INFORMATIQUE

Laboratoires de bases de données

Laboratoire n°2
Sélection simple

par Danièle BAYERS et Louis SWINNEN
Révision 2024 : Vincent Reip

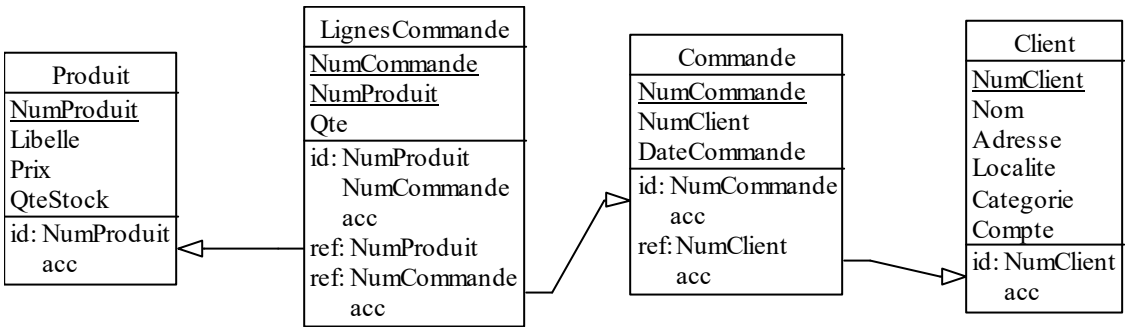
Octobre 2024

1. Objectif

L'objectif de ce laboratoire est de montrer, par quelques exemples pratiques, la **sélection** dans une table. Ensuite, nous aborderons les **fonctions importantes** pour la gestion des chaînes de caractères et des dates en Oracle et SQL Server.

2. La base de données

Dans les exemples, nous supposerons que nous disposons de la base de données suivante :



3. Sélection sur une seule table

En SQL, il est possible d'écrire des requêtes qui vont **sélectionner** un ou plusieurs enregistrements dans la base de données. La forme des requêtes de sélection est la suivante :

```
SELECT [DISTINCT] attributs|*
FROM table
[WHERE condition]
[ORDER BY attributs]
```

L'option `DISTINCT` permet de supprimer les lignes identiques dans le résultat.

Les *attributs* sont les colonnes pour lesquelles on souhaite obtenir une valeur. Si tous les attributs sont nécessaires, il est possible de mentionner un astérisque (qui est un raccourci pour tous les attributs).

Nous verrons dans la partie programmation qu'il vaut toujours mieux nommer les attributs que l'on souhaite dans le résultat. En effet, en cas d'évolution de la base de données (ajout d'une colonne par exemple), avoir des requêtes du type `SELECT *` peut entraîner des effets non désirés dans votre application car elle obtiendrait dans son résultat un nouvel attribut pas nécessairement attendu.

La *table* désigne la relation dans laquelle les données souhaitées sont présentes.

La *condition* permet de sélectionner les enregistrements voulus. Cette condition peut porter sur la clé primaire (et donc, s'il s'agit d'une valeur à chercher, de trouver un enregistrement) ou sur n'importe quel autre attribut de la table considérée. Nous analyserons dans la suite les différentes possibilités pour les *conditions*.

Exemples :

```
SELECT * FROM client
```
Dans cet exemple, on demande tous les attributs de tous les clients enregistrés dans la base de données.

```
SELECT numProduit, libelle, prix
FROM produit
WHERE libelle LIKE '%digital%'
```
Dans cet exemple, on demande les produits dont le libellé contient le mot digital. Seuls les attributs *numProduit*, *libelle* et *prix* intègrent le résultat.

3.1. Condition par comparaison simple

Dans la clause `WHERE`, la condition la plus simple est celle de comparaison. Les opérateurs habituels sont disponibles pour la comparaison : l'égalité (=), la différence (<>), plus grand (>), plus petit (<), plus grand ou égal (>=) ou plus petit ou égal (<=).

Il est également possible d'utiliser des expressions logiques et arithmétiques dans une comparaison.

3.1.1 Opérateurs logiques

Grâce aux opérateurs logiques, il est possible de construire une condition constituée de plusieurs expressions logiques. En SQL, il y a les connecteurs logiques suivants : `AND`, `OR` et `NOT`.

