Chapitre 7

Extensions propriétaires

Oracle constitue depuis de nombreuses années *le* système de gestion de bases de données professionnel. Il a toujours eu à coeur de proposer des fonctionnalités supplémentaires. C'est ainsi qu'il a développé différentes extensions propriétaires qui anticipaient le développement des standards. Parmi ces extensions mentionnons la gestion des jointures gauches et droites à une époque où elles ne faisaient pas partie de la norme SQL, des fonctions de manipulation des champs, la possibilité de générer des rapports. J'ai mis dans un chapitre à part certaines de ces fonctionnalités, parce qu'elles ne peuvent pas s'employer avec un autre système. Nous verrons d'abord les fonctions (en louchant parfois sur certaines équivalences proposées par *mySQL*), ensuite quelques outils complémentaires qui permettent de générer des rapports assez facilement.

7.1 Fonctions diverses

Plutôt que de donner un tableau complexe, nous allons résoudre une série de problèmes courants.[1]

7.1.1 Manipulation des chaînes de caractères

Majuscules et minuscules

Oracle dispose des trois fonctions `UPPER`, `LOWER` et `INITCAP`. mySQL dispose des deux premières (avec aussi deux synonymes `ucase` et `lcase`).

Mettre une chaîne en majuscules (`UPPER`) : Cette manipulation ne répond pas seulement à des impératifs de présentation. Elle permet de trouver des valeurs dont on ignore la casse des caractères. Notons que les accents sont traités.

[1] J'utilise souvent la table Dual dans mes exemples. C'est une table système reconnue par tout système Oracle. Elle comporte un seul champ nommé Dummy et une seule ligne comprenant le caratère X. Elle permet d'afficher le résultat d'une opération. Par exemple :

```
SELECT 4+3 FROM Dual; -- affiche 7
```

mySQL ne connaît pas de table Dual. Pour simplifier les comparaisons, j'en ai créé une à l'aide de deux simples commandes. Elle n'est évidemment accessible que dans la base de données où elle a été créée !

```
CREATE TABLE `Dual`
(Dummy VARCHAR(1) NOT NULL );
INSERT INTO Dual VALUES ('X');
```

```
SELECT UPPER('été')FROM dual; -- affiche ÉTÉ
```

La vue *Dictionary* contient le nom (*TABLE_NAME*) et la description (*COMMENTS*) de nombreuses tables système. Pour trouver ce qui concerne les synonymes, on penserait produire la requête suivante :

```
SELECT Table_name, Comments FROM Dictionary
WHERE Comments like '%synonym%' ;
```

Elle propose une dizaine de table. On en obtiendra dix fois plus avec cette requête plus astucieuse :

```
SELECT Table_name, Comments FROM Dictionary
WHERE upper(Comments) like '%SYNONYM%' ;
```

Mettre une chaîne en minuscules (`LOWER`) : La fonction `LOWER` fait exactement le contraire de la précédente, et gère également les accents.

```
SELECT lower('FAÇADE') FROM Dual;-- affiche façade
```

Mettre une initiale en majuscule(`INITCAP`) : On peut combiner les deux fonctions pour ne garder en majuscule que la première lettre d'un mot :

```
SELECT initcap('jacques␣')||initcap('THOORENS')
FROM Dual; -- afiche Jacques Thoorens
```

La fonction `INITCAP` n'existe pas en mySQL. Il faudrait définir une fonction qui renverrait l'expression suivante (si son argument se nomme PARA) :

```
concat(upper(substring(PARA,1,1)),lower(substring(PARA,2)))
```

Combinaison de deux ou plusieurs chaînes (`CONCAT`)

La fonction `Concat` autorise la combinaison (ou concaténation) de deux chaînes de caractères.

```
SELECT Concat('Bonjour','le␣monde') FROM DUAL.
```

Oracle dispose d'un opérateur `||` qui peut s'utiliser plusieurs fois. De son côté, mySQL autorise plus de deux arguments avec `concat`. Pour insérer l'espace après bonjour, on écrira donc :

```
SELECT 'Bonjour'||'␣'||'le␣monde' FROM Dual; -- Oracle
SELECT Concat('Bonjour', '␣', 'le␣monde') FROM Dual; -- mySQL
```

Extractions de portions de chaînes

Une donnée est par nature atomique. La sagesse veut qu'on ne regroupe pas plusieurs informations dans un même champ. Oracle dispose néanmoins de plusieurs fonctions permettant de créer ou modifier des chaînes de caractères.

Taille d'une chaîne (`LENGTH`) : Principalement utilisée pour des traitements plus complexes, elle renvoie le nombre de caractères dans la chaîne.

```
SELECT length('Bonjour') from dual;
```

mySQL utilise la fonction `char_length`.

Trouver la position d'une sous-chaîne (`INSTR`) : Également utilisée dans des traitements complexes, la fonction renvoie la position d'une sous-chaîne au sein d'une chaîne plus grande, en choisissant éventuellement la position de départ et l'occurrence ;

```
SELECT instr('Bonjour le monde','o') FROM DUAL; -- 2
SELECT instr('Bonjour le monde','o',3) FROM DUAL; -- 5
SELECT instr('Bonjour le monde','o',1,3) FROM DUAL; -- 13
```

Prendre les X premiers caractères d'une chaîne (`SUBSTR`) : Pour des besoins d'analyse ou de formatage, on peut se limiter à prendre une partie d'une chaîne. Selon le cas, on commence au début ou on prend une portion au milieu de la chaîne :

```
-- les quatre premiers caractères
SELECT substr('Bonjour le monde',1,4) FROM Dual;
-- tous les caractères à partir du 4me
SELECT substr('Bonjour le monde',4) FROM Dual;
-- trois caractères à partir du 4me
SELECT substr('Bonjour le monde',4,3) FROM Dual;
```

mySQL utilise la fonction `substring` de manière identique. Il dispose en outre de `left` pour travailler au début de la chaîne.

