Persistance des données

Chapitre 1: SQL - DML

He2b-ESI (CUV-NVS-SRE-ARO)

Année académique 2021 / 2022

Plan du chapitre

- SQL - Généralités
- SELECT - Sous requêtes
- SELECT - Jointures externes
- INSERT - UPDATE - DELETE
- SELECT - compléments

SQL: Généralités

SQL = Structured Query Language

- Non procédural
- Normalisé ISO
  - 1986 - 1989
  - **1992(SQL2)**
  - **1999(SQL3)**
  - 2003 - 2008 - 2011 - 2016 - 2019
- Edgar Frank Codd à l'orgine du modèle relationnel
- Donald Chamberlain et Raymond Boyce pour les premiers développements de SQL chez IBM
- Dans ce cours, nous abordons l'essentiel de la norme SQL2. Des compléments concernant SQL3 sont proposés en fin de chapitre.

SQL: Généralités - Rappels

Vous avez vu en DON2, le SELECT du DML.

```
SELECT [DISTINCT]{ * ou {liste d'expressions
                         et/ou aggregate fonction}}
   FROM liste de tables et/ou de vues et/ou de joins
   [WHERE condition]
   [GROUP BY liste d'attributs]
   [HAVING condition]
   [ORDER BY liste d attributs [ASC/DESC]
             et/ou n°colonne [ASC/DESC] ]
```

SQL: Généralités - Dialectes

Au cours : la norme SQL2

Aux laboratoires : utilisation d'outils avec des différences à la norme qui varient d'un SGBD à l'autre et d'une version à l'autre.

Actuellement à l'ESI :

- Oracle 11g
- SQLite 3.0
- PostgreSQL 9.6

SQL: Généralités - Exemples de dialectes

Quelque exemples de différences pour les SGBD installés à l'ESI requêtes de type SELECT

Alias

```
SELECT empNom AS nom
 FROM Employe AS emp
```

Avec ou sans **AS** pour un alias de table : permis par tous les SGBD installés à l'ESI sauf pour Oracle 11g

Du coup, on prend l'habitude de se passer du **AS**:

```
SELECT empNom AS nom
 FROM Employe emp
```

SQL: Généralités - Exemples de dialectes

Acceptation de syntaxe incorrecte

SQLite accepte des choses n'ayant pas de sens :

```
SELECT empDpt, empNom
 FROM Employe
 GROUP BY empDpt

SELECT COUNT(dptno), dptLib
 FROM Departement
```

SQL: Généralités - Exemples de dialectes

Acceptation de syntaxe incorrecte

Les SGBD installés à l'ESI s'écartent de la norme en acceptant

```
SELECT *
 FROM Departement
 WHERE dptAdm = NULL
```

Mais présentent bien un résultat vide

```
SELECT *
 FROM Departement
 WHERE dptAdm IN ('D21', NULL)
```

Mais ne sélectionnera que les dptAdm égal à 'D21'

SQL: Généralités - Exemples de dialectes

Comportements non normalisés

Position des NULL lors d'un ORDER BY

```
SELECT *
 FROM Departement
 ORDER BY dptAdm
```

Chacun positionnera les NULL à sa façon

SQL: Généralités - Exemples de dialectes

EXCEPT

La norme :

```
SELECT dptNo FROM Departement
EXCEPT
SELECT empDpt FROM Employe
```

Avec le SGBD Oracle installé à l'ESI :

```
SELECT dptNo FROM Departement
MINUS
SELECT empDpt FROM Employe
```

SQL: Généralités - Éléments de syntaxe

- Les identificateurs
- Les délimiteurs
- Les commentaires

SQL: Généralités - Identificateurs

Noms d'objet tel que table, colonne, ...

- Normal (sans "):
  - insensible à la casse
  - maximum 128 caractères (lettres, chiffres, $ et underscore)
  - ne peut pas commencer par un chiffre
  - doit être extérieur à la liste des mots réservés
- Délimité (entre ") :
  - sensible à la casse
  - maximum 128 caractères (lettres, chiffres, caractères spéciaux, blanc)

SQL: Généralités - Identificateurs

Accepté

- MaColonne_2
- “Select”

Interdit

- 21MATable
- from

SQL: Généralités - Délimiteurs

- Par défaut, les instructions SQL sont généralement délimitées par un point-virgule
- Une suite d'espaces a la même valeur qu'un espace
- Le délimiteur de chaîne de caractères est l'apostrophe (')

SQL: Généralités - Commentaires

- Les caractères d'une ligne apparaissant après deux tirets (- -) sont ignorés
- Les caractères repris entre /* et */ sont ignorés

SQL: Généralités - Noms des objets

- Dans une base de donnée, les objets (tables, vues, . . .) sont logiquement stockés dans des schémas, chaque utilisateur a généralement un schéma par défaut qui lui est associé.
- Chez Oracle, lorsqu'un utilisateur **U** est créé, automatiquement un schéma **U** est créé et lui est associé.
- Pour accéder à la table **table1** du schéma **U** :
  - pour U : table1
  - pour tous les autres utilisateurs : U.table1

SQL: Généralités - Les droits

Remarque: pour pouvoir exécuter une opération du DML (SELECT, INSERT, UPDATE, DELETE) sur une table appartenant à l'utilisateur **U**, celui-ci doit vous avoir donné les droits sur cette table. L'utilisateur **U** peut choisir quel droit il donne à quel utilisateur.