Etant donné la valeur spécifique NULL, il est nécessaire de préciser la table de vérité des opérateurs logiques `AND` et `OR`.

AND	Vrai	Faux	Null
Vrai	Vrai	Faux	Null
Faux	Faux	Faux	Faux
Null	Null	Faux	Null

Table de vérité de AND (extrait de [1])

OR	Vrai	Faux	Null
Vrai	Vrai	Vrai	Vrai
Faux	Vrai	Faux	Null
Null	Vrai	Null	Null

Table de vérité de OR (extrait de [1])

	NOT
Vrai	Faux
Faux	Vrai
Null	Null

Table de vérité de NOT

La clause « WHERE *CONDITION* » d'une requête ne retiendra que les lignes pour lesquelles *CONDITION* est évaluée à *Vrai.* Elle écarte donc les lignes pour lesquelles *CONDITION* est évaluée à *Faux* ou *Null*. (extrait de [5])

Exemples : Soit la table *Test* contenant les valeurs suivantes :

ID	VALUE	VALUE2	VALUE3
1	1	VALUE3 NULL	(null)
2	1	BOTH EQUAL	1
3	8	BOTH NOT EQUAL	6
4	(null)	VALUE NULL	3
5	(null)	both null	(null)

```
SELECT *
FROM TEST
WHERE VALUE = null
```

➔

ID	VALUE	VALUE2	VALUE3

***Note** : Il est important de se souvenir que le symbole null n'est pas une valeur (mais représente l'absence de valeur) et ne peut donc être comparé avec un opérateur d'égalité (=). Dans la requête ci-dessus, il n'est donc pas possible que la condition VALUE = null soit évaluée à TRUE d'où le résultat vide.*

```
SELECT *
FROM TEST
WHERE VALUE IS null
```

➔

ID	VALUE	VALUE2	VALUE3
4	(null)	VALUE NULL	3
5	(null)	both null	(null)

***Note** : Dans cette requête nous utilisons l'opérateur spécifique 'IS' qui permet poser une condition par rapport à l'absence de valeur pour l'attribut VALUE.*

```
SELECT *
FROM TEST
WHERE VALUE < 1 AND ID < 10
```

➔

ID	VALUE	VALUE2	VALUE3

***Note** : Dans cet exemple, lorsque le SGBD évaluera la condition VALUE < 1 pour les 2 derniers tuples, le résultat sera null. Conformément aux tables de vérité ci-dessus, l'expression entière VALUE < 1 AND ID < 10 sera évaluée à null AND TRUE ce qui donnera null comme résultat (et les tuples seront donc écartés du résultat).*

SELECT *
FROM TEST
WHERE VALUE < 1 OR ID < 10 ➔

ID	VALUE	VALUE2	VALUE3
1	1	VALUE3 NULL	(null)
2	1	BOTH EQUAL	1
3	8	BOTH NOT EQUAL	6
4	(null)	VALUE NULL	3
5	(null)	both null	(null)

***Note** : Il s'agit d'un exemple similaire au précédent mais pour lequel il faut se référer à la table de vérité de l'opérateur OR.*

3.1.2 Opérateurs arithmétiques

SQL permet, dans la construction de la condition, d'utiliser les opérateurs arithmétiques classiques. Ainsi, les opérations d'addition (`+`), de soustraction (`-`), de multiplication (`*`) et de division (`/`) sont utilisables. Les règles habituelles de priorité sont d'application.

Il faut cependant savoir que si, dans l'expression arithmétique, une des opérandes prend la valeur `NULL`, c'est toute l'expression qui prend également cette valeur.

Exemples : Soit la table *Test* (voir 3.1.1) précédente :

SELECT (value*2)+4 ➔
FROM Test

(VALUE*2)+4
6
6
20
(null)
(null)

Ce qui montre bien que la valeur NULL n'est pas interprétée comme 'zéro' au sein d'une expression arithmétique.

Il est à noter par ailleurs que deux valeurs NULL comparée l'une à l'autre ne sont pas considérée comme égale :

SELECT *
FROM TEST ➔
WHERE VALUE = VALUE3

ID	VALUE	VALUE2	VALUE3
2	1	BOTH EQUAL	1

3.2. Condition par comparaison à un intervalle

SQL permet, via le mot clé `BETWEEN` d'effectuer une comparaison entre deux valeurs extrêmes (i.e. deux bornes).