Prendre les X derniers caractères d'une chaîne :

```
-- les quatre derniers caractères
select substr('Bonjour le monde',-4) from dual;
```

mySQL utilise également `substring` avec un second argument négatif ou la fonction `right`.

Découper une chaîne en deux : Si les données possèdent un formatage adéquat, il est possible de procéder à un découpage. L'exemple suivant présuppose une table contenant un champ *NP* comprenant un nom de famille séparé du prénom par une virgule et un espace. On va extraire le nom, puis le prénom :

```
SELECT substr(NP,1,instr(NP,',')-1) Nom,
substr(Nom,instr(NP,',')+2) Prenom
FROM TableNoms;
```

La moindre variation d'encodage va faire échouer la procédure, par exemple, l'absence d'espace après la virgule donnera un prénom amputé de son initiale. On peut envisager un traitement plus fin pour éviter ce problème, ou encore recourir à des fonctions basées sur les expressions rationnelles, qui dépassent le niveau de ce cours d'initiation.

Remplacements et transformations

Remplacer une portion de chaîne (`REPLACE`) : Il est parfois nécessaire de remplacer une portion de texte par une autre. C'est le travail de la fonction `REPLACE`. L'exemple suivant supprime les accents du mot *événement* :

```
SELECT Replace('événement','é','e') FROM Dual;
```

Transformation et codage (`TRANSLATE`) : Cette curieuse fonction permet de réaliser des cryptages (élémentaires) ou de supprimer les accents d'une chaîne :

```
SELECT Translate('motdepasse',
'abcdefghijklmnopqrstuvwxyz','AZERTYUIOPQSDFGHJKLMWXCVBN')
FROM DUAL; -- affiche DGMRTHALLT
SELECT TRANSLATE('Événement','àâÀÂéèêëÉÈÊËîïÎÏôöÔÖùûüÙÛÜçÇ',
'aaAAeeeeEEEEiiIIooOOuuuUUUcC') FROM DUAL;
```

Formatage des chaînes

Le formatage des chaînes consiste à ajouter ou supprimer des caractères au début et à la fin des chaînes. Ces fonctions permettaient de réaliser des alignements de données dans des impressions à caractères à largeur fixe. La généralisation des caractères à largeur variable rend ce type de rapport impossible à réaliser. Il faut alors utiliser des programmes spécialisés. Il faut noter cependant que les fonctions de nettoyage restent utiles avec les données de type `CHAR`, puisque ces champs sont automatiquement remplis avec des espaces quand la donnée introduite ne suffit pas à les saturer.

```
SELECT '/'||Nom2||'/' Brut,
'/'||trim(Nom2)||'/' Formate
FROM Table1;
```

Cet exemple montre, par le placement de signes '/' autour des mots que la donnée brute comporte des espaces de remplissages. La fonction `TRIM` permet de les supprimer[2].

```
BRUT              FORMATE
----------------- -----------------
/THOORENS       / /THOORENS/
/X              / /X/
```

À l'inverse, les fonctions `LPAD` et `RPAD` autorisent des alignements et des points de suites.

```
SELECT rpad(trim(Nom2),15,'.') data1,
Nom2 data2 ,
lpad(trim(Nom2),15,'␣') data3
FROM Table1;

DATA1           DATA2           DATA3
--------------- --------------- ---------------
THOORENS....... THOORENS               THOORENS
X.............. X                             X
```

[2] Je ne parle pas de `LTRIM` et `RTRIM`, ni des syntaxes alternatives de `TRIM`.

Avec des caractères proportionnels, ces fonctions ne donnent pas de résultats intéressants :

```
DATA1        DATA2        DATA3
___________  ___________  ___________

THOORENS..... THOORENS          THOORENS
X..............X                    X
```

Ces fonctions se retrouvent également dans mySQL.

Conversion en nombre

On dispose de deux fonctions pour convertir le premier caractère d'une chaîne en son code ASCII et l'opération inverse. Ces fonctions ne sont utiles qu'avec les langues qui utilisent des alphabets dérivés du latin.

```
SELECT ASCII('Administration'),CHR(65) FROM Dual;-- 65,A
```

On dispose aussi de fonctions manipulant les différents types de caractères, avec lesquelles j'avoue n'avoir pas réussir à faire grand chose de convaincant.

```
SELECT convert('150€','we8iso8859p15','AL16UTF16')FROM dual;
SELECT unistr('\00d6') FROM Dual
```

7.1.2 Fonctions relatives aux nombres

Fonctions mathématiques

Ici, un petit tableau permettra de tout concentrer en peu d'espace. J'ai omis de détailler les fonctions trigonométriques.

Fonction	Résultat	Exemple
ABS(*n*)	la valeur absolue de *n*	
CEIL(*n*)	le plus petit nombre entier $\geq n$	Ceil(7.3) →8
EXP(*n*)	*e* (2.71828183) à la puissance *n*	
FLOOR(*n*)	le plus grand nombre entier $\leq n$	Floor(7.3)→7
LN(*n*)	le logarithme népérien de *n* (en base *e*)	
LOG(*m*,*n*)	le logarithme de n dans une base m	
MOD(*m*,*n*)	le reste de la division entière de m par n	
POWER(*m*,*n*)	*m* à la puissance *n*	
ROUND(*m*,*n*)	Arrondi à une ou plusieurs décimales	Round(7.317,2)→7.32
SIGN(*n*)	Le signe du nombre	
SQRT(*n*)	La racine carrée du nombre	
TRUNC(*n*,*m*)	Maintient uniquement *m* décimales	Trunc(7.317,2)→7.31
WIDTH_BUCKET	Voir exemple plus bas	

Les fonctions trigonométriques sont les classiques ACOS, ATAN, COS, COSH, SIN, SINH, TAN et TANH.

