Première partie

Interroger une base de données

Interroger une base de données

Les chapitres suivants comportent une approche de la partie de SQL généralement nommée *Data Manipulation Language* (ou DML). Dans ce DML, on distingue les commandes permettant la consultation (ou encore extraction) et celles qui entraînent des modifications. Ces dernières feront l'objet du dernier chapitre de la deuxième partie du cours.

SQL comporte donc des commandes permettant l'extraction des données. On utilise essentiellement le verbe `SELECT` dont l'effet est de prendre une ou plusieurs tables, de leur faire subir des opérations de l'algèbre relationnelle et de produire comme résultat une table virtuelle qui est renvoyée à l'initiateur de la requête. On remarquera la fermeture du système : on utilise des relations/tables pour produire d'autres relations/tables.

Nous verrons successivement, dans les prochains chapitres, les points suivants :

- les requêtes simples, c'est-à-dire basées sur une seule table (chapitre 4) : on y trouve les *projections* et les *restrictions* ;
- les requêtes multi-tables, qui établissent différents types de *jointures* entre deux ou plusieurs tables (chapitre 5) ou encore les opérations ensemblistes comme l'*union* ;
- les requêtes récapitulatives, qui permettent de réaliser des opérations statistiques sur les colonnes (somme, moyenne, maximum, minimum...) ;
- les *requêtes imbriquées* dans lesquelles certaines parties de la requête principale sont remplacées par une requête secondaire. (ces deux derniers points seront regroupés dans le chapitre 6).
- les requêtes hiérarchiques et celles qui font appel à des fonctions propres à Oracle (chapitre 7).
- enfin, dans le chapitre 8, nous verrons comment créer, modifier et supprimer des données à l'aide des commandes `INSERT`, `UPDATE` et `DELETE`.

Tous les exemples de cette partie se trouvent presque exclusivement dans deux schémas de base de données nommés *Scott* et *demo*. Rappelons aussi qu'il est indispensable de savoir que le mot de passe de Scott est, depuis la plus haute préhistoire, *tiger*.

Chapitre 4

Requêtes simples

4.1 Noms et notations

Les commandes peuvent s'écrire sur plusieurs lignes, dans une mise en page libre. SQL est suffisamment ancien pour avoir dû supporter des ordinateurs qui se limitaient à des tables de caractères sur 7 bits. Il ignore donc à priori les caractères internationaux et même les minuscules. L'évolution des technologies a amené les standards à s'ouvrir à d'autres langues et à d'autres manières de concevoir la vie que celle de l'oncle Sam.

4.1.1 commandes

Les commandes SQL sont insensibles à la casse et ne comporte que des mots anglais et quelques opérateurs compris dans les 127 premiers caractères. La tradition veut qu'on les écrive à l'aide de `MAJUSCULES`. Ce n'est qu'une habitude : les instructions `SELECT`, `select` et `SeLeCt` sont donc parfaitement équivalentes.

4.1.2 tables

La plupart des systèmes ne permettent pas la distinction des noms de tables sur base de la casse. Une notable exception est mySQL qui utilise des noms de fichiers pour nommer ses tables et bénéficie par là des caractéristiques du système d'exploitation hôte[1]. L'utilisation des accents est autorisée, mais source d'irritants problèmes. La norme SQL2 oblige à entourer les noms de tables comportant des caractères non standard à l'aide de guillemets doubles. Oracle accepte cette norme. Je déconseille son utilisation.

Le nom d'une table peut se voir précéder de noms divers, comme celui de la base ou d'un schéma. Oracle utilise le schéma chaque fois qu'un utilisateur veut accéder à une table qui n'a pas été créée par lui[2].

```
SELECT * FROM Octobre.Ventes;
```

Cette requête affiche la table *Ventes* de l'utilisateur *Octobre*. Les majuscules sont ici décoratives.

[1] Une base comportant une table *BROL* et une autre *brol* créée sous Linux sera donc impossible à porter sous Windows, ce qui, contrairement à ce que pourrait croire mes étudiants, ne constitue pas à mes yeux un avantage.