Exemple

Pour la table **CUV**.CLIENT présente sur le SGBD Oracle de l'école, les étudiants ne possèdent que le droit en lecture (SELECT).

Le gestion des droits sera étudiée en détails dans le chapitre 3 du cours.

SQL: Généralités - Synonymes

```
CREATE [PUBLIC] SYNONYM [nomSchema1.]nomSynonym
        FOR [nomSchema2.]nomObjet
```

```
DROP [PUBLIC] SYNONYM [nomSchema1.]nomSynonym
```

Les objets auxquels on associe des synonymes peuvent être des

- tables
- vues
- synonymes
- ...

Plan du chapitre

- SQL - Généralités
- **SELECT - Sous-requêtes**
- SELECT - Jointures externes
- INSERT - UPDATE - DELETE
- SELECT - compléments

SELECT - Sous-requêtes

Voir les slides correspondants du cours de DON2

Des exercices de rappels sont prévus durant le labo de revisions.

SELECT - Sous-requêtes - Exemple de dialectes

ALL, ANY

Refusé par SQLite

Select imbriqué

```
SELECT *
 FROM (SELECT dptMgr
         FROM Departement)
```

est accepté par tous mais MySQL et PostgreSQL exigent un alias pour la requête

```
SELECT *
 FROM (SELECT dptMgr
         FROM Departement) mgr
```

SELECT - Sous-requêtes - Exemples de dialectes

Adjonction de fonctionnalités

Oracle, MySQL, PostgreSQL permettent de faire :

```
SELECT MAX(COUNT(*))
 FROM Employe
 GROUP BY empDpt
```

Ceci n'est pas dans la norme et est refusé par les autres SGBD installés à l'ESI

En SQL standard on aura :

```
SELECT MAX(nb)
 FROM ( SELECT COUNT(*) nb
          FROM Employe
          GROUP BY empDpt)
```

Plan du chapitre

- SQL - Généralités
- SELECT - Sous-requêtes
- **SELECT - Jointures externes**
- INSERT - UPDATE - DELETE
- SELECT - compléments

SELECT - Jointures externes

Rappel : le produit cartésien

Que donnera la requête suivante?

```
SELECT dptlib, empnom
  FROM employe, departement
```

Rappel : les jointures internes

La jointure interne = INNER JOIN

```
SELECT dptlib, dptno
   FROM departement
   ORDER BY dptlib
```

```
SELECT dptlib, dptadm
   FROM departement
   ORDER BY dptlib
```

R1	R2
DEVELOPPEMENT A00	DEVELOPPEMENT D21
DIRECTION D21	DIRECTION **NULL**
FORMATION E21	FORMATION E21
MAINTENANCE C01	MAINTENANCE A00
MARKETING E01	MARKETING E11
PRODUCTION B01	PRODUCTION A00
SUPPORT D11	SUPPORT E11
VENTES E11	VENTES D21

==> il y a **8** départements

Rappel : les jointures internes

*Pour **tous** les départements, donner les départements administrateurs*

```
SELECT  dp.dptlib, sup.dptlib supérieur
  FROM departement dp JOIN departement sup
       ON dp.dptadm = sup.dptno
  ORDER BY supérieur
```

dp.dptlib	supérieur
PRODUCTION	DEVELOPPEMENT
MAINTENANCE	DEVELOPPEMENT
DEVELOPPEMENT	DIRECTION
VENTES	DIRECTION
SUPPORT	VENTES
MARKETING	VENTES
FORMATION	VENTES

==> il y a que **7** départements

SELECT - Jointures externes

La joiture externe = OUTER JOIN

- Disponible à partir de SQL2 intermédiaire
- La jointure externe est une extension de la jointure, qui permet d'obtenir en plus des lignes répondant à la condition, les lignes de l'une des tables qui n'y répondent pas.
- **LEFT**: prend tous les tuples de la tables à gauche du JOIN
- **RIGHT**: prend tous les tuples de la table à droite du JOIN

