	**Bases de données**	**NSI T^ale^**
	Le Langage d'Interrogation des Données	Cours/TD

*

1. Introduction

Le LID permet principalement d'écrire des requêtes de consultation de la base de données dont le résultat est fournit immédiatement sur l'écran. Le résultat est présenté sous forme de tableau avec le nom des colonnes en en-tête.
Pour rappel, le langage SQL, même s'il est normalisé, peut être décliné avec de légères différences de syntaxes en fonction des SGBD, avec en particulier un usage encore courant de formes ne respectant pas les standards.
Ce chapitre n'est qu'une introduction au LID dont certaines instructions seront ignorées (entre autres- GROUP BY et HAVING).

2. Exemple d'application - Rappel

Les exemples présentés dans ce chapitre et le chapitre 7 sont tirés des quatre relations suivantes :

CLIENT(Cocli, Nomcli, Ville)
PRODUIT(Copro, Libelle, Pu)
FACTURE(Nufact, Datefac, #Cocli, Montant)
DETAIL(#Nufact, #Copro, Qte)

Voici leurs représentations tabulaires en extension :

PRODUIT

Copro	Libelle	Pu
P01	Agrafes	3,00
P02	Bristol	4,00
P03	Classeur	33,00
P04	Clé USB	40,00
P05	Equerre	1,50

CLIENT

Cocli	Nomcli	Ville
C001	Albert	Toulouse
C003	Bernard	Auch
C007	Claude	Paris
C015	Daniel	Toulouse
C025	Emile	Paris

FACTURE

Nufact	Datefac	Cocli	Montant
001	05/02/94	C001	100,00
002	10/08/94	C003	250,00
003	15/12/94	C007	550,00
004	10/02/95	C003	290,00
005	01/03/95	C001	800,00

DETAIL

Nufact	Copro	Qte
001	P01	10
001	P02	20
002	P01	15
002	P02	50
003	P03	10
004	P01	20
004	P03	10
004	P05	10
005	P03	20

3. Requêtes mono-tables

→ L'instruction SELECT.

```
01 |  select [distinct|all] * |<colonne>|<expression>
02 |  from <nom_table>
03 |  [ <condition>]
04 |  [ <condition>]
05 |  [order by  <colonne>|<expression> [asc/desc],...];
```

Les instructions entre crochets [ ] sont facultatives. La barre verticale | exprime un choix (ou)

- [distinct|all] permet l'élimination ou la conservation des tuples identiques dans le résultat.
- <expression> est une expression arithmétique, résultat de calcul ou de fonction.
- <condition> permet la sélection de tuples.
- [order by] permet l'affichage des tuples dans un ordre donné (tri).

Le ***select*** simple permet d'effectuer des opérations de **projection** (choix de colonnes) et les conditions du ***where*** des opérations de **sélection**.

→ L'affichage

Le résultat de la requête est affiché sous forme d'un tableau avec les noms des colonnes en en-tête.
Pour présenter un tableau avec un nom de colonne différent (nom plus explicite ou cas d'une expression), il suffit de mettre le nom choisi immédiatement après le nom de la colonne (sans ,) ou après le mot clé *as*.

Exemple:

```
01 |   select Copro code_produit, Pu prix_unitaire  from Produit;
02 |
03 |   select Copro as code_produit, Pu as prix_unitaire
04 |   from Produit;
```

→ Les conditions

Syntaxe : La forme générale des conditions est la suivante:

<operande><operateur><operande>

Opérandes et opérateurs doivent être compatibles. On peut regrouper des conditions grâce aux opérateurs logiques *or* et *and*. On peut prendre la condition contraire par l'opérateur *not*.

Opérandes

Les opérandes peuvent être:

- un nom de colonne;
- une valeur:
  - numérique (entière ou décimale),
  - alphanumérique (entre ' '),
  - date: ' DD-MMM-AA' (avec MMM en anglais);
- une expression:
  - arithmétique (+ - * /),
  - utilisant des fonctions;
- une sous-requête (requête fournissant une valeur ou une liste de valeurs compatibles).