[2] Pour utiliser une autre base de données, Oracle requiert l'usage d'une autre convention :

```
SELECT * FROM Octobre.Ventes@AutreBase
```

4.1.3 champs

Les noms de champs suivent les mêmes conventions que ceux des tables. On peut utiliser une notation plus complète en les faisant précéder du nom de leur table. Cette pratique devient obligatoire quand deux champs portent le même nom dans deux tables utilisées dans une seule requête. Pour abréger, on donne souvent un alias à la table.

```
SELECT EName FROM Emp;
SELECT Emp.EName FROM Emp;
SELECT E.EName FROM Emp E;
```

Quand un alias de table a été défini, son emploi est obligatoire dans Oracle et mySQL.

4.2 Tables d'exemple

Nous utiliserons une des tables de l'utilisateur Scott[3] pour illustrer nos exemples. En voici la structure (commande `DESCRIBE Emp`) et le contenu (commande `SELECT * FROM Emp`) :

Nom	NULL ?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(2)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

14 rows selected.

Notons qu'un utilisateur peut à tout moment connaître la liste de ses tables en tapant la commande suivante :

```
SELECT * FROM User_Tables;
```

[3]Lors de la création d'une base de données, une option permet la génération de ces tables mythiques à l'aide desquelles des générations d'utilisateurs ont étudié Oracle. J'ai placé le fichier permettant de les générer sur mon site.

4.3 Projections et pseudo-projections

La projection est une opération algébrique qui consiste à ne garder que certains attributs d'une relation (ou si l'on préfère certaines colonnes). L'opération semble banale dans un cas simple : nous allons projeter les attributs *EmpNo et EName* de la table *Emp*.

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

14 rows selected.

```
SELECT DISTINCT EmpNo, EName FROM Emp;
```

La relation suivante montre une caractéristique intéressante de la projection : en projetant *Emp* sur *Job*, on obtient une relation de cinq tuples alors que la relation de base en comporte quatorze.

JOB
CLERK
SALESMAN
PRESIDENT
MANAGER
ANALYST

```
SELECT DISTINCT Job
FROM Emp;
```

Les lignes disparues correspondent aux fonctions qui reviennent plusieurs fois. On doit garder à l'esprit qu'une relation est un ensemble et qu'un même élément ne peut figurer deux fois dans un ensemble.

4.3.1 Instruction `SELECT`

Dans sa forme la plus simple, l'instruction `SELECT` comporte seulement deux parties :
- l'énumération des champs, séparés par des virgules, qui vient immédiatement après les mots `SELECT DISTINCT`,
- le nom de la table, précédé de `FROM`.

Afficher le nom des employés.

```
SELECT DISTINCT EName FROM Emp;
```

ENAME
ALLEN
JONES
FORD
CLARK
MILLER
SMITH
WARD
MARTIN
SCOTT
TURNER
ADAMS
BLAKE
KING
JAMES

Il est possible de supprimer le mot `DISTINCT`. L'opération produite par cette syntaxe simplifiée ressemble à une projection. En réalité, il s'agit d'une opération non relationnelle, puisque la commande ne produit pas une vraie relation. En effet, lorsque la clé primaire n'est pas comprise dans les champs sélectionnés, le risque est grand de voir figurer des lignes identiques dans le résultat. Pour marquer le caractère anormal de cette requête, on utilise parfois le mot `ALL`, qui est donc facultatif.

L'application de la requête ci-dessous produira une table (il ne faut pas dire ici une relation) comportant 14 lignes :

Afficher les professions exercées

```
SELECT ALL Job
FROM Emp;
```

JOB
CLERK
SALESMAN
SALESMAN
MANAGER
SALESMAN
MANAGER
MANAGER
ANALYST
PRESIDENT
SALESMAN
CLERK
CLERK
ANALYST
CLERK

14 rows selected.

```
SELECT DISTINCT Job
FROM Emp;
```

JOB
CLERK
SALESMAN
PRESIDENT
MANAGER
ANALYST

```
SELECT    [DISTINCT]|[ALL]    Nom des champs
FROM Nom de la table
```

Si on veut afficher tous les champs (ce ne sera plus une projection), on peut utiliser le signe '*' à la place du nom des champs. À ce moment, il est inutile d'utiliser `DISTINCT` puisque la clé primaire garantit l'absence de doublons. Notons que l'usage de cet artifice, pratique en mode interactif, est déconseillé en programmation, parce qu'une modification de la structure de la table peut provoquer l'apparition ou la disparition de champs ou un changement d'ordre, avec parfois des conséquences dramatiques pour le programme utilisateur (qui peut accéder aux champs en fonction de leur position). Il vaut mieux prendre la peine d'énumérer les champs voulus, dans un ordre choisi.

```
SELECT ALL *
FROM Ventes;
```

