

Le langage d'interrogation de données



■ Plan:

- L'instruction SELECT
- Fonctions
- Traitement de groupe
- Tri



Présentation

- Fonctions essentielles
 - Écriture de requêtes de consultation de la base de données
 - Le LID ne permet pas de créer de nouvelles relations!!
- Principales instructions
 - Requêtes mono-tables
 - SELECT Selection, projection
 - Requêtes multi-tables
 - SELECT jointures
 - UNION, INTERSECT, MINUS ensemblistes



Base utilisées pour les exemples

- Base1: Employés/Services
 - Employes(idEmploye; NomEmploye, Salaire, DateEmbauche, #idService)
 - Services(idService, NomService, Etage)
- Base2: Commande_de_Produits
 - Client(Cocli, Nomvli, ville)
 - Produit(Copro, Libellé, Pu)
 - Facture (Nufact, Datefact, #Cocli, Montant)
 - Detail(#Nufact, #Copro, Qte)



SELECT : syntaxe

```
SELECT [DISTINCT/ALL] <colonne>  
FROM <Table>  
[WHERE <condition>]  
[GROUP BY <colonne> | <expression>]  
[HAVING <condition>]  
[ORDER BY <colonne> | <expression> (asc | desc)];
```



SELECT : syntaxe

- **[DISTINCT/ALL]**
 - Élimination ou conservation des tuples doublons
- **<expression>**
 - Expression arithmétique, resultat de calcul ou fonction
- **WHERE <condition>**
 - Selection de tuples
- **[GROUP BY]**
 - Constitution de groupe de tuples ayant des valeurs identique
- **[HAVING <condition>]**
 - Condition sur les groupes identiques
- **[ORDER BY]**
 - Affichage des tuples dans un ordre donné



Affichage

- Analyse syntaxique de la requête
 - Affichage
 - de la ligne erronée
 - de la position de l'erreur dans la ligne
 - d'un message explicatif
- Résultat de la requête
 - Affichage sous forme de tableau avec les noms de colonnes en en-tête



Clause SELECT

SELECT **Liste** des colonnes désirées
* pour toutes les colonnes
expr arithmétique, résultat
d'une **fonction**

Instruction DISTINCT

permet d'éliminer les doublons car SQL affiche le résultat d'une requête sans éliminer les doublons.



Clause SELECT

- Modification de noms des colonnes
 - Pour avoir un nom plus explicite
 - Mettre le nom choisi immédiatement après le nom de la colonne

```
SELECT copro CodeProduit, pu PrixUnitaire  
FROM Produit;
```

nouveau nom

```
SELECT copro AS Codeproduit, pu PrixUnitaire  
FROM Produit;
```




Clause FROM

- Noms des tables concernées
- Il est possible de donner à une table un autre nom: ALIAS
 - Intérêts:
 - Employer une abréviation du nom de la table
 - Permettre une jointure d'une table sur elle-même (voir plus loin)
 - `SELECT copro, pu FROM Produit P;`

Alias



Clause WHERE

<opérande> <opérateur> <opérande>

Opérandes et opérateurs doivent être compatibles

On peut regrouper les conditions grâce aux opérateurs

- OR
- AND

On peut prendre la condition contraire grâce à l'opérateur **NOT**



Conditions: opérandes

- Il peut s'agir:
 - d'un nom de colonne
 - d'une valeur
 - Numérique (entière ou décimale)
 - Alphanumérique (entre ")
 - Date
 - d'une expression
 - Arithmétique + - * /
 - Utilisation des fonctions
 - d'une sous requête
 - Requête fournissant une valeur ou une liste de valeurs compatibles



Conditions: opérateurs (1)

- Opérateurs arithmétiques + - * /
- Opérateur de concaténation de chaîne

<chaîne1> || <chaîne2>

- Opérateurs de comparaison classiques

= <> != > >= < <=

SELECT NuFact FROM Detail

WHERE Copro='P03' AND Qte>20;

Copro type
Varchar2

Qte type Number



Conditions: opérateurs (2)

- Opérateurs de comparaison spécifiques (1)

