2019

SQL

PARTIE 2

JOACHIM ZADI

TABLE DES MATIERES

LE DML

LA COMMANDE SELECT

L'extraction ou sélection de données se base sur une requête. Une requête est une demande de données stockées dans la base de données. Le **SELECT** est la commande de base du SQL destinée à extraire des données d'une base ou calculer de nouvelles données à partir d'existantes... et les renvoie à l'utilisateur dans un ou plusieurs jeux de résultats.

Un jeu de résultats est une présentation, généralement sous forme de tableau, des données extraites par l'instruction SELECT. De même qu'une table SQL, le jeu de résultats se compose de colonnes et de lignes. La syntaxe complète de l'instruction SELECT est complexe, mais en voici les principales clauses.

```
SELECT [DISTINCT ou ALL] * ou liste de colonnes
FROM nom de table ou de la vue
[WHERE prédicats]
[GROUP BY ordre des groupes]
[HAVING condition]
[ORDER BY] liste de colonnes
```

NB *: dans cette syntaxe, les mots clef du SQL sont en gras, les paramètres en minuscule et entre crochets on trouve les parties optionnelles*

En fait l'ordre SQL SELECT est composé de 6 clauses dont 4 sont optionnelles.

Clauses de l'ordre SELECT :

SELECT	Spécification des colonnes du résultat
FROM	Spécification des tables sur lesquelles porte l'ordre
WHERE	Filtre portant sur les données (conditions à remplir pour que les lignes soient présentes dans le résultat)
GROUP BY	Définition d'un groupe (sous ensemble)
HAVING	Filtre portant sur les résultats (conditions de regroupement des lignes)
ORDER BY	Tri des données du résultat

NB : *La plupart du temps, la difficulté réside dans la compréhension de la différence entre le filtre **WHERE** et le filtre **HAVING**. De façon pragmatiquement, le filtre WHERE permet de filtrer les données des **tables** tandis que le filtre HAVING permet de filtrer les données du **résultat**...*

REMARQUE : pour spécifier une valeur littérale il faut l'entourer de guillemets simples.

Un premier exemple basique :

Exemple 1

```
SELECT    CLI_NOM, CLI_PRENOM
FROM      T_CLIENT
WHERE     TIT_CODE = 'M.'
```

```
CLI_NOM  CLI_PRENOM
-------  ----------
DUPONT   Alain
MARTIN   Marc
BOUVIER  Alain
DUBOIS   Paul
DREYFUS  Jean
FAURE    Alain
PAUL     Marcel
DUVAL    Arsène
PHILIPPE André
CHABAUD  Daniel
BAILLY   Jean-François
...
```

Permet de trouver les noms et prénoms des clients dont le titre est 'M.' (monsieur).

NB : *comme tous les paramètres à prendre sous forme de littéraux doivent être exprimés entourés d'apostrophes (simple côtes), il faut dédoubler un tel caractère s'il s'avère présent dans la chaîne utilisé.*

L'OPERATEUR * (ETOILE)

Le caractère * (étoile) récupère toutes les colonnes de la table précisée dans la clause FROM de la requête. Juste après le mot clef SELECT, on précise les colonnes de la table qui doivent être présentées dans la réponse. L'utilisation du caractère étoile ramène toutes les colonnes de la table dans la réponse. Dans le cas contraire il faut expressément nommer chacune des colonnes et les séparer par des virgules.

Exemple 2

```
SELECT    *
FROM      T_CLIENT
WHERE     TIT_CODE = 'M.'
```

```
CLI_ID   TIT_CODE  CLI_NOM   CLI_PRENOM  CLI_ENSEIGNE
-------  --------  --------  ----------  ------------
      1  M.        DUPONT    Alain       NULL
      2  M.        MARTIN    Marc        Transports    MARTIN & fils
      3  M.        BOUVIER   Alain       NULL
      4  M.        DUBOIS    Paul        NULL
      5  M.        DREYFUS   Jean        NULL
      6  M.        FAURE     Alain       Boulangerie du marché
     11  M.        PAUL      Marcel      Cie Internationale des Mach...
     12  M.        DUVAL     Arsène      NULL
     13  M.        PHILIPPE  André       NULL
     16  M.        CHABAUD   Daniel      NULL
...
```

NB : *Notons tout de suite la présence à plusieurs reprises du mot clef "NULL" dans la colonne CLI_ENSEIGNE. Non il ne s'agit pas d'une enseigne particulière, mais simplement de l'absence d'information. Nous verrons que l'absence d'information est le marquée "**NULL**", différent de la chaîne de caractère vide ("") ou encore du zéro(0).*

L'OPERATEUR DISTINCT (OU ALL)

Lorsque le moteur construit la réponse, il rapatrie toutes les lignes correspondantes, généralement dans l'ordre ou il les trouve, même si ces dernières sont en double, c'est à dire qu'il récupère toutes les lignes (ALL par défaut). C'est pourquoi il est souvent nécessaire d'utiliser le mot clef DISTINCT qui permet d'éliminer les doublons dans la réponse.

Exemple 3 :

```
SELECT    CLI_PRENOM
FROM      T_CLIENT
WHERE     TIT_CODE = 'M.'
```

```
CLI_PRENOM
----------
Alain
Marc
Alain
Paul
Jean
Alain
Marcel
Arsène
André
Daniel
...
```

```
SELECT distinct   CLI_PRENOM
FROM              T_CLIENT
WHERE             TIT_CODE = 'M.'
```

```
CLI_PRENOM
----------
Alain
Alexandre
André
Arnaud
Arsène
Bernard
Christian
Christophe
Daniel
Denis
...
```

L'OPERATEUR AS

Vous pouvez rajouter autant de colonnes que vous le désirez en utilisant le mot clef AS. En principe l'opérateur AS sert à donner un nom à de nouvelles colonnes créées par la requête. Il définit un **alias** (nom plus parlant) pour une colonne. Il peut être remplacé par un simple espace.