Le nom des champs peut être précédé du nom de la table à laquelle ils appartiennent (utile, voire nécessaire dans le cas d'une requête multi-table).

```
SELECT ALL Ventes.Client, Ventes.Description
FROM Ventes;
```

4.3.2 Champs calculés

Un champ calculé est un nouveau champ introduit dans la table virtuelle résultante et rempli à l'aide de valeurs calculées sur base des autres champs de la ligne où il figure. On utilisera souvent un alias pour le caractériser.

Afficher les montants et leur conversion en euros.

```
SELECT EName, Sal as Dollars,
Sal/1.25 as Euros FROM Emp;
```

ENAME	DOLLARS	EUROS
SMITH	800	640
ALLEN	1600	1280
WARD	1250	1000
JONES	2975	2380
MARTIN	1250	1000
BLAKE	2850	2280
CLARK	2450	1960
SCOTT	3000	2400
KING	5000	4000
TURNER	1500	1200
ADAMS	1100	880
JAMES	950	760
FORD	3000	2400
MILLER	1300	1040

L'usage des alias permet d'améliorer la lisibilité du résultat, mais ils ne sont nullement nécessaires. Notons que selon les systèmes, l'alias est ou non précédé de AS. En principe, on ne peut pas utiliser l'alias dans une clause `WHERE`[4].

Alias pour colonne	SQL ISO	mySQL	SQL Server	Oracle 10g	Access
Précédé par	`AS` ou rien	`AS` ou rien	`AS`	`AS` ou rien	`AS`
Utilisable dans la requête en cours	non	non	non	dans `Order`	non

```
SELECT ALL Client AS C, Description AS D
FROM Ventes;
```

La majorité des champs calculés résultent d'une opération arithmétique, mais la plupart des systèmes acceptent des opérations complexes sur des dates ou des chaînes de caractères. Ces opérations complexes font usage de fonctions propriétaires du système, ce qui les rend souvent difficilement portables. Nous aborderons les fonctions propres à Oracle et mySQL dans le chapitre 7.

4.3.3 Tris (clause ORDER BY)

Dans la plupart des cas, on désire trier les résultats obtenus. Ce n'est pas une opération ensembliste (les éléments d'un ensemble n'ont pas d'ordre), mais elle est bien utile pour consulter la relation obtenue. On y parvient en ajoutant l'expression `ORDER BY` suivie d'un nom ou d'un numéro de colonne.

Afficher les employés triés par départements et par noms

```
SELECT DeptNo, EName FROM Emp
ORDER BY DeptNo, EName;
```

DEPTNO	ENAME
10	CLARK
10	KING
10	MILLER
20	ADAMS
20	FORD
20	JONES
20	SCOTT
20	SMITH
30	ALLEN
30	BLAKE
30	JAMES
30	MARTIN
30	TURNER
30	WARD

```
SELECT DeptNo, EName FROM Emp
ORDER BY 1,2;
```

[4]Oracle tolère néanmoins l'emploi de l'alias dans la clause `ORDER BY`.

La deuxième syntaxe présente l'avantage de ne pas devoir répéter le nom du champ s'il est particulièrement long. Elle sera surtout utile pour des champs calculés :

```
SELECT EName,Sal/1.25 As Euros
FROM Emp
ORDER BY 2;
```

Notons que si le champ calculé possède un nom d'alias, il n'est utilisable dans le tri que dans Oracle.

```
/* SYNTAXE REFUSEE sauf par Oracle ! ! ! */
SELECT EName,Sal/1.25 As Euros
FROM Emp
ORDER BY Euros;
```

Lorsque le tri fait apparaître des homonymes, on peut ajouter un deuxième critère (et d'autres encore). En pratique, 2 à 3 critères sont suffisants. Dans ce cas, les données sont triées selon le premier critère. Au cas où des homonymes apparaissent, ils sont triés en fonction du deuxième critère. Dans l'hypothèse où ce critère produit encore des homonymes, on utilisera le troisième critère et ainsi de suite.