Exemple :

```
SELECT NumProduit FROM Produit
WHERE QteStock BETWEEN 10 AND 20
```

Dans cet exemple, on liste les produits dont la quantité en stock est comprise entre 10 et 20 unités.

3.3. Condition par comparaison à un masque

SQL permet de sélectionner certains enregistrements dont on connait seulement une partie d'un attribut (cette option fonctionne uniquement pour des attributs de type caractères).

Ainsi, si on souhaite sélectionner tous les produits dont le libellé commence par le mot DVD, on précisera la condition comme ceci : `libelle LIKE 'DVD%'`.

Le caractère : `%` représente 0 à n caractères quelconques
`_` représente 1 caractère quelconque

Exemple :
```
SELECT numProduit, libelle, prix
FROM produit
WHERE libelle LIKE '%digital%'
```
Dans cet exemple, on demande les produits dont le libellé contient le mot digital.

Utiliser l'opérateur LIKE sans caractère de masquage (% ou _) sera pénalisé lors des évaluations – il s'agit d'un choix inadéquat d'opérateur.

A trouver : comment trouver tous les produits dont le libellé contient au moins 8 caractères dont un B en 3ème position ?

3.4. Condition par comparaison à une liste donnée

Très souvent, il est nécessaire de sélectionner des enregistrements sur base d'une liste de valeurs connues. Dans une requête, la liste de valeur est donnée en extension et le mot clé IN est utilisé. Il est possible d'inverser la sélection en combinant NOT et IN.

Forme :
```
attribut [NOT] IN (val1, val2, …, valn)
```

Exemple :
```
SELECT * FROM client
WHERE categorie IN ('C1', 'C3')
```
Dans cet exemple, on sélectionne les clients appartenant à la catégorie C1 et C3.

```
SELECT * FROM client
WHERE categorie NOT IN ('C2', 'C4')
```
Dans cet exemple, on sélectionne les clients n'appartenant pas aux catégories C2 et C4.

Ne pas utiliser l'opérateur IN lorsqu'il y a au moins 3 comparaisons à effectuer sera pénalisé lors des évaluations – il s'agit d'un choix inadéquat d'opérateur qui alourdi inutilement la requête.

3.5. Condition et la valeur NULL

SQL intègre la valeur `NULL` dans les valeurs possibles pour certains attributs (ceux dont on n'a pas mentionné l'option `NOT NULL`). Cette valeur particulière peut être désignée lors d'une sélection en entrant comme condition : `attribut IS NULL`.

L'opérateur inverse est noté : `attribut IS NOT NULL`

Exemple :
```
SELECT * FROM Client
WHERE categorie IS NULL
```
Dans cet exemple, on sélectionne les clients dont la catégorie n'est pas connue.

Question : est-il opportun d'effectuer un test IS NULL sur un attribut clé primaire ?

4. Tri des enregistrements

On peut demander, dans une requête SQL que le résultat soit trié sur base de la valeur d'un champ donné. Il suffit de préciser dans la clause `ORDER BY` le nom de l'attribut ou des attributs sur lequel le tri doit s'opérer.

Il est également possible de mentionner l'index de la colonne sur laquelle le tri doit porter. Par exemple, si l'on souhaite trier sur la 1ère colonne des résultats, on peut écrire `ORDER BY 1`.

Par défaut `ORDER BY` trie de manière croissante. Pour obtenir un tri décroissant, il faut mentionner le mot clé `DESC` à la fin. Par exemple : `ORDER BY 1 DESC`.

Exemple :
```
SELECT * FROM CLIENT
WHERE Categorie = 'C1'
ORDER BY nom DESC
```
Cette requête liste l'ensemble des clients appartenant à la catégorie C1. Le résultat doit être trié sur le nom, de manière décroissante.

A trouver : comment puis-je effectuer un tri sur base de plusieurs critères (ex. trier les étudiants par nom de famille et, s'il y a des homonymes, utiliser le prénom comme second critère) ?