Exemple 4

```
SELECT    CLI_NOM as NOM, 'homme' as SEXE
FROM      T_CLIENT
WHERE     TIT_CODE = 'M.'
```

```
NOM       SEXE
-------   -----
DUPONT    homme
MARTIN    homme
BOUVIER   homme
DUBOIS    homme
DREYFUS   homme
FAURE     homme
PAUL      homme
DUVAL     homme
PHILIPPE  homme
CHABAUD   homme
...
```

```
SELECT    CLI_NOM NOM, 'homme' SEXE
FROM      T_CLIENT
WHERE     TIT_CODE = 'M.'
```

```
NOM       SEXE
-------   -----
DUPONT    homme
MARTIN    homme
BOUVIER   homme
DUBOIS    homme
DREYFUS   homme
FAURE     homme
PAUL      homme
DUVAL     homme
PHILIPPE  homme
CHABAUD   homme
...
```

OPERATEUR DE CONCATENATION

L'opérateur || (double barre verticale) permet de concaténer des champs de type **caractères**.

Exemple 5

```
SELECT    TIT_CODE || ' ' || CLI_PRENOM || ' ' ||  CLI_NOM as NOM
FROM      T_CLIENT
```

```
NOM
-----------------------
M. Alain DUPONT
M. Marc MARTIN
M. Alain BOUVIER
M. Paul DUBOIS
M. Jean DREYFUS
M. Alain FAURE
M. Paul LACOMBE
Melle. Evelyne DUHAMEL
Mme. Martine BOYER
M. Martin MARTIN
...
```

Néanmoins, dans certains SGBDR, on trouve le « **+** »comme opérateur de concaténation, ainsi que la fonction CONCAT.

OPERATEURS MATHEMATIQUES DE BASE

On peut utiliser les opérateurs mathématiques de base pour combiner différentes colonnes « + » , « - » , « * », « / »

Exemple

```
SELECT    CHB_ID, TRF_CHB_PRIX * 1.206 AS TARIF_TTC
FROM      TJ_TRF_CHB
WHERE     TRF_DATE_DEBUT = '01-01-2001'
```

```
CHB_ID  TARIF_TTC
------  ---------
     1      424,51
     2      482,40
     3      617,47
     4      424,51
     5      463,10
     6      482,40
     7      424,51
     8      540,29
     9      482,40
    10      617,47
...
```

LA CLAUSE FROM

La clause FROM permet de spécifier les tables d'où sont extraites les données. Elle est obligatoire dans toutes les instructions SELECT, sauf dans celles qui ne renvoient que des constantes dans certains SGBDR comme SQL Server.

Il est possible de renommer une table dans la clause FROM, dans ce cas, la syntaxe de la partie FROM de la commande SELECT est la suivante :

```
FROM nom_de_table ou nom_de_la_vue surnom
```

Nous verrons dans quel cas ce renommage est nécessaire ou obligatoire.

UTILISATION DU CARACTERE DOUBLE QUOTE (GUILLEMET)

Lorsqu'un nom d'un élément d'une base de données (**table**, **colonne** par exemple) est identique à un mot clef du SQL, il convient de l'entourer de guillemets (double quote). *En principe, les mots réservés du SQL sont déconseillés pour nommer des objets du modèle physique de données...*

Imaginons une table de nom JOIN, composée des champs suivants :

```
NOM      SELECT  DATE         NOT
-------  ------  -------      ---
DURAND   Oui     1999-11-12   F
DUVAL    Non     1998-01-17   M
```

Exemple

On désire sélectionner les colonnes SELECT et DATE lorsque la colonne NOT vaut F...

SELECT SELECT, DATE **FROM** JOIN **WHERE** NOT = 'F'	ERREUR !
SELECT "SELECT", "DATE" FROM "JOIN" WHERE "NOT" = 'F'	Correct : on entoure les mots clefs du SQL par des doubles côtes

Cela est aussi nécessaire lorsque le nom (d'une colonne ou d'une table) est composé de caractères particuliers tels que les blancs ou autres, ce qui est à éviter.

NB : *Les noms des identifiants d'objet de base de données doivent être écrits dans les jeux de caractères restreints suivant : [A…Z] + [a…z] + [0..9] + [_]. Ils ne doivent pas commencer par un chiffre et sont insensibles à la casse (indifférence entre majuscule et minuscule).*

LA CLAUSE ORDER BY

La clause ORDER BY permet de définir le tri des colonnes renvoyées dans une instruction SELECT, soit en précisant le nom littéral de la colonne, soit en précisant son n° d'ordre dans l'énumération qui suit le mot clef SELECT.

ASC spécifie l'ordre ascendant et DESC l'ordre descendant du tri. ASC ou DESC peut être omis, dans ce cas c'est l'ordre ascendant qui est utilisé par défaut.

```
ORDER BY colonne1 | 1 [ASC ou DESC] [, colonne2 | 2 [ASC ou DESC] ...
```

NB : *La valeur « NULL » est toujours considérée comme la plus petite des valeurs.*

Bien que la clause ORDER BY ne soit pas nécessaire, il est souvent utile de trier la réponse en fonction des colonnes. En revanche le temps de réponse s'en ressent souvent.

