SGBD – Oracle
Cours BD
L2_DSI1

Langage d'Interrogation des Données

Interrogation de la base

- Syntaxe générale

```
SELECT ...
FROM ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
```

- L'ordre des clauses est imposé
- SELECT et FROM sont obligatoires

Clause SELECT

Précisions générales sur la clause SELECT :

- cette clause permet de choisir quelle(s) colonne(s) est retournée(s);
- on sépare les colonnes à retourner par des virgules;
- il est possible de préfixer une colonne par : « nom de la table. » (permet de lever l'ambiguïté);
- l'usage de * indique que toutes les colonnes sont sélectionnées;

```
SELECT *                    selection de toutes les colonnes
 FROM Table;
```

Exemple

```
SELECT *                    selection de toutes les colonnes  de la table employé
 FROM  Employe;

SELECT    Nom, Prenom       selection de deux colonne de la table employé
 FROM Employe;
```

select dans une expression

Select ***expression1 AS Alias1, Expression2 as alias2,***

From Tables;

- **alias** pour désigner la colonne dans une autre partie du select

Exemple

- **Select nom , sal aire + commission as "salaire totale"**

from employe;

- **Select nomE, salaire + NVL(commission, 0) as "Salaire Total"**

from employe;

Définir un alias de colonne

- Un alias de colonne :
- Renomme un entête de colonne
- Est utile avec les calculs
- Suit immédiatement le nom d'une colonne (le mot clé facultatif AS peut également être utilisé entre le nom de la colonne et l'alias)
- Nécessité des guillemets s'il contient des espaces ou des caractères spéciaux (# $), ou s'il distingue les majuscules des minuscules.

Définir un alias de colonne

- **SELECT last_name AS name, commission_pct as comm FROM employees;**

NAME	COMM
King	
Kochhar	
De Haan	

- **SELECT last_name " Name" , salary*12 "Annual Salary" FROM employees;**

Name	Annual Salary
King	288000
Kochhar	204000
De Haan	204000

Opérateur de concaténation

Un opérateur de concaténation :

- Lie des colonnes ou des chaînes de caractères à d'autres colonnes.
- Est représenté par deux barres verticales (||).
- Crée une colonne résultante qui est une expression de type caractère.

```
SELECT  last_name||job_id  AS   "Employees"
FROM employees;
```

Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP

Chaînes de caractères littérales

- Un littéral est un caractère, un nombre ou une date inclus dans l'instruction SELECT.
- Les valeurs littérales de type date et caractère doivent être incluses entre apostrophes.
- Chaque chaîne de caractères est sortie une fois pour chaque ligne renvoyée.

Utiliser des chaînes de caractères littérales

```
SELECT last_name ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```

Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Lorentz is a IT_PROG
Mourgos is a ST_MAN
Rajs is a ST_CLERK

. . .

20 rows selected.

Clause FROM

Précisions sur la clause FROM :

- cette clause permet de choisir quelle(s) table(s) est utilisée(s) pour la requête;
- on sépare les tables à utiliser dans la requête par des virgules.

FROM *table1 [synonyme1], table2 [synonyme2], ...*

- Produit cartésien des tables s'il y en a plusieurs
- Possible de se restreindre à un sous-ensemble du produit cartésien (voir jointure)

```
SELECT Employe.num_dept, Departement.num_dept
FROM Employe, Departement ;
```

usage de préfixes

```
SELECT *
FROM Employe, Departement;
```

intéressant => produit cartésien

Clause FROM :Exemple

