Langages de requêtes

M1 Informatique
Année 2010-2011
Jérôme Darmont

http://eric.univ-lyon2.fr/~jdarmont/

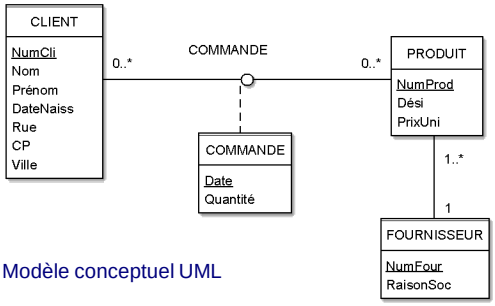
Plan du cours

- Introduction
- Algèbre relationnelle
- Langage SQL
- Langage XQuery

Objectifs du cours

- Logique de l'interrogation de bases de données relationnelles
  - ➲ Algèbre relationnelle
- Mise en œuvre (*création*, *mise à jour* et *interrogation*) de BD relationnelles
  - ➲ Langage SQL
- Interrogation de bases de données XML
  - ➲ Langage XQuery

Base de données exemple



- Modèle conceptuel UML

Base de données exemple

- Modèle logique relationnel

CLIENT (NumCli, Nom, Prénom, DateNaiss, Rue, CP, Ville)
PRODUIT (NumProd, Dési, PrixUni, NumFour#)
FOURNISSEUR (NumFour, RaisonSoc)

COMMANDE (NumCli#, NumProd#, Date, Quantité)

Clés primaires Clés étrangères#

Plan du cours

- ✓ Introduction
- Algèbre relationnelle
- Langage SQL
- Langage XQuery

Définitions

- Une relation R est un ensemble d'attributs $\{A_1, A_2, \ldots, A_n\}$.

  ex. La relation FOURNISSEUR est l'ensemble des attributs {NumFour, RaisonSoc}.

- Chaque attribut A_i prend ses valeurs dans un domaine $dom(A_i)$.

  ex. Quantité ∈ [1, 200]
  Ville ∈ {'Lyon', 'Grenoble', 'Saint-Etienne', ...}

Définitions

- Un n-uplet t est un ensemble de valeurs $t = \langle V_1, V_2, \ldots, V_n \rangle$ où soit :
  - $V_i \in dom(A_i)$
  - V_i est la valeur nulle (NULL).
  - ex. <214, 'Super-produit', 23.75> est un n-uplet de la relation PRODUIT.
- Notation : $R (A_1, A_2, \ldots, A_n)$

  ex. FOURNISSEUR (NumFour, RaisonSoc)

Définitions

- Algèbre relationnelle : ensemble d'opérateurs qui s'appliquent aux relations
- Résultat : nouvelle relation qui peut à son tour être manipulée

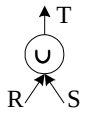
➲ L'algèbre relationnelle permet d'effectuer des recherches dans les relations.

Opérateurs ensemblistes

- Union : $T = R \cup S$ ou T = UNION (R, S)
  R et S doivent avoir même schéma.

  ex. R et S sont les relations PRODUIT de deux sociétés qui fusionnent et veulent unifier leur catalogue.

  Notation graphique :

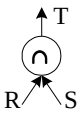


Opérateurs ensemblistes

- Intersection : $T = R \cap S$ ou T = INTERSECT (R, S)
  R et S doivent avoir même schéma.

  ex. Permet de trouver les produits communs aux catalogues de deux sociétés.

  Notation graphique :

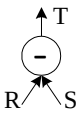


Opérateurs ensemblistes

- Différence : T = R - S ou T = MINUS (R, S)
  R et S doivent avoir même schéma.

  ex. Permet de retirer les produits de la relation S existant dans la relation R.

  Notation graphique :

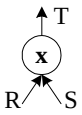


Opérateurs ensemblistes

- Produit cartésien : T = R x S ou T = PRODUCT (R, S)

  Associe chaque n-uplet de R à chaque n-uplet de S.

  Notation graphique :



Produit cartésien

ex.

NumProd	Dési
0	P1
1	P2

X

NumFour	RaisonSoc
10	F1
20	F2
30	F3

=

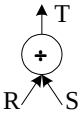
NumProd	Dési	NumFour	RaisonSoc
0	P1	10	F1
1	P2	10	F1
0	P1	20	F2
1	P2	20	F2
0	P1	30	F3
1	P2	30	F3

Opérateurs ensemblistes

- Division : T = R ÷ S ou T = DIVISION (R, S)

  R (A_1, A_2, …, A_n) S (A_{p+1}, …, A_n)
  T (A_1, A_2, …, A_p) contient tous les n-uplets tels que leur concaténation à chacun des n-uplets de S donne toujours un n-uplet de R.

  Notation graphique :



Division

ex.

NumCli	Date	Quantité
1	22/09/991	
1	22/09/995	
1	10/10/992	
2	15/10/999	
3	22/09/991	
3	10/10/992	

÷

Date	Quantité
22/09/991	
10/10/992	

=

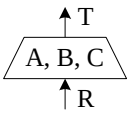
NumCli
1
3

Opérateurs spécifiques

- Projection : T = Π <A, B, C> (R)
  ou T = PROJECT (R / A, B, C)

  T ne contient que les attributs A, B et C de R.
  ex. Noms et prénoms des clients.

  Notation graphique :

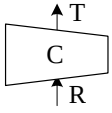


Opérateurs spécifiques

- Restriction : T = σ <C> (R) ou T = RESTRICT (R / C)

  T ne contient que les attributs de R qui satisfont la condition C.
  ex. C = Clients qui habitent Lyon.

