

LE LANGAGE SQL



PLAN



1. Définition
2. Requêtes simples SQL
3. Requêtes sur plusieurs tables
4. Mises-à-jour

1.Définition



- SQL (*Structured Query Language*)
 - Inventé à IBM San Jose, 1974 (Boyce & Chamberlin) pour les systèmes relationnels
 - Basé sur le calcul de tuples & Algèbre Relationnelle

| Année | Appellation |
|-------|------------------|
| 1986 | SQL-86 ou SQL-87 |
| 1989 | SQL-89 ou SQL-1 |
| 1992 | SQL-92 ou SQL-2 |
| 1999 | SQL-99 ou SQL-3 |
| 2003 | SQL:2003 |
| 2008 | SQL:2008 |

- Standard d'accès aux bases de données relationnelles

SQL : Trois langages



- Langage de manipulation de données (LMD /DML): permet de consulter ou de modifier le contenu de la base de donnée.
- Langage de définition de données (LDD/DDDL): permet de modifier la structure de la base de données.
- Langage de contrôle des données (LCD/DCL): permet de gérer les privilèges, ou les différents droits des utilisateurs sur la base de données.

Les instructions SQL



| LMD | LDD | LCD |
|--------|--------|--------|
| SELECT | CREATE | GRANT |
| INSERT | ALTER | REVOKE |
| DELETE | RENAME | |
| UPDATE | DROP | |

2. Requêtes simples SQL



SELECT



```
SELECT nomStation  
FROM Station  
WHERE region = 'Antilles'
```

- **SELECT** : la liste des attributs constituant le résultat.
- **FROM** : la (ou les) tables dans lesquelles on trouve les attributs utiles à la requête.
- **WHERE** : les conditions que doivent satisfaire les n-uplets de la base pour faire partie du résultat.

SELECT



● SELECT élémentaire : **SELECT** *liste_colonnes* **FROM** *liste_tables*

- Les expressions arithmétiques
- Les alias
- DISTINCT

● Exemple:

| NomStation | Libellé | Prix |
|------------|---------|------|
| Venusa | Voile | 150 |
| Venusa | Plongée | 120 |
| Farniente | Plongée | 130 |
| Passac | Ski | 200 |
| Passac | Piscine | 20 |
| Santalba | Kayac | 50 |

La table *Activité*

```
SELECT libelle, prix / 6.56, 'Cours de l'euro = ', 6.56
FROM Activite
WHERE nomStation = 'Santalba'
```

● Résultat:

| | | | |
|---------|-------------|----------------------|--------|
| libelle | prix / 6.56 | 'Cours de l'euro = ' | '6.56' |
| Kayac | 7.62 | 'Cours de l'euro = ' | 6.56 |

SELECT



● SELECT élémentaire

SELECT liste_colonnes **FROM** liste_tables

- Les expressions arithmétiques
- Les alias
- **DISTINCT**

● Exemple:

```
SELECT A.libelle, A.prix / 6.56 AS prixEnEuros, 'Cours de l'euro = ', 6.56
AS cours
FROM Activite A
WHERE A.nomStation = 'Santalba'
```

● Résultat :

| libelle | prixEnEuros | 'Cours de l'euro =' | cours |
|---------|-------------|---------------------|-------|
| Kayac | 7.69 | 'Cours de l'euro =' | 6.56 |

SELECT

● SELECT élémentaire

SELECT liste_colonnes **FROM** liste_tables

- Les expressions arithmétiques
- Les alias
- **DISTINCT**

● Exemple:

```
SELECT libelle  
FROM Activite
```

```
SELECT DISTINCT libelle  
FROM Activite
```

● Résultat :

| libelle |
|---------|
| Voile |
| Plongee |
| Plongee |
| Ski |
| Piscine |
| Kayac |

● Résultat :

| libelle |
|---------|
| Voile |
| Plongee |
| . |
| Ski |
| Piscine |
| Kayac |

Conditions plus complexes - *in* et *between* (1/2)



- Limiter les données

SELECT *liste_colonnes* **FROM** *liste_tables* [**WHERE** **condition(s)**]

- La clause WHERE limite l'interrogation aux lignes qui remplissent les conditions mentionnées.
- Les opérateurs : = , > , >= , < , <= , <>

- La condition **BETWEEN**

- Permet d'afficher les lignes en fonction d'une plage de valeurs

```
select NCLI  
from CLIENT  
where COMPTE between 1000 and 4000;
```

Conditions plus complexes - *in* et *between* (2/2)