SQL propose un tri ascendant par défaut (ASC) et un tri descendant (DESC) surtout utile pour les valeurs monétaires.

Afficher les employés par département, par salaires décroissants (et éventuellement par ordre alphabétique).

```
SELECT DeptNo, Sal, EName
FROM Emp
ORDER BY DeptNo,Sal DESC,EName;
```

DEPTNO	SAL	ENAME
10	5000	KING
10	2450	CLARK
10	1300	MILLER
20	3000	FORD
20	2975	JONES
20	800	SMITH
30	2850	BLAKE
30	1600	ALLEN
30	1500	TURNER
30	1250	MARTIN
30	1250	WARD
30	950	JAMES

```
SELECT [DISTINCT]|[ALL] ListeNomsChamps
FROM NomTable
ORDER BY NomChamp | NuméroChamp [ASC]|[DESC]
[, ...]
```

4.4 Restrictions

Dans une restriction, on ne garde que les tuples qui possèdent une certaine propriété. Par exemple, ceux qui concernent les employés d'un seul département.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/80	800		20
7566	JONES	MANAGER	7839	02/04/81	2975		20
7902	FORD	ANALYST	7566	03/12/81	3000		20

Pour ne conserver que ces lignes, on spécifiera la condition à l'aide du mot `WHERE`.

Afficher les employés du département portant le numéro 20

```
SELECT * FROM Emp
WHERE DeptNo = 20;
```

```
SELECT [DISTINCT]|[ALL] ListeNomsChamps
FROM NomTable
WHERE Condition
```

Dans les requêtes simples, les conditions supposent le plus souvent la comparaison du contenu des champs à des valeurs constantes de références. Nous verrons en fin de chapitre quelques remarques utiles à cet égard.

Afficher les employés engagés après le 30 juin 1981.

```
SELECT EName, HireDate FROM Emp
WHERE HireDate > date '1981-06-30';
```

DEPTNO	ENAME
10	CLARK
10	KING
10	MILLER
20	ADAMS
20	FORD
20	JONES
20	SCOTT
20	SMITH
30	ALLEN
30	BLAKE
30	JAMES
30	MARTIN
30	TURNER
30	WARD

Notons au passage qu'il n'est nullement nécessaire d'afficher le champ sur lequel porte la restriction. Nous le faisons ici dans le but de convaincre le lecteur que cela marche ! Nous reviendrons plus loin sur la manière d'écrire la date.

Les différentes conditions s'exprimeront sous la forme :

– d'une comparaison

- de l'appartenance à un intervalle
- de l'appartenance à une liste
- d'une ressemblance
- d'un test de valeur NULL
- d'une condition complexe

4.4.1 comparaisons (=, <, >, <=, >=, <>)

Les six comparateurs traditionnels des langages de programmation sont utilisés en SQL. Notons que le comparateur d'inégalité est '<>', bien qu'on trouve parfois '!=' (IBM dB2 et Oracle) et '^='.

Afficher le nom des employés dont le salaire est différent de 1250.

```
SELECT EName, Sal FROM Emp
WHERE Sal <> 1250;
```

ENAME	SAL
SMITH	800
ALLEN	1600
JONES	2975
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

4.4.2 intervalles (BETWEEN ... AND)

On se limite ici à conserver les lignes dont le champ spécifié est compris dans un intervalle (les limites sont comprises).

Afficher les employés dont le salaire est compris entre 1000 et 3000.

```
SELECT EName, Sal FROM Emp
WHERE Sal BETWEEN 1000 AND 2000;
```

ENAME	SAL
ALLEN	1600
WARD	1250
MARTIN	1250
TURNER	1500
ADAMS	1100
MILLER	1300

4.4.3 appartenance (IN)

Cette syntaxe permet de ne garder que les lignes dont la valeur d'un champ donné est contenue dans une liste de valeurs. Cette liste peut consister en une série de valeurs explicites

ou être générée au moment de l'exécution. Cette dernière technique sera spécialement utile dans les requêtes imbriquées.