La fonction WIDTH_BUCKET, au nom fort peu aimable pour un francophone, permet de placer des valeurs dans des catégories. Prenons par exemple les salaires de l'entreprise de Scott (champ *Sal*) que nous classerons en six groupes entre 0 et 6000 dollars :

```
SELECT Sal,Width_Bucket(Sal,0,6000,6) Groupe
FROM Emp ORDER BY 1;
```

SAL	GROUPE
800	1
950	1
1100	2
1250	2
1250	2
1300	2
1500	2
1600	2
2450	3
2850	3
2975	3
3000	4
3000	4
5000	6

Réglage des paramètres nationaux

Oracle fait une distinction nette entre la représentation des nombres dans les instructions programmées et les nombres formatés, en entrée comme en sortie. Dans les instructions, les nombres se représentent comme dans la plupart des autres langages. Par contre, pour lire ou afficher une donnée numérique, Oracle tient compte des réglages locaux, qui dépendent de nombreux paramètres :

- la langue du système d'exploitation
- les variables d'environnement modifiées par l'utilisateur
- les réglages propres au logiciel client
- les commandes permettant de modifier le comportement de la session en cours
- une spécification au niveau d'une fonction SQL.

Dans la suite, je me référerai uniquement aux changements propres à la session et éventuellement aux fonctions[3]. Il faut noter que l'utilisation de paramètres linguistiques dépend de la manière dont on a installé le serveur. Le serveur du laboratoire de l'École de Commerce et d'Informatique ne connaît que l'anglais et le français. Voici quelques commandes permettant de modifier le comportement d'une session :

```
ALTER SESSION SET NLS_LANGUAGE=FRENCH|ENGLISH|SPANISH;
ALTER SESSION SET NLS_TERRITORY=FRANCE|AMERICA;
```

Pour savoir quels paramètres sont définis dans une session, il faut utiliser la commande SHOW PARAMETER. Malheureusement, celle-ci ne fonctionne pas dans l'environnement de SQL Developer. On peut lire certains de ces paramètres à l'aide de la fonction `SYS_CONTEXT()`.

```
SELECT
SYS_CONTEXT('USERENV','LANGUAGE'),
SYS_CONTEXT('USERENV','NLS_TERRITORY'),
SYS_CONTEXT('USERENV','NLS_CURRENCY'),
SYS_CONTEXT('USERENV','NLS_DATE_FORMAT'),
SYS_CONTEXT('USERENV','NLS_DATE_LANGUAGE')
SYS_CONTEXT('USERENV','NLS_NUMERIC_CHARACTERS')
```

[3]SQL Developer dispose depuis la version 1.1 d'un paramétrage aisé des paramètres de localisation (Menu *Tools/Parameters.../Database/NLS Parameters).*

```
FROM dual;
```

On peut aussi utiliser la commande suivante pour voir les paramètres par défaut du serveur et connaître la liste des paramètres NLS.

```
SELECT * FROM sys.props$
WHERE name LIKE 'NLS%' ;
```

Dans la suite, nous allons supposer que l'application utilisée travaille en français. L'affichage suivant permet de s'en assurer, on doit voir un nombre avec une virgule et le nom du jour en français :

```
SELECT 1.2, to_char(sysdate,'day') FROM dual;
```

Conversions entre chaînes de caractères et nombres

La plupart des conversions sont implicites :

```
SELECT '12,4'+1 FROM Dual; -- affiche le nombre 13,4
SELECT 'Etudiant n°'||1 from dual; --affiche une chaîne
```

L'expression suivante illustre que les nombres s'affichent avec des virgules décimales, mais que la syntaxe d'Oracle utilise le point :

```
SELECT '12,4'+1, ceil(17.5) from dual;
```

La fonction de conversion explicite TO_CHAR respecte par défaut les conventions locales, mais on peut lui donner un format explicite, ou lui demander de suivre une autre convention en précisant un format.

```
SELECT TO_CHAR(1.2), TO_CHAR(1.2,'9.9')FROM Dual;
-- écrit 1,2 et 1.2 en France
```

On peut utiliser le format pour forcer l'affichage d'un nombre précis de décimales ou de zéros avant le premier chiffre. À noter également les codes L (pour le symbole monétaire), D et G pour le point décimal et le séparateur de milliers si on veut se conformer à l'usage local. Le point ou la virgule peuvent forcer un affichage non conforme à celui-ci.

```
SELECT To_Char(123456.78,'L999G999D99')FROM Dual ;
-- écrit €123 456,78
```

À l'inverse, la fonction TO_NUMBER transforme une chaîne en nombre. Ici encore, on peut préciser un format :

```
SELECT To_Number('€123 456,78','L999G999D99') FROM Dual ;
```

7.1.3 Fonctions relatives aux dates

Pour afficher une date, il suffit d'utiliser le nom du champ ou de la fonction qui retrourne une donnée temporelle. Malheureusement, cela ne nous donne pas le choix du format. Le chapitre 4 a déjà évoqué quelques problèmes liés aux dates. Nous allons ici nous borner à examiner quelques solutions.

Format Oracle	Sortie Oracle	mySQL
`to_char(sysdate,'FMdd')`	*7*	`%e`
`to_char(sysdate,'dd')`	*07*	`%d`
`to_char(sysdate,'ddd')`	*341*	`%j`
`to_char(sysdate,'d')`	*4*	`%w`
`to_char(sysdate,'day')`	*jeudi*	`%W`
`to_char(sysdate,'dy')`	*jeu.*	`%a`
`to_char(sysdate,'mm')`	*12*	`%c`
`to_char(sysdate,'mon')`	*déc.*	`%b`
`to_char(sysdate,'month')`	*décembre*	`%M`
`to_char(sysdate,'MONTH')`	*DÉCEMBRE*	
`to_char(sysdate,'yy')`	*06*	`%y`
`to_char(sysdate,'yyyy')`	*2006*	`%Y`
`to_char(sysdate,'year')`	*two thousand six*	
`to_char(sysdate,'ds')`	*07/12/2006*	
`to_char(sysdate,'dl')`	*jeudi 7 décembre 2006*	
`to_char(sysdate,'ss')`	*25*	`%s`
`to_char(sysdate,'hh')`	*09*	`%l`
`to_char(sysdate,'hh24')`	*09*	`%k`
`to_char(sysdate,'mi')`	*59*	`%s`
`to_char(sysdate,'q')`	*4*	
`to_char(sysdate,'RM')`	*XII*	
`to_char(sysdate,'w')`	*1*	
`to_char(sysdate,'ww')`	*49*	`%u`
`to_char(sysdate,'pm')`	*AM*	`%p`