Pour spécifier l'ordre de tri, on doit placer les noms des colonnes séparées par des virgules, juste après le mot clef "ORDER BY", dans l'ordre voulu… On peut aussi utiliser le rang de chaque colonne dans l'ordre spécifié dans la clause SELECT.

Attention : *le tri est un tri interne, il ne faut donc placer dans cette clause que les noms des colonnes présentes dans la clause SELECT*.

Souvent, le fait de placer DISTINCT suffit, en général, à établir un tri puisque le moteur doit se livrer à une comparaison des lignes mais ce mécanisme n'est pas garanti car ce tri s'effectue dans un ordre non contrôlable qui peut varier d'un serveur à l'autre.

Exemple

```
SELECT      CLI_NOM, CLI_PRENOM
FROM        T_CLIENT
ORDER BY    CLI_NOM, CLI_PRENOM

ou

SELECT      CLI_NOM, CLI_PRENOM
FROM        T_CLIENT
ORDER BY    1, 2
```

```
CLI_NOM   CLI_PRENOM
--------  ----------
AIACH     Alexandre
ALBERT    Christian
AUZENAT   Michel
BACQUE    Michel
BAILLY    Jean-François
BAVEREL   Frédéric
BEAUNEE   Pierre
BENATTAR  Bernard
BENATTAR  Pierre
BENZAQUI  Joël
...
```

NB *: Un problème, qui n'est pas résolu, est de pouvoir choisir l'ordre des colonnes de la réponse. Sur certains serveurs cela peut être obtenu en plaçant les noms des colonnes à obtenir dans l'ordre où l'on veut les voir apparaître dans la clause SELECT, mais cette possibilité n'est jamais garantie...*

ATTENTION : *La clause ORDER BY est la dernière clause de tout ordre SQL et ne doit figurer qu'une seule fois dans le SELECT, même s'il existe des requêtes imbriquées ou un jeu de requêtes ensemblistes.*

LA CLAUSE WHERE

La clause WHERE permet de spécifier un critère de recherche que les lignes devront respecter pour être sélectionnées.

```
WHERE prédicats
```

Le prédicat doit contenir n'importe quelle expression logique renvoyant une valeur vraie. Ainsi, une requête aussi stupide que la suivante, est supposée fonctionner.

```
SELECT    CLI_NOM
FROM      T_CLIENT
WHERE     1=1
```

```
CLI_NOM
-------
DUPONT
MARTIN
BOUVIER
DUBOIS
DREYFUS
FAURE
LACOMBE
DUHAMEL
BOYER
MARTIN
...
```

Attention : La plupart des SGBDR ne comportent pas de colonne de type booléen. Une requête comme la suivante risque d'échouer.

```
SELECT    *
FROM      TJ_CHB_PLN_CLI
WHERE     CHB_PLN_CLI_OCCUPE
```

```
ERREUR !
bien que CHB_PLN_CLI_OCCUPE puisse être du booléen, la
plupart
des compilateurs SQL n'accepte pas ce test direct.
```

Pour pallier au manque de booléen, on utilise soit un littéral « True », « False », « Vrai », « Faux », « Oui », « Non » soit un numérique avec les valeurs « 0 ➔Faux » et « 1 ➔Vrai ». L'avantage des valeurs numériques est que le calcul logique est comparable aux divisions et additions...

```
SELECT    *
FROM      TJ_CHB_PLN_CLI
WHERE     CHB_PLN_CLI_OCCUPE = True
```

```
CORRECT ...
Mais sur certains compilateurs SQL il faut faire :
CHB_PLN_CLI_OCCUPE = 'True' (littéral).
Si le type booléen n'existe pas, alors il faut faire
CHB_PLN_CLI_OCCUPE = 1 si l'on a choisi de définir
les booléens comme INTEGER(1) avec 0 et 1
```

SIMILITUDE ENTRE LE « ET » ET LA MULTIPLICATION

opérateur ET	FAUX	VRAI
FAUX	FAUX	FAUX
VRAI	FAUX	VRAI

multiplication	0	1
0	0	0
1	0	1, <> 0

SIMILITUDE ENTRE LE OU ET L'ADDITION

opérateur OU	FAUX	VRAI
FAUX	FAUX	VRAI
VRAI	VRAI	VRAI

addition	0	1
0	0	1, <> 0
1	1, <> 0	2, <> 0

OPERATEURS DE COMPARAISON

Dans la clause WHERE, on dispose de différents opérateurs de comparaisons logiques :

```
WHERE valeur1 [NOT et] = ou < ou <= ou > ou >= ou <>valeur2 [OR ou AND ...]
```

Exemple

```
SELECT    CLI_NOM, CLI_PRENOM
FROM      T_CLIENT
WHERE     CLI_NOM >= 'A' AND CLI_NOM <'E'

ou
```

```
CLI_NOM   CLI_PRENOM
-------   ----------
DUPONT    Alain
BOUVIER   Alain
DUBOIS    Paul
DREYFUS   Jean
DUHAMEL   Evelyne
BOYER     Martine
```

```
SELECT    CLI_NOM, CLI_PRENOM
FROM      T_CLIENT
WHERE     (CLI_NOM >= 'A')
          AND
          (CLI_NOM < 'E')
```

```
DUVAL     Arsène
DAUMIER   Amélie
CHABAUD   Daniel
BAILLY    Jean-François
...
```

Ici on obtient tous les noms et prénoms des clients dont le nom commence par les lettres A, B, C ou D.