Afficher les employés dont le département est 10 ou 20.

```
SELECT EName, DeptNo FROM Emp
WHERE DeptNo IN(10,20);
```

ENAME	DEPTNO
SMITH	20
JONES	20
CLARK	10
SCOTT	20
KING	10
ADAMS	20
FORD	20
MILLER	10

4.4.4 ressemblance (`LIKE`)

L'opérateur LIKE autorise des recherches sur des valeurs approximatives. Deux caractères génériques sont utilisés :

% désigne un nombre variables de caractères quelconques (éventuellement nul) ;

_ désigne un et un seul caractère quelconque[5].

On pourra, à l'aide de ces caractères, construire des notions telles que :

commence par 'A' : `LIKE("A%")`
contient 'A' : `LIKE("%A%")`
se termine par 'A' : `LIKE("%A")`

Afficher le nom des employées dont le nom commence par J.

```
SELECT EName FROM Emp
WHERE EName LIKE 'J%';
```

ENAME
JONES
JAMES

Afficher le nom des employés se terminant par 'rd'.

```
SELECT EName FROM Emp
WHERE EName LIKE '%RD';
```

ENAME
WARD
FORD

[5]Access, par nostalgie de MS-DOS, remplace ces deux caractères par * et ?. Par contre, les concepteurs de MS-Query sont restés sourds aux ordres du CEO de Microsoft et se sont conformés au standard SQL.

On peut parfois spécifier un caractère qui sert de préfixe aux caractères génériques pour effectuer une recherche de '%' ou de '_' en utilisant un caractère d'échappement défini au moyen de ESCAPE. Par exemple pour rechercher un champ terminé par "%", la simple utilisation de '%%' nous renvoie toutes les lignes.

```
SELECT * FROM Demo.TestLike
WHERE Ristourne LIKE '%%';
```

ID	NOM	RISTOURNE
1	Dupont	Aucune
2	Durant	10%

L'utilisation d'un caractère d'échappement, ici le $, ne garde que la ligne qui nous intéresse.

```
SELECT * FROM Demo.TestLike
WHERE Ristourne LIKE '%$%' ESCAPE '$';
```

ID	NOM	RISTOURNE
2	Durant	10%

Il faut noter qu'une erreur fréquente des débutants consiste à écrire soigneusement la chaîne de recherche en utilisant le signe = au lieu de LIKE. Les caractères génériques sont alors pris pour des caractères normaux et la recherche échoue !

4.4.5 la valeur NULL

Codd, l'inventeur des SGBD, recommande d'incorporer dans les SGBDR une valeur NULL qui signifie l'ignorance d'une valeur ou sa non-pertinence. NULL est distinct de 0, d'une chaîne vide ou de toute autre valeur. Les champs de n'importe quel type peuvent prendre la valeur NULL. Examinons le champ *Comm* de notre table d'employés.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Seuls quatre employés ont une commission, l'un d'entre eux reçoit une commission de 0 euros. Tous les autres n'ont aucune commission mentionnée. Cela se matérialise par une valeur *Null*.

NULL entraîne une curieuse conséquence : celle de générer une logique trivalente. Il faut donc renoncer à raisonner en termes de vrai ou de faux. Une troisième valeur doit être prise en compte. On trouvera dans la section suivante les tables de vérité des connecteurs logiques. Nous allons insister ici sur le comportement de NULL face aux comparateurs et son effet sur la sélection des lignes dans la périphrase WHERE.

Liste des employés dont la commission est supérieure à 1000.

```
SELECT EName, Comm FROM Emp
WHERE Comm>1000;
```

ENAME	COMM
MARTIN	1400

```
ENAME      COMM
---------- ----------------------
MARTIN     1400
```

Liste des employés dont la commission n'est pas supérieure à 1000.

```
SELECT EName, Comm FROM Emp
WHERE NOT Comm>1000;
```

ENAME	COMM
ALLEN	300
WARD	500
TURNER	0

On voit que les deux requêtes, qui s'opposent, selon la logique d'Aristote, ne parviennent pas à énumérer tous les employés. C'est la négation du principe du tiers exclu.