TAB. 7.1 – Les codes utilisés par les fonctions `TO_CHAR()` et `TO_DATE()`

Utilisation de `TO_CHAR` pour convertir les dates

La fonction `TOCHAR()`[4] utilise un deuxième paramètre pour préciser les éléments à afficher, leur format et l'ordre dans lequel ils figurent. On peut utiliser un troisième paramètre pour spécifier la langue. Il est évidemment plus simple de paramétrer le client pour tous les affichages. Dans l'exemple suivant, on paramètre la date en italien sur le client et on affiche une date en espagnol :

```
ALTER SESSION SET nls_date_language='ITALIAN';
SELECT To_Char(sysdate,'dl'),
To_Char(sysdate,'dl','nls_date_language = SPANISH')
FROM Dual;
```

Nous allons examiner quelques-uns des codes de format utlisables avec `TO_CHAR`. MySQL utilise une fonction `DATE_FORMAT` qui utilise des codes à la mode du C. Je donne l'équivalent dans la colonne de droite du tableau suivant 7.1. Notons que mySQL, selon mes information, ne permet pas la localisation des noms de jours et de mois. Il faudra se résoudre à la faire dans le code PHP.

Ces codes servent également à faire les conversions inverses avec la fonction `TO_DATE`. À ce niveau, le programmeur doit rester prudent car on n'est jamais sûr de ce que l'utilisateur

[4] La liste de tous les codes utilisables pour être consultée à l'adresse suivante : *http ://www.adp-gmbh.ch/ora/sql/datetime_format_elements.html*

a tapé.

```
SELECT To_Date('7 décembre 2006','dd month yyyy') FROM Dual;
```

Toujours au niveau des conversions, on peut utiliser la fonction `EXTRACT()` pour obtenir une information partielle (`year`, `month`, `day`, `hour`, `minute`, `second`) à propos d'une date (sous forme numérique) :

```
SELECT Extract(Month FROM sysdate) FROM Dual;
```

Cette fonction se retrouve dans mySQL.

Enfin, quelques fonctions seront utiles en gestion :

- `NEXT_DAY()` : le jour suivant
- `ADD_MONTH()` : ajoute un nombre de mois à une date (utile pour les rendez-vous périodiques).
- `LAST_DAY()` : le dernier jour du mois
- `TRUNC()` et `ROUND()` tronque ou arrondit la date en fonction d'un critère (souvent `MONTH` ou `YEAR`).

La date système

Elle a été illustrée dans les exemples qui précèdent. Les fonctions `SYSDATE` et `CURRENT_DATE` renvoient l'heure et la date actuelle. La différence entre les deux est subtile : la première renvoie l'heure système du serveur, tandis que l'autre est sensible aux fuseaux horaires définis dans les sessions. Il faut donc un réseau mondial pour percevoir la différence.

7.1.4 Fonctions particulières

Caractéristiques de lignes

La fonction `ROWNUM` renvoie un numéro de ligne dans la requête. Cela permet de limiter les affichages à un nombre précis de ligne. Ce n'est malheuresement pas toujours efficace. L'exemple suivant l'illustre.

Afficher les cinq plus gros salaires de la société

```
-- Requête erronnée (le tri s'applique après la sélection)
SELECT Sal, Ename FROM emp
WHERE rownum <= 5
ORDER BY Sal DESC;
```

SAL	ENAME
2975	JONES
1600	ALLEN
1250	MARTIN
1250	WARD
800	SMITH

```
-- Requête correcte
SELECT * FROM(
  SELECT Sal, Ename FROM Emp
  ORDER BY Sal DESC)
WHERE rownum <= 5;
```

SAL	ENAME
5000	KING
3000	SCOTT
3000	FORD
2975	JONES
2850	BLAKE

La fonction `ROWID` renvoie un identifiant unique pour chaque ligne d'une des tables de la base de données. Ce n'est pas une caractéristique relationnelle (on est près de l'adresse absolue), mais cela peut rendre service, notamment pour une surveillance des modifications d'une table. Une ligne reste identifiable même après la modification de son identifiant.

Examen de listes

Les fonctions `GREATEST()` et `LEAST()` renvoient respectivement la plus grande et la plus petite valeur d'une liste.

Gestion des valeurs nulles

La fonction `COALESCE()` renvoie la première valeur non nulle de sa série de valeur. Elle est très utile pour gérer des alternatives permettant de remplacer des valeurs nulles. Prenons l'exemple d'une table comprenant notamment des champs Nom, Prenom et Surnom. Pour certaines personnes, certains de ces champs peuvent être nuls. Pour réaliser une liste de ces personnes, on peut choisir d'utiliser le nom, mais à défaut le prénom ou le surnom.

```
SELECT * FROM Personnes;
```

NOM	PRENOM	SURNOM	TELEPHONE	VILLEDOMICILE	VILLERESIDENCE
Dupont	Jean		12345678	Liège	Liège
	Jean		73591426	Namur	Liège
		le bigleux	75395146	Liège	Theux
			74185293.	Theux	Theux

```
SELECT COALESCE(Nom, Prenom, Surnom, 'personne ' ||rownum)nom,
telephone FROM Personnes;
```

NOM	TELEPHONE
Dupont	12345678
Jean	73591426
le bigleux	75395146
personne 4	74185293.