IN : Couleur IN ('bleu', 'jaune', 'vert');

NOT IN: IdEquipe NOT IN (1, 2, 3);

ANY, SOME: condition vraie si satisfaite pour une ou plusieurs valeurs de la liste (expr arithmetique)

PRIX < ANY (12,24, 45);

PRIX = ANY (12,24, 45);

ALL: condition vraie satisfaite pour toute les valeurs de la liste

PRIX < ALL (12, 23, 43);

IS (NOT) NULL: condition vraie si la valeur de l'attribut est (n'est pas) nulle



Conditions: opérateurs (3)

- Opérateurs de comparaison spécifiques (2)

[NOT] BETWEEN __ AND __: condition vraie si le premier opérateur est (n'est pas) compris entre les deux valeurs données (bornes comprises)

[NOT] EXISTS <sous requête>: condition vraie si la requête retourne (ne retourne pas) au moins un tuple

[NOT] LIKE : condition vraie si la 1ere opérande contient (ne contient pas) un ensemble de caractère

Nom LIKE ' _ _ _0%' : nom de 3 lettres fini par 0

_: caractère quelconque %: indique la fin



Conditions: exemples

```
SELECT nom, ville FROM Client  
WHERE ville LIKE '_a%';
```

```
SELECT * FROM Articles  
WHERE Prix IS NULL;
```

```
SELECT NumFact FROM Factures  
WHERE DateFact between '01-JAN-95' AND '31-DEC-  
95';
```



Fonctions d'agrégats (1)

- On les rencontre:
 - Soit dans les expressions à afficher
 - Soit dans les opérandes de conditions
- Fonctions numériques
 - Fonctions trigonométriques et logarithmiques
 - Fonctions de transformation
 - ABS(n) Valeur absolue de n
 - CEUIL(n) Entier supérieur ou égal à n
 - FLOOR(n) Troncature à valeur entière
 - MOD(m,n) Reste de la division de m par n
 - POWER(m,n) m élevé à la puissance n
 - ROUND(m,n) m arrondi à n décimales
 - SIGN(m,n) Signe (-1 si <0 , 0 si $=0$, 1 si >0)
 - SQRT(n) racine carré de n (0 si <0)
 - TRUNC(m,n) m tronqué à n décimales



Fonctions (2)

- Fonctions sur un ensemble de valeurs: fonctions de groupe
- AVG(n) moyenne des valeurs de n (valeurs nulles non comptées)
- COUNT(*) nombre de tuples renvoyés par la requête
- COUNT(n) nombre de valeurs non nulles
- SUM(n) somme des valeurs de n
- MAX(n) valeur maximum de n
- MIN(n) valeur minimum de n
- STDDEV(n) écart type de n (valeurs nulles non comptées)
- VARIANCE(n) variance de n (valeurs nulles non comptées)

Ces fonctions ne peuvent pas apparaître dans la clause WHERE



Fonctions (3)

- Fonctions sur les chaînes de caractères
- INITCAP(c)
- LOWER(c)
- UPPER(c)
- LTRIM (c)
- RTRIM (c)
- REPLACE (c,c1,c2)
- SOUNDEX (c)
- SUBSTR(c,c1,c2)
- LENGTH(c)



Fonctions (4)

Fonctions sur les dates

Add_MONTH(d,n)

LAST_DAY(d)

MONTH_BETWEEN(d1,d2)

NEW_TIME(..)

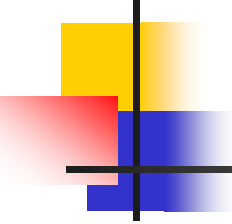
SYSDATE

To_CHAR(...)

TO_DATE(...)

Expressions complémentaires

NVL (s1




Exemples (1)

```
SELECT Copro, libelle, CEIL(pu*1.2)  
FROM Produit;
```

```
SELECT AVG(montant) FROM Facture;
```

```
SELECT COUNT(DISTINCT(Ville))  
FROM Clients
```



Exemples (2)

Résultats (de Pierre)

<i>Matière</i>	<i>Coef</i>	<i>Note</i>
Maths	4	15
Sc Nat	3	9
Sc Phy	3	12
Français	2	13
Sc Hum	2	11
Anglais	1	10
Sport	1	12

SELECT MAX(*Note*) FROM *Résultats* → 15

Quelle est la plus mauvaise note ?