  Notation graphique :

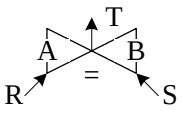


Opérateurs spécifiques

- Jointure naturelle : T = R >< S ou T = JOIN (R, S)

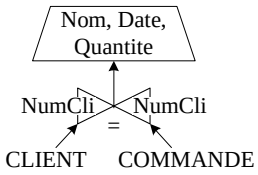
  Produit cartésien R x S et restriction A = B sur les attributs A ∈ R et B ∈ S.

  Notation graphique :



Jointure

Exemple : *Commandes avec le nom du client et pas seulement son numéro*



Requête : enchaînement d'opérations

Jointure

Décomposition des opérations

CL

NumCli	*Nom*
1	C1
2	C2
3	C3

X

CO

NumCli	*Date*	*Quantité*
1	22/09/991	
3	22/09/995	
3	22/09/992	

Jointure

=

CL.NumCli	*Nom*	*CO.NumCli*	*Date*	*Quantité*
1	C1	1	22/09/991	
2	C2	1	22/09/991	
3	C3	1	22/09/991	
1	C1	3	22/09/995	
2	C2	3	22/09/995	
3	C3	3	22/09/995	
1	C1	3	22/09/992	
2	C2	3	22/09/992	
3	C3	3	22/09/992	

Jointure

CL >< CO

CL.NumCli	*Nom*	*CO.NumCli*	*Date*	*Quantité*
1	C1	1	22/09/991	
3	C3	3	22/09/995	
3	C3	3	22/09/992	

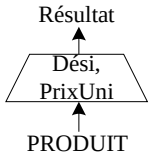
Nom	*Date*	*Quantité*
C1	22/09/991	
C3	22/09/995	
C3	22/09/992	

Π <Nom, Date, Quantité> (CL >< CO)

(Projection sur les attributs Nom, Date, Quantité)

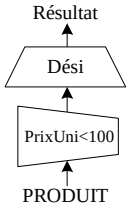
Exemples de requêtes

- Ex. 1 : Désignation et prix unitaire de tous les produits



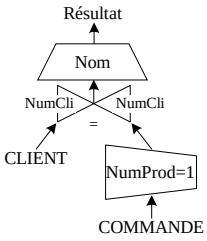
Exemples de requêtes

- Ex. 2 : Désignation des produits de prix inférieur à 100 €



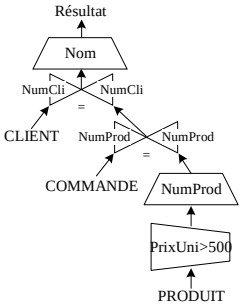
Exemples de requêtes

- Ex. 3 : Nom des clients qui ont commandé le produit n°1



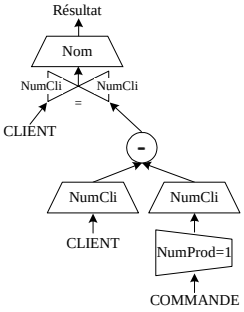
Exemples de requêtes

- Ex. 4 : Nom des clients qui ont commandé au moins un produit de prix supérieur à 500 €



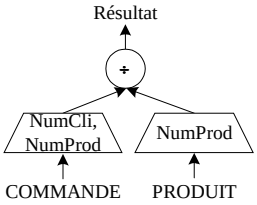
Exemples de requêtes

- Ex. 5 : Nom des clients qui n'ont pas commandé le produit n°1



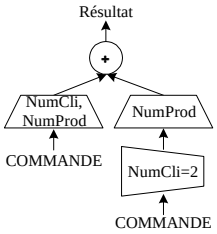
Exemples de requêtes

- Ex. 6 : Numéro des clients qui ont commandé tous les produits



Exemples de requêtes

- Ex. 7 : Numéro des clients qui ont commandé tous les produits commandés par le client n°2



Classification des SGBD relationnels

- Niveau 1 : Systèmes non relationnels.
  Supportent uniquement la structure tabulaire.
- Niveau 2 : Systèmes relationnellement minimaux.
  Permettent les opérations de sélection, projection et jointure.
- Niveau 3 : Systèmes relationnellement complets. Toutes les opérations de l'algèbre relationnelle.
- Niveau 4 : Systèmes relationnellement pleins.
  Permettent la définition des contraintes d'intégrité.

Plan du cours

- ✓ Introduction
- ✓ Algèbre relationnelle
- Langage SQL
- Langage XQuery

Présentation

- SQL : *Structured Query Language*, issu de SEQUEL (*Structured English as a QUery Language*)
- SQL permet la définition, la manipulation et le contrôle d'une base de données relationnelle. Il se base sur l'algèbre relationnelle.
- SQL est un standard ANSI depuis 1986.
- Nous adoptons dans ce chapitre la syntaxe du SQL d'Oracle (très proche de la norme).

Présentation

- SQL se subdivise en trois sous-langages :
  - LDD (*Langage de Définition de Données*) :
    création, modification et suppression des définitions des tables
  - LMD (*Langage de Manipulation de Données*) :
    ajout, suppression, modification et interrogation des données
  - LCD (*Langage de Contrôle de Données*) :
    gestion des protections d'accès

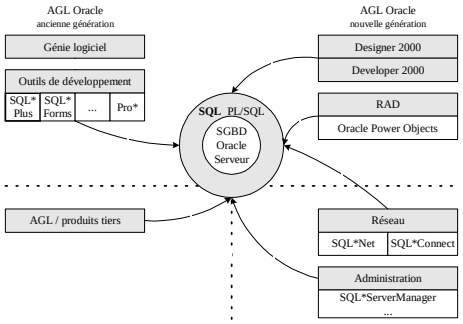
Qu'est-ce qu'Oracle ?