À l'inverse, `NULLIF()` renvoie null si ses deux opérandes sont identiques sinon la première valeur. Cela me paraît moins utile que la fonction précédente. On peut néanmoins faire disparaître à l'affichage des données redondantes. Par exemple :

```
SELECT COALESCE(Nom,'Personne ' ||rownum)nom,
villedomicile, villeresidence FROM Personnes;
```

NOM	VILLEDOMICILE	VILLERESIDENCE
Dupont	Liège	Liège
Personne 2	Namur	Liège
Personne 3	Liège	Theux
Personne 4	Theux	Theux

```
SELECT COALESCE(Nom,'Personne␣'||rownum)nom ,VilleDomicile,
NULLIF(VilleResidence,VilleDomicile)residence
FROM Personnes
```

NOM	VILLEDOMICILE	RESIDENCE
Dupont	Liège	
Personne 2	Namur	Liège
Personne 3	Liège	Theux
Personne 4	Theux	

Fonctions conditionnelles

SQL2 et Oracle disposent d'une version fonctionnelle de si, nommée `CASE WHEN`. Syntaxiquement, cela ressemble plus à un extrait de programme, mais cela peut s'employer dans une requête SQL. Une première syntaxe place une expression après `case` et examine une série de valeurs après chaque `when`. Cela rappelle le `switch` du langage C.

```
SELECT nom, interro1 int,
CASE interro1
  WHEN 9 THEN 'PGD'
  WHEN 8 THEN 'GD'
  WHEN 7 THEN 'D'
  WHEN 6 THEN 'S'
  ELSE 'echec' END AS Resultat
FROM Etudiants;
```

NOM	INT	RESUL
ABBATE	7	D
ABDULLAH	7	D
ABIDAT	4	echec
ABIDI	5	echec
ABU DALU	5	echec
ADAM	8	GD
AENDEKERK	7	D
AERTS	6	S
AIT HMAD	8	GD

La seconde syntaxe ne place aucune expression après `case` mais ajoute des conditions explicites après chaque `when`. Nous nous rapprochons de la syntaxe du `case` de Visual BASIC.

```
SELECT nom, interro1, case
  WHEN interro1 >=6 THEN 'réussite'
  WHEN interro1 >= 5 THEN 'balance'
  ELSE 'echec' END AS Resultat
FROM etudiants;
```

NOM	INTERRO1	RESULTAT
ABBATE	7	réussite
ABDULLAH	7	réussite
ABIDAT	4	echec
ABIDI	5	balance
ABU DALU	5	balance
ADAM	8	réussite
AENDEKERK	7	réussite
AERTS	6	réussite
AIT HMAD	8	réussite

7.1.5 Fonctions équivalentes de mySQL

Chaînes de caractères

Pour la réalisation des exercices à l'aide de mySQL, on retrouve pratiquement le même jeu de fonctions pour les traitements de chaîne :

LOWER	**LCASE** ou **LOWER_CASE**	**INSTR**	**INSTR, LOCATE**
UPPER	**UCASE** ou **UPPER**	**LENGTH**	**LENGTH**
INITCAP	pas d'équivalent	**ASCII**	**ASCII**
CONCAT	**CONCAT**	**CHR**	**CHAR**
SUBSTR	**MID** ou **SUBSTRING**. Utiliser également **LEFT** et **RIGHT**		

Fonctions diverses

Les fonctions mathématiques vues plus haut se retrouvent exactement en mySQL : `MOD`, `POWER`, `SQRT`, `ABS`, `SIGN`, `FLOOR` et `ROUND`. Légères variantes pour `CEILING` (au lieu de CEIL) et `TRUNCATE` (au lieu de TRUNC).

Les fonctions `GREATEST`, `LEAST`, `COALESCE` et `NULLIF` s'utilisent de la même manière. Il en est de même de `CASE WHEN`.

Fonctions de date

La manière de gérer les dates est assez différente. `SYSDATE()` est maitenant une fonction (synonyme de `NOW()`). On dispose de quelques fonctions au sens évident : `DAYNAME()`, `DAYOFMONTH()`, `DAYOFWEEK()`, `DAYOFYEAR()`, `MONTH()`, `MONTHNAME()`, `YEAR()`, `HOUR()`, `MINUTE()`, `SECOND()`. mySQL manipule toujours des dates en anglais. La fonction `EXTRACT()` s'utilise de la même manière que dans Oracle. La transformation d'une date en chaîne se fait à l'aide de `DATE_FORMAT()`, qui utilise des formats comme le langage C[5] :

`date_format( sysdate(),'%a %d %M %Y')` affichera `Thu 27 October 2005`.

Dans l'expression suivante, on utilise un encodage « standard » d'une date (à la « japonnaise ») pour la formater d'une autre manière :

```
date_format('2005-10-27 01:02:03','%a-%d-%M %Y - %H:%i:%S')
```

On peut aussi utiliser `STR_TO_DATE()`, avec les mêmes codes de format que `DATE_FORMAT()`.

```
STR_TO_DATE('03.10.2003 09.20', '%d.%m.%Y %H.%i')
```

[5]La liste complète de ces formats se trouve dans le manuel : *http://dev.mysql.com/doc/refman/5.0/fr/date-and-time-functions.html*

7.1.6 Pour aller plus loin

La liste exhaustive des fonctions d'Oracle peut s'obtenir au moyen de la requête suivante :

```
SELECT distinct Object_Name
FROM all_arguments
WHERE package_name = 'STANDARD'
order by 1;
```

Pour obtenir quelques explications, on pourra consulter la page suivante :
http://www.ss64.com/orasyntax/functions.html

7.2 Outils complémentaires spécifiques à Oracle

Oracle a un long passé au cours duquel il a proposé des additions à la norme afin d'offrir à ses utilisateurs des moyens supplémentaires pour faciliter la gestion. Je citerai rapidement les requêtes hiérarchiques, qu'il semble impossible de réaliser avec des instructions SQL classiques, un générateur de rapport en mode texte et la possibilité de réaliser des calculs verticaux. Toutes les potentialités de ces outils ne sont pas disponibles dans l'interface iSQL*Plus.