- La condition **IN**
 - Vérifie l'appartenance d'une donnée à une liste de valeurs

```
select NCLI  
from CLIENT  
where CAT in ('C1','C2','C3');
```

```
select NCLI  
from CLIENT  
where LOCALITE not in ('Toulouse','Breda');
```

Conditions plus complexes - Les masques

- La condition **LIKE**
 - Recherche les chaînes de caractères valides à l'aide de caractères génériques
 - % représente n'importe quelle séquence de 0 ou plusieurs caractères

```
select NCLI
from CLIENT
where CAT like 'B_';
```

masques

'_' = un caractère quelconque

```
select NPRO
from PRODUIT
where LIBELLE like '%SAPIN%';
```

'%' = une chaîne quelconque

Un **masque** définit une famille de chaînes de caractères :

'B_' → 'B1'
 'Bd'
 'B '

'%SAPIN%' → 'PL. SAPIN 200x20x2'
 'Boite en SAPIN'
 'SAPIN VERNI'

'B_' → 'xB'
 'B'
 'B12'

'%SAPIN%' → 'Boite en Sapin'
 'Achetez S A P I N !'

Conditions plus complexes - les valeurs *null*

- Les conditions **NULL**
 - La vérification de la valeur NULL est effectuée par les opérateurs **IS NULL** et **IS NOT NULL**
 - Recherche des valeurs non attribuées

SELECT nom FROM employés WHERE ville **IS NULL;**

```
select NCLI
from CLIENT
where CAT = null;
```

*null ne peut être comparé à rien,
même pas à lui-même !*

| NCLI |
|------|
|------|

```
select NCLI
from CLIENT
where CAT is null;
```

| NCLI |
|------|
|------|

| |
|------|
| D063 |
| K729 |

```
select NCLI
from CLIENT
where CAT is not null;
```

- Le tri
 - La clause **ORDER BY** permet de trier les lignes

```
SELECT nom  
FROM employés WHERE salaire >= 1000  
ORDER BY salaire DESC
```

```
SELECT nom  
FROM employés WHERE salaire >= 1000  
ORDER BY salaire ASC
```


Les fonctions monolignes



- Ces fonctions renvoient un résultat par ligne, elles permettent de manipuler des éléments de données.
- Elles acceptent des paramètres et renvoient une seule valeur
- Elles peuvent être imbriquées
- On distingue des :

● Fonctions alphanumériques

- LPAD(exp, n, 'chaine') : Ajoute des caractères à gauche
- RPAD(exp, n, 'chaine') : Ajoute des caractères à droite
- TRIM(exp) : Retire les espaces avant et après
- REPLACE(exp, 'ch1', 'ch2') : Remplace ch1 par ch2

● Fonctions numériques

- ROUND(exp, n) : Arrondit à la décimale spécifiée
- TRUNCATE(exp, n) : Tronque à la décimale spécifiée
- MOD(exp1, exp2) : Renvoie le reste de la division de exp1 par exp2

● Fonction de dates

- CURDATE() : Renvoie la date système au format 'yyyy-mm-dd'
- CURTIME() : Renvoie l'heure système au format 'hh:mm:ss'
- DATE_FORMAT(date, format) : Renvoie la date selon le format indiqué
- YEAR(date)

● Fonction de conversions

Les fonctions multi-ligne



- L'objectif est de donner des informations statistiques sur un ensemble de lignes
- **AVG(exp)** : moyenne des valeurs de l'expression
- **COUNT(exp)** : nombre de valeurs non nulles
- **MAX(expr)** : Valeur maximale
- **MIN(expr)** : Valeur minimale

```
select 'Namur', avg (COMPTE) as Moyenne,  
       max (COMPTE) - min (COMPTE) as Ecart_max,  
       count (*) as Nombre  
from CLIENT  
where LOCALITE = 'Namur';
```

| Namur | Moyenne | Ecart_max | Nombre |
|-------|---------|-----------|--------|
| Namur | -2520 | 4580 | 4 |

le résultat ne comprend
qu'une seule ligne

Attention aux valeurs dupliquées

```
select count(NCLI)  
from  COMMANDE;
```

| count(NCLI) |
|-------------|
| 7 |

```
select distinct count(NCLI)  
from  COMMANDE;
```

| count(NCLI) |
|-------------|
| 7 |

```
select count(distinct NCLI)  
from  COMMANDE;
```

| count(NCLI) |
|-------------|
| 5 |

```
select count(NCLI) as Numeros,  
       count(NOM) as Noms,  
       count(LOCALITE) as Localites,  
       count(CAT) as Categories  
from CLIENT;
```

| Numeros | Noms | Localites | Categories |
|---------|------|-----------|------------|
| 16 | 16 | 16 | 16 |

```
select count(distinct NCLI) as Numeros,  
       count(distinct NOM) as Noms,  
       count(distinct LOCALITE) as Localites,  
       count(distinct CAT) as Categories  
from CLIENT;
```

| Numeros | Noms | Localites | Categories |
|---------|------|-----------|------------|
| 16 | 15 | 7 | 4 |