5. Expression comme résultat

A l'instar des expressions dans les conditions, il est possible de définir des expressions dans la liste des attributs souhaités comme résultat. Ainsi, on peut demander au SGBD d'évaluer des expressions arithmétiques contenant les opérateurs vus précédemment.

Forme :
```
SELECT attribut1, attribut2, …, expression1, expression2, …, attributn
FROM table
[WHERE condition]
[ORDER BY attribut]
```

Exemple :
```
SELECT prix*qteStock
```

```
FROM produit
WHERE qteStock > 0 AND libelle LIKE 'pantalon%'
```

Cette requête calcule la valeur du stock pour les pantalons.

A tester : que se passe-t-il si je multiplie une chaîne de caractères avec un nombre ?

6. Fonctions particulières

Chaque SGBD propose des fonctions pour gérer les dates. Nous verrons dans la suite les fonctions uniquement présentes dans Oracle. Les fonctions de SQL Server se trouvent en annexe (voir annexe A, à la fin de ce document).

6.1 Fonctions sur les dates en Oracle[1]

Oracle propose, dans PL/SQL, les fonctions suivantes :

ADD_MONTHS	Ajoute un nombre entier de mois à une date donnée
EXTRACT	Extrait un élément d'une date donnée
LAST_DAY	Retourne la date du dernier jour du mois (pour une date donnée en paramètre)
MONTHS_BETWEEN	Retourne le nombre de mois entre 2 dates. Le résultat est un réel signé.
NEXT_DAY	Donne la date d'un jour particulier
ROUND	« Arrondi » une date suivant un format donné
SYSDATE	Retourne la date actuelle *du système* dans un format DATE
TO_CHAR	Convertit, dans le format indiqué, une date, un intervalle, …
TO_DATE	Convertit une chaîne en date
TRUNC	Tronque une date suivant un format donné

a) ADD_MONTHS

Ajoute un nombre entier de mois à une date donnée

Format :

```
ADD_MONTHS (date, integer)
```

Cette fonction ajoute le nombre de mois *integer* à la date *date*. Une date est retournée comme résultat.

Exemple :

```
SELECT numClient, ADD_MONTHS(DateCommande, 2)
FROM Commande
```

Ajoute 2 mois à la date de la commande

A tester : que se passe-t-il si le nombre de mois est négatif ?

b) EXTRACT

Extrait un élément d'une date donnée

[1] Cette partie s'appuie sur la documentation disponible dans [4]

Format :
```
EXTRACT(part FROM date)
```

Extrait l'élément *part* de la date *date* passée en paramètre. L'élément *part* peut prendre les valeurs suivantes :

- `YEAR` – extrait l'année
- `MONTH` – extrait le mois
- `DAY` – extrait le jour
- `HOUR` – extrait l'heure
- `MINUTE` – extrait les minutes
- `SECOND` – extrait les secondes

Exemple :
```
SELECT numClient, EXTRACT(MONTH FROM DateCommande)
FROM Commande
```

Affiche les clients ayant commandés et le mois où la commande a été passée.

c) LAST_DAY
Retourne la date du dernier jour du mois (pour une date donnée en paramètre)

Format :
```
LAST_DAY (date)
```

Cette fonction retourne la date du dernier jour du mois de la date *date* passée en paramètre.

Exemple :
```
SELECT numClient
FROM Commande
WHERE dateCommande = LAST_DAY(dateCommande)
```
Retourne les clients ayant commandés le dernier jour du mois.


A tester : Cette fonction prend-t-elle en compte les années bissextiles ?

d) MONTHS_BETWEEN
Retourne le nombre de mois entre 2 dates. Le résultat est un réel signé.

Format :
```
MONTHS_BETWEEN(date1, date2)
```

Retourne le nombre de mois séparant les dates *date1* et *date2*. Le résultat sera positif si *date1* est postérieur à *date2*. Le nombre de mois est donné sous la forme d'un réel. Si les 2 dates représentent le même jour d'un mois (ou le dernier jour d'un mois), le résultat sera un entier.

Exemple :
```
SELECT numCommande
FROM Commande
WHERE MONTHS_BETWEEN(dateCommande, SYSDATE) >  12
```
Sélectionne les commandes des clients passées il y a plus de 12 mois.