SELECT MIN(*Note*) FROM *Résultats* → 9

Quelle la somme pondérée des notes ?

SELECT SUM(*Note***Coef*) FROM *Résultats* → 193

Quelle est la moyenne (pondérée) de Pierre ?

SELECT SUM(*Note***Coef*)/SUM(*Coef*) FROM *Résultats*

→ 12,06

Dans combien de matières Pierre a-t-il eu plus de 12 ?

SELECT COUNT(*) FROM *Résultats* WHERE *Note* > 12

→ 2



Traitements de groupe

Syntaxe

GROUP BY *liste_attributs*

HAVING *condition avec fonction*

Cette clause regroupe les résultats par valeur selon la condition

- Group By:
 - Permet de créer des sous ensembles regroupant des tuples ayant une caractéristique commune
 - Permet d'utiliser des fonctions de calcul de groupe
- Having:
 - Permet de ne prendre en compte que les sous-ensembles vérifiant une condition donnée
- Fonctions de groupage:
 - AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE

Clause de groupage GROUP BY

Obligatoire sur les colonnes du SELECT non concernées par le groupage

Résultats (de Pierre)

Matière	Coef	Note
Maths	4	15
Sc Nat	3	9
Sc Phy	3	12
Français	2	13
Sc Hum	2	11
Anglais	1	10
Sport	1	12

Quelle est la note moyenne pour chaque coefficient ?

SELECT coef, Avg(note) as Moyenne
FROM Résultats
GROUP BY coef;

Coef	Moyenne
1	11
2	12
3	10.5
4	15

Quels sont les coefficients auxquels participe une seule matière ?

SELECT coef
FROM Résultats GROUP BY coef
HAVING count(*)=1;

Coef
4



Tris

- Tris seulement sur l'affichage
- Tris implicites
 - DISTINCT, GROUP BY
- Tris explicites
 - ORDER BY ...[ASC|DESC]
 - Peut se faire sur une ou plusieurs colonnes
 - Par défaut, tris croissant (ASC)

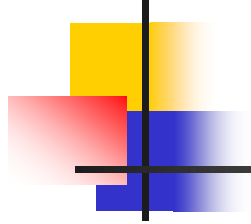
Exemple:

```
SELECT Cocli, ville  
FROM Clients  
ORDER BY CoCli ASC;
```




La table DUAL

- Si l'instruction SELECT est exécutée, la clause WHERE est obligatoire même si on a pas toujours de raison d'utiliser une table particulière.
- Ceci se produit par exemple, si vous voulez
 - Récupérer la date système
 - Faire un appel à une fonction
- Pour se faire, vous disposez d'une table DUAL avec
 - Une seule colonne DUMMY de type VARCHAR2(1)
 - Un seul tuple de valeur ('X')
- Exemple
 - Select Sysdate from Dual;
 - Select 3*2 from Dual;



Le langage d'interrogation de données

Partie 2: Requêtes multi-tables



Présentation

- Trois (3) types de requêtes multi-tables
 - Opérateurs ensemblistes
 - UNION
 - INTERSECT
 - MINUS
 - Requêtes imbriqués
 - Utilisation de sous requêtes dans les conditions WHERE ou HAVING
 - Jointure
- Quand une table est utilisée deux fois dans une requête, on lève l'ambigüité en renommant les tables
 - Client X et Client Y
x.Cocli, Y.Cocli



Opérateurs ensemblistes

Table1 \cup Table2 :

```
SELECT liste_attributs FROM table1  
UNION  
SELECT liste_attributs FROM table2 ;
```

Table1 \cap Table2 :

```
SELECT liste_attributs FROM table1  
INTERSECT  
SELECT liste_attributs FROM table2 ;
```