- Oracle : Système de Gestion de Bases de Données (SGBD) relationnel (SGBDR) édité par *Oracle Corporation* (http://www.oracle.com)
- Oracle est basé sur SQL.
- Première version d'Oracle : 1981
  Version actuelle : Oracle 11*g*

Fonctionnalités d'Oracle

- Aide à la décision (*data warehouses*, Oracle Express, Oracle Discoverer)
- Gestion de grands volumes de données (partitionnement des données)
- Mécanismes de sécurité
- Sauvegarde et récupération des données
- Gestion souple de l'espace disque
- Connectivité ouverte sur différentes plateformes

Architecture d'Oracle



Généralités

- Caractère de fin d'instruction : **;**
- Commentaires :
  - -- Ligne commentée
  - /* Bloc de texte commenté */
- Clauses optionnelles : Notées entre [ ] dans ce support de cours

Tables

- CREATE TABLE nom_table (Attribut1 TYPE,
  Attribut2 TYPE, …,
  contrainte_integrité1,
  contrainte_integrité2,
  …);
- Principaux type de données :
  - NUMBER(n) : Entier à n chiffres
  - NUMBER(n, m) : Réel à n chiffres au total (virgule comprise), m après la virgule
  - VARCHAR(n) : Chaîne de n caractères (entre ' ')
  - DATE : Date au format 'JJ-MM-AAAA'
  - BLOB : *Binary Large Object*

Contraintes d'intégrité

- Clé primaire :
  CONSTRAINT nom_contrainte PRIMARY KEY (attribut_clé [, attribut_clé2, …])
- Clé étrangère :
  CONSTRAINT nom_contrainte FOREIGN KEY (attribut_clé_ét) REFERENCES table(attribut)
- Contrainte de domaine :
  CONSTRAINT nom_contrainte CHECK (condition)

Exemple

```
CREATE TABLE Produit (      NumProd NUMBER(3),
                            Dési VARCHAR(30),
                            PrixUni NUMBER(8,2),
                            NumFour NUMBER(3),

CONSTRAINT produit_cle_pri PRIMARY KEY (NumProd),

CONSTRAINT produit_cle_etr FOREIGN KEY (NumFour)
        REFERENCES Fournisseur(NumFour),

CONSTRAINT prix_ok CHECK (PrixUni > 0)      );
```


Modifications structurelles

- Ajout d'attributs
  ALTER TABLE nom_table ADD (attribut TYPE, …);
  ex. ALTER TABLE Client ADD (tel NUMBER(8));
- Modifications d'attributs
  ALTER TABLE nom_table MODIFY (attribut TYPE, …);
  ex. ALTER TABLE Client MODIFY (tel NUMBER(10));
- Suppression d'attributs
  ALTER TABLE nom_table DROP COLUMN attribut, ...;
  ex. ALTER TABLE Client DROP COLUMN tel;

Modifications structurelles

- Ajout de contrainte

  ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte définition_contrainte;

  ex. ALTER TABLE Client
  ADD CONSTRAINT sal_ok CHECK (salaire > 0);

- Suppression de contrainte

  ALTER TABLE nom_table DROP CONSTRAINT nom_contrainte;

  ex. ALTER TABLE Client
  DROP CONSTRAINT sal_ok;

Copie et destruction de tables

- Destruction

  DROP TABLE nom_table;

  ex. DROP TABLE Client;

- Copie

  CREATE TABLE copie AS requête;

  ex. CREATE TABLE Client_Copie AS
  SELECT * FROM Client;

Index

- Création d'index (accélération des accès)

  CREATE [UNIQUE] INDEX nom_index ON nom_table (attribut [ASC|DESC], …);

  UNIQUE ⇒ pas de double
  ASC/DESC ⇒ ordre croissant ou décroissant

  ex. CREATE INDEX Idx_cli ON Client (Nom);

- Destruction

  DROP INDEX nom_index;

  ex. DROP INDEX Idx_cli;

Insertion et mise à jour

- Ajout d'un n-uplet

  INSERT INTO nom_table
  VALUES (val_att1, val_att2, …);

  ex. INSERT INTO Produit
  VALUES (400, 'Nouveau produit', 78.90, 30);

- Mise à jour d'un attribut

  UPDATE nom_table SET attribut = valeur
  [WHERE condition];

  ex. UPDATE Client SET Nom = 'Dudule'
  WHERE NumCli = 3;

Suppression

- Suppression de n-uplets

  DELETE FROM nom_table [WHERE condition];

  ex. DELETE FROM Produit;

  DELETE FROM Client
  WHERE Ville = 'Lyon';

Requêtes simples

- Forme générale d'une requête

  SELECT [ALL|DISTINCT] attribut(s) FROM table(s)
  [WHERE condition]
  [GROUP BY attribut(s) [HAVING condition]]
  [ORDER BY attribut(s) [ASC|DESC]];

- Tous les n-uplets d'une table
  ex. SELECT * FROM Client;

- Tri du résultat
  ex. *Par ordre alphabétique inverse de nom*
  SELECT * FROM Client
  ORDER BY Nom DESC;

Requêtes simples

- Calcul
  ex. *Calcul de prix TTC*
  SELECT PrixUni + PrixUni * 0.196 FROM Produit;
- Projection
  ex. *Noms et Prénoms des clients, uniquement*
  SELECT Nom, Prenom FROM Client;
- Suppression des doublons
  ex. SELECT DISTINCT Nom FROM Client;
- Restriction
  ex. *Clients qui habitent à Lyon*
  SELECT * FROM Client
  WHERE Ville = 'Lyon';

Requêtes simples

ex. *Commandes en quantité au moins égale à 3*
SELECT * FROM Commande
WHERE Quantite >= 3;

ex. *Produits dont le prix est compris entre 50 et 100 €*
SELECT * FROM Produit
WHERE PrixUni BETWEEN 50 AND 100;

ex. *Commandes en quantité indéterminée*
SELECT * FROM Commande
WHERE Quantite IS NULL;

Requêtes simples

ex. *Clients habitant une ville dont le nom se termine par* « sur-Saône »

SELECT * FROM Client
WHERE Ville LIKE '%sur-Saône';

'sur-Saône%' ➲ commence par « *sur-Saône* »
'%sur%' ➲ contient le mot « *sur* »

Prédicat ensembliste IN

ex. *Prénoms des clients dont le nom est Dupont, Durand ou Martin*

SELECT Prénom FROM Client
WHERE Nom IN ('Dupont', 'Durand', 'Martin');

NB : Possibilité d'utiliser la négation pour tous ces prédicats : NOT BETWEEN, NOT NULL, NOT LIKE, NOT IN.

