

Bases de données avancées

Master 1 Informatique

2025-2026

Jérôme Darmont

<https://eric.univ-lyon2.fr/jdarmont/>



Actualité du cours



https://eric.univ-lyon2.fr/jdarmont/?page_id=3142



<https://eric.univ-lyon2.fr/jdarmont/?feed=rss2>



<https://social.sciences.re/@darmont>

#m1bda

Objectifs du cours

▶ SQL : langage de requêtes (bases de données relationnelles)

- Standard
- Optimiseurs de requêtes
- Non procédural
- Données structurées

▶ Programmation nécessaire pour :

- Tâches complexes
- Interfaces utilisateurs

Langage
PL/pgSQL

▶ 80 % des données sont peu ou pas structurées

- Description via le langage XML

requêtes

Langage
XQuery

Partie 1

PL/pgSQL

Procedural Language/PostgreSQL Structured Query Language





- ▶ Introduction
- ▶ Bases du langage
- ▶ Curseurs
- ▶ Gestion des erreurs
- ▶ Déclencheurs
- ▶ SQL dynamique

Requêtes SQL dans un programme



- ▶ **SQL encapsulé** : Requêtes SQL incorporées dans le code source (PL/SQL, T-SQL, PL/pgSQL, Pro*C...)
- ▶ **API** : Requêtes SQL via des fonctions du langage (Java Persistence API, PHP Data Objects...)
- ▶ **Interfaces de niveau appel** : intergiciel entre le langage et le SGBD (ODBC, JDBC, ADO...)
- ▶ **Procédures stockées** : Fonctions SQL stockées dans la base de données et exécutées par le SGBD (écrites en PL/SQL, T-SQL, PL/pgSQL)

C
U
R
S
E
U
R
S


Choix de PostgreSQL



- ▶ SGBD relationnel-objet
- ▶ Licence libre (BSD)
- ▶ Le plus conforme au standard SQL
- ▶ Disponible sur de nombreuses plateformes
 - Divers UNIX, dont Linux et Mac OS
 - Windows

Historique de PostgreSQL



- ▶ 1974 : Ingres
(INteractive Graphics REtrieval System)
- ▶ 1985 : Postgres (post-Ingres)
- ▶ 1995 : Postgres95 (fonctionnalités  SQL)
- ▶ 1996 : PostgreSQL (v6)
- ▶ 2025 : v17.5

1974



Michael Stonebraker



fr.wikipedia.org

Caractéristiques de PostgreSQL



- ▶ Types structurés (enregistrements, tableaux) possibles dans les tables (relationnel étendu)
- ▶ Comportement stable
- ▶ Langage procédural PL/pgSQL proche du PL/SQL d'Oracle (permet le SQL dynamique)
- ▶ Interfaçage possible avec des modules externes d'autres langages (PERL, Python...)



- ▶ Utilisation des types de données, opérateurs et fonctions de SQL
- ▶ Stockage du code dans la base de données
 - Sécurité liée à celle du SGBD et de ses droits d'accès
- ▶ Exécution au sein du serveur de BD
 - Pas d'allers-retours entre client et serveur
⇒ Performance

Sondage express

- A) Jusqu'ici, tout va bien.
- B) Je suis déjà perdu·e.



N° de la question : 8870

Répondre sur <https://toreply.univ-lille.fr>

Question n° 8870



- ✓ Introduction
- ▶ Bases du langage
- ▶ Curseurs
- ▶ Gestion des erreurs
- ▶ Déclencheurs
- ▶ SQL dynamique



```
[DECLARE  
  -- Déclarations]
```

```
BEGIN  
  -- Instructions PL/pgSQL
```

```
[EXCEPTION  
  -- Gestion des erreurs]
```

```
END
```

[] : clause optionnelle
(idem dans tout le document)



- ▶ Déclaration dans la section **DECLARE** d'un bloc PL/pgSQL

- ▶ **Variables**

ex. dateNaissance DATE;
 compteur INTEGER := 0; -- Initialisation
 compteur2 INTEGER DEFAULT 0; -- Valeur par défaut
 id CHAR(5) NOT NULL := 'AP001';

- ▶ **Constantes**

ex. tauxTVA CONSTANT REAL := 0.2;

Principaux types de données



Type	Description
boolean, bool	Booléen (vrai/faux)
smallint, int2	Entier de -32768 à +32767
integer, int, int4	Entier de -2147483648 à +2147483647
bigint, int8	Entier de -9223372036854775808 à +9223372036854775807
real, float4	Réel simple précision (6 décimales)
double precision, float8	Réel double précision (15 décimales)
numeric, numeric(P, S), decimal, decimal (P, S)	Nombre jusqu'à 131072 chiffres avant la virgule (P), 16383 après (S)
character[N], char[N]	Chaîne de caractères de longueur N fixe
varchar, varchar[N]	Chaîne de caractères de longueur variable (maximum N si précisé)
text	Texte long de longueur variable (nécessite des opérateurs spécifiques)
date	Date (ex. 'jj/mm/aaaa')
time	Heure (ex. 'hh:mm:ss')
timestamp	Date et heure (ex. 'aaaa-mm-jj hh:mm:ss')

Référencer un type existant



- ▶ Type d'une autre variable

ex. credit REAL;
 debit credit%TYPE;

- ▶ Type de l'attribut d'une table

ex. numEmp EMP.EMPNO%TYPE;

- ▶ Type des n-uplets d'une table

ex. unEtudiant STUDENT%ROWTYPE;

- ▶ N-uplet indéfini (enregistrement)

ex. resultat RECORD;

À utiliser
au maximum !



► Tableaux

ex. notes NUMERIC(2, 2)[];
 matrice INTEGER[][];

► Types composites (enregistrements)

ex. CREATE TYPE tEmploye AS (-- Définition
 num NUMERIC,
 nom VARCHAR
); -- à définir hors de PL/pgSQL (commande SQL)

unEmploye tEmploye; -- Déclaration



► Variables

ex. $n := 0;$
 $n := n + 1;$

► Tableaux

ex. $\text{notes} := \text{ARRAY}[10.2, 13.3, 15.5, 9.8];$
 $\text{matrice} := \text{ARRAY}[\text{ARRAY}[4, 2],$
 $\text{ARRAY}[1, 9]];$

► Enregistrements

ex. $\text{unEmploye} := (1501, \text{'DARMONT'});$



► Variables

ex.

```
SELECT custname INTO nomClient  
FROM customer WHERE custnum = 10;
```

```
SELECT ename, sal INTO nom, salaire  
FROM emp WHERE empno = 5000;
```

► Enregistrements

ex.

```
SELECT empno, ename INTO unEmploye  
FROM emp WHERE empno = 1501;
```

```
SELECT * INTO resultat  
FROM customer WHERE custnum = 20;
```

Opérateurs arithmétiques et logiques



▶ Opérateurs arithmétiques

+ - / * **

▶ Opérateur de concaténation

||

▶ Opérateurs de comparaison

= < > <= >= <>
IS NULL LIKE BETWEEN IN

▶ Opérateurs logiques

AND OR NOT



IF-THEN, IF-THEN-ELSE ou IF-THEN-ELSIF

```
IF condition1 THEN
    -- Instructions PL/pgSQL
[ELSIF condition2 THEN
    -- Instructions PL/pgSQL]
[ELSE
    -- Instructions PL/pgSQL]
END IF;
```



CASE simple

CASE variable

WHEN val1 THEN

WHEN val2, val3 THEN

WHEN val4 THEN

[ELSE

END CASE;

-- Instructions PL/pgSQL

-- Instructions PL/pgSQL

-- Instructions PL/pgSQL

-- Instructions par défaut]



CASE par intervalles

CASE

WHEN var BETWEEN val1 AND val2 THEN

-- Instructions PL/pgSQL

WHEN var BETWEEN val2 + 1 AND val3 THEN

-- Instructions PL/pgSQL

END CASE;



► Pour

```
FOR iterateur IN [REVERSE] min..max [BY pas] LOOP  
    -- Instructions PL/pgSQL  
END LOOP;
```

► Tant que

```
WHILE condition LOOP  
    -- Instructions PL/pgSQL  
END LOOP;
```

► Répéter jusqu'à

```
LOOP  
    -- Instructions PL/pgSQL  
    EXIT WHEN condition; -- C'est moche, mais  
END LOOP;                -- pas le choix...
```




► Tableau

```
FOREACH note IN ARRAY notes LOOP -- note est un NUMERIC(2, 2)
    -- Instructions PL/pgSQL
END LOOP;
```

► Matrice

```
FOREACH n IN ARRAY matrice LOOP -- n est un INTEGER
    -- Instructions PL/pgSQL
END LOOP;
-- Hé oui, pas besoin de boucles imbriquées !
```

Quizz

Combien y a-t-il de sections dans un bloc PL/pgSQL ?