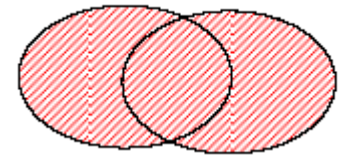
Table1 - Table2 :

```
SELECT liste_attributs FROM table1  
MINUS  
SELECT liste_attributs FROM table2 ;
```

Les résultats fournis par les sous requêtes doivent être UNION-compatibles
- Même nombre de colonnes de même type

UNION

- Cet opérateur permet d'effectuer une UNION des tuples sélectionnés par deux clauses *SELECT* (les deux tables sur lesquelles on travaille devant avoir le même schéma).
- ```
SELECT ---- FROM ---- WHERE -----
UNION
SELECT ---- FROM ---- WHERE -----
```
- Par défaut les doublons sont automatiquement éliminés. Pour conserver les doublons, il est possible d'utiliser une clause *UNION ALL*.





# UNION: Exemple

---

Lister tous les enseignants

```
SELECT Nom, Prénom
```

```
FROM MdC
```

```
UNION
```

```
SELECT Nom, Prénom
```

```
FROM Professeur ;
```



# INTERSECT

---

- Cet opérateur permet d'effectuer une INTERSECTION des tuples sélectionnés par deux clauses *SELECT* (les deux tables sur lesquelles on travaille devant avoir le même schéma).

```
SELECT ---- FROM ---- WHERE -----
INTERSECT
SELECT ---- FROM ---- WHERE -----
```

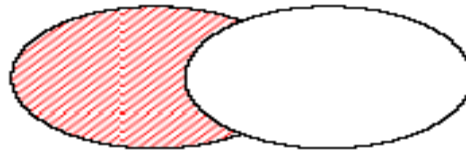
*INTERSECT* n'étant pas implémenté dans tous les SGBD, il est possible de le remplacer par des commandes usuelles :

```
SELECT a,b FROM table1
WHERE EXISTS (SELECT c,d FROM table2
 WHERE a=c AND b=d)
```

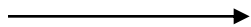


# MINUS: Exemple

---



- `SELECT CoPro FROM Produit`  
`MINUS`  
`(SELECT DISTINCT CoPro FROM Detail)`



Produits non commandés





# Requetes imbriquées

---

Une clause WHERE est elle-même le résultat d'un SELECT

Deux cas:

## Requetes non synchronisées

Les sous requetes sont executées en premier

## Requetes synchronisées

Les sous requetes sont executées pour chaque valeur de la requete principale



# Requetes synchronisées

---

- Deux tables distinctes

```
SELECT Nomcli from Clients
```

```
WHERE EXISTS
```

```
(Select * from Facture
```

```
WHERE Facture.Codecli=Clients.Cocli
and Numfact=3);
```

```
SELECT Nomcli FROM Client
```

```
WHERE NOT EXISTS
```

```
(Select * from Facture
```

```
Where Facture.Cocli=Client.Cocli);
```



# Requetes synchronisées

---

Auteurs n'ayant pas écrits d'ouvrages

```
SELECT * From Auteurs
```

```
Where not exists
```

```
(select * From Ecrit
```

```
where
```

```
Auteurs.NumAuteur=Ecrit.NumAuteur)
```



# Requetes synchronisées

---

- Plus de deux tables

```
SELECT * FROM Produits
```

```
WHERE EXISTS
```

```
(Select * From Detail
```

```
Where Produits.Copro=Detail.Copro
```

```
and EXISTS
```

```
(Select * from Facture
```

```
Where Detail.NuFact=Facture.Nufact
```

```
and DateFact= '01-JUN-95'));
```



# Requêtes synchronisées

---

- Deux fois la même table

```
SELECT Libelle FROM Produits
WHERE Pu >= ALL (Select Pu From Produits);
```



# Requêtes non synchronisées

---

- Deux tables distinctes

Select NomCli From Clients

WHERE Cocli = (Select CoCli From Facture  
Where NuFact=3);



# Requetes non synchronisées

---

- Plus de deux tables

```
Select * From Produits
```

```
Where CoPro IN
```

```
(SELECT DISTINCT Copro From detail
```

```
Where NuFact In
```

```
(SELECT NuFact From Facture
```

```
Where dateFact='04-JUN-45'));
```