Fonctions d'agrégat

- Elles opèrent sur un ensemble de valeurs.
- AVG(), VARIANCE(), STDDEV() : moyenne, variance et écart-type des valeurs
- SUM() : somme des valeurs
- MIN(), MAX() : valeur minimum, valeur maximum
- COUNT() : nombre de valeurs
- ex. *Moyenne des prix des produits*
  SELECT AVG(PrixUni) FROM Produit;

Fonctions d'agrégat

Fonction COUNT et opérateur DISTINCT

ex. *Nombre total de commandes*
SELECT COUNT(*) FROM Commande;
SELECT COUNT(NumCli) FROM Commande;

ex. *Nombre de clients ayant passé commande*
SELECT COUNT(DISTINCT NumCli)
FROM Commande;

Fonctions d'agrégat

Table COMMANDE (simplifiée)

NumCli	*Date*	*Quantite*
1	22/09/991	
3	22/09/995	
3	22/09/992	

COUNT(NumCli) ➲ Résultat = 3

COUNT(DISTINCT NumCli) ➲ Résultat = 2

Jointures

ex. *Liste des commandes avec le nom des clients*

```
SELECT Nom, Date, Quantite
FROM Client, Commande
WHERE      Client.NumCli =
           Commande.NumCli;
```


Jointures

ex. *Idem avec le numéro de client en plus*

```
SELECT C1.NumCli, Nom, Date, Quantite
FROM Client C1, Commande C2
WHERE C1.NumCli = C2.NumCli
ORDER BY Nom;
```

NB : Utilisation d'alias (C1 et C2) pour alléger l'écriture + tri par nom.

Jointures

Jointure exprimée avec le prédicat IN

ex. *Nom des clients qui ont commandé le 23/09*

```
SELECT Nom FROM Client
WHERE NumCli IN (
      SELECT NumCli FROM Commande
      WHERE Date = '23-09-1999'     );
```

NB : Il est possible d'imbriquer des requêtes.

Prédicats d'existence

- Prédicats EXISTS / NOT EXISTS

ex. *Clients qui ont passé au moins une commande [n'ont passé aucune commande]*

```
SELECT * FROM Client C1
WHERE [NOT] EXISTS (
      SELECT * FROM Commande C2
      WHERE C1.NumCli = C2.NumCli          );
```


Prédicats de dénombrement

- Prédicats ALL / ANY

ex. *Numéros des clients qui ont commandé au moins un produit en quantité supérieure à chacune [à au moins une] des quantités commandées par le client n°1.*

```
SELECT DISTINCT NumCli FROM Commande
WHERE Quantite > ALL [ANY] (
      SELECT Quantite FROM Commande
      WHERE NumCli = 1      );
```

Groupement

ex. *Quantité totale commandée par chaque client*

```
SELECT NumCli, SUM(Quantite)
FROM Commande
GROUP BY NumCli;
```

ex. *Nombre de produits différents commandés...*

```
SELECT NumCli, COUNT(DISTINCT NumProd)
FROM Commande
GROUP BY NumCli;
```


Groupement

ex. *Quantité moyenne commandée pour les produits faisant l'objet de plus de 3 commandes*

```
SELECT NumProd, AVG(Quantite)
FROM Commande
GROUP BY NumProd
HAVING COUNT(*) > 3;
```

Attention : La clause HAVING ne s'utilise qu'avec GROUP BY.

Groupement

- Différence entre HAVING et WHERE
  - HAVING : évaluation de condition sur un résultat de groupement (*a posteriori*)
  - WHERE : évaluation de condition *a priori* (avant GROUP BY)

Division

- Ex. Numéros des clients qui ont commandé tous les produits.
- Problème : Il n'existe pas d'opérateur de division en SQL !
- Deux stratégies :
  - Clients tels qu'il n'existe pas de produit tel qu'il n'existe pas de commande pour ce client et ce produit.
  - Clients qui ont commandé un nombre distinct de produits égal au nombre total de produits.

Division : solution « logique »

```
SELECT NumCli FROM Client Cl
WHERE NOT EXISTS (
      SELECT NumProd FROM Produit P
      WHERE NOT EXISTS (
                  SELECT * FROM Commande Co
                  WHERE Cl.NumCli = Co.NumCli
                  AND P.NumProd = Co.NumProd )
      );
```


Division : solution par comptage

-
```
SELECT NumCli FROM Client Cl
WHERE (     SELECT COUNT(DISTINCT NumProd)
            FROM Commande Co
            WHERE Co.NumCli = Cl.NumCli )
= ( SELECT COUNT(*) FROM Produit );
```

ou

-
```
SELECT NumCli FROM Commande
GROUP BY NumCli
HAVING COUNT(DISTINCT NumProd) =
      ( SELECT COUNT(*) FROM Produit );
```

Opérations ensemblistes

Opérateurs ensemblistes :

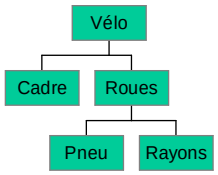
INTERSECT, MINUS, UNION

ex. *Numéro des produits qui soit ont un prix inférieur à 100 €, soit ont été commandés par le client n°2*

```
SELECT NumProd FROM Produit WHERE PrixUni < 100
UNION
SELECT NumProd FROM Commande WHERE NumCli = 2;
```