7.2.1 Requêtes hiérarchiques

Les requêtes hiérarchiques permettent de parcourir des hiérarchies où chaque élément peut avoir un père et plusieurs enfants. On représente ce genre de hiérarchies avec des tables qui contiennent une clé étrangère réflexive. Nous avons vu ailleurs l'exemple d'une table d'employés dont le champ `Mgr` permet de connaître l'employé qui est le supérieur hiérarchique d'un employé donné.

```
SELECT E.EmpNo, E. EName, E. MGR, Chef.EName NomMgr
FROM Emp E INNER JOIN Emp Chef
ON E.MGR=Chef.EmpNo
```

EMPNO	ENAME	MGR	NOMMGR
7369	SMITH	7902	FORD
7499	ALLEN	7698	BLAKE
7521	WARD	7698	BLAKE
7566	JONES	7839	KING
7654	MARTIN	7698	BLAKE
7698	BLAKE	7839	KING
7782	CLARK	7839	KING
7788	SCOTT	7566	JONES
7844	TURNER	7698	BLAKE
7876	ADAMS	7788	SCOTT
7900	JAMES	7698	BLAKE
7902	FORD	7566	JONES
7934	MILLER	7782	CLARK

Oracle permet de réaliser des requêtes reprenant une énumération de la hiérarchie sous forme de table. Nous prendrons l'exemple d'une compagnie d'aviation proposant des vols avec correspondances[6].

[6] L'exemple est repris à Christian SOUTOU et Olivier TEST, *SQL pour Oracle*. Eyrolles, 2004.

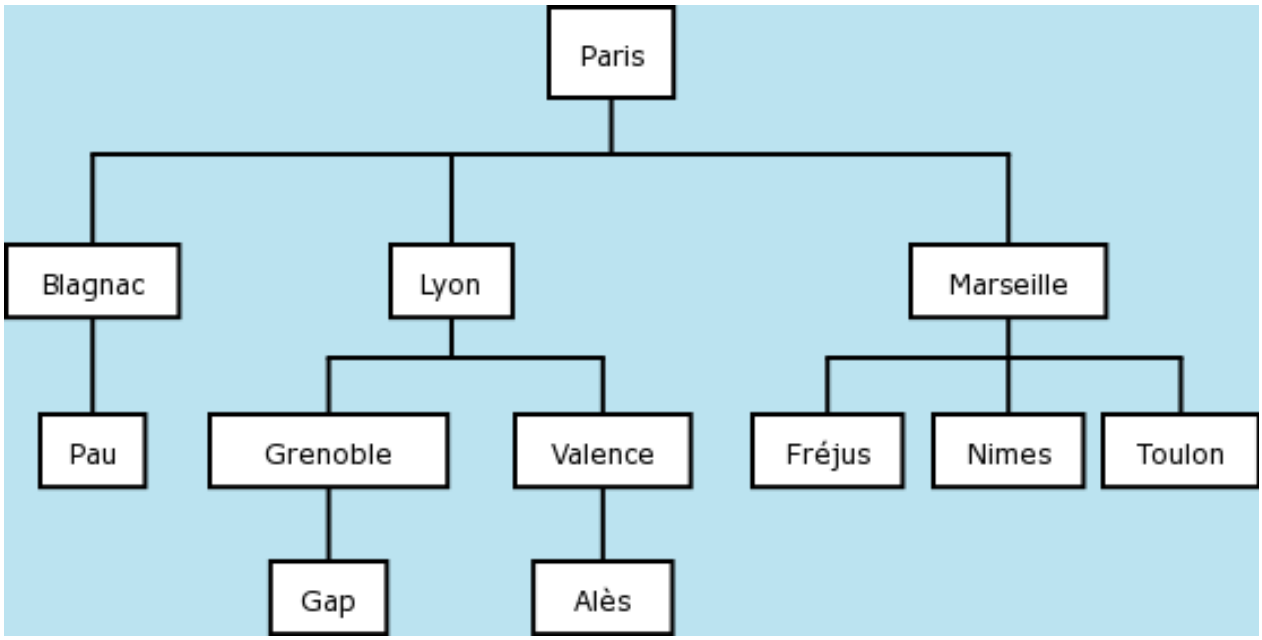

FIG. 7.1 – Hiérarchie de trajets en avion

Les requêtes hiérarchiques vont nous permettre de représenter les différents vols avec correspondances. La table d'exemple ne comporte qu'une seule hiérarchie, mais on peut facilement ajouter d'autres hiérarchies (une autre qui partirait de Londres). De plus les arbres commençant à Blagnac, Lyon et Marseille sont aussi des sommets de hiérarchies.

DEPART	ARRIVEE	TEMPSVOL
Paris	Blagnac	1
Paris	Lyon	.8
Paris	Marseille	.9
Blagnac	Pau	.4
Lyon	Grenoble	.3
Lyon	Valence	.2
Grenoble	Gap	.35
Valence	Ales	.25
Marseille	Fréjus	.2
Marseille	Toulon	.15
Marseille	Nimes	.35

La figure 7.2 montre un exemple du résultat d'une telle requête . Cette requête affiche tous les voyages possibles entre les villes de la table, en énumérant les étapes. On voit par exemple que le voyage de Paris à Gap (en passant par Lyon et Grenoble) occupe trois lignes numérotées de 1 à 3, ce qui correspond aux étapes ou aux niveaux de la hiérarchie.

```
SELECT LEVEL,lpad('␣',4*LEVEL-4,'_')||depart Dep, arrivee
FROM Trajets
CONNECT BY PRIOR arrivee=depart;
```