# Requêtes non synchronisées

- Deux fois la même table

```
Select Libelle From Produits
Where Pu= (Select Max(Pu)
 From Produits);
```





# Traduction de l'opérateur de Division

---

Select NuFact FROM Facture

Where NOT EXISTS

(Select \* From Produits

Where NOT EXISTS

(Select \* FROM Detail

Where

Detail.NuFact=Facture.NuFact

and Produit.Copro=Detail.Copro));



# JOINTURES

---

- but: créer toutes les combinaisons significatives entre tuples de deux relations

- Un seul bloc `Select ...From... Where....`

significatives : portent la même valeur pour les attributs de même domaine  
!

- Précondition: les deux relations ont au moins un attribut de même domaine
- Equi jointures (jointure naturelle)  
`Select ..; From <table1> <table2>`  
`Where <table1>.<col>=<table2>.<col>`

# JOINTURES

- **Jointure** dite « naturelle »:

Coueurs

| Numéro coureur | Nom Coureur      | Code équipe | Code pays |
|----------------|------------------|-------------|-----------|
| 8              | ULLRICH Jan      | TEL         | ALL       |
| 31             | JALABERT Laurent | ONC         | FRA       |
| 61             | ROMINGER Tony    | COF         | SUI       |
| 91             | BOARDMAN Chris   | GAN         | G-B       |

| Code pays | Nom Pays          |
|-----------|-------------------|
| ALL       | Allemagne         |
| FRA       | France            |
| SUI       | Suisse            |
| G-B       | Grande - Bretagne |

Pays

Les pays de  
tous les  
Coueurs?

Relation  
résultat

| Numéro coureur | Nom Coureur      | Code équipe | Code pays | Nom Pays          |
|----------------|------------------|-------------|-----------|-------------------|
| 8              | ULLRICH Jan      | TEL         | ALL       | Allemagne         |
| 31             | JALABERT Laurent | ONC         | FRA       | France            |
| 61             | ROMINGER Tony    | COF         | SUI       | Suisse            |
| 91             | BOARDMAN Chris   | GAN         | G-B       | Grande - Bretagne |

Coueurs.CodePays= Pays.CodePays



# Exemple Equi Jointures

---

- La jointure peut aussi se faire sur une inégalité par l'intermédiaire de n'importe quel opérateur <, >, <=, >=, BETWEEN, LIKE, IN
- Ex: Affichage des pays (hors Grande Bretagne) de tous les coureurs

| Numéro coureur | Nom Coureur      |
|----------------|------------------|
| 8              | ULLRICH Jan      |
| 31             | JALABERT Laurent |
| 61             | ROMINGER Tony    |

Equi jointures: Deux tables distinctes

```
SELECT Numerocoureur, Nomcoureur
```

```
FROM Coureurs, Pays
```

```
WHERE Coureurs.Codepays=Pays.Codepays and Nompays != 'Grande Bretagne');
```



# Exemples qui Jointures

---

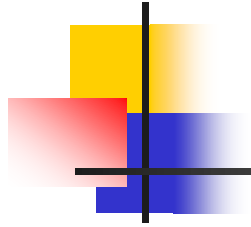
- Plus de deux tables
  - `Select Produit.copro, libellé, pu`  
`From Produits, Detail, Facture`  
`Where Produits.Copro= Detail.Copro`  
`and Detail.Nufact= Facture.Nufact`  
`and datefact ='09-JAN-06';`
  
- Deux fois la meme table
  - Deux cliens habitants la meme ville  
`Select X.Cocli, Y.Cocli, X.Ville From Client X, Client Y`  
`Where X.nom != Y.nom and X.Ville=Y.Ville;`



# En résumé

---

- Il y'a de nombreuses façons de faire une même requête
  - Il est préférable d'utiliser des jointures
- Bien noter la façon dont les divisions de l'algèbre relationnelle sont traitées
  - Même s'il existe d'autres façons de faire...



FIN