Attention : dans certains moteurs de requête SQL l'opérateur « différent de » (<>) s'écrit !=

OPERATEUR IN

L'opérateur IN permet de rechercher si une valeur se trouve dans un ensemble donné, quel que soit le type des valeurs de référence spécifiées (alpha, numérique, date…). Bien entendu, il est possible d'inverser le fonctionnement de l'opérateur IN en lui adjoignant l'opérateur NOT.

```
SELECT    TIT_CODE, CLI_NOM, CLI_PRENOM
FROM      T_CLIENT
WHERE     TIT_CODE IN ('Mme.', 'Melle.')
```

```
TIT_CODE  CLI_NOM     CLI_PRENOM
--------  ----------  ----------
Mme.      BOYER       Martine
Mme.      GALLACIER   Noëlle
Mme.      HESS       Lucette
Mme.      LETERRIER   Monique
Mme.      MARTINET    Carmen
Mme.      DAVID       Jacqueline
Mme.      MOURGUES    Jacqueline
Mme.      ZAMPIERO    Annick
Mme.      ROURE       Marie-Louise
Mme.      DE CONINCK  Patricia
...
```

On recherche les clients de sexe féminin, basés sur le code titre. Le contenu de la parenthèse peut être remplacé par le résultat d'une requête possédant une colonne unique. Dans ce cas on parle de requêtes imbriquées.

OPERATEUR BETWEEN

L'opérateur BETWEEN permet de rechercher si une valeur se trouve dans un intervalle donné, quel que soit le type des valeurs de référence spécifiées (alpha, numérique, date...).

Exemple

```
SELECT    CLI_NOM, CLI_PRENOM
FROM      T_CLIENT
WHERE     CLI_NOM BETWEEN 'A' AND 'E'
```

```
CLI_NOM   CLI_PRENOM
-------   ----------
DUPONT    Alain
BOUVIER   Alain
DUBOIS    Paul
DREYFUS   Jean
DUHAMEL   Evelyne
BOYER     Martine
DUVAL     Arsène
DAUMIER   Amélie
CHABAUD   Daniel
BAILLY    Jean-François
...
```

NB : Les opérateurs IN et BETWEEN sont très pratiques dans le cas où l'on désire effectuer des requêtes où l'utilisateur peut saisir une liste de choix multiples (IN) ou une plage de valeur (BETWEEN).

OPERATEUR LIKE

L'opérateur LIKE permet d'effectuer une comparaison partielle. Il est surtout employé avec les colonnes contenant des données de type *alpha*. Il utilise les jokers % et _ (‘pour cent' et ‘blanc souligné'). Le joker % remplace n'importe quelle chaîne de caractères, y compris la chaîne vide. Le blanc souligné remplace un et un seul caractère.

Exemple

```
SELECT    CLI_NOM, CLI_PRENOM
FROM      T_CLIENT
WHERE     CLI_NOM LIKE 'B%'
```

```
CLI_NOM  CLI_PRENOM
-------  ----------
BOUVIER  Alain
BOYER    Martine
BAILLY   Jean-François
BOUCHET  Michel
BEAUNEE  Pierre
BERGER   Jean-Pierre
BOURA    André
BENZAQUI Joël
BAVEREL  Frédéric
BERTRAND Christophe
...
```

On recherche les clients dont le nom commence par B. Mais si vos données sont susceptibles de contenir un des deux caractères joker, alors il faut recourir à une séquence d'échappement, à l'aide du mot clef ESCAPE...

Cherchons les clients dont l'enseigne contient au moins un caractère blanc souligné :

```
SELECT    *
FROM      T_CLIENT
WHERE     CLI_ENSEIGNE
          LIKE  '%_%'
```

```
CLI_ID TIT_CODE CLI_NOM    CLI_PRENOM    CLI_ENSEIGNE
------ -------- ---------- ------------- ------------------------------
     2 M.       MARTIN     Marc          Transports MARTIN & fils
     6 M.       FAURE      Alain         Boulangerie du marché
    10 M.       MARTIN     Martin        HERMAREX IMPORT_EXPORT
    11 M.       PAUL       Marcel        Cie Internationale des Mach...
    17 M.       BAILLY     Jean-François Entreprise DUPONT CHAUFFAGE
    24 M.       CHTCHEPINE Dominique     HOTEL *** DE LA GARE
    26 M.       GARREAU    Paul          IBM Corp.
    34 Mme.     GALLACIER  Noëlle        Transports GALLACIER
    42 Mme.     LETERRIER  Monique       SA AROMAX ENTREVONT
    49 M.       COULOMB    Renaud        Cabinet COULOMN et CALEMANT
```

```
SELECT    *
FROM      T_CLIENT
WHERE     CLI_ENSEIGNE
          LIKE  '%#_%' ESCAPE '#'
```

```
CLI_ID TIT_CODE CLI_NOM    CLI_PRENOM    CLI_ENSEIGNE
------ -------- ---------- ------------- ------------------------------
    10 M.       MARTIN     Martin        HERMAREX IMPORT_EXPORT
```

Pour traiter ce cas, on définit « # » comme caractère d'échappement. Le caractère qui suit ce caractère d'échappement est donc interprété comme un caractère et non comme un joker.