Quand on compare le contenu d'un champ vide (NULL) avec n'importe quoi, le résultat est toujours NULL. C'est logique. Paraphrasons : est-ce que la valeur que j'ignore est supérieure à 5 ? La réponse ne peut être que «Je n'en sais rien», donc NULL. De manière plus inattendue, si je compare une valeur NULL à NULL, la réponse est encore NULL. Il faut donc disposer d'une expression particulière pour tester si un champ est NULL : c'est le rôle des expressions IS NULL et IS NOT NULL. Si nous sélectionnons les employés de notre table en fonction de leur commission, voici les résultats que nous obtiendrons pour ceux qui n'en ont pas :

Expression	Valeur	Si *Comm* vaut Null	
WHERE Comm > 1000	NULL	Non sélectionné	
WHERE Comm < 1000	NULL	Non sélectionné	
WHERE Comm <> 1000	NULL	Non sélectionné	
WHERE Comm = 1000	NULL	Non sélectionné	
WHERE Comm IS NULL	TRUE	Sélectionné	
WHERE Comm IS NOT NULL	FALSE	Non sélectionné	

Une ligne d'une table est sélectionnée uniquement si la condition énoncée dans la périphrase WHERE est vraie. À l'inverse si la condition est fausse ou inconnue (NULL), la ligne ne sera pas prise en compte.

4.4.6 conditions complexes (AND, OR, NOT)

Lorsque plusieurs conditions interviennent, on est obligé de les unir au moyen de connecteurs logiques.

Il faut rappeler que l'introduction de NULL a modifié la logique traditionnelle et que les tables de vérité de AND, OR et NOT ont dû être revues. Les quatre cases du coin supérieur gauche représentent les tables logiques classiques, les dernières ligne et colonne introduisent la valeur NULL. Pour comprendre ces tables, il suffit de se rappeler que NULL signifie « j'ignore ». Ainsi, « Vrai AND NULL » prend la valeur NULL parce que je dois savoir si la deuxième valeur est vraie ou fausse avant d'envisager la valeur de l'ensemble. Par contre, « Faux AND NULL » est faux, parce que quelle que soit la valeur que j'ignore, elle ne pourra pas rendre vraie l'expression complète.

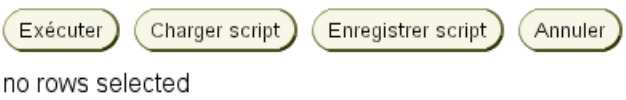
AND	Vrai	Faux	*Null*
Vrai	Vrai	Faux	*Null*
Faux	Faux	Faux	*Faux*
Null	*Null*	*Faux*	*Null*

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

NOT	Vrai	Faux	*Null*
	Faux	Vrai	*Null*

Bien qu'apparemment l'emploi de AND et OR paraisse aller de soi, il y a des contextes dans lesquels une formulation en langue naturelle peut induire en erreur. Si je veux envoyer un mailing aux acheteurs de souris et aux acheteurs de claviers, la requête suivante ne donnera rien :

```
SELECT Demo.Client FROM Ventes
WHERE Description = 'Souris' AND Description = 'Clavier';
```



Il faut bien employer ici le connecteur OR :

```
SELECT Client FROM Demo.Ventes
WHERE Description = 'Souris' OR Description = 'Clavier';
```

CLIENT
Bernard
Henri

L'emploi de plusieurs connecteurs pose aussi le problème de la priorité des opérateurs. Elle se fait comme dans les autres langages (priorité plus grande de `AND`) comme le confirme le test suivant :

```
SELECT Client, Description
FROM Demo.Ventes
WHERE Description = 'Souris' OR
Description = 'Clavier' AND Description='Imprimante';
```

CLIENT	DESCRIPTION
Bernard	Souris

C'est comme si on avait écrit[6] :

```
WHERE Description = 'Souris' OR
(Description = 'Clavier' AND Description='Imprimante');
```

Pour être complet, je mentionnerai que le standard SQL2 propose des connecteurs logiques supplémentaires. Outre l'expression `IS NULL`, il dispose de `IS TRUE`, `IS FALSE` et `IS UNKNOWN` (synonyme de `IS NULL`). Ces extensions ne sont pas reconnues par InterBase, ni par Oracle.

Nous verrons plus loin qu'Oracle propose des fonctions permettant d'agrémenter l'usage de la valeur `NULL`.