SELECT - Jointures externes

*Pour **tous** les départements, donner les départements administrateurs*

- La requête précédente n'est pas complète. Pourquoi?: Elle ne donne pas les départements non administrés.
- Que faire pour les voir? : Demander de lister tous les libellés de la table dp

```
SELECT dp.dptlib, sup.dptlib supérieur
  FROM departement dp LEFT JOIN departement sup
       ON dp.dptadm = sup.dptno
  ORDER BY supérieur
```

dp.dptlib	supérieur
DIRECTION	**NULL**
PRODUCTION	DEVELOPPEMENT
MAINTENANCE	DEVELOPPEMENT
DEVELOPPEMENT	DIRECTION
VENTES	DIRECTION
SUPPORT	VENTES
MARKETING	VENTES
FORMATION	VENTES

SELECT - Jointures externes

Avec l'OUTER JOIN à droite, sur la table sup.

```
SELECT  d.dptlib administré, sup.dptlib supérieur
  FROM departement d RIGHT JOIN departement sup
       ON d.dptadm = sup.dptno
  ORDER BY supérieur
```

administré	supérieur
PRODUCTION	DEVELOPPEMENT
MAINTENANCE	DEVELOPPEMENT
DEVELOPPEMENT	DIRECTION
VENTES	DIRECTION
NULL	FORMATION
NULL	MAINTENANCE
NULL	MARKETING
NULL	PRODUCTION
NULL	SUPPORT
FORMATION	VENTES
MARKETING	VENTES
SUPPORT	VENTES

Sémantique? : Liste de tous les départements précédés de leurs départements administrés s'ils existent.

SELECT - Jointures externes

- Le FULL donne la combinaison de LEFT et RIGHT (pas supporté par tous les SGBD)
- Le mot clé optionnel OUTER sans LEFT, RIGHT, FULL n'a pas de sens et est refusé.
- Ancienne codification SQL1 :
  - la condition de jointure est donnée dans la clause WHERE
  - dans la condition de jointure mettre (+) derrière la colonne dont on veut toutes les valeurs.
  - en SQL1 le FULL est impossible

```
SELECT d.dptlib administré, sup.dptlib supérieur
    FROM departement d, departement sup
    WHERE  d.dptadm = sup.dptno(+)
    ORDER BY supérieur
```

SELECT - Jointures externes - Exemple de dialectes

```
SELECT dptNo
 FROM Employe RIGHT JOIN Departement ON empDpt=dptNo
 WHERE empDpt IS NULL AND empSexe='F'
```

- LEFT JOIN accepté par tous
- SQLite n'accepte pas les RIGHT JOIN ni les FULL OUTER JOIN.

SELECT - Dialectes

Que faire si vous êtes un jour confronté à un SGBD non utilisé à l'ESI (Microsoft SQL Server, IBM Db2, ...) ?

A votre avis ?

Plan du chapitre

- SQL - Généralités
- SELECT - Sous-requêtes
- SELECT - Jointures externes
- **INSERT - UPDATE - DELETE**
- SELECT - compléments

DML - UPDATE

Mise à jour d'un ensemble de tuples

```
UPDATE  nomRelation
    SET attribut1 = expression1
            [, attribut2 = expression2] ...
    [WHERE condition]
```

expression : construite sur des attributs de la relation, des constantes, des fonctions ou SELECT imbriqués

condition : idem que condition du WHERE d'un select

DML - UPDATE

```sql
UPDATE Employe
     SET empSal = empSal * 1.1
     WHERE empNo = '056' ;

UPDATE Employe
     SET empDpt = 'D21', empSal = empSal * 2
     WHERE empNo NOT IN (SELECT dptMgr
                         FROM Departement) ;
```

DML - UPDATE

```sql
UPDATE Employe
 SET empSal = (SELECT MIN(empSal)
                 FROM Employe
                 JOIN Departement ON empNo = dptMgr)
 WHERE empNo IN (SELECT dptMgr FROM Departement) ;
```

DML - UPDATE

Remarque: En fonction de la condition fournie dans la clause WHERE, une requête UPDATE peut donc affecter zéro, une ou plusieurs lignes de la table cible.

Cependant: Les contraintes dont la table est le siège (clé primaire, clé étrangère, ...) **doivent être respectées**, sinon le SGBD refusera d'exécuter la requête !

DML - UPDATE

De plus:

Intégrité en lecture

Chaque instruction du DML agit sur un seul état de la BD.

Tout se fait ou rien ne se fait :

- Si pour l'une ou l'autre raison (en fait, généralement, non respect d'une contrainte) l'une ou l'autre ligne impliquée dans l'opération ne peut subir la manipulation, aucun tuple ne sera affecté.
- Bien évidemment, en cas de non complétion de la requête, un code d'erreur sera émis.