NOTA : *l'opérateur LIKE effectue une recherche en tenant compte de la différence entre lettres majuscules et minuscules. Si vous voulez effectuer une recherche en ne tenant aucunement compte de la différence entre majuscules et minuscules, il convient d'utiliser les opérateurs LOWER et UPPER (voir ci-dessous). Mais la plupart du temps, l'utilisation du LIKE dans un SGBDR donné ignore la casse.*

RESUME DES OPERATEURS POUR LES PREDICATS DE LA CLAUSE WHERE

Voici un tableau résumant les principaux opérateurs utilisés pour la construction des prédicats :

opérateurs de comparaisons	= <> < <= > >=
connecteurs logiques	{OR | AND}
opérateur de négation	NOT
parenthèses	(...)
opérateurs mathématiques	+ - * /
comparaison logique	IS [NOT] {TRUE | FALSE | UNKNOWN}
comparaison avec valeur	IS [NOT] NULL
intervalle	valeur BETWEEN borne_basse AND borne_haute
comparaison partielle de chaîne de caractères	valeur LIKE motif [ESCAPE echappement]
comparaison à une liste de valeur	valeur [NOT] IN (liste)

FONCTIONS DIVERSES

TRANSTYPAGE A L'AIDE DE LA FONCTION CAST

Il permet de changer le type de données d'une colonne afin d'effectuer une comparaison de données de type hétérogène par exemple entre un champ contenant des données numériques et un champ contenant des données de type chaîne de caractères...

Sa syntaxe est CAST (colonne AS nouveau type).

```
SELECT CHB_ID, CHB_NUMERO, CHB_POSTE_TEL
FROM T_CHAMBRE
WHERE CAST (CHB_POSTE_TEL AS INTEGER) / 10 > CHB_NUMERO
```

```
ADR_VILLE             ADR_CP  ADR_CP + 1
------------          ------  ----------
VERSAILLES            78000        78001
MONTMAIZIN            11254        11255
PARIS                 75015        75016
VERGNOLLES CEDEX 452  84524        84525
MARSEILLE             13002        13003
PARIS                 75012        75013
...
```

L'opérateur CAST permet de transtyper les valeurs contenues dans une colonne. Bien entendu il faut qu'un type de donnée puisse être converti dans un autre type (compatibilité de types) afin que la réponse ne soit pas entachée d'erreurs ou d'omissions.

MISE EN MAJUSCULE / MINUSCULE

Les opérateurs LOWER et UPPER permettent de mettre en majuscule ou en minuscule des chaînes de caractères dans les requêtes.

```
SELECT upper(CLI_PRENOM), lower(CLI_NOM)
FROM T_CLIENT
```

```
CLI_NOM  CLI_PRENOM
-------  ----------
ALAIN    dupont
MARC     martin
ALAIN    bouvier
PAUL     dubois
JEAN     dreyfus
ALAIN    faure
PAUL     lacombe
EVELYNE  duhamel
MARTINE  boyer
MARTIN   martin
...
```

NB : pour effectuer une recherche en ne tenant aucunement compte de la différence entre majuscules et minuscules, il faut utiliser l'opérateur UPPER (ou LOWER mais attention à la transformation des accents !) :

```
SELECT *
FROM T_CLIENT
where upper(CLI_PRENOM) = upper(CLI_NOM)
```

```
CLI_ID  TIT_CODE  CLI_NOM  CLI_PRENOM  CLI_ENSEIGNE
------  --------  -------  ----------  ----------------------
    10  M.        MARTIN   Martin      HERMAREX IMPORT_EXPORT
```

NB : certains SGBDR permettent de paramétrer l'activation de la recherche systématique des chaînes de caractères sans tenir compte de la casse. Sur d'autres, le paramétrage permet de confondre les lettres accentuées ou non...

SUPPRIMER LES BLANCS (OU TOUT AUTRE CARACTERES)

La fonction TRIM permet de supprimer en tête ou en queue (ou les deux), le blanc ou tout autre caractère spécifié.

```
TRIM ([LEADING ou TRAILING ou BOTH] [caractère] FROM nom de colonne)
```

- **LEADING** : suppression en tête
- **TRAILING** : suppression en queue
- **BOTH** : suppression en tête et en queue

Dans notre table téléphone, nous voulons supprimer le zéro débutants les numéros afin de pouvoir les communiquer aux étrangers qui n'ont pas besoin de composer ce chiffre (*ils doivent simplement composer le 00 33 suivi du numéro à 9 chiffres*).

```
SELECT    TEL_NUMERO,
          '00~33 ' || TRIM(LEADING '0' FROM TEL_NUMERO)
          AS TEL_INTERNATIONAL
FROM      T_TELEPHONE
```

```
TEL_NUMERO      TEL_INTERNATIONAL
--------------  -----------------
01-45-42-56-63  00~33 1-45-42-56-63
01-44-28-52-52  00~33 1-44-28-52-52
01-44-28-52-50  00~33 1-44-28-52-50
...
```

NB : certains serveurs SQL proposent différentes fonctions comme LTRIM et RTRIM pour une suppression des blancs en tête ou en queue.

EXTRAIRE UNE SOUS CHAINE

La fonction SUBSTRING permet d'extraire une sous chaîne d'une chaîne de caractère. Elle a besoin de l'ordre du premier caractère et du nombre de caractères sur lequel elle doit opérer.

```
SUBSTRING (nom de colonne FROM n TO m)
```

Extrait la sous chaîne de nom de colonne en commençant à n sur m caractères.

```
SELECT   CLI_NOM, CLI_PRENOM,
         SUBSTRING(CLI_PRENOM FROM 1 FOR 1) ||
         SUBSTRING(CLI_NOM FROM 1 FOR 1)
         AS INITIALES
FROM     T_CLIENT
```

```
CLI_NOM  CLI_PRENOM  INITIALES
-------  ----------  ---------
DUPONT   Alain       AD
MARTIN   Marc        MM
BOUVIER  Alain       AB
DUBOIS   Paul        PD
DREYFUS  Jean        JD
FAURE    Alain       AF
LACOMBE  Paul        PL
DUHAMEL  Evelyne     ED
BOYER    Martine     MB
MARTIN   Martin      MM
...
```

Cet exemple construit les initiales des clients à partir des colonnes CLI_NOM et CLI_PRENOM_CLI.