```
select B.dept, A.nomD
from  departement  A, departement B
```

DEPT	NOMD
30	VENTES
20	VENTES
10	VENTES
30	RECHERCHE
20	RECHERCHE
10	RECHERCHE
30	FINANCE

Clause WHERE

- La clause WHERE comporte de nombreuses possibilités :
  - opérateurs de comparaison
  - opérateurs logiques
  - Jointures
  - sous-interrogations

Opérateurs de comparaison

- **=, !=, <, >, <=, >=, BETWEEN, LIKE, NOT LIKE, IN, NOT IN, IS NULL, IS NOT NULL**
- **LIKE permet d'utiliser des jokers :**
  - **% pour une chaîne de caractères de longueur** quelconque
  - **_ pour un seul caractère**
- Attention,

expression = **NULL n'est jamais vrai**,

il faut utiliser expression **IS NULL**

Opérateurs de comparaison

- select * from employe where poste = 'Secrétaire';
- select * from employe where salaire between 10000 and 15000;
- select * from employe where num_dept in (10, 30);
- select * from employe where commission is not null;
- select * from employe where nom like '%A%';

Opérateurs logiques

- **AND, OR, NOT**
- Exemples :
  - **select nomE from employe**
    where dept = 30
    and (salaire > 10000 or commission is null);
  - **select * from employe**
    where not (poste = 'Directeur' or
    poste ='Secrétaire');

et si on met **and** **à la** place de **or ?**

Variables de substitution

- Utilisez les variables de substitution pour :
  - Stocker temporairement des valeurs, via l'esperluette d'interprétation (&) et la double esperluette d'interprétation (&&)
- Utilisez des variables de substitution en complément des éléments suivants :
  - Conditions WHERE
  - Clauses ORDER BY
  - Expressions de colonne
  - Noms des tables
  - Instructions SELECT entières

Utilisez la variable de substitution (&)

- Utilisez une variable précédée d'une esperluette (&) afin d'inviter l'utilisateur à saisir une valeur:

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
```

```
old 3: WHERE employee_id = &employee_num
new 3: WHERE employee_id = 101
```

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
101	Kochhar	17000	90

Valeurs de type caractère et date avec des variables de substitution

- Utilisez des apostrophes pour les valeurs de type date et caractère :

```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```

LAST_NAME	DEPARTMENT_ID	SALARY*12
Hunold	60	108000
Ernst	60	72000
Lorentz	60	50400

Sous-interrogations (sous requêtes)

- Une clause WHERE peut comporter un ordre

SELECT emboîté :

```
select nome
 from emp
Where  poste = (select  poste  from  emp
                          Where  nome = 'Imen');
```

- Cette sous-interrogation doit ramener une seule ligne et une seule colonne
- Remplace le select interne par NULL s'il ne renvoie aucune ligne (ou erreur, suivant les SGBD)

- Affcher les numéros et les noms des employés ayant la même adresse et même salaire que l'employé Mounir
- Select NSS, nom
- From employe
- where (adresse,salaire) =(select adresse,salaire
- from employe
- where nom='Mounir');

Sous-interrogation ramenant 1 ligne, 1 colonne

- WHERE *expression op (SELECT …)*

  où *op est un des opérateurs de comparaison* =, !=, <, >, <=, >=

- Exemple :

```
select nome from  emp
where sal >= (select sal from emp
                    where nome = 'Mourad')
```

- Donner les noms et les numéros des employés du département Num2 ayant un salaires supérieur a tous les employés de département numero3
- Select nom, NSS
- From employ
- Where salaire != any (Select salaire
  - from employe
  - where num_dept=3);

  - In =Any

Sous-interrogation ramenant plusieurs lignes

WHERE *expression op ANY (SELECT ...)*

WHERE *expression op ALL (SELECT ...)*

WHERE *expression IN (SELECT ...)*

WHERE *expression NOT IN (SELECT ...)*

où *op est un des opérateurs de comparaison* =,

!=, <, >, <=, >=

- **ANY : vrai si la comparaison est vraie pour au** moins une des valeurs ramenées par le **SELECT**
- **ALL : vrai si la comparaison est vraie pour toutes**

les valeurs ramenées par le **SELECT**

Sous-interrogation ramenant plusieurs lignes

- Remarque :
  - = ANY est équivalent à IN
  - != ALL est équivalent à NOT IN

Exemple

select nomE, sal from employe

where sal > all (select sal from employe

where dep = 30);

Réflexion sur ALL

Quand « x > all (x1, x2, …, xn) » est faux ?

Si ∃ un xi tel que xi ≥ x

- **Si ∄ un xi tel que xi ≥ x, l'expression est vraie**

Donc si la liste (x1, x2, …, xn) est vide, l'expression est toujours vraie

Sous-interrogations ; optimisation

- Soit un select qui comporte une sous-interrogation :