Opérateurs arithmétiques

+ - * /

Ils ne peuvent être utilisés qu'avec des colonnes ou des valeurs numériques.

Opérateurs de concaténation de chaînes

<chaine1 >|| < chaine2 >

Opérateurs de comparaison

On peut citer les opérateurs classiques:

= <> != > >= < <=

Il existe un nombre important d'opérateurs spécifiques (*IN, EXISTS*, …) que nous ne verrons pas cette année.

Exemples :

```
01 | -- 1. Afficher tous les produits :
02 | select * from Produit;
03 |
04 | -- 2. Afficher les libelles et les prix unitaires en mettant
          PrixUnitaire en entête de colonne :
05 | select Libelle, Pu PrixUnitaire from Produit;
06 |
07 | -- 3. Afficher le nom du client C015 :
08 | select Nomcli from Client
09 | where Cocli = 'C015';
10 |
11 | -- 4. Afficher les numeros de facture de l' annee 2014:
12 | select Nufact from Facture
13 | where Datefac >= '01-JAN-14' and Datefac <= '31-DEC-=14';
14 |
15 | -- 5. Afficher les numeros de toutes les factures qui contiennent le
          produit P03 en plus de 20 exemplaires :
16 | select Nufact from Facture
17 | where Copro = 'P03' and Qte > 20;
```

→ Les fonctions.

On les rencontre:

- • soit dans les expressions à afficher,
- • soit dans les opérandes des conditions. Il faut alors respecter la cohérence des différents opérandes.

On les classe selon le type du paramètre:

- • fonctions numériques:
  - – simples,
  - – sur un ensemble de valeurs;
- • fonctions sur les chaînes:
  - – fournissant une chaîne,
  - – fournissant une valeur numérique;
- • fonctions sur les dates.

La syntaxe et la disponibilité des fonctions dépendent du SGBDR. Certaines des fonctions présentées ci-dessous peuvent donc être absentes ou porter un autre nom. On présente ici des fonctions du SGBDR Oracle.

Les fonctions numériques

On peut citer:

- les fonctions logarithmiques et trigonométriques,
- les fonctions de transformation:
  - *abs(<n>)* Valeur absolue de n
  - *ceil(<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(<n>)* Signe (-1 si <0, 0 si =0, 1 si >0)
  - *sqrt(<n>)* Racine carrée de n (0 si n<0)
  - *trunc(<m>, <n>)* m tronqué à n décimales
- les fonctions sur un ensemble de valeurs: ces fonctions s'appliquent sur l'ensemble des valeurs fournies par une requête ou une sous-requête.
  - *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>)* Ecart-type de n (valeurs nulles non comptées)
  - *variance(<n>)* Variance de n (valeurs nulles non comptées)

  Le paramètre *[distinct]* permet de ne prendre en compte que les valeurs différentes.

  Ces fonctions, dites fonctions de groupe, s'appliquent sur des groupes de lignes.Elles ne peuvent pas être utilisées dans la clause *where* des requêtes.

Les fonctions sur les chaînes de caractères

On distingue:

- les fonctions retournant une chaîne:
  - *initcap(<c>)* La première lettre de chaque mot est mise en majuscule
  - *lower(<c>)* Conversion en minuscules
  - *upper(<c>)* Conversion en majuscules
  - *ltrim(<c>)* Suppression des espaces à gauche
  - *rtrim(<c>)* Suppression des espaces à droite
  - *replace(<c>,<c1>,<c2>)* Remplacement dans c de c1 par c2
  - *soundex(<c>)* Donne la représentation phonétique de la chaîne c permet de faire des recherches phonétiques
  - *substr(<c>,<d>,<l>)* Sous-chaîne extraite commençant au caractère de rang d et de longueur l
- une fonction retournant une valeur numérique:
  - *length(<c>)* Longueur de la chaîne