- A) 1
- B) 2
- C) 3
- D) 4
- E) 5



N° de la question : 8008

Répondre sur <https://toreply.univ-lille.fr>

Question n° 8008



► Dans une fonction

ex. CREATE [OR REPLACE] FUNCTION test() RETURNS VOID AS \$\$
 -- bloc PL/pgSQL
 \$\$ LANGUAGE plpgsql;

► Exécution de la fonction

ex. SELECT test(); -- La forme du résultat peut différer
 SELECT * FROM test(); -- en fonction des clients PostgreSQL.

Exemple de fonction



-- Calcul de prix TTC

```
CREATE OR REPLACE FUNCTION calculPrixTTC(idProduct NUMERIC) RETURNS REAL AS $$  
  DECLARE  
    tauxTVA CONSTANT REAL := 0.2;  
    prixHT demo_product_info.list_price%TYPE;  
  BEGIN  
    -- Lire le prix HT  
    SELECT list_price INTO prixHT FROM demo_product_info  
    WHERE product_id = idProduct;  
    -- Retourner le prix TTC  
    RETURN prixHT * (1 + tauxTVA) ::REAL;  
  END  
$$ LANGUAGE plpgsql;
```

-- Exécution

```
SELECT calculPrixTTC(10);
```

Fonction retournant plusieurs valeurs



Paramètres de sortie

```
CREATE OR REPLACE FUNCTION calculs( n1 INT, n2 INT,  
                                     OUT somme INT, OUT produit INT) AS $$ -- Pas de RETURN  
    BEGIN  
        somme := n1 + n2;  
        produit := n1 * n2;  
    END  
$$ LANGUAGE plpgsql;
```



```
SELECT calculs(4, 5);  
-- OU  
SELECT * FROM calculs(4, 5);
```

Fonction retournant plusieurs enregistrements



```
CREATE OR REPLACE FUNCTION serie(taille INT, pas INT) RETURNS SETOF INT AS $$  
  DECLARE  
    i INT;  
  BEGIN  
    FOR i IN 1..taille BY pas LOOP  
      RETURN NEXT i;  
    END LOOP;  
    RETURN;  
  END  
$$ LANGUAGE plpgsql;
```

```
SELECT serie(10, 2);
```

Appel de fonction dans une fonction



```
CREATE OR REPLACE FUNCTION droleDeDiv(n1 INT, n2 INT) RETURNS REAL AS $$  
    DECLARE  
        s INT;  
        p INT;  
    BEGIN  
        SELECT * INTO s, p FROM calculs(n1, n2);  
        RETURN p / s ::REAL;  
    END  
$$ LANGUAGE plpgsql;  
  
SELECT droleDeDiv(5, 6);
```



```
CREATE OR REPLACE FUNCTION factorielle(n INTEGER) RETURNS INTEGER AS $$  
  DECLARE  
    f INTEGER;  
  BEGIN  
    IF n = 1 THEN -- Condition d'arrêt  
      RETURN 1;  
    ELSE  
      SELECT * INTO f FROM factorielle(n - 1); -- Appel récursif  
      RETURN n * f;  
    END IF;  
  END  
$$ LANGUAGE plpgsql;  
  
SELECT factorielle(10);
```


Sondage express

Peut-on écrire un bloc PL/pgSQL
en-dehors d'une fonction?



N° de la question : 8066

Répondre sur <https://toreply.univ-lille.fr>

Question n° 8066



- ✓ Introduction
- ✓ Bases du langage
- ▶ Curseurs
- ▶ Gestion des erreurs
- ▶ Déclencheurs
- ▶ SQL dynamique



- ▶ Requête qui retourne **un seul n-uplet**
 - SELECT INTO
 - Stockage du résultat dans une ou plusieurs variables ou un enregistrement
- ▶ Requête qui retourne **plusieurs n-uplets**
 - Nécessité d'un **curseur**
 - Structure de données en mémoire qui stocke le résultat de la requête (\Leftrightarrow tableau d'enregistrements)

Curseur implicite (non lié)



-- Parcours complet du curseur

```
CREATE OR REPLACE FUNCTION listeEmp( ) RETURNS SETOF tEmploye AS $$  
  DECLARE  
    e tEmploye;  
  BEGIN  
    FOR e IN SELECT * FROM emp LOOP  
      e.nom := LOWER(e.nom );  
      RETURN NEXT e;  
    END LOOP;  
    RETURN;  
  END  
$$ LANGUAGE plpgsql;  
  
SELECT listeEmp();
```

De très loin
le plus courant !

Curseur explicite (lié)



-- Parcours ad-hoc du curseur (même résultat que listeEmp → utiliser listeEmp !!)

```
CREATE OR REPLACE FUNCTION listeEmp2( ) RETURNS SETOF tEmploye AS $$
```

```
  DECLARE
```

```
    cursEmp CURSOR FOR SELECT * FROM emp;
```

```
    e tEmploye;
```

```
  BEGIN
```

```
    OPEN cursEmp;
```

```
    FETCH cursEmp INTO e;           -- Lecture du 1er n-uplet
```

```
    WHILE FOUND LOOP
```

```
      e.nom := LOWER(e.nom);
```

```
      RETURN NEXT e;
```

```
      FETCH cursEmp INTO e;        -- Lecture du n-uplet suivant
```

```
    END LOOP;
```

```
    CLOSE cursEmp;
```

```
    RETURN;
```

```
  END
```

```
$$ LANGUAGE plpgsql;
```

Curseur explicite (lié)



-- Parcours vraiment ad-hoc du curseur (renvoie 1 résultat sur 3)

```
CREATE OR REPLACE FUNCTION listeEmp3(pas INT) RETURNS SETOF tEmploye AS $$
```

```
  DECLARE -- Pas de changement
```

```
  BEGIN
```

```
    OPEN cursEmp;
```

```
    FETCH cursEmp INTO e;
```

```
    WHILE FOUND LOOP
```

```
      e.nom := LOWER(e.nom);
```

```
      RETURN NEXT e;
```

```
      MOVE FORWARD pas FROM cursEmp; -- MOVE cursEmp; pour un seul décalage
```

```
      FETCH cursEmp INTO e;
```

```
    END LOOP;
```

```
    CLOSE cursEmp;
```

```
    RETURN;
```

```
  END
```

```
$$ LANGUAGE plpgsql;
```

```
SELECT * FROM listeEmp3(2);
```

Curseur paramétré (implicite)



```
CREATE OR REPLACE FUNCTION listeEmp5(salPlancher DECIMAL) RETURNS SETOF tEmploye AS $$  
  DECLARE  
    e tEmploye;  
  BEGIN  
    FOR e IN SELECT * FROM emp WHERE sal >= salPlancher LOOP  
      e.nom := LOWER(e.nom);  
      RETURN NEXT e;  
    END LOOP;  
    RETURN;  
  END  
$$ LANGUAGE plpgsql;  
  
SELECT listeEmp5(2000);
```

Quizz

On veut appliquer un échantillonnage sur les n-uplets d'une table. Quel type de curseur doit-on utiliser pour les sélectionner ?