```
select   nom
 from   employe
where    dept   in  (select  dept
                          from  departement
                          where  lieu  = 'Ariana');
```

- Pour chaque employé, le select peut lancer la sous interrogation pour savoir si l'employé est dans un département qui se trouve à Ariana
- En fait, le moteur de recherche du SGBD va optimiser en lançant d'abord la sous-interrogation et en conservant en mémoire les départements de Ariana

Sous-interrogations synchronisées

- Cette optimisation n'est pas possible quand la sous-interrogation utilise une des valeurs ramenées par l'interrogation principale
- On dit que la sous-interrogation est synchronisée avec l'interrogation principale
- Notation pointée utilisée pour se référer dans la sous-interrogation à une colonne de l'interrogation principale :

```
select nome
 from employe E
where  dept != (select dept
                from employe
                where  matr = E.sup);
```

Sous-interrogation ramenant 1 ligne de plusieurs colonnes

- **WHERE (*expr1, expr2,...) op (SELECT ...)***

où *op est* = *ou* ***!= (mais pas*** <, >, <=, >=*)*

- Exemple :

```
select nomE from emp
where (poste, sal) =(select poste, sal from emp
                     where nome = 'Mourad');
```

- le select renvoie 1 seule ligne

Sous-interrogation ramenant plusieurs lignes de plusieurs colonnes

- **WHERE (*expr1, expr2,...) op ANY (SELECT ...)***
- **WHERE (*expr1, expr2,...) op ALL (SELECT ...)***
- **WHERE (*expr1, expr2,...) IN (SELECT ...)***
- **WHERE (*expr1, expr2,...) NOT IN (SELECT ...)***

où *op est = ou* ***!= (mais pas*** *<, >, <=, >=)*

- Exemple :

```
select nomE from emp
where (poste, sal) in
(select poste, sal from emp
where dept = 10);
```

EXISTS

- La clause « EXISTS(select …) » est vraie si le select renvoie au moins une ligne
- La sous-interrogation est le plus souvent synchronisée :

```
select nome
 from emp E
where  exists (select null
                 from emp
                where sup = E.matr);
```

synchronisation

Jointure de plus de 2 tables

```
select  nome, nomp
from  employe, participation, projet
where employe.matr = participation.matr
and participation.codeP = projet.codeP
```

- Autre syntaxe :

```
select   nome, nomp
from employe   join  participation
on  employe.matr = participation.matr
join   projet on  participation.codep = projet.codep
```

Jointures

Traduction de l'équi-jointure
« employe {dept} departement »

```
select   nomE, nom_Dep
from   employe, deptartement
where   employe.dept = departement.dept
```

interdit à utiliser

- Autre syntaxe :

```
select nomE, nom_Dep
from employe  JOIN  departement
ON employe.dept = departement.dept
```

```
select nomE, nom_Dep
from employe  JOIN  departement using(num_dept)/***si l'attribut num_dept est la meme dans les deu
    tables*/
```

```
select nomE, nom_Dep
from employe  NATURAL JOIN  departement; )/***si l'attribut num_dept est la meme dans les deux **/
```

Jointure d'une table avec elle-même

- Alias indispensable pour le nom de la table afin de lever l'ambiguïté sur les colonnes :

```
select employe.nomE "Employé", supe.nomE "Supérieur"
from employe join employe supe
On employe.num_sup= supe.NSS
```

Jointure naturelle

- La jointure s'effectue sur *toutes les colonnes qui* ont le même nom dans les 2 tables ; ces colonnes ne sont pas répétées dans la jointure
- Les colonnes qui participent à la jointure ne doivent être préfixées par un nom de table

Exemples de jointure naturelle

- **select nomE, nomD, dept**

from employe NATURAL JOIN departement;

- **select nome, nomp**

from employe

NATURAL JOIN participation

NATURAL JOIN projet

Jointures « non équi »

- Les jointures « non équi » peuvent être traduites comme les équi-jointures, en utilisant d'autres opérateurs de comparaison