ATTENTION, certains SGBDR utilisent la fonction **SUBSTR** (ORACLE)

OPERATEUR DE TRAITEMENT DES DATES

EXTRAIRE UN PARAMETRE TEMPOREL D'UNE DATE

L'opérateur EXTRACT permet d'extraire depuis une date, le jour le mois ou l'année…

```
EXTRACT (YEAR ou MONTH ou DAY FROM nom de colonne)
```

Dans la table des réservations on recherche l'identifiant des chambres ayant été réservées au cours du mois de mai de n'importe quelle année et pour 3 personnes.

```
SELECT distinct CHB_ID
FROM  TJ_CHB_PLN_CLI
WHERE EXTRACT(MONTH FROM PLN_JOUR) = 5
  AND CHB_PLN_CLI_RESERVE = 1
  AND CHB_PLN_CLI_NB_PERS = 3
```

```
CHB_ID
------
     1
     5
     6
     8
    11
    12
    16
    17
    18
    20
```

NB : il est dommage de constater que la fonction EXTRACT du standard SQL, souvent fort utile, est rarement présente dans les moteurs de bases de données. Ni Access, ni Oracle, ni Sybase, ni SQL Server n'en sont dotés. Seul le

middleware BDE de Borland INPRISE Corel permet d'exploiter pleinement cette fonction avec les SGBDR PARADOX, DBASE, FoxPro, INTERBASE, MSSQL, Sybase, INFORMIX, DB2, Oracle. Cependant il est courant de trouver des fonctions s'en approchant : Exemple DATEPART dans SQL Server.

HEURE ET DATE COURANTE

L'heure courante, la date courante et le combiné date/heure courant peuvent être obtenu à l'aide des fonctions CURRENT_DATE, CURRENT_TIME et CURRENT_TIMESTAMP.

```
SELECT distinct CHB_ID
FROM TJ_CHB_PLN_CLI
WHERE (CHB_PLN_CLI_RESERVE = 1)
  AND PLN_JOUR BETWEEN CURRENT_DATE and CURRENT_DATE + 14
  AND CHB_PLN_CLI_NB_PERS = 3

attention, le résultat de cette requête varie en fonction de la date
à laquelle vous l'exécutez !
```

```
CHB_ID
------
     …
```

Cette requête renvoie les chambres réservées pour 3 personnes entre la date du jour et pour les deux semaines à venir.

Attention : la plupart des SGBDR n'acceptent pas encore cette version normalisée des fonctions de recherche de temps courant. Voici les fonctions spécifiques aux différents serveurs SQL :

Oracle	SYSDATE()
Sybase	GETDATE()
SQL Server	GETDATE()
Access	NOW()
MySQL	NOW()
Paradox (QBE)	TODAY

OPERATEURS STATISTIQUES

Il est possible de réaliser des comptages statistiques sur les colonnes, à l'aide des opérateurs AVG (moyenne), MAX (maximum), MIN (minimum), SUM (total), COUNT (nombre). On les appelle aussi fonctions d'agrégations.

```
SELECT AVG(TRF_CHB_PRIX) as MOYENNE,
   MAX(TRF_CHB_PRIX) as MAXI,
   MIN(TRF_CHB_PRIX) as MINI,
   SUM(TRF_CHB_PRIX) as TOTAL,
   COUNT(TRF_CHB_PRIX) as NOMBRE
 FROM TJ_TRF_CHB
 WHERE TRF_DATE_DEBUT = '2001-01-01'
```

```
MOYENNE   MAXI      MINI      TOTAL       NOMBRE
--------  --------  --------  ----------  ------
406,74 F  512,00 F  352,00 F  7 728,00 F      19
```

Cette requête calcule la moyenne, le montant maximum, minimum, la totalisation et le nombre des tarifs de chambre pour la date de début du premier janvier 2001.

On peut s'étonner que les opérateurs statistiques VARIANCE ou STDDEV (écart type) soient rarement présents dans les SGBDR car il s'agit de fonctions statistiques qui possèdent une grande utilité. Mais la norme SQL 92 ne les a pas retenu. Cependant, ils existent notamment dans **Oracle**.

ATTENTION : nous verrons que l'utilisation des fonctions statistiques nécessite la plupart du temps la mise en place d'une clause de groupage, afin de déterminé quel est le sous ensemble cible d'agrégation pour les calculs.

AUTRES FONCTIONS NORMALISEES

BIT_LENGTH	Taille d'une colonne de type BIT ou BIT VARYING (nombre de bits)
CHAR_LENGTH	Taille d'une colonne de type caractère (nombre de caractères)
OCTET_LENGTH	Taille d'une colonne de type caractère (nombre d'octets)
CURRENT_DATE	Date en cours
CURRENT_TIME	Heure en cours
CURRENT_TIMESTAMP	Date et heure en cours
CONVERT	Conversion paramétrée d'une chaîne de caractères
POSITION	Position d'une chaîne de caractères dans une sous chaîne
TRANSLATE	Traduction d'une chaîne de caractères dans un format spécifié

AUTRES OPERATEURS MATHEMATIQUES (NON NORMALISES)

Les opérateurs ci-dessous peuvent être implémentés dans différents moteurs.

ABS	valeur absolue
MOD	modulo
SIGN	signe
SQRT	racine carrée
CEIL	plus petit entier
FLOOR	plus grand entier

ROUND	arrondi
TRUNC	tronqué
EXP	exponentielle
LN	logarithme népérien
LOG	logarithme décimal
POWER	puissance
COS	cosinus
COSH	cosinus hyperbolique
SIN	sinus
SINH	sinus hyperbolique
TAN	tangente
TANH	tangente hyperbolique
PI	constante Pi

Certains sont rarement implémentés du fait que les SGBDR sont axés sur l'informatique de gestion, la collecte et le traitement d'informations et non le calcul mathématique.