- A) Curseur implicite
- B) Curseur explicite
- C) Curseur paramétré



N° de la question : 5305

Répondre sur <https://toreply.univ-lille.fr>
Question n° 5305



- ✓ Introduction
- ✓ Bases du langage
- ✓ Curseurs
- ▶ Gestion des erreurs
- ▶ Déclencheurs
- ▶ SQL dynamique



RAISE niveauErreur '**message**';

Niveau d'erreur	Priorité	Description
DEBUG	1	Écrit le message dans le log
LOG	2	Écrit le message dans le log
INFO	3	Écrit le message dans le log et l'envoie au client
NOTICE	4	Écrit le message dans le log et l'envoie au client
WARNING	5	Écrit le message dans le log et l'envoie au client
EXCEPTION	6	Interrompt la transaction courante

Exceptions personnalisées



```
CREATE OR REPLACE FUNCTION testErreur1(noDept INTEGER) RETURNS REAL AS $$  
  DECLARE  
    nbEmp INTEGER;  
    ref CONSTANT INTEGER := 85;  
  BEGIN  
    SELECT COUNT(*) INTO nbEmp FROM emp WHERE deptno = noDept;  
    IF nbEmp = 0 THEN  
      RAISE EXCEPTION 'Pas d'employé dans ce département';  
      -- RAISE EXCEPTION 'Pas d'employé dans le département %', noDept;  
    END IF;  
    RETURN ref / nbEmp ::REAL;  
  END  
$$ LANGUAGE plpgsql;  
  
SELECT testErreur1(99);
```

Exploitation des exceptions systèmes



```
CREATE OR REPLACE FUNCTION testErreur2(noDept INTEGER) RETURNS REAL AS $$  
  DECLARE  
    nbEmp INTEGER;  
    ref CONSTANT INTEGER := 85;  
  BEGIN  
    SELECT COUNT(*) INTO nbEmp FROM emp WHERE deptno = noDept;  
    RETURN ref / nbEmp ::REAL;  
  EXCEPTION  
    WHEN division_by_zero THEN  
      RAISE WARNING 'Pas d'employé dans le département %', noDept;  
      RETURN NULL;  
  END  
$$ LANGUAGE plpgsql;  
  
SELECT testErreur2(99);
```

Traitement par défaut



```
CREATE OR REPLACE FUNCTION testErreur3(noDept INTEGER) RETURNS REAL AS $$
```

```
DECLARE
```

```
    nbEmp INTEGER;
```

```
    ref CONSTANT INTEGER := 85;
```

```
BEGIN
```

```
    IF noDept <= 0 THEN
```

```
        RAISE EXCEPTION 'noDept ne peut pas être négatif';
```

```
    END IF;
```

```
    SELECT COUNT(*) INTO nbEmp FROM emp WHERE deptno = noDept;
```

```
    RETURN ref / nbEmp ::REAL;
```

```
EXCEPTION
```

```
    WHEN division_by_zero THEN
```

```
        RAISE WARNING 'Personne dans le département %', noDept;
```

```
        RETURN NULL;
```

```
    WHEN others THEN
```

```
        RETURN -1;
```

-- On peut aussi remplacer others par raise_exception

```
END
```

```
$$ LANGUAGE plpgsql;
```

Quelques autres exceptions système



Code	Nom
22004	null_value_not allowed
22003	numeric_value_out_of_range
22012	division_by_zero
23503	foreign_key_violation
23505	unique_violation
28P01	invalid_password
42501	insufficient_privilege
42883	undefined column
54011	too_many_columns
P0002	no_data_found
P0003	too_many_rows



- ✓ Introduction
- ✓ Bases du langage
- ✓ Curseurs
- ✓ Gestion des erreurs
- ▶ Déclencheurs
- ▶ SQL dynamique



- ▶ **Définition** : Fonction associée à une table et **exécutée automatiquement** lorsque des **événements** liés à des actions sur la table surviennent (mises à jour, principalement).
- ▶ Complètent les contraintes d'intégrité en permettant de créer des règles d'intégrité complexes.
 - Éléments des **bases de données actives**.

Principaux types de déclencheurs



	Insert	Delete	Update
Before	1	2	3
After	4	5	6

Fonctionnement des déclencheurs



Création/suppression de déclencheur



CREATE TRIGGER **nomDeclencheur**

BEFORE | AFTER
INSERT | DELETE | UPDATE | [INSERT] [[OR] DELETE] [[OR] UPDATE]

ON **nomTable**

FOR EACH ROW | FOR EACH STATEMENT

| : ou

EXECUTE PROCEDURE **nomFonction()**;

DROP TRIGGER **nomDeclencheur** ON **nomTable**;



- ▶ **NEW** : Enregistrement système contenant le n-uplet inséré ou modifié

Ex. INSERT INTO client VALUES (1, 'NouveauClient');

NEW.NumCli prend la valeur 1 dans un déclencheur sur client.

NEW.Nom prend la valeur 'NouveauClient' dans le déclencheur.

- ▶ **OLD** : Enregistrement contenant l'ancien n-uplet supprimé ou modifié

Ex. DELETE FROM client WHERE NumCli = 33;

OLD.NumCli prend la valeur 33 dans le déclencheur.

Exemple de déclencheur (1/2)



-- Emulation de clé primaire sur la table EMP

```
CREATE OR REPLACE FUNCTION checkEmpPK( ) RETURNS TRIGGER AS $$  
  DECLARE  
    n INTEGER;  
  BEGIN  
    -- La clé est-elle vide ?  
    IF NEW.empno IS NULL THEN  
      RAISE EXCEPTION 'La clé primaire doit avoir une valeur !';  
    END IF;
```

Exemple de déclencheur (2/2)



-- La clé existe-t-elle déjà ?

```
SELECT COUNT(empno) INTO n FROM emp
```

```
WHERE empno = NEW.empno;
```

```
IF n > 0 THEN
```

```
RAISE EXCEPTION 'Clé primaire déjà utilisée !';
```

```
END IF;
```

```
RETURN NEW;
```

```
END
```

```
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigEmpPK
```

```
BEFORE INSERT OR UPDATE ON emp
```

```
FOR EACH ROW EXECUTE PROCEDURE checkEmpPK( );
```

Quizz

Deux tables T et T' ont la même structure.
Chaque ajout de n-uplet dans T doit être répercuté dans T'.

Quel type de déclencheur faut-il utiliser ?

- A) BEFORE INSERT FOR EACH STATEMENT
- B) BEFORE INSERT FOR EACH ROW
- C) AFTER INSERT FOR EACH STATEMENT
- D) AFTER INSERT FOR EACH ROW

Répondre sur <https://toreply.univ-lille.fr>

Question n° 1878



N° de la question : 1878



- ✓ Introduction
- ✓ Bases du langage
- ✓ Curseurs
- ✓ Gestion des erreurs
- ✓ Déclencheurs
- ▶ SQL dynamique

SQL statique vs. SQL dynamique



► Exemples

- Fonction qui met la table EMP à jour (ex. changement de salaire)
⇒ **SQL statique** (la requête est connue à la compilation)
- Fonction qui met à jour une table dont le nom est un paramètre
⇒ **SQL dynamique** (la requête complète n'est pas connue à la compilation)

► Définition du SQL dynamique : Construction d'une requête SQL à la volée dans un bloc PL/pgSQL



- ▶ **Exécution :** EXECUTE requete [INTO resultat];
 - **requete** est une chaîne de caractères
 - **resultat** est une variable, un ensemble de variables ou un enregistrement

- ▶ **Note :**
 - Requête paramétrée : **valeurs** des **attributs** de la base de données
→ **requête statique**
 - Si l'on veut paramétrer des **objets** (tables, vues, nom d'attributs...)
→ **requête dynamique**

Exemple de requête dynamique



```
CREATE OR REPLACE FUNCTION tailleTable(nomTable VARCHAR)
RETURNS INTEGER AS $$
    DECLARE
        n INTEGER;
    BEGIN
        EXECUTE 'SELECT COUNT(*) FROM ' || nomTable INTO n;
        RETURN n;
    END
$$ LANGUAGE plpgsql;

SELECT tailleTable('EMP');
```

Curseurs dynamiques (1/2)



-- Exemple 1

```
CREATE OR REPLACE FUNCTION parcoursTable(nomTable VARCHAR) RETURNS VOID AS $$  
  DECLARE  
    dynCurs REFCURSOR;  
    nuplet RECORD;  
  BEGIN  
    OPEN dynCurs FOR EXECUTE 'SELECT * FROM ' || nomTable;  
    FETCH dynCurs INTO nuplet;  
    WHILE FOUND LOOP  
      -- Opérations sur nuplet  
      FETCH dynCurs INTO nuplet;  
    END LOOP;  
    CLOSE dynCurs;  
  END  
$$ LANGUAGE plpgsql;  
  
SELECT parcoursTable('EMP');
```



Une fonction ne peut pas
retourner un SETOF RECORD.