A tester : Combien de mois entre le 24/02/2021 et le 18/06/2018 ?

e) NEXT_DAY
Donne la date d'un jour particulier

Format :
`NEXT_DAY (date, char)`

Retourne la date du prochain jour *char* (par rapport à la date *date* donnée). L'élément *char* doit prendre la valeur d'un jour de la semaine (`MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`, `SATURDAY`, `SUNDAY`).

Exemple :
```
SELECT NEXT_DAY(SYSDATE, 'FRIDAY') FROM DUAL
```
Retourne la date de vendredi prochain (par rapport à la date actuelle).

f) ROUND
« Arrondi » une date suivant un format donné

Format :
`ROUND(date, fmt)`

Retourne la date *date* arrondie à l'unité spécifiée dans le format *fmt*. Le format *fmt* peut prendre les valeurs suivantes :

- `CC` – retourne le siècle d'une date (2000 -> 21e siècle ; 1990 -> 20e siècle ; …)
- `YY` – arrondi à l'année (inférieure si le mois < juillet ; supérieure si mois >= juillet)
- `MM` – Arrondi au mois (supérieur si le jour est >= 16)
- `DD` – Arrondi au jour
- `HH` – Arrondi à l'heure
- `MI` – Arrondi à la minute

A tester : la fonction ROUND peut aussi être utilisé pour arrondir un nombre. Quel serait le résultat de cette requête ?
SELECT ROUND(9.94, 1), ROUND(9.99, 1)
FROM DUAL

Sais-tu ce qu'est cette table 'DUAL' ? (tu devrais pouvoir trouver la réponse sur internet)

g) SYSDATE
Retourne la date actuelle *du système* dans un format DATE

Format :
`SYSDATE`

Exemple :
```
INSERT INTO commande (1,23, SYSDATE)
```
Insère un enregistrement dans la table commande avec comme date de la commande, la date du jour.

A tester : est-ce que SYSDATE renvoie aussi les parties heures/minutes/secondes de la date actuelle ?

h) TO_CHAR

Convertit, dans le format indiqué, une date, un intervalle, …

Format :

```
TO_CHAR(date [, fmt])
```

Convertit la date *date* en une chaîne de caractère. Si aucun format n'est spécifié, c'est le format de la date par défaut qui est utilisé (JJ/MM/AA). Le champ *fmt* est une chaîne désignant un format et qui peut combiner les éléments suivants :

- `DAY` – Jour de la semaine (sous la forme d'une chaîne de caractères)
- `D` – Jour de la semaine (entier de 1 à 7)
- `DD` – Jour
- `DDD` – Jour dans l'année
- `MM` – Mois
- `MON` – Abréviation du mois en 3 lettres ('jan.', 'fév.'…)[2]
- `MONTH` – Mois sous la forme d'une chaîne de caractères (janvier, février,…)
- `HH` – Heure du jour (1-12) combinée avec AM/PM
- `HH24` – Heure du jour (0-23)
- `MI` – Minute
- `SS` – Seconde
- `SSSSS` – Secondes écoulées depuis minuit
- `YY` – Année sur 2 chiffres
- `YYYY` – Année sur 4 chiffres
- `/ ou -` – séparateur souhaité

Exemple :

```
SELECT numCommande
FROM commande
WHERE TO_CHAR(dateCommande, 'YYYYMMDD') > '20040101'
```

Sélectionne les commandes postérieures au 1ier janvier 2004. Dans cet exemple, Oracle réalise une comparaison de chaînes.

[2] La langue utilisée (ex : 'avr.' Vs. 'apr.') dépend de la configuration NLS – idem pour 'month'

A titre d'example, considérons la table ETUDIANT(idetudiant integer, datenaissance DATE) dans laquelle nous insérons quelques tuples avec différentes dates :

```
INSERT INTO ETUDIANT VALUES (1, TO_DATE('121212', 'DDMMYY'));
INSERT INTO ETUDIANT VALUES (2, TO_DATE('12-avr.-2012', 'DD-MON-YYYY'));
INSERT INTO ETUDIANT VALUES (3, TO_DATE('24-déc.-2010 13:54:28', 'DD-MON-YYYY HH24:MI:SS'));
INSERT INTO ETUDIANT VALUES (4, TO_DATE('21-10-2000 1:54:28 PM', 'DD-MM-YYYY HH:MI:SS AM'));
INSERT INTO ETUDIANT VALUES (5, TO_DATE('01/01/60 2:10:30', 'DD-MM-YYYY HH24:MI:SS'));
```