NB : le nom de certains de ces opérateurs peut différer d'un SGBDR à l'autre.

TRAITEMENT DES "VALEURS" NULLES

Le **NULL** n'est pas à proprement parler une valeur, mais bien l'absence de valeur, c'est pourquoi nous parlerons de marqueur NULL et non de valeur NULL.

Le marqueur NULL pose une quantité de problèmes et nous allons dans ce paragraphe soulever un coin du voile, que nous traiterons un peu plus tard dans le cas général de la recherche des occurrences d'inexistence.

LE NULL N'EST NI LA CHAINE VIDE, NI LE ZERO

NULL n'est pas une valeur. C'est un marqueur. Par conséquent le marqueur NULL ne peut jamais être comparé a une valeur. Recherchons par exemple les clients qui n'ont pas d'enseigne :

```
SELECT CLI_ID, CLI_NOM
FROM T_CLIENT
WHERE CLI_ENSEIGNE = ''
```

```
CLI_ID  CLI_NOM
-------  --------
...
```

La réponse doit produire une table vide !

Pour contourner ce problème il faut, soit penser à enregistrer une chaîne de caractère vide lors de l'insertion des données dans la table, soit la clause WHERE avec un opérateur spécialisé dans le traitement des valeurs nulles.

Il y a donc un véritable dilemme à utiliser des requêtes en se basant sur des critères d'absence de valeur et il faut toujours faire très attention aux clauses qui utilisent des références aux valeurs nulles, suivant ce que l'on veut obtenir, d'autant plus que les NULL se propagent dans les calculs.

Voici un extrait de la table T_LIGNE_FACTURE

```
LIF_ID    FAC_ID    LIF_QTE    LIF_REMISE_POURCENT  LIF_REMISE_MONTANT  LIF_MONTANT  LIF_TAUX_TVA
--------  --------  ---------  -------------------  ------------------  -----------  ------------
       1         1       1,00                15,00                NULL     320,00 F         18,60
       2         3       1,00                 NULL             50,00 F     250,00 F         18,60
       3         3       1,00                 NULL             50,00 F     320,00 F         18,60
       4         3       1,00                 NULL             50,00 F     240,00 F         18,60
       5         5       1,00                 NULL                NULL     320,00 F         18,60
       6         5       1,00                 NULL                NULL     220,00 F         18,60
       7         7       1,00                 NULL                NULL     220,00 F         18,60
       8         7       1,00                 NULL                NULL     250,00 F         18,60
       9         7       1,00                 NULL                NULL     320,00 F         18,60
      10         7       1,00                 NULL                NULL     270,00 F         18,60
...
```

Nous voulons calculer le montant total de chacune des lignes de cette table, pour une facture donnée.

La requête pour FAC_ID = 3 est la suivante :

```
SELECT FAC_ID,
       sum (LIF_QTE * (LIF_MONTANT - LIF_REMISE_MONTANT) * (1 - LIF_REMISE_POURCENT / 100))
AS TOTAL_FAC,
       sum((LIF_QTE * (LIF_MONTANT - LIF_REMISE_MONTANT) * (1 - LIF_REMISE_POURCENT / 100))
           * (LIF_TAUX_TVA / (100 +   LIF_TAUX_TVA))) AS TOTAL_TAXES
FROM T_LIGNE_FACTURE
WHERE FAC_ID = 3
GROUP BY FAC_ID
```

```
FAC_ID  TOTAL_FAC  TOTAL_TAXES
------  ---------  -----------
     3       NULL         NULL
```

On constate que pour les lignes qui n'ont pas de valeurs renseignées dans les colonnes LIF_REMISE_POURCENT, LIF_REMISE_MONTANT, le résultat du calcul donne la valeur « **null** » qui se traduit à l'affichage par... rien !

NB : En général, pour se sortir de ce mauvais pas, on peut, lors de la création de la base de données, obliger tous les champs de type numérique (réels ou entiers) a ne pas accepter la valeur nulle et prendre par défaut la valeur zéro...

Attention : L'arithmétique des nuls est assez particulière... Souvenez-vous toujours que les NULL se propagent. Cela est vrai pour les numériques, les dates mais aussi pour les chaînes de caractères. Ainsi SQL opère une distinction entre une chaîne de caractère vide et un champ non renseigné. Dans le cas de la concaténation d'une colonne NULL et d'une colonne proprement renseigné, la valeur renvoyée sera **NULL** !!!

OPERATEURS DE TRAITEMENT DES MARQUEURS NULL

La norme SQL 2 (1992) spécifie une comparaison et différents opérateurs sur les marqueurs NULL :

- **IS NULL / IS NOT NULL** : teste si la colonne est vide ou non vide.
- **COALESCE** qui recherche la première valeur non vide dans un ensemble
- **NULLIF** NULLifie une colonne en fonction d'une valeur donnée

```
COALESCE ( valeur1, valeur2 [, valeur3] ... )
NULLIF ( nom_de_colonne, valeur)
expression IS [NOT] NULL
```

NB : *ISNULL (en un seul mot) est une autre fonction de branchement que l'on rencontre parfois (renvoi une valeur si la valeur est nulle). Dans la même veine, NVL ou VALUE sont des expressions équivalentes à COALESCE que l'on rencontre sur certains SGBDR.*