```
select emp1.nome, emp2.nome
from   employe emp1
join  employe  emp2
on   emp1.salaire < emp2.salaire;
```

Jointure externe

- Dans une jointure n'apparaissent que les lignes qui ont une ligne correspondante dans l'autre table
- Dans l'exemple suivant, un département qui n'a pas d'employé n'apparaîtra pas :

select nomE, nomD

from employe join deptartement

on employe.dept = departement.dept

- Si on veut qu'il apparaisse, on doit utiliser une jointure externe

Syntaxe SQL-2 de la jointure externe

select nomE, nom_dep

from employe RIGHT OUTER JOIN departement

ON employe.dep = departement.dep;

- RIGHT indique que l'on veut afficher toutes les lignes de la table de droite (departement)
- Ça revient à ajouter une « ligne fictive » dans l'autre table employe
- Cette ligne fictive aura toutes ses colonnes à **null, sauf la colonne de jointure**
- Il existe de même **LEFT OUTER JOIN et FULL OUTER JOIN**

Fonctions de groupe

- Les fonctions de groupe peuvent apparaître dans une expression du select ou du having :
- AVG() moyenne
- SUM () somme
- MIN() plus petite valeur
- MAX() plus grande valeur
- COUNT() nombre de lignes
  - COUNT(nom_colonne) nombre de valeurs non NULL dans la colonne
  - COUNT(DISTINCT colonne) nombre de valeurs distinctes

Exemples

- **select count(NSS)**
  from employe;

- **select count(commission)**
  from employe
  where dept = 10;

- **Select sum(salaire)**
  from employe
  where dept = 10;

- **select max(salaire)**
  from employe
  where poste = 'INGENIEUR';

- **select nome, salaire**
  from employe
  where salaire = max(salaire);

Interdit, car dept et max(sal)
ne sont pas au même niveau de regroupement

Niveaux de regroupement

- A un niveau de profondeur (relativement aux sous interrogations) d'un SELECT, les fonctions de groupe et les colonnes doivent être toutes du même niveau de regroupement
- Par exemple, si on veut le nom et le salaire des employés qui gagnent le plus dans l'entreprise.

select nome, salaire from employe
where salaire= max(salaire)

- Solution :

select nome, salaire from employe
where salaire = (select max(salaire) from employe)

Clause GROUP BY

- Il est possible de subdiviser la table en groupes, chaque groupe étant l'ensemble des lignes ayant une valeur commune.
- Permet de regrouper des lignes qui ont les mêmes valeurs pour des expressions :

GROUP BY *expression1, expression2,…*

- Il n'est affiché qu'une seule ligne par regroupement de lignes
- Exemple :

```
select num_dept, count(NSS)
from employe
group by num_dept;
;
```

Clause GROUP BY
Exemples

- **select dept, poste, count(NSS)**

from employe
group by dept, poste;

- **select dept, count(comm)**

from employe
group by dept;

```
select nome, dept, sal from emp
where (dept, sal) in
   (select dept, max(sal) from emp
    group by dept);
```

Schéma à retenir pour obtenir des optima sur des regroupements

- Sans doute moins performant :

select nome, dept, sal
from employe E
where salaire = (select max(salaire)
from employe
where dept = E.dept);

- Donner les numéros et le nombre des employés des département ayant plus de 30 employés

- Select **num_dept**, count(NSS)
- From employe
- group by **num_dept**
- **having** count(NSS)>30;

Clause HAVING

- HAVING prédicat
- sert à préciser quels groupes doivent être sélectionnés.
- Elle se place après la clause GROUP BY.
- Le prédicat suit la même syntaxe que celui de la clause WHERE. Cependant,
  - il ne peut porter que sur des caractéristiques de groupe : fonction de groupe ou expression figurant dans la clause GROUP BY.

Clause HAVING

- On peut évidemment combiner toutes les clauses, des jointures et des sous-interrogations. La requête suivante donne le nom du département (et son nombre de secrétaires) qui a le plus de secrétaires :