Les fonctions sur les dates

Parmi ces fonctions on peut citer:

- *add_month(<d>,<n>)* Ajouter n mois à la date d
- *last_day(<d>)* Dernier jour du mois d
- *month_between(<d1>,<d2>)* Nombre de mois entre les dates d1 et d2
- *new_time(...)* Date et heure dans un autre méridien
- *sysdate* Date et heure système
- *to_char(...)* Convertit une date en chaîne de caractères
- *to_date(...)* Convertit une chaîne en date (très nombreux paramètres)

Exemples

```
01 | --1. Combien y a-t-il de villes différentes dans la table Client ?
02 | select count(distinct Ville)
03 | from Client ;
04 |
05 | -- 2. Afficher les produits avec les prix majorés de 20\% arrondis à
          l' euro supérieur :
06 |
07 | select Copro, Libelle, ceil(Pu * 1.2)
08 | from Produit ;
09 |
10 | -- 3. Idem mais arrondi à l' euro inférieur :
11 | select Copro, Libelle, floor(Pu * 1.2)
12 | from Produit ;
13 |
14 | -- 4. Afficher les codes et les noms de clients dont le nom
          ressemble à 'Dupond' :
15 | select Cocli, Nomcli
16 | from Client
17 | where soundex(nomcli)= soundex('Dupond');
18 |
```

```
19 | -- 5. Afficher la valeur moyenne des factures de l'année 2014:
20 | select avg(Montant)
21 | from Facture
22 | where Datefac >= '01-JAN-14' and Datefac <= '31-DEC-=14';
23 |
24 | -- 6. Afficher toutes les caractéristiques de la plus grosse facture
25 | select *
26 | from Facture
27 | where Montant = (select max(Montant) from Facture);
```

→ Les tris

Les données des tables sont très rarement enregistrées dans l'ordre où on veut les voir afficher.

Tris implicites

La clause *distinct* (et *group by*) entraine le tri automatique de la table. Il est nécessaire de trier les enregistrements pour:

- permettre de détecter les doubles,
- générer des sous-ensembles.

Tri explicite pour affichage

...ORDER BY [asc/ desc]...

Le tri peut se faire sur une ou plusieurs colonnes ou expressions (16 au maximum en général). Il est possible de donner le numéro de la colonne (dans la liste des colonnes du) sur laquelle trier plutôt que son nom (utile lorsque la colonne est un champ calculé). Par défaut, le tri est en séquence croissante (*asc*).

Exemples

```
01 | -- 1. Trier les clients par ordre alphabétique :
02 | select Cocli, Nomcli, Ville
03 | from Client
04 | order by Nomcli ;
05 |
06 | -- Cette requête est équivalente :
07 | select Cocli, Nomcli, Ville
08 | from Client
09 | order by 2;
```

→ Ordre d'éxécution

Maintenant que nous avons détaillé toutes les clauses de l'instruction *select*, il est utile de donner l'ordre d'exécution des clauses par le SGBD (numéro entre []):

```
select [4]
from [1]
[ where [2] ]
[ order by [3] ]
```

1 Le *from* est exécuté en premier pour récupérer la table concernée par la sélection.
2 Ensuite, la ou les conditions du *where* sont exécutées et portent sur tous les tuples de la table.
3 Les tuples restants (non éliminés dans le *where*) sont ensuite triés grâce à la clause *order by*.
4 Enfin les résultats demandés dans la clause du *select* sont présentés à l'utilisateur.

4. Requêtes multi-tables

Il existe trois types de requêtes multi-tables:

- celles qui sont basées sur les opérateurs ensemblistes (*Union*, *Intersect*, *Minus*) qui sont hors programme;
- les requêtes imbriquées : utilisation de sous-requêtes dans les conditions *where* (ou *having*), qui sont hors programme aussi;
- celles qui utilisent des jointures que nous allons étudier en partie.

