMIT & OFFSET

- Les clause LIMIT et OFFSET vous permettent de produire un sous-ensemble du résultat de la requête.
- Ces clauses s'ajoutent aux clause SELECT FROM. Par défaut, elles valent 0.
- Ainsi, il est possible de produire une requête pour laquelle on ne désire que les 3 valeurs débutant à la 2^{e} position.
- Attention, certains SGBD (Oracle) ne supportent pas spécifiquement ces clauses et ils utilisent d'autres stratégies.

```
SELECT nom, prenom, departement
        FROM employe
        ORDER BY nom
        LIMIT 3
        OFFSET 2;
```

DQL I

CASE

- L’expression CASE permet d’obtenir un résultat différent selon plusieurs conditions.
- Elle est considérée comme une expression conditionnelle générique.
- Elle est similaire, mais différente, à une condition switch-case des langages C, C++, C# et Java que vous connaissez bien.
- Le synopsis est :

  CASE WHEN *condition* THEN *résultat pour cette condition*
  [WHEN *condition* THEN *résultat pour cette condition*]*
  [ELSE *résultat si toutes les autres conditions ne sont pas remplies*]
  END

- Puisqu’il n’existe pas de IF, on utilise CASE WHEN … THEN … ELSE … END

DQL I

CASE

- Cet exemple permet de :
  - conjuguer un mot descriptif associé au genre
  - identifier les citadins et les banlieusards (équivalent d'un IF)

```
SELECT
    nom,
    CASE WHEN genre = 'f' THEN 'Femme'
         WHEN genre = 'h' THEN 'Homme'
         WHEN genre = 'x' THEN 'Non binaire'
         ELSE 'Genre inconnu'
    END   AS Genre,
    CASE WHEN ville = 'Montréal' THEN 'Citadin'
         ELSE 'Banlieusard'
    END   AS Habite
FROM employe;
```

DQL I requêtes imbriquées

- Il est possible d'imbriquer deux requêtes afin que la requête externe utilise le résultat de la requête interne.
- Il n'y a pas de limite logique au nombre d'imbrication.
- Une requête interne **doit** se retrouver entre parenthèses.
- Par exemple, on voudra trouver de l'information spécifique dans une table pour alimenter les critères de recherche d'une autre requête. L'exemple suivant permet de retrouver le nom de tous les employés du même département que l'employé Dupuis.

```
SELECT nom, prenom
        FROM employe
        WHERE departement = (SELECT departement
                                        FROM employe
                                        WHERE nom = 'Dupuis');
```

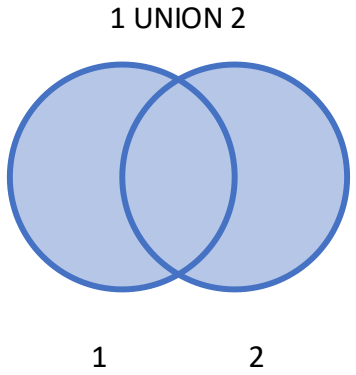
DQL I requêtes imbriquées

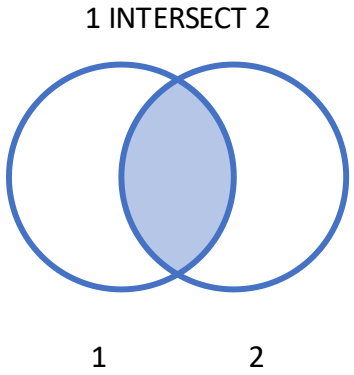
- Il est important de se rappeler les 5 types de résultats d'une requête.
- Ainsi, une requête imbriquée peut servir dans différents contextes :
  - scalaire — opération de comparaison, logique et mathématique
  - Liste issue d'une colonne — IN, ANY, ALL, ...
- Cet exemple donne le nom de tous les employés qui habitent dans les mêmes villes que ceux qui travaillent dans le département de r&d.

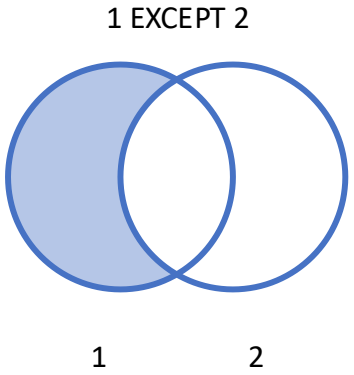
```
SELECT nom, prenom
        FROM employe
        WHERE ville IN (SELECT ville
                                FROM employe
                                WHERE departement = 'r&d');
```

DQL I combinaisons de requêtes

- Il est possible de combiner les résultats de deux requêtes en un seul résultat.
- Les opérateurs sont :
  - UNION : ajoute à la 1ère requêtes les résultats de la 2e
  - INTERSECT : retourne ce qui est commun aux deux requêtes
  - EXCEPT : retourne ce qui se trouve dans la 1ère requête et qui n'est pas dans la 2e requête (MINUS)







DQL I combinaisons de requêtes

- Pour que la combinaison soit possible, il faut que les deux requêtes possèdent le même schéma :
  - le même nombre de colonne
  - les mêmes types dans le même ordre.
- Il n’est pas nécessaire que les données aient un sens en autant que les critères précédents soient respectés. Le sens est la responsabilité du programmeur.

```
SELECT nom FROM employe
UNION
SELECT prenom FROM employe;
```

Exercices 7