Attention aux ensembles vides

```
select count(*) as Nombre, sum(COMPTE) as Somme, max(CAT) as Max  
from CLIENT  
where LOCALITE = 'Alger';
```

| Nombre | Somme | Max |
|--------|--------|--------|
| 0 | <null> | <null> |

Groupement



- Création de groupes

```
SELECT idService, count(nomEmp) FROM Employes  
GROUP BY idService
```

- Exclure des groupes

La clause **HAVING** limite les groupes de la même manière que les lignes pour WHERE

```
SELECT idService, sum(salaire) FROM Employes  
GROUP BY idService  
HAVING sum(salaire) > 3000;
```

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

Employes

| NumEmp | NomEmp | Salaire | idservice |
|--------|--------|---------|-----------|
| 1 | B | 1000 | 1 |
| 2 | A | 1300 | 2 |
| 3 | C | 2000 | 2 |
| 4 | D | 1200 | 1 |

SELECT idService, **sum**(salaire) as somme FROM Employes
GROUP BY idService

| idService | somme |
|-----------|-------|
| 1 | 2200 |
| 2 | 3300 |

SELECT idService, **sum**(salaire) as somme FROM Employes
GROUP BY idService
HAVING **sum**(salaire) > 3000;

| idService | somme |
|-----------|-------|
| 2 | 3300 |



3. Requêtes sur plusieurs tables

Les jointures



- Les jointures permettent d'afficher des données de plusieurs tables
 - sans conditions de jointure le produit cartésien est effectué
SELECT nom, libelé FROM employés, services
- La condition de jointure s'écrit dans la clause WHERE
 - Le nom de table est nécessaire si le nom des colonnes est identique
SELECT nom, libel FROM employes, services
WHERE employes.numService = services.numService
- Alias de table
 - Utilisation de préfixes désignant les tables
 - Permet de simplifier les interrogations
 - Améliore les performances
 - Évite les ambiguïtés sur les noms de colonnes
SELECT E.nom, S.libel FROM employes E, services S
WHERE E.numService = S.numService

Sous interrogation



- Instruction SELECT imbriquée dans une clause d'une autre instruction SELECT
- Elle s'exécute en premier et son résultat est utilisé par la requête principale

**SELECT liste_colonnes FROM liste_tables
WHERE expr IN (SELECT)**

- Exemple:

- *Dans quelle station pratique-t-on une activité au même prix qu'à Santalba ?*

```
SELECT nomStation, libelle
FROM Activite
WHERE prix IN (SELECT prix FROM Activite
               WHERE nomStation = 'Santalba')
```

Exemples :

- *Où (station, lieu) ne peut-on pas faire du ski ?*

```
SELECT nomStation, lieu
FROM Station
WHERE nomStation NOT IN (SELECT nomStation
                           FROM Activite
                           WHERE libelle = 'Ski')
```

- *Quelle station pratique le tarif le plus élevé ?*

```
SELECT nomStation
FROM Station
WHERE tarif >= ALL (SELECT tarif FROM Station)
```

- *Quels sont les clients (nom, prénom) qui ont séjourné à Santalba?*

```
SELECT C.nom, C.prenom
FROM Client C
WHERE EXISTS (SELECT * FROM Sejour S
               WHERE S.station = 'Santalba'
               AND S.idClient = C.id)
```

Prédicats de dénombrement

Test de 'TOUS': Opérateur *ALL* ou 'AU MOINS UN' : Opérateur *ANY*



- L'opérateur *ALL* permet de tester par rapport à tous les éléments de la liste, l'opérateur *ANY* compare par rapport à l'un ou l'autre de ces éléments.

```
SELECT * FROM articles  
WHERE Famille != ALL ('100', '150', '200');
```

- Sélectionne toutes les lignes pour lesquelles Famille est différente de toutes les valeurs de la liste ('100', '150', '200').

```
SELECT * FROM articles  
WHERE Famille = ANY ('100', '150', '200');
```

- Sélectionne toutes les lignes pour lesquelles Famille est égale à l'une des valeurs de la liste.