DML - DELETE

Suppression d'un ensemble de tuples

```
DELETE
    FROM  nomRelation
    [WHERE condition ]
```

```
DELETE
    FROM Employe
    WHERE empDpt = 'C03'
```

```
DELETE
    FROM Departement
    WHERE dptNo NOT IN (SELECT empDpt
                        FROM Employe)
```

DML - INSERT

Deux formats d'insertions

1. Ajout d'un tuple

```
INSERT INTO  nomRelation [ (liste attributs) ]
    VALUES ( liste expressions )
```

2. Ajout d'un ensemble de tuples

```
INSERT INTO  nomRelation [ (liste attributs) ]
    SELECT ...
```

DML - INSERT

1. Ajout d'un tuple

```
INSERT INTO Employe
    VALUES ('015','DUCHEMIN','M',78000,'A00','MARCEL')

INSERT INTO Employe (empNo, empNom, empDpt)
    VALUES ('015','DUCHEMIN','A00')
```

2. Ajout d'un ensemble de tuples

```
INSERT INTO Manager
        SELECT empNom, dptLib
        FROM Employe JOIN Departement ON empNo = dptMgr
```

Plan du chapitre

- SQL - Généralités
- SELECT - Sous-requêtes
- SELECT - Jointures externes
- INSERT - UPDATE - DELETE
- **SELECT - Compléments**

SELECT - Compléments : Gestion du NULL

NULL+1 = ?

NULL AND FALSE = ?

Prennons la table **T1**

a
1
2
NULL

```
SELECT a
 FROM T1
 WHERE a NOT IN (1, NULL) résultat : ?
```

SELECT - Compléments : Gestion du NULL

AND	T	F	**N**	**OR**	T	F	**N**	**NOT**	T
T	T	F	**N**	T	T	T	**T**	T	F
F	F	F	**F**	F	T	F	**N**	F	T
N	**N**	**F**	**N**	**N**	**T**	**N**	**N**	**N**	**N**

- Une expression dont l'évaluation renvoie une valeur numérique, caractères ou temporelle est évaluée à NULL si l'un de ses arguments est NULL
- Une fonction d'agrégat est calculée en ignorant les NULLs (sauf COUNT(*))
- Lors d'un groupement, les NULLs forment un groupe

SELECT - Compléments : Gestion du NULL

T1

a
1
2
NULL

```
SELECT a
 FROM T1
 WHERE a IN (1, NULL)          > résultat : 1

SELECT a
 FROM T1
 WHERE a = 1 OR a = NULL       > résultat : 1
```

SELECT - Compléments : Gestion du NULL

T1

a
1
2
NULL

```
SELECT a
 FROM T1
 WHERE a NOT IN (1, NULL)    > résultat: no data found

SELECT a
 FROM T1
 WHERE a != 1 AND a != NULL > résultat: no data found
```

Donc : **Évitez les NULLs dans le schéma**

SELECT - Compléments

La suite de ces compléments utilisent les normes **SQL3** et suivantes

SELECT - Compléments : CASE

Deux syntaxes:

```
CASE expression
    WHEN valeur1 THEN expression1
    [WHEN valeur2 THEN expression2]
    ...
    [ELSE expression_défaut]
END
```

Ou

```
CASE
    WHEN condition1 THEN expression1
    [WHEN condition2 THEN expression2]
    ...
    [ELSE expression_défaut]
END
```

SELECT - Compléments : CASE

```
SELECT empno, empnom,
    CASE
        WHEN empsexe = 'F' THEN 'Féminin'
        ELSE 'Masculin'
    END AS sexe
    FROM employe

SELECT empno, empnom,
    CASE empsexe
        WHEN 'F' THEN 'Féminin'
        ELSE 'Masculin'
    END AS sexe
    FROM employe
```

SELECT - Compléments : CASE

```
SELECT empno, empnom,
    DECODE(empsexe,'M', 'Masculin','F', 'Féminin') AS sexe
  FROM employe
```

Dialecte Oracle

SELECT - Compléments : CASE

```
SELECT AVG (CASE
                WHEN empsal > 2000 THEN empsal
                ELSE 2000
            END)
        FROM employe
```

```
SELECT *
        FROM ...
        WHERE   (CASE
                    WHEN x <> 0 THEN 1
                    ELSE 0
                END) = 1
```

SELECT - Compléments : CTE

Une **expression de table** (Common Table Expression) est une vue locale à la requête.

- Permet de simplifier certaines requêtes:
  - regroupement sur une colonne qui est le résultat d'un sous-select;
  - sous-requête plusieurs fois utilisée
- Indispensable pour permettre un traitement récursif.
- Peut être utilisé dans SELECT, UPDATE, DELETE

SELECT - Compléments : CTE

Retrouver les départements ayant le plus d'employés

```
SELECT empdpt
    FROM employe
    GROUP BY empdpt
    HAVING COUNT(*) >= ALL(SELECT COUNT(*)
                           FROM employe
                           GROUP BY empdpt)
```

```
WITH dptNb (dpt, nbEmp) AS
    (SELECT empdpt, COUNT(*)
        FROM employe
        GROUP BY empdpt)
SELECT dpt, nbEmp FROM dptNb
    WHERE nbEmp = (SELECT MAX(nbEmp) FROM dptNb)
```