Curseurs dynamiques (2/2)



-- Exemple 2 : on utilise un SETOF VARCHAR (c'est moche, mais ça fonctionne...)

```
CREATE FUNCTION contenuTable(nomTable VARCHAR, nomID VARCHAR)
RETURNS SETOF VARCHAR AS $$
    DECLARE
        dynCurs REFCURSOR;
        id VARCHAR;
    BEGIN
        OPEN dynCurs FOR EXECUTE 'SELECT ' || nomID || ' FROM ' || nomTable;
        FETCH dynCurs INTO id;
        WHILE FOUND LOOP
            RETURN NEXT id;
            FETCH dynCurs INTO id;
        END LOOP;
        CLOSE dynCurs;
        RETURN;
    END
$$ LANGUAGE plpgsql;
```

```
SELECT contenuTable('EMP', 'EMPNO');
```

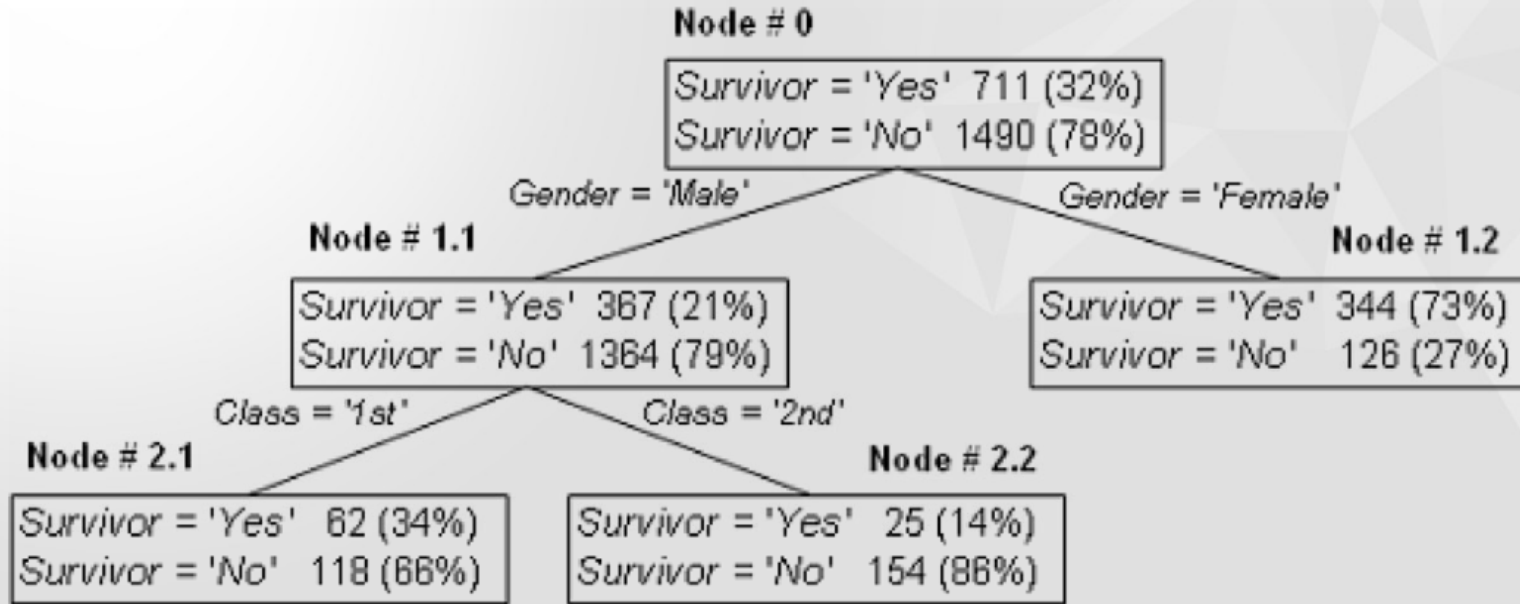
Exemple issu de la recherche (1/3)



F. Bentayeb, J. Darmont, C. Favre, C. Udréa, "Efficient On-Line Mining of Large Databases", International Journal of Business Information Systems, Vol. 2, No. 3, 2007, 328-350.

- **Problématique** (à l'époque !)
Contrainte de mémoire vive pour le *data mining* (arbres de décision)
- **Hypothèse**
Intégration du *data mining* dans un SGBD
- **Moyen**
Vues relationnelles
- **Résultat**
Utiliser un SGBD est plus lent,
mais permet de s'affranchir des problèmes de mémoire

Exemple issu de la recherche (2/3)



Rule #1: if Gender = 'Male' and Class = '1st' then $p(\text{Survivor} = \text{'Yes'}) = 34\%$

Rule #2: if Gender = 'Male' and Class = '2nd' then $p(\text{Survivor} = \text{'No'}) = 86\%$

Rule #3: if Gender = 'Female' then $p(\text{Survivor} = \text{'Yes'}) = 73\%$

Exemple issu de la recherche (3/3)



Node #0: CREATE VIEW **v0** AS SELECT Age, Gender, Class, Survivor
FROM **TITANIC**;

Node #1.1: CREATE VIEW **v11** AS SELECT Age, Class, Survivor
FROM **v0** WHERE **Gender = 'Male'**;

Node #1.2: CREATE VIEW **v12** AS SELECT Age, Class, Survivor
FROM **v0** WHERE **Gender = 'Female'**;

Node #2.1: CREATE VIEW **v21** AS SELECT Age, Survivor
FROM **v11** WHERE **Class = '1st'**;

Node #2.2: CREATE VIEW **v22** AS SELECT Age, Survivor
FROM **v11** WHERE **Class = '2nd'**;

Quizz

Lesquelles de ces requêtes
sont dynamiques ?

- A) `SELECT * FROM emp;`
- B) `SELECT * FROM emp WHERE empno = n;`
- C) `SELECT COUNT(*) FROM nomTable;`
- D) `DROP TABLE emp;`



N° de la question : 9371

Répondre sur <https://toreply.univ-lille.fr/>

Question n° 9371



- ✓ Introduction
- ✓ Bases du langage
- ✓ Curseurs
- ✓ Gestion des erreurs
- ✓ Déclencheurs
- ✓ SQL dynamique



Partie 2

XML/XQuery



Plan

- ▶ Introduction
- ▶ Documents XML
- ▶ Langage XQuery
 - XPath
 - Requêtes FLWOR
 - Requêtes complexes

Données structurées

- ▶ Données organisées en entités
- ▶ Entités similaires : forment des groupes (classes)
- ▶ Entités du même groupe : même description (attributs)
- ▶ Pour toutes les entités d'un groupe :
 - Chaque attribut a le même type
 - Chaque valeur d'attribut a la même taille
 - Tous les attributs sont présents
 - Les attributs sont toujours dans le même ordre
- ▶ Données structurés : décrites par un schéma
 - Généralement stockées dans des bases de données

Données non structurées

- ▶ Données de **tous types**
- ▶ Données qui ne suivent **aucun schéma ni séquence prédéterminé**
- ▶ Données qui ne suivent **aucune règle**
- ▶ Données qui **ne sont pas prévisibles**