Toutes ces valeurs de dates sont stockées dans un format interne (qui inclut les heures, minutes et secondes) et qui peut être formaté de différentes manières lors de l'affichage :

```
SELECT datenaissance AS "default",
       TO_CHAR(datenaissance,'dd-mon-yyyy HH24:MI:SS') AS "dd-mon-yyyy HH24:MI:SS",
       TO_CHAR(datenaissance,'dd-month-yyyy HH:MI:SS') AS "dd-month-yyyy HH:MI:SS",
       TO_CHAR(datenaissance,'dd-month-yyyy HH:MI:SS AM') AS "month-yyyy HH:MI:SS AM"
FROM ETUDIANT
ORDER BY idetudiant ASC
```

default	dd-mon-yyyy HH24:MI:SS	dd-month-yyyy HH:MI:SS	month-yyyy HH:MI:SS AM
12/12/12	12-déc. -2012 00:00:00	12-décembre -2012 12:00:00	12-décembre -2012 12:00:00 AM
12/04/12	12-avr. -2012 00:00:00	12-avril -2012 12:00:00	12-avril -2012 12:00:00 AM
24/12/10	24-déc. -2010 13:54:28	24-décembre -2010 01:54:28	24-décembre -2010 01:54:28 PM
21/10/00	21-oct. -2000 13:54:28	21-octobre -2000 01:54:28	21-octobre -2000 01:54:28 PM
01/01/60	01-janv.-0060 02:10:30	01-janvier -0060 02:10:30	01-janvier -0060 02:10:30 AM

Ne pas utiliser les fonctions de manipulation de date (et se reposer sur le format par défaut) sera pénalisé lors des évaluations. Ceci s'explique par le fait que le format par défaut peut être différent d'un serveur à l'autre et donc potentiellement poser des problèmes dans un cycle de développement logiciel.

i) TO_DATE

Convertit une chaîne en date

Format :

`TO_DATE (char [, fmt])`

Cette fonction convertit la chaîne *char* en date. Si aucun format n'est spécifié, il faut que la chaîne soit conforme au format par défaut JJ/MM/AAAA. Grâce au champ *fmt*, il est possible de définir le format de la date qui est fournie. Le format du champ *fmt* est identique à celui de `TO_CHAR`.

Exemple :

```
SELECT numCommande
FROM commande
WHERE dateCommande > TO_DATE('20040101', 'YYYYMMDD')
```

Sélectionne les commandes postérieures au 1ier janvier 2004. Dans cet exemple, Oracle réalise une comparaison de dates.

Ne pas utiliser les fonctions de manipulation de date (et se reposer sur le format par défaut) sera pénalisé lors des évaluations. Ceci s'explique par le fait que le format par défaut peut être différent d'un serveur à l'autre et donc potentiellement poser des problèmes dans un cycle de développement logiciel.

j) TRUNC

Tronque une date suivant un format donné

Format :

```
TRUNC(date [, fmt])
```

Cette fonction retourne une date tronquée fonction du format *fmt* fourni. Cette fonction est très similaire à la fonction `ROUND`. La différence réside dans le fait qu'aucun arrondi n'est réalisé. Le champ *fmt* est identique à celui de la fonction `ROUND`.

A tester : la fonction TRUNC peut aussi être utilisé pour tronquer un nombre. Quel serait le résultat de cette requête ?

```
SELECT TRUNC(9.94, 1), TRUNC(9.99, 1)
FROM DUAL
```

6.2 Fonction sur les chaînes en Oracle

Oracle propose, dans PL/SQL, les fonctions suivantes (voir [4]):

CONCAT	Concatène 2 chaînes
LOWER – UPPER	Transforme la chaîne en minuscule ou majuscule
LTRIM – RTRIM – TRIM	Supprime des caractères en début et/ou en fin de chaîne
REPLACE	Remplace des éléments dans une chaîne
SUBSTR	Extrait une sous-chaîne
TRANSLATE	Transcrit une chaîne en réalisant des substitutions
LENGTH	Retourne la longueur d'une chaine
INSTR	Cherche une sous-chaîne
TO_NUMBER	Convertit une chaîne en nombre en fonction du format spécifié.