SELECT - Compléments : CTE

Donne le nombre de département ayant le même nombre d'employés

```
WITH dptNb (dpt, nbEmp) AS
     (SELECT dptno, COUNT(empdpt)
          FROM departement LEFT JOIN employe
                              ON empdpt = dptno
          GROUP BY dptno)
SELECT nbEmp, COUNT(*)
     FROM dptNb
     GROUP BY nbEmp
```

SELECT - Compléments : Récursivité

L'ensemble des enfants d'un département.

```
WITH Q2 (dno, dlib, dpere) AS
     (SELECT dptno, dptlib, dptadm
          FROM departement
          WHERE dptadm IS NULL
     UNION ALL
     SELECT dptno, dptlib, dptadm
          FROM departement JOIN Q2
               ON Q2.dno = dptadm )
SELECT * FROM Q2
```

dno	dlib	dpere
D21	Direction	null
A00	Developpement	D21
E11	Ventes	D21
B01	Production	A00
C01	Maintenance	A00
D11	Support	E11
E01	Marketing	E11
E21	Formation	E11

SELECT - Compléments : Récursivité

Pour faire cela :

- une requête ancre qui constitue les points de départ
- une requête d'itération qui doit contenir une référence à la première et qui permet d'avancer dans l'arbre/graphe.
- le contrôle de l'arrêt est implicite; la récursivité s'arrête lorsque aucune ligne n'est retournée par l'invocation précédente.

SELECT - Compléments : Récursivité

La sémantique :

1. Scinder l'expression CTE en membres d'ancrage et en membres récursifs.
2. Exécuter le(s) membre(s) d'ancrage créant la première invocation ou le premier ensemble de résultats de base (T0).
3. Exécuter le(s) membre(s) récursif(s) avec Ti comme entrée et Ti+1 comme sortie.
4. Répéter l'étape 3 jusqu'à ce qu'un ensemble vide soit retourné.
5. Retourner l'ensemble de résultats. Il s'agit d'une opération UNION ALL de T0 à Tn.

SELECT - Compléments : Récursivité

Faites apparaître le nom du père, le niveau hiérarchique:

```
WITH Q2 (dno, dlib, dpere, niv, nompere) AS
    (SELECT dptno, dptlib, dptadm, 0, ''
        FROM departement
        WHERE dptadm IS NULL
    UNION ALL
    SELECT dptno, dptlib, dptadm, Q2.niv+1, Q2.dlib
        FROM departement JOIN Q2
            ON Q2.dno = dptadm )
SELECT * FROM Q2
```

dno	dlib	dpere	niv	nompere
D21	Direction	null	0	''
A00	Developpement	D21	1	Direction
E11	Ventes	D21	1	Direction
B01	Production	A00	2	Developpement
C01	Maintenance	A00	2	Developpement
D11	Support	E11	2	Ventes
E01	Marketing	E11	2	Ventes
E21	Formation	E11	2	Ventes

SELECT - Compléments : Récursivité

Inversez, donnez la liste des parents d'un département (E21) :

```
WITH Q2 (dno, dlib, dpere, niv) AS
     (SELECT dptno, dptlib, dptadm, 0
          FROM departement
          WHERE dptno='E21'
     UNION ALL
     SELECT dptno, dptlib, dptadm, Q2.niv+1
          FROM departement JOIN Q2 ON Q2.dpere = dptno )
SELECT * FROM Q2
```

dno	dlib	dpere	niv
E21	Formation	E11	0
E11	Ventes	D21	1
D21	Direction	null	2

SELECT - Compléments : FETCH

```
SELECT ... FROM ... WHERE ... ORDER BY ...
    FETCH FIRST n ROWS ONLY
```

Apparu dans (SQL:2008) . Oracle n'a inclus ce format qu'à partir de sa version 12c.

SELECT - Compléments : FETCH

```
SELECT dptNo
 FROM departement
 FETCH FIRST 3 ROWS ONLY
```

dptno
D11
C01
A00

```
SELECT dptNo
 FROM departement
 ORDER BY dptno
 FETCH FIRST 3 ROWS ONLY
```

dptno
A00
B01
C01

SELECT - Compléments : Fonction analytique

Pour chaque employé donnons le pourcentage qu'occupe son salaire dans la masse salariale globale

```
SELECT dptLib, dptNo, empNom, empSal * 100 / sumSal,
    FROM employe JOIN Departement ON dptno=empdpt,
         (SELECT sum(empSal) sumSal
            FROM employe )
            -- produit cartésien mais avec une seule ligne !
```

```
SELECT dptLib, dptNo, empNom,
        empSal * 100 / sum(empsal) OVER ()
    FROM employe JOIN Departement ON dptno=empdpt
```

La clause OVER permet d'exprimer que la fonction – ici analytique – SUM doit être appliqué sur l'ensemble des tuples mis en jeu.