4.5 Écriture des constantes

4.5.1 Les nombres

Les nombres s'écrivent comme dans la plupart des langages de programmation. On notera la notation avec mantisse pour représenter les nombres réels s'écartant des nombres de taille « normale » (exemple `7E16` ou `1.5E-8`). Il faut mentionner que l'écriture et l'affichage peuvent se voir influencés par le paramétrage des clients : par exemple sur Oracle, selon qu'on a défini ou non une variable d'environnement chez le client, la langue de dialogue, mais également l'aspect des nombres va changer.

1. après avoir effacé cette variable : `unset NLS_LANG`

```
Connected to : Oracle9i Enterprise Edition Release 9.2.0.1.0
...
SQL> select 1/3 from dual;
1/3
----------
.333333333
```

[6] Je conseille d'ailleurs fortement de mettre les parenthèses pour éviter tout désagrément.

2. après l'avoir redéfinie : `export NLS_LANG="french_france.we8iso8859p1"`

```
Connecté à : Oracle9i Enterprise Edition Release 9.2.0.1.0
...
SQL> select 1/3 from dual;
1/3
----------
,333333333
```

Lorsqu'on utilise un client Web, le paramétrage de la langue est défini sur le serveur Web. Avec SQLDeveloper, nous avons vu au chapitre précédent comment régler les paramètres linguistiques dans les options.

4.5.2 chaînes de caractères

La norme SQL autorisait deux modes d'écriture des chaînes : avec guillemets doubles ou avec apostrophes. La norme SQL2, en réservant l'usage des guillemets doubles aux noms de champs avec caractères spéciaux, oblige à l'emploi de simples apostrophes. Pour placer une apostrophe à l'intérieur d'une chaîne, on la doublera. Oracle n'accepte pas la notation « échappée » à la mode du C. mySQL l'accepte parfaitement.

```
SELECT 'aujourdh''hui' FROM DUAL;
-- Notation refusée par Oracle
SELECT 'aujourdh\'hui' FROM DUAL;
```

4.5.3 Les dates

Coupables d'avoir provoqué le *bug de l'an 2000*, les dates n'ont pas fini de nous en faire voir. Elles posent trois problèmes spécifiques :

- la mauvaise habitude que nous avons de ne noter que les deux derniers chiffres de l'année, source d'ambiguïté,
- l'écriture du mois qui dépend de la langue lorsqu'on l'affiche sous forme de trois lettres[7],
- la disposition respective des jours et des mois qui varient d'un pays à l'autre (31/12/04 en France et en Angleterre, 12/31/04 aux États Unis, 04/12/31 au Japon) .

C'est ainsi que '12/4/49' pourra désigner le 12 avril 1949 comme le 4 décembre 2049.

Pour régler le dernier problème, je conseille d'exprimer chaque fois que c'est possible les années en quatre chiffres, ce qui résout au moins le problème de l'encodage. Pour la lecture, on ne dispose pas toujours de la place pour afficher quatre chiffres. Chaque base de données effectue un choix par défaut de la limite entre les XXe et XXIe siècles. Oracle a choisi d'interpréter les nombres compris entre 50 et 99 comme faisant partie du siècle dernier et 00 et 49 comme faisant partie de notre siècle[8].

Chaque SGBD propose ses conventions. Access est, à ma connaissance, le seul à utiliser des marqueurs spécifiques pour les dates. Le 12 avril 1949 s'écrira pour lui `#12 avril 1949#`. Access interprète les dates selon les normes du pays de l'utilisateur. Malheureusement, une date erronée peut donner lieu à des interprétations malencontreuses : par exemple, le 11 août 2004 encodé par erreur #11/18/04# ne sera pas refusé mais compris comme le 18 novembre 2004, comme une date américaine. Les autres systèmes proposent d'écrire les dates

[7] Les mois de février, avril, mai et août posent problèmes dans un contexte franco-anglais. Pour éviter l'ambiguïté, les mois de juin et juillet s'écrivent *jun* et *jul* dans les deux contextes.