Requêtes hiérarchiques

Exemple de hiérarchie (nomenclature) :



Relation associée :

ELEMENT (No_Elt, Dési, Parent#)

0	Vélo	NULL			
1	Cadre	0			
2	Roue1	0	6	Rayon11	2
3	Roue2	0	7	Rayon12	2
4	Pneu1	2	8	Rayon13	2
5	Pneu2	3	9	Rayon21	3

Requêtes hiérarchiques

ex. *Structure hiérarchique des éléments à partir de la racine*

```
SELECT Dési FROM Element
CONNECT BY Parent = PRIOR No_Elt
START WITH Parent IS NULL;
```

Racine de la hiérarchie

Ordre de parcours de la hiérachie

Requêtes hiérarchiques

ex. *Idem avec indication du niveau dans la hiérarchie*

```
SELECT LEVEL, Dési FROM Element
CONNECT BY Parent = PRIOR No_Elt
START WITH Parent IS NULL;
```

Résultat :

1 Velo	3 Rayon12
2 Cadre	3 Rayon13
2 Roue1	2 Roue2
3 Pneu1	3 Pneu2
3 Rayon11	3 Rayon21

Requêtes hiérarchiques

ex. *Idem avec élagage d'une branche de la hiérarchie*

```
SELECT LEVEL, Dési FROM Element
CONNECT BY Parent = PRIOR No_Elt
           AND Dési <> 'Roue2'
START WITH Parent IS NULL;
```

Résultat :

1 Velo	3 Rayon12
2 Cadre	3 Rayon13
2 Roue1	2 Roue2
3 Pneu1	
3 Rayon11	

Requêtes hiérarchiques

ex. *Nombre d'éléments dans chaque niveau*

Il est possible d'utiliser le groupement.

```
SELECT LEVEL, COUNT(No_Elt)
FROM Element
CONNECT BY Parent = PRIOR No_Elt
START WITH Parent IS NULL
GROUP BY LEVEL;
```

Fonctions SQL

- ABS(n) : Valeur absolue de n
- CEIL(n) : Plus petit entier ≥ n
- FLOOR(n) : Plus grand entier ≤ n
- MOD(m, n) : Reste de m/n
- POWER(m, n) : m^n
- SIGN(n) : Signe de n
- SQRT(n) : Racine carrée de n
- ROUND(n, m) : Arrondi à 10^{-m}
- TRUNC(n, m) : Troncature à 10^{-m}
- CHR(n) : Caractère ASCII n°n
- INITCAP(ch) : 1ère lettre en maj.
- LOWER(ch) : c en minuscules
- UPPER(ch) : c en majuscules
- LTRIM(ch, n) : Troncature à gauche
- RTRIM(ch, n) : Troncature à droite
- REPLACE(ch, car) : Remplacement de caractère
- SUBSTR(ch, pos, lg) : Extraction de chaîne
- SOUNDEX(ch) : Cp. Phonétique
- LPAD(ch, lg, car) : Compléter à gauche
- RPAD(ch, lg, car) : Compléter à droite

Fonctions SQL

- ASCII(ch) : Valeur ASCII de ch
- INSTR(ch, ssch) : Recherche de ssch dans ch
- LENGTH(ch) : Longueur de ch
- ADD_MONTHS(dte, n) : Ajout de n mois à dte
- LAST_DAY(dte) : Dernier jour du mois
- MONTHS_BETWEEN(dt1, dt2) : Nombre de mois entre dt1 et dt2
- NEXT_DAY(dte) : Date du lendemain
- SYSDATE : Date/heure système
- TO_NUMBER(ch) : Conversion de ch en nombre
- TO_CHAR(x) : Conversion de x en chaîne
- TO_DATE(ch) : Conversion de ch en date
- NVL(x, val) : Remplace par val si x a la valeur NULL
- GREATEST(n1, n2…) : + grand
- LEAST (n1, n2…) : + petit
- UID : Identifiant numérique de l'utilisateur
- USER : Nom de l'utilisateur

Exemples

- SELECT UID, USER FROM DUAL;
- SELECT GREATEST(1, 2, 3) FROM DUAL;
- SELECT Nom, Prenom,
  FLOOR(MONTHS_BETWEEN(SYSDATE, DateNaiss) / 12) Age
  FROM Client;
- UPDATE Produit SET PrixUni = NVL(PrixUni, 15);

Vues

- Définition : Une vue est une table *virtuelle* calculée à partir d'autres tables grâce à une requête.
- Création d'une vue
  CREATE VIEW nom_vue AS requête;
  ex. CREATE VIEW Noms AS
  SELECT Nom, Prenom FROM Client;

Intérêt des vues

- Simplification de l'accès aux données en masquant les opérations de jointure

  ex. CREATE VIEW Prod_com AS
  SELECT P.NumProd, Dési, PrixUni, Date, Quantite
  FROM Produit P, Commande C
  WHERE P.NumProd = C.NumProd;

  SELECT NumProd, Dési FROM Prod_com
  WHERE Quantite > 10;

Intérêt des vues

- Sauvegarde indirecte de requêtes complexes
- Présentation de mêmes données sous différentes formes adaptées aux différents usagers particuliers
- Support de l'indépendance logique
  ex. Si la table Produit est remaniée, la vue Prod_com doit être refaite, mais les requêtes qui utilisent cette vue n'ont pas à être remaniées.
- Renforcement de la sécurité des données par masquage des lignes et des colonnes sensibles aux usagers non habilités

Restriction des vues (mise à jour)

- Pour que la mise à jour de données à travers une vue soit possible :
  - Le mot clé DISTINCT doit être absent de la requête.
  - La clause FROM doit faire référence à une seule table.
  - La clause SELECT doit faire référence directement aux attributs de la table concernée (pas d'attribut dérivé).
  - Les clauses GROUP BY et HAVING sont interdites.