SELECT - Compléments : Fonction analytique

Si nous désirons réaliser la même chose pour le pourcentage du salaire sur la masse du département de l'employé

```
SELECT dptLib, dptNo, empNom, empSal * 100 / sumSalDpt
    FROM employe
      JOIN Departement ON dptno=empdpt
      JOIN (SELECT empdpt, sum(empSal) sumSalDpt
              FROM employe GROUP BY empdpt) a
                    ON a.empdpt=dptno
```

```
SELECT dptLib, dptNo, empNom,
       empSal * 100 / sum(empsal) OVER (PARTITION BY dptno)
  FROM employe JOIN Departement On dptno=empdpt
```

SELECT - Compléments : Fonction analytique

La clause se présente donc ainsi :

```
fonction-analytique ([expression1] [ ,expression2] [ ,expr
OVER ( [ PARTITION BY ensemble attributs ]
[ ORDER BY ensemble attributs [NULLS FIRST|LAST] ])
```

On parle aussi de fonctions de fenêtrages : elles agissent sur un ensemble de données (fenêtre) définie par OVER !

SELECT - Compléments : Fonction analytique

En plus des fonctions d'agrégation, nous étudierons les fonctions de fenêtrage suivantes:

- **ROW_NUMBER** donne un indice unique à une ligne.
- **RANK** peut donner le même indice à plusieurs lignes mais renvoie des « trous » dans la plage d'indice renvoyée tandis que
- **DENSE_RANK** fonctionne pareil que RANK mais ne renvoie pas de « trous ».
- **LAG** permet de "regarder" en arrière
- **LEAD** permet de "regarder" en avant.
- **NTILE** ordonne les lignes en N packets, N étant un entier passé en paramètre. La clause order by est donc obligatoire. NTILE renvoie pour chaque ligne le numéro du paquet auquel cette ligne appartient.
- **FIRST_VALUE** et **LAST_VALUE** qui renvoie la première et la dernière valeur d'une expression pour une fenêtre donnée

SELECT - Compléments : Fonction analytique

Les employés avec leur numéro d'ordre d'après une séquence choisie :

```
SELECT empno, empNom, empdpt, empsal,
       ROW_NUMBER() OVER (PARTITION BY empdpt
                          ORDER by empsal) num
    FROM employe
    ORDER BY empdpt, num
```

La différence de salaire de chacun des employés vis à vis de celui qui le suit.

```
SELECT empno, empsal,
       empsal - LEAD (empsal) OVER(ORDER BY empsal)
    FROM Employe
```

SELECT - Compléments : Fonction analytique

Pour comparer quelques fonctions analytiques proches :

```
SELECT empdpt,
       row_number() OVER(ORDER by empdpt) ligne,
       rank() OVER(ORDER by empdpt) rang,
       dense_rank() OVER(ORDER by empdpt) rangDense
    FROM employe

SELECT empdpt, empsal,
       row_number() OVER(PARTITION BY empdpt
                         ORDER by empsal) ligne,
       rank() OVER(PARTITION BY empdpt
                         ORDER by empsal) rang,
       dense_rank() OVER(PARTITION BY empdpt
                         ORDER by empsal) rangDense
    FROM employe
```

SELECT - Compléments : Fonction analytique

Répartition des employés en 4 paquets suivant leur salaire

```
SELECT empnom, empsal,
       ntile(4) OVER(ORDER BY empsal) AS quart
    FROM employe
    WHERE empdpt = 'D11'
```