- ▶ Exemples de données non-structurées :
 - Textes
 - Images
 - Vidéos
 - Sons

Données semi-structurées

Bases de données

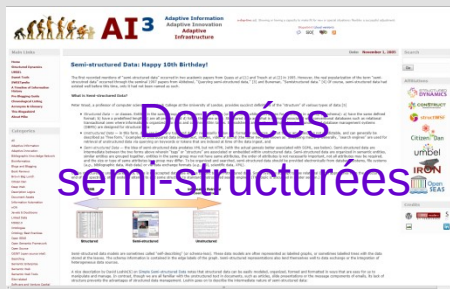
Documents

Années 1990

Langages de requête

Moteurs de recherche

N° employé	Date embauche	Nom	Prénoms	Adresse	Ville	Province	Code postal	Date naissance
0009	12-jan-89	Chrast	Eric	1075 rue Dufluth	Montréal	QC	H4T 2L9	06-sep-49
0101	20-mai-89	Reys	Kimberly	899 de Laramie	Donat	QC	H9P 3T2	15-déc-53
0102	05-mai-89	Lampro	Claude	388 rue Ste-Pierre	Montréal	QC	H9P 1A6	03-mai-65
0103	14-aoû-89	Bédard	Jake	440 rue Talbot	Lasal	QC	H7P 3G8	14-aoû-45
0104	07-mai-89	Ford	Jean-Paul	5380 rue Racine	Montréal-Hor	QC	H2T 2P5	25-aoû-59
0230	19-mars-89	Larocque	Monique	892 rue Victoria	Longueuil	QC	J3P 2T2	02-mar-37
0234	12-aoû-89	Therrien	Jocelyne	502 rue Verchères	Coastal-Park	QC	J4V 2V5	02-mar-71
0387	21-aoû-89	Pinault	Marcel	8885 boul. St-Laurent	Montréal	QC	H3T 2W3	06-mars-54
0390	12-mai-89	Larocque	Hedèle	97 rue du Collège	Ville St-Laurent	QC	H4P 2B8	07-aoû-44
0434	04-juin-89	Connet	Mark	1213 rue de Gaulle	St-Hubert	QC	J4T 3L9	28-juin-69
0500	10-juin-89	Roy	Alain	888 rue Curville	Outremont	QC	H3T 2V2	25-aoû-67
0560	15-juin-89	Sérand	Marth	2575 rue Victoria	St-Lambert	QC	J4L 6P7	03-aoû-74
0728	15-aoû-89	Bédard	Sébastien	3529 rue Rouville	Ste-Thérèse	QC	J3P 3V5	14-aoû-53
0839	14-aoû-89	Leveson	Josée	684 de Montclair	St-Basile	QC	J4L 2V4	12-sept-71
1142	10-aoû-89	Faucher	Maths	45 rue Handel	Candiac	QC	J3P 2B7	31-aoû-54
1181	14-aoû-89	Leveson	Josée	621 rue Comstock	Lasal	QC	H7P 3G7	02-aoû-68
1241	07-jan-90	Sin	Eric	331 rue	Montréal	QC	J3V 5P9	25-nov-50
1300	01-aoû-90	Ver	Francis	5011 rue des Rangées	Montréal	QC	H3P 2V3	06-aoû-58
1341	04-mars-90	Verm	Eric	1075 rue Dufluth	Montréal	QC	H4T 1C5	12-déc-62
1440	05-nov-90	Quinn	Jean-Claude	2889 rue Fleury	Montréal	QC	H4T 1C5	12-déc-62
1481	01-aoû-90	Ward	Hayne	381 rue Labarre	Montréal	QC	J4Z 1T7	07-sept-68
1543	09-jan-91	Blanchard	André	1075 rue Dufluth	Montréal	QC	H4T 2V9	30-mai-66
0800	01-aoû-91	Blanchard	André	1075 rue Dufluth	Montréal	QC	H4T 2V9	30-mai-66
1071	14-aoû-91	Blanchard	André	1075 rue Dufluth	Montréal	QC	H4T 2V9	30-mai-66
2200	15-aoû-91	Blanchard	André	1075 rue Dufluth	Montréal	QC	H4T 2V9	30-mai-66
2112	04-nov-91	Leveson	Camille	1241 rue Handel	Montréal	QC	H3T 2V3	03-aoû-70
2315	04-nov-91	Coutard	Sophie	2889 boul. Décarie	Ville St-Laurent	QC	H4T 2Z2	29-juin-69
2320	04-nov-91	Ingrat	Hathale	305 rue Hutchison	Outremont	QC	H3T 2T4	15-aoû-61
2321	01-aoû-91	Mist	Bernard	8997 rue Viger	Montréal	QC	H3V 2B5	10-mai-73
2322	01-aoû-91	Ward	Gilbert	438 rue Dawson	Montréal	QC	H3L 6L9	19-aoû-67
2324	01-aoû-91	Coutard	Patrick	7228 boul. St-Joseph	Montréal	QC	H3Z 3P5	13-aoû-51
2341	05-aoû-91	Achambault	Manon	673 rue Beaupré	Montréal	QC	H4G 2T1	31-déc-62
2389	04-aoû-92	Blanchard	André	2875 Visseaud	Brossard	QC	J4P 2V3	29-juin-69
2480	04-mars-92	Lagard	Mirella	592 St-Joseph	Montréal	QC	H2T 5G2	26-mar-73
2500	04-mars-92	Provencher	Phane	2687 Christophe-Colum	Montréal	QC	H2T 5G1	09-aoû-72
3501	04-aoû-92	Trudler	Laurent	1955 rue Villeneuve	Longueuil	QC	J3P 8T3	11-juin-72
3502	04-aoû-92	Trudler	Alexandra	1955 rue Villeneuve	Longueuil	QC	J3P 8T3	11-juin-72
3503	15-nov-92	Vissard	Francis	2602 rue Perron	Montréal	QC	H4T 3B9	30-mars-69
3560	21-nov-92	Ward	Hans	62 rue Ste-Barbe	Verdun	QC	J5G 4G2	20-aoû-72
3561	21-nov-92	Gagnon	Eric	2251 rue Belvédère	Montréal	QC	H3P 2J9	06-aoû-66
3580	09-déc-92	Demers	Maria-Claire	1275 rue Van Horne	Outremont	QC	H3T 6P5	06-nov-58
3800	08-déc-92	Rosselle	Celine	7228 boul. St-Joseph	Montréal	QC	H3V 6L3	29-juin-74



Données semi-structurées

L'ESTA*, la nouvelle procédure d'autorisation de voyage pour aller ou transiter aux États-Unis.
 De nouvelles formalités d'entrée et de transit sur le territoire américain vont entrer en vigueur dès le 12 janvier 2009. Cela concerne tous les voyageurs qui sont exemptés de visa.
 En Europe, ce sont les ressortissants de 22 pays bénéficiaires du Programme d'Exemption de Visa, dont la France, qui sont concernés par cette réforme aux États-Unis.
 Désormais, il faudra avant d'embarquer pour un voyage à destination des États-Unis ou avec un transit aux États-Unis, obtenir une autorisation de voyage "ESTA".
 À partir du 12 janvier prochain, avant de monter dans un avion ou d'embarquer à bord d'un paquebot à destination des États-Unis, tout voyageur devra remplir via Internet le nouveau formulaire d'inscription en ligne, l'ESTA.
 Attention, cette formalité est obligatoire pour :

TOUT voyageur, qu'il se rende sur le territoire américain ou qu'il y transite
 que ce soit pour un séjour de tourisme ou d'affaires de moins de 90 jours
 qu'il soit majeur ou mineur (donc même pour votre bébé de 8 mois et 1 an par exemple, accompagné ou non)
 quel que soit le mode de transport (à bord d'une compagnie aérienne ou maritime).

L'autorisation ESTA n'est pas un visa.

Non, l'autorisation "ESTA" n'est pas un visa. Elle instaure un contrôle supplémentaire préalable afin de faciliter la tâche des officiers d'immigration pour l'identification des voyageurs à risque.
 L'autorisation de voyage ESTA assure la libre admission sur le territoire américain.
 Ce n'est pas une garantie d'entrée en territoire américain.
 C'est une obligation pour les citoyens des pays membres du Programme d'Exemption de Visa.
 Attention :

Dans tous les cas, l'officier d'immigration au poste frontière se prononce sur l'admission comme c'est déjà le cas aujourd'hui.

Quelle est la procédure d'obtention d'une autorisation ESTA ?

Le système "ESTA" utilise Internet uniquement.

Pour soumettre votre demande d'autorisation ESTA depuis le 1er août 2008, vous devez vous rendre sur le site suivant : <https://esta.cbp.dhs.gov> et suivre les instructions pour répondre aux questions posées.
 L'accès au site est gratuit et disponible en français (sélectionnez pour cela la langue française dans le menu déroulant en haut à droite).
 Attention :

Si vous n'avez pas accès à Internet, vous devrez recourir à une tierce personne de votre entourage ou à un agent de voyage. Vous restez légalement responsable des réponses fournies.
 La plupart des voyageurs obtiendront la confirmation de l'autorisation de façon quasi immédiate ce qui permet les voyages de dernière minute.