Prédicats d'existence



- **Syntaxe:**

```
SELECT "nom de colonne 1"  
FROM "nom de table 1"  
WHERE EXISTS  
      (SELECT *  
       FROM "nom de table 2"  
       WHERE [condition])
```

EXISTS spécifie que le *SELECT* de niveau inférieur renvoie au moins une ligne

EXISTS teste simplement si la requête interne retourne une ligne. Si elle le fait, la requête externe peut s'exécuter. Sinon, la requête externe ne s'exécutera pas, et l'instruction SQL entière ne retournera aucun résultat.

IMPORTANT :

le prédicat **EXISTS** est en général plus rapide que le prédicat **IN** ;

Exemple 1:



Nous voulons obtenir le total du couchage de l'hôtel, toutes chambres confondues, à condition qu'il y ait au moins une chambre dotée d'un couchage pour au moins 3 personnes :

```
SELECT SUM(CHB_COUCHAGE) AS TOTAL_COUCHAGE
FROM   T_CHAMBRE
WHERE  EXISTS (SELECT *
                FROM   T_CHAMBRE
                WHERE  CHB_COUCHAGE >= 3)
```

| TOTAL_COUCHAGE |
|----------------|
| ----- |
| 54 |

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 );
```

Opérateurs logiques: Opérateurs *AND*, *OR* et *NOT*



- Opérateur *AND* : Les deux conditions unies par un ET logique doivent être remplies pour que le n-uplet soit sélectionné.
- Opérateur *OR* : L'une des deux conditions unies par un OU logique doit être remplie pour que le n-uplet soit sélectionné.
- Opérateur *NOT* : Cet opérateur inverse le résultat de la sélection.
- Ordre d'évaluation des opérateurs logiques :
 - Comparaisons
 - Opérateur NOT
 - Opérateur AND
 - Opérateur OR

```
select NOM, ADRESSE, COMPTE
from CLIENT
where LOCALITE = 'Toulouse' and COMPTE < 0;
```

```
select NOM, ADRESSE, COMPTE
from CLIENT
where COMPTE > 0
and (CAT = 'C1' or LOCALITE = 'Paris')
```


Union, intersection et différence



- Donnez tous les noms de région dans la base.

```
SELECT region FROM Station  
UNION  
SELECT region FROM Client
```

- Donnez les régions où l'on trouve à la fois des clients et des stations.

```
SELECT region FROM Station  
INTERSECT  
SELECT region FROM Client
```

- Quelles sont les régions où l'on trouve des stations mais pas des clients ?

```
SELECT region FROM Station  
EXCEPT  
SELECT region FROM Client
```

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 );
```

3.Mises-à-jour



Création d'une table



```
CREATE TABLE tblClient  
( idClient type_col1 [DEFAULT valeur1] [contrainte_col1],  
  nom_col2 type_col2 [DEFAULT valeur2] [contrainte_col2],  
  ...  
  contrainte_table1,  
  contrainte_table2,  
  ...)
```

- Contraintes

- Contraintes de colonnes et contraintes de tables matérialisent les contraintes d'intégrités posées sur le schéma relationnel et que le SGBD devra prendre en charge
- Contraintes de colonnes portent sur un attribut
- Contraintes de table portent sur plusieurs colonnes

Exemple de création d'une table et de ses colonnes

| CLIENT |
|---------------------|
| NCLI: char (10) |
| NOM: char (32) |
| ADRESSE: char (60) |
| LOCALITE: char (30) |
| CAT[0-1]: char (2) |
| COMPTE: num (9,2) |

```
CREATE TABLE CLIENT (
    NCLI    char(10),
    NOM     char(32),
    ADRESSE char(60),
    LOCALITE char(30),
    CAT     char(2),
    COMPTE  decimal(9,2) );
```

Insertion, Modification et Suppression



- Insertion d'un tuple

INSERT INTO nom_table [(nom_col1 [,nom_col2] ..)]
VALUES (valeur1[,valeur2] ...)

```
INSERT INTO Client (id, nom, prenom)  
VALUES (40, 'Moriarty', 'Dean')
```

- Modifications de tuples

UPDATE nom_table
SET nom_col1 = expression1 [, nom_col2 = expression2] ...
[WHERE condition]

```
UPDATE R SET A1=V1, A2=V2, ... An=Vn  
WHERE condition
```

- Suppression de tuples

DELETE FROM nom_table [WHERE condition];

- ✦ Destruction de tous les clients dont le nom commence par 'M'

```
DELETE FROM Client WHERE nom LIKE 'M%'
```