il y a 6 employés ; 6 MOD 4 = 2 ; les 2 sont répartis sur les premiers paquets

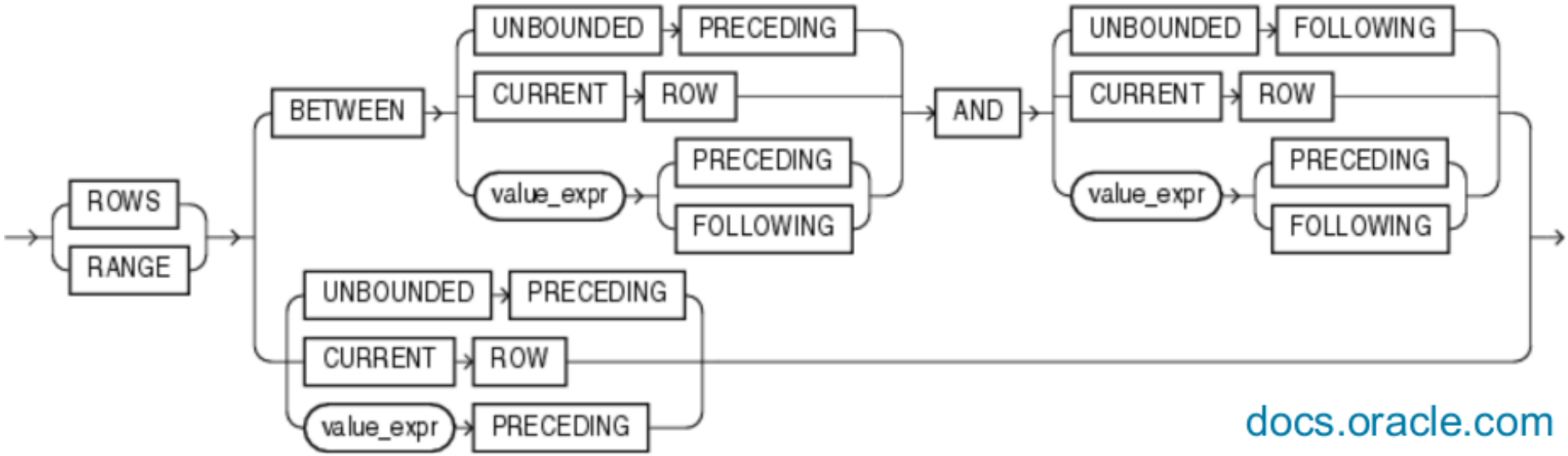
SELECT - Compléments : Fonction analytique

```
SELECT empnom, empsal,
       FIRST_VALUE(empnom)
           OVER(ORDER BY empsal
                    ROWS BETWEEN UNBOUNDED
                    PRECEDING AND UNBOUNDED FOLLOWING )
           AS bas_salaire,
       LAST_VALUE(empnom)
           OVER(ORDER BY empsal
                    ROWS BETWEEN UNBOUNDED
                    PRECEDING AND UNBOUNDED FOLLOWING )
           AS haut_salaire
    FROM employe
    WHERE empdpt = 'D21'
    ORDER BY empsal
```

SELECT - Compléments : Fonction analytique

analytic_clause :

windowing_clause :



docs.oracle.com

SELECT - Compléments : Fonction analytique

Pour chaque employé, nombre d'employés gagnant entre 5000 de plus et 5000 de moins.

```
SELECT empdpt, empNom, empSal,
        COUNT(empno) OVER (ORDER BY empsal
         RANGE BETWEEN 5000 PRECEDING AND 5000 FOLLOWING )
    FROM employe
```

SELECT - Compléments : Erreurs classiques

Attention

Requêtes erronnées à corriger!

```
SELECT dptno, dptlib
 FROM departement JOIN employe ON empdpt = dptno
 GROUP BY dptno

SELECT dptlib
 FROM departement JOIN employe ON empdpt = dptno
 WHERE empnom LIKE 'D%' OR empnom LIKE 'M%'

SELECT dptlib
 FROM departement dpt JOIN employe ON empdpt = dptno
 GROUP BY dptlib
 HAVING SUM(empsal)>(SELECT SUM(empsal)
                     FROM employe admin
                     WHERE dpt.dptadm = admin.empdpt)
```

SELECT - Compléments : Erreurs classiques

Attention

Requêtes erronnées à corriger!

```
SELECT dptlib
 FROM departement JOIN employe ON empdpt = dptno
 GROUP BY dptlib

SELECT empno, empnom
 FROM departement JOIN employe emp ON emp.empdpt = dptno
                  JOIN employe mgr ON mgr.empno = dptmgr
 WHERE emp.empnom = mgr.empnom AND emp.empno != mgr.empno

SELECT empno
 FROM employe emp
 WHERE empsal = (SELECT empsal
                 FROM employe eDpt
                 WHERE emp.empdpt = eDpt.empdpt)
```

SELECT - Compléments : Erreurs classiques

Donnez le nombre d'employés par numéro de département

1.
```sql
SELECT empdpt, COUNT(empno)
  FROM employe
  GROUP BY empdpt
```
2.
```sql
SELECT empdpt, COUNT(*)
  FROM employe JOIN departement ON empdpt = dptno
  GROUP BY empdpt
```
3.
```sql
SELECT empdpt, COUNT(empno)
  FROM employe JOIN departement ON empdpt = dptno
  GROUP BY empdpt
```
4.
```sql
SELECT empdpt, COUNT(*)
  FROM employe RIGHT JOIN departement ON empdpt = dptno
  GROUP BY empdpt
```
5.
```sql
SELECT dptno, COUNT(*)
  FROM employe RIGHT JOIN departement ON empdpt = dptno
  GROUP BY dptno
```