Données semi-structurées

- ▶ Données organisées en entités sémantiques
- ▶ Entités similaires : groupes
- ▶ Entités du même groupe : peuvent ne pas avoir les mêmes attributs
- ▶ Pour toutes les entités d'un groupe :
 - Un même attribut peut avoir des types différents
 - Une même valeur d'attribut peut avoir des tailles différentes
 - Des attributs peuvent être manquants ou dupliqués
 - L'ordre des attributs n'est pas nécessairement important
- ▶ Données semi-structurées : **autodescriptives**
 - Pages web, documents XML, courriels...

Structurées

Non-structurées

Exemple de données semi-structurées

▶ Nom

Jérôme Darmont

▶ Courriel

jerome.darmont@univ-lyon2.fr
jerome.darmont@msh-lse.fr

▶ Courriel

sabine.loudcher@univ-lyon2.fr

▶ Nom

– Prénom

Loudcher

– Nom de famille

Sabine

▶ Nom

Walid Bechkit

▶ Affiliation

Université Lyon 2

Modèle de données semi-structuré

► Avantages

- Peut représenter des informations issues de sources de données qui ne peuvent pas être contraintes par un schéma
- Format flexible pour l'interopérabilité
- Permet de voir des données structurées comme semi-structurées (Web)
- Schéma facilement évolutif

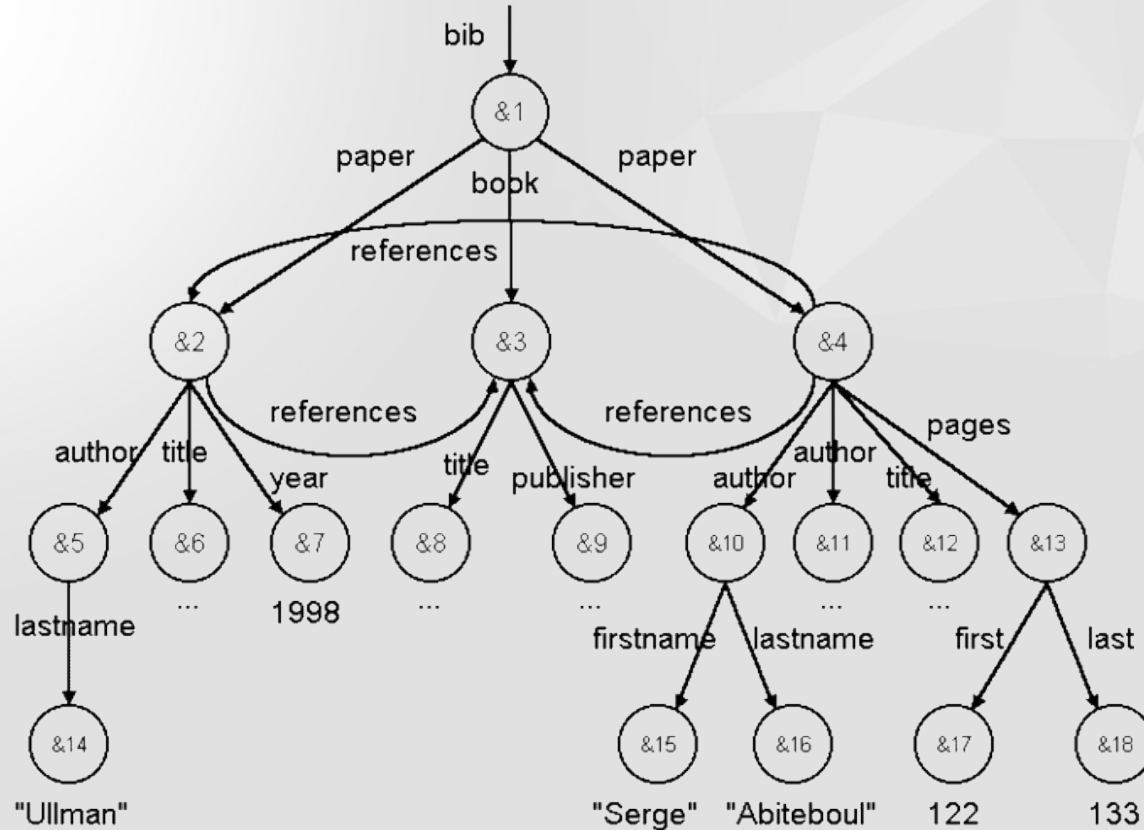
► Inconvénients

- Performance des requêtes sur données à grande échelle

► Représentations

- Electronic Data Interchange (EDI) : domaine financier
- Object Exchange Model (OEM) : modèle basé sur les graphes
- SGML, HTML et XML
- JSON, YAML...

Exemple de graphe OEM



Gestion de données semi-structurées

► Modélisation des données semi-structurées

- Graphes (OEM) Modèle conceptuel
- DTD, XML-Schema Modèle logique
- XML Modèle physique

► Requête des données semi-structurées

- XPath
- XQuery

► Stockage des données semi-structurées

- Fichiers plats
- Bases de données relationnelles, relationnelles-objets ou natives XML

Références

- ▶ Peter Wood, Birkbeck University of London
Semi-Structured Data
<http://www.dcs.bbk.ac.uk/~ptw/>
- ▶ Mike Bergman, Structured Dynamics LLC
Semi-structured Data: Happy 10th Birthday!
<http://www.mkbergman.com/153/semi-structured-data-happy-10th-birthday/>

Sondage express

Pensez-vous avoir compris la
Différence entre données structurées,
non structurées et semi-structurées ?



N° de la question : 7117

Répondre sur <https://toreply.univ-lille.fr/>

Question n° 7117

Pourquoi pas JSON, YAML et les BD NoSQL ?

- ▶ SGBD NoSQL type MongoDB : plus efficaces que XML/XQuery si besoin de **stockage distribué**
- ▶ Pour l'*analytics*, XQuery est plus puissant (opérateur // inexistant dans MongoDB)
- ▶ Les SGBD NoSQL sont étudiés en M2 !



Plan

✓ Introduction

▶ Documents XML

▶ Langage XQuery

- XPath
- Requêtes FLWOR
- Requêtes complexes

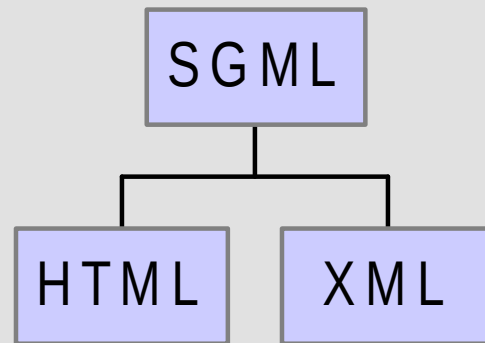


► XML : Extensible Markup Language

- Format de structuration de données et de documents Internet issu de SGML
- Définition, gestion, création, transmission et partage de documents

► XML est un standard du W3C

- 1996 : Brouillon
- 1997 : XML 1.0
- 2004 : XML 1.1



Exemple de document XML



```
<?xml version="1.1" encoding="utf-8" ?> <!-- Prologue (obligatoire) -->
<annuaireProfs> <!-- Élément racine -->
  <!-- Sous-éléments -->
  <prof>
    <nom>Jérôme Darmont</nom>
    <courriel>jerome.darmont@univ-lyon2.fr</courriel>
    <cours>Bases de données avancées</cours>
    <cours>Programmation web backend</cours>
  </prof>
  <prof>
    <nom>Walid Bechkit</nom>
    <courriel>walid.bechkit@univ-lyon2.fr</courriel>
    <cours>Programmation orientée objet</cours>
  </prof>
  <!-- Etc. -->
</annuaireProfs> <!-- Balise de fin -->
```

Ensemble d'éléments imbriqués
matérialisés par des balises

Règles d'écriture d'un document XML (1/2)

XML



- ▶ Un document XML a un et un seul élément racine.
- ▶ Les éléments doivent être correctement emboîtés (les balises ouvrantes et fermantes ne doivent pas se chevaucher).
- ▶ Tout élément doit avoir une balise ouvrante et une balise fermante.
- ▶ Le nom d'un élément doit être identique dans la balise ouvrante et la balise fermante.
- ▶ Les noms d'éléments sont sensibles à la casse. Ils doivent commencer par une lettre ou par _ suivi(e) de lettres, de chiffres, de ., de - ou de _.