La requête précédente s'exprime, à l'aide de l'opérateur ISNULL :

```
SELECT FAC_ID,
        sum (LIF_QTE * (LIF_MONTANT - ISNULL(LIF_REMISE_MONTANT, 0))
            * (1 - ISNULL(LIF_REMISE_POURCENT, 0) / 100)) AS TOTAL_FAC,
        sum((LIF_QTE * (LIF_MONTANT - ISNULL(LIF_REMISE_MONTANT, 0))
            * (1 - ISNULL(LIF_REMISE_POURCENT, 0) / 100))
            * (LIF_TAUX_TVA / (100 +   LIF_TAUX_TVA))) AS TOTAL_TAXES
FROM T_LIGNE_FACTURE
WHERE FAC_ID = 3
GROUP BY FAC_ID
```

```
FAC_ID  TOTAL_FAC  TOTAL_TAXES
------  ---------  -----------
     3      810.0       127.03
```

NB : *En règle générale, dès que l'on traite des colonnes contenant des valeurs monétaires ou numériques, il est bon de faire en sorte que la colonne soit obligatoire et que par défaut elle soit renseignée à zéro. Sinon, il faudra faire un usage systématique des fonctions NULLIF ou COALESCE dans tous les calculs et cela grèvera les performances d'exécution !*

NEGATION DE VALEURS

C'est l'opérateur NOT qui réalise la négation de valeurs et inverse la valeur logique d'un prédicat. L'opérateur NOT peut être combiné avec la plupart des opérateurs de comparaison. Mais il devient très intéressant lorsqu'il est combiné aux opérateurs IN, BETWEEN, LIKE et NULL.

Recherchons par exemple toutes les chambres permettant de recevoir au moins 3 personnes, ne comportant pas le chiffre 4 (chiffre de la mort au japon) ni les chambres portant le n° 7 et 13 pour un client particulièrement superstitieux...

```sql
SELECT CHB_ID, CHB_NUMERO, CHB_COUCHAGE
FROM   T_CHAMBRE
WHERE  NOT (CAST(CHB_NUMERO AS VARCHAR(10))
LIKE '%4%')
       AND CHB_NUMERO NOT IN ('7', '13')
       AND CHB_COUCHAGE >= 3
```

```
CHB_ID      CHB_NUMERO CHB_COUCHAGE
----------- ---------- ------------
1           1          3
5           5          3
6           6          5
8           8          3
11          11         3
12          12         3
15          16         3
16          17         5
17          18         3
19          20         3
```

Nous verrons que le NOT IN est particulièrement précieux dans les requêtes imbriquées, c'est à dire les requêtes multi tables.

Nous voulons maintenant le nom des clients qui ne commence pas par 'DU' :

```sql
SELECT CLI_NOM
FROM T_CLIENT
WHERE CLI_NOM NOT LIKE 'DU%'
```

```
--------------------------------
MARTIN
BOUVIER
DREYFUS
FAURE
LACOMBE
BOYER
MARTIN
PAUL
PHILIPPE
PIERRELAYE
...
```

LES BRANCHEMENTS DANS LE SQL

SQL possède un branchement à la manière des IF et autres structures de test des langages procéduraux. Mais il convient de ne l'utiliser qu'à bon escient, c'est à dire aussi peu souvent que possible, beaucoup de cas pouvant être traités soit par le COALESCE soit par des requêtes avec des opérations ensemblistes de type UNION. En effet les performances se dégradent très vite lors de l'usage du CASE à cause de l'impossibilité d'effectuer des traitements par "paquets".

La structure CASE du SQL comprend deux syntaxes différentes. Le CASE pour branchement sur les valeurs d'une expression et le CASE généralisé.

CASE SUR EXPRESSION

Dans ce cas, la syntaxe est la suivante :

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

```
SELECT CHB_NUMERO, CASE CHB_ETAGE
                       WHEN 'RDC' THEN 0
                       WHEN '1er' THEN 1
                       WHEN '2e'  THEN 2
                   END AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
ORDER  BY  ETAGE, CHB_COUCHAGE
```

```
CHB_NUMERO ETAGE       CHB_COUCHAGE
---------- ----------- ------------
2          0           2
3          0           2
4          0           2
1          0           3
7          1           2
9          1           2
10         1           2
5          1           3
8          1           3
11         1           3
12         1           3
6          1           5
14         2           2
15         2           2
19         2           2
21         2           2
16         2           3
18         2           3
20         2           3
17         2           5
```

CASE GENERALISE

L'expression disparait au profit de différents prédicats.

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

```
SELECT CHB_NUMERO, CASE WHEN CHB_ETAGE = 'RDC' THEN 0
                        WHEN CHB_ETAGE = '1er' THEN 1
                        WHEN CHB_ETAGE = '2e'  THEN 2
                   END AS ETAGE, CHB_COUCHAGE
FROM   T_CHAMBRE
ORDER  BY  ETAGE, CHB_COUCHAGE
```

```
CHB_NUMERO ETAGE       CHB_COUCHAGE
---------- ----------- ------------
2          0           2
3          0           2
4          0           2
1          0           3
7          1           2
9          1           2
10         1           2
5          1           3
8          1           3
11         1           3
12         1           3
6          1           5
14         2           2
15         2           2
19         2           2
21         2           2
16         2           3
18         2           3
20         2           3
17         2           5
```

Qui donne le même résultat !

ATTENTION : Tous les SGBDR ne supportent pas ces deux syntaxes.

NOTA : dans les deux cas il est possible de remplacer l'IF d'un langage procédural :

`CASE WHEN condition1 THEN expression1`
`    [ELSE expression_défaut]`
`END`
	`CASE expression`
	`   WHEN valeur1 THEN expression1`
	`  [ELSE expression_défaut]`
	`END`