```
SELECT   NOM_Department  ,COUNT(employee_id) "Nombre de secretaires"
FROM EMPLoyees   NATURAL  JOIN  DEPTARTMENTS
WHERE  POSTE = 'SECRETAIRE'
GROUP BY NOM_Department
 HAVING    COUNT(NSS) =          (SELECT   MAX(COUNT(NSS))
                                FROM EMPLOYESS
                                      WHERE POSTE = 'SECRETAIRE'
                                      GROUP BY DEPTARTMENT_ID);
```

On remarquera que la dernière sous-interrogation est indispensable car MAX(COUNT(*)) n'est pas au même niveau de regroupement que les autres expressions du premier SELECT.

Objet Séquence

- Permet d'obtenir des valeurs incrémentales
- Équivalent des colonnes AUTO_INCREMENT de MySql ou IDENTITY de SqlServer
- N'est pas associée à une colonne particulière
- Verrouillage automatique en cas de concurrence d'accès
- Valeur suivante : <nom_séquence>.NEXTVAL
- Valeur courante : <nom_séquence>.CURRVAL

Objet Séquence création et utilisation

```
CREATE SEQUENCE nom_séquence
START WITH valeur_départ
INCREMENT BY incrément;
```

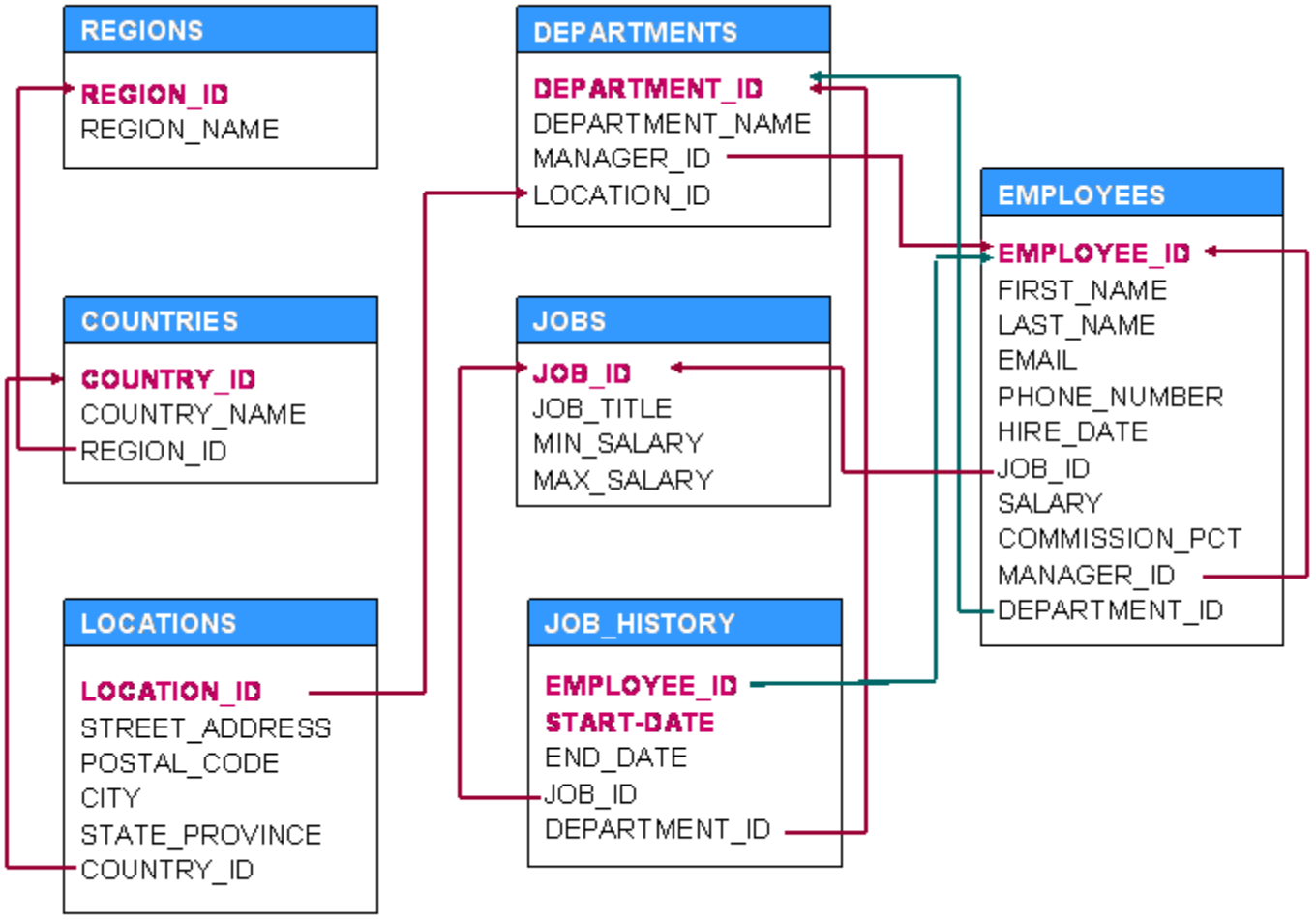
```
INSERT INTO t1 VALUES
(nom_séquence.NEXTVAL, …..);
INSERT INTO t2 VALUES
(……., nom_séquence.CURRVAL);
```