Règles d'écriture d'un document XML (2/2)

XML



- ▶ Les noms d'éléments commençant par XML (dans toutes combinaisons de minuscules et majuscules) sont réservés à des fins de standardisation.
- ▶ Un document XML respectant ces règles est dit **bien formé**.
- ▶ Un document XML **doit** être bien formé !
- ▶ Un document XML peut de plus être **valide** s'il se conforme à la structure définie dans une DTD ou un Schéma XML.

Document Type Definition

XML Schema



DTD	XML Schema
Syntaxe spécifique (non XML)	Exprimé dans un document XML
Typage faible	Typage fort
Modélisation partielle impossible	Modélisation partielle possible
Interprétable par un·e utilisateur·trice humain·e	Conçu pour des traitements automatiques



- ▶ Caractères non-autorisés : `< & []>`
- ▶ Éléments emboîtés : profondeur non limitée
 - Ex.

```
<annuaireProfs>  
  <prof>  
    <nom>  
      <nom_famille>Zighed</nom_famille>  
      <prenom>Abdelkader</prenom>  
      <prenom>Djamel</prenom>  
    </nom>  
  </prof>  
</annuaireProfs>
```



- ▶ **Section CDATA** : Bloc de texte libre dans lequel seule la chaîne `]]>` est interdite

- Ex.

```
<nom>  
    <![CDATA[<Darmont> & <Loudcher>]]>  
</nom>
```


- ▶ **Élément vide** : sans contenu

- Ex.

```
<courriel></courriel>
```
- Formulation équivalente

```
<courriel />
```



- ▶ **Attributs** : données associées à un élément, complémentaires du contenu
- ▶ **Définition** : couple nom/valeur dans la balise ouvrante de l'élément
 - Ex. `<bureau campus="PdA" batiment="K">067</bureau>`

attribut valeur attribut valeur
 - ▶ Les attributs sont possibles dans les éléments vides.
 - Ex. `<image source="ma-bobine.png" />`



► Que choisir ?

- `<prof>`
 `<nom>Darmont</nom>`
 `</prof>`
- `<prof nom="Darmont" />`

► 4 principes pour décider

- D'après Uche Ogbuji, Fourthought, Inc.



Contenu d'élément XML vs. attributs (2/2)

Principe	Élément	Attribut
Contenu principal	Information essentielle	Information périphérique
Information structurée	Information hiérarchisée	Information atomique
Lisibilité	Utilisateur·trice humain·e	Traitement automatique
Relation élément/attribut	Information précisée par une autre	

Exemple de relation élément/attribut

```
<stock>
  <produit quantité="1500">
    <nom>Ordinateur</nom>
  </produit>
  <produit quantité="500">
    <nom>Imprimante</nom>
  </produit>
</stock>
```

Quizz

Dans un document XML, les données sont stockées dans :

- A) Les éléments
- B) Les attributs



N° de la question : 9088

Répondre sur <https://toreply.univ-lille.fr/>

Question n° 9088

Plan

✓ Introduction

✓ Documents XML

► Langage XQuery

- XPath
- Requêtes FLWOR
- Requêtes complexes



- ▶ Langage de requêtes pour **données XML**
- ▶ Similarités avec **SQL**
- ▶ Conçu par le **W3C**
- ▶ Basé sur des expressions **XPath** (mêmes modèle de données, fonctions, opérateurs)
- ▶ **Versions**
 - 2007 : XQuery 1.0 \supset XPath 2.0
 - 2017 : XQuery 3.1 \supset XPath 3.1
- ▶ Standardisation en cours (**standard de fait**)
- ▶ Soutenu par les **éditeurs de SGBD** (Oracle, Microsoft, IBM...)

Document XML exemple (1/2)

XQ



```
<?xml version="1.1" encoding="utf-8" ?> <!-- le fichier s'appelle films.xml -->
```

```
<catVOD>
```

```
  <film sigJeune="-12">
    <titre>Blade runner</titre>
    <realisateur>Ridley Scott</realisateur>
    <annee>1982</annee>
    <langue>Anglais</langue>
    <prix>4.79</prix>
  </film>
```

```
  <film>
    <titre>La grande vadrouille</titre>
    <realisateur>Gérard Oury</realisateur>
    <annee>1966</annee>
    <duree>122</duree>
    <langue>Français</langue>
    <prix>9.82</prix>
  </film>
```

```
<!-- (...) -->
```

Document XML exemple (2/2)



```
<film sigJeune="-10">  
  <titre>Le fabuleux destin d'Amélie Poulain</titre>  
  <realisateur>Jean-Pierre Jeunet</realisateur>  
  <annee>2001</annee>  
  <duree>120</duree>  
  <langue>Français</langue>  
  <prix>4.99</prix>  
</film>
```

```
<film sigJeune="-12">  
  <titre>The big Lebowski</titre>  
  <realisateur>Ethan Coen</realisateur>  
  <realisateur>Joel Coen</realisateur>  
  <annee>1997</annee>  
  <duree>112</duree>  
  <langue>Français</langue>  
  <langue>Anglais</langue>  
  <prix>9.82</prix>  
</film>
```

```
</catVOD>
```

Plan

✓ Introduction

✓ Documents XML

✓ Langage XQuery

- XPath
- Requêtes FLWOR
- Requêtes complexes



- Document XML entier
`doc("films.xml")/catVOD`

Résultat

Tout le document

- Un élément donné
`doc("films.xml")/catVOD/film`
`doc("films.xml")/catVOD/film/titre`

Résultat

`<titre>Blade runner</titre>`

`<titre>La grande vadrouille</titre>`

`<titre>Le fabuleux destin d'Amélie Poulain</titre>`

`<titre>The big Lebowski</titre>`





- ▶ Un attribut donné

`doc("films.xml")/catVOD/film/data(@sigleune)`

Résultat

-12 -10 -12

- ▶ Un élément donné quel que soit son niveau hiérarchique

`/catVOD//titre`

`//titre`

Résultat

`<titre>Blade runner</titre>`

`<titre>La grande vadrouille</titre>`

`<titre>Le fabuleux destin d'Amélie Poulain</titre>`

`<titre>The big Lebowski</titre>`



► Tous les sous-éléments d'un élément //film/*

Résultat

```
<titre>Blade runner</titre>  
<realisateur>Ridley Scott</realisateur>  
<annee>1982</annee>  
<duree>117</duree>  
<langue>English</langue>  
<prix>4.79</prix>  
<titre>La grande vadrouille</titre>  
<realisateur>Gérard Oury</realisateur>  
<annee>1966</annee>  
<duree>122</duree>  
<langue>French</langue>  
<prix>9.82</prix>
```

Etc.



- ▶ ^{ie}, dernier, i premiers/derniers éléments

```
//film[1]
```

```
//film[last()]
```

```
//film[position() < 3]/titre
```

Résultat

```
<titre>Blade runner</titre>
```

```
<titre>La grande vadrouille</titre>
```

- ▶ Éléments possédant un sous-élément ou attribut donné

```
//film[duree]/titre
```

Résultat

```
<titre>La grande vadrouille</titre>
```

```
<titre>Le fabuleux destin d'Amélie Poulain</titre>
```

```
<titre>The big Lebowski</titre>
```

```
//film[@sigJeune]
```

- ▶ Condition sur un élément ou un attribut

`//film[prix < 15]`

`//film[@signJeune = "-10" and prix < 5]/titre`

Résultat

`<titre>Le fabuleux destin d'Amélie Poulain</titre>`

- ▶ Combinaison de chemins

`//titre | //prix`

Résultat

`<titre>Blade runner</titre>`

`<titre>La grande vadrouille</titre>`

`<titre>Le fabuleux destin...</titre>`

`<titre>The big Lebowski</titre>`

`<prix>4.79</prix>`

`<prix>9.82</prix>`

`<prix>4.99</prix>`

`<prix>9.82</prix>`



- ▶ Fonctions d'accès : `data()`...
- ▶ Fonctions numériques : `abs()`, `floor()`, `ceiling()`, `round()`, `number()`...
- ▶ Fonctions de chaînes : `string-length()`, `upper-case()`, `lower-case()`, `normalize-space()`, `substring()`, `substring-after()`, `replace()`, `contains()`...
- ▶ Fonctions temporelles : `day-from-date()`, `year-from-date()`...
- ▶ Fonctions de séquences : `exists()`, `distinct-values()`, `reverse()`, `sort()`...
- ▶ Fonctions contextuelles : `last()`, `position()`...
- ▶ Fonctions booléennes : `not()`...
- ▶ Fonctions d'agrégat : `count()`, `sum()`, `avg()`, `max()`, `min()`...