Catalogue du système

- Définition : Ensemble de vues maintenues automatiquement par le système et contenant sous forme relationnelle la définition de tous les objets créés par le système et les usagers.
- Ces vues sont accessibles avec SQL (en mode consultation uniquement).

Vues systèmes

- ALL_TABLES (OWNER, TABLE_NAME, …)
- ALL_VIEWS (OWNER, VIEW_NAME, …)
- ALL_CONSTRAINTS (OWNER, TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION, ...)
- ALL_CONS_COLUMNS (OWNER, TABLE_NAME, CONSTRAINT_NAME, COLUMN_NAME, ...)
- USER_CATALOG (TABLE_NAME, TABLE_TYPE)
- USER_TAB_COLUMNS (TABLE_NAME, COLUMN_NAME, …)
- USER_IND_COLUMNS (INDEX_NAME, TABLE_NAME, COLUMN_NAME, …)
- USER_CONSTRAINTS (TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION, ...)

Exemples

- Tables qui contiennent un attribut *Intitulé*

```
SELECT TABLE_NAME FROM USER_TAB_COLUMNS
WHERE COLUMN_NAME = 'INTITULE';
```

- Attributs de la table *Client*

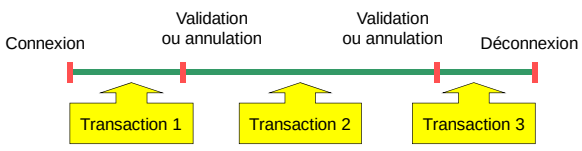
```
SELECT COLUMN_NAME FROM USER_TAB_COLUMNS
WHERE TABLE_NAME = 'CLIENT';
```

- Contraintes des tables de l'utilisateur courant

```
SELECT TABLE_NAME, CONSTRAINT_NAME
FROM ALL_CONSTRAINTS
WHERE OWNER = USER;
```


Gestion des transactions

- Transaction : ensemble de mises à jour des données (≠ modifications structurelles)
- Début de transaction : début de la session de travail ou fin de la transaction précédente



Contrôle des transactions

- Validation (et fin) d'une transaction :

```
COMMIT;
```

- Annulation (et fin) d'une transaction :

```
ROLLBACK;
```

- Fin de session de travail (avec la commande EXIT ou QUIT) ➲ validation automatique

Sécurité : utilisateurs

- Création
  - Ex. CREATE USER moi_meme
    IDENTIFIED BY mon_mot_de_passe;
- Suppression
  - Ex. DROP USER moi_meme CASCADE;
- Modification
  - Ex. ALTER USER moi_meme IDENTIFIED BY a;

Sécurité : privilèges globaux

- Droit d'effectuer une action sur les objets de l'utilisateur seulement
  - Ex. CREATE TABLE
    ALTER INDEX
    DROP VIEW
- Droit d'effectuer une action dans tous les schémas de la base de données
  - Ex. CREATE ANY TABLE
    ALTER ANY INDEX
    DROP ANY VIEW

Sécurité : privilèges sur les objets

Privilège	Signification	Tables	Vues
ALTER	Destruction	X	
DELETE	Suppression	X	X
INDEX	Construction	X	
INSERT	Insertion	X	X
REFERENCES	Clé étrangère	X	
SELECT	Lecture	X	X
UPDATE	Mise à jour	X	X
ALL	Tous	X	X

Sécurité : rôles

- Rôles prédéfinis
  - CONNECT : droit de création de tables, vues, synonymes, etc.
  - RESOURCE : droit de création de procédures stockées, déclencheurs, etc.
  - DBA : administrateur de la BD
- Création de nouveaux rôles
  - Ex. CREATE ROLE role1;

Sécurité : attribution de privilèges

- Transmission de privilèges
  GRANT privilège ON table|vue
  TO user|PUBLIC [WITH GRANT OPTION];
- Privilèges sur des objets
  - Ex. GRANT SELECT ON ma_table TO toto;
  - Ex. GRANT SELECT ON ma_table TO PUBLIC;
  - Ex. GRANT SELECT ON ma_table TO role1;
- Privilèges globaux et rôles
  - Ex. GRANT CREATE ANY TABLE TO toto;
  - Ex. GRANT CONNECT, RESOURCE TO toto;
  - Ex. GRANT role1 TO toto;

Sécurité : révocation de privilèges

- Suppression de privilèges
  REVOKE privilège ON table|vue FROM user|PUBLIC;
- Privilèges sur des objets
  - Ex. REVOKE SELECT ON ma_table FROM toto;
  - Ex. REVOKE SELECT ON ma_table FROM PUBLIC;
  - Ex. REVOKE SELECT ON ma_table FROM role1;
- Privilèges globaux et rôles
  - Ex. REVOKE CREATE ANY TABLE FROM toto;
  - Ex. REVOKE CONNECT, RESOURCE FROM toto;
  - Ex. REVOKE role1 FROM toto;

Tutoriel SQL

Pour approfondir SQL en ligne…

http://eric.univ-lyon2.fr/~jdarmont/tutoriel-sql/

Plan du cours

- ✓ Introduction
- ✓ Algèbre relationnelle
- ✓ Langage SQL
- ■ Langage XQuery

Généralités

- Langage de requêtes sur données XML
- Similitudes avec SQL
- Développé par le W3C
- Bâti sur des expressions de chemins XPath
- XQuery 1.0 inclut XPath 2.0 et utilise les même modèle de données, fonctions et opérateurs
- En cours de standardisation
- Soutenu par les grands éditeurs de SGBD (Oracle, Microsoft, IBM…)