Le pseudo champ `LEVEL` correspond au niveau hiérarchique et sert à la fois à l'affichage du niveau et au formatage (4 espaces soulignés de décalage par niveau). On notera que le passage d'une ligne à l'autre, qui s'apparente à un traitement récursif, est garanti par la dernière

LEVEL	DEP	ARRIVEE
1	Blagnac	Pau
1	Grenoble	Gap
1	Lyon	Grenoble
2	___ Grenoble	Gap
1	Lyon	Valence
2	___ Valence	Ales
1	Marseille	Fréjus
1	Marseille	Toulon
1	Marseille	Nimes
1	Paris	Blagnac
2	___ Blagnac	Pau
1	Paris	Lyon
2	___ Lyon	Grenoble
3	_____ Grenoble	Gap
2	___ Lyon	Valence
3	_____ Valence	Ales
1	Paris	Marseille
2	___ Marseille	Fréjus
2	___ Marseille	Toulon
2	___ Marseille	Nimes
1	Valence	Ales

FIG. 7.2 – Exemple de requête hiérarchique

ligne. On peut également limiter la requête à une seule hiérarchie en utilisant la clause `START WITH`.

```
SELECT LEVEL,lpad('␣',4*LEVEL-4,'_')||depart Dep, arrivee
FROM Trajets
START WITH Depart='Paris'
CONNECT BY PRIOR arrivee=depart;
```

Le tableau suivant donne une synthèse des trois commandes :

Commande	Description
`LEVEL`	Renvoie le niveau hiérarchique d'une ligne par rapport au point de départ choisi.
`START WITH` *Champ* = *Valeur*	Permet de spécifier le point de départ du parcours
`CONNECT BY PRIOR` *Champ1* = *Champ2*	Décrit la manière de parcourir l'arbre en marquant quel champ de la ligne précédente sert de point de départ de la ligne suivante.

Si une feuille terminale possède la même valeur qu'un point de départ, une requête hiérarchique va se mettre à boucler. Cette circonstance est détectée et provoque un message d'erreur.

7.2.2 Génération de rapports

En mode console, Oracle offre quelques possibilités de formatages de rapports. Cela ne permet pas de générer des documents très sophistiqués (il existe pour cela des programmes

Je Nov 25 — Examen de fin d'année — Année 2003-2004 — page 1

NOM	EXA
COPEIRO	008.0
KOUAM TSOMOGNE	008.0
VASSEUR	008.0
WARLET	008.0
ANACLERIO	007.0
FRANKINET	007.0
HANOT	007.0
HOUNTONDJI	007.0
LEFORT	007.0
DELATTRE	006.0
FRIPPIAT	006.0
LOUYS	006.0
AWOUTERS	005.0
GEORGES	005.0
LAMBRETTE	005.0
MBENOUN	005.0
LENAERTS	004.0
LHOMME	004.0
RUBENS	004.0
SPIRLET	004.0
ARETZ	003.0
MBELO LOSE LOSE	003.0

Version avant délibération

22 ligne(s) sélectionnée(s).

FIG. 7.3 – Exemple de rapport

plus spécialisés), mais ça peut servir pour des petits rapports internes dont la structure n'est pas définie *a priori*. Certaines de ces commandes sont prises en compte par l'interface Web (pas celles qui concernent la largeur des colonnes, toujours traitée dynamiquement).

Voici un petit exemple de rapport et son résultat (voir fig 7.3) :

```
set pagesize 50
ttitle 'Examen␣de␣fin␣d''année|Année␣2003-2004'
btitle 'Version␣avant␣délibération'
column nom format a15
column exa format 099.9
SELECT Nom,Examen exa
FROM Etudiants
WHERE Section = 'Espagnol␣1'
ORDER BY 2 DESC, 1;
```

Ces commandes peuvent être désactivées ou réactivées avec les mots `OFF` et `ON` :

```
ttitle off
column nom off
```

7.2.3 Sous-totaux ou ruptures

Oracle reprend les opérations statistiques classiques pour les intégrer dans des lignes de ruptures (qu'on obtiendrait en utilisant `GROUP BY`) à l'intérieur d'une table. Il faut utiliser pour cela une instruction qui introduit la base de cette rupture et une instruction pour exposer la manière de les calculer. On combinera `BREAK` avec une instruction `COMPUTE`[7].

[7]Il faudra penser à supprimer plus tard l'effet de ces commandes à l'aide de `CLEAR BREAKS` et `CLEAR COMPUTES`.

Afficher les étudiants par sections et par ordre alphabétique, en donnant la moyenne de chaque section[8].

```
BREAK ON section
COMPUTE avg LABEL Moyenne of total on section
SELECT Upper(section) section,nom,total
FROM delibe
WHERE rownum<=10 AND Section like 'All%'
ORDER BY 1,2;
```

L'instruction `ORDER BY` est nécessaire au bon fonctionnement du système (le tri ne se fait pas implicitement comme avec `GROUP BY`).

SECTION	NOM	TOTAL
ALLEMAND 1	CHAUMONT	21
	ENGELEN	16
	FARNIR	15
	ROSSA	17
**********		----
Moyenne		17,25
ALLEMAND 2	BASTOGNE	17
	EVRARD	19
	GRANDJEAN	16
	PETERS	17
**********		----
Moyenne		17,25
ALLEMAND 3	CHEBABI	20
	REGGERS	19
**********		----
Moyenne		19,5

`BREAK` peut se limiter à améliorer la lisibilité en évitant la répétition des valeurs identiques dans un champ.