Lorsqu'une même table est utilisée deux fois dans la même requête, on peut lever l'ambiguïté en utilisant un synonyme local pour chacune.
Par exemple, si on a besoin deux fois de la table *Client* dans une requête, on utilise les synonymes *X* et *Y* pour chacune des deux tables (*Client X* et *Client Y*) et la sélection des champs de chacune des tables se fait en préfixant le nom du champ du synonyme de la table (par exemple *X.Cocli* et *Y.Cocli*).

→ Les jointures

Remarque :
« Les syntaxes données pour les jointures dans ce cours respectent la syntaxe ANSI (préconisée dans le programme de Terminale). Une autre forme de syntaxe existe (non-ANSI), que vous croiserez peut-être dans vos recherches. Cette dernière syntaxe exprime les jointures directement dans la clause *where* des requêtes, et a été historiquement utilisée par les SGBD.
Un exemple d'équivalence ANSI / non ANSI est donné plus loin. »

Les jointures permettent d'éviter les imbrications de requêtes : un seul bloc *select-from-where* est utilisé. On peut indiquer dans la clause *select* des colonnes résultats provenant de toutes les tables spécifiées dans la clause *from* .

Syntaxe

On s'intéresse uniquement aux équi-jointures (ou jointures naturelles) c'est à dire les requêtes qui ne renvoie que les informations pour lesquelles la jointure permet une correspondance entre une donnée de chacune des tables impliquées.

```
...
from <table1> [inner] join <table2>
on <table1>.<col> = <table2>.<col> ;
```

Remarque :

« Le mot-clé *inner* est optionnel. »

Pour comparaison, l'écriture non-ANSI équivalente est la suivante:

```
...
from <table1>,<table2>
where <table1>.<col> = <table2>.<col>;
```

La jointure peut aussi se faire sur une inégalité par l'intermédiaire de n'importe quel opérateur **>, <, >=, =<** (ou *BETWEEN, LIKE, IN* hors programme).

Exemples :

a) Deux tables distinctes

```
01 | -- 1. Nom du client correspondant à la facture n°3:
02 | select Nomcli from Client join Facture
03 | on Client.Cocli = Facture.Cocli
04 | where Nufact = 003;
05 |
01 | -- 2. Code et libellé des produits qui se trouvent
          dans la facture n°1:
02 | select Produit.Copro, Libelle
03 | from Produit join Detail
04 | on Produit.Copro = Detail.Copro
05 | where Facture.Nufact = 1;
06 |
07 | -- 3. Numéro et date des factures du client 'Emile':
08 | select Nufact, Datefact
09 | from Facture join Client
10 | on Facture.Cocli = Client.Cocli
11 | where Nomcli = 'Emile';
12 |
13 | -- 4. Nom des clients par ordre alphabétique
          avec le numéro et la date de facture :
14 | select Nomcli , Nufact
15 | from Facture join Client
16 | on Facture.Cocli = Client.Cocli
17 | order by Nomcli, Nufact ;
```

b) Deux fois la même table

```
01 | -- 7. Code et nom des clients qui habitent la même ville que le client 'Emile' (supposé unique) :
02 | select X.Cocli, X.Nomcli
03 | from Client X join Client Y
04 | on X.Ville = Y.Ville
05 | where Y.Nomcli = 'Emile'
06 | and X.NomCli != 'Emile ;
```

c) Plus de 2 tables

```
01 | -- 9. Code, libellé et prix des produits facturés le 3 juin 2014:
02 | select Produit.Copro, Libelle , Pu
03 | from Produit join Detail on Produit.Copro = Detail.Copro
04 | join Facture on Detail.Nufact = Facture.Nufact
05 | where Datefact = '03-JUN-14';
06 |
07 | -- Syntaxe non-ANSI équivalente
08 | select Produit.Copro, Libelle, Pu
09 | from Produit, Detail, Facture
10 | where Produit.Copro = Detail.Copro
11 | and Detail.Nufact = Facture.Nufact
12 | and Datefact = '03-JUN-14';
```