a) CONCAT

Concatène 2 chaînes

Format :

```
CONCAT (char1, char2)
```

Cette fonction retourne une nouvelle chaîne qui est le résultat de la concaténation des 2 chaînes *char1* et *char2*. Pour information, la concaténation peut également être réalisée entre 2 chaînes par l'opérateur `||`

Exemple :

```
SELECT numClient, nomClient, CONCAT(CONCAT(Adresse, ' '), localite)
FROM Client
```

Affiche chaque client avec son adresse complète

```
SELECT numClient, nomClient, adresse || ' ' || localite
FROM Client
```

Identique à l'exemple précédent

b) LOWER – UPPER

Transforme la chaîne en minuscule ou majuscule

Format :

```
LOWER(char)
UPPER(char)
```

Cette fonction retourne la chaîne *char* en minuscule ou majuscule.

Exemple :

```
SELECT * FROM Client
WHERE LOWER(nomClient) = 'dupont'
```

Sélectionne tous les clients dont le nom est Dupont, quelque soit la manière dont il est écrit dans la base de donnée.

c) LTRIM, RTRIM, TRIM

Supprime des caractères en début et/ou en fin de chaîne

Format :

```
LTRIM (char[, set])
RTRIM (char[, set])
TRIM([LEADING|TRAILING|BOTH] char FROM string)
```

La fonction `LTRIM` supprime les caractères présents dans *set* du début de la chaîne *char*. Dès qu'un caractère différent de ceux listés dans l'ensemble *set* est rencontré, cette fonction s'arrête. La fonction `RTRIM` réalise cette opération à partir de la fin de la chaîne.
Pour ces deux fonctions, si *set* n'est pas fourni, la valeur par défaut est : un seul caractère blanc.

La fonction `TRIM` permet de supprimer le caractère *char* qui se trouve au début (LEADING), à la fin (TRAILING) ou les deux (BOTH – valeur par défaut) de la chaîne *string*. Tant que ce caractère est trouvé, il est supprimé.

Exemples :

```
SELECT LTRIM(matricule, 'EDB')
FROM Etudiant
```

On affiche les matricules des étudiants HELMo (ex : E140023, B140124,…) en retirant la première lettre, qu'elle soit E (catégorie économique, D (technique court) ou B (technique long).

```
SELECT RTRIM(phrase, '.')
FROM Texte
```

On affiche les phrases de la table Texte en enlevant les éventuels points situé à la fin de la phrase.

```
SELECT *
FROM Client
WHERE LOWER(LTRIM(nomClient)) = 'dupont'
```

On sélectionne les clients dont le nom est Dupont, quelque soit la manière dont il est écrit et s'il contient des espaces au début ou non.
La fonction LTRIM enlèvera tous les espaces en début de chaîne et la fonction LOWER mettra le résultat en minuscule. Le résultat comprendra donc les clients dont le nomClient contient des valeurs telles que : 'Dupont', ' DUPONT', ' dupont', ' Dupont', ' DUPOnT', ….

```
SELECT *
FROM Client
WHERE LOWER(RTRIM(LTRIM(nomClient))) = 'dupont'
```

On sélectionne les clients dont le nom est Dupont, quelque soit la manière dont il est écrit et s'il contient des espaces au début ou à la fin. . Le résultat comprendra donc les clients dont le nomClient contient des valeurs telles que : 'Dupont', ' DUPONT ', ' dupont ', ' Dupont', ' DUPOnT ', ….
On obtiendra le même résultat avec la requête suivante :

```
SELECT *
FROM Client
WHERE LOWER(TRIM(BOTH ' ' FROM nomClient)) = 'dupont'
```

```
SELECT numClient, numCommande,
               TO_CHAR(TRIM(LEADING 0 FROM dateCommande))
FROM commande
```

Exemple inspiré de [4]. Dans cet exemple, les zéros se trouvant en début du champ dateCommande sont supprimés.

d) REPLACE
Remplace des éléments dans une chaîne

Format :
REPLACE (char, str1[, str2])

Retourne la chaîne *char* dans laquelle chaque occurrence de la chaîne *str1* est remplacée par *str2*. Si *str2* n'est pas donné, la chaîne retournée est la chaîne *char* dans laquelle toutes les occurrences de *str1* ont été supprimées.