7.2.4 Champs calculés verticaux

Oracle offre la possibilité de créer des champs reprenant des données provenant d'autres lignes. On utilise une fonction statistique suivie de `OVER` pour créer un champ calculé qui reprend des données provenant de plusieurs lignes. On devra spécifier le champ utilisé comme source en paramètre de la fonction. Les options d'`OVER` spécifient les ruptures éventuelles (`PARTITION BY`) et l'ordre (la même commande que dans le standard SQL). L'ordre spécifié dans la définition doit être cohérent avec celui de la requête, faute de quoi, les résultats sont inexacts. Dans l'exemple suivant, on regroupe les employés par départements, on les numérote et on calcule la somme cumulée des salaires par département et pour l'entreprise.

```
-- Mise en page
set pagesize 100
ttitle 'Extensions␣Oracle'
btitle 'Données␣imaginaires'
-- Définition d'une rupture (avec 1 ligne de séparation)
Break on DeptNo skip 1
-- Opération intermédiaire (moyenne du salaire par département)
Compute avg label moyenne of sal on deptno
-- Requête
```

[8] J'ai limité ici l'affichage à certaines sections et aux 20 premières lignes de la table.

```
SELECT DeptNo,
       -- Numéro séquentiel par département
       ROW_NUMBER() OVER(PARTITION BY Deptno ORDER BY Ename) NumDep,
       Ename,
       Sal,
       -- Deux colonnes de sommes croissantes par département et totale
       SUM(SAL) OVER(PARTITION BY DeptNo ORDER BY DeptNo,Ename) "Total/Dep",
       SUM(SAL) OVER(ORDER BY DeptNo) "Total/Entrep"
FROM Emp;
```

Me Dec 01 Extensions Oracle page 1

DEPTNO	NUMDEP	ENAME	SAL	Total/Dep	Total/Entrep
10	1	CLARK	2450	2450	8750
	2	KING	5000	7450	8750
	3	MILLER	1300	8750	8750
**********			———		
moyenne			2916,66667		
20	1	ADAMS	1100	1100	19625
	2	FORD	3000	4100	19625
	3	JONES	2975	7075	19625
	4	SCOTT	3000	10075	19625
	5	SMITH	800	10875	19625
**********			———		
moyenne			2175		
30	1	ALLEN	1600	1600	27525
	2	BLAKE	2850	4450	27525
	3	JAMES	950	5400	27525
	4	MARTIN	1250	6650	27525
	5	WARD	1250	7900	27525
**********			———		
moyenne			1580		

Données imaginaires

13 ligne(s) sélectionnée(s).

7.2.5 Quelques exemples et la suite sur la toile...

La place et le temps me manquent pour expliquer en détail toutes les possibilités offertent par les extensions Oracle. Les étudiants intéressés peuvent se référer à la documentation fournie par Oracle (les sites `Oracle.com` et `Oracle.fr` sont un bon point de départ, dans les moteurs de recherche, il vaut mieux ne pas simplement taper « Oracle »). Sur le sujet qui nous occupe, je conseille :

`http://www.akadia.com/services/ora_analytic_functions.html`[9].

Pour terminer, je donnerai la liste des fonctions statistiques utilisables avec `OVER` et deux exemples de requêtes qui permettent de régler des problèmes classiques.

```
AVG, CORR, COVAR_POP, COVAR_SAMP, COUNT, CUME_DIST,
DENSE_RANK, FIRST, FIRST_VALUE, LAG, LAST, LAST_VALUE,
LEAD, MAX, MIN, NTILE, PERCENT_RANK, PERCENTILE_CONT,
PERCENTILE_DISC, RANK, RATIO_TO_REPORT, STDDEV, STDDEV_POP,
STDDEV_SAMP, SUM, VAR_POP, VAR_SAMP, VARIANCE.
```

Donner la liste des trois employés de chaque département qui touchent le plus gros salaire.

```
SELECT deptno,ename,sal
```

[9]Le reste du site est très intéressant (principalement en anglais, quelques sujets en allemand) : des dizaines d'articles sur Oracle, d'autres sur Windows, .net, SQL Server et Unix. Malheureusement depuis novembre 2008, il faut demander un mot de passe pour y accéder.

```
FROM
( SELECT deptno, ename, sal,row_number()
  OVER (PARTITION by deptno ORDER BY sal DESC) top
  FROM emp)
WHERE top <=3;
```

Cette requête combine judicieusement l'utilisation de la commande `OVER` et la possibilité d'utiliser une requête imbriquée à la place d'un nom de table. La requête imbriquée regroupe les employés par département et les trie par salaire décroissant. La colonne top contiendra un numéro séquentiel. Il reste à la routine principale à sélectionner ceux qui portent les numéros 1 à 3.

DEPTNO	ENAME	SAL
10	KING	5000
10	CLARK	2450
10	MILLER	1300
20	FORD	3000
20	SCOTT	3000
20	JONES	2975
30	BLAKE	2850
30	ALLEN	1600
30	MARTIN	1250

Afficher les différentes opérations d'un compte en banque, avec le solde et des informations complémentaires

```
SELECT NumOp,Destinataire,Montant, DateOp,
-- Solde (somme des montants précédents)
SUM(Montant) OVER(ORDER BY NumOp) as Solde,
-- Affichage de la date de la ligne précédente
LAG(dateOp,1,NULL) OVER(ORDER BY DateOp) as Prec,
-- Affichage de la date de la ligne suivante
LEAD(dateOp,1,NULL) OVER(ORDER BY DateOp) as Suiv,
-- Somme des deux lignes précédentes + courante
SUM(Montant) over(ORDER BY NumOp ROWS 2 PRECEDING ) as Somme3,
-- idem avec la moyenne
AVG(Montant) over(ORDER BY NumOp ROWS 2 PRECEDING ) as Moyenne3
FROM Compte;
```

NUMOP	DESTINATAIRE	DATEOP	MONTANT	SOLDE	PREC	SUIV	SOMME3	MOYENNE3
1	001-0001234-01	01/01/04	100	100		05/01/04	100	100
2	002-3456789-02	05/01/04	200	300	01/01/04	04/02/04	300	150
3	-	04/02/04	-300	0	05/01/04	03/03/04	0	0
4	003-7894512-98	03/03/04	300	300	04/02/04	15/06/04	200	66,6666667
5	-	15/06/04	-100	200	03/03/04		-100	-33,333333