Document XML exemple

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<catalogue>
        <dvd zone="1">
                <titre>Blade runner</titre>
                <realisateur>Ridley Scott</realisateur>
                <annee>1982</annee>
                <duree>117</duree>
                <langue>anglais</langue>
                <prix>14.79</prix>
        </dvd>
        <dvd zone="2">
                <titre>La grande vadrouille</titre>
                <realisateur>Gérard Oury</realisateur>
                <annee>1966</annee>
                <duree>122</duree>
                <langue>français</langue>
                <prix>19.82</prix>
        </dvd>                                          <!-- (…) -->
```


Document XML exemple

```
        <dvd zone="2">
                <titre>Le fabuleux destin d'Amélie Poulain</titre>
                <realisateur>Jean-Pierre Jeunet</realisateur>
                <annee>2001</annee>
                <duree>120</duree>
                <langue>français</langue>
                <prix>14.99</prix>
        </dvd>
        <dvd zone="2">
                <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>19.82</prix>
        </dvd>
</catalogue>
```


Expressions de chemins simples

- Document XML entier
  doc("dvd.xml")/catalogue
- Un élément particulier
  doc("dvd.xml")/catalogue/dvd
  doc("dvd.xml")/catalogue/dvd/titre
- Un attribut particulier
  doc("dvd.xml")/catalogue/dvd/data(@zone)
- Un élément quel que soit son niveau hiérarchique
  doc("dvd.xml")/catalogue//titre
  //titre
- Sous-élements d'un élément
  doc("dvd.xml")/catalogue/dvd/*

Prédicats sur les chemins

- i^{ème}, dernier, i premiers/derniers éléments
  doc("dvd.xml")/catalogue/dvd[1]
  doc("dvd.xml")/catalogue/dvd[last()]
  doc("dvd.xml")/catalogue/dvd[position() < 3]
- Éléments qui possèdent l'élément / l'attribut
  doc("dvd.xml")/catalogue/dvd[duree]
  doc("dvd.xml")/catalogue/dvd[@zone]
- Conditions sur un élément / attribut
  doc("dvd.xml")/catalogue/dvd[prix < 15]
  doc("dvd.xml")/catalogue/dvd[@zone = "1"]/titre
- Combinaison de chemins
  doc("dvd.xml")//titre | doc("dvd.xml")//prix

Requêtes FLWOR

- FLWOR (« flower ») : For, Let, Where, Order by, Return
- Clause For : lie une variable à chaque élément retourné par une expression (itération)

Exemple :

```
for $x in (1 to 3)
return <res>{$x}</res>
```

Résultat :

```
<res>1</res>
<res>2</res>
<res>3</res>
```


Clause For

Exemple :

```
for     $x in (1, 2),
        $y in (10, 20)
return <res>x = {$x} et y = {$y}</res>
```

Résultat :

```
<res>x = 1 et y = 10</res>
<res>x = 1 et y = 20</res>
<res>x = 2 et y = 10</res>
<res>x = 2 et y = 20</res>
```


Clause For

Exemple :

```
for $x in doc("dvd.xml")/catalogue/dvd
return $x/titre
```

Résultat :

```
<titre>Blade runner</titre>
<titre>La grande vadrouille</titre>
<titre>Le fabuleux destin d'Amélie Poulain</titre>
<titre>The big Lebowski</titre>
```


Clause For

Exemple :

```
for $x at $i in doc("dvd.xml")/catalogue/dvd/titre
return <dvd id="{$i}">{data($x)}</dvd>
```

Résultat :

```
<dvd id="1">Blade runner</dvd>
<dvd id="2">La grande vadrouille</dvd>
<dvd id="3">Le fabuleux destin d'Amélie Poulain</dvd>
<dvd id="4">The big Lebowski</dvd>
```


Clause Let

- Clause Let : Assignation de valeur(s) à une variable (pas d'itération)

Exemple :

```
let $x := (1 to 5)
return <res>{$x}</res>
```

Résultat :

```
<res>1 2 3 4 5</res>
```

Clause Where

- Clause Where : Spécification de critère(s) sur le résultat

Exemple :

```
for $x in doc("dvd.xml")/catalogue/dvd
where $x/prix > 15
return $x/titre
```

Exemple :

```
for $x in doc("dvd.xml")/catalogue/dvd
where $x/@zone = "2" and $x/prix < 15
return $x/titre
```


Clauses Order by et Return

- Clause Order by : Tri du résultat

Exemple :

```
for $x in doc("dvd.xml")/catalogue/dvd
order by $x/titre
return $x/titre
```

Exemple :

```
for $x in doc("dvd.xml")/catalogue/dvd
order by $x/@zone, $x/titre descending
return $x/titre
```

- Clause Return : Spécification du résultat

Présentation HTML du résultat

Exemple :

```
<ul> {
for $x in doc("dvd.xml")/catalogue/dvd/titre
return <li>{$x}</li>
} </ul>
```

Résultat :

```
<ul>
        <li><titre>Blade runner</titre></li>
        <li><titre>La grande vadrouille</titre></li>
        <!-- (…) -->
</ul>
```


Présentation HTML du résultat

Exemple :

```
<ul> {
for $x in doc("dvd.xml")/catalogue/dvd/titre
return <li>{data($x)}</li>
} </ul>
```

Résultat :

```
<ul>
        <li>Blade runner</li>
        <li>La grande vadrouille</li>
        <!-- (…) -->
</ul>
```


Présentation HTML du résultat

Exemple :

```
<ul> {
for $x in doc("dvd.xml")/catalogue/dvd
return <li class="zone{data($x/@zone)}">
        {data($x/titre)}</li>
} </ul>
```

Résultat :

```
<ul>
        <li class="zone1">Blade runner</li>
        <li class="zone2">La grande vadrouille</li>
        <!-- (…) -->
</ul>
```


Expressions conditionnelles

Exemple :