Fonctions XQuery (les mêmes que XPath !) (1/2)

XQ



- ▶ Fonctions d'accès : `data()`...
- ▶ Fonctions numériques : `abs()`, `floor()`, `ceiling()`, `round()`, `number()`...
- ▶ Fonctions de chaînes : `string-length()`, `upper-case()`, `lower-case()`, `normalize-space()`, `substring()`, `substring-after()`, `replace()`, `contains()`...
- ▶ Fonctions temporelles : `day-from-date()`, `year-from-date()`...
- ▶ Fonctions de séquences : `exists()`, `distinct-values()`, `reverse()`, `sort()`...
- ▶ Fonctions contextuelles : `last()`, `position()`...
- ▶ Fonctions booléennes : `not()`...
- ▶ Fonctions d'agrégat : `count()`, `sum()`, `avg()`, `max()`, `min()`...



Exemple d'appel à une fonction

```
for $x in //film/titre  
let $titreMAJ := upper-case($x)  
return <film>{$titreMAJ}</film>
```

Résultat

```
<film>BLADE RUNNER</film>  
<film>LA GRANDE VADROUILLE</film>  
<film>LE FABULEUX DESTIN D'AMÉLIE POULAIN</film>  
<film>THE BIG LEBOWSKI</film>
```


Plan

✓ Introduction

✓ Documents XML

✓ Langage XQuery

✓ XPath

✓ Requêtes FLWOR

Requêtes complexes

Clause Group by (XQuery 3)



Regroupement sur un critère

```
for $d in /catVOD/film
group by $z := $d/@sigJeune
let $p := avg($d/prix) (: pour la lisibilité du résultat :)
return <sigJeune value="{ $z }">
        <prix_moyen>{p}</prix_moyen>
</sigJeune>
```

Regroupement multiple

```
for $d in /catVOD/film
group by $z := $d/@sigJeune, $a := $d/annee
let $p := avg($d/prix)
return <groupe sigJeune="{ $z }" annee="{ $a }">
        <prix_moyen>{ $p }</prix_moyen>
</groupe>
```

Jointures – Documents exemples (1/3)

XQ



```
<?xml version="1.1" encoding="utf-8" ?>      <!-- document 1 : clients.xml -->
```

```
<clients>
```

```
  <client id="1">
    <nom>Loudcher</nom>
    <prenom>Sabine</prenom>
    <adresse>Bureau K073</adresse>
  </client>
```

```
  <client id="2">
    <nom>Bentayeb</nom>
    <prenom>Fadila</prenom>
    <adresse>Bureau K061</adresse>
  </client>
```

```
  <client id="3">
    <nom>Darmont</nom>
    <prenom>Jérôme</prenom>
    <adresse>Bureau K067</adresse>
  </client>
```

```
</clients>
```

Jointures – Documents exemples (2/3)

XQ



```
<?xml version="1.1" encoding="utf-8" ?> <!-- document 2 : produits.xml -->
```

```
<produits>
```

```
  <produit id="10">  
    <nom>Ordinateur</nom>  
  </produit>
```

```
  <produit id="20">  
    <nom>Moniteur</nom>  
  </produit>
```

```
  <produit id="30">  
    <nom>Imprimante</nom>  
  </produit>
```

```
</produits>
```

Jointures – Documents exemples (3/3)

XQ



```
<?xml version="1.1" encoding="utf-8" ?> <!-- document 3 : commandes.xml -->
<commandes>
  <commande cli-id="1" prod-id="10">
    <quantite>3</quantite>
  </commande>
  <commande cli-id="1" prod-id="20">
    <quantite>15</quantite>
  </commande>
  <commande cli-id="2" prod-id="10">
    <quantite>7</quantite>
  </commande>
  <commande cli-id="2" prod-id="30">
    <quantite>10</quantite>
  </commande>
  <commande cli-id="3" prod-id="30">
    <quantite>5</quantite>
  </commande>
</commandes>
```



Exemple

```
for $c in doc("clients.xml")//client,  
    $o in doc("commandes.xml")//commande  
where $c/@id = $o/@cli-id  
return    <nom>{data($c/nom)}</nom>  
          <prenom>{data($c/prenom)}</prenom>  
          <qte>data($o/quantite)}</qte>
```

Résultat

```
<nom>Loudcher</nom> <prenom>Sabine</prenom> <qte>3</qte>  
<nom>Loudcher</nom> <prenom>Sabine</prenom> <qte>15</qte>  
<nom>Bentayeb</nom> <prenom>Fadila</prenom> <qte>7</res>  
<nom>Bentayeb</nom> <prenom>Fadila</prenom> <qte>10</qte>  
<nom>Darmont</nom> <prenom>Jérôme</prenom> <qte>5</qte>
```

Jointures de documents XML (2/3)

XQ



Exemple

```
for $c in doc("clients.xml")//client,  
    $o in doc("commandes.xml")//commande,  
    $p in doc("produits.xml")//produit  
where $c/@id = $o/@cli-id  
and $o/@prod-id = $p/@id  
return    <nom-cli>{data($c/nom)}</nom-cli>  
          <prenom>{data($c/prenom)}</prenom>  
          <qte>{data($o/quantite)}</qte>  
          <nom-prod>{data($p/nom)}</nom-prod>
```

Résultat

```
<nom-cli>Loudcher</nom-cli> <prenom>Sabine</prenom> <qte>3</qte> <nom-prod>Ordinateur</nom-prod>  
<nom-cli>Loudcher</nom-cli> <prenom>Sabine</prenom> <qte>15</qte> <nom-prod>Moniteur</nom-prod>  
<nom-cli>Bentayeb</nom-cli> <prenom>Fadila</prenom> <qte>7</qte> <nom-prod>Ordinateur</nom-prod>  
<nom-cli>Bentayeb</nom-cli> <prenom>Fadila</prenom> <qte>10</qte> <nom-prod>Imprimante</nom-prod>  
<nom-cli>Darmont</nom-cli> <prenom>Jérôme</prenom> <qte>5</qte> <nom-prod>Imprimante</nom-prod>
```

Variantes avec les conditions de jointures exprimées en prédicats de chemins

```
for $c in doc("clients.xml")//client,  
    $o in doc("commandes.xml")//commande[@cli-id=$c/@id]  
return <nom>{data($c/nom)}</nom> <prenom>{data($c/prenom)}</prenom>  
    <qte>{data($o/quantite)}</qte>
```

```
for $c in //client,  
    $p in //produit,  
    $o in //commande[@cli-id=$c/@id and @prod-id=$p/@id]  
return <nom-cli>{data($c/nom)}</nom-cli> <prenom>{data($c/prenom)}</prenom>  
    <qte>{data($o/quantite)}</qte> <nom-prod>{data($p/nom)}</nom-prod>
```




FLWOR
=
lisibilité



XPath
=
concision

Quizz



N° de la question : 7246

Quelles clauses de requêtes FLWOR sont également exprimables en XPath ?

- A) For
- B) Where
- C) Order by
- D) Group by

Répondre sur <https://toreply.univ-lille.fr/>

Question n° 7246

Plan

✓ Introduction

✓ Documents XML

✓ Langage XQuery

✓ XPath

✓ Requêtes FLWOR

✓ Requêtes complexes



Sondage express

Que pensez-vous de ce cours ?



N° de la question : 2617

Répondre sur <https://toreply.univ-lille.fr>

Question n° 2617