```
DROP SEQUENCE nom_séquence;
```

Objet Séquence
Exemple de mise en oeuvre

```
SQL> CREATE TABLE client (idClient NUMBER PRIMARY KEY,
  2  nomClient VARCHAR(20));
Table créée.
SQL> CREATE TABLE compte (idCompte NUMBER PRIMARY KEY,
  2  nomCompte VARCHAR(30), idClient REFERENCES client);
Table créée.
SQL> CREATE SEQUENCE seq_client START WITH 1 INCREMENT BY 1;
Séquence créée.
SQL> CREATE SEQUENCE seq_compte START WITH 1 INCREMENT BY 1;
Séquence créée.
SQL> INSERT INTO client VALUES(seq_client.NEXTVAL,'Mourad');
1 ligne créée.
SQL> SELECT seq_client.CURRVAL FROM dual;
   CURRVAL
----------
```

```
SQL> INSERT INTO compte VALUES(seq_compte.NEXTVAL,'Compte Courant
Mourad',seq_client.CURRVAL);
1 ligne créée.
SQL> INSERT INTO compte VALUES(seq_compte.NEXTVAL,'Compte Epargne
Mourad',seq_client.CURRVAL);
1 ligne créée.
SQL> SELECT * FROM client;
  IDCLIENT NOMCLIENT
---------- --------------------
         1 Mourad
SQL> SELECT * FROM compte;
  IDCOMPTE NOMCOMPTE                      IDCLIENT
---------- ------------------------------ ----------
         1 Compte Courant Mourad                   1
         2 Compte Epargne Mourad                   1
```


REGIONS
REGION_ID
REGION_NAME
DEPARTMENTS
DEPARTMENT_ID
DEPARTMENT_NAME
MANAGER_ID
LOCATION_ID
EMPLOYEES
EMPLOYEE_ID
FIRST_NAME
LAST_NAME
EMAIL
PHONE_NUMBER
HIRE_DATE
JOB_ID
SALARY
COMMISSION_PCT
MANAGER_ID
DEPARTMENT_ID
COUNTRIES
COUNTRY_ID
COUNTRY_NAME
REGION_ID
JOBS
JOB_ID
JOB_TITLE
MIN_SALARY
MAX_SALARY
LOCATIONS
LOCATION_ID
STREET_ADDRESS
POSTAL_CODE
CITY
STATE_PROVINCE
COUNTRY_ID
JOB_HISTORY
EMPLOYEE_ID
START-DATE
END_DATE
JOB_ID
DEPARTMENT_ID