SELECT - Compléments : Erreurs classiques

Donnez par département le nombre de femmes

1.
```sql
SELECT dptno, COUNT(*)
   FROM departement JOIN employe ON empdpt = dptno
   WHERE empsexe = 'F'  GROUP BY dptno
```
2.
```sql
SELECT dptno, COUNT(*)
   FROM departement LEFT JOIN employe ON empdpt = dptno
   WHERE empsexe = 'F'
   GROUP BY dptno
```
3.
```sql
SELECT dptno, COUNT(empno)
   FROM departement LEFT JOIN employe
                ON (empdpt = dptno AND empsexe = 'F')
   GROUP BY dptno
```
4.
```sql
SELECT dptno, COUNT(empno)
   FROM departement LEFT JOIN employe ON empdpt = dptno
   WHERE empsexe = 'F' OR empsexe IS NULL
   GROUP BY dptno
```

SELECT - Compléments : Erreurs classiques

- Dans une UNION/EXCEPT/INTERSECT l'ORDER BY ne peut intervenir qu'au niveau du résultat global
- Un oubli de jointures lors de la spécification de plusieurs tables, donne un produit cartésien

SELECT - Compléments : Bonnes pratiques

Pour optimiser vos requêtes et/ou les rendre plus lisibles:

A éviter: SELECT *

```
SELECT *
 FROM employe
```

Préférer: SELECT liste attributs

```
SELECT empno, empnom, empsexe, empdpt, empsal
 FROM employe
```

SELECT - Compléments : Bonnes pratiques

A éviter: DISTINCT si non nécessaire

```
SELECT DISTINCT empno, empnom
 FROM employe
```

Préférer

```
SELECT empno, empnom
 FROM employe
```

SELECT - Compléments : Bonnes pratiques

A éviter: COUNT(attribut) si non nécessaire

```
SELECT COUNT(empno)
 FROM employe
```

Préférer: COUNT(*)

```
SELECT COUNT(*)
 FROM employe
```

SELECT - Compléments : Bonnes pratiques

A éviter: AND pour les intervalles fermés

```
SELECT empno
 FROM employe
 WHERE empsal >= 20000 AND empsal <= 30000
```

Préférer: BETWEEN

```
SELECT empno
 FROM employe
 WHERE empsal BETWEEN 20000 AND 30000
```

SELECT - Compléments : Bonnes pratiques

A éviter: Sous-requête non nécessaire

```
SELECT empno
 FROM employe
 WHERE empdpt IN (SELECT dptno
                  FROM departement
                  WHERE dptmgr = '030')
```

Préférer: Jointure

```
SELECT empno
 FROM employe JOIN departement ON dptno = empdpt
 WHERE dptmgr = '030'
```

SELECT - Compléments : Bonnes pratiques

A éviter: UNION-INTERSECT-EXCEPT

```
SELECT empno
 FROM employe
EXCEPT
SELECT dptmgr
 FROM departement
```

Préférer: Jointure

```
SELECT empno
 FROM employe LEFT JOIN departement ON dptmgr = empno
 WHERE dptmgr IS NULL
```

SELECT - Compléments : Bonnes pratiques

A éviter: Sous-requête corrélée

```
SELECT dptlib, (SELECT COUNT(*)
                  FROM employe
                  WHERE empdpt = dptno)
 FROM departement
```

Préférer: Jointure

```
SELECT dptlib, COUNT(empno)
 FROM departement LEFT JOIN employe ON empdpt = dptno
 GROUP BY dptno, dptlib
```

SELECT - Compléments : Bonnes pratiques

A éviter: MAX(id)+1

```
SELECT MAX(empno) + 1
 FROM employe
```

Préférer: SEQUENCE

```
CREATE SEQUENCE empSeq START WITH 1;
SELECT empSeq.NEXTVAL FROM dual;
SELECT empSeq.CURRVAL FROM dual;
```

SELECT - Compléments : Bonnes pratiques

A éviter: != ALL

```
SELECT empno
 FROM employe
 WHERE empdpt != ALL (SELECT empdpt
                      FROM employe
                      WHERE empsexe = 'F')
```

Préférer: NOT IN

```
SELECT empno
 FROM employe
 WHERE empdpt NOT IN (SELECT empdpt
                      FROM employe
                      WHERE empsexe = 'F')
```

SELECT - Compléments : Bonnes pratiques

A éviter: = ANY

```
SELECT empno
 FROM employe
 WHERE empdpt = ANY (SELECT empdpt
                     FROM employe
                     WHERE empsexe = 'F')
```

Préférer: IN

```
SELECT empno
 FROM employe
 WHERE empdpt IN (SELECT empdpt
                  FROM employe
                  WHERE empsexe = 'F')
```

SELECT - Compléments : Bonnes pratiques

- Pas de préfixe inutile
- Alias court et significatif
- Pas de jointure inutile
- Éviter les négations (NOT) et différences (!=)
- Utiliser l'ORDER BY que si nécessaire

Slides pour Persistance des données à l'ESI,
école supérieure d'informatique.

Crédits

Linux, pandoc, beamer, LaTeX