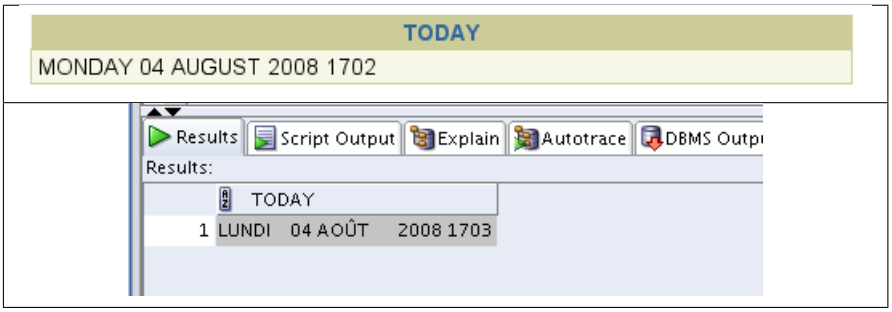
[8] SQL-Server 7.0 place la barre à 2030. Il reste 23 ans avant le bug.

sous forme de chaîne et effectuent eux-même la conversion, pas toujours comme nous l'aurions voulu.

tester son système

La commande suivante affiche l'heure système :

```
SELECT to_char(sysdate,'DAY␣DD␣MONTH␣YYYY␣HH24MI')as today
FROM DUAL;
```



On peut déjà déterminer si le serveur/le client est paramétré pour le français.

écrire une constante date

Lors de l'encodage ou lors de comparaisons, il est nécessaire d'avoir un meilleur contrôle de la manière d'écrire une date. Selon le type de données choisies lors de la création de la table, les résultats peuvent être corrects ou non. Voici à titre d'exemple, un ajout de données dans une table comportant un champ *date* et un champ *timestamp* (qui contient aussi l'heure). L'affichage des données montre une erreur évidente :

```
INSERT INTO Demo.TestDate
VALUES ('31␣dec␣2004','31␣dec␣2004');
SELECT * FROM Demo.TestDate;
```

C1	C2
31-DEC-04	31-DEC-20 04.00.00.000000 AM

Dans le deuxième champ, 2004 est décomposée en 2020 et 04 heures. Voici une méthode plus lourde mais plus fiable pour remplir les deux champs :

```
INSERT INTO Demo.TestDate
VALUES (to_date('31␣dec␣2004', 'DD␣MM␣YYYY' ),
        to_date('31␣dec␣2004', 'DD␣MM␣YYYY' ));
SELECT * FROM Demo.TestDate;
```

C1	C2
31-DEC-04	31-DEC-04 12.00.00.000000 AM

On aurait pu employer aussi la notation `date '2004-12-04'` .Le chapitre **??** présentera plus en détails ces fonctions `to_date()` et `to_char()`.

4.5.4 Null

La valeur *null* ne doit pas s'écrire entre apostrophes, sous peine d'être considérée comme une chaîne de caractères.

4.5.5 Le comparateur `LIKE`

Nous avons vu plus haut la richesse de cet opérateur. Il donne parfois des résultats décevants, à cause d'un mauvais formatage de la chaîne. Deux exemples de recherches avec Oracle l'illustreront[9].

- Rechercher le nom des étudiants qui ont une note comportant 1/3 de points de moyenne. La requête suivante ne fonctionne pas dans l'environnement français de SQL-Developer.

```
SELECT Nom FROM Demo.Etudiants
WHERE (Interro1+Interro2+Examen)/3 LIKE '%.3%';
```

  Effectivement, le format de sortie des nombres flottants comporte une virgule. Il fallait donc écrire `LIKE '%,3'`. Dans l'interface Web, par contre, tous les étudiants qui ont des points comportant des chiffres trois dans la partie fractionnaire sont listés.
- Rechercher les étudiants nés en janvier. Bien qu'il soit conseillé d'utiliser le nom des mois en lettre, on devra ici se plier au format de sortie qui utilise des chiffres.

```
SELECT Nom, Datenaiss FROM Etudiants
WHERE Datenaiss LIKE '%/01/%';
```

La leçon à tirer de tout cela est que l'écriture d'une requête est souvent fortement conditionnée par son contexte. Il faut donc étudier en détails l'environnement de production pour être certain d'obtenir les bons résultats. Si le programme est amené à tourner dans des environnements variés, on fera bien de paramétrer un maximum l'écriture des requêtes automatiques. On peut par exemple disposer de variables LANGUE, DECPOINT qui contiendront la langue et la manière d'écrire le point décimal, de manière à ce que le programme utilise les bons formats.

[9] Ces exemples sont basés sur les réglages propres à mon système, qui sont en principe aussi ceux du serveur de l'école.