Exemple :
```
UPDATE Produit
SET libelle = REPLACE(libelle, 'digital', 'numérique')
WHERE libelle LIKE '%digital%'
```
On met à jour les libellés des produits en remplaçant le mot digital par numérique.

e) SUBSTR
Extrait une sous-chaîne

Format :
```
SUBSTR (char, pos[, length])
```

Retourne une sous-chaine de *char* débutant à la position *pos* sur une longueur *length*. Si la position est négative, Oracle compte à partir de la fin de la chaîne. La première position est 1. Si le champ *length* n'est pas rempli, la longueur restante jusqu'à la fin de la chaîne est supposée.

Exemple :
```
SELECT * FROM Client
WHERE SUBSTR(nomClient, 2) = 'upont'
```
Cet exemple retourne tous les clients dont le nom se termine par 'upont'.

f) TRANSLATE
Transcrit une chaîne en réalisant des substitutions.

Format :
```
TRANSLATE (expr, f_str, t_str)
```

Retourne la chaîne *expr* dans laquelle chaque caractère présent dans *f_str* est substitué par le caractère correspondant de *t_str*.

Exemple :
```
UPDATE CLIENT
SET nomClient = TRANSLATE(nomClient, 'abcdefghijklmnopqrstuvwxyz',
                    'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
```
Dans cet exemple, on convertit le nom du client en majuscule.

g) LENGTH
Retourne la longueur d'une chaîne

Format :
```
LENGTH(char)
```

Retourne la longueur de la chaîne *char*.

h) INSTR
Cherche une sous-chaîne

Format :
```
INSTR(str1, str2[, pos[, occ]])
```

Retourne la position de la *occ*^{ème} occurrence de la chaîne *str2* dans la chaîne *str1* en cherchant à partir de la position *pos*. Si *pos* n'est pas indiqué, la 1ère position est supposée. Si *occ* n'est pas indiqué, la 1ère occurrence est recherchée. Si la chaîne n'est pas trouvée, la valeur 0 est retournée.

Exemple :
```
SELECT * FROM Client
WHERE INSTR(nomClient, 'dupont') > 0
```
Cherche dans les clients ceux dont le nom contient dupont.

i) TO_NUMBER
Convertit une chaîne en nombre en fonction du format spécifié.

Format :
```
TO_NUMBER(chaine [, fmt])
```

Cette fonction convertit la chaîne *chaine* en nombre. Le format spécifié (optionnel) indique comment la conversion peut se faire.

Exemples (extrait de http://www.techonthenet.com/oracle/functions/to_number.php):
```
TO_NUMBER('1210.73', '9999.99')
```
Retourne le résultat décimal : 1210,73

```
TO_NUMBER('546', '999')
```
Retourne le résultat décimal : 546

```
SELECT TO_NUMBER(SUBSTR(DateCommande, 1,2),'99') AS JourCommande
FROM Commande
```
Cette convertit les 2 premiers caractères de la *DateCommande* (en supposant que ceux-ci sont forcément des chiffres) en nombre.

Remerciements

Un merci particulier à mes collègues Vincent REIP et Vincent WILMET pour leur relecture attentive et leurs propositions de correction et d'amélioration.

Bibliographie

[1] C. MAREE et G. LEDANT, *SQL-2 : Initiation, Programmation*, 2ème édition, Armand Colin, 1994, Paris

[2] P. DELMAL, *SQL2 – SQL3 : application à Oracle*, 3ème édition, De Boeck Université, 2001, Bruxelles

[3] Microsoft, MSDN Microsoft Developper Network, http://msdn.microsoft.com, consulté en janvier 2009, Microsoft Corp.

[4] Diana Lorentz, et al., Oracle Database SQL Reference, 10g Release 2 (10.2), published by Oracle and available at http://www.oracle.com/pls/db102/homepage, 2005

[5] Jean-Luc Hainaut, Bases de données – 3ème édition, Dunod, 2015, Paris