```
for $x in doc("dvd.xml")/catalogue/dvd
return   if ($x/@zone="1")
        then <zoneUS>{data($x/titre)}</zoneUS>
        else <zoneEU>{data($x/titre)}</zoneEU>
```

Résultat :

```
<zoneUS>Blade runner</zoneUS>
<zoneEU>La grande vadrouille</zoneEU>
<zoneEU>Le fabuleux destin d'Amélie
        Poulain</zoneEU>
<zoneEU>The big Lebowski</zoneEU>
```

Fonctions XQuery

- Fonctions d'accès : data()…
- Fonctions numériques : abs(), floor(), ceiling(), round()…
- Fonctions de chaînes : string-length(), upper-case(), lower-case(), substring(), contains()…
- Fonctions temporelles : day-from-date()…
- Fonctions de séquences : exists(), distinct-values(), reverse()…
- Fonctions d'agrégat : count(), avg(), max(), min(), sum()
- Fonctions de contexte : last(), position()…
- Fonctions booléennes : not()…

Fonctions XQuery

Exemple d'appel :

```
for $x in doc("dvd.xml")/catalogue/dvd/titre
let $maj := upper-case($x)
return <film>{$maj}</film>
```

Résultat :

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


Regroupement

Exemple :

```
for $z in distinct-values(/catalogue/dvd/@zone)
let $p := avg(/catalogue/dvd[@zone = $z]/prix)
order by $z
return      <zone valeur="{$z}">
                <prixmoy>{$p}</prixmoy>
            </zone>
```

Résultat :

```
<zone valeur="1">
        <prixmoy>14.79</prixmoy>
</zone>
<zone valeur="2">
        <prixmoy>18.21</prixmoy>
</zone>
```


Regroupement multiple

```
for    $z in distinct-values(/catalogue/dvd/@zone),
       $a in distinct-values(/catalogue/dvd/annee)
let $p := /catalogue/dvd[@zone = $z and annee = $a]/prix
order by $z, $a
return      <zone valeur="{$z}" annee="{$a}">
                <prixmoy>{avg($p)}</prixmoy>
            </zone>
```

Problèmes :

- Lisibilité du résultat (éléments vides)
- Efficacité (multiples parcours du document)

Jointure

```
<?xml version="1.0" encoding="iso-8859-1" ?>          <!-- clients.xml -->
<clients>
        <client id="1">
                <nom>Lallich</nom>
                <prenom>Stéphane</prenom>
                <adresse>Bureau 04</adresse>
        </client>
        <client id="2">
                <nom>Bentayeb</nom>
                <prenom>Fadila</prenom>
                <adresse>Bureau 09B</adresse>
        </client>
        <client id="3">
                <nom>Darmont</nom>
                <prenom>Jérôme</prenom>
                <adresse>Bureau 09B</adresse>
        </client>
</clients>
```


Jointure

```
<?xml version="1.0" encoding="iso-8859-1" ?>          <!-- produits.xml -->
<produits>
        <produit id="10">
                <libelle>Ordinateur</libelle>
        </produit>
        <produit id="20">
                <libelle>Ecran 17'</libelle>
        </produit>
        <produit id="30">
                <libelle>Imprimante</libelle>
        </produit>
</produits>
```

Jointure

```
<?xml version="1.0" encoding="iso-8859-1" ?>          <!-- commandes.xml -->
<commandes>
        <commande idcli="1" idprod="10">
                <quantite>3</quantite>
        </commande>
        <commande idcli="1" idprod="20">
                <quantite>15</quantite>
        </commande>
        <commande idcli="2" idprod="10">
                <quantite>7</quantite>
        </commande>
        <commande idcli="2" idprod="30">
                <quantite>10</quantite>
        </commande>
        <commande idcli="3" idprod="30">
                <quantite>5</quantite>
        </commande>
</commandes>
```


Jointure

Exemple :

```
for     $cli in doc("clients.xml")//client,
        $com in doc("commandes.xml")//commande
where $cli/@id = $com/@idcli
return <res>{data($cli/nom)}, {data($cli/prenom)} :
        {data($com/quantite)}</res>
```

Résultat :

```
<res>Lallich, Stéphane : 3</res>
<res>Lallich, Stéphane : 15</res>
<res>Bentayeb, Fadila : 7</res>
<res>Bentayeb, Fadila : 10</res>
<res>Darmont, Jérôme : 5</res>
```


Jointure

Exemple :

```
for     $cli in doc("clients.xml")//client,
        $com in doc("commandes.xml")//commande,
        $prod in doc("produits.xml")//produit
where $cli/@id = $com/@idcli
and $com/@idprod = $prod/@id
return <res>{data($cli/nom)}, {data($cli/prenom)} :
        {data($com/quantite)} x {data($prod/libelle)}</res>
```

Résultat :

```
<res>Lallich, Stéphane : 3 x Ordinateur</res>
<res>Lallich, Stéphane : 15 x Ecran 17'</res>
<res>Bentayeb, Fadila : 7 x Ordinateur</res>
<res>Bentayeb, Fadila : 10 x Imprimante</res>
<res>Darmont, Jérôme : 5 x Imprimante</res>
```


Jointure

Variantes :

```
for     $cli in doc("clients.xml")//client,
        $com in doc("commandes.xml")//commande[@idcli=$cli/@id]
return <res>{data($cli/nom)}, {data($cli/prenom)} :
                {data($com/quantite)}</res>
```

```
for     $cli in //client,
        $prod in //produit,
        $com in //commande[@idcli=$cli/@id and @idprod=$prod/@id]
return <res>{data($cli/nom)}, {data($cli/prenom)} :
        {data($com/quantite)} x {data($prod/libelle)}</res>
```


Plan du cours

- ✓ Introduction
- ✓ Algèbre relationnelle
- ✓ Langage SQL
- ✓ Langage XQuery